Kai Heng Gan

Exploring Unlabelled Speaker Recognition

Abstract

This document explores unsupervised speaker recognition using a dataset of microphone recordings without speaker labels. The primary objective is to group recordings by speaker identity without any ground truth annotations. To achieve this, we leverage ECAPA-TDNN speaker embeddings combined with various clustering algorithms (HDBSCAN, K-Means, Spectral Clustering). Our goal is to assess the feasibility of speaker separation and identity inference based purely on vocal characteristics embedded in vector space, validated via internal metrics like Silhouette Score and Davies-Bouldin Index.

1 Data Exploration and Analysis

Working with unlabelled audio data for speaker recognition presents unique challenges. Without any ground-truth labels, we cannot directly train or evaluate a supervised model, so we must rely on inherent patterns in the audio to differentiate speakers. The dataset of ~200 WAV recordings likely contains a wide variety of speaker characteristics (e.g. differences in pitch, timbre, accent, speaking speed) and possibly varying recording conditions (background noise, microphone types). Identifying distinct speakers in unlabelled data means discovering consistent acoustic features that differ from speaker to speaker, which can be subtle. Moreover, we must be cautious about factors like silence or multi-speaker segments in the recordings, as these can complicate analysis.

Initial exploration steps would include both listening and visual analysis of the audio to understand its properties and potential challenges:

- **Listening to Samples:** By auditioning a subset of the recordings, we can qualitatively gauge how different the voices sound and note any obvious similarities. This helps identify factors like gender (e.g. male vs female voices), accent or speaking style, and whether some recordings contain background noise or multiple speakers. Early listening can reveal if some voices are extremely similar (which foreshadows clustering difficulty) or if there are issues like loudness differences or significant silence in the files.
- **Visualizing Waveforms:** Plotting the waveform of audio files can show differences in amplitude envelope and speaking rate. For example, one speaker may have consistently higher energy or a distinctive rhythm of speech (pauses between phrases). Large sections of silence or differing loudness levels might become apparent in waveforms. While waveforms alone may not directly distinguish speakers, they can highlight anomalies (like one recording being significantly noisier or quieter than others).
- **Spectrogram Analysis:** Generating spectrograms for sample recordings is a powerful way to visualize frequency content over time. A spectrogram displays time on the horizontal axis and frequency on the vertical axis, with intensity (amplitude) indicated by color brightness.[1] Different speakers produce different spectral patterns – for instance,

the resonant frequencies of the vocal tract (formants) will vary with each individual's voice. By comparing spectrograms, we might observe that one speaker has a higher fundamental frequency (higher-pitched voice) or distinct formant structures compared to another. These visual cues can hint at which features might be useful for distinguishing speakers (e.g. formant positions, frequency distribution of energy). If the recordings are long, we might also see segments of silence or different speakers in the same spectrogram (which would indicate the need for voice activity detection or segmentation).

- **Statistical Feature Examination:** We can compute basic audio features for each recording to see broad differences. A key feature set is **Mel-Frequency Cepstral Coefficients (MFCCs)**, which are widely used to represent the spectral characteristics of speech. MFCCs compress the audio's frequency information (the shape of the spectral envelope) into a small number of coefficients related to how humans perceive sound.[2] Importantly, MFCCs capture vocal tract characteristics of the speaker and are largely independent of the spoken content. By computing, say, the average MFCC vector or the distribution of MFCCs over time for each recording, we might notice clustering or separation even without labels (e.g. one group of recordings might have consistently higher values in certain MFCC dimensions, indicating a similar vocal tract shape among those speakers). Other features like pitch (fundamental frequency), speaking rate, or formant frequencies could also be extracted – for example, we might find some recordings have a much higher average pitch (likely female speakers) versus others. This feature analysis can indicate which characteristics vary most across the dataset.
- **Duration and Silence Patterns:** It's important to check if all recordings contain a single continuous speech segment or if some have long pauses/silences. If a recording has significant silence or multiple speaking turns, we will consider applying **Voice Activity Detection (VAD)** to isolate regions of active speech. VAD can help ensure we analyze only the parts where someone is speaking, ignoring silence or background noise. During exploration, detecting if any files have more than one speaker (e.g. an interview or overlapping dialogue) is crucial – if so, those would need to be handled differently (such as by splitting by speaker turns) since our goal assumes one speaker at a time.
- **Visualization of Feature Space:** As a more advanced exploratory step, we could apply dimensionality reduction to the extracted features or embeddings to visualize potential clustering. For example, taking an initial embedding (like the mean MFCCs or i-vectors if we derive them) for each recording and using PCA or t-SNE to project them to 2D, we might see natural groupings emerge. If distinct clusters appear in such a plot, it's a strong indication that the speakers are differentiable via those features. Conversely, heavy overlap would warn that more sophisticated features or algorithms are needed.

From this exploration, we expect to learn about the data characteristics and challenges. We might discover, for instance, that recordings have varying quality (some noisy, some clean), or that a few speakers have very similar vocal profiles. We might also find simple patterns – e.g., perhaps half the recordings are low-pitched and half high-pitched, suggesting gender grouping, but within those, finer distinctions are needed. Identifying such patterns or the lack thereof will guide the choice of features and algorithms in the next steps. Overall, initial data analysis lays the groundwork by revealing how separable the speakers might be with basic means and what hurdles (like noise or silence) we must account for in an unsupervised solution.

2 Proposed Solution(s) and Justification

In the absence of labels, the task becomes an unsupervised clustering problem: we want to group the recordings such that each group corresponds to a single speaker. The central idea of the proposed solution is to leverage speaker embeddings – fixed-length vector representations of speech segments that capture a speaker's unique vocal characteristics – and then cluster these embeddings to identify distinct speakers. By using a pretrained speaker recognition model to generate embeddings, we inject prior knowledge of how human voices differ, which is extremely useful given we have 200 unknown speakers to separate.

Speaker Embeddings for Unsupervised Recognition: Modern speaker recognition systems use neural network-based embeddings to represent voices. Two well-known approaches are **x-vectors** and **ECAPA-TDNN** embeddings:

- **x-vectors:** An x-vector system is a neural network trained on a large dataset of speakers to discriminate between them. It produces a vector (typically 512-dim) for an input utterance, capturing the speaker's identity in that fixed-size vector. Essentially, x-vectors are embeddings extracted with a deep neural network, designed such that same-speaker utterances map to nearby points in embedding space.[3] This approach was pioneered to replace earlier i-vector methods, and x-vectors have become a fundamental component of state-of-the-art speaker diarization and recognition systems. The network usually operates on acoustic features (e.g. MFCC frames) and aggregates information across time (using pooling) to produce one embedding per recording. An **x-vector embedding space** has the property that distance (cosine or Euclidean) correlates with speaker similarity – i.e. two clips from the same person will yield x-vectors that are closer than those from different people. This makes them ideal for clustering: we expect 200 tight clusters (one per speaker) in the x-vector space. In fact, research systems cluster x-vectors to achieve speaker diarization,[4] demonstrating that they can be grouped effectively to separate speakers.
- **ECAPA-TDNN:** This is an advanced neural embedding model (Emphasized Channel Attention, Propagation and Aggregation in TDNN). It builds on the x-vector/TDNN architecture with enhancements like Squeeze-Excitation blocks and attentive statistical pooling. The result is an embedding that is even more discriminative and robust, especially under challenging conditions (noise, reverberation, distant mic). ECAPA-TDNN has shown impressive performance in speaker verification, providing robust speaker embeddings even in noisy or far-field scenarios.[5] For our purposes, using a pretrained ECAPA-TDNN (such as one trained on VoxCeleb) would likely yield high-quality vectors for each audio file that emphasize speaker traits and minimize effects of noise or channel. ECAPA embeddings are typically 192-dimensional and have become a new state-of-the-art in speaker recognition tasks.

Using either of these embedding methods (or similar pretrained models), the solution would proceed as follows: extract an embedding for each recording, then apply a clustering algorithm to the collection of embedding vectors. The expectation is that embeddings naturally form clusters corresponding to individual speakers because the model has been trained to make same-speaker audio points close in the vector space. This approach is fully unsupervised with respect to our data (we don't need any labels from the 200-speaker dataset), yet it leverages powerful

discriminative features learned from other data. In essence, we transform the raw audio into a speaker feature space where the clustering task is tractable.

Why use pretrained embeddings? Without labels, a purely data-driven approach would be to find structure in raw features, but voices are complex signals. Pretrained speaker models provide a strong prior: they bring in knowledge of what features distinguish voices (for example, formant patterns, pitch range, speaking style) as learned from thousands of speakers in their training set. This drastically improves the chances of separating 200 speakers correctly. If we attempted to cluster the raw audio or basic features directly, we might run into issues: MFCCs or spectrograms of different speakers can still be quite similar, especially if some speakers have similar vocal characteristics. A neural embedding intentionally amplifies inter-speaker differences and suppresses intra-speaker variation (due to content or noise). Empirically, approaches that cluster speaker embeddings (x-vectors) have been very successful in diarization challenges. By using such embeddings, we increase accuracy because the distance in embedding space is a meaningful indicator of speaker identity.

Additionally, using embeddings is computationally efficient and scalable. Instead of dealing with extremely high-dimensional raw data (e.g. hundreds of frames of MFCCs per second * per recording), we reduce each recording to one vector. Clustering a few hundred vectors (for ~200 recordings) is much easier and faster than clustering potentially millions of frame-level feature vectors. The embedding extraction itself is also efficient – models can process audio in real-time or faster – and this step can be parallelized if needed.

Clustering approach: Once we have embeddings, we need to cluster them into (approximately) 200 groups. There are multiple algorithms we could employ:

- A straightforward option is **k-means clustering** with *k=200* (since we expect 200 speakers). K-means will attempt to find 200 centroids and assign each embedding to the nearest centroid. This could work as an initial step, but k-means assumes clusters are spherical and of similar size, which may not hold true for speaker embeddings. Some speakers might have more recordings than others, or some clusters might naturally be tighter than others. K-means also requires specifying k exactly; if our data actually has fewer than 200 speakers active (or if some speakers don't appear), k-means could over-partition, or if a speaker's embeddings form a non-spherical shape, it might be split across centroids.
- A more flexible and often better approach for speaker data is **hierarchical clustering**. **Agglomerative Hierarchical Clustering (AHC)** can be used with a suitable distance metric (for example, cosine distance between embeddings, or PLDA score if using a statistical model).[4] We could start with each recording as its own cluster and iteratively merge the closest clusters until we reach 200 clusters. This method, especially when combined with a scoring metric like PLDA (Probabilistic Linear Discriminant Analysis) tuned for speaker verification, was traditionally used in diarization systems. It has the advantage of not assuming equal cluster sizes. We would, however, need to decide on a stopping criterion (merge until 200 clusters remain, or until a distance threshold is exceeded).

- **Density-based clustering** is another appealing option. **HDBSCAN (Hierarchical DBSCAN)** is well-suited for our scenario because it can automatically determine the number of clusters and handle clusters of varying density. One of HDBSCAN's strengths is the ability to find clusters of arbitrary shape/size and label outliers as noise.[6, 7] In an embedding space, it's possible some speaker clusters are very tight while others are more diffuse (perhaps due to more variability in that speaker's recordings). HDBSCAN can accommodate this by finding dense regions (tight clusters) and not forcing every point into a cluster if something doesn't fit well. For example, if one recording is very noisy, it might not clearly belong with its true speaker's other recordings – HDBSCAN could flag it as an outlier rather than incorrectly clustering it. This algorithm would not require knowing there are 200 speakers a priori, though we could expect it to find roughly 200 clusters if the data naturally separates.
- **Spectral clustering** is another technique that has proven effective in speaker diarization research.[5] Spectral clustering treats each embedding as a node in a graph and uses a similarity matrix (e.g., cosine similarity between all pairs of embeddings) to partition the graph into communities (clusters). It's powerful when cluster boundaries are not well separated in a simple metric space but can be teased apart by graph connectivity. We could construct a N x N similarity matrix for the N recordings, convert it to a graph Laplacian, and compute the top eigenvectors to find clusters. Spectral clustering often yields good results in difficult clustering problems and could capture more nuanced speaker groupings, especially if some speakers are marginally close to each other in the embedding space.

In practice, we might try multiple clustering methods and compare results, possibly even combining them (e.g., use AHC to estimate cluster count then refine with HDBSCAN). The fully unsupervised nature allows experimentation – since we have no labels, we'd validate clusters with internal metrics or manual checks as described later.

Alternative approaches considered: The primary approach described uses pretrained embeddings (a **"knowledge-based" unsupervised method**). There are other unsupervised strategies one might consider:

- *Traditional i-vector + clustering:* Before deep learning, speaker recognition often relied on **i-vectors** (intermediate-size vectors derived from a factor analysis of GMM mixture weights). We could train a **Universal Background Model (UBM)** on all the audio (which doesn't require labels), and then extract a low-dimensional i-vector for each recording. These i-vectors could then be clustered similarly. This approach is fully unsupervised with respect to speaker labels, but in practice x-vectors have largely supplanted i-vectors due to better accuracy. i-vectors might capture unwanted variabilities (e.g., channel noise) along with speaker info, whereas x/ECAPA vectors are more directly optimized to separate speakers. Still, i-vectors would be a feasible baseline approach if deep models were not available.
- *Clustering on raw features:* One could attempt clustering using statistical features directly (e.g., cluster on the average MFCCs or pitch of each recording). This is simpler but likely insufficient to discriminate 200 individuals. It might separate broad categories (for example, you might get a cluster of all high-pitched voices versus low-pitched

voices), but it won't capture the richer distinctions needed for individual identities. We mention it for completeness, but it would not be very effective.

- *Unsupervised representation learning:* A more research-oriented approach would be to train a model on the unlabelled data itself to learn embeddings, using methods like autoencoders or contrastive learning (e.g., clustering-based self-learning, where you iteratively refine pseudo-labels). For instance, one could use a technique where the model tries to cluster the data in embedding space while simultaneously learning the embedding (a bit like clustering-friendly autoembedder [8,9]). This is quite complex and beyond the scope of a short take-home assignment, but it could be a future direction. It essentially means building a speaker embedding from scratch using the unlabelled data (leveraging the assumption that there are 200 distinct classes to discover). Given the time and complexity, using an existing pretrained model is a much more straightforward and reliable choice.

In summary, the proposed solution is to use a pretrained speaker embedding model (such as **x-vector** or **ECAPA-TDNN**) to convert each audio recording into a speaker-discriminative vector**,** and then apply an unsupervised clustering algorithm (like **HDBSCAN** or **spectral clustering**) to group these vectors by speaker. This approach is justified by state-of-the-art practices in speaker recognition: it uses proven techniques that can differentiate a large number of speakers without labels, and it takes advantage of the fact that voices can be represented in a machine-learnable space where simple clustering reveals speaker identities**.** By leveraging powerful embeddings, we significantly improve the accuracy of unsupervised speaker recognition in this 200-speaker problem, compared to naive methods.

3 Implementation Strategy (Conceptual)

Implementing the above solution involves several stages: data preprocessing, feature extraction to obtain embeddings, clustering, and evaluation of the results. Below we outline each of these steps at a high level:

1. Data Preprocessing

Before extracting features or embeddings, we need to clean and prepare the audio data to ensure that we're feeding consistent, quality inputs to our models:

- **Resampling and Formatting:** Ensure all audio files have the same sampling rate (e.g., 16 kHz) and format (mono channel). If recordings come from different devices, they might have different sample rates/bit-depths; standardizing them eliminates one source of variation. Most pretrained models expect 16 kHz mono audio, so we would resample as needed.
- **Voice Activity Detection (VAD):** Apply a VAD to each recording to detect and isolate regions containing human speech, removing silence and background noise. This helps to trim out long silences or low-level noise. For example, if a file has a few seconds of someone speaking and then trailing silence, VAD can segment and isolate the spoken portion. We can use an unsupervised VAD method (like an energy threshold or more advanced algorithms like `rVAD` which is robust and unsupervised) to automatically remove non-speech segments. By focusing on active speech, we improve the quality of

features extracted and avoid clustering silence (which otherwise could wrongly suggest similarity between recordings that merely share silence).

- **Noise Reduction:** If our exploration found significant background noise in some recordings, we might apply noise reduction or filtering. A simple approach is a high-pass filter to remove low-frequency hum, or spectral noise gating if consistent noise is detected. The goal is to make the speaker's voice more prominent in the features. However, we must be careful – aggressive noise reduction could distort voice characteristics. Pretrained embeddings like ECAPA are designed to be robust to noise, [5] so this step is optional and should be used only if noise is truly hindering the feature extraction.
- **Volume Normalization:** Normalize the amplitude of recordings to a standard level. Speakers who were recorded louder vs softer should ideally be brought to a comparable loudness so that the embedding model isn't influenced by volume differences (which carry no speaker information). This can be done by peak normalization or better, loudness normalization (e.g., adjust to a target LUFS level).
- **Segmentation (if needed):** If any recording is known to contain multiple speakers or multiple distinct speech segments, consider segmenting it before embedding. For instance, if one file has two people talking (even if not overlapping), we'd ideally want to separate those portions and treat them individually in the clustering. This is more relevant to speaker diarization; in our case, if we assume each recording is a single speaker, we may not need to segment by speaker, but VAD might still segment long silences out.

By preprocessing, we ensure that the subsequent feature extraction yields representations that reflect speaker traits as purely as possible, with minimal interference from silence or noise.

2. Feature Extraction and Embedding Generation

Next, we extract features from the audio and feed them into a model to get speaker embeddings:

- **Low-Level Feature Extraction:** We start by converting audio signals into a time-frequency representation suitable for the embedding model. Common choices are frames of MFCCs or log Mel-spectrogram. For example, we might extract 23-dimensional MFCCs from 25 ms frames of audio (with 10 ms hop) as inputs to an x-vector model, since x-vector networks are often trained on MFCC features. Similarly, an ECAPA-TDNN model might take a Mel-spectrogram or filterbank features as input. If using a tool or library, much of this is handled internally, but it's important to apply the same feature extraction that the pretrained model expects (including any mean normalization, etc.). MFCCs are a good summary of the vocal tract characteristics, which is why most speaker embedding models use them or related spectral features as input. We should also apply any feature normalization required (many systems perform cepstral mean normalization to remove channel effects).
- **Embedding Inference:** Using a pretrained speaker embedding model, compute the embedding for each recording. There are readily available models (from frameworks like PyTorch/SpeechBrain, TensorFlow, or Kaldi) for both x-vectors and ECAPA. For instance, using **SpeechBrain's ECAPA** model, we would load the model and call an `encode_batch` function on the audio waveform to get a 192-D embedding vector. Each recording (or each VAD-segmented speech region if we split) will yield one

embedding vector. It's important to note that if a recording is long, we might get a better representation by breaking it into a couple of chunks and averaging their embeddings – but for simplicity, one embedding per file might suffice if each file isn't too long. The result of this step is a set of N vectors (where N is the number of audio segments considered – likely N ≈ number of recordings after preprocessing).

- **Optional – PCA/LDA on embeddings:** In some speaker ID pipelines, after obtaining embeddings (like x-vectors), an optional step is to apply dimensionality reduction or projection (e.g., Principal Component Analysis or Linear Discriminant Analysis trained on an external dataset) to reduce noise in the embedding space. For example, x-vectors (512D) are often reduced to 200D using LDA trained on a labeled dataset to emphasize speaker-separating dimensions. Since we may not have a labeled set for these 200 speakers, we likely won't do LDA specifically for them, but using a pretrained model's built-in dimensionality reduction (some toolkits include an LDA or PLDA transform from their training) could be beneficial. This step can enhance clustering by removing dimensions that mostly carry noise or session variability.
- **Similarity Measure Preparation:** Decide on a distance or similarity measure for clustering. Typically, cosine similarity is used for speaker embeddings (treating each vector as a direction in space). We can compute the cosine distance (1 – cosine similarity) between any two embeddings to feed into distance-based clustering algorithms. For spectral clustering, we would compute a full similarity matrix (cosine similarities between all pairs). For algorithms like HDBSCAN, we just need a distance metric to define density; Euclidean distance in the raw embedding space could serve, or we could use cosine distance. In practice, cosine and Euclidean are equivalent for normalized embeddings (one can normalize all vectors to unit length so that Euclidean distance correlates with cosine).

After this step, we have a numeric representation for each recording that is ready for the clustering algorithm. Each embedding is hopefully a point in a 192 or 256 or 512-dimensional space that encapsulates that speaker's vocal characteristics.

3. Clustering the Embeddings

With all embeddings in hand, we perform **unsupervised clustering** to group them by speaker identity:

- **Clustering Algorithm:** We will choose an algorithm as discussed in the Proposed Solution section. To recap two strong options: **HDBSCAN** and **Spectral Clustering** are front-runners due to their flexibility. Using HDBSCAN, we would not specify the number of clusters; instead, we set parameters like `min_cluster_size` (which we might set to, say, 2 or 3, assuming at least a few recordings per speaker in general) and let it find dense clusters. HDBSCAN will output labels for each point or mark it as noise. We expect it to find around 200 clusters, but it might find fewer or more if some speakers are very close together or some recordings are outliers. Using spectral clustering, we would likely need to specify the number of clusters (we can set it to 200 since we know the number of speakers). Spectral clustering would involve constructing a similarity matrix S where S_{ij} is the cosine similarity between embedding i and j, then computing the graph Laplacian and its top eigenvectors, etc. This can be done with libraries like scikit-learn easily. Agglomerative clustering with a set number of clusters (200) is also an

option and is actually quite similar to what spectral clustering will do if combined with a cosine/PLDA distance.

- **Practical execution:** We might try multiple algorithms: for instance, run HDBSCAN and see how many clusters it finds; if it finds significantly not 200, we may adjust parameters or compare with a fixed 200 solution. We could also run k-means with 200 just to have a baseline grouping. **Agglomerative Hierarchical Clustering (AHC)** can be tried with average linkage using cosine distance, cutting the dendrogram at 200 clusters. Each method might yield slightly different groupings; if one method's result is clearly more coherent (by internal metrics), we'd favor that.
- **Recording cluster assignments:** The output of clustering will be a label for each embedding (e.g., cluster 1, cluster 2, ..., cluster M). Ideally $M=200$ if everything goes perfectly and every speaker formed one cluster. We need to capture these assignments for evaluation. If using HDBSCAN, there is a possibility of some points labeled as -1 (noise/outlier); we would then have to decide how to handle those – maybe assign them to the closest cluster or leave them as "unknown speaker" for manual inspection. In a perfect scenario, there are no outliers, and each recording is assigned to a cluster. We should also be mindful of cluster sizes – if we find a cluster has an implausibly large number of recordings (say 20 recordings in one cluster, when average should be ~1 recording per speaker if everyone spoke roughly equal amounts), that could mean the algorithm merged multiple speakers together. Likewise, very tiny clusters (1 recording by itself) could be a correct singleton (if that speaker only has one recording) or could mean that recording was very different (perhaps noise).
- **Refinement:** After initial clustering, we could consider a refinement step. For example, if we suspect an over-merged cluster, we could take that cluster's embeddings and re-run a finer clustering just on that subset to see if it naturally splits. Conversely, if an algorithm produced some very small clusters or singletons that we suspect are actually part of another cluster (maybe due to noise causing one recording to look out-of-place), we could manually or programmatically examine distances and possibly merge those. Since this is a fully unsupervised scenario, such refinement would be heuristic. An advanced refinement might involve using the cluster assignments to train a lightweight classifier or do a round of model adaptation – e.g., use pseudo-labels (cluster IDs) to fine-tune the embedding model slightly on this data, then re-embed and re-cluster. This could improve separation if done carefully, but it also risks reinforcing any mistakes in clustering. For the scope here, it may be unnecessary, but it's a consideration if initial clustering wasn't satisfactory.

The end result we aim for is a partition of the recordings into 200 clusters, which essentially gives each recording **a** predicted speaker label (cluster label). At this point, we have "recognized" speakers in the sense that we can label recordings that belong to the same person with the same ID (even if that ID is just a cluster index with no human-readable name). The final part is to evaluate and verify how well this clustering likely did.

4. Evaluation of Clustering (Without Ground Truth)

Evaluating an unsupervised speaker clustering is tricky without true labels, but we can employ several strategies to gauge the quality of our solution:

- **Silhouette Coefficient:** This is an internal validation metric that assesses how well-separated the clusters are. The silhouette value for each point measures how close it is to other points in its cluster compared to points in the nearest other cluster.[10] It ranges from -1 to +1, where a higher value means the point is deep inside a well-matched cluster, and values near 0 or negative mean it might be in the wrong cluster or ambiguous. We can compute the average silhouette score across all recordings to get a sense of overall clustering quality. A high average silhouette (e.g., > 0.5) would indicate reasonably separated speaker clusters. If we see a lot of negative silhouette values, that flags that some recordings might actually be closer to a different cluster than the one they were assigned – a sign of potential mis-clustering.
- **Cluster Separation and Compactness:** Related to silhouette, we can also look at metrics like Davies-Bouldin Index or Calinski-Harabasz Index, which consider the scatter within clusters vs separation between clusters. These give a single number assessment; lower Davies-Bouldin or higher Calinski-Harabasz generally means better-defined clusters. They are useful to compare different clustering outcomes (e.g., if we try multiple algorithms or parameter settings, these metrics help pick the best one).
- **Cluster Size Distribution:** We know in theory each speaker might have a different number of recordings, but if our dataset was roughly one recording per speaker (it's not explicitly stated, but "over 200 recordings, 200 speakers" could mean ~1 each, or some have more), then we'd expect cluster sizes to be fairly small or equal. If we see a cluster containing, say, 10% of the data, that's suspicious. We would review such a cluster by listening to a few recordings from it – perhaps the algorithm merged several people. Similarly, many singletons might indicate over-separation (or just that many speakers had one recording – which could be true). While not a direct metric, checking the cluster size histogram against expectations can reveal issues.
- **Manual Spot-Checking:** One reliable (though time-consuming) evaluation is to manually inspect a sample of clusters. We could randomly pick a few clusters and listen to a snippet from each recording in that cluster to verify if they indeed sound like the same person. If our clustering is good, every cluster we sample should contain homogenous speaker voice. If we find a cluster where the voices clearly differ, that cluster is a mistake (over-clustering). Likewise, if we find the same voice appearing in two different clusters in our sampling, that indicates an under-cluster (the same speaker was split). By doing enough spot checks, we can build confidence in the clustering. This qualitative evaluation is especially useful given no ground truth – our ears (or those of someone familiar with the speakers) are the ultimate judge. Even if we can't do this for all 200 speakers, checking, say, 10-20 clusters can reveal if there are systemic errors.
- **Speaker Verification Model Check:** As an optional evaluation, since we have access to speaker embedding models, we can use them in verification mode. For example, take pairs of recordings that our clustering says are the same speaker (within a cluster) and run a speaker verification scoring (cosine similarity or a pretrained verification head) on them. We'd expect high similarity scores for true same-speaker pairs. If many pairs within clusters yield low similarity (below a typical threshold for same speaker), that could indicate cluster errors. Conversely, take some pairs from different clusters and verify the model judges them as different. This kind of evaluation uses the model as a

proxy for "ground truth" consistency. Essentially, we trust that the embedding model would agree if we got it right.

- **Silhouette per cluster and outlier detection:** We can compute the silhouette for each recording as mentioned; recordings with particularly low or negative silhouette could be investigated individually. Often these will be edge cases – perhaps a noisy recording that doesn't fit well anywhere, or a speaker who is very close in voice to another. These could be flagged for manual review or perhaps reclassified if we find they truly belong with another cluster.

Given we lack true labels, our evaluation will focus on these internal metrics and qualitative assessments to ensure the clustering is sensible. If the silhouette score is strong and manual checks on a handful of clusters look good, we can be reasonably confident in the solution. If not, we would iterate adjust the clustering parameters or even reconsider the feature extraction (for example, if two speakers are always confused, maybe the embedding isn't capturing a nuance that we need – perhaps we could incorporate an additional feature like vocal jitter or something, but that's an edge case).

4 Challenges and Considerations

This unsupervised speaker recognition task comes with several challenges and inherent limitations, as well as assumptions we need to make about the data:

- **No Ground Truth for Evaluation:** As discussed, one major difficulty is knowing how well we're doing. Without labeled speakers, we can't calculate accuracy or error rates in the usual way. This means we have to rely on indirect validation. A consequence is that some errors might go unnoticed – for example, if two very similar-sounding people get clustered together, only careful manual checking might catch it. We mitigate this by using internal metrics (silhouette, etc.) and as much manual verification as feasible. In a real scenario, one might involve a human-in-the-loop to spot-check clusters or perhaps identify a few anchor samples for each speaker to validate clustering (semi-supervised hints), but strictly unsupervised, it's hard to be 100% sure. We assume for this task that a good clustering metric score implies a successful outcome.
- **Large Number of Speakers (200):** Differentiating 200 speakers is a challenge even for supervised systems; doing it unsupervised is especially daunting. With so many speakers, the likelihood that some voices are very similar is high. Some speakers might be siblings or have the same accent and voice type, making their audio almost indistinguishable in certain feature spaces. The embedding model might place those speakers' voices very close together. **Potential overlap in speaker characteristics** means clusters could blur – one cluster might accidentally contain two people with near-identical voices. We must recognize this risk. Clustering algorithms like HDBSCAN can sometimes split dense clusters if there are subtle density variations, but it might not always separate two look-alike speakers. One mitigation strategy is to use the knowledge of 200 speakers to our advantage: if our clustering yields, say, 190 clusters, we know we likely merged some distinct speakers (and we could attempt to increase sensitivity to reach 200). Conversely, if we get 210 clusters, perhaps we over-split someone. The assumption here is that indeed all 200 speakers are present and each forms a unique cluster – if the reality were that

only, say, 180 of the 200 spoke in the dataset and 20 didn't, then any clustering expecting 200 will overcluster (but the problem statement suggests all are present).

- **Similar Voices and Overlap:** Expanding on overlap of characteristics – this is arguably the hardest part of speaker recognition. Two different people can have uncannily similar voices. If our embedding cannot differentiate them, those two will end up in one cluster. There's not a perfect solution for this without additional information. However, using high-quality embeddings (ECAPA) gives us the best shot, as they are state-of-the-art in capturing subtle differences. Another consideration is if any recordings have **overlapping speech (two people talking at once)** – since these are microphone recordings, that could happen if it was, for example, a meeting recording. Overlapping speech segments can confuse embedding extractors (they might produce a single vector representing a mixed voice which doesn't cleanly belong to either speaker's cluster). The task didn't explicitly mention overlapping speech, but if it were present, a possible mitigation is to use a diarization pre-processing to split overlapping voices or at least mark those segments as unusable. We will assume for simplicity that each recording is predominantly one speaker at a time (no significant overlap).
- **Noise and Channel Variability:** The recordings might have different background noises or channel effects (some could be recorded on a high-quality mic in a quiet room, others on a phone with background chatter). Noise and channel differences can reduce the similarity of recordings from the same speaker or increase similarity of different speakers (e.g., two different people both recorded with the same background noise might cluster together if the algorithm latches onto the noise pattern). We addressed this partly with preprocessing (noise reduction) and by choosing robust embeddings. ECAPA-TDNN's robust performance on distant recordings is a big plus here – it was explicitly designed with noise/reverb augmentation to not be fooled by such factors. We may also assume the dataset, being an assignment, is somewhat controlled (e.g., all recorded with similar microphones or conditions), but that might not be true. If we find clusters forming by background environment rather than voice, we'd need to compensate by either filtering out those aspects (perhaps focusing on vocal frequencies only) or training the model further with augmentation to suppress that. For now, we mitigate noise by preprocessing and by the inherent robustness of our chosen model.
- **Scalability:** In terms of computational scalability, 200 recordings is not large. But if each speaker had multiple recordings, say the total segments to cluster were in the thousands, we need to ensure our method scales. Extracting embeddings is linear in number of recordings – that's fine (modern models can process hundreds of hours relatively quickly especially with GPU acceleration). Clustering is usually the potential bottleneck: a spectral clustering will have to compute an N x N similarity matrix, which for, say, 1000 segments is 1000x1000 (1e6 entries) – completely manageable. Even 10,000 segments would be 100 million similarities, which starts to push memory but could still be done with efficient sparse techniques if needed. HDBSCAN is roughly O(N log N) for many cases and should handle thousands easily. Agglomerative clustering is $O(N^2)$ in worst case, but with N in the low thousands it's okay. So practically, the approach scales to moderately large datasets. If the number of speakers grew very large (say this approach needed to scale to *open-set* with unknown numbers or hundreds of thousands of samples), we might need to incorporate more efficient nearest-neighbor search methods or divide-

and-conquer clustering. But 200 speakers are within a range that standard methods can handle. We assume the dataset is on the order of a few hundred recordings total.

- **Labeling and Using the Results:** Another consideration: once we have clusters, each cluster is essentially an anonymous speaker ID. If the end goal is to "recognize" speakers in new recordings, we have basically created a model of each speaker in an unsupervised way. If a new recording comes in (from one of the 200 speakers), we could embed it and compare to cluster centroids to predict which cluster (speaker) it belongs to. This would be an **evaluation** of how well the clustering learned the speaker identities. Without ground truth, we can't fully test this, but it's something to consider – our clustered model could serve as a pseudo speaker recognition system for these 200 speakers. If that was required, we'd ensure to maybe compute cluster centroids (average embedding per cluster) and use those as references for identification. This is somewhat beyond the assignment scope but is a logical next step once clustering is done.
- **Assumptions about Data:** To make this problem tractable, we assume each recording contains one speaker at a time, and exactly one of the 200 known speakers. We assume every speaker has at least one recording in the dataset (no completely missing speaker). We also likely assume that the voices are mostly distinct (no pair of identical twins with identical voices, for example, which could be an impossible case). We assume the recordings have enough speech content to get a reliable embedding – if some recordings are just 1 or 2 seconds long, embeddings might be less accurate, possibly causing miscluster. Hopefully the data has reasonably long samples (e.g., 10-30 seconds each) to capture the speaker's characteristics. If not, a possible mitigation is to concatenate two short utterances if we know they are same speaker (not possible since unlabelled) or at least be aware that very short utterances might be outliers.

In summary, while completely unsupervised speaker recognition is challenging – it's essentially the problem of speaker diarization applied across disjoint recordings – our approach tries to minimize risks by using proven techniques (robust embeddings, flexible clustering algorithms) and by carefully validating the outcomes. We addressed potential pitfalls like similar-sounding speakers (which might require manual intervention in the worst case), noise differences (handled by model robustness and preprocessing), and the difficulty of evaluation (using silhouette scores and sample checking as proxies). With these considerations in mind, the proposed method should provide a solid solution for differentiating and "labelling" the 200 speakers in the dataset without any prior training labels, effectively creating an unsupervised speaker recognition system.

5 Experiments & Results

5.1 Dataset Preparation

The experiments are based on the Audio MNIST corpus (60 speakers, 30 000 one-second WAV files of the digits 0 – 9).

For every speaker the ten digit files belonging to the same repetition were concatenated in natural order (0 → 9) to form one longer utterance, e.g.

$$rep\text{-}00: \ 01\_0\_01\_0.wav \longrightarrow 01\_9\_01\_0.wav \ \rightarrow \ 01\_combined\_0.wav$$

$$rep\text{-}01: \ 01\_0\_01\_1.wav \longrightarrow 01\_9\_01\_1.wav \rightarrow 01\_combined\_1.wav$$

$$\vdots$$

$$rep\text{-}49: \ 01\_9\_01\_49.wav \longrightarrow 01\_9\_01\_49wav \rightarrow 01\_combined\_49.wav$$

50 such "count-to-9" clips per speaker × 60 speakers = 3 000 combined recordings (≈ 6 s each after concatenation).

The motivation for cross-digit concatenation is two-fold:

1. **Phonetic diversity** – a single clip now spans all vowel/consonant classes, exposing the embedding model to a richer set of articulatory gestures than any one-second digit snippet would provide.
2. **Robust speaker modelling** – longer, varied speech segments enable the ECAPA-TDNN (CAPA-TDNN) network to average out digit-specific formant positions and capture speaker-invariant characteristics, which in turn yields tighter clusters in the embedding space.

5.2 Processing Pipeline

Stage	Description
Resampling & normalization	All audio resampled to 16 kHz mono and peak-normalized to –0.45 dBFS.
Voice-Activity Detection	Energy-based VAD removes residual silence introduced by concatenation.
Embedding extraction	Each clip passed through a **pre-trained ECAPA-TDNN** network (SpeechBrain release); 192-D L2-normalised speaker embeddings obtained.
Clustering	Three unsupervised algorithms applied to the embedding matrix $\mathbf{X} \in \mathbb{R}^{3000\times192}$: 1. *k*-Means *($k = 60$, $n_{init} = 20$, random_state = 42)* 2. **HDBSCAN** *(min_cluster_size = 2)* 3. **Spectral clustering** *(k-nearest-neighbour affinity, k = 60)*.
Internal validation	Silhouette coefficient (higher = better) and Davies-Bouldin index (lower = better) computed on the embedding space.
Auditory spot-check	Five random clips from ten random clusters were auditioned to subjectively verify within-cluster consistency.

5.3 Evaluation Metrics

Metric	Formula / intuition	Why it matters for speakers
Silhouette (S)	Intra-cluster cohesion vs nearest-cluster separation, $S \in [-1,1]$	High $S \Rightarrow$ embeddings of the same voice lie close and far from others.
Davies-Bouldin (DB)	Mean ratio of cluster scatter to inter-cluster distance	Low DB means clusters are both compact and well-separated.

5.4 Results

Algorithm	# clusters detected	Silhouette ↑	Davies-Bouldin ↓
k-Means (k = 60)	60 (by definition)	**0.51**	**0.824**
HDBSCAN	60 clusters + 0 outliers	**0.51**	**0.824**
Spectral clustering (k = 60)	60 (by definition)	−0.024	2.614

* Cluster map can be found in [Exploring-Unlabelled-Speaker-Recognition/data/embeddings/cluster_map.json]

Observations

- Both k-Means and HDBSCAN converge to an identical partition of the ECAPA-TDNN space (confirmed by matching cluster labels in the JSON map and equal metric scores).
    - Silhouette ≈ 0.51 and DBI ≈ 0.82 indicate well-separated, compact speaker clusters for an unsupervised task of this scale.
- HDBSCAN produced no outliers, suggesting the concatenation strategy delivered consistently informative speech segments.
- Spectral clustering collapsed several speakers into mixed communities (negative silhouette, high DBI), confirming that a kNN-graph on this embedding set is less discriminative than centroid-based density methods.

Conclusion

This document explored the feasibility of unsupervised speaker recognition using a dataset of unlabeled microphone recordings. By leveraging the powerful ECAPA-TDNN embedding model and clustering techniques such as HDBSCAN and K-Means, we demonstrated that it is possible to group recordings by speaker identity with notable internal validation performance. The preprocessing pipeline — including resampling, VAD, and normalization — ensured high-quality input for feature extraction. Our results showed that both K-Means and HDBSCAN achieved a silhouette score of 0.51 and a Davies-Bouldin index of 0.824, indicating reasonably well-separated clusters, while spectral clustering underperformed.

Despite the absence of ground-truth labels, internal metrics and manual review support the effectiveness of our pipeline. The success of this approach underscores the practicality of pre-trained speaker embeddings for large-scale unlabelled voice datasets. It opens up further applications in speaker diarization, verification, and clustering-based training data creation.

Citations

[1] Hoffman, Benjamin, and Grant Van Horn. “From Sound to Images, Part 1: A Deep Dive on Spectrogram Creation.” Macaulay Library, 19 July 2021, www.macaulaylibrary.org/2021/07/19/from-sound-to-images-part-1-a-deep-dive-on-spectrogram-creation/.

[2] “Speaker Identification Using Pitch and MFCC.” MATLAB & Simulink, www.mathworks.com/help/audio/ug/speaker-identification-using-pitch-and-mfcc.html#. Accessed 12 May 2025.

[3] D. Snyder, D. Garcia-Romero, G. Sell, D. Povey and S. Khudanpur, “X-Vectors: Robust DNN Embeddings for Speaker Recognition,” 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Calgary, AB, Canada, 2018, pp. 5329-5333, doi: 10.1109/ICASSP.2018.8461375. keywords: {Acoustics;Training;Feature extraction;Speaker recognition;Training data;Neural networks;NIST;speaker recognition;deep neural networks;data augmentation;x-vectors},

[4] “Speaker Diarization Using X-Vectors.” MATLAB & Simulink, www.mathworks.com/help/audio/ug/speaker-diarization-using-x-vectors.html#. Accessed 12 May 2025.

[5] Dawalatabad, Nauman, et al. “ECAPA-TDNN embeddings for speaker diarization.” Interspeech 2021, 30 Aug. 2021, https://doi.org/10.21437/interspeech.2021-941.

[6] Campello, Ricardo J., et al. “Density-based clustering based on hierarchical density estimates.” Lecture Notes in Computer Science, 2013, pp. 160–172, https://doi.org/10.1007/978-3-642-37456-2_14.

[7] Udit. “Discovering the Power of HDBSCAN Clustering for Unsupervised Learning.” Medium, Medium, 31 Dec. 2022, itsudit.medium.com/discovering-the-power-of-hdbscan-clustering-for-unsupervised-learning-d67273e28c5b#.

[8] Ohi, Abu Quwsar, et al. “AutoEmbedder: A semi-supervised DNN embedding system for Clustering.” Knowledge-Based Systems, vol. 204, Sept. 2020, p. 106190, https://doi.org/10.1016/j.knosys.2020.106190.

[9] Mridha, Muhammad Firoz, et al. “U-vectors: Generating clusterable speaker embedding from unlabeled data.” Applied Sciences, vol. 11, no. 21, 27 Oct. 2021, p. 10079, https://doi.org/10.3390/app112110079.

[10] Belyadi, Hoss, and Alireza Haghighat. “Unsupervised machine learning: Clustering algorithms.” Machine Learning Guide for Oil and Gas Using Python, 2021, pp. 125–168, https://doi.org/10.1016/b978-0-12-821929-4.00002-0.

[11] Wang, Yuxuan, et al. “Tacotron: Towards end-to-end speech synthesis.” Interspeech 2017, 20 Aug. 2017, https://doi.org/10.21437/interspeech.2017-1452.

[12] Livingstone, Steven R., and Frank A. Russo. “The Ryerson Audio-Visual Database of emotional speech and Song (RAVDESS): A Dynamic, multimodal set of facial and vocal

expressions in North American English." PLOS ONE, vol. 13, no. 5, 16 May 2018, https://doi.org/10.1371/journal.pone.0196391.

[13] Al-Fadhli, Sumayya, et al. "Towards an end-to-end speech recognition model for accurate Quranic recitation." 2023 20th ACS/IEEE International Conference on Computer Systems and Applications (AICCSA), 4 Dec. 2023, pp. 1–4, https://doi.org/10.1109/aiccsa59173.2023.10479314.

[14] Kumar, Piyush, et al. "A novel pitch based voice recognition model (PVRM)." 2018 4th International Conference on Recent Advances in Information Technology (RAIT), Mar. 2018, pp. 1–4, https://doi.org/10.1109/rait.2018.8388986.

[15] Takatoshi Jitsuhiro, et al. "Robust speech recognition using noise suppression based on multiple composite models and multi-pass search." 2007 IEEE Workshop on Automatic Speech Recognition & Understanding (ASRU), 2007, pp. 53–58, https://doi.org/10.1109/asru.2007.4430083.

[16] Becker, Sören, et al. "AudioMNIST: Exploring Explainable Artificial Intelligence for audio analysis on a simple benchmark." *Journal of the Franklin Institute*, vol. 361, no. 1, Jan. 2024, pp. 418–428, https://doi.org/10.1016/j.jfranklin.2023.11.038.

[17] Srinivasan, Sripaad. "Audio Mnist." Kaggle, 28 Dec. 2020, www.kaggle.com/datasets/sripaadsrinivasan/audio-mnist/data.