Speaker Identification System

This system allows you to train a machine learning model to identify specific speakers by name, rather than just labeling them generically as "Speaker A", "Speaker B", etc.

Quick Start

1. Test the System (Demo)

```
python test_speaker_identification.py
```

This runs a demo with simulated voice data to show how the system works.

2. Collect Real Training Data

```
python collect_training_data.py
```

This will guide you through recording voice samples from different speakers.

3. Use in Transcriber

```
python transcriber.py
```

The transcriber will automatically use the trained model if available.

How It Works

Current System (Generic Labels)

```
Audio → Speaker Detection → "Speaker A/B/C"
```

New System (Named Identification)

```
Audio → Speaker Identification → "John/Sarah/Mike" (with confidence)
```

Training Process

Step 1: Configure Speakers

Edit `collect_training_data.py` to specify your speakers:

```
speakers_config = {
    'John': {'samples': 10, 'duration': 3},    # 10 samples, 3 seconds each
    'Sarah': {'samples': 10, 'duration': 3},   # 10 samples, 3 seconds each
    'Mike': {'samples': 10, 'duration': 3}     # 10 samples, 3 seconds each
}
```

Step 2: Record Training Data

`python collect_training_data.py`

The script will: - Ask you to record samples for each speaker - Extract voice features from each recording - Save audio files to `training_data/` folder - Save training data to `speaker_training_data.json`

Step 3: Train the Model

The training happens automatically during data collection, but you can also train manually:

```
from speaker_identifier import SpeakerIdentifier
from collect_training_data import TrainingDataCollector

# Load training data
collector = TrainingDataCollector()
training_data = collector.load_training_data("speaker_training_data.json")

# Train model
speaker_id = SpeakerIdentifier()
train_score, test_score = speaker_id.train_on_labeled_data(training_data)

# Save model
speaker_id.save_model("speaker_identification_model.pkl")
```

Voice Features Extracted

The system extracts 14 different voice characteristics:

Basic Features:

- `energy`: Average audio amplitude
- `pitch_estimate`: Pitch variation
- `zero_crossings`: Frequency content
- `spectral_centroid`: Spectral center of mass
- `energy_variance`: Energy variation over time
- `peak_amplitude`: Maximum amplitude
- `rms_energy`: Root mean square energy

Advanced Features:

- `mfcc_1/2/3`: Mel-frequency cepstral coefficients
- `formant_1/2`: Formant frequencies (vocal tract resonances)
- `jitter`: Pitch variation over time
- `shimmer`: Amplitude variation over time

Model Performance

Expected Accuracy:

- **Training Accuracy**: 95-99% (on training data)
- **Test Accuracy**: 85-95% (on unseen data)
- **Real-world Performance**: 70-90% (depending on audio quality)

Confidence Thresholds:

- **High Confidence**: > 0.8 (likely correct identification)
- **Medium Confidence**: 0.6-0.8 (probably correct)
- **Low Confidence**: < 0.6 (uncertain, may show as "Unknown")

Best Practices for Training

Audio Quality:

- Use a good microphone
- Record in a quiet environment
- Maintain consistent distance from microphone
- Speak clearly and naturally

Training Data:

- **Minimum**: 5 samples per speaker
- **Recommended**: 10-20 samples per speaker
- **Sample Duration**: 3-5 seconds each
- **Content**: Different phrases/words (not just repeating the same thing)

Speaker Diversity:

- Include different speaking styles (normal, excited, quiet)
- Record at different times of day
- Include natural variations in voice

Usage Examples

In Transcriber Output:

```
"Hello, how are you?"
   John (0.92) |  2 voice(s) | Joy (0.95)  Question

Speaker change detected: Sarah

"I'm doing great, thanks!"
   Sarah (0.88) |  2 voice(s) | Joy (0.87)
```

Programmatic Usage:

```python
from speaker_identifier import SpeakerIdentifier

# Load trained model
speaker_id = SpeakerIdentifier()
speaker_id.load_model("speaker_identification_model.pkl")

# Identify speaker from audio features
features = speaker_id.extract_speaker_features(audio_data)
speaker_name, confidence = speaker_id.identify_speaker(features)

print(f"Speaker: {speaker_name} (confidence: {confidence:.2f})")
```

File Structure

```
cortex_bridge/
  speaker_identifier.py              # Core speaker identification system
  collect_training_data.py           # Training data collection script
  test_speaker_identification.py     # Demo/test script
  transcriber.py                     # Main transcriber (integrated)
  speaker_identification_model.pkl   # Trained model (generated)
  speaker_training_data.json         # Training data (generated)
  training_data/                     # Audio samples (generated)
      John_sample_1.wav
      John_sample_2.wav
      Sarah_sample_1.wav
      ...
```

Advanced Configuration

Model Parameters:

```python
# In speaker_identifier.py
self.classifier = RandomForestClassifier(
    n_estimators=100,    # Number of trees (higher = better but slower)
    random_state=42      # For reproducible results
)
```

Confidence Threshold:

```python
# In transcriber.py
if confidence > 0.7:  # Adjust this threshold
    speaker_info = f" {speaker_name} ({confidence:.2f})"
else:
    speaker_info = f" Unknown ({confidence:.2f})"
```

Feature Extraction:

You can modify the feature extraction in `speaker_identifier.py` to add more voice characteristics or adjust existing ones.

Troubleshooting

Low Accuracy:

- Increase number of training samples
- Improve audio quality
- Ensure speakers are clearly different
- Check for background noise

Model Not Loading:

- Ensure `speaker_identification_model.pkl` exists
- Check file permissions
- Verify model was saved correctly

Poor Real-time Performance:

- Reduce number of features extracted
- Lower RandomForest tree count
- Use smaller audio chunks

Adding New Speakers

Method 1: Retrain Entire Model

```
# Load existing training data
training_data = collector.load_training_data("speaker_training_data.json")

# Add new speaker data
new_speaker_data = {
    'speaker_name': 'Emma',
    'audio_features': [feature1, feature2, ...]
}
training_data.append(new_speaker_data)

# Retrain and save
speaker_id = SpeakerIdentifier()
speaker_id.train_on_labeled_data(training_data)
speaker_id.save_model("speaker_identification_model.pkl")
```

Method 2: Incremental Training

```
# Load existing model
speaker_id = SpeakerIdentifier()
speaker_id.load_model("speaker_identification_model.pkl")

# Add new speaker
speaker_id.add_speaker("Emma")

# Train on new data only
# (Implementation depends on your needs)
```

Performance Monitoring

Accuracy Tracking:

```
# Get model performance
train_score, test_score = speaker_id.train_on_labeled_data(training_data)
print(f"Training accuracy: {train_score:.2f}")
print(f"Test accuracy: {test_score:.2f}")
```

Confidence Analysis:

```
# Monitor confidence levels
speaker_name, confidence = speaker_id.identify_speaker(features)
if confidence < 0.6:
    print("  Low confidence identification")
```

Integration with Other Systems

The speaker identification system integrates seamlessly with: - **Conversation Logging**: Speaker names are logged in the database - **Session Analysis**: Speaker-specific analysis in AI summaries - **Vectorization**: Speaker-aware semantic search - **Emotion Analysis**: Per-speaker emotion tracking

Future Enhancements

- **Deep Learning Models**: CNN/LSTM for better accuracy
- **Adaptive Learning**: Online learning from new samples
- **Speaker Clustering**: Automatic speaker discovery
- **Multi-language Support**: Language-specific voice models
- **Real-time Adaptation**: Continuous model updates

Note: This system works completely offline and doesn't require internet connectivity once trained.