Speaker Recognition for Mobile Applications

Introduction

This project aimed to create a speaker identification application that is easily implementable for app developers. My motivation for this project stems from its potential to significantly improve accessibility for older individuals and those who are visually impaired. By implementing speaker identification, traditional authentication barriers can be overcome. In essence, the project aims to contribute to a more inclusive and user-friendly digital landscape, ensuring that diverse user populations can engage seamlessly with technology. When creating a biometric system it's important to understand what kind of applications would benefit from such. Given that the goal is to allow users to authenticate themselves based on voice recognition and cater settings to them, any apps that use unique user profiles could benefit greatly from such a system.

Project Plan

When originally brainstorming the layout of the system, I immediately knew that I wanted to use a GUI for demonstration purposes as this would allow for easy navigation of the project's features. The original diagram for the program was high-level and allowed me to divide the entire project into 3 components. These were the main menu, enrolling, and authenticating:

```
Ask user if they want to enroll or authenticate -> wait 5 seconds -> default to authenticate

Enroll -> ask the user their name -> ask the user to record their passcode -> ask the user to select their favorite color -> store all of these in the database

Authenticate -> ask the user to say their passcode -> extract features and match with database -> display the user's name and their favorite color
```

Key features that I wanted to include in the system were:

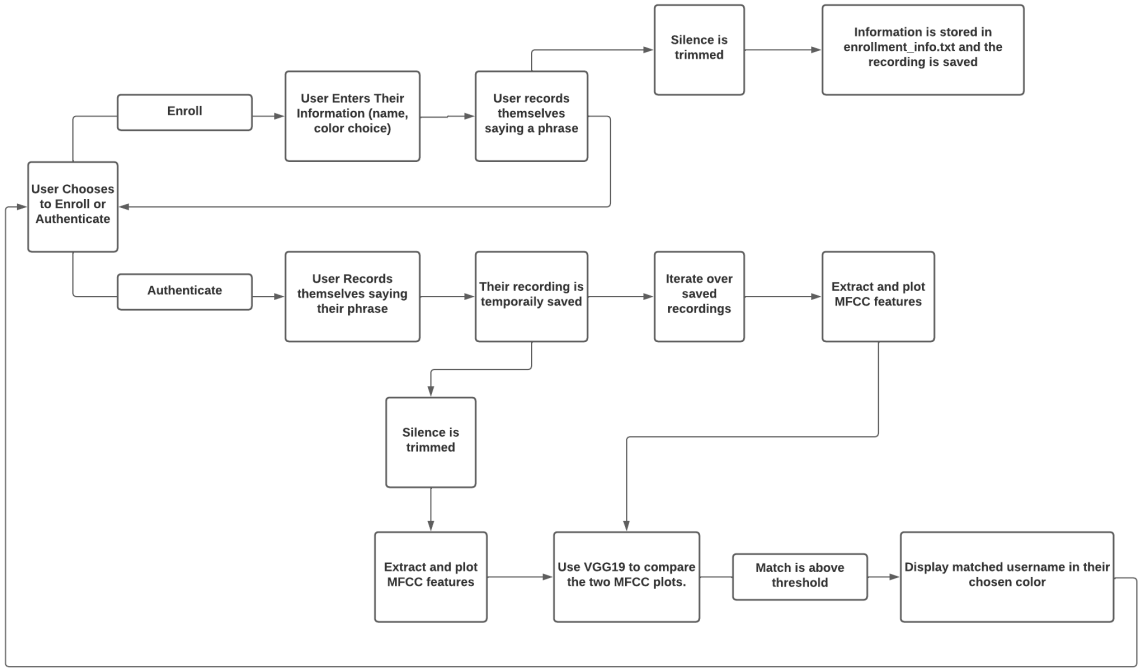
- Having the user be able to say their own phrase
- Display the user's name and color upon authentication
- Allow the user to easily navigate between the enrolling and authentication processes without having to restart the entire system.

Implementation

The final implementation of the system asks the user if they want to enroll or authenticate. Then if the user chooses to enroll, they are asked to input their name, choose a color, and record their phrase. We then trim the silence from the recording using PyDub and the information is stored in enrollment_info.txt in this format:

```
Color: #0000ff
Audio File: data\Joseph_enrollment_recording.wav
```

If the user chooses to authenticate they record themselves saying their phrase, the silence gets trimmed and it is saved. The recording then gets compared to every saved enrolled audio recording. Whichever saved recording has the highest similarity score and is above the set threshold, we print that username in the respective color.



Pipeline Diagram of the program.

- Authentication
    - After creating the basic GUI, I quickly realized that my original plan to use the SpeechRecognition library fell short as this library only supported recognizing speech and transcribing it. Thus, to use this library I would have had to pivot my program to recognizing a user's special phrase and authenticated based on if the two phrases matched. However, I thought that this could be easily defeated as all you needed to trick the system was someone's pass-phrase. So, I stuck to my original plan and instead looked for libraries to help with speaker identification so that even if two people said the same phrase, the system would only authenticate if they also sounded the same. To do this, I wanted to compare the user's audio recording when they enrolled and their audio recording when they chose to authenticate based on their waveforms. I elected to use cosine similarity to compare the two but upon implementation, I realized this was not possible as the waveforms are stored in a unique format. The final authentication method used is based on the plots of the MFCC features of the waveforms. In the current implementation of the program, we extract the MFCC features from each of the waveforms, plot them, and then convert them to black and white. Then, we use VGG19 to compare the two plots.

- Trimming and Saving Audio Recordings
    - In class, we discussed the importance of preprocessing information like pictures or audio signals to maximize the accuracy of any biometric system. So, I knew that to maximize the accuracy of this system, I had to trim the silence from any of the audio recordings as this would minimize the error within the authenticator. I researched different ways to do this but ultimately selected using PyDub's strip_silence method. Also, to make the process faster, I didn't want to store the user's authentication recording in memory as we only needed it for the authentication process. However, I quickly realized that this was not possible due to how the recording was temporarily stored in the program. Its format would not allow me to pass it to the authentication method easily and as such, I had to store it temporarily. Whenever the authentication process is initiated and the user

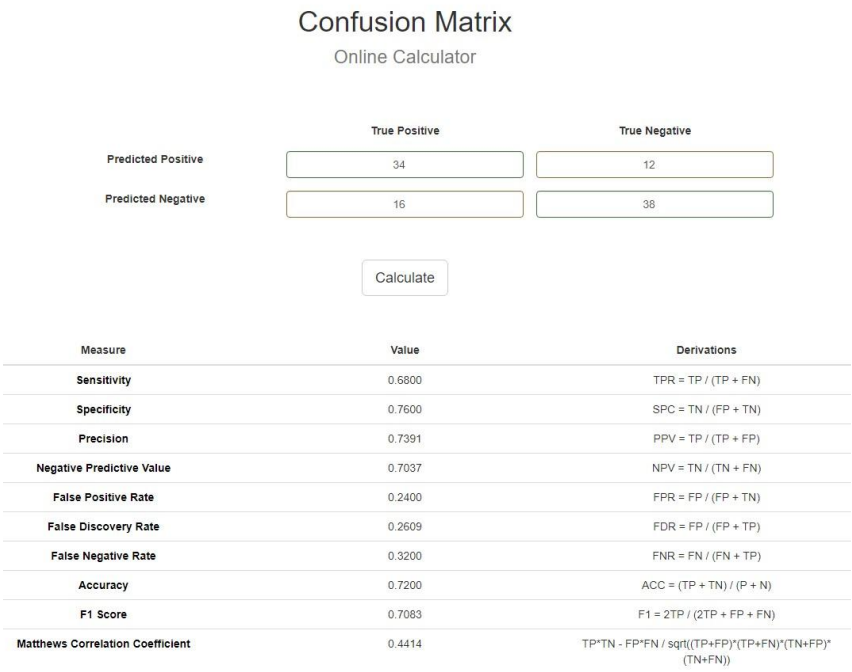
records their authentication phrase, the old one is deleted. Also for the purpose of privacy concerns, no names are stored or linked to the temporary audio recordings.

To make the program more understandable to others, I chose to break down the entire program into 5 major components/ files. These are Main.py, Gui.py, Gui_helper.py, Authentication.py, and Audio_processing.py. All of these files are integral to the system's functionality and uniquely interact with each other to formulate the biometric system.

Main.py is responsible for creating the instance of the Tkinter GUI.
Gui.py is the file that actually drives the program. In this file, we create all the frames of the GUI and control the navigation between them. This is also the file in which the BiometricSystem class is defined and serves as the entry point for the entire program. This class is responsible for calling all the functions defined in the other files.
Gui_helper.py contains helper functions used in Gui.py. More specifically the functions responsible for displaying the matched user's name and getting rid of it after 10 seconds.
Authentication.py contains the functions that do the actual matching and authenticating of the audio files - Match_recordings which iterates over the saved audio recordings and compare_audio which uses VGG19 to compare the MFCC plots.
Lastly, Audio_processing.py contains the trim_silence function that uses PyDub to trim the silence from the audio recordings before they're saved.

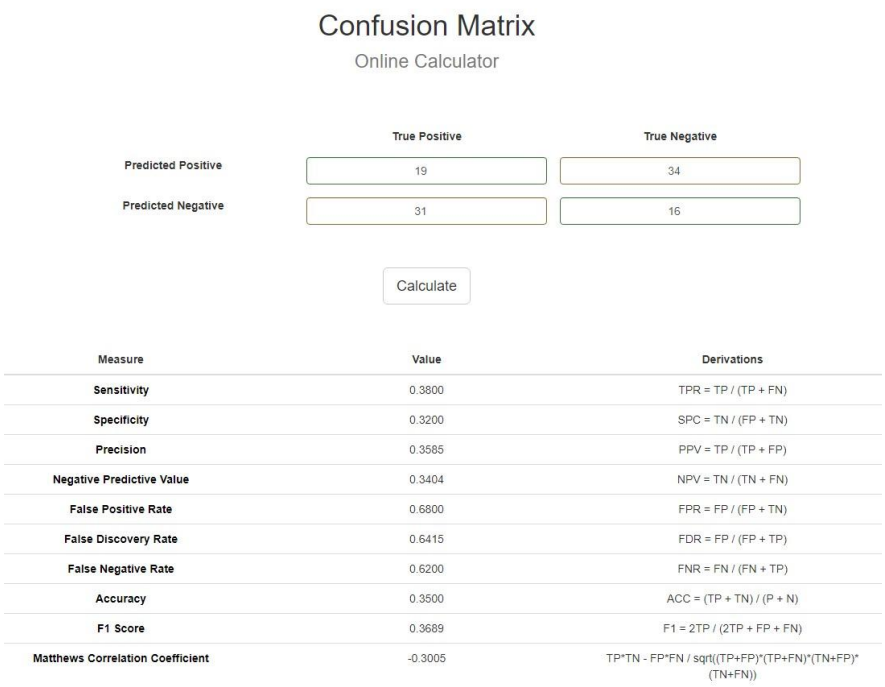
Experiments and Results

To experiment with the system I tested it 100 times with 50 Genuine attempts and 50 Imposter attempts. For Genuine attempts, I said my phrase 50 times and if the system correctly identified me, I counted it as a positive, and if it incorrectly identified me, a negative. For the imposter attempts, I randomly selected an enrolled user and if it correctly identified them I counted it as a predicted negative. If incorrect, a predicted positive.



Confusion Matrix with 3 Enrolled Users

After the 100 test attempts, I determined the accuracy of the system to be 72% with a 24% FAR and a 32% FRR. During my original presentation, I demonstrated that to count the system as successful, I would want the accuracy to be higher than that of face detection. However, the accuracy of this system was lower than face detection so I know it can still be improved. Nonetheless, I was pleased with the results as the accuracy was high enough to be a plausible system implemented within apps. In its current state, the system works as a proof of concept. As I was testing the system, I wondered what would happen if there were more enrolled users. As such, I enrolled 3 other users making the total enrollment count 6. I then tested the system the same way as before but this time the results were very different.



Confusion Matrix

Online Calculator

	True Positive	True Negative
Predicted Positive	19	34
Predicted Negative	31	16

Calculate

Measure	Value	Derivations
Sensitivity	0.3800	TPR = TP / (TP + FN)
Specificity	0.3200	SPC = TN / (FP + TN)
Precision	0.3585	PPV = TP / (TP + FP)
Negative Predictive Value	0.3404	NPV = TN / (TN + FN)
False Positive Rate	0.6800	FPR = FP / (FP + TN)
False Discovery Rate	0.6415	FDR = FP / (FP + TP)
False Negative Rate	0.6200	FNR = FN / (FN + TP)
Accuracy	0.3500	ACC = (TP + TN) / (P + N)
F1 Score	0.3689	F1 = 2TP / (2TP + FP + FN)
Matthews Correlation Coefficient	-0.3005	TP*TN - FP*FN / sqrt((TP+FP)*(TP+FN)*(TN+FP)*(TN+FN))

Confusion Matrix with 6 Enrolled Users

With 6 enrolled users the accuracy of the system dropped to a staggering 35% with a 68% FAR and a 62% FRR. When testing, I noticed that users were consistently getting misidentified for another user I believe that this is due to our authentication method. The MFCC plots are too similar between some users, causing the system to think they are the same user.

Future Directions

Given the time frame, I think that this was a very successful project as we have a proof of concept. In the future, I aim to improve the accuracy of the program by exploring methods to compare the actual audio waveforms for a more precise speaker identification. An alternative way to improve accuracy may be to incorporate color information into the MFCC plots, potentially providing more data for VGG19 to compare. This step could unlock a more nuanced Furthermore, I want to host the program on a web server, facilitating its utilization on mobile devices. Lastly, I would want to modify the GUI so that it's more catered toward mobile devices and make integration with mobile applications easier.

Conclusion

Our speaker identification project has succeeded in creating a user-friendly application that uses voice recognition to load a user's preset settings. The graphical interface makes it easy for users to sign up and verify their identity effortlessly. By using methods like MFCC plots and VGG19, we achieved a good accuracy rate, proving the system's potential for real-world applications. However, as more users were enrolled, challenges appeared - signaling areas for improvement. Looking ahead, we aim to refine the system by exploring ways to compare actual voice recordings for better accuracy. Adding color details to our methods and hosting the system on a web server for mobile use are exciting steps for the future. This project is a strong starting point, demonstrating how speaker identification can enhance user settings and accessibility.

Citations

1. Lundh, F. (1999). An introduction to tkinter. URL: Www. Pythonware. Com/Library/Tkinter/Introduction/Index. Htm.
2. Robert, J., Webbie, M., & others. (2018). Pydub. GitHub. Retrieved from http://pydub.com/
3. Paszke, Adam and Gross, Sam and Chintala, Soumith and Chanan, Gregory and Yang, Edward and DeVito, Zachary and Lin, Zeming and Desmaison, Alban and Antiga, Luca and Lerer, Adam. (2017). Automatic differentiation in PyTorch. NIPS-W
4. OpenAI. (2022). GPT-3 Language Model.
5. Harris, C.R., Millman, K.J., van der Walt, S.J. et al. Array programming with NumPy.
6. Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., … SciPy 1.0 Contributors. (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. Nature Methods, 17, 261–272. https://doi.org/10.1038/s41592-019-0686-2
7. Geier, M., & others. (2020). Sounddevice. GitHub. Retrieved from https://python-sounddevice.readthedocs.io/en/0.3.15/index.html