IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661,p-ISSN: 2278-8727, Volume 26, Issue 1, Ser. 1 (Jan. – Feb. 2024), PP 19-26
www.iosrjournals.org

Speaker Verification Using Cosine Similarity

Dr.A.Annie Micheal[1], Dr.B.U.Anu Barathi[2], Lakshmapuram Abhinesh[3], Kolluri Yasaswini[4]

[1,2,3,4]*(Department Of Computer Science And Engineering, Sathyabama Institute Of Science And Technology, India)*

Abstract:
Background: *Voice Similarity Analysis, is useful to identify and compare voices and is crucial in various applications, including speaker identification, music analysis, and surveillance. Traditional methods rely on features like pitch and formants, but they can be susceptible to variations in speaking style and environment. Spectrogram analysis offers a robust alternative by capturing the overall frequency distribution of the voice signal. Proposed a GUI application that utilizes cosine similarity on Mel-spectrograms to compare voice samples. User upload audio files, and the application extracts, resizes, and compares their spectrograms. A predefined similarity threshold determines whether the voices are considered similar or different.*
Libraries and Techniques: *In this analysis, we used some libraries called tkinter, OS, librosa, scikit-learn, numpy, matplot lib. Techniques in this program captures the overall frequency distribution of the voice signal, providing a robust representation for comparison. Mel-spectrograms, focusing on frequencies relevant to human perception, are particularly useful in this context. Cosine similarity, metric measures the angle between two vectors, in this case the Mel-spectrograms of two audio samples. A higher cosine similarity indicates greater similarity between the voices. Thresholding, A predefined similarity threshold is used to classify whether two voices are considered similar or different based on the calculated cosine score. User interaction, the program utilizes Tkinter widgets and event handling to enable users to upload audio files, analyze them, and view the results within the application.*
Result*: When user gives the input audio files it analyses the audio using cosine similarity and delivers a clear verdict of audio files with transparent view of spectrograms.*
Conclusion: *we conclude that, the script provides a basic framework for analyzing audio spectrograms and comparing them using cosine similarity. users can interact with a simple GUI to upload audio files, visualize spectrograms, and analyze similarities**.** users can modify the spectrogram visualization to include more interactive features, such as zooming, panning, or allowing users to select specific regions of interest.*

Keywords: *Mel-spectogram; Frequency; Libraries; Cosine Similarity, matplot lib; librosa; tkinter; Scikit-learn*

Date of Submission: 01-01-2024 Date of Acceptance: 09-01-2024

I. Introduction

Voice analysis has gained prominence in various domains, including security, entertainment, and human-computer interaction. This paper presents a Voice Similarity Analyzer—a computational tool designed to process, visualize, and compare audio signals based on their spectrograms. The Python script serves as an interactive tool for audio spectrogram analysis, leveraging the capabilities of Tkinter for GUI development, librosa for audio processing, and matplotlib for visualizations. Through a user-friendly interface, the script allows users to upload multiple audio files, computes and displays their spectrograms, and performs pairwise comparisons to assess the similarity between different voices. The script initiates by prompting the user for the multiple audio files and a similarity threshold through the console. Subsequently, a Tkinter window is created, featuring entry widgets for user input, buttons for uploading audio files and initiating the analysis, and a visually informative layout. The GUI enables users to seamlessly interact with the script, providing file paths for analysis.

II. Libraries and Techniques

Libraries used: os, librosa, tkinter, matplot lib, numpy, sklearn.metrics.pairwise,

OS: os checks file existence, extracts basename, Checks directory Path. It also imports the os module from the Python standard library. A method of interacting with the operating system is provided by the OS module.allowing the script to carry out different tasks pertaining to the manipulation of files and directories, environment variables, and other OS functionalities.

Librosa: librosa.load is used to load an audio file, providing the audio data (y) and the sampling rate (sr). librosa.stft is used to compute the Fourier Transform in Short Time (STFT), which is thereafter changed to a

spectrogram using np.abs and librosa.amplitude_to_db. librosa.display.specshow is used to display the computed spectrogram using matplotlib.

Tkinter: Tkinter is used to create GUI. It is used to create entry widgets, allowing users to input files. Also used to create clickable buttons,such as the "Upload Audio Files" button and the "Analyze" button. filedialog.askopenfilename is used to open a file dialog box, allowing users to select audio files.In this code, it enables the creation of a user-friendly interface for interacting with the audio spectrogram analysis functionalities. Users can input file paths, trigger actions with buttons, and receive visual feedback through the GUI.

Matplot Lib: Matplot Creates a subplot for displaying the spectrogram. The fig and ax variables are used to customize the subplot. It Uses librosa.display.specshow to display the computed spectrogram (D) on the specified subplot (ax). The parameters set the time axis to 'time' and the frequency axis to 'log'. Finally, plt.show() is used to display the plotted spectrogram.

Numpy: NumPy plays a crucial role in handling and manipulating numerical data throughout the audio processing and similarity analysis pipeline. It provides efficient array operations and mathematical functions that are essential for tasks involving numerical data, making it a valuable library for scientific computing. NumPy arrays are utilized to perform operations such as resizing the spectrogram to a fixed shape and initializing arrays to store the spectrogram data.

Sklearn.metrics.pairwise: The 'Sklearn.metrics.pairwise'module is used for computing pairwise similarities between flattened spectrograms using the cosine similarity metric. The 'cosine similarity' function takes two arrays (or matrices) as input and returns a similarity matrix. Since only a single pair of spectrograms is being compared at a time, the '[0][0]'indexing is used to extract the actual similarity value from the matrix.

Techiques used:

Audio Processing with librosa:

Librosa Library: Librosa is used for audio processing, providing functions for loading audio files, calculating spectrograms using the Fourier Transform in Short Time (STFT), and converting amplitude to decibels.

spectrum Analysis:

the Fourier Transform in Short Time (STFT): Spectrograms in fact computed using the STFT, which allows the analysis of the audio signal's frequency content with time.

Cosine Similarity: Spectrograms are compared using cosine similarity, a metric that measures the angle between two vectors' cosine. It is often used for comparing the similarity of spectrograms or other feature representations.

Tkinter's Graphical User Interface (GUI):

The Tkinter Library: The tkinter library is employed for creating A basic graphical user interface. It gives functions for creating entry widgets, buttons, and handling user interactions.

Data Visualization with Matplotlib:

Matplotlib: Matplotlib is used for plotting and displaying spectrograms. It enables the visualization of audio data in the form of spectrograms on a graphical interface.

User Input Handling:

Entry Widgets: tkinter entry widgets are used for users to input the paths of audio files.

Button Widgets: tkinter buttons are used for triggering specific actions, such as uploading audio files and initiating the analysis.

Time Delays for Visualization:

After Method: The after method in tkinter is used to introduce time delays between displaying spectrograms. This allows users to observe spectrograms one after the other.

File Dialog for File Selection:

filedialog Module: The filedialog module from tkinter is used to open a file dialog for users to select audio files for analysis.

Dynamic User Interface:

Dynamic Widget States: The GUI is designed to dynamically change the state of the "Analyze" button based on user actions, allowing it to be enabled or disabled as needed.

Error Handling:

File Existence Check: Before processing, the code checks whether the provided audio files exist to handle potential errors gracefully.

Code Organization and Modularization:

Functions: The code is organized into functions to encapsulate specific tasks, promoting modularity and readability.

User Guidance:

Print Statements: Print statements are used to provide feedback to the user regarding the progress of the analysis and any issues encountered, such as missing files.

User Interaction Loop:

tkinter Mainloop: The mainloop method is used to start the main event loop of the tkinter GUI, allowing for continuous user interaction until the user closes the GUI.

III. Proposed Methodology

Real-time Analysis Integration:

In response to the evolving needs of real-time audio analysis, our proposed methodology aims to integrate capabilities for processing and visualizing audio streams as they are recorded or received. This enhancement seeks to elevate the Voice Similarity Analysis applicability in scenarios requiring instantaneous feedback, such as live events, voice communication systems, and continuous monitoring.

Implementation Steps:

Streaming Audio Input: Incorporate mechanisms for real-time audio input, leveraging existing libraries or APIs that support streaming functionality.

Incremental Spectrogram Generation: Adapt the current spectrogram generation process to accommodate incremental updates in real-time. Ensure that the spectrogram display dynamically evolves as new audio data is received.

Dynamic Similarity Calculations: Develop algorithms for dynamic voice similarity calculations, allowing users to receive immediate feedback on changing voice patterns as the spectrogram unfolds in real-time.

Machine Learning Integration:

To augment the tool's analytical capabilities, the proposed methodology introduces the integration of machine learning models for automated identification of speakers and advanced feature extraction. This expansion aims to move beyond simplistic similarity metrics, introducing predictive modeling for voice classification, emotion recognition, or other pertinent attributes.

Implementation Steps:

Dataset Collection and Labeling: Curate a diverse dataset of audio samples with appropriate labels for training the machine learning model. This dataset should encompass a wide range of speakers, accents, and emotional states.

Feature Engineering: Extract relevant features from the spectrograms and other audio characteristics, exploring techniques Mel-frequency cepstral coefficients (MFCCs), for example, or deep learning-based feature extraction.

Model Training: Develop and train a machine learning model using the curated dataset. Experiment with well-established models, including Neural networks with convolutional properties (CNNs) and recurrent properties (RNNs), to achieve optimal categorizing performance.

prototype Integration: Embed the prepared machine learning model into the Audio Spectrogram Analyzer, allowing users to perform advanced voice analysis beyond the confines of simple similarity metrics.

Diverse Audio Format Compatibility:

Recognizing the importance of compatibility with a wide array of audio file formats, our proposed methodology seeks to refine the tool's file handling capabilities. This refinement ensures seamless integration with different recording devices and platforms.

Implementation Steps:

Format Detection and Conversion: Implement a dynamic format detection mechanism upon upload. Integrate converters or adaptors to handle various formats, automatically converting them into a standardized format compatible with the analysis pipeline.

User Guidance: Provide clear instructions and feedback to users regarding supported audio formats. Implement informative error messages if an incompatible format is detected, guiding users on potential solutions.

Testing with Diverse Datasets: Validate the enhanced compatibility with a diverse set of audio files, conducting thorough testing to ensure robust performance across different recording sources and formats.

Usability Testing and Iterative Refinement:

Acknowledging the iterative nature of software development and the significance of user feedback, our proposed methodology emphasizes ongoing usability testing and iterative refinement.

Implementation Steps:

Sessions for User Testing: Regularly hold user testing sessions with a varied participant base to gather qualitative and quantitative feedback on the tool's usability, effectiveness, and overall user experience.

Feedback Integration: Establish a feedback loop, incorporating user suggestions and criticisms into the development process. Prioritize enhancements that align with user needs and expectations.

Iterative Development: Release updates and new features in an iterative manner incrementally. This approach allows for continuous improvement, ensuring that the tool remains relevant and effective in dynamic user environments.

By executing these proposed methodologies, the Voice Similarity Analysis will evolve into a sophisticated and adaptive tool, expanding its utility across diverse scenarios and user requirements in the realm of audio processing and analysis.

IV. Result

The spectrum of frequencies of a signal as they change over time is shown visually in a spectrogram. It is a twofold plot

which displays how the frequencies of a signal change over time. Time is represented by the x-axis, frequency is represented by the y-axis, and intensity is of the colors or shade at each plot point denotes the frequency's amplitude or power at that particular time and frequency.

Here are the key components of a spectrogram:

Time Axis (X-Axis): This axis represents the progression of time. Each point along the x-axis corresponds to a specific time instant or time frame.

Frequency Axis (Y-Axis): This axis represents the range of frequencies present in the signal. The lowest frequencies are typically at the bottom, and the highest frequencies are at the top.

Color or Intensity: The color or tinting each area within the spectrogram indicates the frequency and time-specific amplitude or power of the frequencies. Brighter colors or higher intensity often correspond to higher amplitude.

Windowing and Fourier Transform: Spectrograms are computed using a series of short-time Fourier transforms (STFT). The The Fourier transform is applied to each of the tiny time periods that the signal is separated into. This process captures the signal's frequency content at various time intervals.

Frequency Resolution and Time Resolution: The width of the frequency bins and the duration of the time windows determine the frequency and time resolution of the spectrogram. Shorter time windows provide better time resolution, while narrower frequency bins provide better frequency resolution.

When we run the code, we have to enter the number of audio files we are going to upload and also we have to give the silmilarity threshold value (0.9). Figure 1 shows the dashboard to upload files.

Figure 1: File upload

Now, let's try the code with real time cases:

Sample 1: Here we are taking 3 voices of same person with different time duration and checking for the result

First audio file is of 15 seconds, second audio file is of 2.5 seconds and third audio file is of 3 seconds. After uploading these audio files, we have to click on analyse button then it shows spectrograms of each audio file respectively with 1 second gap.

Table 1 shows the spectrograms of given audio files

Table no 1: Spectrograms of Audio file

Audio no.	Audio name	Duration
1	Audio-4	15 sec
2	Audio-3	2.5 sec
3	Audio-2	3 sec

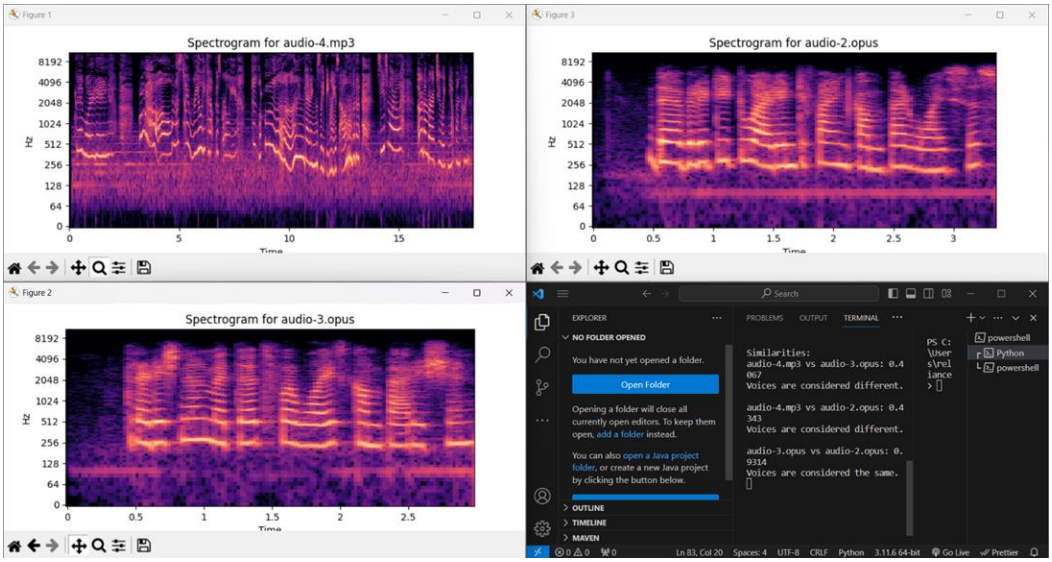

Figure 2: Spectrogram of audio file

Figure 2 shows the spectrograms of given input audio files. Table 2 shows the output as voices.

Table no 2: Output of Voices

Audio number	Audio name	Duration
1	furina	15
2	Hu tau	2.5

Sample 2:

Now, let's give 2 audio files as input. For the first audio file the time duration is 15 seconds and for the second file time duration is 2.5 seconds.

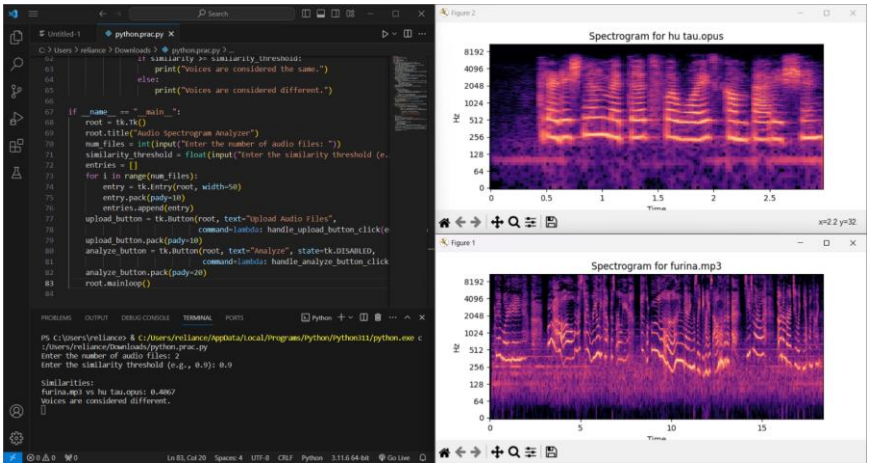

Figure 3: Audio files

Figure 3 shows both the audio files are considered as different.

Sample 3:

Here we are giving four input files and seeing that result(Table 3). First audio file is of 15 seconds, second audio file is of 20 seconds, third is of 3 seconds and the final one, fourth is of 3 seconds.

Table no 3: Input files of audio

Audio number	Audio name	Duration
1	itto	15
2	Hu tau	20
3	Audio-3	3
4	Audio-2	3

The spectrograms are displayed in figure 4. Figure 5 shows the output is “Voices are considered the same”.

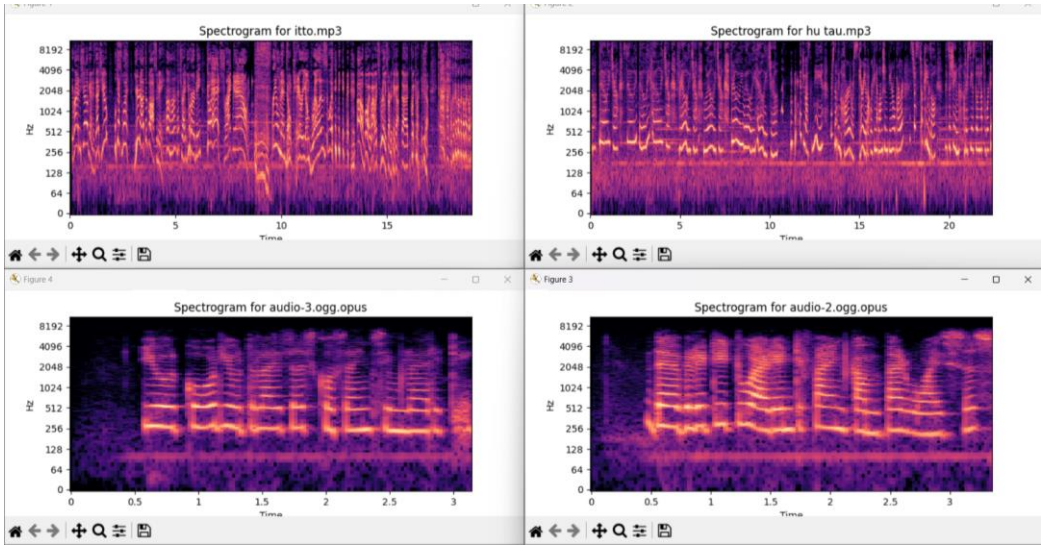

Figure 4: Spectrogram of input files

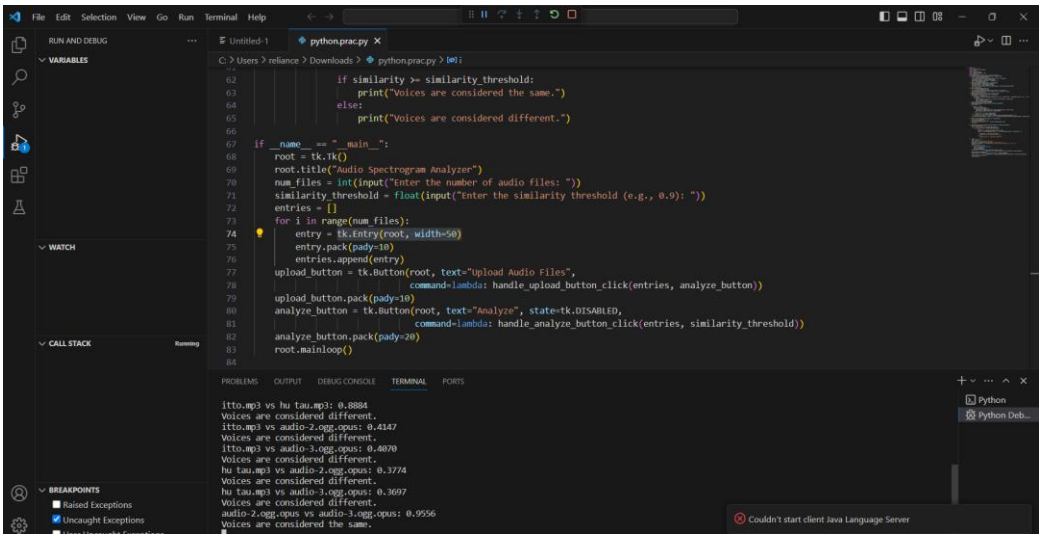

Figure 5: Output of the audio file

IV. Discussion

The use of a spectrogram in voice-related code typically involves extracting features from audio signals for analysis. The spectrum of frequencies of a signal as they change over time is shown visually in a spectrogram. In the In the context of voice-related applications, spectrograms are frequently utilized to transform an audio signal in the time domain into a frequency-domain representation. Here are some common use cases for spectrograms in different domains:

Speech Recognition and Speaker Identification: Spectrograms are used to represent the frequency content of speech signals over time.Deep learning models for speech recognition or speaker identification may take spectrograms as input features.

Voice Transformation and Synthesis: Spectrograms are used to analyze the spectral characteristics of a voice signal.Voice transformation models may manipulate the spectrogram to modify specific characteristics of the voice.

Voice Analysis in Forensic Science: Spectrograms can be used to analyze the characteristics of a voice signal for forensic speaker recognition.Fine details in the frequency domain may be crucial in distinguishing between different speakers.

Voice Comparison and Forensics: Spectrograms can aid in comparing voice signals and identifying unique patterns or characteristics.They are used in forensic voice analysis to visually inspect and compare the spectral content of different voice samples.

Voice Biometrics: Spectrograms may be used as features for voice biometrics applications.The unique spectral patterns in a person's voice can contribute to the creation of a voiceprint for identification.

Here's an overview of the code's functionality:

GUI Setup: The script creates a Tkinter window with entry widgets for users to input the paths of audio files. There are also buttons for uploading audio files and analyzing them.

Spectrogram Computation: The code defines functions to compute and resize spectrograms from audio files. It uses Librosa to load audio files, calculate spectrograms, and resize them to a fixed shape.

Cosine Similarity: The cosine similarity between two flattened spectrograms is computed using scikit-learn's cosine_similarity function. The similarity is then printed, and a decision is made whether the voices are considered the same based on a similarity threshold.

GUI Interaction: Users can upload audio files using the "Upload Audio Files" button. The "Analyze" button is disabled initially and is only enabled after uploading files. Clicking the "Analyze" button triggers the analysis of spectrograms and the computation of similarities.

Visualization: The script includes a function to plot and display spectrograms using Matplotlib.

Event Handling: The code utilizes the Tkinter after method to introduce delays in the analysis process, simulating the simultaneous display of spectrograms and the subsequent calculation of similarities.
The displayed spectrogram is shown based on two parametres. On the x-axis are time, and on the y-axis are hzs.

spectrogram Visualizations:

The quality of spectrogram visualizations is crucial for the tool's effectiveness in audio analysis. The implementation utilizes the librosa library to convert audio signals into time-frequency representations. The choice of a logarithmic frequency scale (y-axis) enhances the tool's ability to capture both low and high-frequency components, providing a more comprehensive view of the spectrograms. The visual representations enable users to identify patterns, distinguish unique characteristics in different voices, and potentially recognize distinct features associated with emotions or speech variations. The implementation of a delay before displaying spectrograms sequentially contributes to a user-friendly experience. This intentional delay allows users to focus on individual visualizations, preventing information overload. However, the effectiveness of this design choice could be further validated through user feedback and usability studies.

Voice Similarity Analysis:

The tool leverages the cosine similarity metric, a common measure in vector space models, to quantify the similarity between flattened spectrograms. The decision to use cosine similarity is appropriate for comparing the overall shapes and patterns of spectrograms. However, It is crucial to remember that whereas cosine similarity offers a helpful measure for voice similarity, it might not capture nuances associated with variations in pitch, tone, or other vocal characteristics. The presented results in the console provide a clear indication of the similarity scores between pairs of audio files. However, to enhance user experience, future iterations of the tool could implement a graphical representation of the similarity matrix, offering a more intuitive and comprehensive overview of relationships between all uploaded audio files.

V. Conclusion

The Voice Similarity Analysis stands as a functional tool for audio analysis, catering to both qualitative and quantitative aspects of voice characterization. The discussion highlights the importance of user experience in spectrogram visualizations, emphasizing the need for clear and informative representations of audio content.
Moving forward, the tool could benefit from additional features such as:

Real-time Analysis: Enabling users to analyze audio streams in real-time could expand the tool's applicability, especially in scenarios requiring immediate feedback.

Machine Learning Integration: Integration with machine learning models could enhance the tool's ability to automatically identify speakers or classify emotions based on audio content.

Diverse Audio Format Compatibility: While the tool currently supports WAV, MP3, and OGG formats, future developments could focus on expanding compatibility with a broader range of audio file formats.

References

[1]. Librosa: Mcfee, B., Raffel, C., Liang, D., Ellis, D. P., Mcvicar, M., & Battenberg, E. (2015). Librosa: Audio And Music Signal Analysis In Python. In Proceedings Of The 14th Python In Science Conference (Pp. 18-25).

[2]. Tkinter:Python Software Foundation. (2023). Tkinter Documentation. Retrieved From Https://Docs.Python.Org/3/Library/Tkinter.Html

[3]. Matplotlib:Hunter, J. D. (2007). Matplotlib: A 2D Graphics Environment. Computing In Science & Engineering, 9(3), 90-95.

[4]. Numpy: Harris, C. R., Millman, K. J., Van Der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., ... & Oliphant, T. E. (2020). Array Programming With Numpy. Nature, 585(7825), 357-362.

[5]. Scikit-Learn (For Cosine_Similarity): Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Vanderplas, J. (2011). Scikit-Learn: Machine Learning In Python. Journal Of Machine Learning Research, 12(Oct), 2825-2830.

[6]. Voice Biometrics: Reynolds, D. A., & Rose, R. C. (1995). Robust Text-Independent Speaker Identification Using Gaussian Mixture Speaker Models. IEEE Transactions On Speech And Audio Processing, 3(1), 72-83.

[7]. Deep Learning In Speaker Recognition: Heigold, G., Vanhoucke, V., Senior, A., & Nguyen, P. (2016). End-To-End Text-Dependent Speaker Verification. In Acoustics, Speech And Signal Processing (ICASSP), 2016 IEEE International Conference On (Pp. 5115-5119). IEEE.

[8]. Voice Transformation And Similarity: Stylianou, Y. (1996). Applying The Harmonic Plus Noise Model In Concatenative Speech Synthesis. In Proceedings Of The International Conference On Acoustics, Speech, And Signal Processing (ICASSP) (Vol. 1, Pp. 373-376). IEEE.

[9]. Voice Comparison And Forensics: Auckenthaler, J., Carey, M., Lloyd-Thomas, H., Mccree, A., & Thean, A. (2004). Voice Comparison And The Acceptability Of Forensic Speaker Comparisons In The UK. Forensic Linguistics, 11(2), 133-149.

[10]. Voice Analysis In Forensic Science: Rose, P., & Morrison, G. S. (2017). A Critical Analysis Of The Value Of Evidence In Forensic Speaker Recognition For Judicial Purposes. Science & Justice, 57(1), 17-28.