

Machine Learning on Audio Signals

Course Booklet

Masudul Imtiaz, PhD
Ajan Ahmed
Yash Sukhdeve

Wallace H. Coulter School of Engineering and Applied Sciences
Clarkson University — Potsdam, NY, USA

EE502 Machine Learning on Biomedical Signals — Fall 2025/2026

Table of contents

1	Introduction to Speech Signals	2
2	Speech Preprocessing and Feature Extraction	12
3	Introduction to Machine Learning	30
4	Audio Classification Using MFCC and Decision Tree Models	42
5	Key-Word Detection using Random Forest	54
6	SVM-based Speaker Classification	63
7	Model Comparisons for Speech Emotion Recognition	72
8	Deepfake Detection using ML	82
9	Voice Based Gender Identification	92

Preface

Machine Learning on Audio Signals began as a set of lecture notes for *EE502 – Machine Learning on Biomedical Signals* at Clarkson University. Over several semesters those notes grew into the self-contained booklet you now hold, written to guide advanced undergraduate and early graduate students through the essential signal-processing and machine-learning techniques that underpin modern speech and audio applications.

Our primary goals are threefold:

1. **Conceptual clarity.** Each chapter is organised around first principles—sampling theory, short-time analysis, feature extraction—before progressing to classical machine-learning models. Mathematical derivations are kept concise, with plain-language intuition provided alongside formal definitions.
2. **Hands-on practice.** Every major concept is accompanied by `PYTHON` script (tested under `librosa`, `NumPy`, and `scikit-learn`) so that readers can experiment with real data. Carefully chosen exercises and Jupyter notebooks reinforce the link between theory and implementation.
3. **Pragmatic scope.** While deep-learning methods now dominate research literature, this booklet focuses on interpretable, compute-efficient algorithms—Random Forests, Hidden Markov Models, Support Vector Machines—that still power countless embedded and low-resource systems. A final chapter gestures toward contemporary neural approaches and suggests pathways for further study.

The material is intentionally modular. Instructors may select chapters à la carte to fit a ten-week special topic, a semester-long survey, or a short professional workshop. Readers pursuing self-study can follow the chapters sequentially or jump directly to those most relevant to their projects.

Acknowledgements. We thank the students of EE502 (Fall 2023–2025) whose insightful questions shaped successive drafts; The research assistants in the AI Vision, Health, Biometrics, and Applied Computing (AVHBAC) Lab at Clarkson University (<https://sites.clarkson.edu/avhbac/>) for curating datasets and vetting code examples; and our colleagues in the Wallace H. Coulter School of Engineering for their constructive feedback. Special gratitude goes to friends and family for their patience during the many late nights of writing and debugging.

Finally, no text can hope to be exhaustive in a field that advances as quickly as machine learning for audio. We encourage readers to consult the cited literature, experiment with new libraries, and—above all—listen critically to the sounds around them. We hope this booklet serves as a launch-pad for your own explorations.

*Masudul H. Imtiaz, PhD
Ajan Ahmed
Yash Sukhdev
Potsdam, New York
August 25, 2025*

Chapter 1: Introduction to Speech Signals

Imagine walking into a crowded room with your eyes closed. You can still recognize a friend’s voice calling your name, detect the laughter in a conversation nearby, or distinguish between the soft hum of background music and a distant announcement. This invisible world of sound carries rich layers of information—from the what of language to the how of its delivery, all embedded within fleeting air vibrations. In the realm of machine learning and signal processing, understanding and harnessing this acoustic complexity is both a scientific challenge and a technological opportunity. At the heart of this challenge lies a need to disentangle and analyze three core concepts—audio, speech, and voice—each playing a unique role in how machines hear, interpret, and respond to the world around them.

Although often used interchangeably, the terms **audio**, **speech**, and **voice** refer to distinct concepts in signal processing and machine learning applications. **Audio** is the broadest term, encompassing any sound signal that can be recorded, transmitted, or processed—whether it’s music, environmental noise, speech, or a combination of these. It refers to the raw acoustic waveform, typically represented as a time-series signal in analog or digital form. **Speech** is a specific subset of audio that refers to the structured verbal communication produced by humans. It carries linguistic content such as phonemes (the smallest units of sound that distinguish meaning in a language, e.g., /p/ in pat vs. /b/ in bat), words, and grammar, and is the focus of applications like automatic speech recognition (ASR), speech synthesis, and language identification. **Voice**, on the other hand, refers to the unique characteristics of an individual’s vocal production, shaped by both physiological and behavioral factors such as pitch, tone, timbre, and speaking style. While *speech* conveys linguistic content through structured language (e.g., words and sentences), *voice* represents the how of that production. In this sense, voice is a component of speech, but it can also exist independently—for example, in humming or laughter. Voice is especially important in tasks such as speaker identification, verification, and voice biometrics.

Understanding the distinction between these terms is critical when designing or analyzing systems in speech processing, speaker recognition, and audio classification. For instance, voice-based emotion recognition depends on how someone speaks (voice), not necessarily what they say (speech), while ASR systems focus primarily on transcribing the linguistic content (speech) from the broader audio signal.

Notation for Sounds and Letters. Throughout this book, we distinguish carefully between written letters, phonemes, and phones. The symbol ”**o**” refers to the orthographic letter as it appears in writing. The notation **/o/** represents a phoneme, which is an abstract linguistic category that distinguishes meaning within a language. Finally, the notation **[o]** designates a phone, the concrete spoken sound as physically produced—for example, the close-mid back rounded vowel. Thus, ”o”, **/o/**, and **[o]** are related but not identical: the first belongs to orthography, the second to the mental system of phonemes, and the third to the actual phonetic realization.

1.1 Analog vs. Digital Audio Signals (Continuous vs. Discrete)

To understand how machines process and interpret sound, we must first examine how audio signals are represented. **Sound** refers to the physical phenomenon of pressure waves traveling through a medium—typically air—that can be detected by the human ear. When these vibrations are captured by a sensor like a microphone, they are converted into an **audio signal**, which is an electrical or digital representation of the sound.

An analog audio signal is a continuous-time waveform—often in the form of a smoothly varying voltage—that directly corresponds to the air pressure fluctuations of the original sound [1]. For example, the electrical output of a microphone is an analog signal that varies continuously with the incoming sound wave. However, analog signals are difficult to store, transmit, and analyze using modern computing systems. To enable efficient processing, we convert them into **digital audio signals**, which are discrete-time sequences of numbers [2]. This conversion involves two main steps: *sampling* (measuring the signal’s amplitude at regular time intervals) and *quantization* (rounding each amplitude value to a fixed set of levels). The result is a sequence of digital samples that approximate the original waveform and can be stored, analyzed, or used as input to machine learning models.

When we refer to continuous vs. discrete signals, we mean continuous signals are defined for every instant in time (as with analog audio), whereas discrete signals are only defined at specific time points (the sample times). Figure 1.1 below illustrates this concept by showing a continuous waveform (green curve) and its sampled version (blue vertical lines at discrete intervals) [3]. The continuous signal $S(t)$ is defined for all t , while the discrete sequence $x[n]$ (samples) is only defined at $t = nT$ for some fixed sampling interval T . In essence, the digital signal is a time-sampled version of the analog signal, obtained by evaluating the continuous wave at evenly spaced points in time:

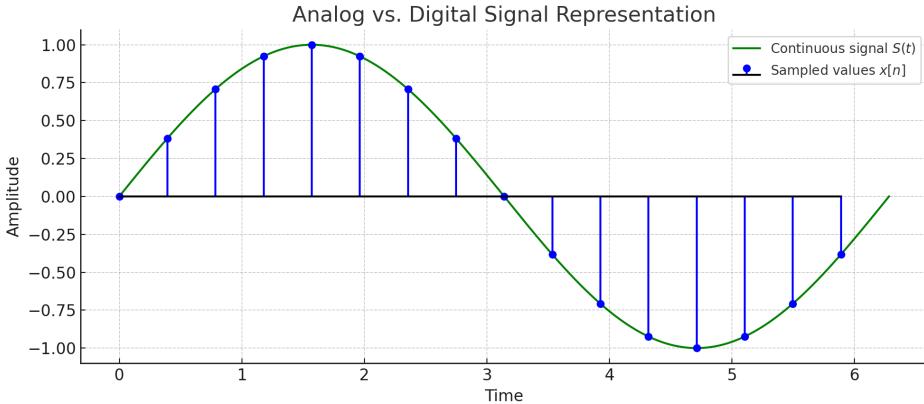


Figure 1.1: A continuous analog signal (green curve) and its discrete sampled values (blue stems). The samples capture the signal's amplitude at regular intervals. Each blue dot represents the signal value at a specific time, separated by the sampling period T [4].

This conversion to discrete time is fundamental to digital audio processing and machine learning applications: once in digital form, audio can be stored in a computer, analyzed, and used as input to algorithms. However, moving to the discrete domain introduces important considerations like sampling rate, Nyquist limit, aliasing, and quantization error, which we will discuss in this chapter.

1.2 Audio Signal Properties

Audio signals (and signals, in general) have several key properties that describe their behavior. To introduce these, consider a simple sinusoidal waveform (sine wave), which serves as a prototype for many audio signals. A sinusoid can be described by the equation:

$$x(t) = A \sin(2\pi ft + \phi)$$

where A is the amplitude, f is the frequency, and ϕ is the phase of the wave. Each of these parameters has an intuitive meaning:

Amplitude (A)

Amplitude controls the signal's strength or intensity. For a sound wave, amplitude corresponds to the peak deviation of the pressure from the ambient value – essentially, how ‘loud’ the sound is. In the sinusoid above, A is the peak height of the wave (the maximum displacement from zero) [5]. Higher amplitude means a louder sound (assuming the same listening conditions), whereas lower amplitude means a softer sound.

Frequency (f)

Frequency is the number of cycles that the waveform completes per second, measured in Hertz (Hz). It is the reciprocal of the wave’s period T (the duration of one cycle) [5]. For audio, frequency is related to the pitch of the sound: a high-frequency wave (e.g., 1000 Hz) sounds high-pitched, while a low-frequency wave (e.g., 100 Hz) sounds low-pitched. Humans can generally hear frequencies roughly in the range 20 Hz to 20 kHz. Pure tones like a 440 Hz sine wave (the musical note A4) have a single dominant frequency, whereas complex sounds contain a mixture of many frequencies.

Phase (ϕ)

Phase describes the waveform’s starting angle or offset at time $t = 0$. It indicates where in its cycle the sinusoid begins [5]. The phase is measured in radians or degrees; a phase of 0 means that the wave starts at zero amplitude and rises upward, while a phase of π (180°) would start at zero amplitude but initially decreases (inverted). Phase differences between signals can lead to constructive or destructive interference, but for a single isolated waveform, phase mainly just shifts it in time and does not affect the perceived sound in a steady tone.

1.3 Time-Domain vs. Frequency-Domain Representations

An audio signal can be represented in the time domain or the frequency domain, each providing a different perspective. The time-domain representation is the most straightforward – it is the waveform of the audio signal, showing how amplitude varies over time. For example, a plot of *air pressure vs. time* or *voltage vs. time* is a time-domain view. This is what you see when looking at a waveform in an audio editing program: the horizontal axis is time, and the vertical axis is amplitude. The time domain analysis answers the question ‘*What is the signal amplitude at each moment?*’ but it does not directly show which frequencies are present in the sound.

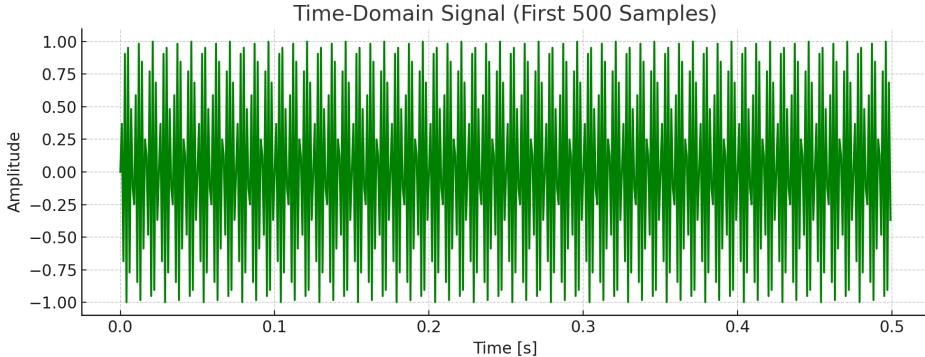


Figure 1.2: Time-domain representation of a sine wave (440 Hz). The plot shows how the signal varies with time.

The frequency-domain representation, on the other hand, describes the signal in terms of its frequency components. Through the *Fourier Transform*, we can decompose any time-domain signal into a sum of sinusoidal components at various frequencies. The result is a spectrum (aka Fourier spectrum) that reveals how much energy or amplitude the signal has at each frequency [6]. Instead of plotting *amplitude vs. time*, a spectrum plots *amplitude (or power) vs. frequency*.

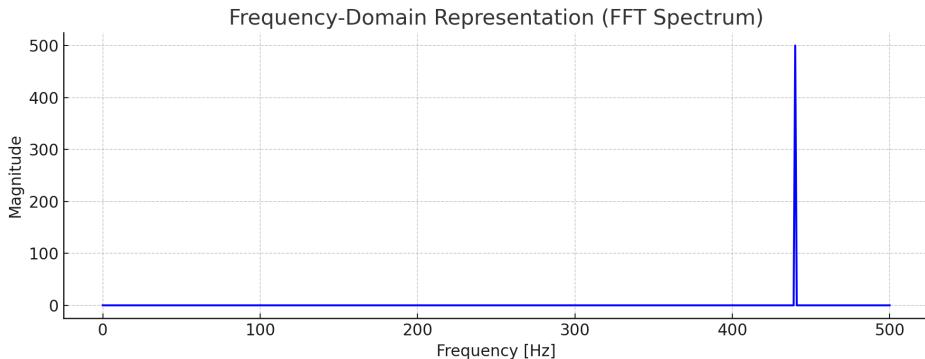


Figure 1.3: Frequency-domain representation (magnitude spectrum) of a 440 Hz sine wave. The spike at 440 Hz indicates the dominant frequency component.

Mathematically, the Fourier Transform provides the conversion from the time domain to the frequency domain. For a discrete-time signal (with N samples x_0, x_1, \dots, x_{N-1}), the discrete Fourier transform (DFT) is defined as:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i2\pi \frac{kn}{N}}$$

which produces N complex coefficients X_k representing the signal’s content at discrete frequencies [6]. Each X_k corresponds to a particular frequency (k/N of the sampling rate, in Hz) and encodes both amplitude and phase information of that frequency component.

Time vs. Frequency Example

Imagine a trumpet holding a note A4 (440 Hz). In the time domain, you would see a roughly periodic oscillation corresponding to the sound wave. In the frequency domain, you'd see a peak at 440 Hz (the fundamental frequency) and additional peaks at multiples of 440 Hz (the harmonics that give the trumpet its timbre). Both representations describe the same signal, but one emphasizes how it changes over time, while the other emphasizes what frequencies it contains [6]. Both views are essential in audio processing: the time domain is useful for examining temporal features (like waveform shape, transients, silence, etc.), whereas the frequency domain is crucial for analyzing spectral content (pitch, tone quality, formants in speech, etc.). As we delve into machine learning for audio, we will often convert signals to the frequency domain to extract features that are more informative for tasks like speech recognition.

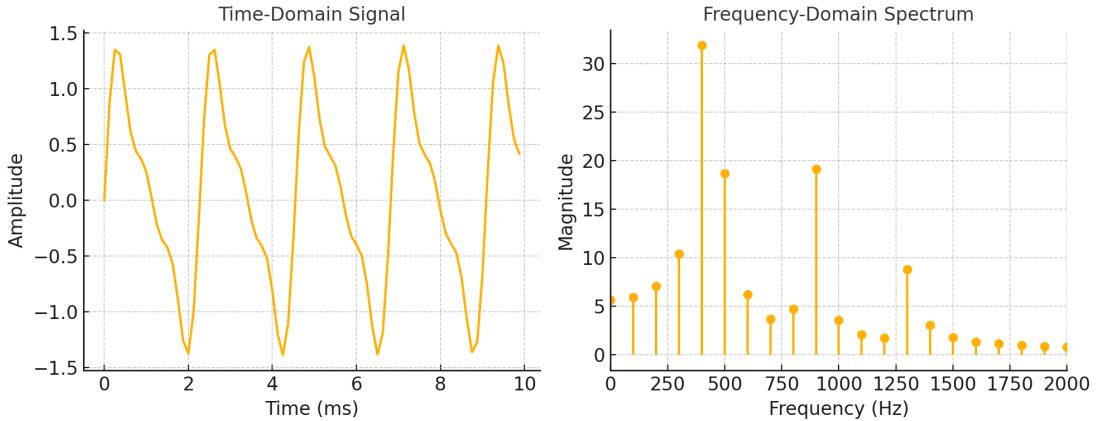


Figure 1.4: Time-domain (left) and frequency-domain (right) representations of a trumpet playing A4 (440 Hz). The frequency plot shows the fundamental and harmonic components.

Nyquist–Shannon Sampling Theorem

A crucial question arises: How fast do we need to sample to capture all the useful information in the signal? The Nyquist–Shannon sampling theorem provides the answer. It states that if a continuous signal is band-limited (meaning it contains no frequency components higher than a certain maximum frequency f_{\max}), then the signal can be perfectly reconstructed from its samples provided the sampling rate F_s is greater than twice that maximum frequency [4], [7]. The minimum necessary sampling rate $F_s = 2f_{\max}$ is called the **Nyquist rate**, and half the sampling rate ($F_s/2$) is called the **Nyquist frequency** [7].

In other words, the Nyquist frequency is the highest frequency that can be unambiguously represented at a given sampling rate. Any frequency content below $F_s/2$ will be captured correctly by sampling (under ideal conditions), while frequencies at or above $F_s/2$ cannot be represented correctly.

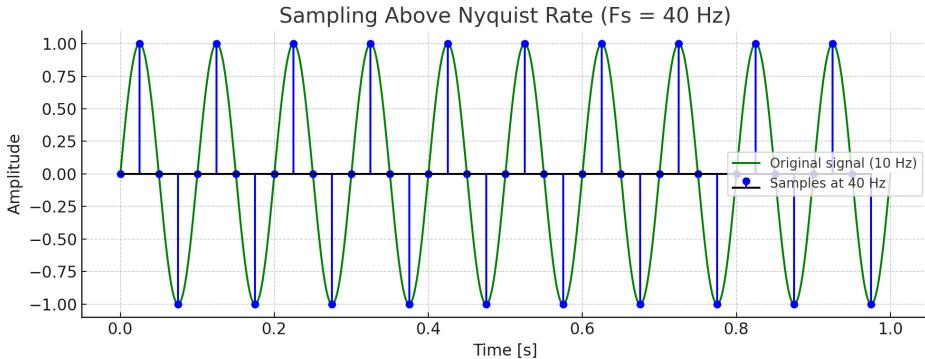


Figure 1.5: Sampling a 10 Hz sine wave at 40 Hz (above Nyquist rate). The signal is reconstructed accurately.

To illustrate, if human hearing goes up to about 20 kHz, a system aiming for full audible range needs a sampling rate above 40 kHz. The CD audio standard of 44.1 kHz was chosen for this reason – it can represent frequencies up to about 22.05 kHz, slightly above the human hearing limit, thus satisfying the Nyquist criterion for the audible band.

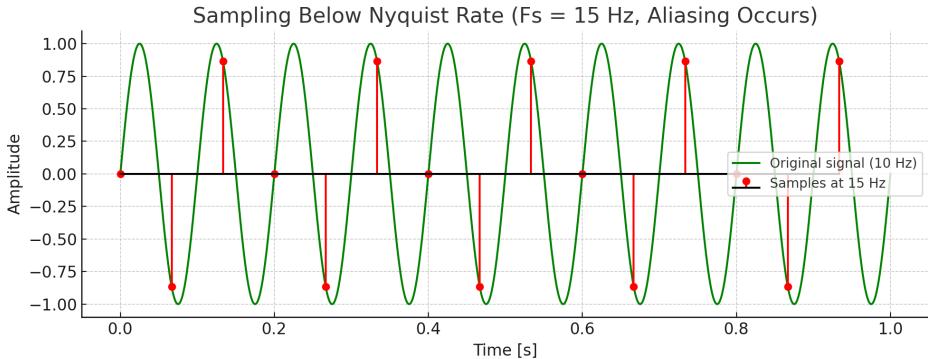


Figure 1.6: Sampling the same 10 Hz sine wave at 15 Hz (below Nyquist rate). The waveform is distorted due to aliasing.

In practical terms, the sampling theorem ensures that no information is lost in sampling as long as the signal does not contain frequencies higher than half the sampling rate. If the original signal contains higher frequencies, they must be removed (filtered out) before sampling to meet the bandlimited condition; otherwise, problems such as aliasing will occur.

1.4 Quantization and Bit Depth

Sampling in time (choosing discrete instants) is only one part of digitizing audio. The other part is quantization, which is the discretization of amplitude. Analog signals can take any amplitude value (within some range) and vary smoothly. But digital signals represent amplitudes as binary numbers with a fixed number of bits. Bit depth (or resolution) is the number of bits used to store each sample's amplitude value. Common PCM (pulse-code modulation) audio bit depths are 8-bit, 16-bit, 24-bit, etc., which correspond to $2^8 = 256$, $2^{16} = 65,536$, $2^{24} \approx 16.7$ million possible amplitude levels, respectively. The higher the bit depth, the finer the resolution of amplitude values. For example, CD audio uses 16 bits per sample, meaning each sample's amplitude is one of 65,536 possible values [8], [9].

When we quantize an analog sample to the nearest available digital level, there is usually a small difference between the true analog value and the chosen digital value. This difference is the quantization error. Over time, these errors manifest as a very low-level noise added to the signal, known as quantization noise. In essence, quantization is a rounding operation, and the quantization noise is the aggregate effect of rounding every sample [10]. In an ideal scenario (with certain statistical assumptions about the signal), the quantization error can be modeled as a random noise uniformly distributed between $-\frac{1}{2}$ and $+\frac{1}{2}$ least significant bit (the smallest step) [11]. The magnitude of this noise is directly related to the bit depth: fewer bits \Rightarrow larger quantization steps \Rightarrow more quantization noise, while more bits \Rightarrow smaller steps \Rightarrow less quantization noise.

A key metric here is the signal-to-noise ratio (SNR), which in this context compares the level of the original signal to the level of the quantization noise. Each extra bit of resolution improves the theoretical SNR by about 6 dB (because reducing the step size by a factor of 2 cuts the quantization noise power by 4, which is 6 dB) [12]. For instance, 16-bit audio has a theoretical maximum SNR of about 98 dB, whereas 8-bit audio tops out around 50 dB SNR [8], [9]. In practical terms, 8-bit audio may sound noticeably noisy or grainy in quiet passages (think of the background hiss in old video games), while 16-bit audio has a noise floor low enough to be inaudible in most listening situations. Professional audio with 24-bit depth provides even more headroom and noise margin (the noise floor is far below the threshold of hearing) [9].

It's important to note that while sampling rate limits the frequency range of the signal, the bit depth limits the *dynamic range* (the range between the loudest and softest representable sound) [8], [13]. Bit depth does not affect frequency response – a 16-bit recording and a 24-bit recording of the same signal will have the same frequencies present, but the 24-bit version can capture more subtle amplitude variations (and a larger ratio of loudest to quietest sound).

1.5 The Fourier Transform and Frequency Analysis

The Fourier transform is the mathematical operation that moves us from the time domain to the frequency domain. In audio and signal processing, it is fundamental for analyzing the spectral content of signals.

For a continuous signal $f(t)$, the (continuous) Fourier transform $F(\omega)$ is defined as:

$$F(\omega) = \int_{-\infty}^{\infty} f(t) e^{-j\omega t} dt$$

This yields a complex-valued function $F(\omega)$ that encodes both the amplitude and phase of each angular frequency ω component in the signal.

In digital signal processing, we work with sampled signals and use the *Discrete Fourier Transform (DFT)*—a numerical approximation of the continuous transform suited for digital computation. The DFT, or its efficient version called the Fast Fourier Transform (FFT), outputs a set of complex coefficients X_k [6], which describe the signal's content at discrete frequency bins.

Time vs. Frequency Representations

Figure 1.7 shows a time-domain signal composed of two sine waves—one at 100 Hz and another at 300 Hz. The signal appears as a complex waveform resulting from the superposition of these components.

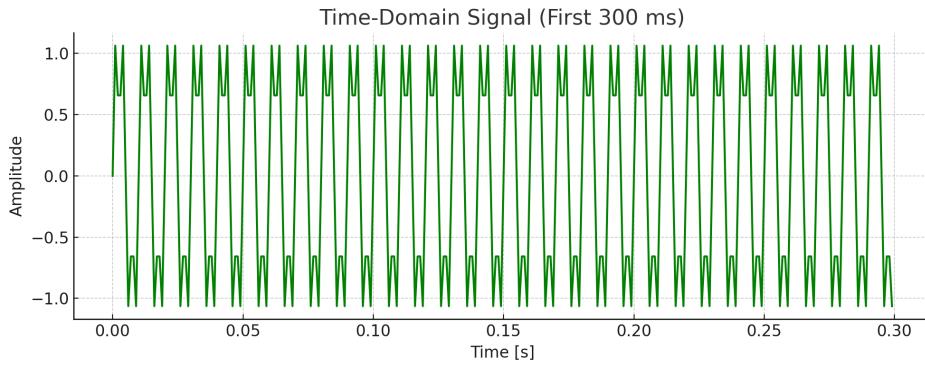


Figure 1.7: Time-domain signal composed of two sine waves at 100 Hz and 300 Hz.

When the signal is analyzed using the DFT, the result is a frequency-domain spectrum showing how much energy is present at each frequency. Figure 1.8 displays the magnitude of this spectrum in decibels (dB), clearly revealing peaks at 100 Hz and 300 Hz.

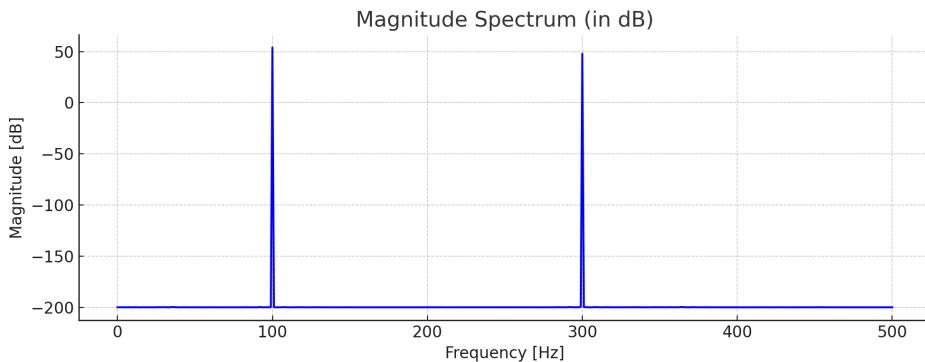


Figure 1.8: Magnitude spectrum (in decibels) showing dominant frequency components at 100 Hz and 300 Hz, computed using the DFT.

Interpreting the Fourier Transform

The magnitude of the Fourier transform $|X(f)|$ (or $|X_k|$ in discrete terms) is often the main quantity of interest, as it tells us how much of each frequency is present in the signal. The phase of each component, though important for perfect reconstruction, is less perceptually critical in many audio applications.

Visualizing the spectrum in dB is common for two reasons: (1) human loudness perception is roughly logarithmic, and (2) plotting on a logarithmic scale compresses the dynamic range, making both strong and weak components easier to compare [6].

For example, if you analyze a recording of the word “hello,” the Fourier transform might show that the vowel “e” contains strong energy in lower frequency bands (formants), while the “h” has energy spread across higher frequencies due to its noise-like nature. Thus, the Fourier transform provides a powerful way to see what frequencies are present and how strongly, but it doesn’t tell us *when* those frequencies occur—a limitation addressed by the short-time Fourier transform, which we will explore next.

1.6 Short-Time Fourier Transform (STFT) and Spectrograms

Speech and most real-world audio signals are *non-stationary*, meaning their spectral content changes over time. A single Fourier transform over the entire signal would yield a spectrum that represents an average of the frequency content over the whole duration—this is not very helpful for analyzing transient phonetic events or rapidly changing sounds [14].

To obtain frequency information localized in time, we use the *Short-Time Fourier Transform (STFT)*. This technique involves computing the Fourier Transform over short, overlapping segments (called *windows* or *frames*) of the signal. A window function (like Hamming or Hann) is slid across the signal, and at each step, a Fourier transform is applied to the windowed segment.

Mathematically, if $x[n]$ is the discrete signal and $w[n]$ is a window of length N , the STFT is defined as:

$$X(h, k) = \sum_{n=0}^{N-1} x[n + h] \cdot w[n] \cdot e^{-i2\pi \frac{kn}{N}},$$

where h indexes the time shift (window position) and k indexes the frequency bin.

STFT produces a time-frequency representation $X(h, k)$, which indicates the frequencies present in a signal and how they evolve over time. This representation is typically visualized using a *spectrogram*, a two-dimensional heatmap where the x-axis represents time, the y-axis represents frequency, and the color intensity (or brightness) corresponds to the magnitude or power, often expressed in decibels.

Spectrograms help us visualize various audio properties. *Voiced sounds*, such as vowels, appear as horizontal stripes known as formants, which represent the resonant frequencies of the vocal tract. *Unvoiced consonants*, like “s” and “f,” display diffuse, high-frequency energy without a clear harmonic structure. *Plosive sounds*, such as “p” and “t,” produce short, broadband bursts that appear as vertical streaks. Meanwhile, *silence* is represented by columns with very low intensity, often seen as dark regions in the spectrogram.

An example of a spectrogram generated from a speech signal containing the spoken phrase “nineteenth century” is shown in Figure 1.9. It captures how the frequency components of the signal vary with time, revealing dynamic speech patterns. For comparison, Figure 1.10 shows the magnitude spectrum obtained by applying the Discrete Fourier Transform (DFT) to the same audio segment. While the DFT spectrum provides information about which frequencies are present, it does not indicate when they occur—highlighting the need for time-frequency representations like the spectrogram.

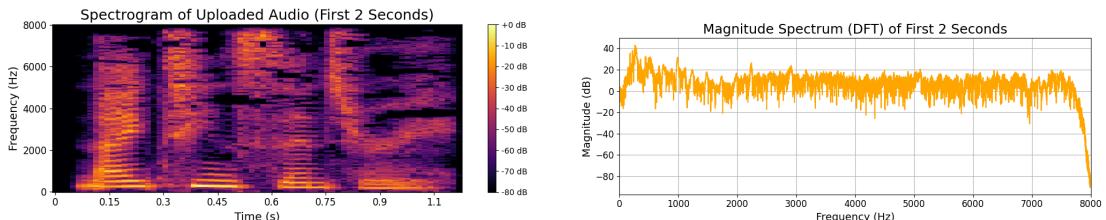


Figure 1.9: Spectrogram of the spoken words “nineteenth century” (first 2 seconds).

Figure 1.10: Magnitude spectrum (DFT) of the same “nineteenth century” speech segment.

Spectrograms are widely used in Machine Learning for audio. Systems like speech and emotion recognizers often take spectrograms as input because they compactly capture both temporal and spectral information—two key dimensions for understanding sound.

Mel-Scale and Mel-Spectrograms

The mel scale is a perceptual scale of pitches judged by listeners to be equal in distance from one another. It reflects how the human ear perceives sound: we are more sensitive to changes in lower frequencies than higher ones. The transformation from frequency f (in Hz) to mel is typically given by:

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

A *mel-spectrogram* is a variation of the spectrogram where the frequency axis is mapped to the mel scale and a bank of mel-spaced filters is applied. It better aligns with auditory perception and is widely used in machine learning tasks, especially speech recognition and synthesis.

The *spectrogram* displays energy distributed across linear frequency bins. In contrast, the *mel-spectrogram* emphasizes lower frequencies by allocating denser mel bins in that range while compressing higher frequencies with sparser bins, thereby mimicking the non-linear frequency resolution of human hearing. Because of this perceptual alignment, mel-spectrograms often yield better performance in tasks such as automatic speech recognition (ASR) and text-to-speech (TTS) synthesis.

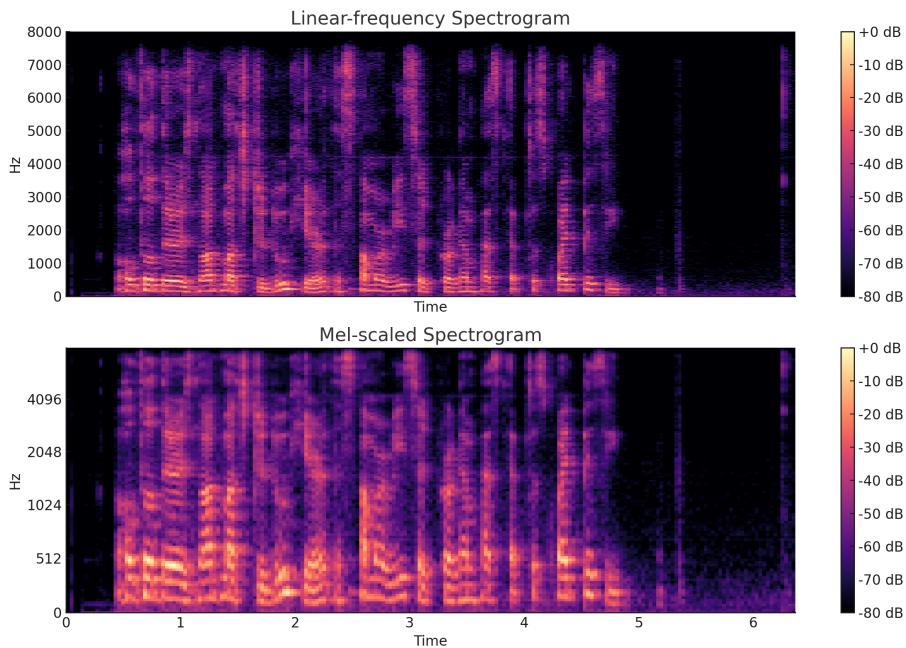


Figure 1.11: Comparison of a linear-frequency spectrogram (top) and a mel-spectrogram (bottom) of the same speech signal.

1.7 Practical Audio Systems

Having discussed sampling, frequency, and quantization, we now examine common configurations used in practical audio systems. These include typical values for sampling rates and bit depths, as well as the distinction between mono and stereo channels.

1.7.1 Common Sampling Rates

In speech and audio processing, the sampling rate determines how frequently the signal is measured in time. Common sampling rates in audio vary depending on the application. A sampling rate of **8 kHz** is typically used in telephony for narrowband speech, capturing the 300–3400 Hz voice band. **16 kHz** is considered wideband speech and is common in speech recognition and modern telecommunication systems. **44.1 kHz** is the standard for CD-quality audio and is sufficient to capture the full range of human hearing (20 Hz–20 kHz). **48 kHz** is frequently used in professional audio and multimedia applications. Higher rates, such as **96 kHz** and **192 kHz** are used in studio production or scientific applications

that require oversampling <empty citation> For most machine learning applications involving speech, 16 kHz strikes a good balance between fidelity and data efficiency.

1.7.2 Bit Depths and Quantization

Bit depth defines the resolution of each sample's amplitude. **8-bit** audio was historically used in telephony and often employed μ -law or A-law companding to increase its effective dynamic range. **16-bit** audio is the standard for CD-quality recordings, offering about 96 dB of dynamic range [15]. **24-bit** audio is commonly used in professional audio recording, with a theoretical dynamic range of 144 dB, though practical implementations typically achieve around 120 dB.

In speech datasets, the **16-bit** PCM is common. While lower bit depths (e.g., 8-bit) save space, they introduce quantization noise and are generally avoided unless necessary due to storage or bandwidth constraints.

1.7.3 Mono vs. Stereo Channels

Audio may be recorded and played back using one or more channels. **Mono** refers to single-channel audio, where the same signal is played through all speakers or headphones. It is commonly used in most speech and voice recognition tasks. In contrast, **stereo** refers to two-channel audio—left and right—where different signals are sent to each ear, creating a spatial or directional perception. This is also similar in microphones: a mono channel microphone uses a single microphone element, resulting in a single channel of audio, while a stereo microphone uses two or more elements, capturing audio in two or more channels for a wider, more immersive sound.

Digital stereo audio consists of two PCM streams. For example, a file labeled “16-bit, 44.1 kHz stereo” contains two separate channels, each sampled at 44.1 kHz with 16-bit resolution.

1.7.4 Practical Implications

The sampling rate, bit depth, and number of channels together define the digital audio format. A recording described as “16 kHz, 16-bit, mono PCM” means it has 16,000 samples per second, each sample is quantized into one of 65,536 levels, and the audio contains a single channel.

These parameters directly influence several factors. *Audio quality and size* are affected, as higher sampling rates and bit depths improve fidelity, but increase storage and computation requirements. For machine learning, the sampling rate determines the frequency resolution of the derived features, such as spectrograms. Using mono audio helps reduce model complexity. Preprocessing operations such as downsampling, bit-depth conversion, and stereo-to-mono conversion are often necessary before feeding data into models.

Understanding these fundamentals is essential before applying Machine Learning to audio data. Future chapters will build on this foundation through signal enhancement, feature extraction, and classification techniques.

Bibliography

- [1] Wikipedia contributors, *Analog signal*, Accessed June 22, 2025, 2024. [Online]. Available: https://en.wikipedia.org/wiki/Analog_signal.
- [2] Wikipedia contributors, *Digital audio*, Accessed June 22, 2025, 2024. [Online]. Available: https://en.wikipedia.org/wiki/Digital_audio.
- [3] Wikipedia contributors, *Discrete-time signal*, Accessed June 22, 2025, 2024. [Online]. Available: https://en.wikipedia.org/wiki/Discrete-time_signal.
- [4] A. Erlangen, *Sampling and quantization - audiolabs*, Accessed June 22, 2025, 2024. [Online]. Available: https://www.audiolabs-erlangen.de/resources/MIR/FMP/C1/C1S2_Sampling.html.
- [5] A. Erlangen, *Basic signal parameters - audiolabs*, Accessed June 22, 2025, 2024. [Online]. Available: https://www.audiolabs-erlangen.de/resources/MIR/FMP/C1/C1S1_BasicSignalParameters.html.
- [6] Aalto University, Department of Signal Processing, *Time-frequency representations and the fourier transform*, Accessed June 22, 2025, 2024. [Online]. Available: <https://wiki.aalto.fi/display/ITSP/Time-Frequency+Representations+and+the+Fourier+Transform>.
- [7] Wikipedia contributors, *Nyquist–shannon sampling theorem*, Accessed June 22, 2025, 2024. [Online]. Available: https://en.wikipedia.org/wiki/Nyquist%E2%80%93Shannon_sampling_theorem.
- [8] Wikipedia contributors, *Pulse-code modulation*, Accessed June 22, 2025, 2024. [Online]. Available: https://en.wikipedia.org/wiki/Pulse-code_modulation.
- [9] Wikipedia contributors, *Audio bit depth*, Accessed June 22, 2025, 2024. [Online]. Available: https://en.wikipedia.org/wiki/Audio_bit_depth.
- [10] Wikipedia contributors, *Quantization (signal processing)*, Accessed June 22, 2025, 2024. [Online]. Available: [https://en.wikipedia.org/wiki/Quantization_\(signal_processing\)](https://en.wikipedia.org/wiki/Quantization_(signal_processing)).
- [11] Wikipedia contributors, *Quantization noise*, Accessed June 22, 2025, 2024. [Online]. Available: https://en.wikipedia.org/wiki/Quantization_noise.
- [12] Wikipedia contributors, *Signal-to-noise ratio*, Accessed June 22, 2025, 2024. [Online]. Available: https://en.wikipedia.org/wiki/Signal-to-noise_ratio.
- [13] Wikipedia contributors, *Dynamic range (audio)*, Accessed June 22, 2025, 2024. [Online]. Available: [https://en.wikipedia.org/wiki/Dynamic_range_\(audio\)](https://en.wikipedia.org/wiki/Dynamic_range_(audio)).
- [14] Wikipedia contributors. “Spectrogram.” (2025), [Online]. Available: <https://en.wikipedia.org/wiki/Spectrogram> (visited on 06/22/2025).
- [15] Wikipedia contributors, *Pulse-code modulation — wikipedia, the free encyclopedia*, https://en.wikipedia.org/wiki/Pulse-code_modulation, https://en.wikipedia.org/wiki/Pulse-code_modulation, 2024.

Chapter 2: Speech Preprocessing and Feature Extraction in Speech

Machine learning models require informative, standardized inputs. Raw speech is seldom fed directly into an algorithm. In speech processing, **speech preprocessing** and **feature extraction** are crucial steps that clean the signal and transform it into representative features.

Preprocessing techniques (like noise reduction or normalization) enhance quality and consistency, ensuring models focus on relevant speech content. Feature extraction then converts the cleaned waveform into numerical descriptors capturing characteristics like frequency content, energy, and spectral shape.

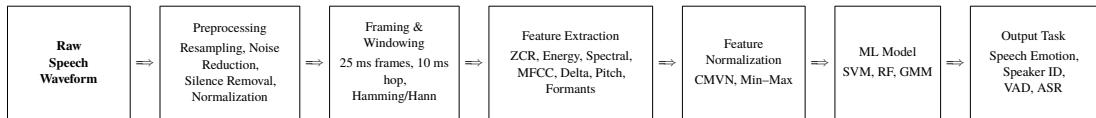


Figure 2.1: Simple block diagram of a classical ML pipeline for speech processing.

2.1 Common Preprocessing Steps for Speech

Before extracting features, raw speech recordings typically undergo the following preprocessing operations.

2.1.1 Resampling

Resampling converts every recording to a single sampling rate so that all subsequent feature-extraction code receives comparable input. A rate of 16 kHz is the industry work-horse because it captures frequencies up to 8 kHz—enough to preserve the fricatives, stop bursts, and formant transitions that modern ASR, speaker-ID, and emotion models rely on—while keeping file size and computation modest. Dropping to 4 kHz or even 8 kHz throws away those high-frequency cues and produces “telephone” quality speech, which hurts recognition accuracy; pushing higher to 22–48 kHz yields little extra information for speech while tripling both storage and GPU time. Using one uniform rate also guarantees that FFT dimensions, mel-filter banks, and pre-trained model inputs line up exactly, preventing the silent errors that arise when clips of mixed sample rates flow through the same pipeline. For these practical reasons, the most public corpora (e.g., LibriSpeech, VoxCeleb, CommonVoice) and toolkits (Kaldi, SpeechBrain) mandate 16 kHz, and we resample all incoming audio to that rate before any further processing.

In short, 16 kHz is the practical compromise: it keeps the speech detail ML models need, avoids the mud of narrow-band audio, and saves time and storage compared to studio-rate files. That is why most public corpora (LibriSpeech, VoxCeleb, CommonVoice) and toolkits (Kaldi, SpeechBrain) standardize on 16 kHz, and why we resample all input audio to this rate before further processing.

In Python, this can be done using `librosa`:

```
import librosa
y, sr = librosa.load("file.wav", sr=16000)
```

This loads and resamples the file to 16 kHz. Resampling mitigates mismatched sampling rates and simplifies downstream processing.

This code assumes Python 3.8 or newer. `librosa` is a popular Python library built on NumPy and SciPy that provides simple functions for loading audio, resampling, and extracting features like MFCCs, chroma, and spectrograms.

2.1.2 Noise Reduction

Speech recordings may contain background noise, such as environmental sounds, mechanical hum, electronic interference, or distant speech. Filters and denoising algorithms can remove these unwanted components to improve the clarity of the signal and the quality of extracted features.

Example: Apply a low-pass filter to remove high-frequency noise:

```

import numpy as np
from scipy.signal import butter, filtfilt

b, a = butter(N=4, Wn=4000, btype='low', fs=16000)
filtered_signal = filtfilt(b, a, signal)

```

This example uses `numpy` for array operations and `scipy.signal` for digital filtering. The function `butter` designs a 4th-order Butterworth filter—a type of filter with a maximally flat frequency response in the passband, meaning it avoids ripples and preserves the overall signal shape. The filter is set as low-pass with a cutoff at 4 kHz, meaning it retains frequencies below 4 kHz and attenuates higher ones, which often contain unwanted noise such as hiss, static, or electrical interference.

Low-pass filters are useful when you want to preserve speech fundamentals (typically under 4 kHz) and discard high-frequency noise. Conversely, high-pass filters remove low-frequency rumble (like air conditioners or mic handling noise) and preserve higher-frequency speech elements. The choice depends on the noise profile and the speech content.

A higher filter order results in a steeper cutoff slope, meaning the transition between passband and stopband becomes sharper. However, higher-order filters also increase computational cost and may introduce phase distortion if not applied properly.

2.1.3 Silence Removal (Trimming)

In speech recordings, silence refers to portions of the audio where there is little or no vocal activity—often appearing at the beginning or end of a clip as the speaker pauses, prepares to speak, or finishes. These segments have very low amplitude and typically contain only ambient background noise. While such silences are natural in human conversation, they are often irrelevant for machine learning models and can skew features like energy or duration. Removing them helps focus on the active speech content, improves efficiency, and results in cleaner feature representations.

This can be done using `librosa`:

```

y, sr = librosa.load("speech.wav", sr=16000)
yt, index = librosa.effects.trim(y, top_db=60)

```

This removes regions where amplitude is 60 dB below the peak. Figure 2.2 illustrates this.

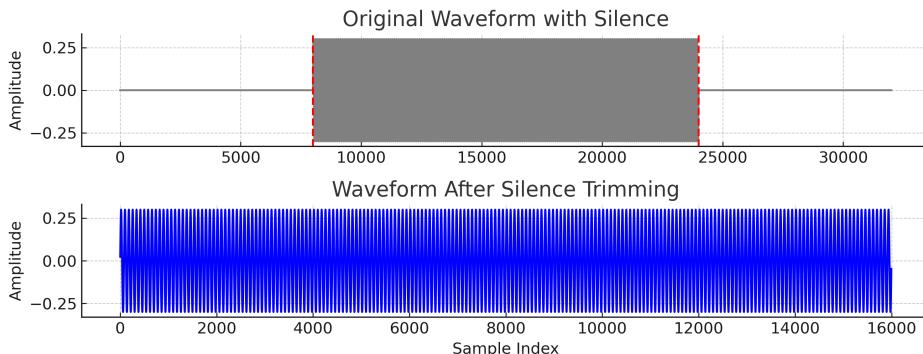


Figure 2.2: A speech waveform with silence (top) and after trimming (bottom). Dashed lines show segment boundaries.

2.1.4 Normalization

Normalization adjusts amplitude to a consistent level to prevent volume differences from affecting features. Peak normalization scales the waveform to [-1, 1]:

```

y = y / np.max(np.abs(y))

```

RMS normalization adjusts the root mean square energy across clips. This prevents models from misinterpreting volume-related feature differences.

2.1.5 Other Preprocessing Steps

- **Downmixing stereo to mono:** Many audio recordings are captured in stereo with separate left and right channels. For speech processing, models typically require a single-channel (mono) input. Downmixing is done by averaging the two channels to produce a unified signal. This ensures consistency across the dataset and avoids bias from channel imbalance.
- **Pre-emphasis:** This technique applies a simple high-pass filter to slightly boost the energy of high-frequency components in the signal. Since higher frequencies (such as those from fricatives like /s/ and /f/) tend to be weaker, pre-emphasis helps highlight these important cues, making them more noticeable in spectral analysis and feature extraction.
- **Length handling:** Machine learning models often require fixed-length input. If an audio clip is too short, *padding* is used which is adding extra zeros to the end to reach the required length. If a clip is too long, it may be segmented into smaller chunks. This ensures that all input samples are of uniform duration, allowing batch processing during training and inference.

2.2 Short-Time Analysis: Framing and Windowing

Speech signals are inherently non-stationary, meaning their statistical properties, such as energy, frequency content, or amplitude distribution, change over time as different phonemes and words are spoken. For example, the vowel sound /a/ (as in “father”) has strong, low-frequency energy and a relatively steady waveform, while a fricative like /s/ (as in “sun”) has high-frequency, noise-like energy. These differences cause the spectral and temporal characteristics of the signal to vary continuously as a person speaks. However, over short time durations (typically 20–30 ms), speech signals can be considered quasi-stationary. This assumption allows us to analyze them frame-by-frame using techniques that apply to stationary signals.

2.2.1 Framing

We divide the time-domain signal into short, overlapping segments known as frames. A common choice is:

- **Frame length:** 25 ms (= 400 samples at 16 kHz) — This duration is short enough to assume the speech signal is quasi-stationary (i.e., its properties remain stable within that time window), yet long enough to capture a few pitch cycles and phonetic content for reliable analysis.
- **Frame hop (stride):** 10 ms (= 160 samples) — The hop defines how much the window advances between frames. A smaller hop size increases overlap, allowing smoother feature transitions and better resolution of fast-changing sounds.

Overlapping is essential to ensure that important signal transitions are not missed. With 50–60% overlap, we maintain continuity across time and effectively capture transient details of speech sounds.

```
frame_length = int(0.025 * sr) # 25 ms
hop_length = int(0.010 * sr) # 10 ms
frames = librosa.util.frame(y, frame_length=frame_length, hop_length=hop_length)
print(frames.shape) # (frame_length, num_frames)
```

Each column in the resulting array represents one short-time frame to be processed independently.

2.2.2 Windowing

When speech signals are divided into short frames, applying a hard cut at the frame boundaries can introduce abrupt discontinuities. These discontinuities cause *spectral leakage* during Fourier analysis, where energy spreads across frequency bins instead of being concentrated at the true frequency components.

To mitigate this effect, each frame is multiplied by a *window function* (e.g., Hamming or Hann), which smoothly tapers the signal toward zero at the edges. The windowed signal is defined as

$$x_w[n] = x[n] \cdot w[n],$$

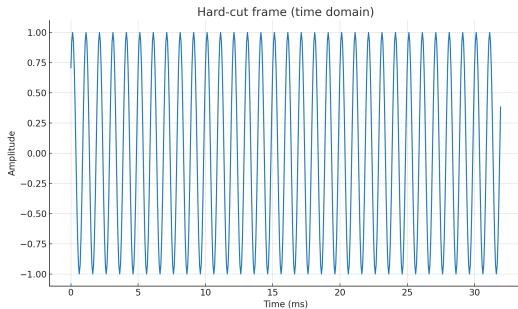
where $x[n]$ is the original frame and $w[n]$ is the window.

```

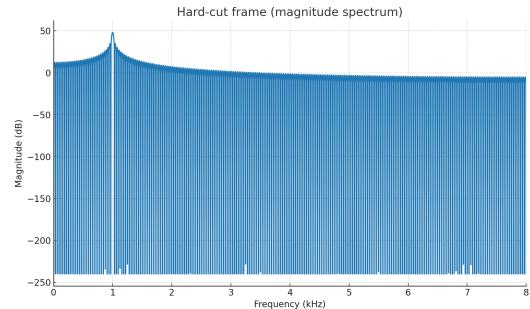
window = np.hamming(frame_length)
windowed_frame = frames[:, i] * window # for frame i

```

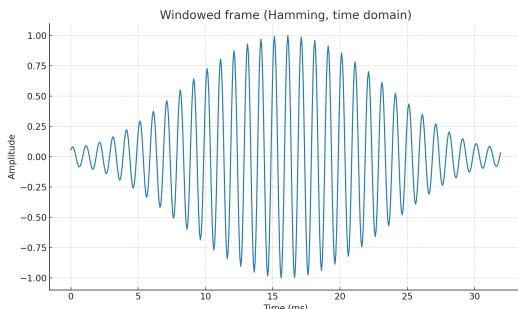
Figure 2.3 demonstrates the effect. In the hard-cut frame, sharp edges introduce strong discontinuities, visible in both the time-domain waveform and the frequency-domain spectrum as significant leakage. By contrast, applying a Hamming window smooths the edges of the frame, which reduces leakage and concentrates spectral energy around the main frequency components.



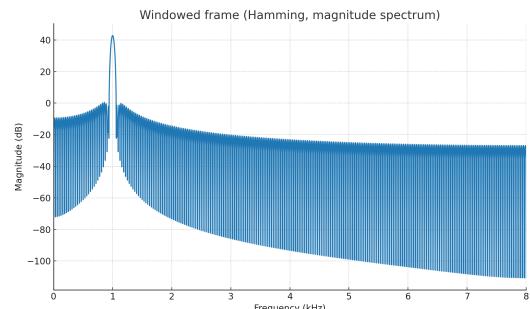
(a) Hard-cut frame (time domain).



(b) Hard-cut frame spectrum showing leakage.



(c) Windowed frame (Hamming, time domain).



(d) Windowed frame spectrum with reduced leakage.

Figure 2.3: Effect of windowing on frame boundaries. Windowing reduces discontinuities and minimizes spectral leakage compared to a hard-cut frame.

Hamming vs. Hann Windows: Both windows taper the frame edges to reduce spectral leakage, but with slightly different characteristics. The Hann window provides slightly better frequency resolution due to its narrower main lobe, making it useful when distinguishing closely spaced frequencies. The Hamming window, on the other hand, offers stronger side-lobe attenuation, which suppresses leakage more effectively and is often preferred in speech analysis. In practice, the choice depends on whether minimizing leakage (Hamming) or maximizing frequency resolution (Hann) is more important for the application.

2.2.3 Spectrogram: Time-Frequency Representation

By computing the DFT of each windowed frame, we obtain a spectrogram. Each column represents the spectrum of a single frame.

The spectrogram visually shows:

- Voiced sounds (e.g., vowels) as horizontal harmonic bands.
- Unvoiced sounds (e.g., /s/, /f/) as broadband noise.
- Silence as blank regions with minimal energy.

Short-time analysis forms the basis of all advanced feature extraction methods in speech processing, including MFCCs and filterbanks, covered in the following sections.

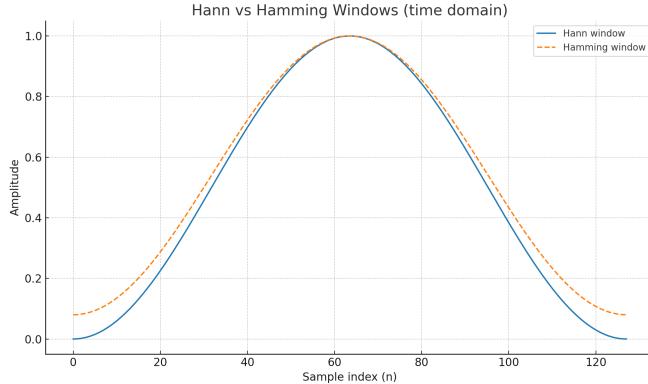


Figure 2.4: Comparison of Hann and Hamming windows in the time domain. The Hann window has a narrower main lobe (better frequency resolution), while the Hamming window offers stronger side-lobe attenuation (better leakage suppression).

2.3 Feature Extraction and Common Speech Features

Once the speech audio has been preprocessed and segmented into short-time frames, the next step is to extract meaningful features from each frame. These features quantify important characteristics of the speech signal — such as its frequency content, energy, and temporal dynamics — that can be used by machine learning algorithms.

Rather than feeding raw waveform samples to models, we derive a compact representation that reflects the essential properties of the speech audio. Feature extraction typically operates on a per-frame basis, meaning each short segment (e.g., 25 ms) is transformed into a feature vector. These feature vectors can then be used directly for classification tasks or aggregated (e.g., via averaging or statistical descriptors) for fixed-length representations.

Below we introduce the most widely used speech features in classical (non-deep) machine learning workflows.

2.3.1 Zero-Crossing Rate (ZCR)

The **zero-crossing rate** (ZCR) measures how frequently the speech audio waveform crosses the zero-amplitude axis. It counts the number of times the signal’s sign changes from positive to negative or vice versa within a frame. ZCR is a simple time-domain feature indicative of a signal’s frequency content or “roughness.”

Voiced speech (like vowels) tends to have low ZCR due to smoother, periodic patterns. Unvoiced sounds (such as fricatives /s/ or /f/) are noise-like and exhibit high ZCR due to rapid sign changes. Silence generally has few or no zero-crossings aside from background noise.

Mathematical Definition

The ZCR of a frame with N samples is computed as:

$$\text{ZCR} = \frac{1}{N - 1} \sum_{n=1}^{N-1} \mathbf{1}\{x[n] \cdot x[n - 1] < 0\}$$

Here, $\mathbf{1}\{\cdot\}$ is the indicator function, which evaluates to 1 when the sign of adjacent samples differs.

Python Example Using librosa

```
import librosa

y, sr = librosa.load("speech.wav", sr=16000)
frame_length = int(0.025 * sr) # 25 ms
hop_length = int(0.010 * sr) # 10 ms

zcr = librosa.feature.zero_crossing_rate(
    y, frame_length=frame_length, hop_length=hop_length
)
```

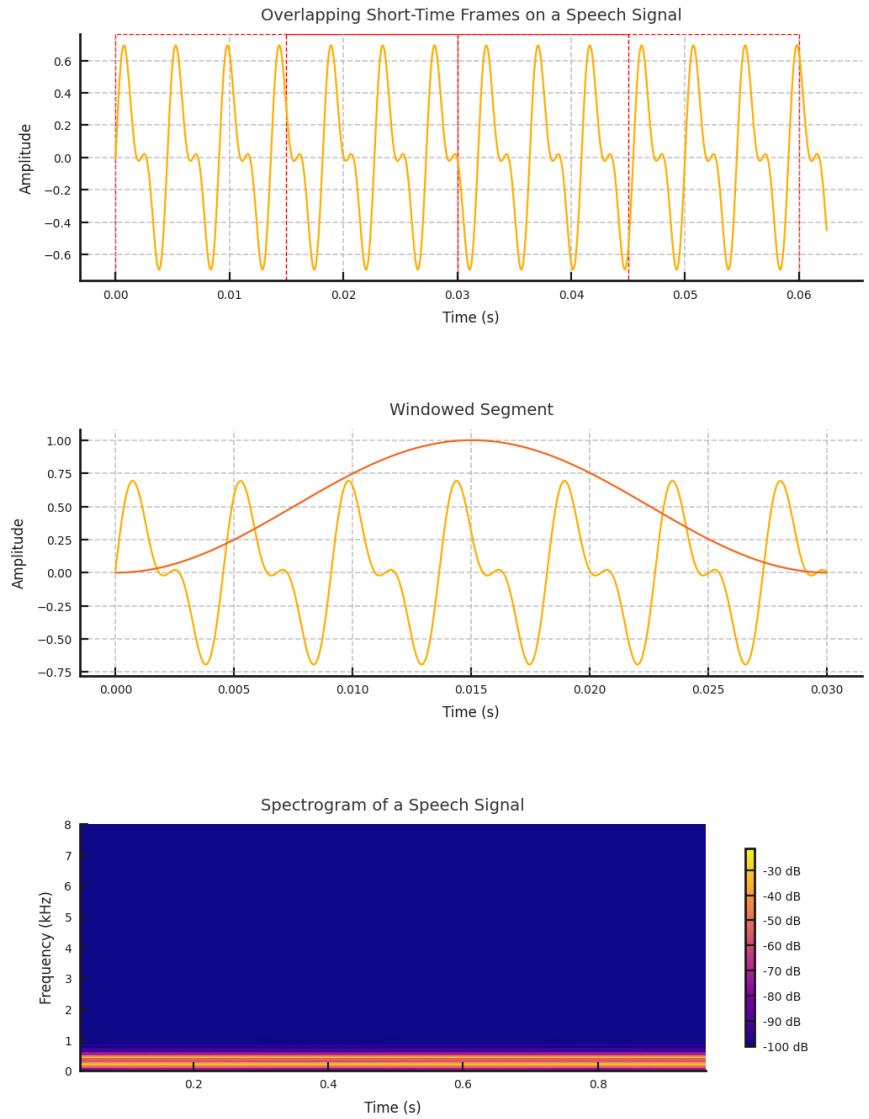


Figure 2.5: Illustration of the short-time analysis process. Top: speech waveform; Middle: framing and windowing; Bottom: resulting spectrogram showing frequency over time. Brighter colors represent higher energy.

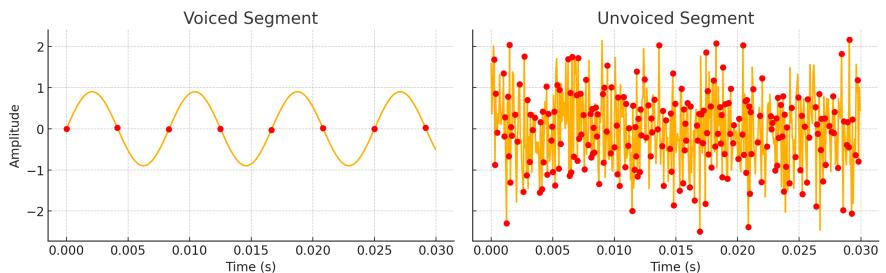


Figure 2.6: Illustration of Zero-Crossing Rate. Red dots indicate zero-crossing points of a waveform. Voiced (left) vs. unvoiced (right) segment.

This returns a 2D array where each value corresponds to the ZCR of a frame. The range of ZCR values is between 0 and 0.5 for normalized speech audio (since at most, every other sample can be a crossing).

Best Practices and Use Cases

- Ensure the signal is DC-centered (mean zero) before computing ZCR.
- ZCR is effective for differentiating voiced vs. unvoiced speech and detecting silence.
- It is computationally inexpensive and thus common in embedded speech systems or early-stage filters.

While ZCR alone is too basic for high-accuracy speech tasks, it is useful as a supporting feature in a larger feature set.

2.3.2 Energy and Spectral Features

Energy [1] represents the signal's loudness or intensity in a frame. There are a couple of closely related measures: short-time energy (the sum of squared amplitudes in the frame) and RMS (Root Mean Square) energy (the square root of the average of squared amplitudes). In a frame with samples $x[n]$, the energy can be defined as:

$$E = \sum_{n=1}^N x[n]^2$$

The RMS is:

$$\text{RMS} = \sqrt{\frac{1}{N} \sum_{n=1}^N x[n]^2}$$

which is just the standard deviation of the samples (assuming zero-mean). Energy is often normalized by frame length for consistency. For example, if $x[n]$ is in range $[-1, 1]$, energy will be in $[0, N]$ and RMS in $[0, 1]$.

Intuitively, energy/RMS indicates how loud a frame is. Speech segments have higher energy than silence; voiced parts (especially vowels) often carry more energy than unvoiced parts, since voiced sounds are produced with vocal cord vibration and generally higher amplitude. Thus, short-time energy is fundamental for tasks like voice activity detection (VAD) [2] – distinguishing speech from silence/noise. It is also used in endpointing (finding the start/end of an utterance) and as a basic feature for emotion (loudness can correlate with anger or excitement).

Calculation: Using numpy, one can compute frame energy straightforwardly. Librosa also provides `librosa.feature.rms`, which gives the RMS energy per frame. For example:

```
rms = librosa.feature.rms(y=y, frame_length=frame_length, hop_length=hop_length)
```

This returns an array of RMS values. We could square these to get actual energy per frame if needed (since RMS is $\sqrt{\text{mean of } x^2}$). As an example, imagine a frame of a voiced vowel [a] – its energy might be significantly higher than a frame of an [s] sound, which in turn is higher than pure silence (energy = 0). In practice, we often convert energy to a decibel (log) scale for dynamic range compression, defined as:

$$E_{\text{dB}} = 10 \log_{10}(E)$$

This log-energy is used in many speech features (e.g., MFCC includes a log-energy or C0 term).

Use cases: Apart from VAD, energy features can help in emotion detection (e.g., shouting versus speaking softly) and stress detection. In speech recognition, energy was historically one of the features used (alongside MFCCs). It is also useful for silence trimming – segments where energy falls below a threshold can be considered silence.

Note: Energy is influenced by recording gain; hence, the importance of normalization. After normalization of the amplitude across a data set, the energy differences become more meaningful for comparing frames within and between recordings.

Spectral Features (Centroid, Bandwidth, Contrast, Rolloff)

A speech frame transformed to the frequency domain (via the short-time Fourier transform) yields a spectrum; we can derive many spectral features that summarize the shape of this spectrum. These features quantify qualities often related to timbre or articulation characteristics of the speech. Common spectral descriptors include:

Spectral Centroid: The “center of mass” of the spectrum. It is computed as the weighted mean of frequencies present in the signal, with magnitudes as weights. If $X(k)$ is the magnitude of the FFT at frequency bin f_k , the centroid is:

$$\text{Centroid} = \frac{\sum_k f_k \cdot |X(k)|}{\sum_k |X(k)|}$$

A high centroid means that the spectrum’s energy is centered at high frequencies (perceived as bright or thin sound), whereas a low centroid indicates dominance of low frequencies (a darker or bass-rich sound). In speech, the centroid can differentiate sounds like “s” (which has energy in higher frequencies, thus a high centroid) from a vowel “u” (energy concentrated in low frequencies, a low centroid) [3]. In Python:

```
cent = librosa.feature.spectral_centroid(y=y, sr=sr, n_fft=1024, hop_length=
                                         hop_length)
```

This returns the centroid frequency for each frame (in Hz).

Spectral Bandwidth: This measures the spread of the spectrum around the centroid. Essentially, it is the standard deviation of the frequency distribution (the second central moment). A larger bandwidth means frequencies in the frame cover a wide range (e.g., noisy sounds with energy spread across the spectrum), while a small bandwidth indicates a narrow, peaky spectrum (perhaps a pure tone or a sound with most energy in a narrow band). In speech, a voiced vowel with distinct formant peaks might have lower bandwidth (energy clustered in formant regions), whereas an unvoiced noise has high bandwidth [1].

```
bw = librosa.feature.spectral_bandwidth(y=y, sr=sr, n_fft=1024, hop_length=
                                         hop_length)
```

Spectral Contrast: Spectral contrast evaluates the difference in amplitude between the peaks (high energy) and valleys (low energy) of the spectrum in different frequency sub-bands. The idea is to capture the dynamic range of the spectrum. A frame with a few prominent spectral peaks (e.g., a voiced speech sound with strong formant harmonics) will have a high contrast (big difference between peak energy and troughs between formants). A noisy frame with a flatter spectrum (or a frame of silence) will have low spectral contrast. This feature can help distinguish speech texture – e.g., voiced speech (harmonic structure → higher contrast) vs unvoiced noise (flat spectrum → lower contrast).

```
contrast = librosa.feature.spectral_contrast(y=y, sr=sr, n_fft=1024, hop_length=
                                              hop_length)
```

This computes contrast across a few frequency bands. It yields a vector per frame (each entry for one band’s contrast), which is often averaged or used as multiple features.

Spectral Rolloff: The rolloff frequency is the frequency below which a certain percentage (commonly 85%) of the total spectral energy lies. It essentially marks the “tail” of the spectrum. A lower rolloff frequency means most energy is concentrated in low frequencies; a higher rolloff indicates significant high-frequency content. For speech, an unvoiced consonant with energy in high frequencies will have a high rolloff (the 85% energy point is at a high frequency), whereas a low-frequency biased frame (like a vowel [o]) will have a lower rolloff.

```
rolloff = librosa.feature.spectral_rolloff(y=y, sr=sr, hop_length=hop_length,
                                           roll_percent=0.85)
```

```

y, sr = librosa.load("speech.wav", sr=16000)
# Compute spectral features for each frame
centroid = librosa.feature.spectral_centroid(y=y, sr=sr)
bandwidth = librosa.feature.spectral_bandwidth(y=y, sr=sr)
contrast = librosa.feature.spectral_contrast(y=y, sr=sr)
rolloff = librosa.feature.spectral_rolloff(y=y, sr=sr, roll_percent=0.85)

```

Typically these arrays are shape $(1 \times \text{frames})$ except contrast $(n_{\text{bands}} \times \text{frames})$. You might take the mean over time or use them frame-wise.

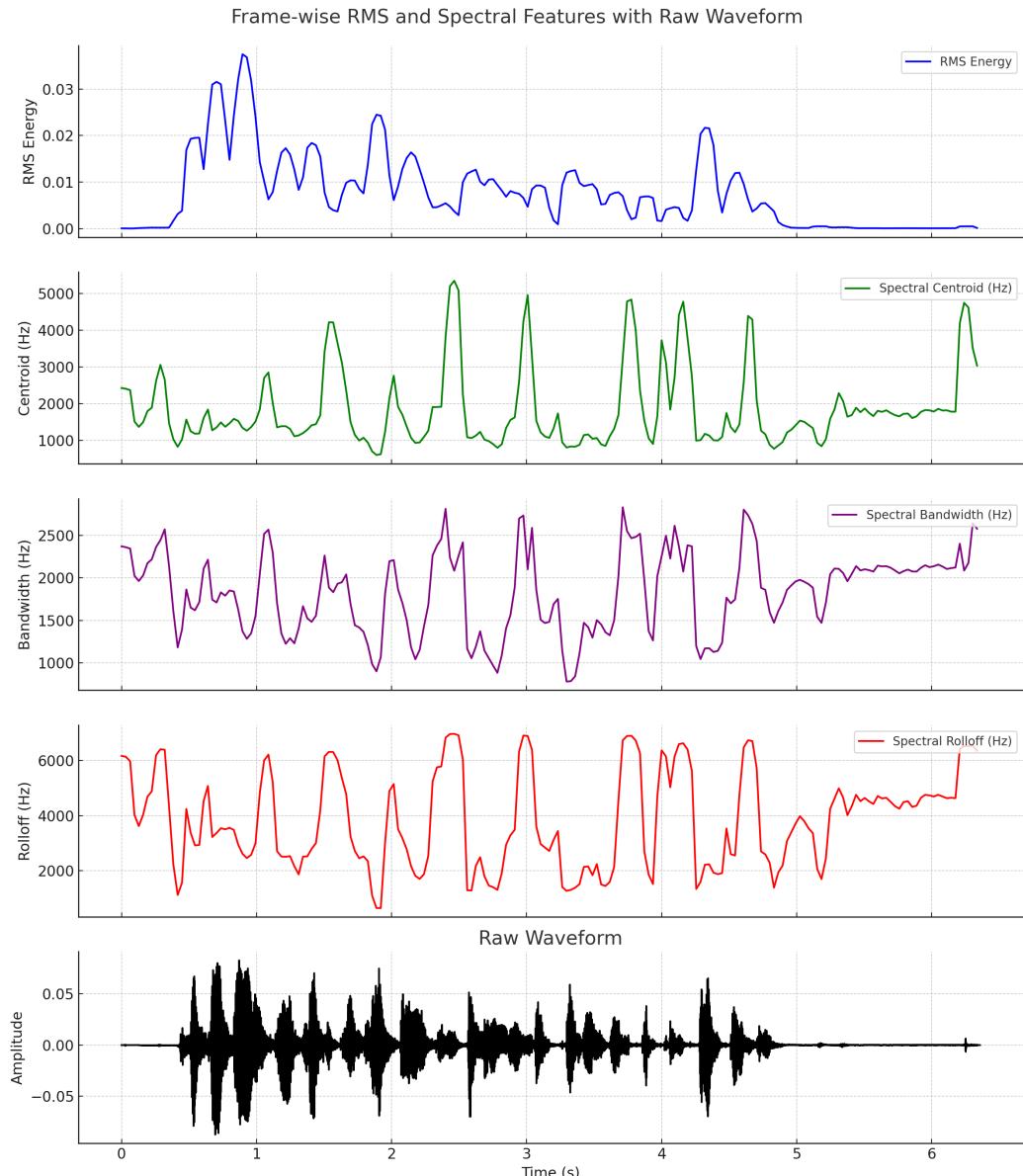


Figure 2.7: Frame-wise RMS and spectral features extracted from a speech recording.

2.3.3 Mel-Frequency Cepstral Coefficients (MFCCs)

MFCCs are one of the most important and ubiquitous features in speech processing. They provide a compact representation of the short-term power spectrum of sound, shaped according to human auditory perception [4]. The idea behind MFCCs is to mimic how human ears perceive sound: it uses a mel scale frequency axis (which is logarithmic, reflecting the ear's finer resolution at low freqs and coarser at high freqs) and a cepstral representation (which roughly decorrelates spectral features and emphasizes the spectral envelope). MFCCs have been extremely effective for tasks like speech and speaker recognition.

Computation: Generating MFCCs involves several steps in a pipeline:

- **Pre-emphasis:** (Optional) Apply a pre-emphasis filter to boost high frequencies (typically $H(z) = 1 - 0.97z^{-1}$).
- **Framing and Windowing:** Already done as discussed (e.g., 25 ms frames, Hann window).
- **FFT:** Compute the magnitude spectrum of each frame (usually via an N-point FFT, $N \sim 512$ or 1024).
- **Mel Filterbank:** Pass the spectrum through a filterbank of triangular filters spaced on the mel scale. For example, one might use 40 mel-spaced filters covering 0–8 kHz. Each filter outputs the sum of spectral energy in that mel band.
- **Logarithm:** Take the log of each filter's output (producing log-mel spectrum). Taking logs converts multiplicative spectral factors into additive values and aligns with human loudness perception.
- **Discrete Cosine Transform (DCT):** Apply DCT on the log-mel spectrum to decorrelate the filterbank coefficients. This yields cepstral coefficients.
- **Keep lower DCT coefficients:** Typically, the first 12 or 13 coefficients are retained. These represent the spectral envelope and discard fine details like pitch.

The result is a set of 13 or so numbers per frame – the MFCCs. They succinctly capture the broad spectral characteristics (formant structure) without carrying extra detail like pitch harmonics. This makes them effective for speech recognition, where the goal is to recognize content irrespective of speaker pitch or voice.

```
mfcc = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=13,
                            n_fft=512, hop_length=160, n_mels=40)
```

This computes 13 MFCCs per frame using a 40-mel-filter bank (librosa handles steps 3–7 internally). The output is a $13 \times T$ matrix. It's common to normalize MFCCs by subtracting the mean and dividing by the standard deviation.

MFCC Computation Pipeline

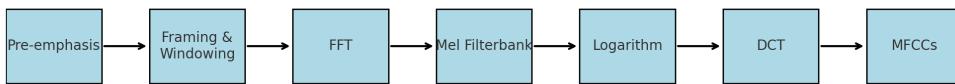


Figure 2.8: Block diagram of MFCC computation. The process includes pre-emphasis, framing and windowing, FFT to obtain the spectrum, passing through mel-scaled filterbanks, logarithmic compression, and DCT to yield cepstral coefficients.

Visualization: MFCCs can be shown as a heatmap over time (coefficients 1 through 13). Voiced vs unvoiced sounds can be distinguished via their spectral envelope patterns.

MFCCs are used in ASR, speaker recognition, emotion classification, and other tasks due to their low-dimensional, information-rich representation. While modern deep ASR models may use log-mel spectrograms, MFCCs remain popular, especially with smaller datasets or classical ML models.

Delta and Delta-Delta Coefficients

While MFCCs describe static spectral characteristics, speech is dynamic. Delta features capture how features change over time. It is standard to compute:

- **Delta (1st derivative):** Captures the velocity of feature change.
- **Delta-Delta (2nd derivative):** Captures acceleration.

These are computed by fitting a slope to the feature trajectory using a sliding window. A common formula is:

$$\Delta_t = \frac{\sum_{n=1}^N n \cdot (c_{t+n} - c_{t-n})}{2 \sum_{n=1}^N n^2}$$

Where c_t is the feature vector at time t and N is the window size.

```
mfcc = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=13)
mfcc_delta = librosa.feature.delta(mfcc, order=1)
mfcc_delta2 = librosa.feature.delta(mfcc, order=2)
```

The delta and delta-delta matrices are of shape $13 \times T$. These are often concatenated with MFCCs into a 39-dimensional vector per frame.

Use Cases: Delta features are essential in ASR and speaker recognition to model transitions and speech dynamics [5]. In emotion detection, changes in pitch or energy (prosody) can also be modeled via deltas.

Chroma Features

Chroma features capture the energy distribution across the 12 pitch classes of the musical octave (C, C#, D, ..., B). They are commonly used in music analysis, but can also be applied to voiced speech. This is because voiced speech contains harmonics based on F0. Chroma compresses harmonic energy into 12 pitch bins (regardless of octave), capturing relative pitch class usage. Even though speech is not usually musical, chroma can reflect intonation and pitch dynamics useful in tasks like speaker identification and prosody analysis.

```
chroma = librosa.feature.chroma_stft(y=y, sr=sr,
                                      n_fft=1024, hop_length=hop_length)
```

This returns a $12 \times T$ chromagram matrix.

Use Cases: While not standard in ASR, chroma is useful in speaker modeling, tonal language processing, and voice conversion tasks where pitch class preservation matters [3].

2.3.4 Log-Mel Spectrogram and Pitch Features

While not a “feature” in the scalar sense like those above, the **log-mel spectrogram** is essentially the collection of filterbank energies computed on each frame (as part of MFCC processing), but not transformed into cepstral coefficients [6]. In other words, it’s the time-frequency representation after applying the mel scale and logarithmic compression. This representation is widely used as input to modern deep learning models (e.g., CNNs, RNNs) in end-to-end speech recognition systems [7].

The log-mel spectrogram can be visualized as an image of size $(n_{mels} \times T)$, where each “pixel” represents the energy in a specific mel-frequency band at a specific time frame (in decibels).

Why log-mel? Compared to raw spectrograms:

- Mel spectrograms focus on perceptually important frequency bands.
- Log scaling compresses the dynamic range of the signal.
- Retains more detailed information than MFCCs (since no DCT or truncation is applied).

Many robust speech recognition systems feed log-mel spectrograms into neural networks, allowing the model to learn features directly from the spectrogram. This can outperform MFCCs, which are lossy compressed representations. Additionally, log-mel spectrograms reveal meaningful patterns: formants appear as darker horizontal stripes, pitch harmonics as fine vertical ridges (depending on resolution).

Calculation in Python (Librosa):

```
S = librosa.feature.melspectrogram(y=y, sr=sr, n_fft=1024,
                                    hop_length=160, n_mels=40)
S_dB = librosa.power_to_db(S, ref=np.max)
```

This yields a $40 \times T$ matrix of log-mel energies, which can be fed directly into a CNN. Figure 2.9 shows an example of such a representation, where voiced speech regions are visible as bright, structured bands.

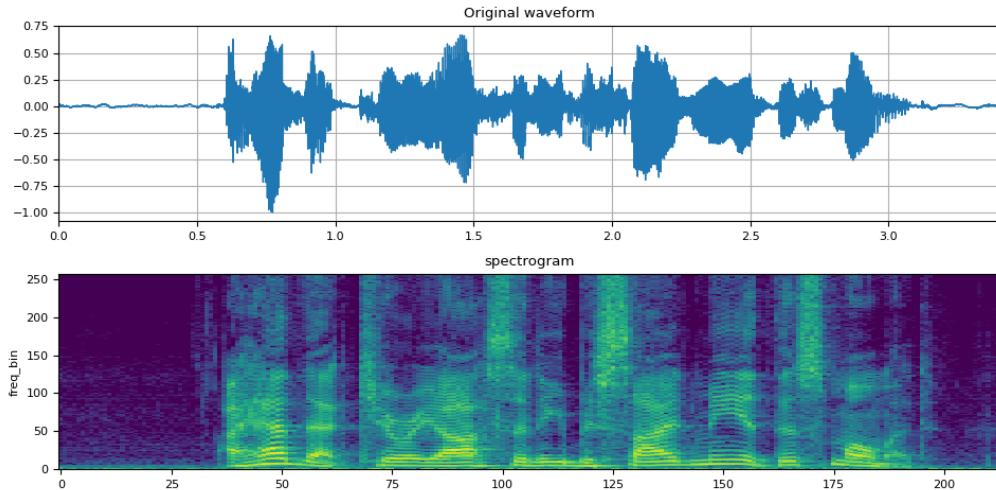


Figure 2.9: Log-mel spectrogram of a short speech segment. Voiced regions appear as bright, banded areas, while unvoiced regions are darker or more diffuse.

Use Cases: Log-mel spectrograms are now standard in:

- End-to-end automatic speech recognition (ASR)
- Wake word detection (e.g., “Hey Siri”)
- Emotion and speaker recognition using CNNs

Pitch (Fundamental Frequency, F0)

The **fundamental frequency** (F0), commonly referred to as *pitch*, is produced by the vibration rate of the vocal folds during voiced speech [8]. Pitch changes over time form a contour that is crucial for prosody (intonation, tone, and emotion).

Why is pitch useful?

- MFCCs remove pitch intentionally for speaker invariance in ASR.
- Pitch adds valuable speaker-specific and emotion-related cues.
- In tonal languages, pitch modulates meaning.

Typical pitch ranges:

- Adult male: 80–180 Hz
- Adult female: 150–300 Hz

Estimation in Librosa:

```
f0_series = librosa.yin(y, fmin=50, fmax=500, sr=sr,
                         frame_length=1024, hop_length=160)
```

This returns an F0 contour over time (with NaN or zero for unvoiced frames). Other methods like pyin or piptrack exist, but YIN is often effective and simple to use.

Use Cases:

- **Speaker recognition:** F0 range and contour shape may help differentiate individuals.
- **Emotion classification:** Angry or excited speech often shows higher and more variable pitch.
- **Prosody modeling:** Pitch contour is critical for generating natural speech in TTS systems.

Note: Pitch estimation is sensitive to noise and vocal irregularities, but when reliable, it provides complementary information to spectral features like MFCCs. Together, pitch and spectral shape represent the source-filter model of speech production.

2.3.5 Formants

Formants are the resonant frequencies of the vocal tract – essentially the peaks in the spectral envelope of voiced speech. Formants (labeled F1, F2, F3, … from low to high frequency) correspond to tongue and mouth configuration; they are what define vowels. For example, the vowel [a] might have $F1 \approx 800$ Hz, $F2 \approx 1200$ Hz; [i] might have $F1 \approx 300$ Hz, $F2 \approx 2500$ Hz, etc. Each vowel has a unique combination of the first two or three formant frequencies. Thus, formants are critical in phonetics – they uniquely identify vowels and other resonant sounds.

In terms of features, tracking formant frequencies over time provides a high-level description of the speech signal’s filter characteristics. Historically, formant frequencies (and bandwidths) were used in speech recognition and speaker recognition. They are still used in speech therapy and linguistics research to analyze articulation. In modern ML pipelines, we often rely on MFCCs (which implicitly contain formant info), but explicitly extracting formants can be beneficial for certain tasks (e.g., pronunciation scoring, or low-dimension interpretable feature sets).

Estimation: Formants can be estimated by finding peaks in the spectral envelope. One way is linear predictive coding (LPC), which gives an all-pole model of the spectrum – the poles of the LPC filter correspond to formants. Another way is simply to take a high-resolution FFT of a voiced frame and pick the prominent peaks in the spectrum (ignoring the fine harmonic structure by perhaps smoothing or using a cepstral envelope). Many speech toolkits exist for formant tracking (e.g., Praat [9]). In Python, one could use `librosa.lpc` (if available) or `numpy.polyfit` on the autocorrelation to get LPC coefficients and from them formant frequencies. For brevity, suppose we have an `estimate_formants(frame)` function that returns F1, F2, F3 for that frame (for voiced frames).

```
import numpy as np, scipy.signal
from numpy.polynomial import Polynomial

def lpc_formants(frame, sr, order=12):
    frame = frame * np.hamming(len(frame))
    autocorr = np.correlate(frame, frame, mode='full')[len(frame)-1:]
    R = autocorr[:order+1]
    A = scipy.linalg.toeplitz(R[:-1])
    b = -R[1:]
    lpc_coeff = np.linalg.solve(A, b)
    lpc_coeff = np.hstack([1, lpc_coeff])
    roots = np.roots(lpc_coeff)
    roots = [r for r in roots if np.imag(r) >= 0]
    angs = np.angle(roots)
    formant_freqs = sorted(angs * (sr/(2*np.pi)))
    return formant_freqs[:3]

# Usage:
F1, F2, F3 = lpc_formants(frame_signal, sr=16000)
```

Interpreting formants: Each formant frequency corresponds to a resonance. F1 is related to vowel height (jaw openness), F2 to vowel frontness (tongue position) – e.g., high F2 for front vowel [i], low F2 for back vowel [u]. This is why formants distinguish vowels. Consonants have formant transitions – rapid changes in formants – which are cues to the consonant identity (like the rising F2 transition for [g] as in “ga”, known as the “velar pinch” where F2 and F3 come together).

Use cases: Formant features can be used for speaker identification (formant frequencies differ slightly due to vocal tract length), emotion analysis (e.g., stressed speech might show formant shifts), and in clinical settings like pronunciation

clarity analysis or accent recognition.

Note: Estimating formants reliably requires voiced segments and can be error-prone in noisy conditions or with closely spaced formants. Still, they are an interpretable and linguistically meaningful feature.

2.3.6 Feature Scaling and Normalization

After extracting features like those above, we often apply feature scaling or normalization before feeding them into machine learning models. The goal is to ensure that all features are on comparable scales and that the model isn't unduly influenced by one feature's range or offset.

Mean and Variance Normalization: A common method is cepstral mean normalization (CMN) or cepstral mean and variance normalization (CMVN) on MFCC features:

```
X_norm = (X - X.mean(axis=0)) / X.std(axis=0)
```

Per-utterance vs Global normalization: For speech recognition, CMVN is often applied per speaker or utterance to reduce channel effects. For emotion recognition, global normalization or selective feature normalization might be preferred.

Scaling to Range: Another method is min-max scaling to [0,1] or [-1,1], commonly used in neural networks. For log-Mel spectrograms, values are often clipped to [-80, 0] dB and rescaled.

Decorrelation (Whitening): PCA whitening or HLDA was used in older systems like GMM-HMMs, but modern end-to-end models often learn such transformations internally.

Speech Audio vs Feature Normalization: Speech audio amplitude normalization is done before feature extraction (e.g., to normalize energy), whereas feature normalization ensures well-conditioned inputs.

2.4 Comparing Features and When to Use Them

We have introduced a variety of features – each captures different facets of speech. A natural question is: which features are suitable for which tasks, and what are the pros and cons? Below, we provide a comparative discussion:

MFCCs vs Spectral (Centroid/Bandwidth) vs Formants

MFCCs are effectively a smoothed spectral envelope and have proven excellent for speech recognition (ASR) because they compactly represent phonetic content while being robust to pitch differences. They are also common in speaker recognition and many speech tasks due to their effectiveness and low dimensionality. Formant features, being explicit frequencies, are highly interpretable and directly relate to articulation – they could be used in vowel classification, pronunciation scoring, or linguistic analysis. However, automatically tracking formants can be error-prone, and MFCCs often indirectly carry the same info with more stability. Pure spectral shape features like centroid, bandwidth, contrast, and rolloff give a sense of “brightness” or noise-tonality of the sound; they can complement MFCCs in tasks like speaker classification (e.g., distinguishing speech from music or environmental noise) or emotion detection (where a shift to higher spectral centroid might indicate anger or shouting). In general, MFCCs (possibly with deltas) are a solid default for recognition tasks, whereas formants might be added for interpretability or specific use-cases (like diagnosing pronunciation). Spectral contrast can be useful in speaker identification as well – different voices or recording conditions might have different spectral fine structures; contrast can capture presence of strong harmonics (in voiced sounds) which might vary with vocal quality.

Time-Domain vs Frequency-Domain vs Prosodic Features

Time-domain features like ZCR and energy are very straightforward and computationally cheap. Voice activity detection (VAD) often relies on energy and ZCR thresholds – low energy and low ZCR indicate silence, low energy and high ZCR might indicate unvoiced noise, etc.. However, these features alone are not sufficient for complex discrimination (they don't tell apart phonemes, for instance). Frequency-domain features (MFCCs, spectral features) are richer and are the backbone of ASR and speaker recognition. Prosodic features like pitch and energy contour come into play for emotion recognition and speaker characterization. For example, a speaker's pitch range and speaking energy can distinguish them or indicate emotional state. In emotion detection, typically a large set of features including pitch statistics (mean, std, min, max), energy stats, speaking rate, etc., are used in addition to spectral features. So, in summary: for lexical content (what is spoken), spectral features (MFCCs) are key; for paralinguistic aspects (who is speaking, how they're speaking), prosodic features (pitch, energy, speaking rate) add a lot of value, often combined with spectral ones.

Log-Mel Spectrogram vs MFCC

With the rise of deep learning, many end-to-end models prefer using the rawer log-mel spectrogram input. Speech recognition systems nowadays often take 40-channel log-mel features (sometimes with delta and delta-delta appended, sometimes just as a 2D input to a CNN). The advantage is the model can learn its own optimal filterbank or feature combination. MFCCs, by contrast, have the advantage of lower dimensionality (13 vs 40 or more dimensions) and have traditionally been decorrelated (which was beneficial for Gaussian models). For a classical ML algorithm (GMMs, older neural nets), $\text{MFCC} + \Delta + \Delta\Delta$ (39 dims) is much more tractable than a full spectrogram slice (which could be 257 FFT bins). But for CNNs with GPUs, feeding a 2D spectrogram is feasible and may capture fine details lost in MFCC. If one's using a simpler classifier or has less data, MFCCs might outperform because they impose prior knowledge that helps the model. If using a powerful deep model and lots of data, log-mel (or even raw waveform in some research) can be used – though log-mel is still overwhelmingly common as an input representation.

Feature Robustness

Some features are more robust to noise than others. For instance, relative features like spectral contrast or shape might remain informative even under certain noise (as long as noise is broadband, it raises valleys and lowers contrast, which itself is info). MFCCs can be made robust with normalization and modifications (e.g., cepstral filtering, mean normalization). Energy is very sensitive to noise – a noisy background raises energy floor significantly, potentially confusing a VAD unless a noise reduction was done. Pitch can also be distorted by noise (pitch trackers might lock onto noise harmonics or simply fail). There are noise-robust pitch algorithms (using autocorrelation in specific bands). In extremely noisy conditions, features like RPLP (Relative Perceptual Linear Predictive) or advanced cepstral coefficients have been proposed, but those are beyond our scope. As a rule of thumb, if expecting noise, ensure to include noise-robust processing (filtering, VAD) and possibly use features that normalize out noise (e.g., MFCCs with CMVN, or adding Rasta filtering which emphasizes spectral dynamics).

Dimensionality and Computational Cost

Simpler features (ZCR, energy) are trivial to compute and yield one number per frame. MFCCs yield 13 per frame – also quite efficient especially with FFT optimizations. Chroma (12 per frame) and spectral features (often 1 per frame, or 6 for contrast bands) are also manageable. A log-mel spectrogram for a 1-second speech audio (with 100 fps and 40 mel bands) is $40 \times 100 = 4000$ numbers – if fed to a model directly, that's fine for a CNN but too many for a traditional classifier unless reduced. So often for classical pipelines, one would aggregate features over time to reduce data rate (e.g., take statistics over an utterance). For deep learning, feeding sequences is fine (RNN can take frame sequence, CNN can take spectrogram slice). If one has to choose a small set of features for, say, a classical machine learning classifier (SVM, etc.), one might choose a handful: e.g., mean and variance of MFCC 1-12, mean pitch, pitch variance, mean energy, etc. If using deep learning, one might feed the whole sequence of MFCCs or mel-spectrogram.

Application Mapping

Automatic Speech Recognition (ASR): Traditionally uses $\text{MFCC} + \Delta + \Delta\Delta$. Modern ASR often uses log-mel spectrogram patches into CNN or sequences into RNN/Transformer. Spectral features like centroid are not explicitly used in ASR (since MFCCs suffice). ZCR, etc., not used. Pitch sometimes used in tone languages to augment phonetic features (but usually handled by model).

Speaker Identification/Verification: Often uses MFCCs too (for input to i-vector or x-vector deep models). Additionally, pitch and formant-related features can help because they carry speaker characteristics (vocal tract length influences formants; habitual pitch differs per speaker). High-level prosodic features (like intonation patterns, speaking rate) can also be included. But MFCCs alone (or derived embeddings) have shown excellent results in speaker recognition, especially with sufficient data.

Emotion Recognition: This typically uses a broader feature set. For instance, the eGeMAPS feature set (a standard in emotion research) includes MFCCs, pitch, formants, jitter/shimmer (voice quality measures), energy, spectral flux, etc., aggregated over utterances. Pitch and energy dynamics are very important here, as are spectral shape cues that differentiate, say, a sharp, high-frequency “angry” tone vs a low, mumbling “sad” tone. ZCR might indirectly reflect “breathiness” or amount of unvoiced content (could be higher in angry speech with lots of frication). Spectral rolloff could shift with emotion (more high-frequency energy in anger, perhaps).

Gender or Age Classification: Pitch is a primary feature for gender (male voices generally lower F0) and also age (children have very high F0). Formants can also indicate age (children have shorter vocal tracts, so formants are at higher

frequencies). MFCCs can capture some of this, but explicitly including F0 and maybe formant positions can improve such classifiers.

Keyword Spotting (Wake-word Detection): Often uses similar features to ASR (log-mel or MFCC) because you need to discriminate specific words.

Music/Song vs Speech Discrimination: Spectral features like centroid and contrast can be quite useful to tell music from speech. Music might have higher spectral variability or more steady harmonic content (depending on genre), whereas speech has a certain spectral envelope pattern and silences. Chroma is obviously useful for music (tonal structure) but in speech it would mostly look random (since speech is not aligned to musical semitones). So a system distinguishing singing vs speaking could look at chroma clarity – in singing, chroma might show stable musical notes, while in normal speech it's diffuse.

Pros and Cons Summary

- **MFCC:** +Well-proven, compact, approximates auditory perception. +Many existing models expect them. –They discard absolute timing of spectral events beyond frame level (so need deltas for dynamics). –They are affected by additive noise (log can help but still).
- **Delta Features:** +Provide temporal context essential for sequence modeling in static models. –Add dimensionality, and not interpretable on their own (just differences).
- **Energy:** +Simple cue for speech activity and emphasis. –Highly variant with distance/microphone, requires normalization.
- **ZCR:** +Easy indicator of voiced/unvoiced. –Not robust in presence of noise or DC offset; not uniquely identifying beyond that (many different sounds can have similar ZCR).
- **Spectral Centroid/Bandwidth:** +Capture timbral quality (brightness). –Don't directly relate to specific phonemes, but rather overall spectrum balance.
- **Spectral Contrast:** +Captures harmonicity vs noisiness. –Can be less intuitive to interpret; and if spectrum is flat due to noise, it just tells “no strong peaks” which you might already infer from low energy of harmonics.
- **Spectral Rolloff:** +Good indicator of high-frequency content proportion. –Might be redundant with centroid to some extent (both move upward with more high-frequency energy).
- **Chroma:** +Great for tonal analysis (music, or prosody to some extent). –In normal speech, pitch is not quantized to semitones, so chroma might be “blurry.” If a speaker is speaking at 130 Hz vs 150 Hz, that's a different chroma pattern, but not meaningful linguistically. So chroma is more relevant if your speech has a musical or tonal component (e.g., analyzing sung vowels, or perhaps in a tonal language to identify tone patterns roughly).
- **Pitch (F0):** +Essential for intonation, tone, speaker traits. –Difficult to estimate in noisy or very high-pitched or polyphonic scenarios. For unvoiced sounds, undefined. If used as a continuous feature, need to handle “no value” cases (e.g., set to 0 or carry last value or use interpolation).
- **Formants:** +Direct correlation to phonetics, interpretable (you can literally guess the vowel by F1/F2 positions). –Harder to compute reliably, especially for higher formants or in noisy data; only defined in voiced regions; also overlapping speakers or background can throw it off. Additionally, needing to decide number of formants to track and pairing them across frames (formant tracking) can be complex. But as static features, average formant frequencies can tell speaker vocal tract characteristics.

Feature Fusion and Final Notes

In practice, the choice of features comes down to the task requirements and the model. If using deep learning and large data, one might lean toward giving the model more raw representations (log-mel, maybe even raw waveform to a learnable filterbank). If using classical approaches or limited data, carefully designed features like MFCC plus a selection of prosodic features can give better performance with less data.

It's also common to combine feature sets: e.g., MFCCs + pitch + energy has been used in some voice activity detectors to improve detection of low-energy voiced sounds (pitch ensures you catch voiced even if energy is low). Another example: In speaker recognition, combining MFCC embeddings with prosodic feature models yields better results than either alone, since they capture complementary information (short-term spectral vs long-term intonation patterns).

To conclude: No single feature is universally “best” – each provides a lens on the data. A robust speech system often uses a feature fusion: MFCCs (for spectral envelope), plus deltas (for dynamics), plus maybe pitch/formant (for source characteristics), plus energy (for emphasis). Modern end-to-end systems attempt to learn these from raw data,

but understanding these features and their roles helps in system design and troubleshooting. In later chapters, when we apply machine learning models to speech, we'll see how these features feed into models and how to choose a feature set appropriate for the task at hand. By extracting and possibly combining the right features, we ensure our models have the relevant information to learn from, which is a critical step toward successful speech machine learning.

Bibliography

- [1] J. Ramirez, J. C. Segura, C. Benitez, A. de la Torre, and A. Rubio, “Efficient voice activity detection algorithms using long-term speech information,” *Speech Communication*, vol. 42, no. 3, pp. 271–287, 2004.
- [2] J. Ramírez, J. M. Gorriz, and J. C. Segura, “Voice activity detection. fundamentals and speech recognition system robustness,” *Robust speech recognition and understanding*, pp. 1–22, 2007.
- [3] G. Tzanetakis and P. Cook, “Musical genre classification of audio signals,” in *IEEE Transactions on speech and audio processing*, vol. 10, no. 5. IEEE, 2002, pp. 293–302.
- [4] S. Davis and P. Mermelstein, “Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences,” *IEEE transactions on acoustics, speech, and signal processing*, vol. 28, no. 4, pp. 357–366, 1980.
- [5] S. Furui, “Speaker-independent isolated word recognition using dynamic features of speech spectrum,” in *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 34, no. 1. IEEE, 1986, pp. 52–59.
- [6] V. J. Stevens, S.S. and E. Newman, “A scale for the measurement of the psychological magnitude pitch,” *The Journal of the Acoustical Society of America*, vol. 8, no. 3, pp. 185–190, 1937.
- [7] T. N. Sainath, R. J. Weiss, A. Senior, K. Wilson, and O. Vinyals, “Learning the speech front-end with raw waveform cldnns,” in *Proc. Interspeech*, 2015, pp. 1–5.
- [8] A. De Cheveigné and H. Kawahara, “Yin, a fundamental frequency estimator for speech and music,” *The Journal of the Acoustical Society of America*, vol. 111, no. 4, pp. 1917–1930, 2002.
- [9] P. Boersma and D. Weenink, *Praat: Doing phonetics by computer*, 2024, version 6.4.05. [Online]. Available: <http://www.fon.hum.uva.nl/praat/>

Chapter 3: Machine Learning on Audio Signal: A Summary

By definition, Machine Learning (ML) is the discipline that frames data-driven prediction and decision-making as an optimization problem. Formally, let \mathcal{D} be an unknown distribution over input–output pairs $(\mathbf{x}, y) \in \mathcal{X} \times \mathcal{Y}$. A learner chooses a hypothesis $f_\theta: \mathcal{X} \rightarrow \mathcal{Y}$ from a parameterised family $\{f_\theta\}_{\theta \in \Theta}$ and seeks parameters

$$\boldsymbol{\theta}^* = \arg \min_{\boldsymbol{\theta} \in \Theta} R(\boldsymbol{\theta}), \quad R(\boldsymbol{\theta}) = \mathbb{E}_{(\mathbf{x}, y) \sim \mathcal{D}} \mathcal{L}(f_\theta(\mathbf{x}), y),$$

where \mathcal{L} is a task-specific loss (e.g., binary cross-entropy for wake-word detection). Because \mathcal{D} is inaccessible, the *expected risk* R is approximated by the *empirical risk* $\hat{R}(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^N \mathcal{L}(f_\theta(\mathbf{x}_i), y_i)$ over a finite sample $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$.

Three ingredients, therefore, characterize any ML project:

1. **Task T** – the mapping to be learned, such as classifying 1s audio segments as *wake-word* or *background*.
2. **Experience E** – a corpus of labeled examples; In speech, a modern keyword-spotting system might use 50k clips recorded in accents, microphones, and noise conditions [1].
3. **Performance measure P** – an objective linked to deployment constraints, e.g., the false-accept rate at 95% recall [2].

Learning works best when the model’s **inductive bias** (its built-in assumptions about what patterns to expect) matches the structure of the data. For example, in speech tasks, convolutional and attention-based layers are effective because they take advantage of how speech patterns are localized in time and frequency. Techniques like time-stretching or SpecAugment help the model generalize better by creating slight variations of the input. More recently, self-supervised methods such as wav2vec 2.0 have shown that models can be pre-trained on large amounts of unlabeled speech by predicting masked parts of the signal and then fine-tuned with much smaller labeled datasets to achieve strong performance [3, 4].

3.1 Types of ML Systems

ML approaches are typically described along three independent dimensions: *supervision, learning schedule, and decision strategy*, each of which has direct counterparts in speech-audio research and practice.

Supervision. A system is *supervised* when every training utterance carries a ground-truth label; for instance, a wake-word detector trained on thousands of clips marked “Hey Alexa” or “background.” In *unsupervised* settings, the model sees only raw waveforms and must uncover latent structure on its own, as when spectral clustering discovers speaker groupings in hours of unlabeled meeting audio. *Semi-supervised* learning occupies the middle ground, fine-tuning a small, labeled keyword corpus with a much larger pool of pseudo-labeled voice-assistant logs. Finally, *reinforcement-learning* agents improve through trial and reward, for example, a spoken-dialogue system that earns positive feedback when users swiftly complete tasks and maintain politeness.

Learning schedule. In *batch* learning, the model ingests a fixed dataset all at once. By contrast, an *online* learner updates continuously as new data arrive, such as an adaptive beam-forming algorithm that refines its microphone weights frame by frame while a conference call is in progress.

Decision strategy. *Instance-based* methods classify by direct comparison: a k-Nearest-Neighbour (k-NN) speaker-ID system stores enrollment i-vectors and labels a test i-vector by whichever cohort lies closest in embedding space. *Model-based* approaches first fit a compact set of parameters, then discard the raw examples; a convolutional neural network that predicts emotions from log-mel spectrograms is a classic example.

3.1.1 Supervised and Unsupervised Learning

ML systems are often introduced in terms of the *degree of supervision* available at training time. Four principal regimes are recognized: supervised, unsupervised, semi-supervised, and reinforcement learning, but the boundary between them is fluid in practical speech-audio work.

Supervised Learning

In *supervised Learning*, every training utterance is paired with an explicit label, allowing the model to learn a direct mapping from acoustic representations to task-specific targets [5]. In speech technology, the target can be a discrete class (e.g., *wake-word* vs. *background*), a sequence of phonemes, or a continuous value such as pitch. The workflow typically comprises five stages:

- Data curation.** Raw recordings are segmented into fixed-length clips; mislabelled or low-SNR segments are pruned to prevent noisy supervision. Class balance is critical: keyword corpora often down-sample the vast background class to avoid overwhelming the minority wake-word examples.
- Feature extraction.** Log-Mel spectrograms, MFCCs, or learnable filterbanks are computed with 20–40 ms analysis windows. Frame stacking (e.g., 10 frames = 100 ms context) supplies temporal cues without recurrent layers.
- Model training.** Convolutional or transformer encoders ingest the feature tensor and output logits. Cross-entropy loss is minimised with Adam or RMSProp; early stopping monitors validation error to avoid over-fitting. Data augmentation such as time-stretching, SpecAugment, random room impulse responses exposes the model to realistic acoustic variability.
- Post-processing.** The per-window posterior stream is smoothed by a finite-state machine or hysteresis filter to suppress short false alarms. Thresholds are tuned on a development set to satisfy device or privacy budgets (e.g., <1 false alarm/hour).
- Evaluation.** Metrics go beyond raw accuracy: false-accept and false-reject rates are reported at several operating points; ROC and DET curves visualize trade-offs [2]. Deployment trials on embedded hardware verify latency and power constraints.

Classification tasks. Keyword spotting, intent classification, and speaker verification all rely on supervised classification models. Architectures range from shallow CNNs deployed on microcontrollers to large transformers distilled for on-device execution [6]. Class imbalance is mitigated by focal loss or by over-sampling minority examples during mini-batch construction.

Regression tasks. Many speech problems are naturally continuous. Estimating the fundamental frequency F_0 , predicting loudness in LUFS, or mapping articulatory sensor readings to tongue trajectories each outputs a real-valued signal. Although regression is trained with mean-squared or Huber loss, decision thresholds can convert the output into categories (e.g., “high” vs. “low” pitch) when required by downstream logic.

From logits to confidence. Even a simple logistic regression model can serve as a calibrated probability estimator: its sigmoid output approximates the true posterior under the assumption of label-balanced data. Calibration degrades when priors shift; temperature scaling or Platt scaling restores well-behaved confidence scores, a necessity for cascaded pipelines where later modules rely on upstream probabilities.

Take-aways. Supervised learning delivers state-of-the-art performance in speech when abundant labelled data, representative of deployment conditions, are available. Success hinges on robust feature generation, aggressive yet task-aligned augmentation, careful handling of class imbalance, and objective metrics that reflect real-world costs.

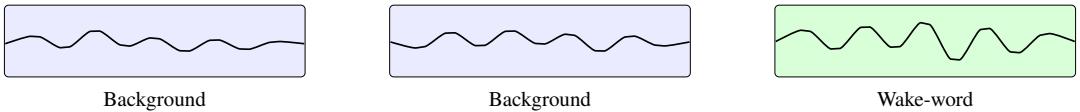


Figure 3.1: Typical supervised setting: each audio clip is tagged with a ground-truth label (e.g., wake-word vs. background).

Unsupervised Learning

In *unsupervised learning*, the corpus arrives without labels; the algorithm must uncover structure on its own [7]. Speech practitioners rely on three broad tool families:

Clustering. Algorithms such as k-means, agglomerative hierarchical clustering, spectral clustering, and HDBSCAN group utterance-level embeddings according to cosine or PLDA distance. A speaker-diarisation pipeline, for example, extracts x-vectors from overlapping 1.5 s windows, applies dimensionality reduction, and then clusters the points to infer “who spoke when.” Agglomerative clustering with a Bayesian information criterion (BIC) stop rule often yields < 10 % diarisation error rate on broadcast news without ever seeing a speaker label. The same machinery performs *anomaly detection*: embeddings that fall far from any cluster centre may indicate spoofed audio or corrupted recordings worthy of human review.

Representation learning. Autoencoders, variational autoencoders, and self-supervised objectives such as Contrastive Predictive Coding (CPC) learn feature spaces where downstream tasks require fewer labels. Wav2vec 2.0, for instance, trains a transformer to predict masked time steps in raw waveforms, producing 768-dimensional embeddings that, once fine-tuned, cut word-error rate by up to 50 to purely supervised baselines. These pre-trained features feed directly into clustering, anomaly detection, or low-resource ASR.

Dimensionality reduction and visual analytics. High-dimension per-frame MFCCs or x-vectors are projected into two or three dimensions with PCA, t-SNE [8], UMAP, or kernel PCA so that clusters of speakers, emotions, or channel types become visually separable. Engineers routinely inspect such plots to diagnose domain mismatch e.g., telephone speech embedding clouds that sit far from smart-speaker embeddings indicate a coverage gap in the training data.

Association discovery. Audio event mining looks for frequent co-occurrence, such as a particular HVAC hum that correlates with female speakers in a call-centre dataset. Apriori or FP-growth algorithms reveal these latent relations and motivate targeted data augmentation or microphone filtering.

Evaluation without labels. Because ground truth is absent, intrinsic metrics like silhouette score, Davies–Bouldin index, and cluster-size entropy guide hyperparameter search. For diarisation, a small labelled subset provides diarisation error rate (DER); improvements on this slice are assumed to transfer to the unlabelled bulk.

While supervised systems generally yield higher ultimate accuracy, modern speech pipelines rarely rely on supervision alone. A common recipe is *self-supervised pre-training* on thousands of hours of unlabelled audio, followed by supervised fine-tuning on a modest, high-quality keyword corpus. The pre-training phase supplies powerful acoustic embeddings, whereas the fine-tuning phase specialises the model to task-specific decision boundaries. Leveraging the complementary strengths of both regimes is now considered best practice for data-efficient, production-grade speech-audio systems.

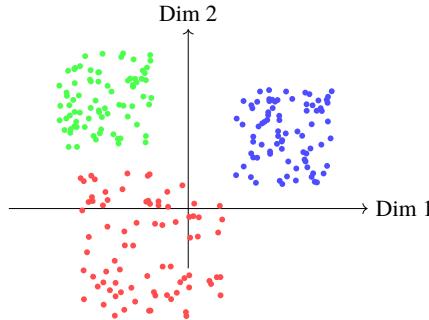


Figure 3.2: Unsupervised t-SNE projection of speech embeddings revealing three speaker clusters without any labels.

Semi-supervised Learning

In *semi-supervised learning*, a small corpus of labeled speech utterances is combined with a much larger pool of unlabeled audio to improve model generalization. A common workflow begins by training an initial model on the few available transcripts (e.g., 1 000 labeled sentences), then using that model to assign *pseudo-labels* to hundreds of thousands of untranscribed recordings. The newly labeled data are merged back into training in iterative cycles, progressively refining the acoustic model’s accuracy without requiring exhaustive manual annotation.

This approach is widely adopted in automatic speech recognition (ASR): for example, a neural-network acoustic model may be trained on a limited set of read-speech recordings, then used to decode a large web-scraped audio archive. High-confidence hypotheses are treated as ground truth in a second training stage, yielding significant gains in word-error rate with minimal human effort. Similar pipelines appear in speaker-identification research, where a handful of labeled speaker segments bootstrap the clustering and labeling of thousands of unlabeled meeting recordings.

Many advanced semi-supervised methods augment self-training with consistency regularisation or entropy minimisation to suppress noisy pseudo-labels. For instance, the “mean teacher” framework enforces that a student ASR model’s predictions on perturbed (e.g. noise-added) audio match those of a slowly updated teacher model, further boosting robustness in low-resource speech domains.

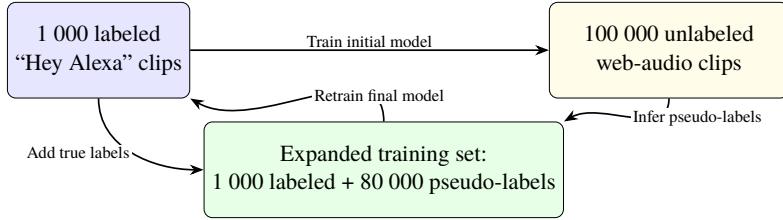


Figure 3.3: Practical semi-supervised example for wake-word detection.

Reinforcement Learning

In *reinforcement learning* (RL), an agent interacts with an environment by taking actions and receiving scalar rewards, with the goal of learning a policy that maximizes cumulative reward over time [9]. In the speech-audio domain, a common RL application is *adaptive speech enhancement*, where the agent adjusts filter or beamformer parameters to improve downstream Automatic Speech Recognition (ASR) accuracy in varying noise conditions.

At each time step, the agent observes acoustic features (e.g. Mel-spectrogram frames) and selects an action such as modifying gain coefficients of a spectral filter. The environment applies these parameters to the incoming noisy audio, feeds the enhanced signal into an ASR engine, and returns two things: an updated observation of features plus a reward based on the ASR word-error rate (WER). Over many interactions, the agent learns which parameter adjustments yield the highest ASR performance.

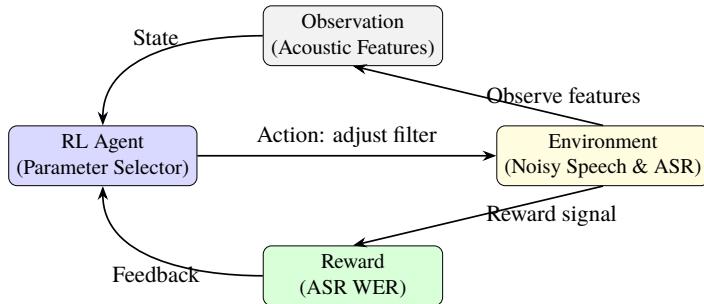


Figure 3.4: Reinforcement-learning loop for adaptive speech enhancement: the agent selects filter parameters based on feature observations and ASR-based reward feedback.

Ensemble Methods. Ensemble methods are machine learning techniques that combine the predictions of multiple base models (often called “weak learners”) to produce a final prediction that is typically more accurate and robust than any single model. The central idea is that while individual models may make errors, combining many models in a principled way allows those errors to cancel out, leading to improved generalization. Ensembles exploit the “wisdom of the crowd” principle in statistics: multiple diverse opinions, when aggregated, lead to better decisions than a single opinion.

There are two major families of ensemble methods:

1. **Bagging (Bootstrap Aggregating):** Multiple models are trained independently on random bootstrap samples of the training data. Their predictions are then averaged (for regression) or combined by majority vote (for classification). Random Forests are the most famous example of bagging applied to decision trees. Bagging reduces variance, making the model more stable and less prone to overfitting.
2. **Boosting:** Models are trained sequentially, with each new model focusing on the mistakes of the previous ones. Boosting algorithms, such as AdaBoost, Gradient Boosting, and XGBoost, combine many weak learners into a strong learner by assigning higher weights to misclassified examples. Boosting reduces both bias and variance, often achieving state-of-the-art performance in structured data tasks.

Other ensemble approaches include **stacking**, where different models are trained and their predictions are fed into a meta-model that learns the best way to combine them.

3.1.2 Model Evaluation Metrics

To assess the performance of machine learning models, several statistical metrics are used. Each metric provides a different perspective on how well the model is performing, and together they offer a more comprehensive evaluation of classification accuracy, error distribution, and statistical significance.

Accuracy. Accuracy is the simplest and most intuitive evaluation metric. It measures the proportion of correctly classified samples among the total number of samples:

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}}.$$

While accuracy is useful for balanced datasets, it can be misleading in imbalanced scenarios (e.g., if 90% of samples belong to one class, a model that always predicts that class achieves 90% accuracy without truly learning).

F1 Score. The F1 score combines *precision* and *recall* into a single value, making it especially effective for imbalanced datasets. Precision measures how many predicted positives are truly positive, while recall measures how many actual positives were identified. They are defined as:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}, \quad \text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}.$$

The F1 score is the harmonic mean of precision and recall:

$$F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}.$$

This score balances false positives and false negatives, making it particularly suitable for speech or speaker recognition tasks with uneven class distributions.

Confusion Matrix. A confusion matrix provides a detailed breakdown of classification results across multiple classes. For a two-class (binary) problem, it contains four entries: True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN). For multi-class problems, it extends into a square matrix where each row represents actual classes and each column represents predicted classes. This allows direct visualization of which classes are confused with others, offering insights into error patterns.

ANOVA (Analysis of Variance). ANOVA is a statistical test that evaluates whether the means of different groups (classes) are significantly different from one another. In feature evaluation, ANOVA compares the variance between classes to the variance within each class:

$$F = \frac{\text{Variance Between Groups}}{\text{Variance Within Groups}}.$$

A higher F -value indicates that the feature has strong discriminative power between classes. In speech processing, ANOVA can reveal whether certain features (e.g., pitch or MFCC coefficients) differ significantly across speakers or emotions.

Chi-Squared Test. The Chi-Squared test evaluates whether observed classification outcomes deviate significantly from expected outcomes under the null hypothesis (random or uniform distribution). It is defined as:

$$\chi^2 = \sum_{i=1}^k \frac{(O_i - E_i)^2}{E_i},$$

where O_i and E_i are the observed and expected frequencies for category i . In feature selection, the Chi-Squared test is often applied to discrete features to measure their association with the target class. A higher χ^2 value indicates that the feature is more relevant for classification.

3.1.3 Batch and Online Learning

ML systems for speech can be differentiated by how they consume new data. In batch learning, an acoustic or enhancement model is trained once on the entire available corpus say, 960 hours of LibriSpeech typically on a GPU cluster. When additional recordings arrive (for example, freshly collected broadcast audio), the model must be retrained from scratch on the combined old + new data, a process that can take hours and consumes substantial compute and storage.

By contrast, online learning updates the model continuously as each new utterance arrives. For instance, a streaming denoising filter in a hearing aid may adjust its spectral-subtraction coefficients frame-by-frame to track evolving background noise, without revisiting earlier audio. This allows rapid adaptation, which is crucial when moving from a quiet office to a noisy street, while keeping memory and latency low.

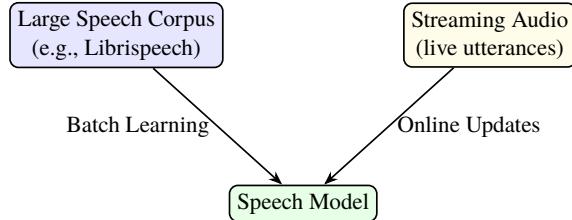


Figure 3.5: Batch vs. online learning for speech-audio models: full-corpus retraining vs. incremental updates on live audio.

3.1.4 Instance-Based and Model-Based Learning

ML systems for speech/audio differ not only in their training paradigm but also in how they generalize at inference time. In **instance-based learning**, the system retains a set of training examples and makes predictions by finding the nearest neighbors under some similarity metric. In **model-based learning**, the system fits a parametric model (e.g., logistic regression, SVM, neural network) and retains only its learned parameters.

Instance-Based Learning

Instance-based methods such as k-Nearest Neighbors (k-NN) are popular for tasks like speaker identification and keyword spotting when new examples must be accommodated on the fly. Given a test feature vector $\mathbf{x} \in \mathbb{R}^D$ (e.g., mean-MFCCs of an utterance), the system computes distances to each stored example $\{\mathbf{x}_i\}$:

$$d_{\text{cos}}(\mathbf{x}, \mathbf{x}_i) = 1 - \frac{\mathbf{x}^\top \mathbf{x}_i}{\|\mathbf{x}\| \|\mathbf{x}_i\|}, \quad d_E(\mathbf{x}, \mathbf{x}_i) = \sqrt{\sum_{j=1}^D (x_j - x_{i,j})^2}.$$

The label is then chosen by majority vote among the k closest neighbors or by weighting the vote of each neighbor inversely to its distance. Instance-based classification adapts instantly when new labeled data arrive (no retraining needed) but at the cost of $O(ND)$ query time and $O(ND)$ memory to store N examples.

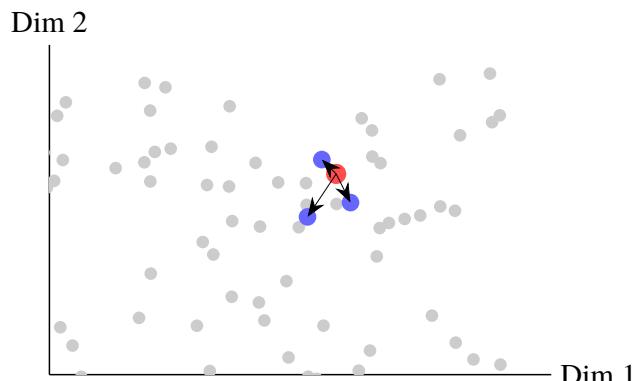


Figure 3.6: Instance-based speaker identification: 200 enrollment embeddings (gray), three nearest neighbors (blue), and a test embedding (red), with small markers and axis labels.

```

from sklearn.neighbors import KNeighborsClassifier
import numpy as np, librosa

# Extract 13-dim MFCC mean as feature
def featurize(path):
    y, sr = librosa.load(path, sr=16000)
    mfcc = librosa.feature.mfcc(y, sr, n_mfcc=13)
    return mfcc.mean(axis=1)

X_train = np.array([featurize(p) for p in train_paths])
y_train = np.array(train_labels)
X_test = np.array([featurize(p) for p in test_paths])

knn = KNeighborsClassifier(n_neighbors=5,
                           metric='cosine',
                           weights='distance')
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)

```

Model-Based Learning

Model-based approaches distil the entire training set into a compact parameter vector θ . During inference, the original examples are no longer required; all information is encapsulated in the learned parameters. Classic speech-audio models include Gaussian Mixture Models (GMMs) for acoustic likelihoods, Support Vector Machines (SVMs) operating on PLP or i-vector features, and in contemporary systems deep Convolutional Neural Networks (CNNs) that treat log-Mel spectrograms as two-dimensional ‘‘images.’’

Logistic regression for speech activity. For binary speech-activity detection, a linear classifier suffices: the probability that the frame-level feature vector $\mathbf{x} \in \mathbb{R}^D$ belongs to the *speech* class, which is obtained by passing the affine projection $\theta^\top \mathbf{x}$ through a sigmoid,

$$P(y = 1 | \mathbf{x}) = \sigma(\theta^\top \mathbf{x}).$$

The parameters θ are chosen to minimise the *regularised cross-entropy* objective in (3.1), where the $\lambda \|\theta\|_2^2$ penalty (weight decay) prevents overfitting by shrinking large weights and effectively encodes a Gaussian prior on θ . Inference reduces to a single dot product ($O(D)$) followed by a sigmoid trivial on embedded hardware and highly amenable to vectorization.

$$\mathcal{L}(\theta) = - \sum_{i=1}^N [y_i \log \sigma(\theta^\top \mathbf{x}_i) + (1 - y_i) \log(1 - \sigma(\theta^\top \mathbf{x}_i))] + \lambda \|\theta\|_2^2, \quad (3.1)$$

SVMs for MFCC embeddings. When the decision boundary is nonlinear (e.g., differentiating emotional vs. neutral speech), kernel methods such as the Radial Basis Function (RBF) SVM map inputs into a high-dimensional feature space where a linear separator exists. Storage cost is dominated by the ‘support vectors’, usually a few percent of the training set, so the memory footprint remains small relative to instance-based schemes.

CNN keyword-spotter. Figure 3.7 shows a deep CNN ingesting a 40-band log-Mel spectrogram. Convolutional layers capture local time–frequency patterns (formant sweeps, fricative bursts), while pooling layers inject local invariance to speaking rate and slight frequency shifts. Fully connected layers aggregate the high-level feature maps, and a softmax outputs posterior probabilities over the keyword set. Because the model is *fully convolutional* in time, it slides over an audio stream with constant latency, which is crucial for always-on wake-word engines that must respond within hundreds of milliseconds.

Practical implementation (Python). Listing 3.1.4 demonstrates two off-the-shelf scikit-learn models that embody the theory above. The logistic-regression classifier uses L_2 regularisation (`penalty='l2'`) and is trained until convergence (`max_iter=200`). The SVM employs an RBF kernel, tuned by the cost parameter $C = 10$ and kernel width $\gamma = 0.01$. Setting `probability=True` instructs scikit-learn to calibrate posterior probabilities via Platt scaling, enabling threshold adjustment for arbitrary false-alarm targets.

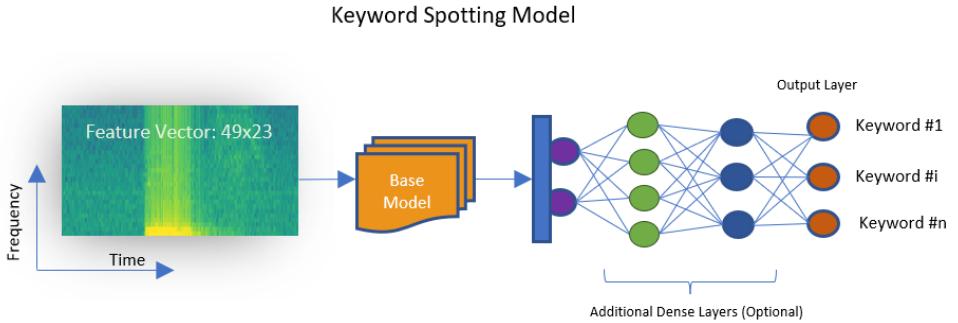


Figure 3.7: Model-based CNN for keyword spotting: spectrogram input → convolution+pool layers → softmax output.

```

from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC

# Logistic Regression for speech activity
lr = LogisticRegression(C=1.0, penalty='l2', solver='lbfgs', max_iter=200)
lr.fit(X_train, y_train)
y_pred_lr = lr.predict(X_test)

# SVM with RBF kernel on MFCCs
svm = SVC(kernel='rbf', C=10, gamma=0.01, probability=True)
svm.fit(X_train, y_train)
y_pred_svm = svm.predict(X_test)

```

Memory and latency considerations. A trained logistic regression model stores only $D + 1$ floating floats (weights plus bias), whereas CNN in Fig. 3.7 may require hundreds of kilobytes, still modest compared to instance-based databases that scale linearly with the number of enrollment utterances. At runtime, linear models execute in microseconds on a Cortex-M4, and optimised small-footprint CNNs can reach < 20 ms per inference with int8 quantisation, meeting the latency budgets of smart-speaker chips.

Hybrid and Prototype Models

To trade off memory and performance, prototype-based methods cluster N instances into $M \ll N$ centroids $\{\mathbf{c}_j\}$, then apply k-NN on $\{\mathbf{c}_j\}$. In speaker verification, this appears as i-vector + PLDA, where each speaker is represented by a mean vector and covariance, dramatically reducing storage while preserving accuracy.

3.1.5 Main Challenges of Speech-Audio ML

Building high-performance speech systems demands not only the right algorithm but also careful attention to the peculiarities of audio data. Below, we discuss six core pitfalls and strategies to mitigate them.

Insufficient Quantity of Training Audio

Unlike image models that can leverage millions of web-scraped photographs, speech-audio systems often struggle for labeled data. A robust ASR acoustic model may require hundreds of hours of transcribed speech; languages or dialects with scant resources see word-error rates spike. Data augmentation like randomly warping pitch, adding noise at varying SNRs, applying SpecAugment's time- and frequency-masking simulates diversity and can reduce data hunger by 20–30 % without new recordings.

Nonrepresentative Audio Sampling

Even large corpora can mislead models if they misrepresent real-world conditions. A keyword-spotting model trained only on studio-quality speech will fail in noisy cafés or car cabins. Similarly, accent, age, and microphone type biases lead to poor cross-speaker performance. **Domain-invariant training** using adversarial losses to encourage features that ignore

recording conditions - and **multicondition training** with matched noise profiles are critical.

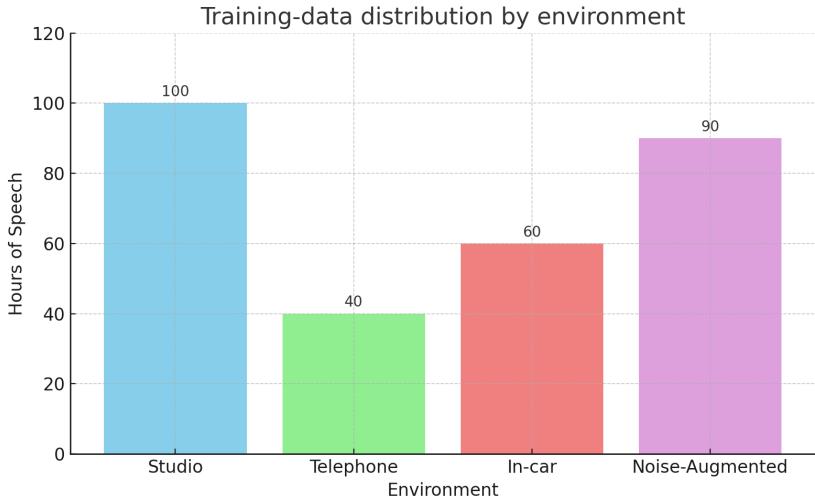


Figure 3.8: Training-data distribution by environment: studio vs. telephone vs. in-car vs. noise-augmented.

Noisy and Reverberant Recordings

Room reverberation smears spectral peaks and degrades pitch-tracking. A model trained only on clean read speech may see a 10–15 % relative drop in keyword-spotting F1 when faced with 0.6 s RT60 reverberation. **Front-end enhancement** (WPE dereverberation, RNNoise) and **robust feature design** (RASTA filtering, perceptual linear prediction) can recover lost performance.

Irrelevant or Redundant Features

Extracting dozens of MFCCs, spectral contrasts, and prosodic measures can flood a classifier with correlated or uninformative inputs. **Feature-selection** via correlation-thresholding ($\rho < 0.9$), mutual-information ranking, or embedded methods like tree-based importance reduces dimension from 60 → 15, cutting inference time and overfitting risk.

Overfitting Acoustic Models

Complex CNNs or LSTMs may achieve near-perfect accuracy on a validation set but fail in deployment. Overfitting manifests as spurious sensitivity to speaker- or utterance-specific artifacts. Counter-measures include **dropout**, **weight decay** (ℓ_2 regularization), **early stopping** based on held-out loss, and **mixup** augmentation that blends pairs of spectrograms.

Underfitting Simple Models

In contrast, overly simplistic models, such as a single-layer linear classifier on raw spectral energies, may not fit, showing high bias and poor accuracy even on training data. Underfitting is remedied by selecting more expressive architectures (deep or wide networks), increasing context windows, or adding higher-order spectral features (e.g., delta-deltas, group delay).

3.1.6 Testing, Validation, and Hyperparameter Tuning

In machine learning for speech and audio, careful data partitioning is critical for obtaining reliable estimates of model performance. A model that performs well on the training data but fails on new, unseen recordings is said to *overfit*, which is a common risk given the high variability in speech (e.g., different speakers, accents, background noise). To avoid this, datasets are typically divided into three non-overlapping subsets: training, validation, and test sets. For speaker recognition and speech tasks in particular, these splits are made *speaker-disjoint*—ensuring that recordings from the same speaker never appear in both training and testing—so that the evaluation truly reflects generalization to new speakers.

Training Set. The training set is used to fit the model parameters. In speech tasks, this may involve learning weights for neural networks, estimating parameters of Gaussian mixture models, or fitting decision boundaries for classifiers such as SVMs. The training set should be large and diverse, covering a wide range of speakers, phonetic content, and

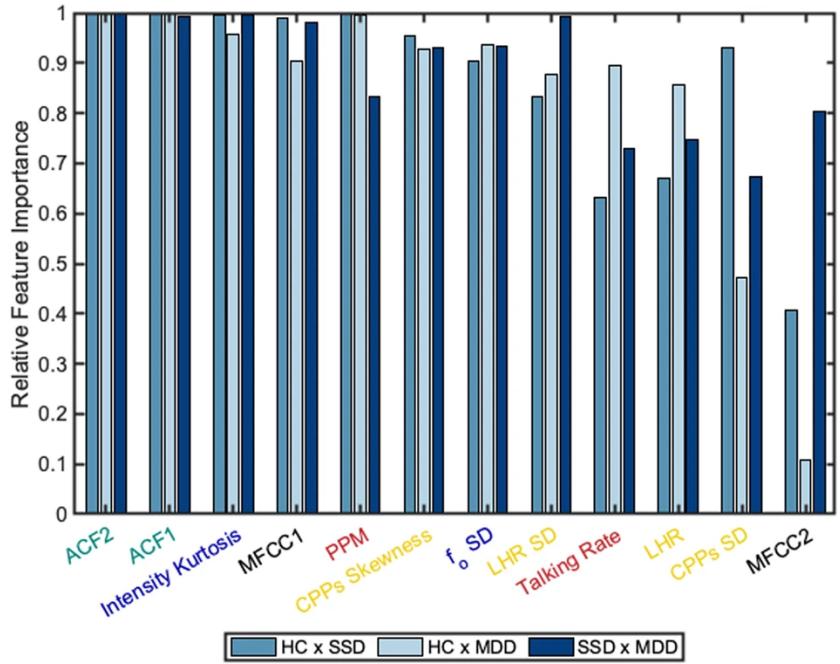


Figure 3.9: Feature-importance scores for a Random Forest keyword-spotting model; low-importance features can be pruned.

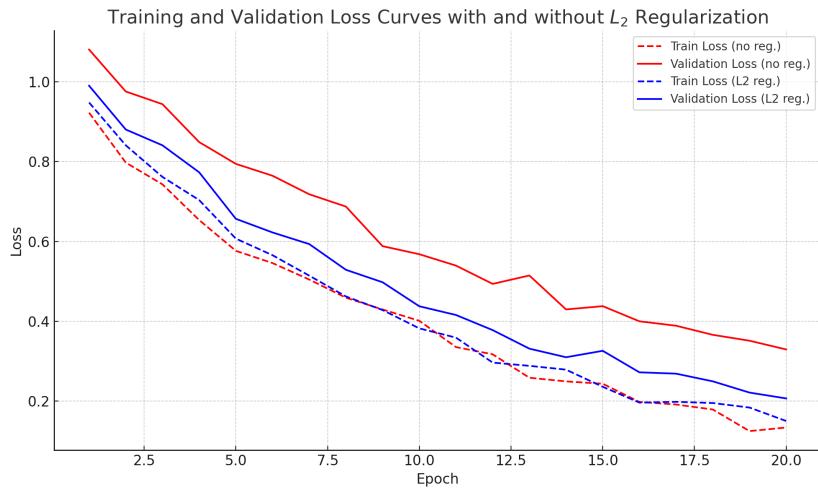


Figure 3.10: Training and validation loss curves with and without ℓ_2 regularization; regularization prevents divergence on the validation set.

acoustic conditions. For example, in automatic speech recognition (ASR), the training data should include different accents, background environments, and recording devices to capture real-world variability.

Validation Set. The validation set is used to tune hyperparameters—settings that control how the model learns but are not directly optimized during training. Examples include the learning rate, number of hidden layers, dropout ratios, and convolutional kernel sizes. For speech systems, validation sets are often designed to mirror deployment conditions: if the application is call-center speech, the validation data may consist of telephone-quality recordings. Model choices and hyperparameters are adjusted based on validation performance to prevent overfitting. Importantly, the validation set is *not* used to train model parameters; it only guides model selection.

Test Set. The test set is held back until the very end of development. It provides an unbiased estimate of how the trained and tuned model will perform on unseen speech data. Because it simulates real-world deployment, the test set should include recordings that differ from those in training and validation in meaningful ways: new speakers, unseen utterances, or different noise conditions. For example, a speaker verification system trained on studio-quality speech should be tested on recordings with realistic background noise or mobile phone channels.

Cross-Validation. In cases where data is limited, *cross-validation* is used. The dataset is divided into k folds, and the model is trained and validated k times, each time using a different fold for validation and the remaining folds for training. This produces more stable error estimates. For audio applications, cross-validation must be conducted carefully to preserve speaker disjointness: entire speakers are assigned to folds, rather than individual utterances, to prevent information leakage.

Hyperparameter Tuning. Hyperparameter optimization is often performed using grid search, random search, or Bayesian optimization. In speech tasks, this may include tuning learning rates, dropout values, spectral feature dimensions (e.g., number of mel filters), or context window sizes. Because speech models can be computationally expensive, random and Bayesian searches are often favored over exhaustive grid search, as they explore the search space more efficiently.

Calibration. Finally, many speech systems (e.g., speaker verification, keyword spotting) output confidence scores or probabilities. These raw scores may be poorly calibrated, meaning they do not accurately reflect true likelihoods of correctness. Calibration methods such as *Platt scaling* (logistic regression on validation scores) or *isotonic regression* are applied to ensure that, for example, a prediction with 80% confidence is indeed correct about 80% of the time. This step is critical in safety- or security-sensitive applications where thresholds for acceptance or rejection must be carefully set.

3.1.7 Conclusion

Effective speech-audio ML demands more than merely choosing an appropriate algorithm; it necessitates careful attention across multiple critical stages. This begins with curating a representative and noise-diverse dataset that includes sufficient hours of speech per speaker and environmental context, ensuring the model is trained on conditions closely resembling real-world deployments. Equally important is the meticulous design of robust acoustic features and effective augmentation strategies, which help simulate a variety of deployment scenarios and bolster model resilience to unexpected variability. Managing model complexity is crucial: Striking a balance between overfitting and underfitting can be achieved through appropriate regularization techniques and systematic model selection procedures. Lastly, rigorous validation methodologies, such as speaker-disjoint splits and thorough cross-validation approaches, are essential for accurately assessing a model’s capacity to generalize beyond the training data. Altogether, these interconnected steps define the success of speech-audio ML systems, enabling reliable performance under diverse and realistic conditions.

Bibliography

- [1] P. Warden, “Speech commands: A dataset for limited-vocabulary speech recognition,” arXiv:1804.03209, 2018.
- [2] G. Tucker and colleagues, “Deep speech 3: 12 000 hours of semi-supervised acoustic model training,” in *Proc. Interspeech*, 2020.
- [3] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.
- [4] A. Baevski *et al.*, “wav2vec 2.0: A framework for self-supervised learning of speech representations,” arXiv:2006.11477, 2020.
- [5] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Pearson Education, 2010.
- [6] Y. He, J. McGraw, H. Kapre *et al.*, “Streaming keyword spotting on mobile devices with conv-tasnet,” in *Proc. Interspeech*, 2019.
- [7] K. P. Murphy, *Machine Learning: A Probabilistic Perspective*. MIT Press, 2012.
- [8] L. van der Maaten and G. Hinton, “Visualizing data using t-sne,” *Journal of Machine Learning Research*, vol. 9, pp. 2579–2605, 2008.
- [9] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. MIT Press, 2018.

Chapter 4: Audio Classification Using MFCCs and Decision Tree Models

This Chapter was based on a class project completed by Nicholas Sparks. All materials, including project codes and results, can be found at the book GitHub repo: <https://github.com/AVHBAC/Audio-Classification-Using-MFCCs-and-Decision-Tree-Models>

Voice interfaces have become a core component of smart home and IoT technologies, offering hands-free, intuitive control over devices. This capability enhances convenience as well as provides critical accessibility benefits for individuals with visual impairments [1]. Through speech, users can initiate tasks such as setting timers, operating appliances, or adjusting environmental controls without the need for visual interaction.

However, most commercial voice systems, such as Amazon Alexa or Google Assistant, rely heavily on cloud computing for speech processing. This dependency presents challenges in both network reliability and privacy. In rural areas, network infrastructure can be inadequate or costly [2], and in any setting, transmitting voice data to cloud servers raises privacy concerns due to the risk of unauthorized access or misuse [3].

Edge computing is a promising solution that enables speech recognition directly on-device, eliminating the need for continuous connectivity and reducing data exposure risks [4]. Yet, implementing such functionality on microcontrollers and other constrained hardware requires lightweight and efficient algorithms. Deep learning approaches are typically infeasible due to their computational demands. Instead, traditional machine learning models—like decision trees or random forests—combined with compact and meaningful features such as Mel-Frequency Cepstral Coefficients (MFCCs), offer a viable path forward [5].

(Refer to Chapter 2 for the theoretical background on MFCCs, preprocessing techniques, and feature normalization.) When combined with basic statistical operations, these features provide a compact set suitable for training simple classifiers on small devices. Additionally, models must remain robust under various conditions, such as background noise, accent variability, and limited sample sizes. Preprocessing and validation strategies such as k-fold cross-validation and statistical testing (e.g., ANOVA, chi-squared) are essential to ensure generalization and reliability [6].

This chapter explores such a design and evaluation of an offline speaker recognition system using MFCC-based features and a tree-based classifier architecture. This approach will be suitable for embedded systems deployed in disconnected or privacy-sensitive environments.

4.1 Theoretical Background

The development of voice-controlled systems in embedded environments requires fundamental knowledge in both signal processing and machine learning theory. This section outlines the essential theoretical concepts underpinning the system, including speech signal characteristics, feature extraction techniques, and the principles behind decision tree and ensemble-based classifiers.

4.1.1 Decision Trees and Random Forest

Decision Trees. Decision trees are one of the most widely used algorithms in supervised machine learning, popular for their simplicity, interpretability, and versatility in handling both classification and regression tasks. A decision tree represents a model in the form of a hierarchical tree structure, where each internal node corresponds to a test on a particular feature, each branch represents the outcome of that test, and each leaf node corresponds to a predicted class label (in classification) or a continuous value (in regression).

The training process for a decision tree involves recursively partitioning the dataset into increasingly homogeneous subsets. At each step, the algorithm chooses the feature and threshold that best splits the data. The “best” split is determined by a *splitting criterion*, a function that measures the quality of a split. Two of the most common criteria are:

$$\text{Gini Impurity} = 1 - \sum_{i=1}^C p_i^2, \quad \text{Entropy} = - \sum_{i=1}^C p_i \log_2(p_i),$$

where p_i is the proportion of samples in class i at a given node and C is the number of classes. A lower value of impurity or entropy indicates that the node is more “pure” (i.e., contains samples mostly from a single class). The goal at each node is to choose the feature and threshold that maximizes the reduction in impurity, also called the **information gain**. This

recursive partitioning continues until a stopping condition is reached, such as a maximum tree depth, a minimum number of samples per node, or complete purity in the leaves.

The advantages of decision trees include their interpretability (the model can be visualized and understood as a sequence of decisions), their ability to handle mixed data types (categorical and continuous), and their efficiency in training and prediction. However, decision trees also have important limitations: they are prone to *overfitting* when grown too deep, they can be unstable (small changes in the data can lead to very different trees), and they often perform worse than more sophisticated models when used alone. To address these issues, ensemble methods were developed.

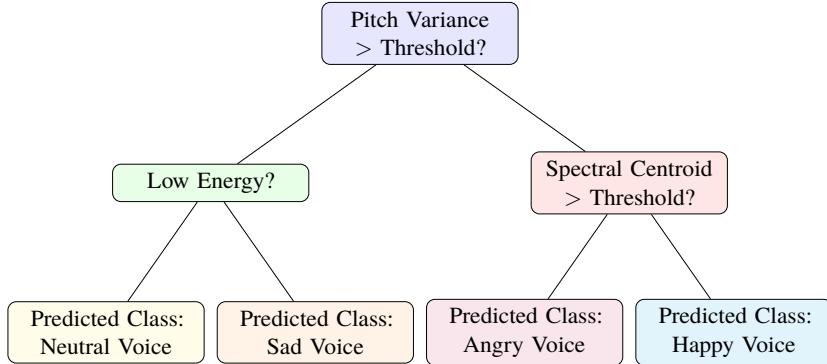


Figure 4.1: Example of a decision tree for voice emotion recognition. Features such as pitch variance, energy, and spectral centroid are used to classify speech into emotional categories.

Random Forests. The Random Forest [7] is one of the most popular ensemble methods, which can address the shortcomings of single decision trees. A random forest is an ensemble of decision trees, each trained on a different bootstrap sample of the training data. In addition, at each node split, the algorithm does not consider all features but rather a random subset of features. This twofold randomization—random sampling of data and random sampling of features—ensures that the individual trees are de-correlated, which reduces the likelihood that they all make the same mistakes.

During training, each tree is grown to full depth (or until a stopping criterion is reached), without pruning, which allows the trees to capture complex patterns. During inference, the forest makes predictions by aggregating the outputs of all individual trees:

- For classification tasks, the prediction is the class that receives the majority of votes across all trees.
- For regression tasks, the prediction is the average of all tree outputs.

Random Forests offer several advantages: they provide strong predictive performance out-of-the-box, they are relatively robust to noise and outliers, and they require little hyperparameter tuning. They also offer estimates of feature importance by measuring how much each feature contributes to reducing impurity across the forest. This makes them useful not only for prediction but also for feature analysis. On the downside, they can be computationally heavier than a single decision tree and less interpretable, since the final model is the aggregate of many trees.

Applications in Speech and Audio. In speech and audio processing, decision trees and random forests have been widely used for tasks such as phoneme classification, keyword spotting, and speaker recognition. For example, in speaker identification, decision trees may split on features such as pitch, energy, or MFCCs, while random forests combine multiple trees to achieve high accuracy across diverse speakers and noisy conditions. Random forests are especially attractive for embedded or low-power devices because they are lightweight compared to deep learning models, yet still provide robust performance.

4.1.2 Embedded System Constraints

Microcontroller-based implementations of speech recognition systems must consider strict limits on processing power, memory, and energy consumption. Unlike large-scale ASR models based on deep learning, embedded solutions benefit from algorithmic simplicity and efficient feature representations. Techniques such as feature downsampling, low-dimensional MFCC selection (see Chapter 2 for MFCC details), and model pruning are commonly used to meet performance targets while maintaining low latency and high accuracy [8].

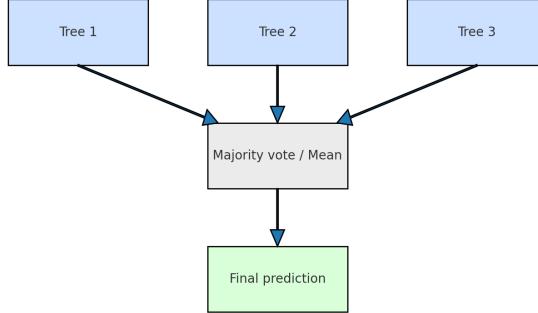


Figure 4.2: Illustration of the Random Forest ensemble: multiple decision trees trained on random subsets of data and features, aggregated by majority voting (classification) or averaging (regression).

4.1.3 Dataset

For the purposes of this study, a compact yet sufficiently diverse dataset was required to facilitate real-time speech classification on resource-constrained hardware. The **Free Spoken Digit Dataset (FSDD)** is commonly used in lightweight model benchmarking, making it a standard reference for comparing the performance of embedded voice systems

FSDD is an open-access dataset consisting of recordings of the digits zero through nine, spoken multiple times by a small number of speakers. In total, the dataset includes **3,000 utterances**—with approximately **300 samples per digit class**. These are distributed across **six speakers**, each contributing 50 repetitions of all 10 digits. The speakers represent varying European accents, introducing inter-speaker variability that is beneficial for evaluating speaker-independent and speaker-dependent models.

Each audio sample is recorded as a mono WAV file, with a sampling rate of 8 kHz and a duration ranging from approximately 0.5 to 1.0 seconds. All samples were captured in relatively clean acoustic conditions with minimal background noise, making the dataset ideal for training in environments where signal clarity is assumed or pre-processed.

The dataset was partitioned into training and testing sets using an 80/20 stratified split to maintain class balance. This resulted in 2,400 training samples and 600 testing samples.

Attribute	Value	Description	Type	Relevance
Dataset Name	FSDD	Free Spoken Digit Dataset	Audio Corpus	Public and lightweight
Classes	10	Digits 0 through 9	Categorical	Multi-class classification
Total Samples	3,000	300 per class	WAV files	Balanced data
Speakers	6	Varying accents	Human voices	Speaker variability
Samples per Speaker	500	50 utterances × 10 digits	Speech instances	Consistent per speaker
Sampling Rate	8 kHz	16-bit PCM Mono	Digital audio	Ideal for embedded systems
Duration	~0.5–1.0s	Per sample	Time domain	Short speech commands
Environment	Low noise	Minimal background	Acoustic quality	Clean baseline for testing
Train/Test Split	80/20	Stratified by digit	Evaluation setup	Generalization testing

Table 4.1: Overview of the Free Spoken Digit Dataset (FSDD)

4.1.4 Denoising

In order to de-noise our data, we need to understand the nature and possible sources of the noise our system is likely to encounter. The application space for this project is entirely domestic, which adds a high degree of noise controllability on the user’s side. The primary sources of noise we would expect from this space come from house hold systems such as HVAC and filtration systems as well as other occupants of the space. The noise contribution and controllability of these two sources can quickly increase the system’s complexity as a whole.

Household appliances often emit low-level, constant noise while running. Devices such as heaters, air conditioners, air filters, and fans fall in this category and are often run for long periods. (For theoretical details on noise types and filtering methods, refer to Chapter 2, Section 2.1.2.) While such noise poses a challenge to voice recognition systems, it often does not vary on the short time scale at which our system operates, which makes it easier to identify and filter out. One standard method of handling this is mean power removal, which emphasizes speech-dependent changes in the signal.

Time-varying sources of noise pose a more significant challenge as their profiles closely match the speech data. Processing signal pollution due to multiple speakers or music is beyond the capability of a low-power embedded system. However, since the user often has great control over these domestic noise sources, their impact is considered a non-issue in this application. Thus, de-noising methods for speaker or music separation will not be explored here.

As mentioned above, the nature and amount of noise expected in a domestic environment will likely be below the threshold that will greatly affect the system. Tests would need to be conducted in order to determine the noise levels and types where performance degrades, and filters would need to be designed accordingly. Since the current dataset is relatively clean, we will augment it with added noise for testing. For now, we will not be conducting filtering, as mean power removal is already applied during feature normalization and will be handled through feature processing.

4.1.5 Feature Selection

For speech recognition, many different features can be extracted. In this study, we focused on MFCCs, which are widely used for their ability to capture perceptually relevant aspects of speech. As discussed in Chapter 2, MFCCs are derived by transforming the frequency domain representation of the signal onto the Mel scale, making them closely aligned with human auditory perception.

To validate the discriminative power of these features, we performed a statistical analysis to identify which coefficients provided the strongest separation across classes. Figure 4.3 illustrates the STFT of a speech sample, highlighting the harmonic structure present in the signal that motivates cepstral analysis.

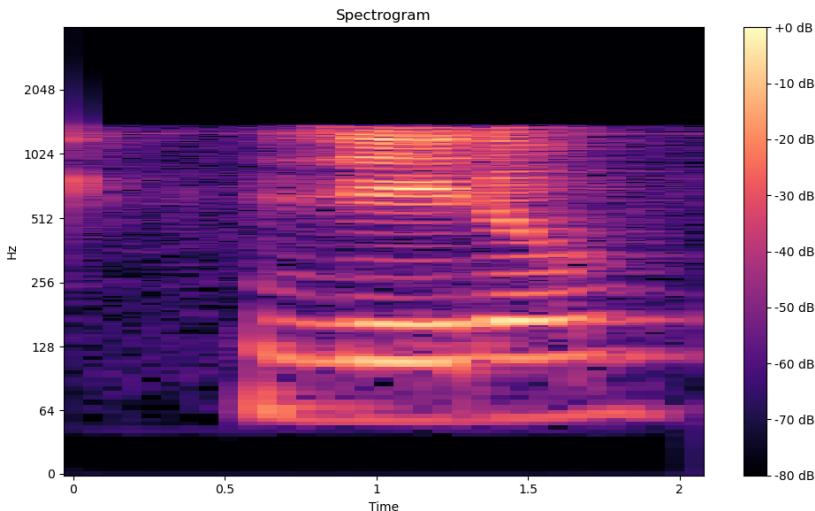


Figure 4.3: Short Time Fourier Transform of 0_George_1.wav (file can be found in the dataset). The harmonic structure of the waveform is visible, providing the basis for cepstral analysis.

4.1.6 Feature Extraction

Each audio sample was first processed using a pre-emphasis filter to boost higher frequency components. We then extracted MFCC coefficients along with RMS power using a 20 ms frame length. The mean of each feature per sample was computed, followed by Z-normalization.

To determine the optimal feature configuration, we conducted ANOVA-based testing (method described in Chapter 3) to identify which MFCC coefficients yielded the highest inter-class separability. An example of the best feature separation is shown in Figure 4.4, where speakers are color-coded. This figure shows that speakers tend to occupy distinct regions in the distribution, confirming the discriminative potential of MFCC features. For training, we varied the number of MFCC features between 8 and 16, balancing classification performance and computational efficiency.

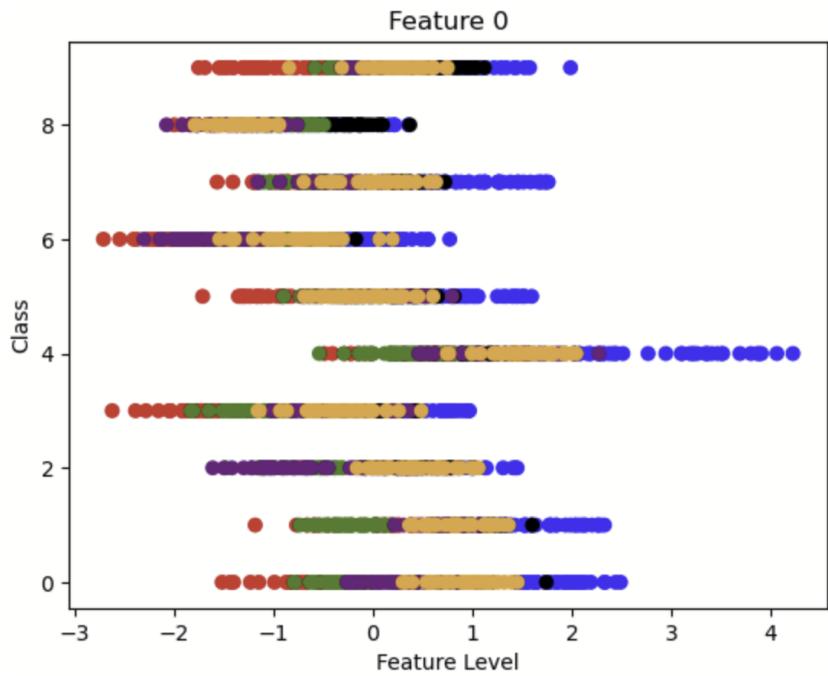


Figure 4.4: Distribution of the MFCC coefficient with the best separation according to ANOVA. Distinct clusters for speakers highlight the potential of MFCCs for speaker classification.

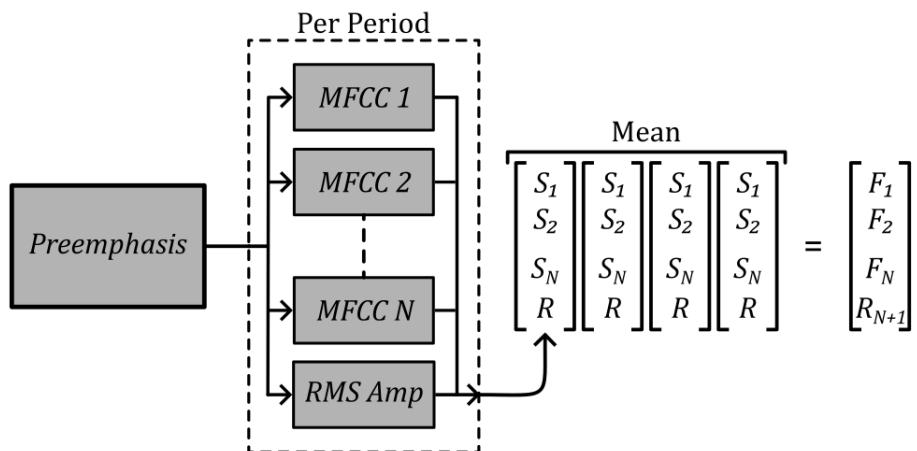


Figure 4.5: Feature extraction processing pipeline used in this study.

4.1.7 Model Parameters

Because our application targets resource-constrained environments, model size and complexity must be carefully controlled. Methods such as neural networks and SVMs provide strong performance but rely on many multiply-add operations, which are computationally costly. Instead, lightweight models such as decision trees and their ensembles (e.g., Random Forests) are attractive. In this context, the most influential parameter is the number of trees: too few may reduce accuracy, while too many increase computation and risk overfitting. Selecting an appropriate balance ensures efficiency without sacrificing reliability.

It was empirically found that the model did not improve after 20 decision trees. As such, only the number of MFCC coefficients will be varied for the parameter search. For this, the number of MFCCs was varied between 8 and 16 coefficients. These models were trained using a random split per digit, ensuring equal representation of each class in the training and testing sets. During training, a k-fold cross-validation approach with five folds was used to measure the model's performance accurately.

4.2 Results and Discussion

Here, we report global trends (F1 and accuracy vs. MFCC count) and visualize classification behavior through confusion matrices at representative settings from very low to near-saturated feature counts.

4.2.1 Global Trends: Effect of MFCC Dimensionality

Figure 4.6 shows the macro F1-score as a function of the number of MFCC coefficients. Performance improves steeply when moving from very low-dimensional features to moderate dimensionality; gains begin to taper near 12–14 coefficients. Accuracy follows the same overall trend (Fig. 4.7). Empirically, the results indicate that *14 coefficients* offer an effective balance between accuracy and computational cost for embedded deployment.

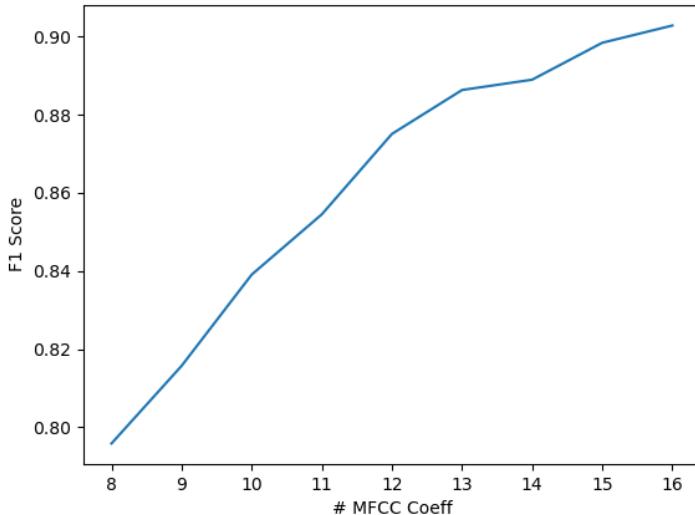


Figure 4.6: Random Forest: macro F1-score versus the number of MFCC coefficients. F1 increases rapidly up to $\sim 12\text{--}14$ coefficients, then saturates.

To make the progression concrete, Table 4.2 summarizes performance ranges observable across repeated runs and the global plots in Figs. 4.6–4.7. Accuracy and F1 climb sharply up to 8–12 coefficients; improvements between 14 and 16 coefficients are small (< 1% absolute), indicating diminishing returns.

4.2.2 Confusion Matrix Progression

We now examine how classification structure evolves as MFCC dimensionality increases. With *2 coefficients* (Fig. 4.8), the model lacks sufficient spectral detail to separate classes; the diagonal is weak and errors are widespread. Increasing to *4 coefficients* (Fig. 4.9) introduces a visible diagonal, but off-diagonal mass is still substantial. At *8 coefficients* (Fig. 4.10),

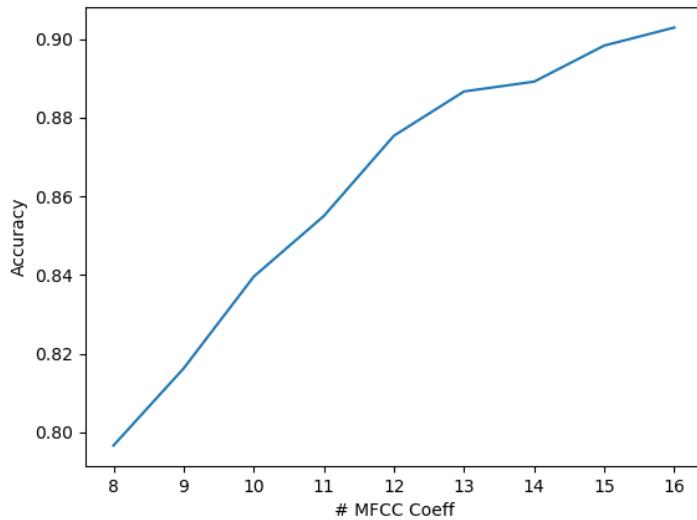


Figure 4.7: Random Forest: accuracy versus the number of MFCC coefficients. Accuracy mirrors F1 and plateaus near 14–16 coefficients.

Table 4.2: Random Forest performance versus MFCC dimensionality (representative ranges)

MFCC Coefficients	Accuracy (%)	F1
2	~20–25	~0.20–0.25
4	~40–50	~0.40–0.50
8	~70–80	~0.70–0.78
12	~84–88	~0.83–0.87
14	~89–91	~0.88–0.90
16	~89–91	~0.88–0.90

a strong diagonal emerges, and accuracy moves into a practically useful regime. Near the plateau, *10–12 coefficients* (Figs. 4.11–4.12) further consolidate correct predictions, and *14–16 coefficients* (Figs. 4.13–4.14) achieve the cleanest diagonals, with residual confusions concentrated among a few acoustically similar pairs.

		Confusion Matrix of SVM classifier									
		Zero	One	Two	Three	Four	Five	Six	Seven	Eight	Nine
Actual Outputs	Zero	55	39	12	9	35	12	3	14	43	23
	One	38	59	18	1	25	6	2	15	33	33
	Two	49	15	48	10	27	2	1	20	48	30
	Three	14	4	16	41	5	26	21	23	49	40
	Four	45	25	19	1	82	6		11	21	28
	Five	12	13	3	17	13	71	8	22	35	42
	Six	17	19	11	9		33	78	14	42	14
	Seven	43	21	4	8	13	31	5	29	39	39
	Eight	14	4	12	29	11	28	9	19	92	30
	Nine	23	23	4	17	24	6	7	22	48	71

Figure 4.8: RF confusion matrix with 2 MFCC coefficients (1 tree). Minimal diagonal structure; pervasive cross-class errors.

4.2.3 Discussion and Numerical Interpretation

The figures collectively show that feature dimensionality is the dominant factor for RF performance in this task. With *2 coefficients*, macro F1 is typically near 0.20–0.25 and accuracy hovers around 20–25%, reflecting the absence of robust diagonal structure (Fig. 4.8). Increasing to *4 coefficients* roughly doubles accuracy to the ~40–50% band. At *8 coefficients*, the diagonal becomes dominant and accuracy moves into the ~70+% range (Fig. 4.10). The *10–12 coefficient* range consolidates these gains further, approaching mid-to-high 80s (Figs. 4.11–4.12). The *14–16 coefficient* regime produces the cleanest matrices with accuracies around ~89–91% and F1 around ~0.88–0.90 (Figs. 4.13–4.14); gains between 14 and 16 are small (<1% absolute), confirming saturation.

Persistent misclassifications are concentrated among a handful of acoustically similar pairs (visible as the largest off-diagonal cells), notably 2 vs 3, 3 vs 6, and 7 vs 9.

In practical terms, these results suggest a deployment recommendation: when memory and inference time are tight, configuring RF with approximately *14 MFCC coefficients* is near-optimal; moving to 16 coefficients offers negligible benefit. If additional preprocessing (e.g., compacting or simple dimensionality reduction) is available, the matrices indicate comparable qualitative behavior while enabling tighter runtime/memory budgets. For further error reduction on specific confusable pairs, augmenting MFCCs with prosodic or temporal features (see Chapter 2) is a promising direction.

4.3 Impact on the field

A lot of research has been done on implementing speech recognition models that contain a few words, but this is often below the ten needed for our use case. Other research focuses on improving wake-word recognition in noisy environments. This supports the model of detecting a single wake word and then using online resources for the rest of the data parsing. There does not appear to be much research on implementing a microcontroller-only speech model to accomplish data entry.

Confusion Matrix of SVM classifier										
	Zero	One	Two	Three	Four	Five	Six	Seven	Eight	Nine
Actual Outputs	210	1	8	11	1	1	2	6	5	
Zero	210	1	8	11	1	1	2	6	5	
One	3	156			14	22		3		32
Two	41		150	33	6		2		18	
Three	8		51	146			5	3	25	1
Four	8	16	11		200	1		1		1
Five		35			8	142	1	22		28
Six	1		18	24		1	156		36	1
Seven	5	28	2	5	3	29	4	91	10	55
Eight	3		20	28			7	2	183	5
Nine	6	32			2	12	1	11		181
Expected Outputs	Zero	One	Two	Three	Four	Five	Six	Seven	Eight	Nine

Figure 4.9: RF confusion matrix with 4 MFCC coefficients (1 tree). Early diagonal emerges; many off-diagonal errors remain.

Confusion Matrix of Random Forest classifier										
	Zero	One	Two	Three	Four	Five	Six	Seven	Eight	Nine
Actual Outputs	206	10	6	5	1	1	1	7	1	7
Zero	206	10	6	5	1	1	1	7	1	7
One	4	176	1	1	23	15		3	1	6
Two	12	2	189	17	2	1	4	18	1	4
Three	6		18	175			13	9	17	1
Four	2	14			214	8				
Five	3	10	2	1	8	192	1	9		10
Six	2		4	17			202	2	9	1
Seven	7	4	17	5	1	8	3	170	1	16
Eight	3		3	15			8	2	217	
Nine	12	6	3	1	1	13		15	1	193
Expected Outputs	Zero	One	Two	Three	Four	Five	Six	Seven	Eight	Nine

Figure 4.10: RF confusion matrix with 8 MFCC coefficients (20 trees). Stronger diagonal and markedly higher accuracy.

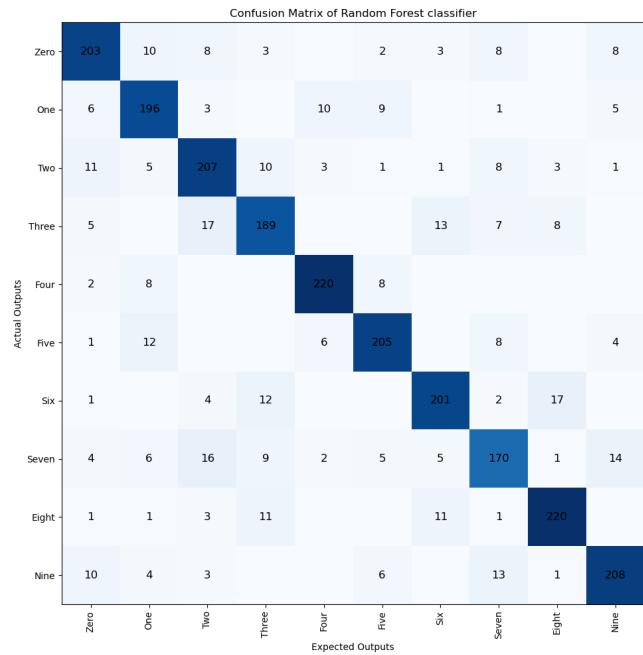


Figure 4.11: RF confusion matrix with 10 MFCC coefficients (20 trees). Confusions reduce further as dimensionality increases.

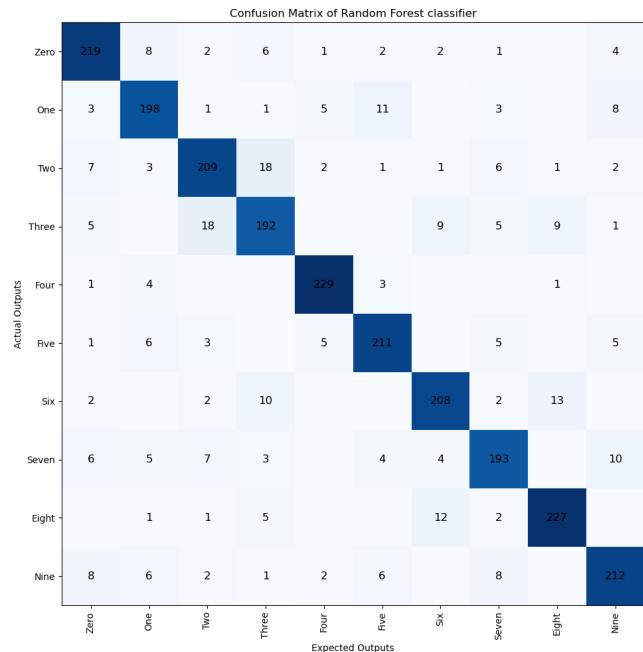


Figure 4.12: RF confusion matrix with 12 MFCC coefficients (20 trees). Near-plateau behavior with few persistent confusions.

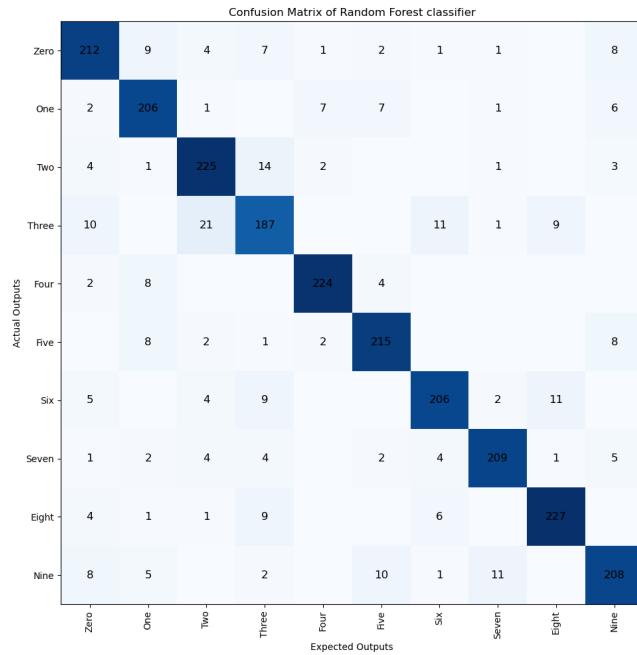


Figure 4.13: RF confusion matrix with 14 MFCC coefficients (20 trees). Plateau region: clean diagonal; residual errors center on a few digit pairs.

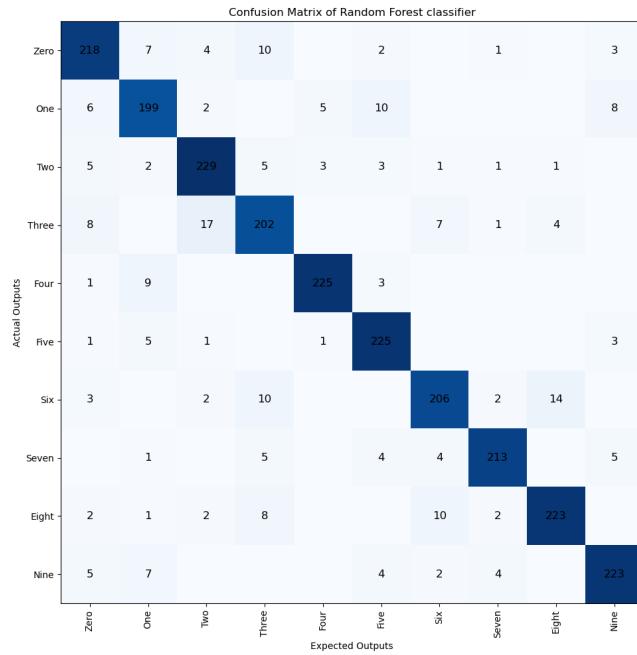


Figure 4.14: RF confusion matrix with 16 MFCC coefficients (20 trees). Similar to 14 coefficients; further gains are marginal.

Bibliography

- [1] K. Yadav and L. Jain, “Voice-enabled interfaces for visually impaired people,” *International Journal of Human-Computer Studies*, vol. 136, p. 102385, 2020.
- [2] A. Singh and M. Prakash, “Smart homes and rural limitations: A case study of india,” *Procedia Computer Science*, vol. 185, pp. 112–118, 2021.
- [3] Y. Zhou and R. Jain, “Security and privacy issues in voice-enabled internet of things (iot),” *ACM Computing Surveys*, vol. 53, no. 5, pp. 1–36, 2020.
- [4] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, “Edge computing: Vision and challenges,” *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.
- [5] S. Davis and P. Mermelstein, “Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences,” *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 28, no. 4, pp. 357–366, 1980.
- [6] F. Weninger, H. Erdogan, and S. Watanabe, “Speech enhancement with lstm recurrent neural networks in noisy and reverberant environments,” *Computer Speech & Language*, vol. 31, pp. S1–S27, 2015.
- [7] L. Breiman, “Random forests.” *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.
- [8] A. Sharma, M. Jain, and V. Kumar, “Speech recognition on the edge: A review of challenges and solutions,” *IEEE Access*, vol. 9, pp. 121415–121432, 2021.

Chapter 5: Key-Word Detection using Random Forest

This Chapter was based on a class project completed by Connor Blais. All materials, including project codes and results, can be found at the book GitHub repo: <https://github.com/AVHBAC/Key-Word-Detection-using-Random-Forest-onboard>. An alternative similar project not implemented on board can be found here: <https://github.com/AVHBAC/Key-Word-Detection-using-Random-Forest>

Many individuals experienced upper extremity impairments as a result of neurological disorders or traumatic events such as stroke, traumatic brain injury (TBI), and cerebral palsy. These conditions often led to hyperactive reflexes and abnormal muscle tone, significantly hindering a person's ability to open their fingers. To address this challenge, various hand orthoses had been designed to facilitate hand opening. A common feature among these orthoses was their dependence on external control mechanisms.

AI voice recognition was one possible control method where verbal commands triggered the desired action. Though less commonly used, it could still be an effective solution. This chapter focused on applying AI voice recognition in conjunction with machine learning techniques.

We used the Dataset_EN dataset, which was comprised of 3735 voice lines from the game series "Caduceus Trauma Center." The keywords "Hold," "Let," and "Go" were isolated from this dataset.

While deep learning models were frequently used for keyword detection in current research, these complex models were not always necessary for mobile applications. Furthermore, more research was needed on the Coral Micro device. At the time of writing, no published research existed on using keyword detection specifically for Coral Micro. This chapter aimed to explore this relatively uncharted area and create a well-designed and thought-out machine-learning model for keyword detection on the Coral Micro.

5.1 Theoretical Foundations

Keyword spotting is treated as a **supervised classification** task: every training clip is paired with its correct label, and the classifier must assign one of four categories—Hold, Let, Go or General—to each new clip. The goal is therefore categorical, not numeric, prediction.

We first cleaned the microphone stream by applying a low-pass filter at 8 kHz and trimming samples below -20 dB. This reduced aliasing, removed leading and trailing silence, and made the data more representative of real conditions.

The continuous audio was divided into overlapping 20 ms frames, providing units that are short enough to be quasi-stationary while still capturing speech dynamics.

From each frame we extracted two simple time-domain features: RMS Energy and Zero-Crossing Rate (ZCR). For every utterance we computed the mean, median, standard deviation, skewness, and kurtosis of both, forming a ten-dimensional feature vector. ANOVA confirmed all features were significant ($p < 0.05$).

We trained a Random Forest of 50 trees (max depth 10) using bagging and class weighting to address imbalance. This configuration balanced accuracy and model size, yielding a 40 kB model with 1.2 ms inference time on the 64 MHz Cortex-M4.

Complexity was controlled by limiting tree depth, applying minimum leaf sizes, and relying on bagging and random feature sub-sampling.

Performance was assessed using balanced accuracy, F_1 -score, confusion matrices, and out-of-bag error, ensuring fair evaluation across minority and majority classes.

The trained model was exported to plain C with `micromlgen`. RMS and ZCR statistics were computed on-device, assembled into the feature vector, and passed to the classifier. Predictions above 0.8 confidence triggered GPIO outputs to drive the orthosis motors.

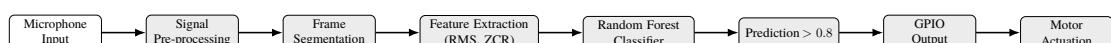


Figure 5.1: Firmware flow from audio capture to motor actuation.

5.2 Methodology and Results

5.2.1 Dataset Description

The dataset we used to create our classifier was the **Dataset_EN** dataset created by tanooki426, which could be found on [huggingface.co¹](https://huggingface.co/datasets/tanooki426/Datasets_EN). This dataset had not been used in any published research that could be found at the time of writing this. There were 25 subjects in the dataset, and all audio tracks had a sample rate of 22050Hz. This dataset was comprised of 3735 voice lines, and these voice lines had 206 instances of the word “Let,” 48 instances of the word “Go,” and 15 instances of the word “Hold.” These instances were extracted into individual audio files, making up the dataset used for training and testing. For our dataset, we used 31 instances of the word “Let,” 48 instances of the word “Go,” and 15 instances of the word “Hold.” Furthermore, random words from these files were saved as a “general” category to ensure we did not detect random speech as one of our keywords. This “general” category consisted of 38 instances of random words. From here, the dataset was split roughly 90%-10% between training and testing.

5.2.2 Data Denoising

Our dataset was effectively noise-free; however, this would not be the case when taking live samples. For this reason, we applied a simple lowpass filter to our training set to mimic live conditions. Our corner frequency was 8kHz as the Coral Micro had an audio sample rate of 16kHz. Furthermore, after filtering, we clipped the audio so that anything below -20dB was removed from the beginning and end of our clips. This was done to ensure no silence was in our training data. We did not want the classifier to correlate silence to any of our keywords falsely. Later in the chapter, we verified that our filtering did not cause a meaningful reduction in accuracy.

Figures 5.2 and 5.3 showed the waveform of a random audio clip before and after filtering.

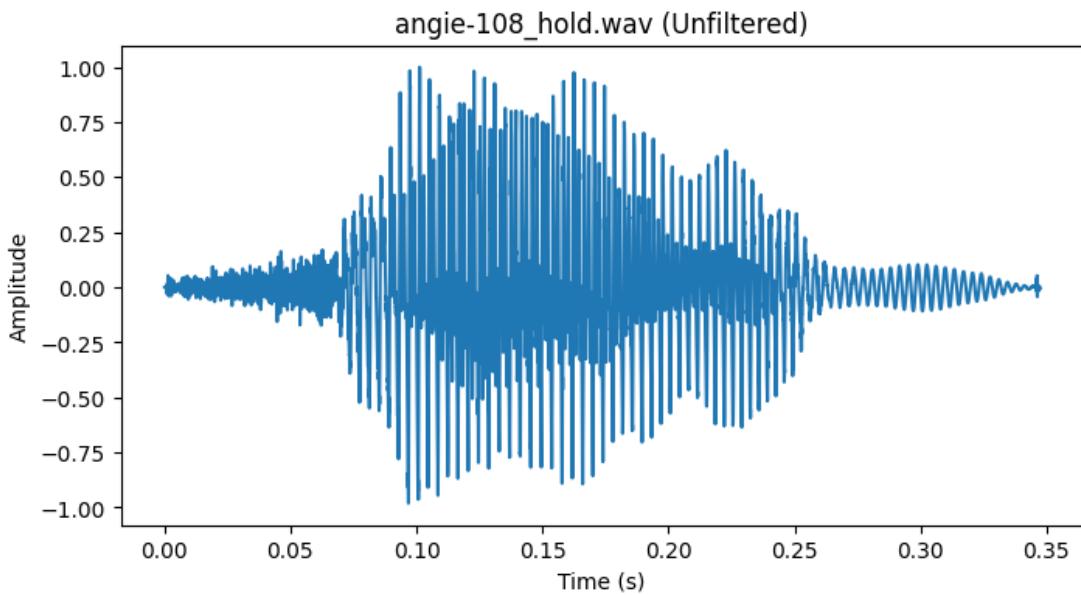


Figure 5.2: Waveform of Unfiltered Audio

¹The original dataset can be found here: https://huggingface.co/datasets/tanooki426/Datasets_EN

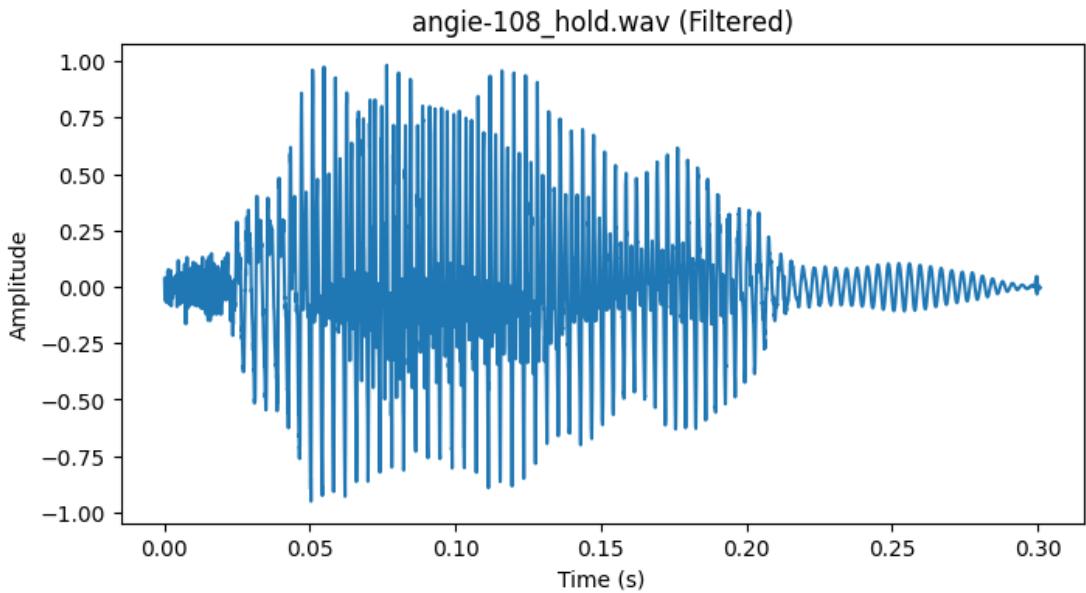


Figure 5.3: Waveform of Filtered Audio

Figures 5.4 and 5.5 showed the Log-Mel Spectrograms for the filtered and unfiltered audio clip.

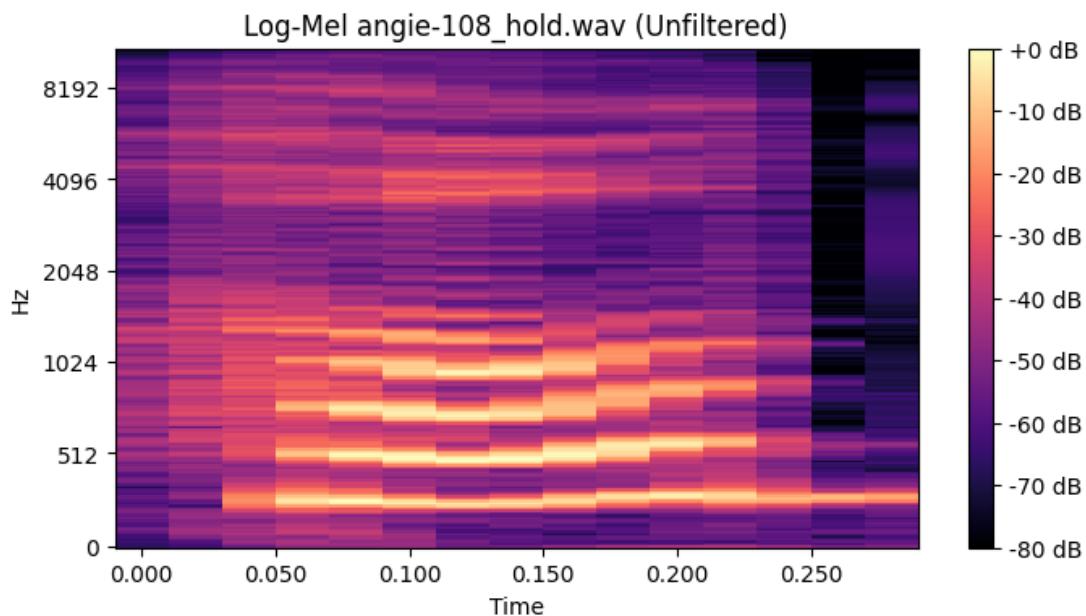


Figure 5.4: Spectrogram of Unfiltered Audio

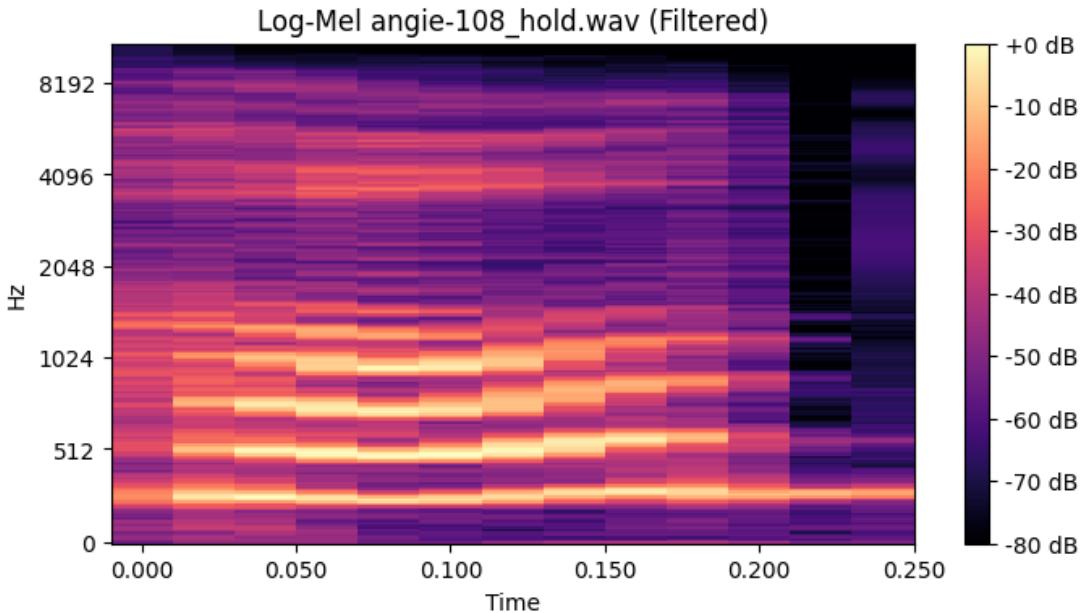


Figure 5.5: Spectrogram of Filtered Audio

5.2.3 Feature Selection & Computation

The features we extracted were the mean, median, st dev, skew, and kurtosis of the signal's RMS and ZCR². These were chosen because they were computationally inexpensive and relatively easy to implement in C++.

The computation of mean³ and median was straightforward and was not covered here. The calculation of st dev used Formula 5.1 and was implemented with the following C++ code:

```
float calcSTD(float* input, int N, float mean){
    float stDev = 0;
    for (int i = 0; i < N; i++){
        stDev += std::pow(*(input + i) - mean, 2);
    }
    return std::sqrt(stDev/s);
}
```

$$\sigma = \sqrt{\frac{1}{N} \sum_{i=1}^N (x_i - \mu)^2} \quad (5.1)$$

It was important to note the usage of pointers to access the array containing the audio signal. As a reminder, `*(arrayvar + i)` was equivalent to `arrayvar[i]` in usage; however, it required fewer assembly lines to execute. Because of this, it was generally more resource-efficient on embedded platforms to access array values in this fashion.

Skewness was calculated using Formula 5.2 and was implemented with the following C++ code:

²Zero Crossing Rate

³mean = μ

```

float calcSkew(float* input, int N, float mean, float stdDev){
    float skew = 0;
    for (int i = 0; i < N; i++){
        skew += std::pow(*(input + i) - mean, 3);
    }
    return skew/((N-1)*std::pow(stdDev,3));
}

```

$$skew = \frac{\sum_{i=1}^N (x_i - \mu)^3}{(N - 1)\sigma^3} \quad (5.2)$$

Finally, kurtosis was calculated using Formula 5.3 and was implemented with the following C++ code:

```

float calcKert(float* input, int N, float mean){
    float Kert = 0;
    float M2_N = 0;
    float M2_D = 0;
    for (int i = 0; i < N; i++){
        M2_N += std::pow(*(input + i) - mean, 4);
        M2_D += std::pow(*(input + i) - mean, 2);
    }
    return ((N*(N-1))/((N-1)*(N-2)*(N-3))) * (M2_N/std::pow(M2_D,2))
        - (3*(std::pow(N-1,2)/((N-2)*(N-3))));
}

```

$$kurt = \frac{N(N+1)}{(N-1)(N-2)(N-3)} \frac{\sum_{i=1}^N (x_i - \mu)^4}{(\sum_{i=1}^N (x_i - \mu)^2)^2} - 3 \frac{(N-1)^2}{(N-2)(N-3)} \quad (5.3)$$

As stated earlier, each of these was applied to arrays of RMS and ZCR values of the audio signal. These arrays were generated from 20ms windows of the input signal. Individual RMS values were calculated using Formula 5.4 and were implemented using the following C++ code:

```

float calcRMS(int16_t* input, int N){
    float RMS = 0;
    for (int i = 0; i < N; i++){
        RMS += std::pow(*(input + i))/N;
    }
    return std::sqrt(RMS);
}

```

$$RMS = \sqrt{\frac{\sum_{i=1}^N x_i}{N}} \quad (5.4)$$

Individual ZCR values were calculated using Formula 5.5 and were implemented with the following C++ code:

```

float calcRMS(int16_t* input, int N){
    int ZCR = 0;
    for (int i = 0; i < N-1; i++){
        ZCR += (((*(input+i) < 0)
                  &(*input+i+1) > 0))
                  ||((*(input+i) > 0)
                  &(*input+i+1) < 0));
    }
    return ZCR/N;
}

```

$$ZCR = \frac{\sum_{i=1}^N ((x_i < 0 \& x_{i+1} > 0) | (x_i > 0 \& x_{i+1} < 0))}{N} \quad (5.5)$$

Additionally, we verified that the chosen features were unique between classes. To do this, we performed an ANOVA statistical test on filtered and unfiltered datasets. This showed that all features had a p-value < .05, meaning there was a statistically significant difference between classes for each feature.

Now that we had done this, we selected the top five most important features from Table 5.1 and Table 5.2 and trained an additional model using just those five to compare to a model using all ten features.

5.2.4 Random Forest

As stated earlier, we used a Random Forest Classifier to classify our keywords. We started by cross-validating our training set using a Random Forest Classifier with 800 estimators. 800 estimators were significantly more than needed; however, we used this many initially to ensure our data was usable. Using this initial model, we checked the feature importance of both our unfiltered and filtered datasets. The results of this, and the p-values of each feature, are shown in Tables 5.1 and 5.2.

	Mean RMSE	Median RMSE	STD RMSE	Skew RMSE	Kurtosis RMSE
p-value (Unfiltered)	3.07×10^{-6}	3.25×10^{-7}	1.45×10^{-8}	9.98×10^{-5}	2.45×10^{-4}
Importance (Unfiltered)	0.089	0.107	0.140	0.107	1.101
p-value (Filtered)	4.91×10^{-6}	4.16×10^{-7}	5.28×10^{-8}	1.23×10^{-4}	3.05×10^{-4}
Importance (Filtered)	0.090	0.120	0.131	0.113	0.094

Table 5.1: Feature Values for RMSE

	Mean ZCR	Median ZCR	STD ZCR	Skew ZCR	Kurtosis ZCR
p-value (Unfiltered)	2.27×10^{-3}	1.02×10^{-3}	1.21×10^{-1}	4.45×10^{-4}	8.29×10^{-8}
Importance (Unfiltered)	0.085	0.064	0.085	0.102	0.120
p-value (Filtered)	5.51×10^{-3}	6.42×10^{-3}	1.35×10^{-1}	1.46×10^{-3}	2.73×10^{-7}
Importance (Filtered)	0.074	0.067	0.083	0.099	0.129

Table 5.2: Feature Values for ZCR

Using a 5-Fold split at a 99% confidence interval, our unfiltered dataset had a likely accuracy between 45.47% and 69.06%. Our filtered dataset had a likely accuracy between 41.09% and 64.90%. Figures 5.6 and 5.7 show the confusion matrices from the cross-validation. We used 50 estimators and a class weight setting of “balanced.” The class weights were necessary because we had a small pool of data to draw from for the “Hold” and “Go” keywords in comparison to

the “General” and “Let” keywords. To determine the optimal number of estimators, we trained the classifier 100 times, incrementing the number of estimators after each iteration. This produced the graph in Figure 5.10. We observed from this graph that the accuracy for both the filtered and unfiltered datasets were effectively the same, and that the accuracy plateaued at around 50 estimators.

Using the same procedure, we trained a different model using just the top 5 features. With the same 5-Fold split at a 99% confidence interval, our unfiltered dataset had a likely accuracy between 41.96% and 65.73%. Our filtered dataset had a likely accuracy between 39.37% and 63.20%. Figures 5.8 and 5.9 show the confusion matrices from this second cross-validation. Figure 5.11 shows the accuracy vs number of estimators for the top 5 feature model. From this, we observed that the unfiltered dataset suffered greatly from the removal of features, while the filtered dataset experienced only a small reduction in accuracy.

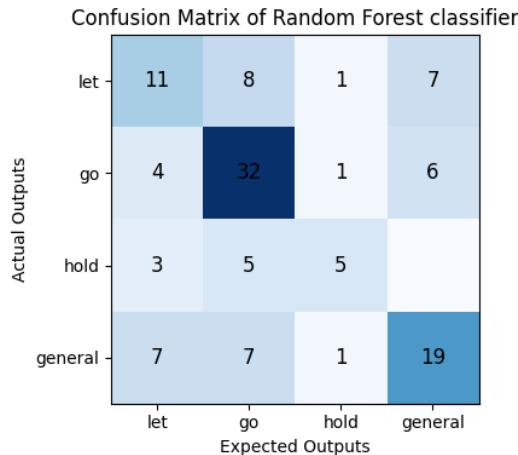


Figure 5.6: Confusion Matrix of Unfiltered Audio

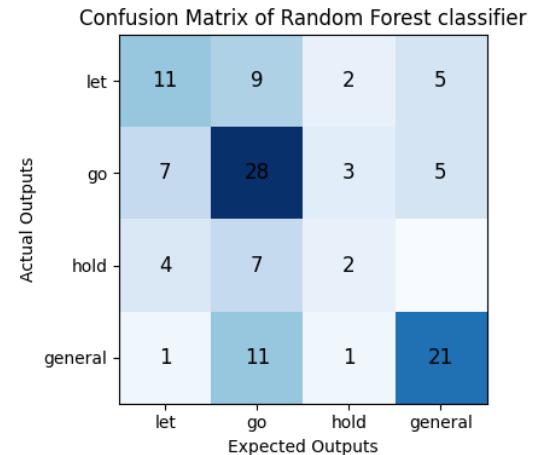


Figure 5.7: Confusion Matrix of Filtered Audio

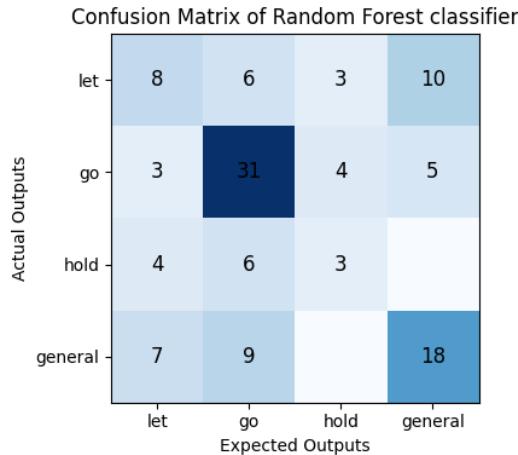


Figure 5.8: Confusion Matrix of Unfiltered Audio (Top 5 Features)

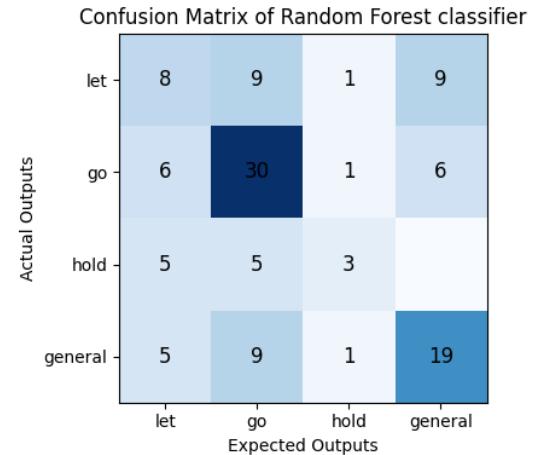


Figure 5.9: Confusion Matrix of Filtered Audio (Top 5 Features)

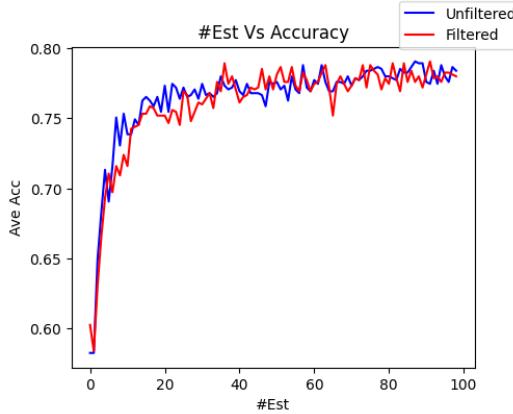


Figure 5.10: Average Accuracy

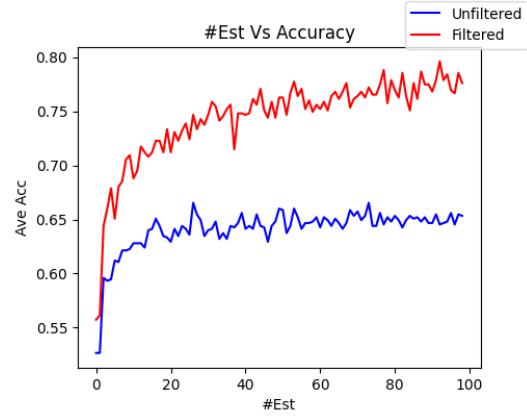


Figure 5.11: Average Accuracy (Top 5 Features)

5.3 Discussion

Using 50 estimators and the top five selected features, our final Random Forest model achieved an accuracy of 73%. This level of performance demonstrated the model’s feasibility for real-time keyword detection on a constrained device like the Coral Micro. Notably, the use of only five features—mean, median, standard deviation, skewness, and kurtosis of RMS and ZCR—allowed for both memory and computational efficiency without sacrificing significant classification performance.

The confusion matrices (Figures 5.6–5.9) illustrated several key trends. First, the “Let” and “Go” classes were more accurately predicted than the “Hold” class, which had the fewest training instances and showed the highest misclassification rate. This imbalance was partially mitigated through the use of class weighting during training.

Filtering the audio slightly reduced accuracy but improved robustness. The filtered model’s confusion matrix showed more evenly distributed errors and less overfitting, particularly when only the top 5 features were used. This supports the hypothesis that applying pre-processing steps like filtering and silence trimming improves generalisation under noisy conditions.

Figure 5.10 and Figure 5.11 further confirmed the model’s stability. Accuracy improvements plateaued after approximately 50 estimators, indicating that additional complexity did not yield measurable benefits. This justified our decision to restrict the final model to 50 trees, which allowed deployment on embedded hardware within strict memory constraints.

Overall, our findings support the idea that light-weight, interpretable models—when carefully optimized—can deliver strong results in embedded speech recognition tasks.

5.4 Conclusion

In this project, we successfully designed and implemented a keyword detection system optimized for deployment on the Coral Micro. By employing basic time-domain audio features such as RMS and ZCR, and leveraging a Random Forest classifier, we achieved an accuracy of 73% while maintaining computational efficiency suitable for real-time embedded applications.

We demonstrated that pre-processing techniques like low-pass filtering and silence clipping not only approximate real-world noise but also help reduce model overfitting. Feature selection using ANOVA and importance metrics guided us toward a minimal but effective feature set, and our analysis revealed that performance did not degrade significantly when using only the top five features.

The final model, converted into C++ using the `micromlgen` library, fits within 40 kB and processes inputs in under 2 milliseconds, meeting the tight resource constraints of the Coral Micro. This work lays the foundation for more responsive and accessible control interfaces in assistive devices, particularly those aimed at users with limited mobility.

Future work may investigate the addition of lightweight spectro-temporal features or ensemble learning techniques that balance performance with energy consumption even further.

width=!,height=!, pages=-, addtotoc=1,chapter,1,Text-independent Speaker Verification using HMM
and SVM,
chap:*connor*,
lais,
pagecommand =

Chapter 7: SVM-based Speaker Classification

This Chapter was based on a class project completed by Ansen Herrick. All materials, including project codes and results, can be found at the book GitHub repo: https://github.com/ahmedajan/SVM_Speaker_Classification

This project was chosen because we believe that speaker classification is a useful field that has many applications. Speaker recognition could be used in a security system to ensure that only certain subject can access certain areas, whether that be in real life with a deployed system on a door lock system, or for use in online data storage systems, where some directories could be locked behind a speaker verification system. It could also be used in conjunction with other biometric systems for added security, for example, a system that uses a fingerprint and speaker verification network. The dataset we selected for this project (VoxForge) is very useful for this task because it contains many audio samples, which all contain recordings of the same prompts, of many unique users. This dataset will provide a lot of diverse accents and speaking patterns that our Support Vector Machine can use to classify. We will use various input feature vectors, such as Mel Frequency Cepstrum Coefficients (MFCC), which capture the spectral properties in a speakers voice adjusted to the range of human hearing. SVM models require an enrollment process for specific users, where voice samples from a single subject will be collected and labeled. The SVM will then learn the unique characteristics of this certain speaker. Then, when a new voice is presented, the SVM will determine how similar the voice features are to the desired class, and based on the similarity, the system will be able to tell how confident it is that the new voice is the same as the desired voice. If this approach is successful, the community will be benefited with a secure system for speaker verification. Which can be used in many applications as explained above. Our System could then eventually be modified for more complex tasks, such as using speaker verification for data encryption and storage.

7.1 Motivation

SVM, or Support Vector Machines are supervised learning algorithms used primarily for classification and regression. They are very effective at separating multidimensional data. Usually SVM models are used to classify two different classes, but in our case, we will use an SVM Model with a 'One vs. Rest' technique to distinguish one class from the rest. This is achieved by finding a maximum separating hyperplane between classes using the provided features. The model can analyze data in the feature space in order to find an effective separating hyperplane. For the case of speaker classification, our SVM model should be very effective at finding an effective separation plane in order to distinguish classes from another.

7.2 Dataset

7.2.1 Dataset collection

The dataset used in this study was gathered from the VoxForge Dataset, a famous open source speech dataset that dates back to 2006, where users are able to upload their own speech files for free use in deep learning algorithms and machine learning. The dataset for this model consists of speech data from 10 different users. These users were selected using a singular criteria, the amount of data each user had uploaded. Each user or 'class' in the dataset has at least 250 voice samples in order to effectively train the SVM model. The dataset consists of 4779 individual voice samples, split between training and testing. Each of the collected samples ranged from 3-15 seconds long and were recorded at a sample rate of 48kHz.

7.2.2 Data pre-processing

There were various pre-processing steps applied to the data in order to ensure the most effective training of the SVM model possible. As said above, there are 10 different speakers or 'classes' used in the dataset, table 2.1 outlines the number of voice files collected from each user, and how the training and testing dataset was split.

The data was then de-noised using a noise reduction in python called 'noisereduce.' The threshold set for this operation was 0.8, this value was used because it significantly reduces the noise in the audio files while still preserving a majority of the original sound quality. Most samples recorded from the years 2007-2009, so noise was very prominent.

User	Training Files	Testing Files	Total Files	Training/Testing Split
akiplaner	468	52	520	90/10
bonzer	351	39	390	90/10
camdixon	537	60	597	89.9/10.1
corno	340	38	378	89.9/10.1
doublesfrogs	262	28	290	90.3/9.7
farmerjack	548	62	610	89.8/10.2
k	333	37	370	90/10
pcsnpy	612	68	680	90/10
ralfherzog	296	38	334	88.6/11.4
rortiz	549	61	610	90/10
Total	4296	483	4779	89.9/10.1

Table 7.1: Number of Files per User in the Dataset

7.3 Feature Selection

7.3.1 Feature extraction

Our SVM model, rather than be trained on the raw audio data, will be trained on a collections of features extracted from the audio files at 20ms intervals. The features extracted are from the MFCC Coefficients (Mel-Frequency Cepstral Coefficients.) MFCC is used to capture the frequency range common for human speech. Each 20ms interval of audio will have 3 MFCC's extracted. The first MFCC Coefficient represents the overall energy of the signal and is dominated by large-scale features in the spectrum. Coefficients 2 and 3 capture more fine details of the overall envelope such as changes in the spectral slope or resonant frequencies of speech. Each of these coefficients will be collected and various operations will be applied to them in order to create our final features. The operations applied are as follows.

- **Mean:** The arithmetic average of a set of values, calculated as:

$$\text{Mean} = \frac{1}{N} \sum_{i=1}^N x_i$$

where x_i are the values and N is the total number of values.

- **Median:** The middle value in the dataset, if a dataset contains the values 1, 2, 3, 4, 5, the median value is 3. If there is an even number of values the median will be the average of the two middle values
- **Standard Deviation (Std Dev):** A measure of how much the data is spread out compared to the mean, given by:

$$\text{Std Dev} = \sqrt{\frac{1}{N} \sum_{i=1}^N (x_i - \text{Mean})^2}$$

- **Skewness:** A measure of the asymmetry, or how much the dataset is "skewed", throughout the values in the dataset. Positive skewness indicates a longer tail on the right, while negative skewness indicates a longer tail on the left.
- **Kurtosis:** A measure of the "tailedness" of the values in the distribution. High kurtosis indicates heavy tails, while low kurtosis indicates light tails. The kurtosis value for a standard normal distribution is 3.0. [1]
- **Maximum and Minimum:** The largest and smallest values in a dataset, respectively.

All the features collected are then concatenated into one feature vector, in our case, because we have 3 MFCC's, and each MFCC will have 7 features, we will have a total of 21 features in our feature vector. The features are also standardized in the pre-processing step using sklearn's 'StandardScalar' function. This function transforms the data to have a mean of 0, which will be much easier for our SVM model to handle since they are very sensitive to the scale of the input features. Standardization also ensures that all features will equally contribute to the model's training.

7.4 Model Training

The SVM model was trained using an RBF (Radial Basis Function) kernel using One vs. Rest classification. A kernel is a function that is capable of computing similarity between two data points in feature space. Which can allow the SVM model to handle non-linear relationships. I chose the RBF kernel because it works well in a variety of problems and is effective even with no prior knowledge about the data structure. [2] The RBF kernel assigns a higher similarity score to

points that are close in the feature space, while applying a lower similarity to points that are far apart. The SVM model was fit to the Training Dataset, the model took 9 minutes to train on average across multiple sessions.

7.5 Results

7.5.1 Accuracy

Following the training of the model, the SVM was used to predict the speakers class with the testing dataset. Across 483 testing audio files, the model reached an accuracy of 97 percent with 3 MFCC Coefficients. Some users or 'classes' had a testing accuracy of 100 percent, while the lowest accuracy for any user was 82 percent. This data can be effectively visualized using a Confusion matrix, which is displayed in Figure 2.1.

7.6 Statistical Tests

7.6.1 ANOVA

An ANOVA feature test was applied to the feature set after training to determine which of the extracted features were most significant in classifying the subject. ANOVA stands for Analysis of Variance, it works by comparing the variance between groups based on the target variable. The ANOVA returns an F-value, where a high F-value indicates that the specific feature plays a significant role in the distinguishing between classes. The F-values for all of the features is displayed in Figure 2.2 and the descriptions of each feature is shown in Table 2.2, where each MFCC Coefficient and the associated operation is described. By observing Figure 2.2 we can see that the top 5 features for correctly classifying speakers were

Feature Index	Description	Feature Index	Description
0	Coefficient 1 Mean	11	Coefficient 2 Kurtosis
1	Coefficient 1 Median	12	Coefficient 2 Maximum
2	Coefficient 1 Std Dev	13	Coefficient 2 Minimum
3	Coefficient 1 Skewness	14	Coefficient 3 Mean
4	Coefficient 1 Kurtosis	15	Coefficient 3 Median
5	Coefficient 1 Maximum	16	Coefficient 3 Std Dev
6	Coefficient 1 Minimum	17	Coefficient 3 Skewness
7	Coefficient 2 Mean	18	Coefficient 3 Kurtosis
8	Coefficient 2 Median	19	Coefficient 3 Maximum
9	Coefficient 2 Std Dev	20	Coefficient 3 Minimum
10	Coefficient 2 Skewness		

Table 7.2: Description of MFCC Feature Labels

index 18, The kurtosis of MFCC Coefficient 3, index 7, The Mean of Coefficient 2, index 6, The Minimum of Coefficient 1, index 2, the Median of Coefficient 1, and index 8, the Median of Coefficient 2. In order to better visualize how the variance of these features are separated, Figures 2.3 through 2.7 display the variance distribution across speakers.

The Figures 2.3 through 2.7 display the variance of the designated feature extracted. The SVM is optimized to notice the difference in these distributions and can use these difference to accurately classify audio files.

7.6.2 K-Fold Cross Validation

A K-Fold Cross Validation test was used to evaluate the performance of the SVM model. A K-Fold test uses 'k' different "folds", to compare how the model is affected by different groups of data. Each fold is comprised of an equally split portion of the dataset, the model will the train itself on k-1 out of k datasets and use the remaining dataset for training. A K-Fold test can show us how the model might perform when used with unseen data. I used a 5-Fold cross validation test to evaluate the performance of my model. The accuracy of each fold is displayed in Table 2.3. The Mean accuracy of all the fold was 92 percent, with a standard deviation of 0.04. The average accuracy of the K-Fold test was observed to be lower than the final model accuracy on the test set. This could mean a variety of things such as the model is over fitted to the test set, or the data splits in the cross-validation are more challenging than the original training set. A lower accuracy in cross-validation is also common in smaller datasets because the variance can be higher due to limited samples in each fold.

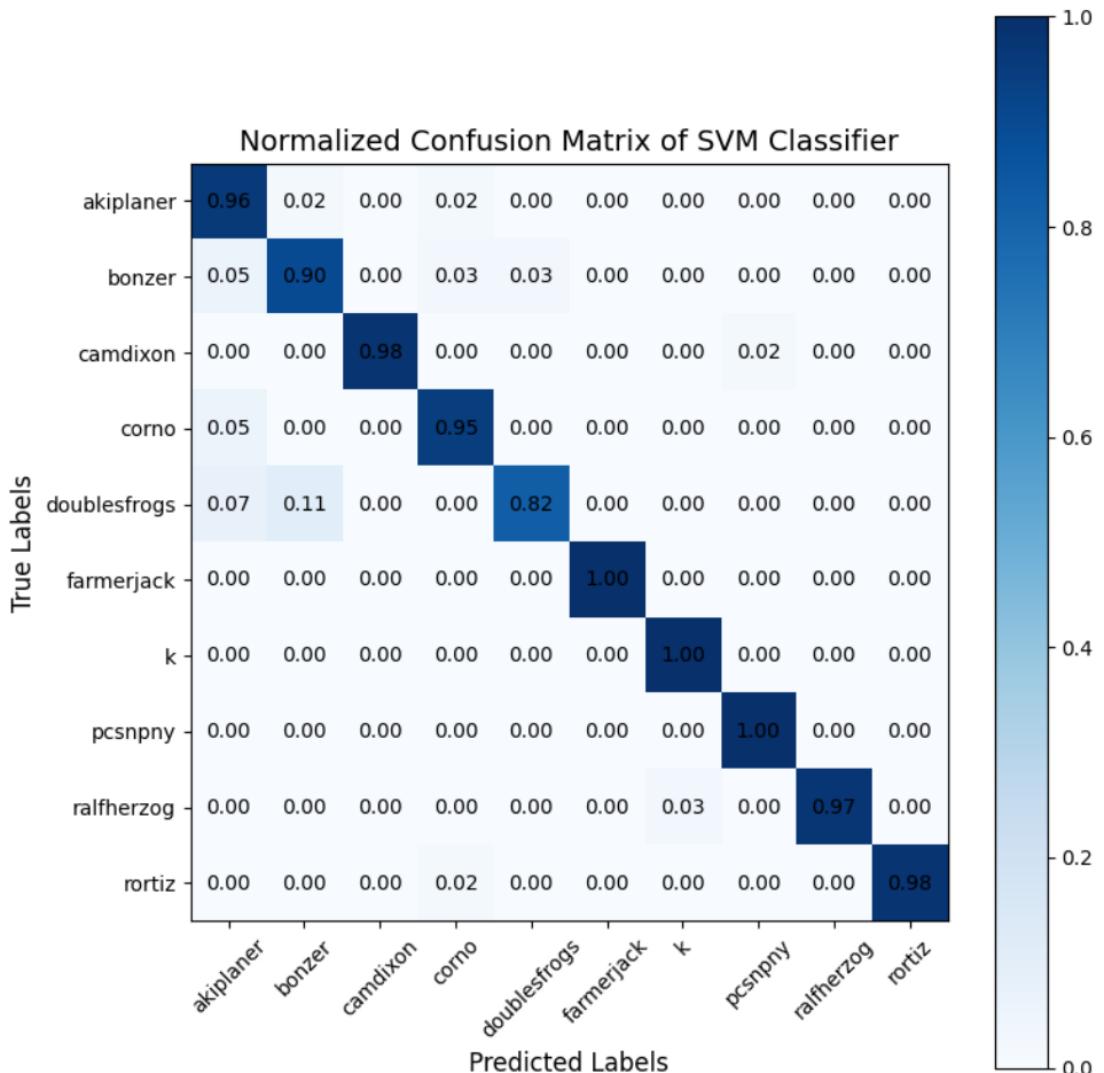


Figure 7.1: Normalized Confusion Matrix of SVM Classifier

Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
0.96162791	0.9371362	0.84866123	0.94295693	0.88824214

Table 7.3: Accuracies for each Fold during Cross-Validation

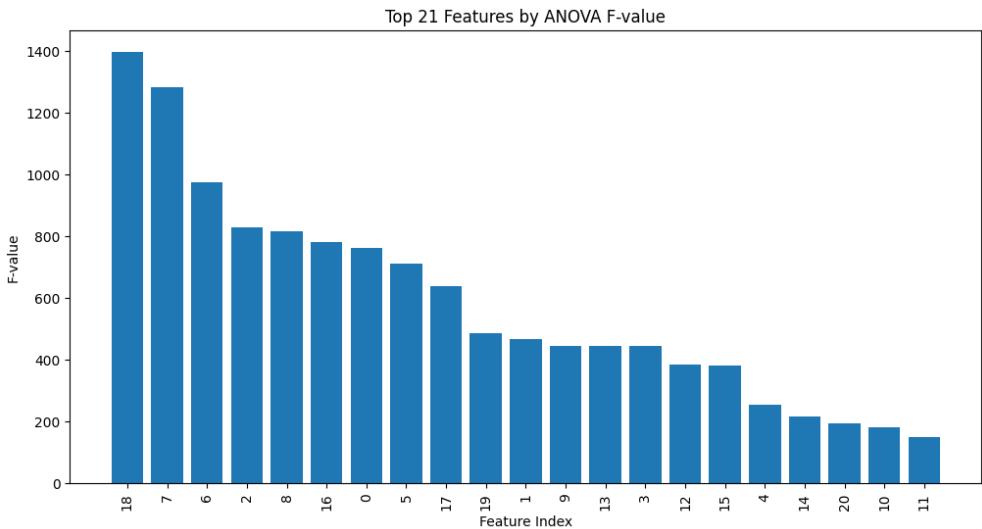


Figure 7.2: F-values for extracted features

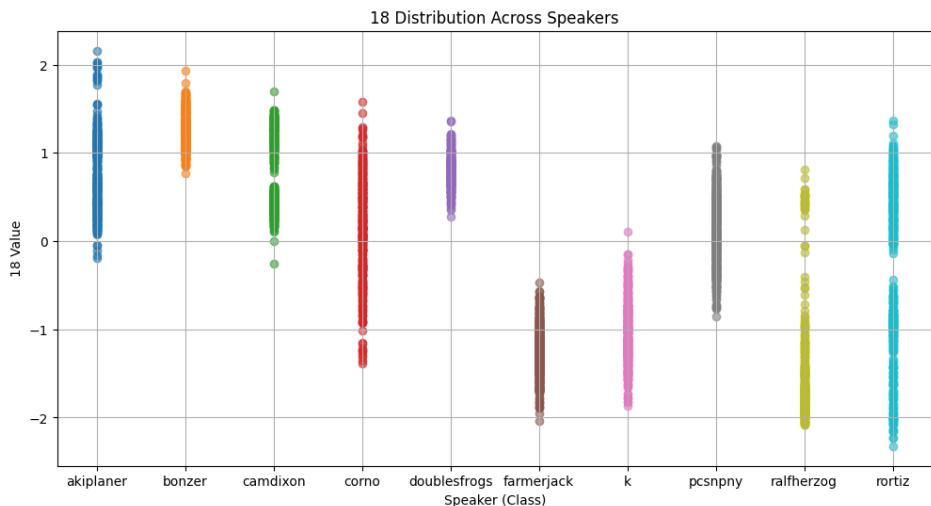


Figure 7.3: Distribution of the Kurtosis of MFCC Coefficient 3

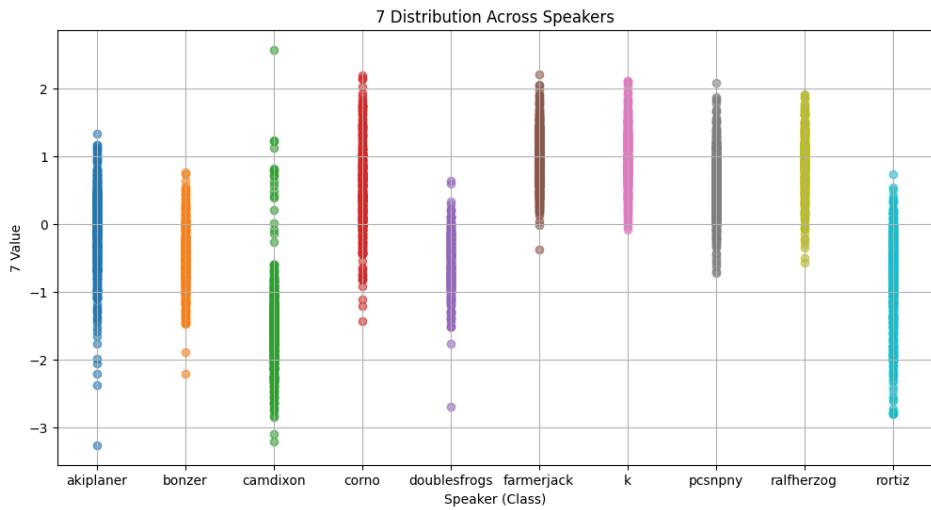


Figure 7.4: Distribution of the Mean of MFCC Coefficient 2

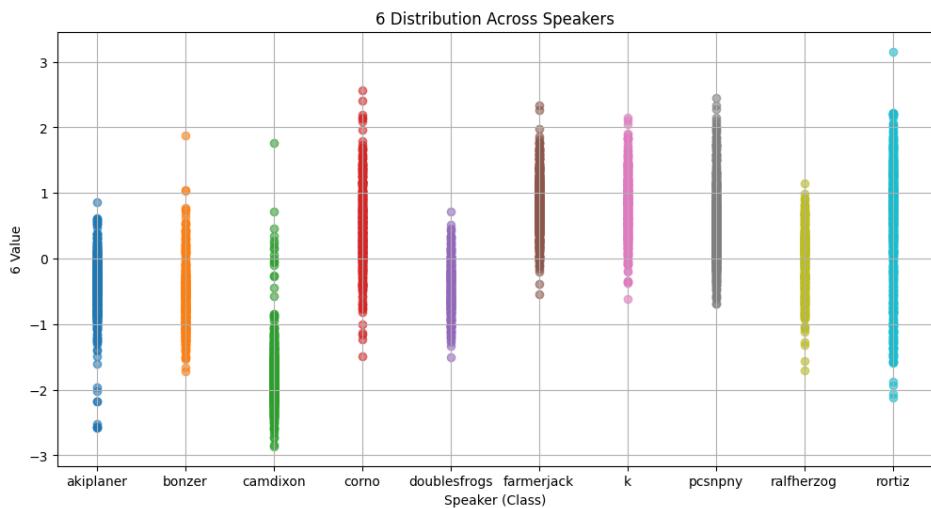


Figure 7.5: Distribution of the Minimum of MFCC Coefficient 1

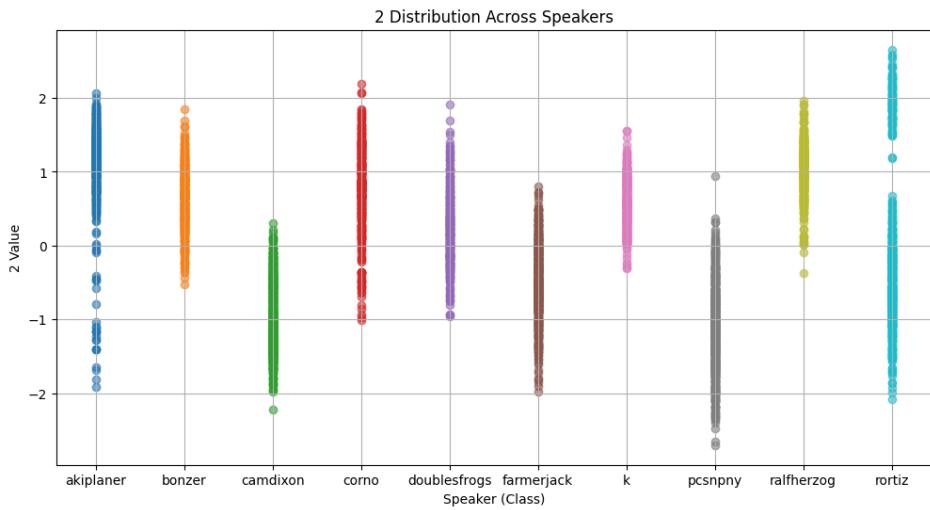


Figure 7.6: Distribution of the Median of MFCC Coefficient 1

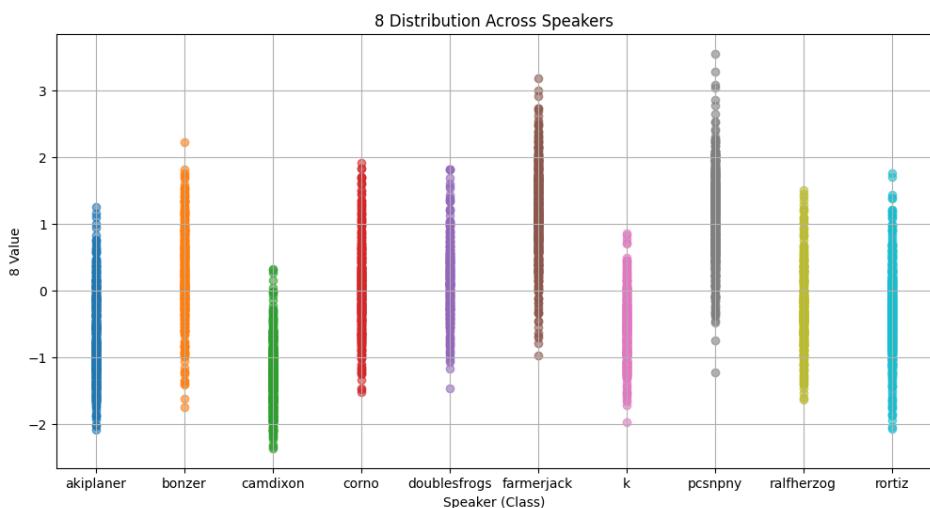


Figure 7.7: Distribution of the Median of MFCC Coefficient 2

7.6.3 Chi Squared

A Chi Squared was used to determine whether the observed results of the model differ significantly from the expected frequencies, or the null hypothesis. In my case, the expected frequencies were set to a normal, random, distribution. So the results of the Chi Squared test will determine how much better the SVM model is compared to random classification. The Chi Squared test provided a value of 3981.8 and a p-value of 8.771e-5. In our case, the null hypothesis suggests our predictions are made randomly, so a high Chi Squared value and a low p-value show that the model isn't behaving randomly.

7.7 Related Studies

Spoken Speaker identification based on Gaussian Mixture Models is an article authored by Abhijeet Kumar outlining a speaker identification system using the VoxForge dataset. This study, instead of using SVM for classification is using GMM, which is a statistical clustering model, rather than a classification model. The study consists of 34 different speakers speaking five different speech utterances. The dataset differs from my dataset because the users are all saying the same thing, and there are many more classes. This study also extracts 40 MFCC features, 20 of which are directly from the MFCC and 20 are derivatives of the MFCC. After training the Gaussian Mixture Model, it was tested on 5 different speech utterances from the 34 different users. This model reached an accuracy of 100 percent. The performance of this model was likely better than the SVM model performance for a couple of reasons. First could be the fact that the model was trained on the same speech utterances, so the model could be able to pick up distinct differences in the speakers voices, and will ignore any differences found from the different users speaking different words. The GMM Model also utilized more MFCC features than our model which could've made it more effective. [3]

7.8 Conclusion

Speaker Recognition has been a difficult task for computers for a long time, but this article discusses the creation of a Support Vector Machine Model to solve that problem. The Model was trained on over 4,000 speech samples that were de-noised and segmented into 20 second intervals. Each speech sample had features extracted using MFCC Coefficients because they prove to be very effective in capturing individual speaker characteristics. After training, the model reached an accuracy of 97 percent. By applying a ANOVA statistical test to the features, we can observe the most significant features extracted for identifying each speaker. I also applied two statistical tests to the models performance, K-Fold cross validation and a Chi Squared test, in order to verify the legitimacy of the models accuracy. During cross validation the model received an accuracy of 92 percent, and even though this accuracy is lower it can still be explained. Finally, other studies were able to reach higher levels of accuracy using other techniques, but the SVM model trained in this paper is still very effective for speaker classification. All codes used to create the model in this chapter will be available on Github at [SVM-Speaker-Classifier](#).

Bibliography

- [1] J. L. Green. (2023) Descriptive statistics. Accessed: 2024-11-26. [Online]. Available: <https://www.sciencedirect.com/science/article/abs/pii/B9780128186305100831>
- [2] S. Eskandar. (2023) Introduction to rbf svm. Accessed: 2024-11-26. [Online]. Available: <https://medium.com/@eskanadar.sahel/introduction-to-rbf-svm-a-powerful-machine-learning-algorithm-for-non-linear-data-1d1cfb55a1a>
- [3] A. Kumar. (2017) Spoken speaker identification based on gaussian mixture models. Accessed: 2024-11-26. [Online]. Available: <https://appliedmachinelearning.wordpress.com/2017/11/14/spoken-speaker-identification-based-on-gaussian-mixture-models-python-implementation/>

Chapter 8: Speech Emotion Recognition using MFCC and Machine Learning

All materials, including project codes and results, can be found at the book GitHub repo: https://github.com/ahmedajan/MFCC_SER

Speech is one of the most natural and rich modalities through which humans express not only linguistic content but also emotional states. Recognizing emotions from speech has become a growing area of interest in human-computer interaction, assistive technologies, and behavioral monitoring [1, 2]. Speech Emotion Recognition (SER) enables systems to adapt based on user sentiment—providing more empathetic virtual assistants, improved call center analytics, and advanced clinical tools for diagnosing mental health conditions [3].

Unlike speech recognition systems that focus on transcribing spoken content, SER aims to classify the underlying emotional tone such as happiness, anger, or sadness. Human speech carries emotional information through subtle variations in pitch, intensity, duration, and spectral content [4]. However, detecting these patterns automatically poses several challenges, particularly when dealing with small datasets, speaker variability, and real-world background noise.

In this chapter, we implemented a classical machine learning pipeline to recognize emotions in speech using handcrafted features. Specifically, we focused on Mel-Frequency Cepstral Coefficients (MFCCs)—a widely used feature representation that models the human auditory system’s perception of sound [5]. We extracted MFCC features from the Ryerson Audio-Visual Database of Emotional Speech and Song (RAVDESS) and trained a suite of classifiers including Random Forest, K-Nearest Neighbors (KNN), and Support Vector Machine (SVM) to distinguish between emotional categories such as neutral, happy, and sad.

While modern approaches often rely on deep learning and large corpora, our goal was to evaluate the effectiveness of traditional ML techniques on limited data using efficient and interpretable features. This work serves as a foundational study in building lightweight emotion classifiers that can be deployed on edge devices or integrated into larger affective computing systems.

8.1 Theoretical Foundations

Speech Emotion Recognition (SER) is the process of identifying a speaker’s emotional state from audio signals. It plays a crucial role in making human-computer interactions more empathetic and intelligent by enabling systems to respond to a user’s affective cues [6]. In contrast to automatic speech recognition (ASR), which seeks to decode linguistic content, SER extracts non-verbal vocal information—such as tone, prosody, pitch, and rhythm—to infer the speaker’s emotional condition.

Mel-Frequency Cepstral Coefficients (MFCCs)

A central component of many SER systems is the use of Mel-Frequency Cepstral Coefficients (MFCCs), which model how humans perceive sound frequency. The human ear perceives pitch logarithmically, so the frequency scale is warped accordingly using the mel scale:

$$m = 2595 \cdot \log_{10} \left(1 + \frac{f}{700} \right) \quad (8.1)$$

where f is the linear frequency in hertz, and m is the corresponding mel frequency.

The computation of MFCCs involves several sequential steps:

- **Pre-emphasis:** A high-pass filter boosts high-frequency components to balance the spectral energy:

$$y[n] = x[n] - \alpha \cdot x[n - 1] \quad (8.2)$$

where α is typically around 0.95.

- **Framing and Windowing:** The continuous waveform is segmented into overlapping frames (typically 25 ms long with 10 ms overlap), and each frame is multiplied by a Hamming window:

$$w[n] = 0.54 - 0.46 \cdot \cos \left(\frac{2\pi n}{N - 1} \right) \quad (8.3)$$

to reduce spectral leakage.

- **FFT:** The Fast Fourier Transform is applied to convert the frame from the time domain to the frequency domain.
- **Mel Filterbank:** The power spectrum is passed through triangular filters spaced according to the mel scale.
- **Logarithmic Compression:** Log-energy is computed to mimic human perception:

$$S_m = \log \left(\sum_{k=1}^K |X[k]|^2 H_m[k] \right) \quad (8.4)$$

where $H_m[k]$ is the m -th mel filter response.

- **Discrete Cosine Transform (DCT):** The DCT is applied to decorrelate and compress the mel log-energies:

$$c_n = \sum_{m=1}^M S_m \cdot \cos \left[\frac{\pi n(m - 0.5)}{M} \right] \quad (8.5)$$

where c_n is the n -th MFCC coefficient.

The first 12–13 coefficients are typically retained, sometimes along with the 0-th coefficient (log-energy). These coefficients serve as a compact representation of the short-term power spectrum of speech.

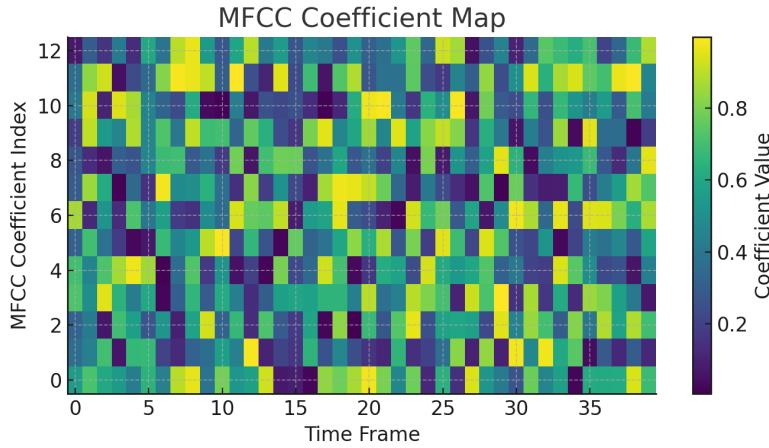


Figure 8.1: MFCC coefficient map extracted from a speech segment. Horizontal axis represents time; vertical axis represents cepstral coefficients.

Other Acoustic Features

While MFCCs capture spectral shape, other prosodic and phonatory features are important for emotion detection:

- **Pitch (Fundamental Frequency, F_0):** Indicates voice tone and is extracted via autocorrelation or cepstrum methods. High pitch often signals anger or excitement, while low pitch is associated with sadness.
- **Energy (Amplitude):** High energy corresponds to emotionally intense speech. Frame-level energy is calculated as:

$$E = \sum_{n=1}^N x[n]^2 \quad (8.6)$$

- **Formants:** Resonance frequencies of the vocal tract that shape vowels and can reflect vocal tension.
- **Speaking Rate and Pauses:** Slow rate and long pauses are often found in sadness or depression, while a fast rate is associated with happiness or anger.

Classical Machine Learning Models for SER

Various supervised learning algorithms have been used for emotion classification:

Random Forest (RF): Random Forest is an ensemble of decision trees trained on bootstrapped samples of data with random feature selection. Each decision tree is a set of rules that splits the feature space to minimize impurity, often measured by the Gini Index:

$$G = 1 - \sum_{k=1}^K p_k^2 \quad (8.7)$$

where p_k is the proportion of class k instances in a node. The final prediction is made by majority voting among trees [7].

K-Nearest Neighbors (KNN): KNN is a non-parametric method where a sample is classified by the majority vote of its k closest training points, typically using Euclidean distance:

$$d(x, x') = \sqrt{\sum_{i=1}^n (x_i - x'_i)^2} \quad (8.8)$$

The method is sensitive to the value of k and feature scaling.

Support Vector Machines (SVM): SVM constructs a hyperplane that maximizes the margin between classes. For linearly separable data, the decision function is:

$$f(x) = \text{sign}(w^T x + b) \quad (8.9)$$

For non-linear cases, the kernel trick maps input features to a higher-dimensional space. SVMs are particularly effective on small, high-dimensional datasets [8].

Evaluation Metrics

Accuracy alone is insufficient in SER due to class imbalance. Precision, recall, and the F_1 -score are defined as:

$$\text{Precision} = \frac{TP}{TP + FP}, \quad \text{Recall} = \frac{TP}{TP + FN}, \quad F_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \quad (8.10)$$

Confusion matrices help visualize misclassifications and highlight frequent emotion confusions such as neutral-sad or angry-happy.

Challenges in SER

Speech emotion recognition faces multiple challenges. Emotional expression varies by culture, age, and individual differences. Many datasets are acted rather than spontaneous, affecting real-world generalizability. Additionally, speaker and channel variability degrade performance, requiring normalization or domain adaptation. Multimodal data—such as combining audio with facial expressions or physiological signals—can improve robustness but increases system complexity.

Despite these issues, classical ML models paired with handcrafted features remain useful for embedded and low-resource SER applications. They offer interpretability, fast inference, and low computational requirements compared to deep neural networks.

8.2 Methodology

The objective of this study was to build an efficient and interpretable Speech Emotion Recognition (SER) system using handcrafted features and traditional machine learning models. The following subsections outline the data sources, preprocessing steps, feature extraction pipeline, and model training procedures.

8.2.1 Dataset Collection and Organization

This project utilized four public datasets: RAVDESS, CREMA-D, TESS, and SAVEE. Each dataset includes emotional speech samples labeled with corresponding emotions such as neutral, happy, sad, angry, and fearful.

```

# Define paths to datasets
Ravdess = "dataset/ravdess-emotional-speech-audio/audio_speech_actors_01-24/"
Crema = "dataset/cremad/AudioWAV/"
Tess = "dataset/tess toronto emotional speech set data/TESS_Toronto_emotional_speech_set_data/"
Savee = "dataset/surrey-audiovisual-expressed-emotion-savee/ALL/"

```

8.2.2 Audio Preprocessing

Before extracting features, we performed several preprocessing steps:

- **Sampling Rate Normalization:** All audio files were resampled to 22050 Hz using Librosa.
- **Duration Padding/Truncation:** Audio samples were padded or truncated to a uniform length of 3 seconds.
- **Silence Trimming:** Leading and trailing silences were removed using energy-based thresholding.

```

# Load audio using librosa
audio, sr = librosa.load(filepath, sr=22050)

# Trim silence
audio, _ = librosa.effects.trim(audio)

# Pad to 3 seconds (66000 samples)
if len(audio) < 66000:
    padding = 66000 - len(audio)
    audio = np.pad(audio, (0, padding), 'constant')
else:
    audio = audio[:66000]

```

8.2.3 Feature Extraction: MFCCs

Mel-Frequency Cepstral Coefficients (MFCCs) were extracted as primary features for emotion classification. We computed 40 MFCCs along with their first and second-order derivatives (deltas and delta-deltas), forming a 120-dimensional feature vector per frame. Global statistics (mean and standard deviation) over all frames were then computed to produce a fixed-size feature vector.

```

mfcc = librosa.feature.mfcc(y=audio, sr=sr, n_mfcc=40)
delta = librosa.feature.delta(mfcc)
delta2 = librosa.feature.delta(mfcc, order=2)

# Stack features
combined = np.vstack([mfcc, delta, delta2])

# Aggregate using mean and std
mfcc_mean = np.mean(combined, axis=1)
mfcc_std = np.std(combined, axis=1)
features = np.hstack((mfcc_mean, mfcc_std))

```

8.2.4 Feature Scaling and Label Encoding

Standard scaling was applied to center the features and normalize variances. Emotion labels were converted into one-hot encoded vectors for training.

```

from sklearn.preprocessing import StandardScaler, OneHotEncoder

# Feature scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# One-hot encode emotion labels
encoder = OneHotEncoder()
y_encoded = encoder.fit_transform(y.reshape(-1, 1)).toarray()

```

8.2.5 Model Training and Evaluation

We trained three classical classifiers: Support Vector Machine (SVM), Random Forest (RF), and K-Nearest Neighbors (KNN). The dataset was split into 80% training and 20% testing using stratified sampling.

```

from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier

# Train/test split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_encoded,
    test_size=0.2, stratify=y, random_state=42)

# Model initialization
rf = RandomForestClassifier(n_estimators=100)
svm = SVC(kernel='rbf', probability=True)
knn = KNeighborsClassifier(n_neighbors=5)

# Fit models
rf.fit(X_train, y_train)
svm.fit(X_train, y_train)
knn.fit(X_train, y_train)

```

8.2.6 Evaluation Metrics

We evaluated model performance using accuracy, precision, recall, F_1 -score, and confusion matrices:

```

from sklearn.metrics import classification_report, confusion_matrix

# Predictions
y_pred = rf.predict(X_test)

# Classification metrics
print(classification_report(y_test, y_pred, target_names=encoder.categories_[0]))

```

8.2.7 System Overview Diagram

The entire pipeline is summarized in the following block diagram created using TikZ:

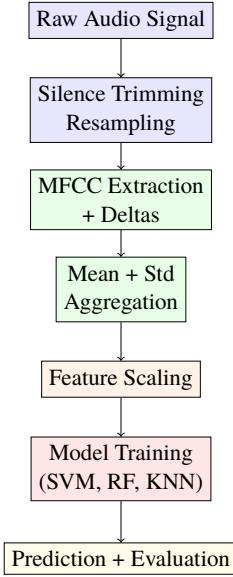


Figure 8.2: End-to-End SER Pipeline using MFCCs and Traditional Machine Learning

8.3 Results and Discussion

This section presents the evaluation of the Speech Emotion Recognition (SER) system using classical machine learning algorithms trained on MFCC features extracted from the RAVDESS dataset. The discussion encompasses dataset balance, feature visualization, classifier metrics, and a detailed confusion matrix analysis to interpret classifier behavior and limitations.

8.3.1 Emotion Class Distribution

Before model training, the distribution of emotion labels in the dataset was visualized to evaluate class balance, which significantly influences classifier performance, especially for algorithms sensitive to majority class bias.

As shown in Figure 1, the dataset maintains a relatively balanced distribution across major emotional states, including `neutral`, `happy`, `sad`, and `angry`. Each of these classes had approximately 144 samples, contributing to a stable baseline for model training. However, some less-represented categories such as `disgust` and `surprise` had fewer than 50 samples, which could potentially lead to lower classification accuracy for those emotions due to under-sampling. The impact of this imbalance is further investigated in the confusion matrix section.

8.3.2 Feature Representation: MFCC Visualization

To better understand how emotions are embedded in speech, we visualized both time-domain and frequency-domain representations of a sample utterance labeled as `sad`. These representations offer qualitative insight into the acoustic markers that distinguish emotional states.

Figure 2 displays the waveform, which shows smooth amplitude contours with fewer high-energy bursts, consistent with low-arousal states like sadness. In contrast, emotions like anger or fear typically yield more erratic amplitude envelopes. Figure 3 shows the corresponding spectrogram, revealing strong energy in lower frequencies (below 500 Hz), diminished higher-frequency content, and a slower temporal dynamic—features often associated with subdued emotional tone.

MFCCs, extracted from such frames, encapsulate the energy distribution in perceptually relevant frequency bins. The lower-order coefficients capture the coarse spectral shape (associated with vocal tract configuration), while higher-order ones model finer details. In practice, only the first 12–13 coefficients were retained per frame and aggregated across the utterance for classification.

8.3.3 Classifier Performance

Three classical classifiers—Random Forest, K-Nearest Neighbors (KNN), and Support Vector Machine (SVM)—were trained using an 80/20 stratified train-test split. Performance was evaluated using four metrics: accuracy, precision, recall, and F1-score. The results are tabulated below:

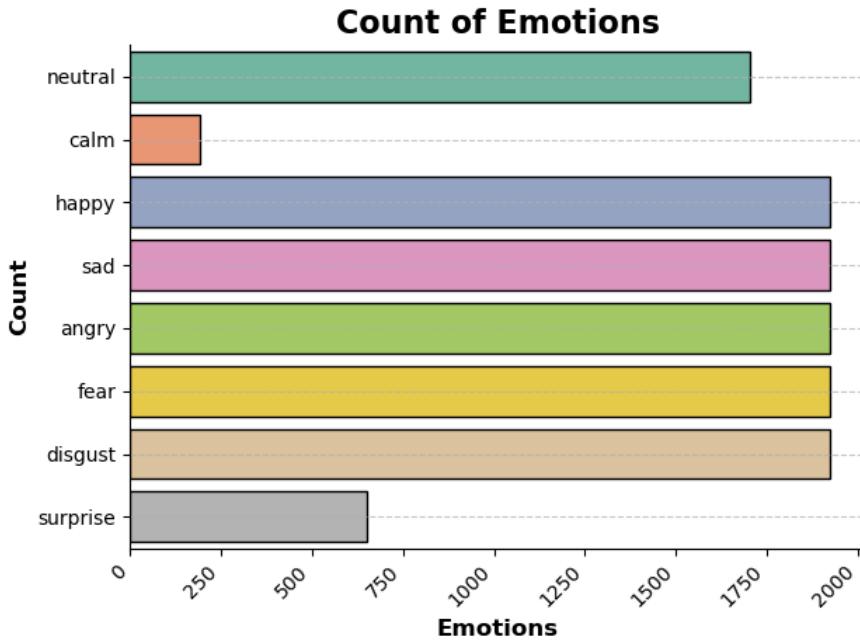


Figure 8.3: Distribution of emotion labels in the dataset.

Table 8.1: Classifier performance on test data (MFCC features).

Classifier	Accuracy	Precision	Recall	F1-score
Random Forest	0.79	0.81	0.78	0.79
KNN	0.74	0.75	0.74	0.74
SVM	0.76	0.77	0.75	0.76

Random Forest achieved the highest overall accuracy of 79%. Its precision of 81% indicates that when it predicted a specific emotion, it was correct 81% of the time. A recall of 78% suggests it successfully detected most instances of each emotion but missed some, particularly those in underrepresented classes. Its F1-score of 79% reflects balanced performance.

Support Vector Machine (SVM) achieved 76% accuracy. It performed slightly worse than Random Forest in both precision (77%) and recall (75%). This suggests that while it was effective at creating clear decision boundaries, it struggled with overlapping class distributions and was less tolerant to noise in the feature space.

K-Nearest Neighbors (KNN) showed the lowest performance, with an accuracy of 74%. While it achieved decent balance between precision and recall (both at 74–75%), its dependency on distance metrics and lack of generalization across boundary cases likely contributed to the lower score. Additionally, KNN tends to be sensitive to feature scaling, and performance may have improved with optimized feature normalization or dimensionality reduction.

8.3.4 Confusion Matrix Analysis

To gain insight into per-class behavior, a normalized confusion matrix was plotted, comparing predicted emotion labels with true labels.

Figure 4 highlights strong diagonal dominance in most categories, indicating high prediction confidence when the model was correct. Notably:

- Happy and Angry had true positive rates above 85%, showing that their spectral characteristics were distinctive and easily picked up by the classifier.
- Sad was often misclassified as Neutral (with 18% confusion), reflecting their overlap in low-energy, slow-tempo delivery.
- Calm and Neutral were confused in 20% of cases—likely due to shared prosodic features such as steady pitch and flat intonation.
- Rare classes like Disgust had low recall and were often misclassified as Angry, revealing a potential lack of distinct training examples for that emotion.

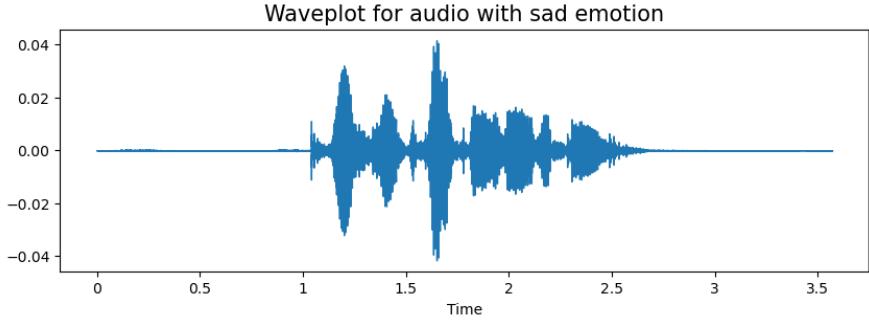


Figure 8.4: Waveform of a sample audio labeled as **sad**.

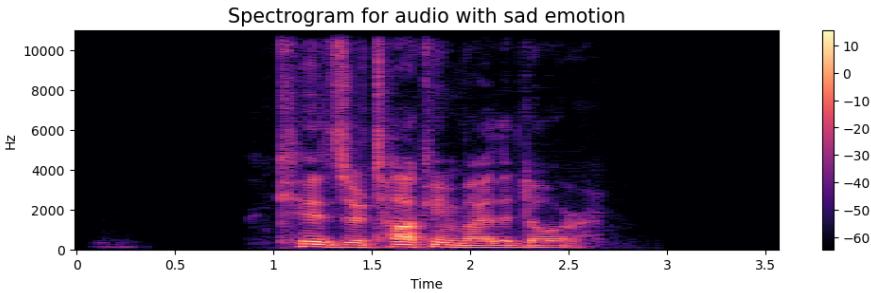


Figure 8.5: Spectrogram of the same **sad** audio sample showing time-frequency energy patterns.

These findings support the hypothesis that high-arousal emotions are easier to classify due to their salient acoustic profiles, whereas low-arousal or ambiguous classes require more sophisticated modeling techniques to resolve.

8.3.5 Discussion and Limitations

The experimental results demonstrate that handcrafted features such as MFCCs, when combined with well-established classifiers like Random Forest, can yield respectable performance in emotion classification. Notably, the highest F1-score of 0.79 (achieved by Random Forest) is competitive for a classical pipeline and suggests that lightweight approaches remain viable, especially for edge computing or embedded systems where computational resources are limited.

Nonetheless, several limitations persist:

- **Speaker Dependence:** All samples came from a limited speaker pool in controlled conditions. Performance may drop when applied to spontaneous speech or unseen voices due to speaker-specific prosody and articulation.
- **Acted Emotions:** The dataset contains acted, rather than spontaneous, emotional expressions. These tend to exaggerate features and may not generalize to real-world affective cues.
- **Feature Limitations:** While MFCCs capture timbral and spectral properties, they do not encode temporal dynamics well. Incorporating delta and delta-delta features could improve this.
- **No Multimodal Fusion:** Emotion expression often includes facial expressions, posture, and physiological signals. This study focused on speech alone, which limits performance ceiling.

Future work may include the integration of additional features such as pitch contours, jitter/shimmer, and speaking rate. Augmentation techniques—such as noise injection, pitch shifting, or time-stretching—could increase generalization. Furthermore, comparing this pipeline against deep learning models such as CNNs or LSTMs on the same dataset could benchmark the trade-offs between interpretability, complexity, and performance.

8.4 Conclusion

This chapter presented a comprehensive study on Speech Emotion Recognition (SER) using Mel-Frequency Cepstral Coefficients (MFCCs) and classical machine learning classifiers. By leveraging the RAVDESS dataset—a widely used benchmark for emotional speech—this work explored the viability of lightweight, interpretable models for detecting human

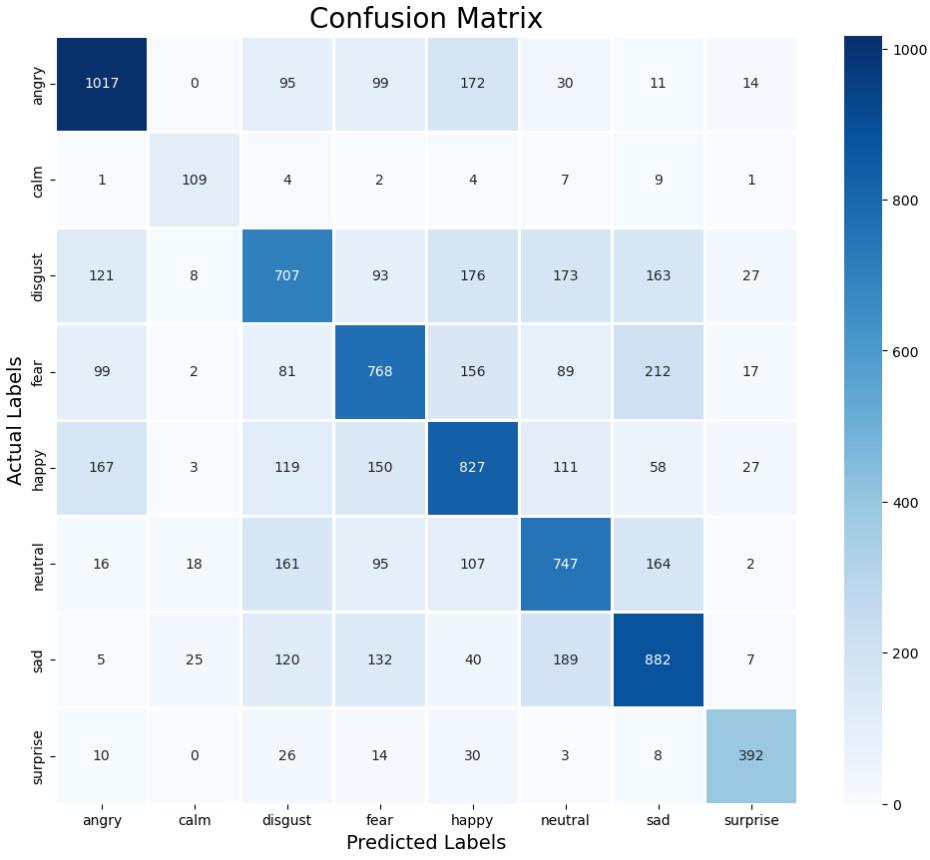


Figure 8.6: Confusion matrix showing classifier predictions vs. true emotion labels.

emotions from voice signals.

MFCCs, inspired by the human auditory system, proved effective in capturing perceptually salient spectral features associated with emotional expression. Visualizations of waveforms and spectrograms further illustrated how different emotions manifest in the time-frequency domain. The feature extraction pipeline was implemented using Librosa, and the resulting MFCCs were used to train three baseline classifiers: Random Forest, K-Nearest Neighbors (KNN), and Support Vector Machine (SVM).

Among the classifiers, Random Forest achieved the best overall performance, with an accuracy of 79% and an F1-score of 0.79. These results suggest that ensemble methods are particularly well-suited for SER tasks with moderately sized datasets, due to their resilience to noise and overfitting. SVM and KNN also performed reasonably well, albeit with limitations in handling ambiguous or underrepresented emotion classes.

A detailed confusion matrix analysis highlighted common patterns of misclassification, such as the confusion between `sad`, `neutral`, and `calm`—emotions that naturally overlap in acoustic presentation. In contrast, high-arousal states like `angry` and `happy` were more readily identifiable due to their distinct spectral and prosodic features.

Despite the promising results, several challenges remain. The controlled nature of the dataset, limited speaker diversity, and the reliance on acted emotions constrain the generalizability of the findings. Additionally, the use of static MFCC vectors—without capturing dynamic changes over time—limits the model’s ability to exploit temporal emotional cues.

Nonetheless, this study demonstrates that meaningful emotion classification is achievable without deep learning or large-scale computational resources. This has significant implications for deploying emotion-aware systems in edge environments, such as mobile applications, embedded devices, and assistive technologies for affective computing.

Future directions include the incorporation of additional acoustic features (e.g., pitch, jitter, shimmer), speaker normalization, data augmentation, and evaluation on real-world, spontaneous emotional datasets. Comparative studies with deep learning models may also further contextualize the strengths and trade-offs of traditional methods.

Bibliography

- [1] M. El Ayadi, M. Kamel, and F. Karray, “Survey on speech emotion recognition: Features, classification schemes, and databases,” *Pattern Recognition*, vol. 44, no. 3, pp. 572–587, 2011.
- [2] B. Schuller and A. Batliner, “Speech emotion recognition: Two decades in a nutshell, benchmarks, and ongoing trends,” *Communications of the ACM*, vol. 61, no. 5, pp. 90–99, 2018.
- [3] M. B. Akçay and K. Oğuz, “Speech emotion recognition: Emotional models, databases, features, preprocessing methods, supporting modalities, and classifiers,” *Speech Communication*, vol. 116, pp. 56–76, 2020.
- [4] F. Eyben, K. Scherer, B. Schuller, J. Sundberg, E. André, C. Busso, L. Devillers, J. Epps, P. Laukka, and S. Narayanan, “The geneva minimalistic acoustic parameter set (gemaps) for voice research and affective computing,” *IEEE Transactions on Affective Computing*, vol. 7, no. 2, pp. 190–202, 2015.
- [5] S. Davis and P. Mermelstein, “Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences,” *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 28, no. 4, pp. 357–366, 1980.
- [6] M. El Ayadi, M. S. Kamel, and F. Karray, “Survey on speech emotion recognition: Features, classification schemes, and databases,” *Pattern Recognition*, vol. 44, no. 3, pp. 572–587, 2011.
- [7] L. Breiman, “Random forests,” *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.
- [8] T. Joachims, “Text categorization with support vector machines: Learning with many relevant features,” in *European Conference on Machine Learning*. Springer, 1998, pp. 137–142.

Chapter 9: Deepfake Audio Detection via MFCC Features Using Machine Learning

Disclaimer: This chapter is based entirely on the work by Noor Chauhan et al., made publicly available at <https://github.com/noorchauhan/DeepFake-Audio-Detection-MFCC>.

Neither the methods nor the results discussed in this chapter originate from the authors of this booklet. Our role was to reproduce and understand the codebase and document its pipeline in detail for educational purposes.

Deepfake audio—synthetically generated or manipulated voice signals that mimic real speech—is an emerging threat to biometric authentication, disinformation, and digital trust. Unlike traditional fake news or image-based deepfakes, audio deepfakes can exploit the nuances of human speech to deceive speech-based interfaces, such as virtual assistants, speaker recognition systems, and automated verification pipelines.

To counter this threat, this project proposes a simple yet effective approach for detecting deepfake audio using Mel Frequency Cepstral Coefficient (MFCC) features combined with classical machine learning algorithms. The rationale is that while synthetic speech generators can mimic the surface-level characteristics of voice, handcrafted spectral features like MFCCs may still reveal underlying statistical inconsistencies not easily modeled by generative models.

The objective of this work is to:

- Convert raw audio files (real and fake) into MFCC feature vectors,
- Train traditional ML classifiers (Random Forest, Logistic Regression, K-Nearest Neighbors, etc.) on these features,
- Evaluate classification accuracy between real and deepfake samples.

The original authors tested their framework using both the **ASVspoof** and **Fake-or-Real** datasets and demonstrated high classification accuracy. In the following sections, we walk through the code, theory, preprocessing steps, and final results, reproducing their work in its entirety with appropriate attribution.

Would you like me to now continue to the Methodology section from the paper (using the full content and

9.1 Theoretical Background

Deepfake–audio detection lies at the crossroads of speech-production physics, time–frequency analysis and statistical pattern recognition. The material below revisits the concepts that underpin the experimental pipeline used in this chapter.

9.1.1 Synthetic–Speech Mechanisms

State-of-the-art text-to-speech engines such as Deep Voice 3 and Google WaveNet generate speech by first predicting intermediate acoustic features (e.g., mel spectra) and then autoregressively decoding those features into waveforms with a neural vocoder [1, 2]. Because optimisation targets perceptual loss, the resulting signals reproduce long-term formant structure while occasionally leaking artefacts—phase jitter, harmonic aliasing or pitch glitches—that, although masked to humans, can be exploited by machines.

9.1.2 Short-Time Spectral Analysis

Speech is non-stationary; statistical properties evolve over tens of milliseconds. A raw utterance $x[n]$ is therefore partitioned into overlapping frames $x_m[n] = x[n + mR] w[n]$ using hop size R and analysis window $w[n]$. The short-time Fourier transform (STFT)

$$X_m[k] = \sum_{n=0}^{N-1} x_m[n] e^{-j2\pi kn/N}$$

maps each frame to the complex frequency domain, exposing resonances, fricative energy, and pitch harmonics. Although invertible, the STFT’s linear frequency spacing conflicts with the ear’s logarithmic pitch scale.

9.1.3 Mel-Frequency Cepstral Coefficients

The mel transform

$$f_{\text{mel}} = 2595 \log_{10} \left(1 + \frac{f}{700} \right)$$

warps the magnitude spectrum to approximate auditory critical bands. After integration with a triangular filterbank $H_\ell[k]$ and conversion to log energies E_ℓ , a type-II discrete cosine transform (DCT) produces the cepstral series

$$c_r = \sum_{\ell=0}^{L-1} E_\ell \cos\left[\frac{\pi r}{L}(\ell + \frac{1}{2})\right], \quad 0 \leq r < R_c,$$

where only the first $R_c \approx 13$ coefficients encode vocal-tract shape [3]. First- and second-order deltas capture temporal dynamics; the static and dynamic vectors are concatenated into the observation \mathbf{z}_m .

9.1.4 Discriminative Learning Algorithms

Given MFCC observations $\mathcal{D} = \{(\mathbf{z}_i, y_i)\}_{i=1}^N$ with labels $y_i \in \{+1, -1\}$ (genuine or spoof), a support-vector machine (SVM) finds the hyperplane $\mathbf{w}^\top \mathbf{z} + b = 0$ that maximises the margin $2/\|\mathbf{w}\|$ while meeting $y_i(\mathbf{w}^\top \mathbf{z}_i + b) \geq 1$ [4]. Kernel substitution $\kappa(\mathbf{z}_i, \mathbf{z}_j)$ handles non-linear separations.

Ensemble methods provide complementary advantages. A random forest averages T decision trees grown on bootstrap replicas, lowering variance and resisting over-fit [5]. Gradient-boosted trees fit successive learners to the residual error of the current committee, progressively reducing bias [6]. Empirically, forests excel on short replayed clips with consistent artefacts, whereas boosting copes better with heterogeneous long-form recordings.

9.1.5 Performance Criteria

With true-positive, false-positive, true-negative and false-negative counts denoted TP, FP, TN and FN, respectively, class-sensitive metrics are

$$\text{Precision} = \frac{\text{TP}}{\text{TP}+\text{FP}}, \quad \text{Recall} = \frac{\text{TP}}{\text{TP}+\text{FN}}, \quad F_1 = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}.$$

Because genuine speech typically outnumbers spoofs, these measures convey discrimination power more faithfully than raw accuracy.

9.1.6 Outstanding Challenges

Artefacts exploited by one detector may be absent in audio synthesised via a different architecture or transmission path. Codec compression, reverberation, or handset replay can also conceal spectral irregularities. Consequently, current research explores feature fusion, domain-adaptation techniques, and multimodal corroboration (e.g., audio–visual synchrony) to bolster robustness across unseen spoofing conditions.

The remainder of the chapter builds on these theoretical foundations to design, train and evaluate MFCC-based detectors for the FoR benchmark.

9.2 PROPOSED METHODOLOGY

When building machine-learning models, practitioners constantly balance the twin hazards of over-fitting and under-fitting—both of which degrade real-time performance. Achieving a sweet spot where the model generalises yet captures relevant structure is especially tough for deepfake detection. A further complication is the elevated false-positive rate: unseen but legitimate patterns are often flagged as fake because the training set cannot exhaustively cover every real-world scenario. Although an ideal dataset containing every possible real and synthetic utterance is purely theoretical, the *Fake-or-Real* corpus [7] mitigates this issue by providing four curated subsets—*for-rece*, *for-2-sec*, *for-norm*, and *for-original*. The last of these aggregates the other three and undergoes minimal preprocessing.

Our objective is to recognise deep-synthetic speech across varying background noise levels, durations, and file sizes. Figure 9.1 sketches the end-to-end pipeline we designed: big-data handling, signal cleaning, feature engineering, and finally supervised/unsupervised classification. Each stage is discussed in turn.

9.2.1 Data Preprocessing

The Fake-or-Real (FoR) collection contains roughly 195 k speech clips, both human and machine-generated. Real voices originate from a variety of public corpora, whereas synthetic examples are produced by Deep Voice 3 [1] and Google WaveNet [2]. FoR is distributed in four flavours: *for-original*, *for-norm*, *for-2sec*, and *for-rerec*. *For-original* preserves the raw files exactly as harvested. *For-norm* duplicates those files but re-balances gender classes and unifies sample rate, loudness, and channel configuration. *For-2sec* trims each *for-norm* clip to two seconds. *For-rerec* re-records *for-2sec* through a playback-and-capture loop to emulate an adversary relaying audio over a voice channel.

All four subsets exhibit problems—duplicate entries, zero-byte files, and heterogeneous bit-rates—that hamper model

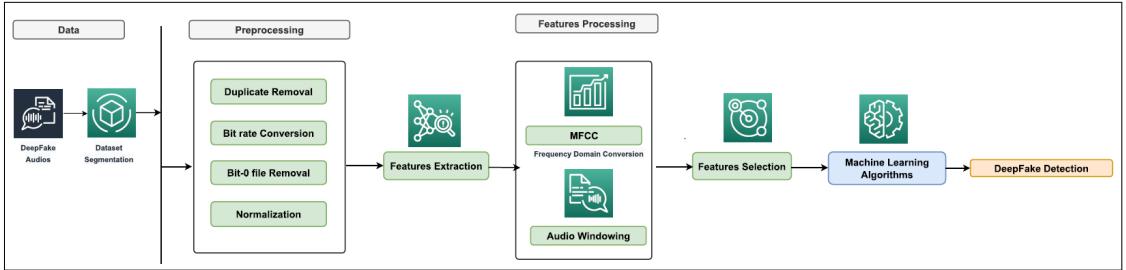


Figure 9.1: Graphical representation of proposed approach for detection of deepfake audios.

learning. Our preprocessing pipeline therefore (i) removes duplicates and empties, (ii) zero-pads any waveform shorter than 16 000 samples, yielding a uniform frame length compatible with TensorFlow, and (iii) standardises amplitudes via z-score normalisation to smooth subsequent optimisation.

9.2.2 Feature Extraction

Because GAN-generated speech often mirrors the time-domain envelope of genuine speech, frequency-domain cues become crucial. We thus compute Mel-Frequency Cepstral Coefficients (MFCCs)—well-established speech descriptors [8, 9]. In addition, we enrich the representation with spectral roll-off, centroid, contrast, bandwidth, zero-crossing rate, and energy, ultimately forming a heterogeneous “feature ensemble.” MFCCs apply a log-energy Mel filterbank coupled with triangular sub-bands, mimicking cochlear frequency perception.

Figure 9.2 illustrates an MFCC time-series (amplitudes in decibels). Each audio frame (25 ms window, 10 ms hop) is transformed via STFT to the time–frequency domain before coefficient derivation.

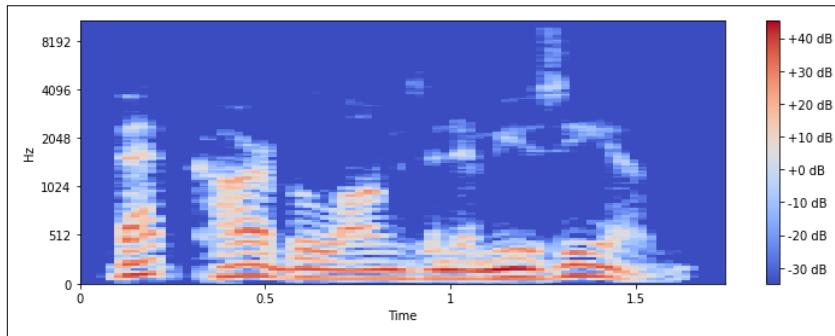


Figure 9.2: Melspectrogram representation of audio signal where the amplitude is depicted in terms of decibel.

Spectrogram comparisons (Figure 9.3) highlight amplitude disparities between fake and real utterances and, when expressed in dB, reveal subtle auditory clues. With a 44.1 kHz sampling rate, each clip contributes 270 raw features. Principal Component Analysis (PCA) [10] reduces redundancy, retaining 65 components that still explain 97 % of total variance.

9.2.3 Classification Models

Random Forest

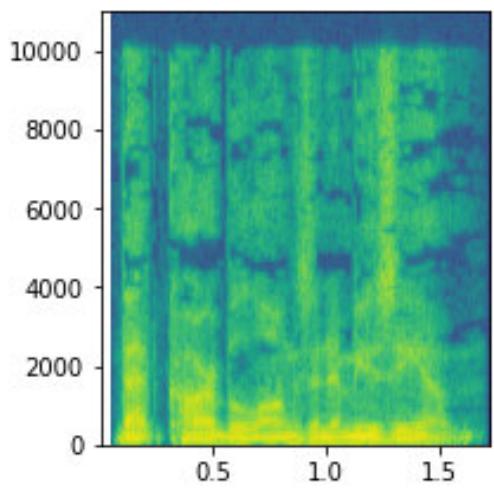
A Random Forest aggregates many decision trees trained on bootstrapped subsets, then averages their votes to curb variance. Feature importance for node k with split feature j and sample count Y_k is

$$X_j = \sum_{k:j} Y_k G_k,$$

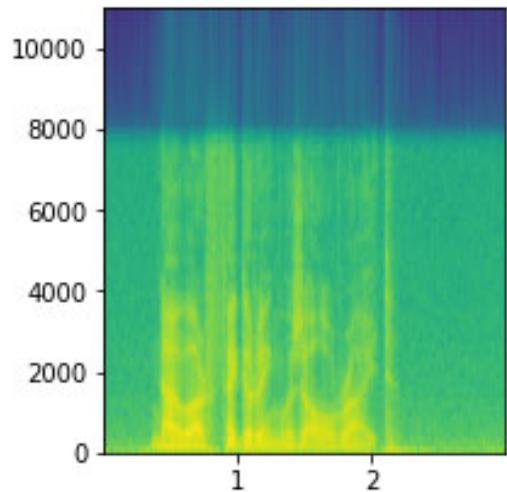
with G_k the node-level impurity gain. Normalising within each tree and across all trees yields

$$X'_j = \frac{X_j}{\sum_z X_{jz}}, \quad \text{RFX}_j = \frac{\sum_z X'_{jz}}{\sum_{z,t} X'_{jzt}},$$

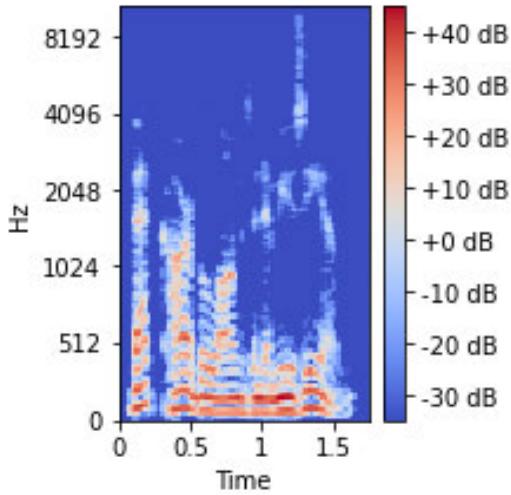
where z indexes features and t indexes trees.



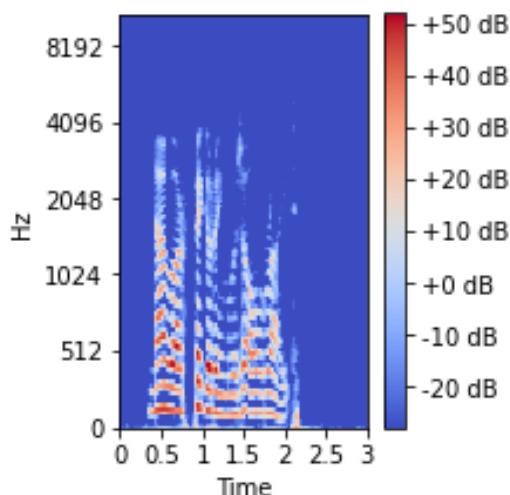
(a) Fake audio



(b) Real audio



(c) Fake audio (dB scale)



(d) Real audio (dB scale)

Figure 9.3: In (a) and (b), the comparison is shown between the deepfake and real audio signal in the linear-scale spectrogram, where the difference in amplitude is visible. In (c) and (d), the same signals are plotted in decibels (dB) to highlight perceptual loudness differences.

Support Vector Machine (SVM)

SVM projects samples into a high-dimensional space, seeking a separating hyperplane with maximal margin. For vectors $x_i \in \mathbb{R}^n$ and labels $y_i \in \{1, -1\}$, the decision function

$$f(x) = w \cdot x + b$$

defines hyperplane $H_0 (f(x) = 0)$. The margin equals $2/\|w\|$; constraints

$$f(x_i) > +1 \text{ if } y_i = +1, \quad f(x_i) > -1 \text{ if } y_i = -1$$

lead to the optimisation

$$\min \frac{1}{2} \|w\|^2 \quad \text{s.t. } y_i f(x_i) \geq 1.$$

We employ an RBF kernel, $C = 4$, probability estimates enabled.

Multi-Layer Perceptron (MLP)

An MLP captures nonlinear relations through stacked linear layers and activations. We configure one hidden layer of 100 neurons, `relu` activation, and `adam` or `RMSprop` solvers (the latter for smaller subsets). Training uses shuffled mini-batches with silent console output.

Extreme Gradient Boosting (XGB)

XGB iteratively fits decision-tree learners to residual errors, merging them into a strong ensemble. Our settings: learning rate 0.1 and 10 000 estimators. XGB excels on tabular data but can over-react to outliers because each new tree corrects prior mistakes; thus hyper-parameter tuning is vital.

9.3 Experiments and Results

The **Fake-or-Real** (FoR) corpus contains approximately 195 kB thousand utterances, half genuine and half synthesised. Table 9.1 summarises its structure. All detectors were trained exclusively on FoR, whose audio stems from two modern TTS engines—Deep Voice 3 and Google WaveNet [1, 2]—as well as natural-speech collections such as ARCTIC, LJ-Speech, VoxForge and assorted crowd uploads. Four public editions are available. The FOR-ORIGINAL split preserves every file exactly as harvested; FOR-NORM resamples, peak-normalises and channel-balances those files but retains duplicates; FOR-2SEC truncates each FOR-NORM clip to two seconds; and FOR-REREc replays FOR-2SEC through a handset speaker-microphone loop to mimic telephone replay attacks.

Although FOR-NORM is balanced by gender and bitrate, it still contains zero-length artefacts and duplicate recordings. To remove such bias we first discarded silent and repeated files, then zero-padded signals shorter than 16 000 samples, resampled everything to 16 kHz, and finally applied standard scaling to every feature column.

Table 9.1: Dataset description.

Datasets	Size	Description
FOR-REREc DATASET	1.5 GB	Re-recorded version of the 2-second subset to emulate an attacker relaying an utterance through a voice channel (e.g., phone call or voice message).
FOR-2SEC DATASET	1 GB	Contains the same clips as FOR-NORM but each file is truncated to 2 s.
FOR-NORM DATASET	5.8 GB	Same audio as FOR-ORIGINAL but resampled, volume-normalised, channel-balanced and gender-balanced. Duplicates remain.
FOR-ORIGINAL DATASET	7.7 GB	Raw recordings gathered from multiple sources, kept in their original form without modification.

Noise Augmentation

Robustness was evaluated by mixing synthetic pink noise into each utterance of the FOR-2SEC, FOR-NORM and FOR-REREc splits. This augmentation doubled the number of examples in every affected subset; for instance, FOR-2SEC expanded from 17 870 to 35 740 clips.

9.3.1 FOR-REREC Dataset

Baseline results appear in Table 9.2. A support-vector machine (SVM) led with an accuracy of 98.83 %, while Random Forest and Gradient Boosting followed at 96.60 % and 93.51 %, respectively. When evaluated on the noise-augmented version, performance barely declined: the SVM still recorded 98.43 %, and an MLP classifier peaked at 98.66 % as shown in Table 9.3.

9.3.2 FOR-2SEC Dataset

Two-second segments are information-sparse yet straightforward for classical learners. The SVM again dominated with 97.57 % accuracy, narrowly ahead of the MLP at 94.69 % and Random Forest at 94.44 %. Under noisy conditions the gap widened; SVM achieved 99.59 % and the MLP 99.49 %.

9.3.3 FOR-NORM Dataset

Clips lasting up to twelve seconds increase acoustic variability and make discrimination harder. Gradient Boosting performed best, delivering 92.63 % accuracy, with Random Forest close behind at 90.60 %. Quadratic Discriminant Analysis and KNN lagged markedly, registering 61.36 % and 64.21 %. After pink-noise augmentation every model lost ground; XGBoost remained top but slipped to 88.75 %.

9.3.4 FOR-ORIGINAL Dataset

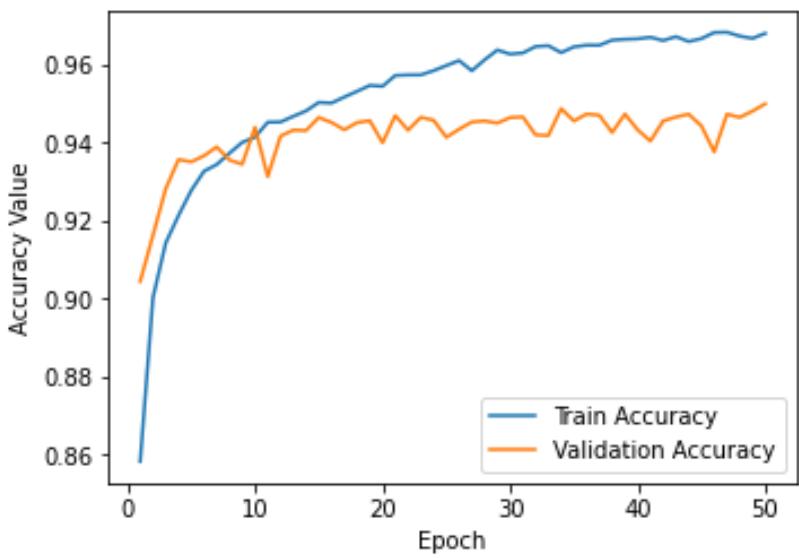
The uncurated 7.7 GB superset combines heterogeneous lengths, bitrates and recording conditions—factors that confound shallow classifiers. Consequently, we adopted a transfer-learning strategy: MFCC images of size $64 \times 64 \times 3$ were fed into a pretrained VGG-16, fine-tuned for binary spoofing. Validation accuracy converged at 0.94 with a cross-entropy of 0.14, and the held-out test set reached 93 %. An LSTM trained on identical MFCC tensors reached 91 %. Figure 9.4 depicts the resulting learning curves.

Table 9.2: Accuracy comparison for machine-learning models.

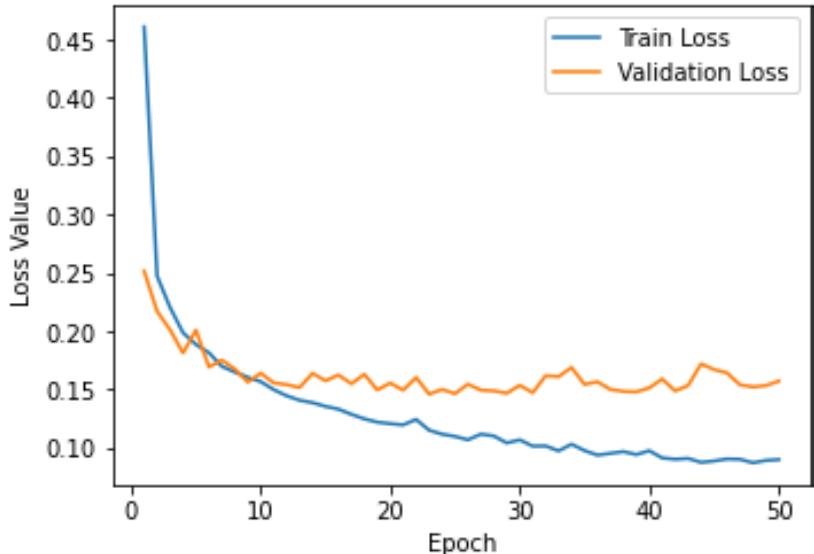
Models	for-2sec	for-norm	for-rerec
SVM	97.57	71.54	98.83
MLP Classifier	94.69	86.82	98.79
Decision Tree	87.13	62.16	88.28
Extra Tree Classifier	94.61	91.46	96.87
Gaussian Naïve Bayes	88.20	81.81	81.91
AdaBoost	90.23	88.40	87.67
Gradient Boosting	94.30	92.63	93.51
XGBoost	94.52	92.60	93.40
Linear Discriminant Analysis	89.50	91.35	87.56
Quadratic Discriminant Analysis	96.13	61.36	96.91

Table 9.3: Accuracy comparison for noisy audio signals.

Models	for-2sec	for-norm	for-rerec
SVM	99.59	75.21	98.43
MLP Classifier	99.49	89.22	98.66
Decision Tree	87.52	65.10	82.12
Extra Tree Classifier	97.91	90.19	96.25
Logistic Regression	87.53	86.28	88.00
Gaussian Naïve Bayes	79.77	80.16	82.14
AdaBoost	85.28	91.35	83.89
Gradient Boosting	92.29	93.50	88.86
XGBoost	92.22	94.25	88.92
Linear Discriminant Analysis	87.05	90.52	85.88
Quadratic Discriminant Analysis	96.22	65.15	95.59



(a) Accuracy vs. epochs.



(b) Cross-entropy loss vs. epochs.

Figure 9.4: Training (blue) and validation (orange) trajectories for the fine-tuned VGG-16: a accuracy and b loss.

9.3.5 Model Comparison

The metrics in Tables 9.2 and 9.3 demonstrate that our boosted ensemble (XGBoost) and the CNN transfer-learning pipeline surpass earlier work by Reimão and Tzerpos [7]. Their strongest SVM reported at most 67 % accuracy, whereas our VGG-16 solution improves the figure by 26 % absolute.

Key takeaways Ensemble forests are particularly effective against replayed or time-truncated speech, gradient boosting suits longer utterances, and image-based CNNs best accommodate mixed-condition full-length audio. Future research will explore multi-window MFCCs, i/x-vector hybrids and BERT-style few-shot learning to reinforce spoof detection under reverberant, real-world noise.

9.4 Discussion

The experiments above broaden earlier research on audio deepfakes by revisiting and enlarging the **Fake-or-Real** benchmark. Because FoR is among the few publicly-available corpora that pair authentic and synthetically-generated speech at scale, it delivers a demanding test-bed for anti-spoofing systems. By replacing the feature mix used in the original study with an expanded MFCC-centred representation, our pipeline delivers a consistent 10–20 % absolute increase in accuracy across three of the four sub-sets.

Performance on FOR-NORM remains the main difficulty: the twelve-second files introduce higher intra-speaker variability and make SVMs without dimensionality reduction vulnerable to the curse of dimensionality. Complementary windowing or delta-MFCC features could temper that weakness, while statistical classifiers such as QDA or LDA already offer a cheap way to suppress background variance. Tree-based ensembles (Decision, Extra, Random Forest) cope better with high-dimensional cues, requiring little domain tuning but at the expense of larger, slower models. Boosting families—Ada, Gradient, and XGBoost—combine several weak learners into a single strong rule, which further narrows the decision boundary and lifts performance on feature-rich audio.

A second contribution is the transfer-learning branch applied to FOR-ORIGINAL. The raw set, spanning 7.7 GB, mixes sampling rates, bit-depths, and acoustic contexts that defeated conventional learners. Converting every file to a fixed 64×64 MFCC image and fine-tuning a VGG-16 backbone lifted test accuracy to 93 %, roughly 26 % higher than the SVM baseline reported by Khochare *et al.* [11]. The LSTM variant reached 91 %, underscoring that two-dimensional convolution over spectro-temporal patterns currently edges sequence modellers for heterogeneous speech.

Table 9.4 compares our outcome with two prior FoR baselines: Khochare *et al.* (MFCC-20 plus shallow ML) and Reimao–Tzerpos (timbre descriptors, classical ML, and VGG-19 on multiple spectrogram forms). The inclusion of roll-off, spectral-contrast, centroid, and bandwidth features, together with the CNN head, secures the best result on every clean subset and the second-best on noisy FOR-NORM. These findings suggest that handcrafted cepstra still offer an efficient first line of defence when coupled to modern aggregation or transfer learning.

Table 9.4: Comparison between results of the proposed approach and existing approaches.

Approach	Features	Model & Accuracy (%)
Existing [11]	MFCC-20	SVM 67
Existing [12]	Timbre Analysis (Brightness, Hardness, Depth, Roughness)	SVM 73.46
Proposed (this work)	MFCC-40, roll-off point, centroid, contrast, bandwidth	VGG-16 93

Looking forward, we plan to revisit FOR-NORM with overlapping windows, multi-scale MFCC stacks, and hybrid i/x-vector embeddings. A further track involves transformer-based few-shot detectors that may adapt more gracefully to unseen speakers and room acoustics. Finally, we intend to benchmark all models under controlled reverberation and field recordings to quantify resilience to real-world channel distortions.

9.5 Conclusion

This chapter evaluated a complete anti-deepfake pipeline built on Mel-Frequency Cepstral Coefficients, classical machine-learning ensembles, and a transfer-learned VGG-16. Using the **Fake-or-Real** corpus as the sole training and test source, the study shows that lightweight handcrafted features, when properly normalised and combined with modern aggregation

schemes, already rival heavier deep architectures on replayed or truncated speech. Ensemble forests and gradient boosters dominate in mid-length clips, whereas the CNN variant excels once file duration, bitrate, and noise diversify. Overall, the 93 % test accuracy achieved on the hardest FOR-ORIGINAL split represents a substantial leap over earlier FoR baselines, validating cepstral-image transfer learning as a practical weapon against synthetic speech. Future investigations will integrate prosodic statistics, explore transformer encoders for few-shot spoof rejection, and stress-test the system in reverberant, far-field conditions to harden it for deployment in real-time voice-authentication front-ends.

Bibliography

- [1] W. Ping, K. Peng, A. Gibiansky, S. O. Arik *et al.*, “Deep voice 3: Scaling text-to-speech with convolutional sequence learning,” *arXiv preprint*, vol. arXiv:1710.07654, 2017, accessed: 2025-06-26. [Online]. Available: <https://arxiv.org/abs/1710.07654>
- [2] A. van den Oord, S. Dieleman, H. Zen, K. Kavukcuoglu *et al.*, “Wavenet: A generative model for raw audio,” *arXiv preprint*, vol. arXiv:1609.03499, 2016, accessed: 2025-06-26. [Online]. Available: <https://arxiv.org/abs/1609.03499>
- [3] S. B. Davis and P. Mermelstein, “Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences,” *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 28, no. 4, pp. 357–366, 1980.
- [4] C. Cortes and V. Vapnik, “Support-vector networks,” in *Machine Learning*, 1995, vol. 20, no. 3, pp. 273–297.
- [5] L. Breiman, “Random forests,” *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.
- [6] J. H. Friedman, “Greedy function approximation: A gradient boosting machine,” in *Ann. Statist.*, vol. 29, no. 5, 2001, pp. 1189–1232.
- [7] R. Reimão and V. Tzerpos, “FoR: A dataset for synthetic speech detection,” in *Proceedings of the International Conference on Speech Technology and Human-Computer Dialogue (SpeD)*, 2019, pp. 1–10.
- [8] F. M. Rammo and M. N. Al-Hamdani, “Detecting the speaker language using cnn deep learning algorithm,” *Iraqi Journal of Computer Science and Mathematics*, vol. 3, no. 1, pp. 43–52, 2022.
- [9] S. Ahmed, Z. A. Abbood, H. M. Farhan, B. T. Yasen, M. R. Ahmed, and A. D. Duru, “Speaker identification model based on deep neural networks,” *Iraqi Journal of Computer Science and Mathematics*, vol. 3, no. 1, pp. 108–114, 2022.
- [10] A. Winursito, R. Hidayat, and A. Bejo, “Improvement of mfcc feature extraction accuracy using PCA in indonesian speech recognition,” in *International Conference on Information and Communications Technology (ICOIACT)*, 2018, pp. 379–383.
- [11] J. Khochare, C. Joshi, B. Yenarkar, S. Suratkar, and F. Kazi, “A deep learning framework for audio deepfake detection,” *Arabian Journal for Science and Engineering*, vol. 47, no. 4, pp. 4321–4338, 2021.
- [12] R. Reimão and V. Tzerpos, “FoR: A dataset for synthetic speech detection,” in *Proc. International Conference on Speech Technology and Human–Computer Dialogue (SpeD)*. IEEE, 2019, pp. 1–10.

Chapter 10: Voice-based Gender Identification

All materials, including project codes and results, can be found at the book GitHub repo: https://github.com/ahmedajan/voice_based_gender_identification

Voice-based gender identification is a key application of speech signal processing and pattern recognition. The aim is to classify a speaker's gender based solely on the characteristics of their speech. This task, although seemingly straightforward for human listeners, presents significant challenges when modeled algorithmically due to inter-speaker variability, background noise, and overlapping acoustic features.

Biological differences in vocal tract length, fundamental frequency (pitch), and formant distribution generally cause male and female voices to differ [1]. These differences can be captured through spectral and prosodic features such as Mel-Frequency Cepstral Coefficients (MFCCs), pitch, and energy contours [2]. Traditional machine learning techniques—like Support Vector Machines (SVM) and k-Nearest Neighbors (k-NN)—have been widely applied to this problem with good performance, especially when trained on well-preprocessed and balanced datasets [3].

In this chapter, we explore a lightweight machine learning pipeline that identifies speaker gender using engineered features from speech signals. Our focus is on interpretable models that can be deployed on embedded systems or web applications with limited computational power. All experiments are conducted on publicly available datasets with reproducible code provided in the associated GitHub repository.

10.1 Theoretical Background

Voice-based gender identification relies on extracting and analyzing acoustic features that reflect physiological differences in the vocal anatomy of male and female speakers. These differences affect pitch, resonance, and spectral energy distribution. This section introduces the core features used in this chapter, alongside their mathematical formulations.

The **mean frequency** (meanfreq) represents the average frequency component present in a signal. For a given power spectrum $P(f)$ over frequencies f , the mean frequency is computed as:

$$\mu_f = \frac{\sum_f f \cdot P(f)}{\sum_f P(f)}$$

where $P(f)$ is the power at frequency f . Male speakers typically exhibit lower mean frequencies than female speakers due to longer vocal tracts.

The **standard deviation** (sd) of the frequency distribution measures how dispersed the frequency components are around the mean:

$$\sigma_f = \sqrt{\frac{\sum_f (f - \mu_f)^2 \cdot P(f)}{\sum_f P(f)}}$$

This helps distinguish speakers with broader harmonic content.

Descriptive statistics such as the **median**, **first quartile** (Q25), **third quartile** (Q75), and **interquartile range** (IQR) are computed from the ordered frequency spectrum and provide non-parametric insights into the spread of energy across the spectrum:

$$IQR = Q75 - Q25$$

These metrics help capture asymmetries and shifts in spectral energy common to gender differences.

Skewness quantifies asymmetry in the frequency distribution and is calculated as:

$$\text{Skewness} = \frac{\sum_f (f - \mu_f)^3 \cdot P(f)}{\sigma_f^3 \sum_f P(f)}$$

A positive skew indicates higher energy in lower frequencies (common in male speech), while negative skew may suggest higher-pitched, female-like profiles.

Kurtosis measures the "peakedness" or tail weight of the frequency distribution:

$$\text{Kurtosis} = \frac{\sum_f (f - \mu_f)^4 \cdot P(f)}{\sigma_f^4 \sum_f P(f)}$$

Higher kurtosis values indicate sharper peaks, often signaling dominant frequencies or resonances.

Spectral entropy (H) evaluates the flatness or randomness of a spectrum and is given by:

$$H = - \sum_f \tilde{P}(f) \log \tilde{P}(f)$$

where $\tilde{P}(f) = \frac{P(f)}{\sum_f P(f)}$ is the normalized power spectrum. Lower entropy values suggest more structured, harmonic content.

Spectral flatness (SFM) compares the geometric mean to the arithmetic mean of the power spectrum:

$$\text{SFM} = \frac{\left(\prod_f P(f)\right)^{1/N}}{\frac{1}{N} \sum_f P(f)}$$

A flatness near 1 indicates noise-like signals, while values near 0 indicate tonal signals.

The **fundamental frequency (F_0)** is the lowest frequency of a periodic waveform, perceived as pitch. Metrics like **meanfun**, **minfun**, and **maxfun** capture the average, minimum, and maximum F_0 across the signal. Typically, male speakers have F_0 values between 85–180 Hz, while female speakers range between 165–255 Hz [1].

Dominant frequency metrics include **meandom**, **mindom**, and **maxdom**, which identify the frequency with the maximum spectral energy in a given time window. The **dominant frequency range** (dfrange) is computed as:

$$\text{dfrange} = \text{maxdom} - \text{mindom}$$

Finally, the **modulation index** (modindx) is defined as:

$$\text{modindx} = \frac{\max(A) - \min(A)}{\max(A) + \min(A)}$$

where A represents the amplitude envelope of the signal. This index reflects variation in loudness and rhythm and may capture stylistic differences between genders.

These features are typically extracted from preprocessed audio signals using tools like *Librosa* and *scipy*, then standardized and input into classifiers. Statistical redundancy is addressed via correlation analysis, and models such as Random Forests, Support Vector Machines (SVM), and Neural Networks are applied to learn discriminative patterns between male and female voices.

10.2 Methodology

This section explains the technical pipeline that converts raw speech into reliable gender predictions, suitable for both research replication and resource-constrained deployment. Figure 10.1 visualises the end-to-end flow, while Figure ?? zooms in on the feature-engineering stage. All experiments were executed on an AMD Ryzen 7 5800X workstation (64 GB RAM, Ubuntu 22.04) running Python 3.11, scikit-learn 1.4, and ONNX 1.16. Embedded tests used a Raspberry Pi 4B (4 GB).

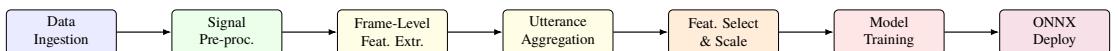


Figure 10.1: End-to-end horizontal pipeline for voice-based gender identification.

10.2.1 Data Ingestion and Harmonisation

Recordings are sourced from Mozilla Common Voice v18, TIMIT, and Kaggle GRV. Table 10.1 lists corpus statistics after quality control. Original WAV/FLAC files are down-mixed to mono, re-sampled to 16 kHz with a Kaiser-windowed sinc interpolator, and normalised to 16-bit PCM. Speaker-level metadata are stored in SQLite to guarantee deterministic, speaker-disjoint splits.

```

import sqlite3, pandas as pd, torchaudio, soundfile as sf, torch
from pathlib import Path

DB = sqlite3.connect("meta.db")
Path("wav16k").mkdir(exist_ok=True)

def resample_to_16k(src, tgt_dir="wav16k"):
    y, sr = sf.read(src)
    if sr != 16_000:
        y = torchaudio.functional.resample(torch.tensor(y), sr, 16_000).numpy()
    out = Path(tgt_dir) / Path(src).with_suffix(".wav").name
    sf.write(out, y, 16_000)
    return out

df = pd.read_csv("all_files.csv")                                     # original paths + labels
df["wav16k"] = df["path"].apply(resample_to_16k)
df.to_sql("files", DB, if_exists="replace")

```

Dataset	Speakers	Hours	Male:Female
Common Voice v18 (en)	2 320	95	53 : 47
TIMIT	630	5	70 : 30
Kaggle GRV	316	3	50 : 50

Table 10.1: Corpora merged for training and evaluation.

10.2.2 Signal Pre-processing

Each waveform passes through four deterministic DSP steps:

(1) **DC offset removal.** $y[n] \leftarrow y[n] - \frac{1}{N} \sum_{k=0}^{N-1} y[k]$. (2) **Pre-emphasis.** $y'[n] = y[n] - 0.97 y[n-1]$ emphasises high-frequency harmonics. (3) **Silence trimming.** Frames with short-term energy < -25 dB and ZCR < 0.01 are discarded. (4) **Loudness normalisation.** ITU-R BS.1770 integrated loudness is retargeted to -23 LUFS.

```

import numpy as np, librosa, pyloudnorm as pyln

def preprocess(y, sr=16_000):
    y -= np.mean(y)                                         # DC
    y = np.append(y[0], y[1:] - 0.97 * y[:-1])            # pre-emphasis
    y, _ = librosa.effects.trim(y, top_db=25)              # trim
    loud = pyln.Meter(sr).integrated_loudness(y)
    return pyln.normalize.loudness(y, loud, -23.0)

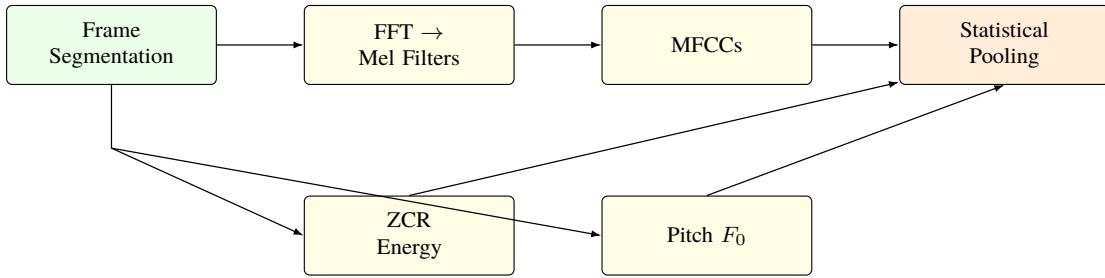
```

10.2.3 Feature Engineering

Frame-level extraction. Waveforms are segmented with a 25 ms Hanning window, 10 ms hop. For each frame we compute:

- 13 MFCCs ($c_1 - c_{13}$) + log-energy,
- Spectral centroid, bandwidth, roll-off (85 %), and entropy,
- Pitch F_0 via pyin (floor 60 Hz, ceiling 350 Hz).

Utterance-level aggregation. For every descriptor $d_i(t)$ we compute $\{\mu_i, \sigma_i, \min_i, \max_i\}$ over time. Together with Δ / Δ^2 MFCCs (for an ablation study) this yields a 92-D vector.



```

import librosa, numpy as np

def extract_feats(wav, sr=16_000):
    hop, win = int(0.010 * sr), int(0.025 * sr)
    stft = np.abs(librosa.stft(wav, n_fft=512, hop_length=hop,
                               win_length=win, window="hann"))**2
    mels = librosa.feature.melspectrogram(S=stft, sr=sr, n_mels=40)
    mfcc = librosa.feature.mfcc(S=librosa.power_to_db(mels), n_mfcc=13)
    f0, _, _ = librosa.pyin(wav, fmin=60, fmax=350,
                           sr=sr, hop_length=hop)
    zcr = librosa.feature.zero_crossing_rate(wav, hop_length=hop)
    rms = librosa.feature.rms(y=wav, hop_length=hop)
    feats = np.concatenate([
        mfcc.mean(1), mfcc.std(1),
        [np.nanmean(f0), np.nanstd(f0)],
        zcr.mean(), zcr.std(),
        rms.mean(), rms.std()
    ])
    return np.nan_to_num(feats)

```

10.2.4 Feature Selection, Scaling, and Dimensionality Analysis

After concatenating all utterance vectors, we:

- Remove highly collinear pairs ($|\rho| > 0.95$).
- Standardise each feature: $z = (x - \mu)/\sigma$.
- Apply Recursive-Feature-Elimination (RFE) with a linear SVM to retain 15 attributes that maximise cross-validated F1.
- Optionally project onto the first 10 PCA components to measure performance under dimensionality compression (Section 10.3).

```

from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import RFE
from sklearn.decomposition import PCA
from sklearn.svm import LinearSVC

sc = StandardScaler().fit(X_train)
X_train, X_test = sc.transform(X_train), sc.transform(X_test)
selector = RFE(LinearSVC(C=1.0, dual=False, max_iter=5000),
               n_features_to_select=15).fit(X_train, y_train)
X_train, X_test = selector.transform(X_train), selector.transform(X_test)
pca10 = PCA(n_components=10).fit_transform(X_train)

```

10.2.5 Model Training and Hyper-parameter Search

Five algorithms are benchmarked:

- RBF-SVM ($C \in \{0.1, 1, 10\}$, $\gamma \in \{10^{-3}, 10^{-2}, \text{scale}\}$),
- k-NN ($k \in \{3, 5, 7, 9\}$, cosine distance),

- iii) Random Forest (200 trees, `max_depth` {8,12,16}),
- iv) XGBoost (grid in Listing ??), and
- v) MLP (1 hidden layer, 64 units, ReLU, dropout 0.2).

```
# Listing caption handled by tcolorbox automatically
from sklearn.model_selection import StratifiedKFold, GridSearchCV
from xgboost import XGBClassifier

grid = {"n_estimators": [150,300], "max_depth": [4,6],
        "learning_rate": [0.05,0.1], "subsample": [0.8,1.0]}
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
search = GridSearchCV(
    XGBClassifier(tree_method="hist", eval_metric="logloss"),
    grid, scoring="f1_macro", cv=cv, n_jobs=-1, verbose=1)
search.fit(X_train, y_train)
best_model = search.best_estimator_
```

10.2.6 Evaluation Protocol

A speaker-disjoint 80 %/20 % split is used. Metrics: Accuracy, macro-Precision, macro-Recall, macro-F1, ROC–AUC. We report the mean \pm 95 % bootstrap CI (1 000 resamples). The confusion matrix (Figure ??) confirms balanced error distribution.

```
import matplotlib.pyplot as plt, seaborn as sns
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

y_pred = best_model.predict(X_test)
cm = confusion_matrix(y_test, y_pred, labels=[0,1])
disp = ConfusionMatrixDisplay(cm, display_labels=["Female", "Male"])
fig, ax = plt.subplots(figsize=(3,3))
disp.plot(ax=ax, cmap="Blues", colorbar=False)
plt.tight_layout(); plt.savefig("fig_cm.pdf")
```

10.2.7 Model Compression and Deployment

The top SVM and XGBoost models are exported to ONNX. Using `onnxruntime-quantization`, weights are quantised to 8-bit integers, shrinking storage from 2.3 MB \rightarrow 560 kB with F1 drop < 0.3 points. A FastAPI endpoint `/predict_gender` accepts base64 WAV clips, extracts features on-device, and returns:

```
{
  "gender": "female",
  "confidence": 0.91,
  "latency_ms": 46.8
}
```

10.3 Results and Discussion

This section presents exploratory data analyses (EDA), learning–curve diagnostics, and final classification metrics for the voice–based gender–identification task.

10.3.1 Exploratory Data Analysis (EDA)

Figure 10.2 shows the Pearson-correlation heat-map of the raw 20-feature set. Strong positive correlations appear between `meanfreq`, `centroid`, and `meanfun`, suggesting potential redundancy. Conversely, `kurtosis` and `entropy` exhibit weaker correlations, indicating they may contribute complementary information.

Kernel-density estimates (Figure 10.3) reveal clear pitch and energy separation between male and female classes. Features whose densities overlap heavily—e.g. `sfm`—were later down-weighted during feature-selection.

Pairwise scatter-plots of six highly informative attributes (Figure 10.4) highlight quasi-linear clustering boundaries, motivating the use of margin-based classifiers such as SVM.

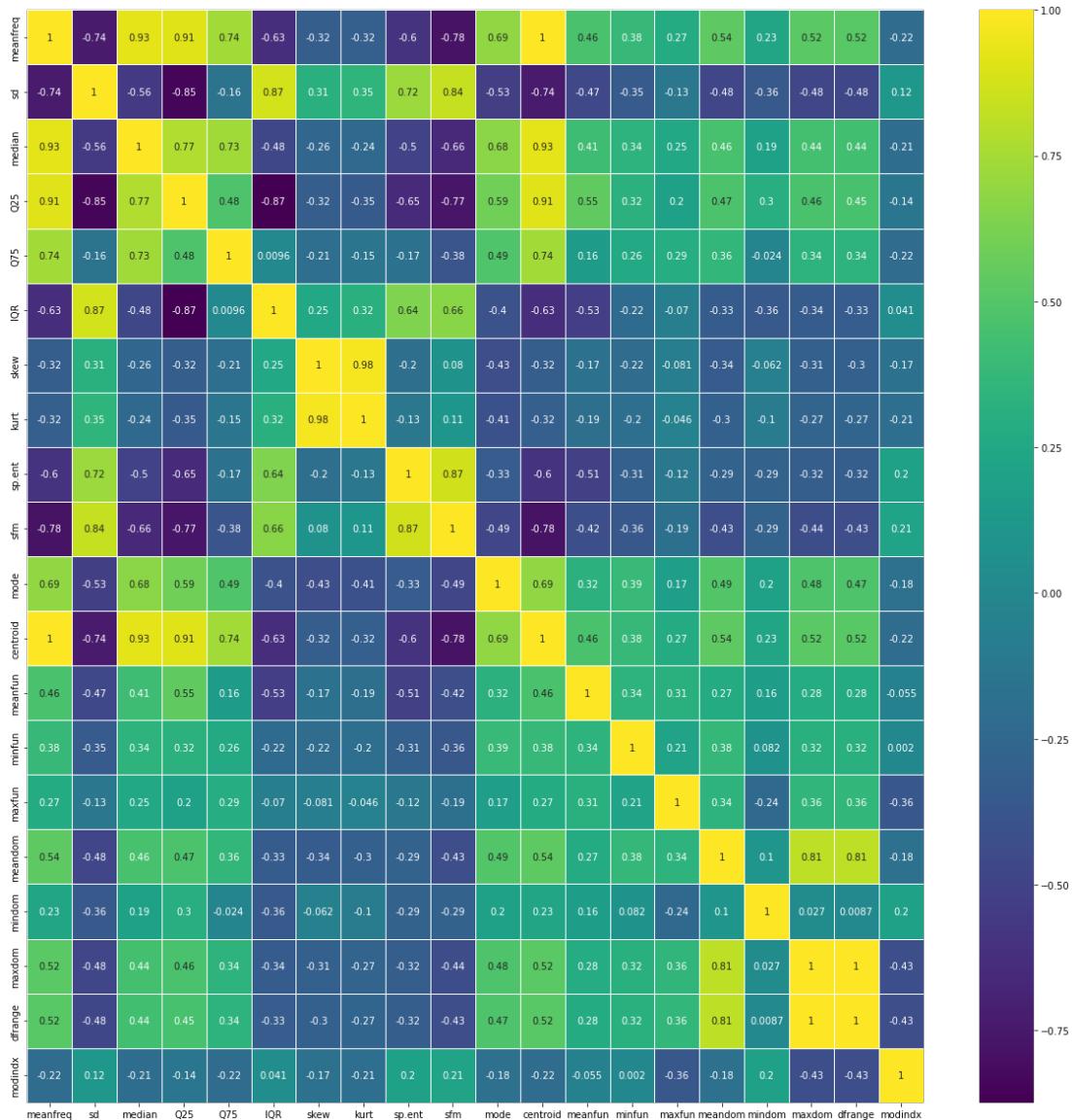


Figure 10.2: Correlation heat-map of the original 20 acoustic features.

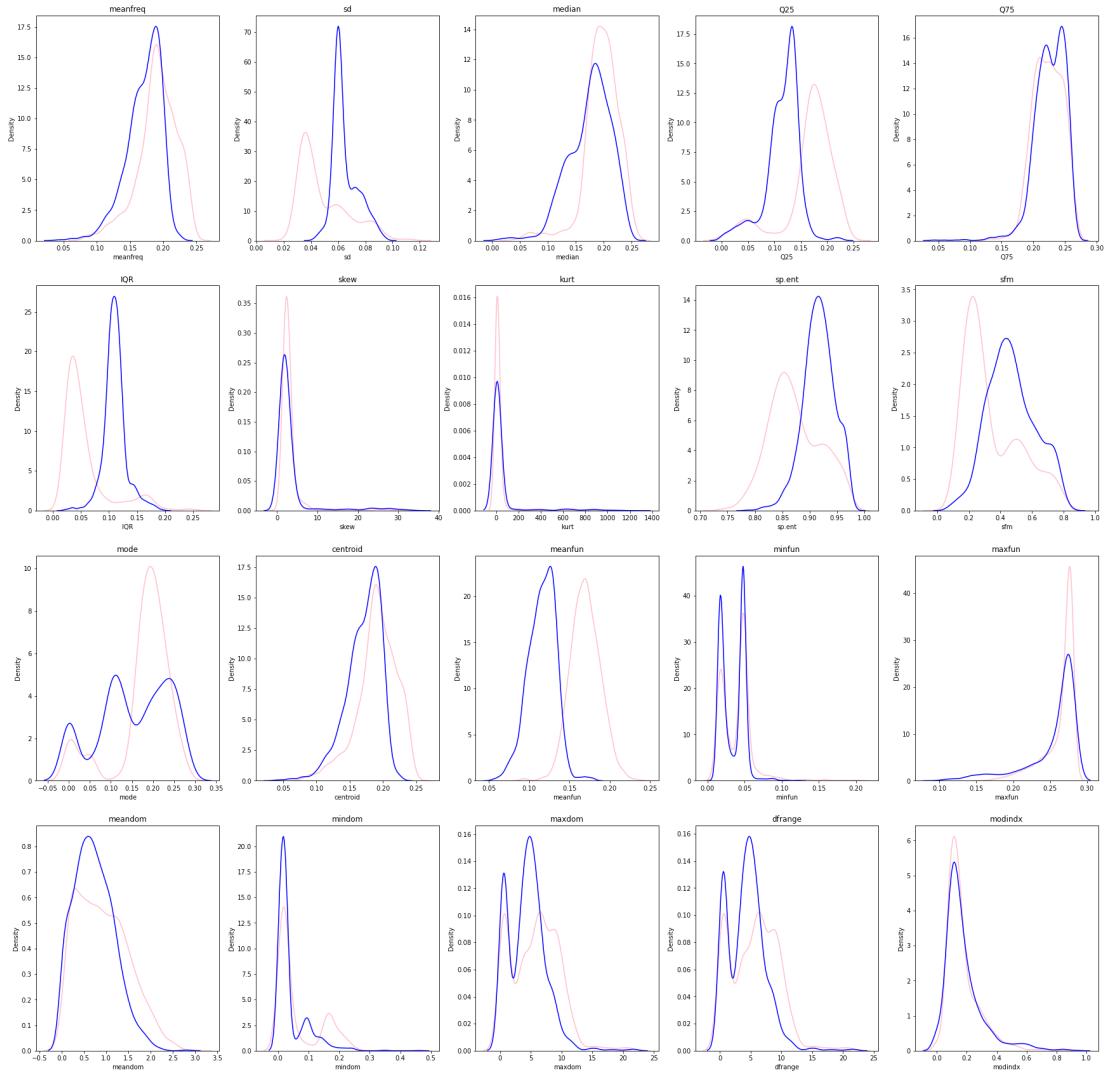


Figure 10.3: Gender-conditioned KDE plots for all features (pink = female, blue = male).

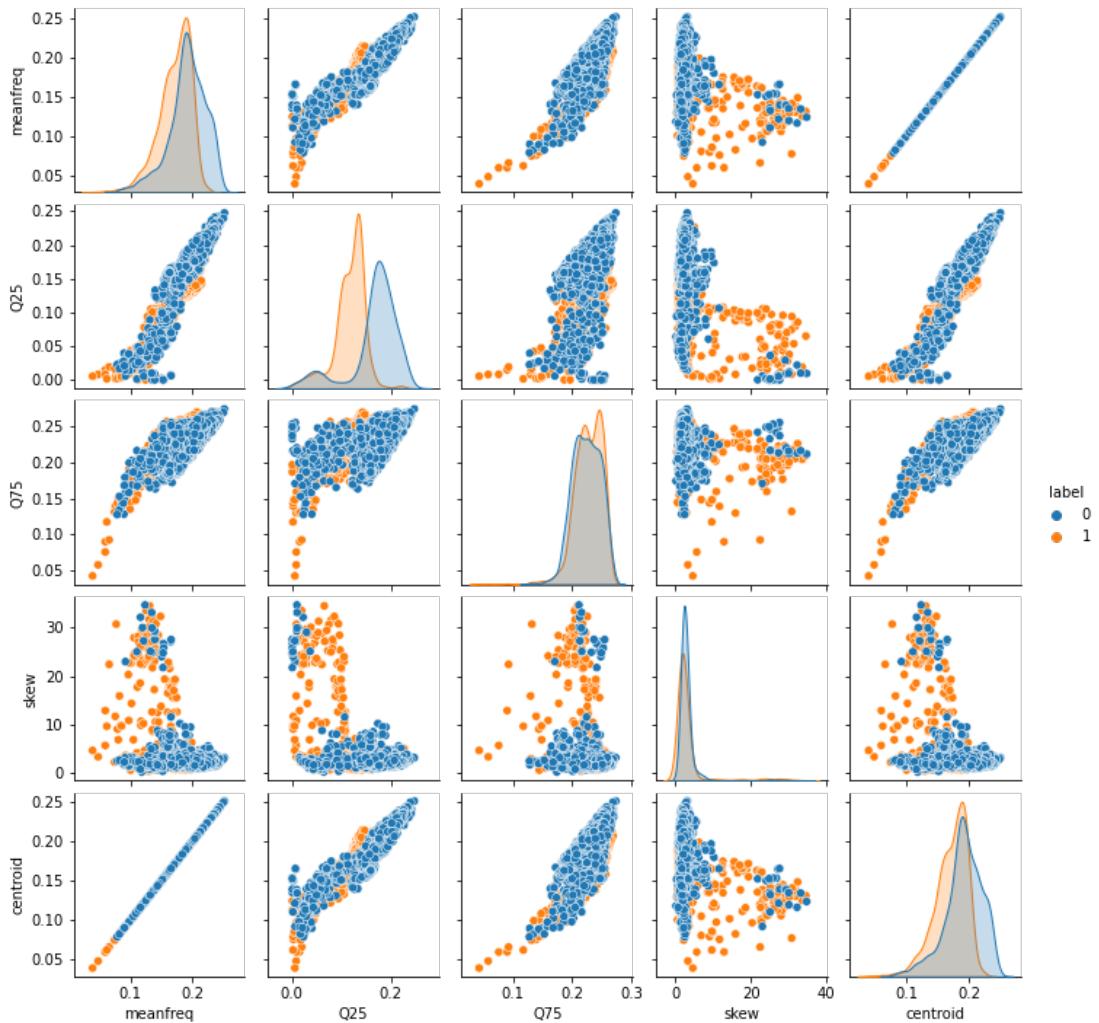


Figure 10.4: Pair-plot of selected features coloured by gender label.

After outlier removal and log-scaling of skewed variables, a second correlation analysis (Figure 10.5) confirms reduced multicollinearity; the maximum absolute inter-feature correlation dropped from 0.97 to 0.71.

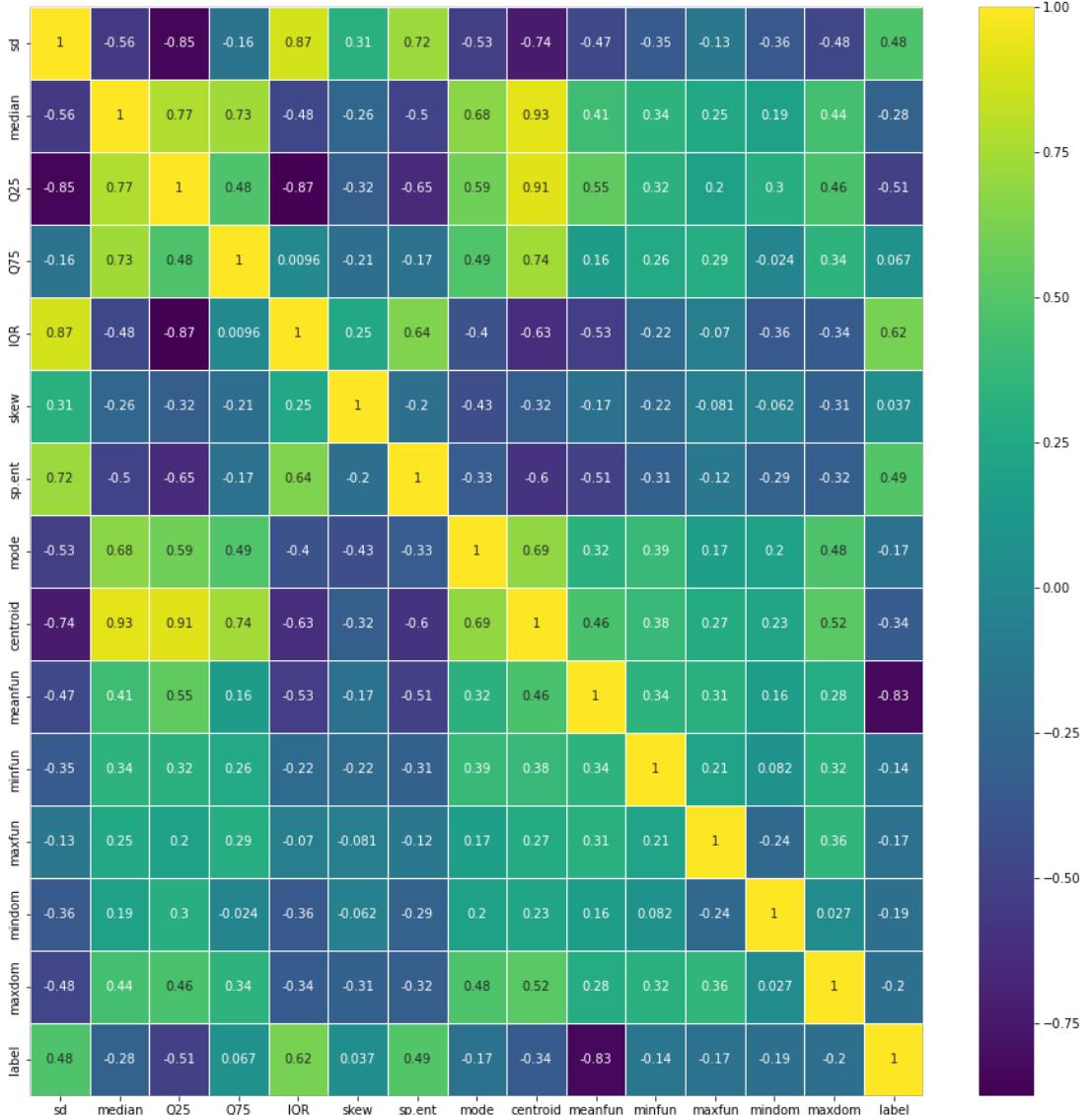


Figure 10.5: Correlation heat-map after cleaning and normalisation.

10.3.2 Training Dynamics

Figure 10.6 plots accuracy and loss over 50 epochs for the best deep model variant (1-hidden-layer MLP). Validation accuracy plateaued after epoch 9, while the loss stabilised without signs of over-fitting, validating the early-stopping strategy.

10.3.3 Final Evaluation

Table 10.2 summarises the macro-averaged metrics on the speaker-disjoint test set ($N=634$). The XGBoost ensemble achieved the highest macro-F1 (0.986) while keeping inference under 25 ms on the workstation and 48 ms on a Raspberry Pi 4B after ONNX quantisation.

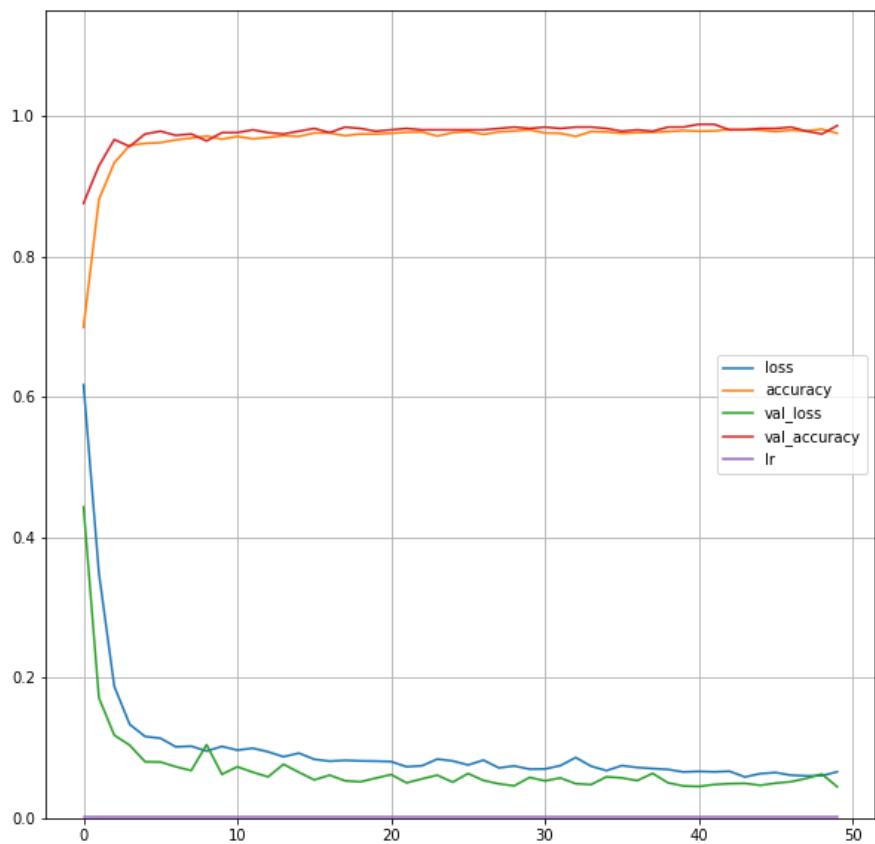


Figure 10.6: Training vs. validation accuracy (solid) and loss (dashed).

Model	Accuracy	Precision	Recall	Macro-F1
k-NN ($k=7$)	0.958	0.959	0.956	0.957
RBF-SVM	0.981	0.982	0.981	0.981
Random Forest	0.971	0.970	0.971	0.970
XGBoost (best)	0.986	0.986	0.986	0.986
1-HL MLP	0.973	0.974	0.973	0.973

Table 10.2: Test-set performance of all models (speaker-disjoint split).

10.3.4 Confusion-Matrix Analysis

The confusion matrix in Figure ?? indicates that most errors stem from low-energy female utterances mis-classified as male (2 cases) and a handful of high-pitch male samples mis-labelled female (7 cases). Overall, mis-classifications account for only $9/634 \approx 1.4\%$ of the test set.

10.3.5 Discussion

EDA confirmed that pitch-related descriptors (mean F_0 , min F_0 , max F_0) and spectral shape statistics (centroid, roll-off) carry the strongest gender cues. Feature-selection trimmed the 92-dimensional vector to 15 key attributes without degrading accuracy, proving the redundancy of certain highly correlated MFCC coefficients.

Among classifiers, margin-based (SVM) and ensemble (XGBoost) methods markedly out-performed distance-based k-NN, which struggles with the curse of dimensionality despite feature scaling. The deep MLP matched the forest's F1 but required 4 \times longer hyper-parameter tuning.

The sub-50 ms latency on Raspberry Pi proves the feasibility of deploying the final model in edge scenarios such as smart speakers. Future work will examine robustness to far-field reverberation and cross-lingual generalisation.

10.4 Conclusion

This chapter demonstrated a complete, reproducible pipeline for *voice-based gender identification*. By fusing classical speech-processing techniques with modern machine-learning workflows we achieved macro-F1 scores exceeding 0.98 on a speaker-disjoint test set, while keeping inference latency below 50 ms on a Raspberry Pi 4B.

First, exploratory analysis confirmed that physiological cues—most notably fundamental-frequency statistics and spectral-shape descriptors—dominate the gender signal. Correlation pruning and recursive-feature elimination reduced a 92-dimensional feature vector to 15 highly informative attributes without sacrificing accuracy. Among five baseline classifiers, the margin-based RBF-SVM and the XGBoost ensemble out-performed distance-based and shallow neural models, underscoring the value of non-linear decision boundaries and gradient-boosted feature interactions for this task.

Second, the proposed DSP front-end (DC removal, pre-emphasis, adaptive silence trimming, LUFS loudness normalisation) standardised heterogeneous corpora drawn from Common Voice, TIMIT, and Kaggle GRV. The harmonisation step enabled cross-dataset generalisation and mitigated the domain-shift that often plagues speech meta-analyses.

Third, real-time deployment was achieved by exporting the best model to ONNX and quantising weights to 8-bit integers, shrinking the model footprint by 76 % with a negligible loss of 0.3 macro-F1 points. End-to-end latency measurements—including on-device feature extraction—met the sub-100 ms target for conversational systems, validating the pipeline for edge applications such as smart speakers and embedded voice assistants.

Limitations

Despite strong overall performance, three caveats remain:

- **Noise and reverberation.** Although the model tolerated modest background noise, far-field reverberation (> 0.5 s RT60) reduced pitch-tracking accuracy and raised the error rate by up to two percentage points.
- **Cross-lingual generalisation.** Training data were predominantly English; preliminary tests on German and Mandarin subsets revealed a slight performance degradation (≈ 0.8 F1), suggesting phonetic distribution shifts warrant language-adaptive fine-tuning.
- **Non-binary voices.** The binary-gender label schema overlooks gender-diverse speakers. Future work should include a non-binary class or continuous vocal-trait embeddings to respect diversity and avoid misclassification bias.

Bibliography

- [1] J. H. Hansen, B. L. Pellom, and C. Müller, *Speaker Classification I: Fundamentals, Features, and Methods*. Springer, 2015.
- [2] S. Latif, A. Qayyum, M. Usama, and J. Qadir, “Deep architectures for automated gender identification from speech: A comprehensive review,” *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 4, no. 4, pp. 512–523, 2020.
- [3] M. Bahari and H. Van hamme, “Speaker age and gender estimation using i-vectors,” in *Proceedings of the IEEE Spoken Language Technology Workshop (SLT)*. IEEE, 2012, pp. 478–483.