DATA MINING CIA-1
TYPE OF DATA :SPEECH

TEAM MEMBERS:
NILANJANA.T 23011101088
NITHYASRI.D 23011101090

1.Documentation on type of data chosen.

Speech processing is a field that studies how speech is produced, understood, and recognized. Speech data consists of recorded spoken words or sentences and serves as a fundamental resource for various applications, including voice assistants, language learning, and pronunciation correction. Unlike textual data, speech data carries additional characteristics such as tone, pitch, speed, and pronunciation, making it more complex and rich in information.

Speech Sounds: Phonemics and Phonetics

- **Phonemes**: The smallest units of sound that carry meaning. English has **40–45 phonemes**.
- **Allophones**: Variations of a phoneme (e.g., **[p] and [ph]** in "spit" vs. "pit").

Pitch and Timbre in Speech

- **Pitch**: Determined by the **vibration frequency** of the vocal cords, helping differentiate sounds.
- **Timbre**: The **quality of sound**, shaped by harmonics, allowing us to distinguish different voices even if pitch and loudness are the same.

Data Characteristics

Speech data contains both **linguistic** and **acoustic** features:

- **Linguistic Features**: Words, phrases, and sentence structures.
- **Acoustic Features**: Tone, pitch, speed, stress, and pronunciation variations.
- **Variability Factors**: Differences in accents, speaking styles, and environmental noise levels can affect speech data quality and usability.

Storage Formats

Speech data is stored in different **audio file formats**:

- **WAV**: Uncompressed, high-quality format.
- **MP3**: Compressed format with reduced file size.
- **FLAC**: Lossless compressed format, retaining high fidelity.

Structured vs. Unstructured Data

Speech data can be categorized as:

- **Structured Speech Data**: Includes **text transcriptions** alongside audio recordings, making analysis and model training easier.
- **Unstructured Speech Data**: Raw audio recordings **without transcriptions**, requiring additional processing to extract meaningful information.

2.Detail the fundamental technical components involved based on the type of data selected.

1. Collect & Load Speech Data

- Speech data is collected in raw audio formats like **WAV, MP3, or FLAC**.
- The audio is loaded into a processing environment such as **Python**.

Technical Components:

- **Sampling Rate**: Number of audio samples per second
- **Bit Depth**: Determines the resolution of each sample

Initial: Raw speech file (**WAV, MP3**).
Output: Digital waveform representation (**array of numerical values**).

2. Preprocessing (Cleaning & Normalization)

- The goal is to **remove noise, trim silences, and normalize amplitude**.

Technical Components in Speech Data:

1. **Noise Reduction**: Removes unwanted background noise.
2. **Silence Removal**: Eliminates long pauses for efficiency.
3. **Amplitude Normalization**: Adjusts volume levels for consistency.

Initial: Raw waveform with noise, varying amplitude, and silent segments.
Output: Cleaned and normalized waveform, making **feature extraction** more effective.

3. Framing & Windowing

- **Speech is a non-stationary signal** (it changes over time).
- To analyze it, we **divide it into short frames** .
- Each frame is multiplied by a **window function** to reduce signal discontinuities.

Technical Components in Speech Data:

- **Frame Size**: Typically 20–30 ms
- **Window Function**: Used to smooth transitions

Input: Continuous waveform.
Output: Overlapping frames of speech, ready for **feature extraction**.

4. Extracting Features

- Each frame is **converted into feature vectors** representing its **acoustic properties**.
- Various **feature extraction techniques** are used.

Common Speech Features & Their Technical Components:

Feature	Description
MFCC (Mel-Frequency Cepstral Coefficients)	**Captures the shape of the speech spectrum**
Spectral Features (Centroid, Bandwidth)	**Capture frequency characteristics**
Pitch (Fundamental Frequency, F0)	**Represents pitch variations**
Zero-Crossing Rate (ZCR)	**Measures frequency of sign changes**
Chroma Features	**Capture harmonic and pitch information**

Initial: Framed waveform.
Output: **Numerical feature vectors**.

5. Feature Scaling & Normalization

- Converts extracted features into a **uniform scale**.
- Helps **machine learning models** converge faster and perform better.

Technical Components in Speech Data:

1. **Min-Max Scaling**: Rescales values between 0 and 1.
2. **Standardization** (Z-score Normalization): Mean 0, standard deviation 1.

Initial: Raw extracted features with different ranges.

Output: Scaled features

3. Application for the type of data selected

APPLICATION:SPEECH RECOGNITION

Speech recognition is a technology that converts spoken language into text. It is a key application of speech processing, which involves analyzing, understanding, and manipulating speech signals.
Speech recognition is used in virtual assistants (e.g., Siri, Google Assistant), transcription services, voice search, language learning apps like Duolingo to assess pronunciation and provide real-time feedback

Challenges in Speech Recognition Processing

1. Data Collection Stage

- **Background Noise (Denoising Required)**
  - Unwanted sounds such as traffic, fan noise, or conversations interfering with speech clarity.
  - **Example:** A phone call at a busy café picks up chatter and clinking dishes, making speech extraction difficult.

2. Preprocessing Stage

- **Accent & Pronunciation Variations**
  - Different accents cause variations in pronunciation, making recognition harder for models trained on limited speech patterns.
  - **Example:** "Tomato" is pronounced as *tə-MAY-toh* in the U.S. and *tə-MAH-toh* in the UK, leading to recognition errors in accent-specific models.

3. Framing & Segmentation Stage

- **Segmentation Errors**
  - Errors in splitting continuous speech into words or phonemes, leading to incorrect feature extraction.
  - **Example:** The phrase *"I scream"* is misinterpreted as *"Ice cream"* due to the lack of a pause.

4. Feature Extraction Stage

- **Homophones (Word Ambiguity)**
  - Words that sound the same but have different meanings, making differentiation difficult without context.
  - **Example:** *"I'll write it down"* is transcribed as *"I'll right it down,"* causing meaning distortion.

4.Problem description in detail.

Background noise in Speech data
Solution : Denoising

Background noise in speech recognition refers to unwanted environmental sounds, such as traffic, chatter, or machinery, that interfere with capturing clear speech. It reduces the accuracy of speech recognition models by making it harder to distinguish spoken words from noise. This can lead to misinterpretations, lower transcription quality, feature extraction and increased errors in voice-controlled applications

What is Denoising?

Denoising is the process of removing unwanted noise from speech signals while preserving the speaker's voice and linguistic content.

5.Methods to solve the problem(Denoising)

1. Median Filtering (Basic Filtering Approach)

How It Works:

- A moving window (e.g., 3-5 samples wide) slides over the signal.
- Each sample in the window is replaced with the median of its neighboring samples.
- Effectively removes impulse noise (sharp, sudden spikes).

Disadvantages:

- Cannot remove continuous background noise (e.g., traffic or crowd noise).
- If the window size is too large, it slightly distorts the speech signal.

2. Moving Average Filter

How It Works:

- Similar to the median filter but replaces each sample with the mean of its neighbors.
- Reduces sharp variations in the signal, making it smoother.
- Helps suppress random noise but does not selectively remove background sounds.

Disadvantages:

- Causes blurring of the speech signal, leading to loss of clarity.
- Ineffective for removing non-stationary noise (e.g., sudden background sounds).

3. Fast Fourier Transform (FFT-Based Noise Reduction)

How It Works:

- Converts the speech signal from the **time domain** to the **frequency domain** using FFT.
- Identifies and removes noise frequencies by applying a filter.
- Converts the cleaned signal back to the **time domain** using **Inverse FFT (IFFT)**.

Why is it better than the other methods?
Works well for both **stationary** and **non-stationary** noise.
More **adaptive** to variations in speech and background noise.
Preserves speech clarity better than **moving average** or **median filtering**.

FINAL SOLUTION USED : DISCRETE FOURIER TRANSFORM.

Steps to solve the problem(Denoising) Using DFT

Step 1: Import Required Libraries

- Import numerical computation libraries (e.g., NumPy) for mathematical operations.
- Import audio handling libraries to read and write audio files.

Step 2: Read the Audio File

- **Input Data:** The input is an audio file (e.g., WAV, FLAC) containing sound, including noise.
- **Data Format:** The audio signal is stored as numerical samples, usually integers or floating-point values.
- **Sample Rate:** This defines how many samples are taken per second

Step 3: Convert Audio to Frequency Domain Using FFT

- **Fourier Transform:** Converts the time-domain audio signal into the frequency domain.
- **Mathematical Formula (Discrete Fourier Transform - DFT):**

$$X(\omega) = \sum_{n=-\infty}^{\infty} x(n) e^{-j\omega n}.$$

- X[k]=∑(n=0to n=N−1)x[n]e^−j(2π/N)kn
  - X[k] = Frequency domain representation of the signal
  - x[n] = Time-domain audio sample at index n
  - N = Total number of samples
  - k = Index of the frequency component
  - e−j(2π/N)kn = Complex exponential term representing the basis functions
- **Explanation:**
  - The formula sums all time-domain samples after multiplying them by exponential terms (cosine and sine waves).
  - This helps break down the signal into individual frequency components.

Step 4: Identify and Remove Noise Frequencies

- **Filtering:** Identify noise frequencies and set them to zero in the FFT output.
- **Complex Representation:** The FFT output is stored as complex numbers (real and imaginary parts).
- **Magnitude Calculation Formula:**
- ∣ X[k] ∣ =Re(X[k])2+Im(X[k])2|X[k]| = \sqrt{\text{Re}(X[k])^2 + \text{Im}(X[k])^2}
  - ∣ X[k] ∣ |X[k]| = Magnitude of the frequency component
  - Re(X[k])= Real part of the complex number
  - Im(X[k])= Imaginary part of the complex number
- **Explanation:**
  - The magnitude represents the intensity of each frequency component.
  - By identifying and removing specific frequency components (e.g., background noise), the unwanted noise is reduced.

Step 5: Convert the Filtered Signal Back to Time Domain Using Inverse FFT (IFFT)

- **Mathematical Formula (Inverse Discrete Fourier Transform - IDFT):**

$$x(n) = \frac{1}{N}\sum_{k=0}^{N-1} X(k)e^{j2\pi\frac{kn}{N}}.$$

- x[n]=1/N∑(k=0 to k=N−1) X[k]e^j(2π/N)knx[n]
  - x[n] = Reconstructed time-domain signal
  - X[k] = Filtered frequency-domain signal
  - e^(j(2π/N)kn)= Complex exponential function for transformation back to the time domain
- **Explanation:**
  - This formula reconstructs the original audio signal after removing noise.
  - The inverse transformation ensures the cleaned signal retains its original structure.

Step 6: Save the Cleaned Audio File

- Convert the cleaned signal back to a standard format (e.g., WAV).
- Normalize the signal to maintain audio quality.

Step 7: Visualize the Original and Filtered Signals

- Plot the waveform before and after noise cancellation to observe differences.
- Display frequency spectrums to verify noise removal.

PSEUDO CODE :

```
FUNCTION simple_noise_cancellation(input_file, output_file, noise_freqs)
```

```
// Step 1: Load the audio file
data, sample_rate = READ_AUDIO(input_file)
// Step 2: Apply FFT to convert time-domain signal to frequency-domain
spectrum = APPLY_FFT(data)
// Step 3: Create an array of frequency values corresponding to FFT output
frequencies = CALCULATE_FREQUENCIES(spectrum, sample_rate)
// Step 4: Filter out noise frequencies
FOR each freq IN noise_freqs DO
    // Identify indices for the noise frequency range
    indices = FIND_INDICES(frequencies, freq, 50)
    // Set noise frequencies to zero in the spectrum
    SET_SPECTRUM_VALUES_TO_ZERO(spectrum, indices)
END FOR
// Step 5: Apply Inverse FFT to get back to the time-domain
filtered_signal = APPLY_IFFT(spectrum)
// Step 6: Save the cleaned audio to a new file
WRITE_AUDIO(output_file, filtered_signal, sample_rate)
// Step 7: (Optional) Plot the original and filtered signals
IF PLOT_ENABLED THEN
    PLOT_SIGNALS(data, filtered_signal)
END IF
END FUNCTION
```

Functions Involved in Noise Cancellation

1. **Function Definition:** Main function to process the audio file.
2. **READ_AUDIO:** Reads audio data and sample rate from the file.
3. **APPLY_FFT:** Converts the audio signal to the frequency domain.
4. **CALCULATE_FREQUENCIES:** Generates the frequency values for FFT output.

5. **FIND_INDICES:** Identifies noise frequency indices for removal.
6. **SET_SPECTRUM_VALUES_TO_ZERO:** Eliminates unwanted noise frequencies.
7. **APPLY_IFFT:** Converts the cleaned frequency-domain signal back to time domain.
8. **WRITE_AUDIO:** Saves the cleaned audio file.
9. **PLOT_SIGNALS:** Visualizes the original and filtered audio signals.

Final Output

- **Time-Domain Signal:** A noise-free audio signal.

This process effectively enhances the audio signal by leveraging Fourier Transform principles to isolate and remove noise frequencies.