SoundSync Combined Documentation

File: PESQ_MSE.py

Purpose:

This script evaluates audio quality by computing two key metrics:

- **PESQ (Perceptual Evaluation of Speech Quality)**: An ITU-T standard metric for evaluating the perceived quality of audio.
- **MSE (Mean Squared Error)**: Measures the difference in the spectral representations of two audio files.

It compares an original audio sample against a modified one and stores the results in a CSV file.

Libraries Used:

- `numpy`: For array operations and numerical computations.
- `librosa`: For audio loading, resampling, STFT, and audio feature extraction.
- `scipy.ndimage.zoom`: For resizing spectrograms to match dimensions.
- `scipy.signal.resample_poly`: For resampling audio to a target sample rate.
- `matplotlib.pyplot`: Imported for potential plotting (not used in this script).
- `pesq`: To compute PESQ scores using ITU-T P.862 standard.
- `csv`: For writing metrics to a CSV file.

Main Function: `calculate_metrics(y1, y2, sr1, sr2, target_sr)`

Steps:

1. **Resample Both Audio Files**:
    - `y1` and `y2` are resampled to a `target_sr` (16 kHz).
2. **Zero Padding**:
    - Pad both resampled signals to the same length to enable metric comparison.
3. **PESQ Calculation**:
    - PESQ is computed in 'wideband' mode. If computation fails, returns `None` and logs the error.
4. **STFT and Spectrogram Conversion**:
    - Converts audio signals to spectrograms using `librosa.stft()`.
    - Transforms them to decibel scale using `librosa.amplitude_to_db()`.
5. **Frequency Range Filtering**:
    - Focuses on the 100 Hz to 4000 Hz range, relevant for voice analysis.
6. **Dimension Matching**:
    - Uses `zoom()` to resize the spectrograms if they are not aligned in time.
7. **MSE Computation**:
    - Calculates the Mean Squared Error between the decibel-scaled spectrograms.

Returns:

- `pesq_score`: PESQ score or 'N/A' if calculation failed.
- `mse`: Mean Squared Error between spectrograms.

Execution Block:

- Loads a pair of audio files: `NormalAudio1.wav` (original) and `vocals.wav` (modified).
- Computes PESQ and MSE.
- Writes results to a CSV file named `metrics_results.csv`.
- Output is printed to the terminal and saved to the CSV.

Output:

A file `metrics_results.csv` with columns:

PercentageError, PESQ, MSE
1, <value or N/A>, <value>

Usage:

Ensure the following files exist in your directory:

- `Audio_file_1.wav`
- `Audio_file_2.wav`

Then run:

python PESQ_MSE.py

Ensure dependencies are installed:

pip install -r requirements.txt

Notes:

- `PercentageError` is currently hardcoded to `1` for demonstration. Loop can be expanded for batch evaluation.
- `matplotlib` is imported but not used.

File: STOI.py

Purpose:

This script calculates the **Short-Time Objective Intelligibility (STOI)** score between two audio files, which estimates the intelligibility of speech degraded by noise, processing, or transmission.

Libraries Used:

- `librosa`: For loading audio with sample rate control.
- `numpy`: For padding arrays.
- `soundfile`: For audio I/O (though unused in current version).
- `pystoi`: For calculating STOI scores.

Key Functions:

`load_audio(file_path, target_sr=16000)`

- Loads audio in mono mode at the specified sampling rate (default is 16 kHz).

`pad_audio(audio1, audio2)`

- Pads the shorter of two audio arrays with zeros so both are equal in length.

`calculate_stoi(reference_file, test_file)`

1. Loads both reference and test audio files.
2. Pads them to equal length.
3. Computes the STOI score using `pystoi.stoi()`.
4. Prints and returns the score.

Example Execution:

python STOI_Evaluator.py

Make sure the following files exist:

- `Audio_file_1.wav`
- `Audio_file_2.wav`

Output:

Prints the STOI score, e.g.:

STOI Score: 0.8642

Notes:

- `extended=False` in `pystoi.stoi()` indicates standard (non-extended) STOI is used.
- Sample rates are automatically matched.

File: ASR_Word_Extractor.py

Purpose:

This script performs **automatic speech recognition (ASR)** using OpenAI's Whisper model and extracts individual words with their timestamps. The results are written to a CSV file for further analysis or use.

Libraries Used:

- `whisper`: For loading the Whisper model and transcribing audio.
- `sys`: For reading command-line arguments.
- `csv`: For writing recognized words to a CSV file.

Key Function:

`extract_words_with_timestamps(audio_path, output_csv)`

1. Loads the Whisper model (`small` version).
2. Transcribes the input audio file with word-level timestamps.
3. Extracts and validates each word entry.
4. Writes valid words to a CSV file with one word per row.

Execution:

python perform_ASR.py <audio_file> <output_csv>

Example:
python perform_ASR.py speech.wav words.csv

Output:

CSV file with header `word` and one word per line.

word
Hello
world
...

Notes:

- Only entries with valid word, start, and end times are included.
- The script includes debug prints to aid in development.
- Useful for applications like silence trimming, word-level alignment, or further linguistic processing.

File: WER_CER_Batch_Evaluator.py

Purpose:

This script evaluates **Word Error Rate (WER)** and **Character Error Rate (CER)** for a batch of transcription files against a reference transcript. It also provides a human-readable quality rating and saves the metrics to a CSV.

Libraries Used:

- `os`: For directory traversal and file path handling.
- `csv`: For writing evaluation results.
- `re`: For pattern matching to extract percent loss.
- `jiwer`: For computing WER.
- `numpy`: For computing Levenshtein distance for CER.

Key Functions:

`levenshtein_distance(ref, hyp)`

- Computes the character-level Levenshtein distance between two strings.

`interpret_quality(score)`

- Returns a qualitative interpretation of the WER score (e.g., Excellent, Good, Fair).

`extract_percent_loss(filename)`

- Extracts the percentage of packet loss encoded in the file name, e.g., `1_0_5.txt` → 0.05.

`compute_batch_metrics(folder_path, reference_file, output_csv)`

1. Loads the reference transcript.
2. Iterates over all matching `.txt` files.

3. Computes WER and CER.
4. Interprets quality based on WER.
5. Sorts results by percent loss.
6. Saves metrics to a CSV file.

Example Execution:

python WER_CER_Batch_Evaluator.py

(Or include in another batch evaluation script.)

Output:

CSV file with header:

file_name, percent_loss, WER, CER, Quality

Notes:

- Files should follow a naming pattern like `1_0_5.txt` where numbers represent the loss.
- `1_0_1.txt` is used as the reference file in the example.
- Output is sorted by percent loss for easier interpretation.
- WER is computed using `jiwer`, while CER uses custom Levenshtein logic.
- All metric scores are normalized to the [0, 1] range.