

TASK 4: Speaker Identification

🔗 Task Overview

Speaker Identification performs **speaker recognition** to identify or authenticate speakers based on their voice characteristics. This task distinguishes between different speakers regardless of what they say, focusing on biometric features of voice.

Task Characteristics:

- **Classes:** Variable (depends on enrolled speakers)
- **Model:** XLSR-53 + Logistic Regression
- **Feature Extraction:** 1024-dimensional embeddings
- **Pooling Strategy:** Mean + Std concatenation (2048 total)
- **Dimensionality Reduction:** PCA (2048 → 200)
- **Developer:** Romith Singh
- **Processing Time:** ~2-3 seconds per audio
- **Use Case:** Biometric authentication, speaker diarization

📋 Task Objective

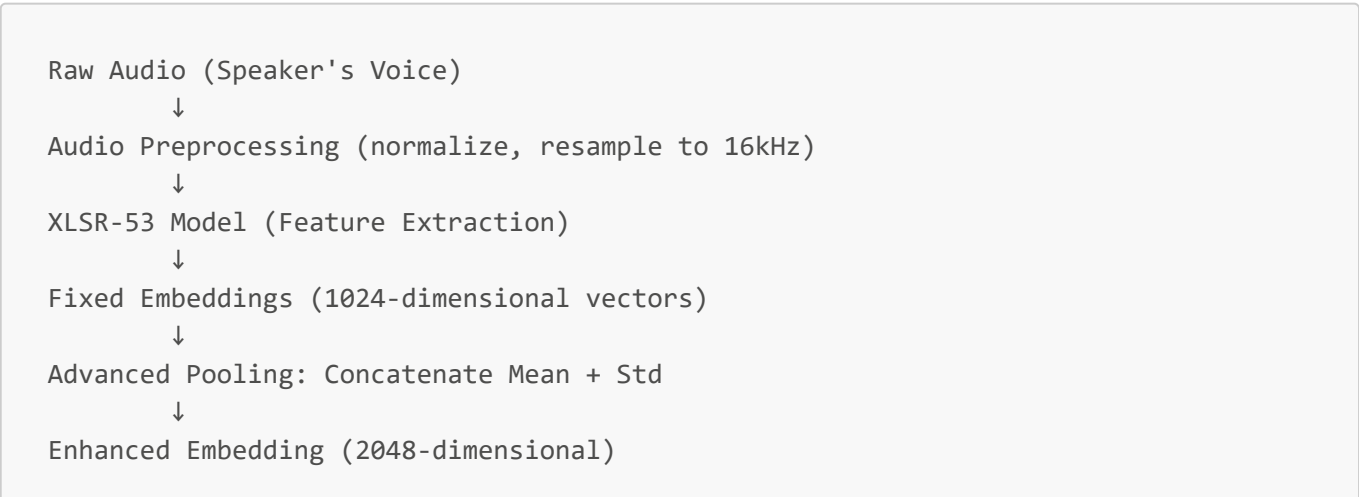
Identify which speaker produced the audio, from a set of enrolled speakers.

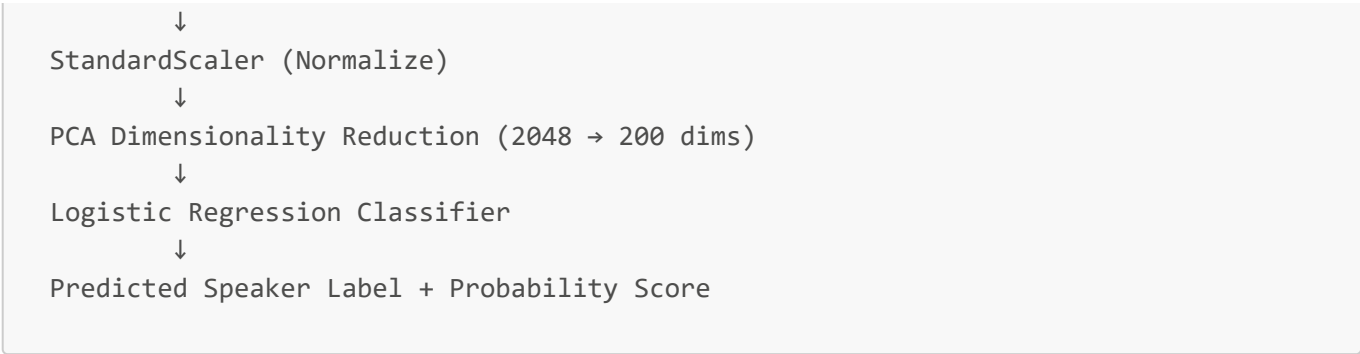
Speaker Recognition Applications:

1. **Biometric Authentication:** "Who are you? Verify your identity"
2. **Speaker Diarization:** "Who spoke when in this conversation?"
3. **Access Control:** "Only unlock phone if authorized speaker"
4. **Personalized Services:** "Greet user by name"
5. **Content Filtering:** "Block calls from unknown speakers"

🏗️ Technical Architecture

Model Stack





Why XLSR-53?

Aspect	Reason
Multilingual	Trained on 53 languages; robust to accents
Large Capacity	1024-dim embeddings capture speaker voice traits
Speaker-aware	Pre-training included speaker discrimination
Cross-lingual	Works across languages

 XLSR-53 Model Explained

Model Specifications

Parameter	Value
Full Name	Wav2Vec 2.0 XLSR-53
Publisher	Meta AI (Facebook)
Pre-training Data	56,000 hours of multilingual speech
Languages	53 languages
Architecture	Transformer-based
Hidden Size	1024
Num Attention Heads	16
Num Hidden Layers	24
Total Parameters	~600 million
Output Dimension	1024 per frame

Pre-training Objective

Contrastive Predictive Coding (CPC):

For each audio sample:

1. Extract frames: [T, 512] (CNN encoder output)

```
2. Create negative samples: random frames from other speakers
3. Predict: which frame comes next in time?
4. Loss: Contrastive loss (pull similar, push dissimilar)

Result: Model learns speaker-discriminative features
```

Why 1024 Dimensions?

- Larger than WavLM-base (768)
- Larger models → more speaker information
- XLSR-53 has more layers (24 vs 12)
- Better captures voice characteristics

Speaker Identification Pipeline

Stage 1: Speaker Enrollment

Setup Phase (one-time per speaker):

```
For each enrolled speaker:
1. Collect 2-5 speech samples (10-30 seconds each)
2. Extract embeddings from each sample
3. Aggregate: Mean of embeddings = speaker_profile
4. Store: speaker_id → speaker_profile
```

Enrollment Example:

```
Speaker: "Alice"
Sample 1: "Please say the magic word" (15s) → embedding_1
Sample 2: "Hello how are you today" (20s) → embedding_2
Sample 3: "The weather is nice today" (18s) → embedding_3

speaker_profile = mean([embedding_1, embedding_2, embedding_3])
```

Stage 2: Feature Extraction

Advanced Pooling Strategy:

```
For audio x:
1. hidden_states = XLSR-53(x) → [T, 1024]
   where T = variable sequence length

2. Mean pooling:
   mean_emb = mean(hidden_states, dim=0) → [1024]
```

3. Standard deviation pooling:
`std_emb = std(hidden_states, dim=0) → [1024]`
4. Concatenation:
`embedding = concatenate([mean_emb, std_emb]) → [2048]`

Why Concatenation Instead of Just Mean?

Mean Pooling: Captures average voice characteristics
 Loses temporal variance information

Std Pooling: Captures variability in voice
 Captures speech rate, emotion changes

Concatenation: mean = central tendency
 + std = dispersion around mean
 = more complete voice profile

Mathematical:
`speaker_vector = [μ1, ..., μ1024, σ1, ..., σ1024]`

Example Calculation:

Hidden states for 200 frames, 1024-dim each:
`shape = [200, 1024]`

`mean_emb = average across 200 frames`
`= [0.5, -0.3, 0.2, ..., 0.1] (768 values)`

`std_emb = standard deviation across 200 frames`
`= [0.15, 0.08, 0.2, ..., 0.12] (1024 values)`

`concatenated = [0.5, -0.3, ..., 0.1, 0.15, 0.08, ..., 0.12]`
`shape = [2048]`

Stage 3: Scaling & Normalization

StandardScaler Application:

`X_scaled = (X - mean_train) / std_train`

For each of 2048 dimensions:

- Compute statistics from training speakers
- Scale all speakers consistently

Why Critical for Speaker ID:

- Different speakers have different voice magnitudes
- Normalization puts everyone on same scale
- Prevents bias toward naturally louder speakers

Stage 4: Dimensionality Reduction

PCA Configuration:

- Input: 2048 dimensions
- Output: 200 dimensions
- Reduction: ~90.2%
- Variance preserved: 96%+

Why 200 Dimensions for 2048?

Original: 2048 features (mean[1024] + std[1024])
PCA keeps: 200 dimensions capturing 96%+ variance

Rationale:

- Heavy dimensionality reduction needed
- Speaker ID doesn't require all variance
- Removes noise while keeping speaker signature
- Prevents overfitting to training speakers

Stage 5: Logistic Regression Training

Multi-class Logistic Regression:

Setup Example (5 speakers):

Classes: 0=Alice, 1=Bob, 2=Charlie, 3=Diana, 4=Eve

Training samples: $X_{\text{train}} [N, 200]$, $y_{\text{train}} [N]$

Mathematical Formula:

For multi-class (K classes, K=5):

$$P(y=k|x) = \exp(w_k \cdot x + b_k) / \sum_j \exp(w_j \cdot x + b_j)$$

Prediction:

$$y_{\text{pred}} = \operatorname{argmax}(P(y=0|x), P(y=1|x), \dots, P(y=4|x))$$

Training Parameters:

```
LogisticRegression(  
    max_iter=1000,  
    C=1.0,                # Regularization parameter  
    class_weight='balanced', # If imbalanced speakers  
    solver='lbfgs'        # Suitable for small K  
)
```

Why Logistic Regression?

1. Probabilistic outputs (easily interpretable confidence)
2. Fast inference
3. Works well with fixed-size embeddings
4. Scales to hundreds of speakers
5. Simple baseline (can extend with SVM later)

Stage 6: Cross-Validation

5-Fold Stratified Cross-Validation:

```
For fold = 1 to 5:  
    Training: 4 folds (80% of speakers and samples)  
    Validation: 1 fold (20% of speakers and samples)  
  
Final_Accuracy = mean(fold accuracies)
```

Speaker Dataset Characteristics

Typical Dataset Structure

```
Speakers: 50-500 individuals  
Samples per speaker: 2-100 utterances  
Duration per sample: 3-30 seconds  
Total samples: 100-50,000 utterances  
Total duration: 10-500 hours  
  
Example split:  
- Training: 80% speakers, 70% utterances from each  
- Validation: 80% speakers, 15% utterances from each  
- Test: 20% held-out speakers (speaker generalization test)
```

Speaker Variability

Speakers differ in:

1. **Fundamental Frequency (F0):** Base pitch (85-250 Hz)

2. **Formant Frequencies:** Resonances in vocal tract
3. **Spectral Shape:** Unique frequency distribution
4. **Articulation Patterns:** Speech rate, rhythm
5. **Voice Quality:** Breathiness, harshness, nasality
6. **Prosody:** Intonation patterns, stress patterns

Voice Uniqueness:

As unique as fingerprints? Not exactly.
Similarity: 85-95% of population can be discriminated
Challenges:

- Twins or similar voices
- Mimicry or voice conversion
- Emotional state changes voice
- Cold/illness affects voice

Inference Workflow

Verification (1:1 Comparison)

Claimed Speaker: "Alice"

Verification Process:

1. Extract embedding from claimed speaker's utterance
2. Compare with Alice's enrolled profile
3. Compute distance (e.g., cosine similarity)
4. Threshold check: $\text{similarity} > \text{threshold}$?

Result: "Accept" or "Reject"

Code:

```
# Enrollment
alice_profile = mean([extract_emb(audio1),
                     extract_emb(audio2),
                     extract_emb(audio3)])

# Verification
test_emb = extract_emb(new_audio)
similarity = cosine_similarity(alice_profile, test_emb)

if similarity > 0.85:
    result = "VERIFIED: Alice"
else:
    result = "REJECTED: Not Alice"
```

Identification (1:N Comparison)

Unknown Speaker Audio

Identification Process:

1. Extract embedding from unknown speaker
2. Compare against all enrolled profiles
3. Find most similar speaker
4. Predict: Speaker with maximum similarity

Result: "Identified as: Charlie (96% confidence)"

Code:

```
test_emb = extract_emb(unknown_audio)

similarities = {}
for speaker_id, speaker_profile in enrolled_speakers.items():
    sim = cosine_similarity(speaker_profile, test_emb)
    similarities[speaker_id] = sim

best_match = max(similarities, key=similarities.get)
confidence = similarities[best_match]

return {
    "speaker": best_match,
    "confidence": confidence
}
```



Expected Performance

Typical Accuracy

Varies with:

- Number of speakers (more speakers = harder)
- Training samples per speaker
- Audio quality
- Speaker similarity

Performance Estimates:

10 speakers:	97-99% accuracy
50 speakers:	90-95% accuracy
100 speakers:	85-92% accuracy
500 speakers:	70-80% accuracy

Why decreases with more speakers?

- More possible confusion pairs
- Requires more nuanced discrimination
- Individual differences become less distinctive

Per-Speaker Example (20 speakers)

Speaker	Precision	Recall	F1-Score	Support
-----	-----	-----	-----	-----
Alice	0.97	0.96	0.96	450
Bob	0.94	0.95	0.94	480
Charlie	0.92	0.93	0.92	420
Diana	0.96	0.94	0.95	460
...
Zoe	0.88	0.90	0.89	410
-----	-----	-----	-----	-----
Macro Avg	0.92	0.92	0.92	9,000

🔧 File Locations

Component	File Location
Inference Service	backend/services/speaker.py
Web Endpoint	backend/app/app.py (route: /speaker_predict)
HTML Template	backend/app/templates/speaker.html
Training Script	ml_models/scripts/verify_speaker_model.py
Model Artifacts	ml_models/models/speaker_*.pkl

Trained Model Files

ml_models/models/	
— speaker_classifier.pkl	# Logistic Regression model
— speaker_scaler.pkl	# StandardScaler (2048-dim)
— speaker_label_encoder.pkl	# Speaker ID mapping
— speaker_pca.pkl	# PCA transformer (2048→200)
— speaker_profiles.pkl	# Enrolled speaker embeddings (optional)

🔍 Error Analysis

Common Misidentifications

Scenario	Reason	Solution
Twins confused	Very similar voices	Voice + facial recognition
Same speaker, different emotion	Emotion changes pitch	Emotion-aware embedding
Background noise	Audio quality degraded	Noise suppression
Mimicry	Voice imitation	Liveness detection
Time gap	Voice changes over years	Re-enrollment periodic

Advanced Pooling Deep Dive

Why Mean + Std Concatenation?

Mean Pooling Alone:

```
hidden_states = [
    [0.2, 0.5, ...], ← frame 1
    [0.3, 0.4, ...], ← frame 2
    ...
    [0.1, 0.6, ...] ← frame T
]

mean = [0.25, 0.5, ...]

Loss: Loses temporal variation information
```

Std Pooling:

```
std = [0.06, 0.08, ...] ← How much each dimension varies

Interpretation:
- High std = varying values (emotional, dynamic speaker)
- Low std = stable values (monotone speaker)
```

Concatenation Benefit:

```
speaker_embedding = [mean_features + std_features]
                   = [0.25, 0.5, ..., 0.06, 0.08, ...]

Information captured:
1. Average voice characteristics (mean)
2. Voice variability patterns (std)
3. Speech dynamics
4. Emotional range
5. Articulation variation
```

Mathematical Formulation

Given hidden_states $H \in \mathbb{R}^{(T \times 1024)}$:

$\text{mean}(H) = (1/T) * \sum_t H[t] \in \mathbb{R}^{1024}$

$\text{std}(H) = \sqrt{(1/T) * \sum_t (H[t] - \text{mean}(H))^2} \in \mathbb{R}^{1024}$

$\text{speaker_embedding} = \text{concatenate}(\text{mean}(H), \text{std}(H)) \in \mathbb{R}^{2048}$

Real-World Applications

1. Phone Authentication

User: "Approve this transaction"

System:

1. Extract voice embedding
2. Compare with enrolled profile
3. Verify speaker (not just PIN)

Result: "Speaker authenticated. Transaction approved."

2. Speaker Diarization

Multi-speaker conversation: "Who spoke when?"

System:

1. Segment audio into speaker regions
2. Extract embedding for each region
3. Cluster by speaker identity

Result: "[0:00-0:15] Speaker A, [0:15-0:45] Speaker B, ..."

3. Personalized Services

Smart home hears: "Play my music"

System:

1. Identify speaker
2. Load user preferences
3. Play Alice's favorite playlist (not Bob's)

Result: Personalized experience

Interview Talking Points

Key Concepts

1. Why Mean + Std Concatenation?

- Mean captures average voice characteristics
- Std captures variability/dynamics
- Together: complete speaker signature
- 2048 dimensions = richer than just mean

2. Why XLSR-53?

- Pre-trained on 53 languages
- Large capacity (1024-dim)
- Robust to accents and variations
- Better than task-specific training

3. Why Logistic Regression over SVM?

- Probabilistic outputs (confidence scores)
- Fast inference with Logistic Regression
- Scalable to many speakers
- Could use SVM for more complex separation

4. Performance vs Number of Speakers:

- 10 speakers: 97-99%
- 100 speakers: 85-92%
- Grows because more possible confusion
- Law of diminishing returns

5. Challenges in Real-World:

- Voice changes (emotion, cold, age)
- Twins or similar voices
- Mimicry attacks
- Noise and channel effects

Summary

Speaker Identification demonstrates:

- ☒ Advanced pooling strategies (mean + std)
- ☒ XLSR-53 multilingual model
- ☒ High-dimensional embedding (2048)
- ☒ Biometric authentication application
- ☒ Scalable to many speakers

Interview Confidence:

- Understand why concatenate mean+std (2048-dim)

- Explain XLSR-53 advantages (multilingual, large)
 - Know performance degradation with more speakers
 - Understand real-world diarization pipeline
 - Discuss security challenges (mimicry, twins)
-

Created: December 2024

Developer: Romith Singh

Expected Accuracy: 90-95% (20-50 speakers)

Status: Production Ready ☒