Speaker Voice Similarity Analysis and Evaluation

Carmel Gafa

***Abstract*—This study investigates the effectiveness of a WavLM-based system in distinguishing between 285 different speakers from the ABI-1 dataset.**

I. Introduction

Speaker identification is the determination of a speaker's identity from a segment of their speech. It is crucial in applications such as personalized voice assistants, security systems, and partitioning audio streams according to each speaker's identity. Unlike speech recognition, which focuses on what is being said, speaker identification is concerned with who is speaking.

This task can effectively be approached as a downstream application of pre-trained self-supervised speech models, such as `Wav2Vec2`[**baevski2020wav2vec**] or its enhanced variant `WavLM`[**chen2022wavlm**]. These models are trained on large-scale unlabelled audio datasets to learn audio representations that capture phonetic and speaker-specific characteristics.

In this project, `WavLM-base-plus-sv`, a fine-tuned version of `WavLM` specialized for speaker verification, is used to extract speaker embeddings from audio samples. These embeddings are compared using cosine similarity to measure the similarity of two voice samples. This architecture enables a simple speaker identification pipeline without the need to train a model from scratch.

II. Data preprocessing

The Accents of the British Isles (ABI) Corpus represents 13 regional accents. Figure 1 shows how these accents are categorized into four broad accent groups; Scottish, Irish, Northern English, and Southern English. Each broad group is further split into its respective regional accents[**najafian2016improving**].

code		Location	code		Location
Southern	sse	Standard Southern English	Northern	ncl	Newcastle
	crn	Cornwall		lvp	Liverpool
	ean	East Anglia		brm	Birmingham
	ilo	Inner London		lan	Lancashire
				eyk	East Yorkshire
				nwa	North Wales
Irish	roi	Republic of Ireland	Scottish	shl	Scottish Highlands
	uls	Ulster		gla	Glasgow

Fig. 1. Accents represented by the ABI Corpus[**najafian2016improving**].

The corpus includes 285 subjects, with speech collected from individuals who have lived in each regional accent area since birth. Each of the 285 subjects read the same three short passages. These are short paragraphs forming the 'sailor passage', with lengths of 92, 92, and 107 words with average durations of 43.2, 48.1, and 53.4 seconds, respectively[**najafian2016improving**].

The sources do not explicitly describe the recording conditions of the ABI corpus, however, it is mentioned that the speakers were selected by a phonetician, suggesting an effort for a standard quality recording for at least that accent[**najafian2016improving**].

A. Signal resampling

Audio signals depend on two main parameters

- number of channels C, that is 1 for mono and 2 for stereo.
- number of samples $T = t \times f_s$; that in turn depends on the
  - duration t of the audio in seconds.
  - sampling rate f_s (e.g. 48,000 Hz).

For a mono signal, that is common in these applications, a vector representing the utterance is available for processing, $\mathbf{x}_{\text{raw}} \in \mathbb{R}^{1\times T}$ in this way.

The model by Microsoft that we are using in this example expects a signal rate of 16,000 Hz, and it is therefore necessary to resample the signal to this frequency. The resampling step involves selecting a subset of samples from the original vector, so that the original sampling rate f_s^{raw} becomes a lower one f_s^{target}. The down-sampling ratio in this case, $R = \frac{f_s^{\text{raw}}}{f_s^{\text{target}}} = 3$

Before reducing the number of samples, the high-frequency components from the original signals are removed, as they could cause aliasing. Aliasing occurs when frequencies above the new Nyquist frequency (which is one-half the new sampling rate) "fold back" into the signal, corrupting it. The Nyquist frequency after down-sampling becomes:

$$f_N^{\text{new}} = \frac{f_s^{\text{target}}}{2} = 8000 \text{ Hz}$$

To remove the high frequency components, a low-pass filter is applied, generating a filtered signal, $\tilde{x}$

$$\tilde{x}[n] = x_{raw}[n] * h[n]$$

where $h[n]$ is the impulse response of a low-pass filter, typically a windowed `sinc` function:

$$h[n] = \text{sinc}\left(\frac{n}{R}\right) \cdot w[n]$$

Here, $w[n]$ is a window function (like Kaiser, Hamming, etc.) to localize the infinite `sinc` filter in time.

If, as an example, we consider this function:

$$x_{raw} = [x[0],\ x[1],\ x[2],\ x[3],\ x[4],\ x[5],\ x[6],\ x[7],\ x[8]]$$
$$= [2,\ 4,\ 6,\ 8,\ 10,\ 8,\ 6,\ 4,\ 2]$$

and use the following windowed `sinc` filter (kernel)

$$h = [h[0],\ h[1],\ h[2]]$$
$$= [0.2,\ 0.6,\ 0.2]$$

We slide the kernel over the signal and compute:

$$\tilde{x}[n] = \sum_{k=0}^{2} x[n+k] \cdot h[k]$$

So that the first two samples of the filtered signal becomes

$$\begin{aligned}
\tilde{x}[0] &= 0.2 \cdot x[0] + 0.6 \cdot x[1] + 0.2 \cdot x[2] \\
&= 0.2 \cdot 2 + 0.6 \cdot 4 + 0.2 \cdot 6 \\
&= 0.4 + 2.4 + 1.2 \\
&= 4.0 \\
\tilde{x}[1] &= 0.2 \cdot x[1] + 0.6 \cdot x[2] + 0.2 \cdot x[3] \\
&= 0.2 \cdot 4 + 0.6 \cdot 6 + 0.2 \cdot 8 \\
&= 0.8 + 3.6 + 1.6 \\
&= 6.0
\end{aligned}$$

Once the signal is filtered, we can keep every R^{th} sample and discard the rest (decimation), or alternatively, apply interpolation to produce a smoother, resampled signal at the desired rate.

$$x_{\downarrow R}[n] = x[Rn]$$

So that, for example,

$$x_{\downarrow 3}[n] = x[0],\ x[3],\ x[6],\ x[9], \ldots$$

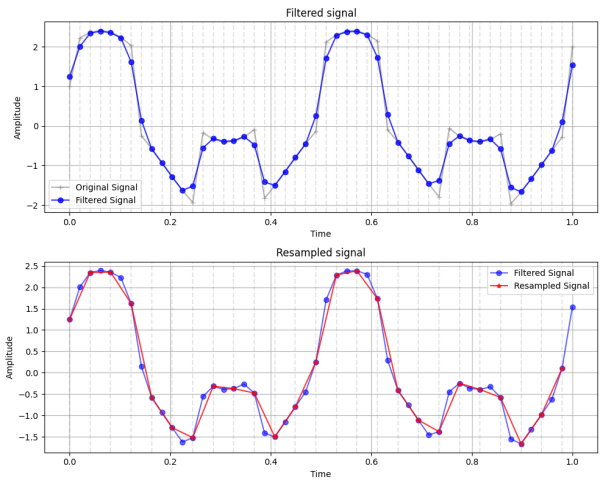

Fig. 2.

III. Methodology

In this section, we will discuss the implementation of the speaker voice similarity pipeline. Several general criteria were observed during this exercise:

- Code that manipulated data was implemented in Python scripts, whereas code that analysed data in Python notebooks. This approach is preferred for several reasons:
  - Scripts encourage modularity better.
  - It is easier to import developed functions when using scripts.
  - Scripts avoid the notebook's overheads like kernel restarts and variable state issues.
  - The notebooks' interactive nature makes them very attractive for data analysis and tasks with a predominant visualisation component.
- The analysis of voice similarity is broken down into discrete steps and implemented as stand-alone modules. Each module will generate one or more results that are, in most cases, based on the results generated by another module.
- Modules are stringed logically together to perform all the tasks necessary in this project, including the download of the data and model.
- Data is manipulated only using code. The whole project can be executed without any manual intervention whatsoever.
- A simple logging tool was created so all modules can inject information about their execution. This logger is also used by a timer decorator that was created to measure the duration of the various modules of this project.

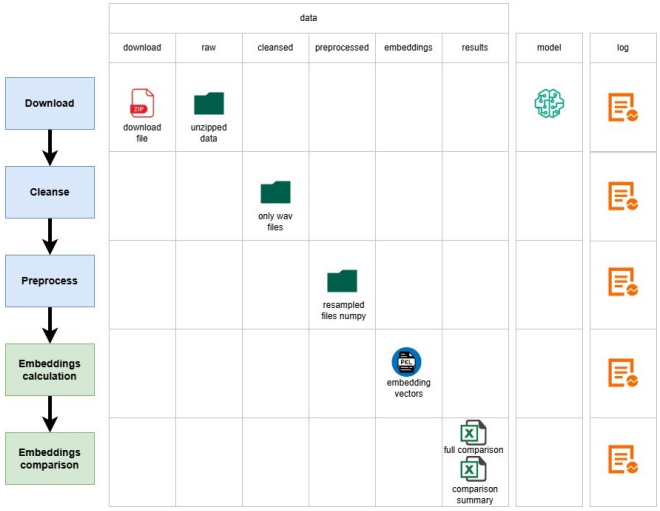

Fig. 3.

Embedding Extraction Using the WavLM Model

I used the pre-trained `microsoft/wavlm-base-plus-sv` model, downloaded locally from Hugging Face using the `transformers` library. The model was loaded in evaluation mode and used to generate speaker-level embeddings through

`WavLMForXVector`. Input waveforms were prepared via the associated `Wav2Vec2FeatureExtractor`, ensuring correct sampling rate and tensor formatting.

Audio Preprocessing

Only audio files matching the pattern `shortpassage*.wav` were retained from the ABI-1 Corpus. Each file was resampled to 16 kHz using `torchaudio.transforms.Resample` and saved as NumPy arrays (`.npy`) for efficient processing. This ensured compatibility with the WavLM model while reducing preprocessing overhead during embedding extraction.

Aggregation and Similarity Computation

Each speaker's embeddings were aggregated by concatenating all available utterance-level vectors. Pairwise cosine similarities were computed for:

- Intra-speaker comparisons (same speaker, different recordings)
- Inter-speaker comparisons (different speakers)

The `torch.nn.CosineSimilarity` function was used to calculate similarity values. These were saved into both raw and summary CSV files for further evaluation.

Dimensionality Reduction and Visualization

To analyze speaker embedding separability, I plotted cosine similarity distributions using `seaborn` histograms and KDE curves. Separate visualizations were generated for:

- Same-speaker pairs
- Different-speaker same-gender pairs
- Cross-gender speaker pairs

Range plots were used to highlight typical similarity intervals, offering visual guidance on threshold selection.

Threshold Selection and Accuracy Evaluation

Several thresholds were tested to distinguish between "same" and "different" speaker pairs. Using these thresholds, identification accuracy was computed as the proportion of correctly labeled pairs over total comparisons. Descriptive statistics and distribution plots supported the choice of an optimal threshold that maximized true positives while minimizing false positives.

Confusion Matrix Analysis

Confusion matrices were generated at varying thresholds to measure classification performance in terms of true positives, false positives, true negatives, and false negatives. These matrices helped identify critical error patterns and assess the system's robustness under different decision boundaries.

Experimental Setup and Execution Pipeline

The full experiment was orchestrated using the script `execute_pipeline.py`, which sequentially executes:

1) Dataset and model download
2) Data cleansing and preprocessing
3) Embedding extraction
4) Similarity computation and analysis

Speaker IDs followed a standardized `Accent-Gender-SpeakerID` format and were later expanded for analysis of accent, gender, and individual trends.

IV. Evaluation and results

V. Analysis and discussion

VI. Conclusion and future work

Generative AI