**Distinguishing Between Two Human Voices Using AI**

**Pretrained Models / Toolkits**

You can use existing toolkits instead of building from scratch:

* **pyannote.audio** (Speaker diarization)
* **Kaldi** (Open-source speech recognition toolkit)
* **DeepSpeaker** (Speaker identification)
* **SpeechBrain** (Diarization and recognition)

**Parameters:**

To measure the difference between two human voices, we analyse various **acoustic and linguistic parameters**. Here are the key parameters used in speaker differentiation:

**1. Acoustic Features (Physical characteristics of sound)**

**a. Spectral Features (Frequency-based)**

* **Mel-Frequency Cepstral Coefficients (MFCCs):** Represents the vocal tract shape, widely used for speaker recognition.
* **Linear Predictive Coding (LPC):** Estimates the formants (resonant frequencies) of speech.
* **Spectral Centroid:** Measures the "brightness" of a sound, helping distinguish voices.
* **Spectral Flatness:** Helps differentiate between voiced and unvoiced sounds.
* **Formants (F1, F2, F3):** Resonance frequencies unique to each speaker.

**b. Prosodic Features (Pitch & Rhythm)**

* **Fundamental Frequency (F0):** The base pitch of a person’s voice. Men typically have lower F0 (~85-180 Hz) than women (~165-255 Hz).
* **Pitch Variation:** Differences in how a speaker’s pitch changes over time.
* **Speaking Rate (Words per Minute, WPM):** Faster/slower speech can be distinctive.
* **Intensity (Loudness):** Measured in decibels (dB), varies per speaker.
* **Jitter & Shimmer:** Measures of pitch and amplitude instability, respectively.

**c. Temporal Features (Time-based)**

* **Speech Duration & Pauses:** Some speakers naturally take longer pauses.
* **Voice Onset Time (VOT):** Time difference between the release of a consonant and the onset of vocal cord vibration.

**2. Linguistic Features (Speech patterns and content)**

* **Choice of Words & Grammar:** Some people use more complex sentences, while others prefer direct speech.
* **Accent & Pronunciation:** Different regional or foreign accents influence speaker differentiation.
* **Filler Words (e.g., "uh," "um")**: The frequency and type of filler words vary.
* **Speech Disfluencies (Pauses, Repetitions):** Unique to each speaker’s communication style.

**3. Biometric Features (Physical and anatomical factors)**

* **Vocal Tract Length & Shape:** Determines resonance frequencies and formant structure.
* **Larynx Size:** Affects fundamental frequency (men have larger larynxes, leading to deeper voices).

**4. Deep Learning-Based Features**

Using **deep neural networks (DNNs)**, models learn high-dimensional voice representations (embeddings) using:

* **VGGVox, x-vector, d-vector models:** Used for speaker verification.
* **Speaker embeddings:** Compact representations of a speaker’s voice, often extracted using models like **VoxCeleb**.

**Voice Embeddings**

Use a **pretrained deep learning model** to extract **voice embeddings** (fixed-size vector representations of speakers). Some popular models include:

* **SpeakerNet**
* **VoxCeleb (ResNet-based)**
* **DeepSpeaker**
* **Whisper (for speech + speaker ID)**

| **Step** | **What we want to build a model from scratch** |
| --- | --- |
| **Step 1** | Collected dataset (VoxCeleb, CommonVoice, or custom recordings) |
| **Step 2** | Extracted **Mel Spectrograms** or **MFCCs** |
| **Step 3** | Built a **CNN-based deep learning model** to learn voice embeddings |
| **Step 4** | Trained the model on the dataset |
| **Step 5** | Extracted **voice embeddings** and used **Cosine Similarity** to compare voices |

**Dynamic Time Warping (DTW):**

**Step 1: Collecting a Dataset (Creating Voice Samples)**

**1. Number of Voice Clips Needed**

* At least **50-100 samples per speaker** for a small project.
* For better accuracy, **500+ samples per speaker** is ideal.

**2. Duration of Each Clip**

* Each clip should be **2-10 seconds** long.
* Ensure diversity in speech (different sentences, emotions, speeds).

**3. Number of Speakers**

* Start with **at least 2 speakers**.
* If you plan to scale, include **10-50 speakers**.

**4. Audio Format & Quality**

* Format: **WAV (preferred) or MP3**
* Sample Rate: **16kHz or 44.1kHz**
* Bit Depth: **16-bit for clear voice quality**

**5. Recording Methods**

* Use different devices (smartphones, microphones) to introduce variations.
* Record in different environments (quiet room, noisy background) for robustness.

**Step 2: Preparing the Dataset**

import sounddevice as sd

import numpy as np

import wave

import os

# Function to record audio

def record\_audio(filename, duration=3, samplerate=16000):

print(f"Recording: {filename}...")

audio\_data = sd.rec(int(duration \* samplerate), samplerate=samplerate, channels=1, dtype=np.int16)

sd.wait() # Wait until recording is finished

# Save the recorded audio as a WAV file

with wave.open(filename, "wb") as wf:

wf.setnchannels(1)

wf.setsampwidth(2) # 16-bit PCM

wf.setframerate(samplerate)

wf.writeframes(audio\_data.tobytes())

print(f"Saved: {filename}")

# Main function to record multiple samples for each speaker

def record\_dataset(num\_speakers=2, samples\_per\_speaker=5, duration=3):

dataset\_path = "dataset"

os.makedirs(dataset\_path, exist\_ok=True)

for speaker\_id in range(1, num\_speakers + 1):

speaker\_folder = os.path.join(dataset\_path, f"speaker\_{speaker\_id}")

os.makedirs(speaker\_folder, exist\_ok=True)

print(f"\nRecording for Speaker {speaker\_id}:")

for sample\_num in range(1, samples\_per\_speaker + 1):

filename = os.path.join(speaker\_folder, f"sample\_{sample\_num}.wav")

record\_audio(filename, duration)

print("\nRecording Complete!")

# Run the recording function

record\_dataset(num\_speakers=2, samples\_per\_speaker=5, duration=3)

**Step 3: Feature Extraction**

Extract key features from the audio that help differentiate speakers:

* **Mel-Frequency Cepstral Coefficients (MFCCs)** – Capture vocal tract characteristics.
* **Spectrogram Analysis** – Visual representation of frequency variations.
* **Pitch, Tone, and Formants** – Help distinguish speakers based on vocal characteristics.

You need to convert audio into numerical features. Use **Python libraries** like librosa or SpeechBrain.

Features to extract:

* **MFCCs** (captures vocal tract info)
* **Fundamental Frequency (F0)**
* **Spectral Features (Formants, Spectral Centroid, Bandwidth)**
* **Prosodic Features (Pitch Variation, Speech Rate, Pauses)**

Code:

import librosa

import numpy as np

import pandas as pd

import os

# Function to extract features from an audio file

def extract\_features(audio\_path):

y, sr = librosa.load(audio\_path, sr=16000) # Load audio

mfccs = librosa.feature.mfcc(y=y, sr=sr, n\_mfcc=13) # Extract 13 MFCCs

pitch = librosa.yin(y, fmin=50, fmax=300) # Extract pitch

spectral\_centroid = librosa.feature.spectral\_centroid(y=y, sr=sr)

spectral\_bandwidth = librosa.feature.spectral\_bandwidth(y=y, sr=sr)

rms\_energy = librosa.feature.rms(y=y)

# Convert to mean values (to create a fixed-size feature vector)

return np.hstack([

np.mean(mfccs, axis=1),

np.mean(pitch),

np.mean(spectral\_centroid),

np.mean(spectral\_bandwidth),

np.mean(rms\_energy)

])

# Function to process all audio files and save features to a CSV file

def process\_dataset(dataset\_path="dataset"):

features\_list = []

for speaker\_folder in os.listdir(dataset\_path):

speaker\_id = speaker\_folder.split("\_")[-1] # Extract speaker ID

speaker\_path = os.path.join(dataset\_path, speaker\_folder)

for audio\_file in os.listdir(speaker\_path):

audio\_path = os.path.join(speaker\_path, audio\_file)

features = extract\_features(audio\_path)

features\_list.append([speaker\_id] + list(features)) # Add speaker label

# Convert to a DataFrame and save as CSV

columns = ["Speaker\_ID"] + [f"MFCC\_{i}" for i in range(13)] + ["Pitch", "Spectral\_Centroid", "Spectral\_Bandwidth", "RMS\_Energy"]

df = pd.DataFrame(features\_list, columns=columns)

df.to\_csv("voice\_features.csv", index=False)

print("Feature extraction complete! Data saved to voice\_features.csv")

# Run the feature extraction

process\_dataset()

**Step 4: Training the Model**

**1️ Traditional Machine Learning Models**

**Support Vector Machine (SVM)** – Works well for small datasets, finds the best decision boundary.  
**Random Forest (RF)** – Handles feature importance, good for tabular data.  
**K-Nearest Neighbors (KNN)** – Simple, works well if data is well-separated.  
**Gradient Boosting (XGBoost)** – Boosts weak models to improve performance.

**2️ Deep Learning Models (Neural Networks)**

**Multi-Layer Perceptron (MLP - Feedforward Neural Network)** – Can learn complex patterns.  
**Convolutional Neural Networks (CNNs)** – Usually used for raw spectrograms but can work on feature data.  
 **Recurrent Neural Networks (RNNs/LSTMs)** – Better for sequential audio signals but needs more data.

**Step 5: Testing and Evaluating the Model**

* Use a **train-test split (80-20)** or **cross-validation**.
* Measure performance using **Accuracy, Precision, Recall, and F1-score**.
* For deep learning, use **ROC curves and loss graphs** to optimize training.

Bottom of Form

**Step 6: Improving Accuracy with Hyperparameter Tuning**

Now that we have trained a basic **Random Forest** and **Neural Network (MLP)** model, let’s optimize them for **higher accuracy**. This is called **Hyperparameter Tuning**.

**Hyperparameter Tuning for Random Forest**

**Parameters to Tune:**

n\_estimators – Number of trees in the forest.  
 max\_depth – Maximum depth of each tree.  
min\_samples\_split – Minimum samples required to split a node.  
min\_samples\_leaf – Minimum samples required at a leaf node.

**Grid Search for Best Parameters**

from sklearn.model\_selection import GridSearchCV

from sklearn.ensemble import RandomForestClassifier

# Define parameter grid

param\_grid = {

'n\_estimators': [100, 200, 300], # Number of trees

'max\_depth': [10, 20, 30], # Depth of each tree

'min\_samples\_split': [2, 5, 10], # Minimum samples required to split

'min\_samples\_leaf': [1, 2, 4] # Minimum samples at leaf

}

# Initialize model

rf = RandomForestClassifier(random\_state=42)

# Perform Grid Search

grid\_search = GridSearchCV(rf, param\_grid, cv=3, scoring='accuracy', n\_jobs=-1)

grid\_search.fit(X\_train, y\_train)

# Get best parameters

best\_rf = grid\_search.best\_estimator\_

print(f"Best Random Forest Parameters: {grid\_search.best\_params\_}")

# Evaluate tuned model

y\_pred = best\_rf.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

print(f"Tuned Random Forest Accuracy: {accuracy \* 100:.2f}%")

**Step 2: Hyperparameter Tuning for Neural Network (MLP)**

**Parameters to Tune:**

hidden\_layer\_sizes – Number of neurons per layer.  
 activation – Activation function (relu, tanh, sigmoid).  
learning\_rate – How fast the model learns.  
batch\_size – Number of samples per training batch.

**Install Keras Tuner**

bash

pip install keras-tuner

**code:**

import keras\_tuner as kt

import tensorflow as tf

from tensorflow import keras

# Define function to build a tunable model

def build\_model(hp):

model = keras.Sequential()

model.add(keras.layers.Dense(hp.Int('units1', min\_value=32, max\_value=256, step=32), activation='relu', input\_shape=(X\_train.shape[1],)))

model.add(keras.layers.Dense(hp.Int('units2', min\_value=32, max\_value=128, step=32), activation='relu'))

model.add(keras.layers.Dense(len(np.unique(y\_train)), activation='softmax'))

model.compile(optimizer=keras.optimizers.Adam(learning\_rate=hp.Choice('learning\_rate', [0.001, 0.01, 0.1])),

loss='sparse\_categorical\_crossentropy',

metrics=['accuracy'])

return model

# Define tuner

tuner = kt.RandomSearch(

build\_model,

objective='val\_accuracy',

max\_trials=10,

executions\_per\_trial=1,

directory='tuning\_results',

project\_name='voice\_classifier'

)

# Run the tuner

tuner.search(X\_train, y\_train, epochs=50, validation\_data=(X\_test, y\_test))

# Get the best model

best\_hps = tuner.get\_best\_hyperparameters(num\_trials=1)[0]

print(f"Best Hyperparameters: {best\_hps.values}")

# Train final optimized model

best\_model = tuner.hypermodel.build(best\_hps)

best\_model.fit(X\_train, y\_train, epochs=50, validation\_data=(X\_test, y\_test))

**Expected Accuracy After Tuning**

| **Model** | **Before Tuning** | **After Tuning** |
| --- | --- | --- |
| Random Forest | 85-90% | **90-95%** |
| Neural Network (MLP) | 85-92% | **95-98%** |