Lab 7 : HMM for Single word speech Recognition

Lab 7: HMM for Single-Word Speech Recognition

Objective:

To implement **Hidden Markov Models (HMMs)** for **isolated digit recognition (0-9)** using **MFCC features** extracted from speech recordings.

Interface:

The system consists of **two phases**:

1. **Training Phase**
   - Extract **MFCC** features from speech recordings of digits.
   - Train an **HMM model** for each digit using Gaussian HMM.
2. **Testing Phase**
   - Extract **MFCC** features from test audio files.
   - Use **Viterbi decoding** to determine the most likely digit.
   - Compute **recognition accuracy**.

Input:

- **Speech dataset:** Audio recordings of spoken digits **(0-9)**.
- **MFCC extraction:** Converts speech into numerical features.
- **Train/Test Split:** Separate training and testing datasets.

Example:

- `train/one/file1.wav, file2.wav, ...`
- `test/one/file3.wav, file4.wav, ...`

Output:

1. **Trained HMM models** for each digit.
2. **Recognition accuracy** on test data.
3. **Predicted vs Actual Digits** for evaluation.

Example Output:

```
HMM training completed successfully!
Recognition Accuracy: 47.50%
```

HMM for Digit Recognition

1 Extract MFCCs from audio recordings (0-9).

2 Train an HMM model for each digit.

3 Use the Viterbi algorithm to predict the digit from an audio file.

In [2]:

```
import os
import numpy as np
import librosa
from hmmlearn import hmm
from sklearn.metrics import accuracy_score

# Define digit classes and their corresponding folder names
digit_map = {
    "0": "zero", "1": "one", "2": "two", "3": "three", "4": "four",
    "5": "five", "6": "six", "7": "seven", "8": "eight", "9": "nine"
}

dataset_path = "../dataset/numbers-split/"  # Update path if needed
models = {}

def extract_mfcc(audio_path):
    y, sr = librosa.load(audio_path, sr=None)  # Load audio
    mfcc = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=13)  # Compute MFCCs
    return mfcc.T  # Transpose for HMM training

# Training phase
train_path = os.path.join(dataset_path, "train")
for digit, folder in digit_map.items():
    mfccs = []  # Store all MFCCs for this digit

    digit_path = os.path.join(train_path, folder)
    if not os.path.exists(digit_path):
        print(f"Warning: Folder not found - {digit_path}")
        continue  # Skip if the folder is missing

    for file in os.listdir(digit_path):
        if file.endswith(".wav"):
            audio_path = os.path.join(digit_path, file)
            mfccs.append(extract_mfcc(audio_path))

    if not mfccs:
        print(f"Warning: No WAV files found for digit {digit} ({folder})")
        continue  # Skip training if no audio files

    # Stack MFCCs and store lengths
    X = np.vstack(mfccs)
    lengths = [len(mfcc) for mfcc in mfccs]

    # Train HMM model
    model = hmm.GaussianHMM(n_components=5, covariance_type="diag", n_iter=100)
    model.fit(X, lengths)

    models[digit] = model  # Store trained model

print("HMM training completed successfully!")

# Testing phase
test_path = os.path.join(dataset_path, "test")
y_true, y_pred = [], []

for digit, folder in digit_map.items():
    digit_path = os.path.join(test_path, folder)
    if not os.path.exists(digit_path):
        continue  # Skip if the folder is missing

    for file in os.listdir(digit_path):
        if file.endswith(".wav"):
            audio_path = os.path.join(digit_path, file)
            mfcc = extract_mfcc(audio_path)
            log_likelihoods = {d: model.score(mfcc) for d, model in models.items()}
            predicted_digit = max(log_likelihoods, key=log_likelihoods.get)

            y_true.append(digit)
            y_pred.append(predicted_digit)

# Compute accuracy
accuracy = accuracy_score(y_true, y_pred)
print(f"Recognition Accuracy: {accuracy * 100:.2f}%")
```

```
HMM training completed successfully!
Recognition Accuracy: 47.36%
```

Inference & Insights:

✅ **HMM-based digit recognition** effectively models **temporal dependencies** in speech.
✅ **MFCC features** provide a compact representation of speech sounds.
✅ **Higher accuracy** can be achieved with **more training data** and **hyperparameter tuning**.
✅ **Extensible** to **large vocabulary** speech recognition tasks.