**Final Project Report**

Our project is to realize a voiceprint recognition and speech recognition. Usually, there are two modes for voiceprint recognition, speaker identification and speaker verification. The former is to judge who the sample voice belongs to in a range of candidates while the latter is to judge if the sample voice is from specific person. We realize the speaker identification in our project to detect the speaker in a close set which all voices have been trained. In this close set, the system supposes the sample voice is in the set and compare the probability of the sample voice to the voice in the set to determine the speaker. And the other two modes for voiceprint are text-dependent and text-independent. Text-dependent requires speakers to pronounce with specified content while text-independent doesn’t fix the content. We let speakers speak with arbitrary content and build the voiceprint model base on it.

**Voiceprint Recognition**

The voiceprint recognition has two key parts, one is feature extraction and the other one is pattern identification.

**Feature Extraction**

The mfcc is the process to extract the feature from human voice, and convert it to vectors. The mfcc feature is more close to human being’s ear feature, so it’s often used in human voice character extraction.

The step of extracting using mfcc has the followings:

1. Frame the signal into short frames.

In this process, we will frame the voice data to 25ms standard, and each frame will overlap with each other a little bit.

2. For each frame calculate the periodogram estimate of the power spectrum.

Then we will use the Discrete Fourier Transform on the frame. We take the absolute value of the complex Fourier transform, and square the result. We would generally perform a 512 point FFT and keep only the first 257 coefficients.

3. Apply the mel filterbank to the power spectra, sum the energy in each filter.

This is a set of 20-40 (26 is standard) triangular filters that we apply to the periodogram power spectral estimate from step 2. Our filterbank comes in the form of 26 vectors of length 257 (assuming the FFT settings from step 2). Each vector is mostly zeros, but is non-zero for a certain section of the spectrum. To calculate filterbank energies we multiply each filterbank with the power spectrum, then add up the coefficients. Once this is performed we are left with 26 numbers that give us an indication of how much energy was in each filterbank.

4. Take the logarithm of all filterbank energies.

This will give us the result of 26 filterbank energies

5. Take the DCT of the log filterbank energies.

This will generally give 26 cepstral coefficents

6. Keep DCT coefficients 2-13, discard the rest

7. Computing the Mel filterbank

8. Calculate delta coefficients

After the following steps, we can get the target voice feature information, and for mfcc extraction procedure, we used a python package to realize it.

**Pattern Identification**

The second part is pattern identification. There are several methods to process the identification, such as Vector Quantization, [hidden Markov models](https://en.wikipedia.org/wiki/Hidden_Markov_model), [Gaussian mixture models](https://en.wikipedia.org/wiki/Gaussian_mixture_model), [pattern matching](https://en.wikipedia.org/wiki/Pattern_matching) algorithms, [neural networks](https://en.wikipedia.org/wiki/Neural_networks) and [decision trees](https://en.wikipedia.org/wiki/Decision_tree_learning). We use [Gaussian mixture models](https://en.wikipedia.org/wiki/Gaussian_mixture_model) here to deal with the voice features.

A Gaussian mixture model is a probabilistic model that assumes all the data points are generated from a mixture of a finite number of Gaussian distributions with unknown parameters. One can think of mixture models as generalizing k-means clustering to incorporate information about the covariance structure of the data as well as the centers of the latent Gaussians. Gaussian mixture model can also be a continuing state Hidden Markov Model (HMM). Using GMM in voiceprint recognition is to build a GMM for each speaker while training the voices. The difference between the different speakers is mainly manifested in the difference in the short-term speech spectrum, which in turn can be measured by the probability density function of each speaker's short-time spectrum. The Gaussian mixture model sum the probability density of the spatial distribution with the weighted of multiple Gaussian probability density functions. Thus, it can smoothly approximate the probability density function of arbitrary shape. To build the model for each speaker, the system estimates parameters of GMM.

Usually, it uses [expectation-maximization](http://scikit-learn.org/stable/modules/mixture.html#expectation-maximization) (EM) algorithm to estimate the maximum likelihood estimation. Here we implement a python package called ‘sklearn’ to realize the GMM. The ‘sklearn.mixture’ is a package which enables one to learn Gaussian Mixture Models, sample them, and estimate them from data. The [GaussianMixture](http://scikit-learn.org/stable/modules/generated/sklearn.mixture.GaussianMixture.html#sklearn.mixture.GaussianMixture) object implements the [expectation-maximization](http://scikit-learn.org/stable/modules/mixture.html#expectation-maximization) (EM) algorithm for fitting mixture-of-Gaussian models. It can also draw confidence ellipsoids for multivariate models, and compute the Bayesian Information Criterion to assess the number of clusters in the data. A [GaussianMixture.fit](http://scikit-learn.org/stable/modules/generated/sklearn.mixture.GaussianMixture.html#sklearn.mixture.GaussianMixture.fit) method is provided that learns a Gaussian Mixture Model from train data. And it uses GaussianMixture.score method to get the level of each model. Given test data, it can assign to each sample the Gaussian it mostly probably belong to using the [GaussianMixture.predict](http://scikit-learn.org/stable/modules/generated/sklearn.mixture.GaussianMixture.html#sklearn.mixture.GaussianMixture.predict) method.

**Speech Recognition**

The second part in our project is speech recognition. Since we use text-independent speaker identification, speakers could say anything to the system. Thus, we would like to transcribe the audio input to a text output to build a question and answer model as Siri. We import in our system with an open source API for python which is Google Speech Recognition API. This API converts spoken text (microphone) into written text (Python strings), briefly Speech to Text. Speaker can simply speak in a microphone and Google API will translate this into written text. The API has excellent results for English language. And we also register a Google Cloud platform to use Google Cloud Speech API to transcribe the speech. This API enables developers to convert audio to text by applying powerful neural network models in an easy to use API. The API recognizes over 80 languages and variants, to support the global user base.

With the Speech API, we implement two modes, audio transcription and microphone transcription. In audio transcription, the system read the local prepared audio files as the input and output the text in a good accuracy. Our prepared audio files are recorded in a quiet environment. Thus, the result of audio transcription could transcribe the text correctly and also recognize the speaker correctly. While, in the microphone transcription mode, we first use Google record method to record the real time audio, but it gets poor audio file and recognition accuracy. So, we implement record method using ‘PyAudio’ and fix the sample rate, channels and record seconds. ‘PyAudio’ records the microphone voice to local and read the audio file to Google recognition to output the text. In this way, the audio would sound in a normal way if the computer is not in a lag phase and the accuracy will be improved.

What’s more, we would like the system to receive and answer the input voice like Siri. We implement a ‘espeak’ API to let the system pronounce, and write some key words to let it recognize to make relative answers. These answers include simple daily Q & A and actions as opening website, playing music, recording voice etc. Though it cannot be intelligent as the real AI and Siri, our system is taking shape.

**Interface**

We build the interface using pyqt5 in python3. At first we tried to use Qt Designer to build a ui file and use “pyuic5 -x filename.ui -o filename.py” command to convert the ui into py. But after using that we find that it cannot convert the file into pyqt5-pattern completely because of having some patterns in pyqt4. So, we tried to use the ui file directly in the project. There are two method for introducing and processing. One is Compile Time Form Processing. The other is Run Time Form Processing using the QtUiTools model to load the ui file dynamicly. But it still faced some pattern problem with pyqt5. So eventually, we decided to handwrite the interface code. Absolute layout is used to position the widget in the frame. When you click the button, it will use the singnal/slot to link the button to the specific override slot function. Finally, we used the ReadOnly-TextEdit to show the information that the program prints to the console.

**Print the console to GUI**

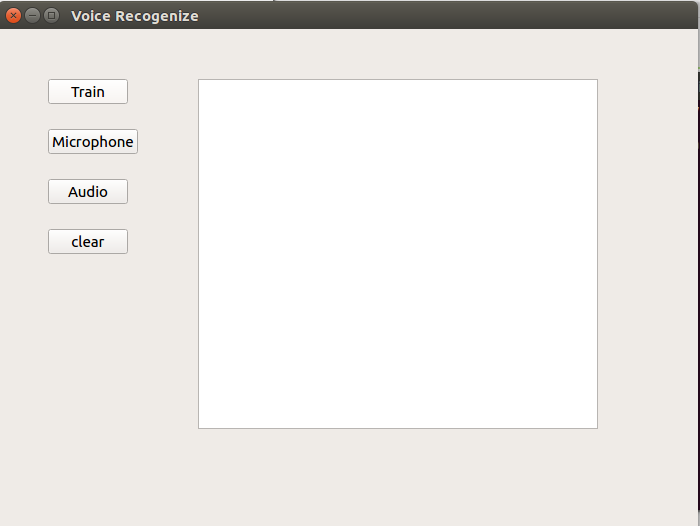
1. Import all the useful py file and import all the functions of that
2. Using the \_\_del\_\_(self) to restore sys.studout and sys.sterr.
3. Building the def normalOutputWritten(self,text) to append the text to the QTextEdit. It will append the text after finding the cursor.
4. Create the class EmittingStream to get the textWrittern signal, and store that into sys.studout
5. After clicking the specific button, we use a QProcess to start the specific action-function

**Problem and solutions of Print**

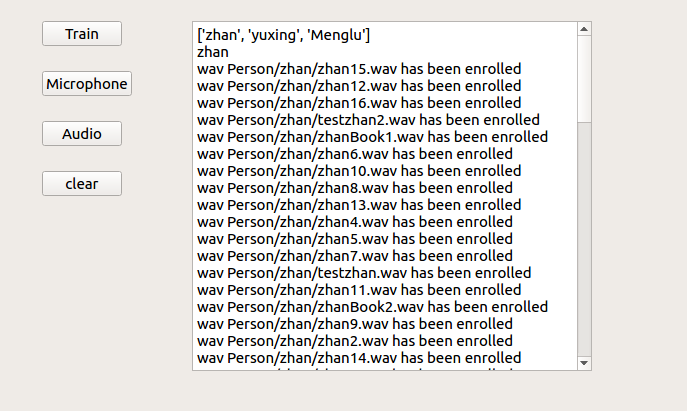
Qt provide a QProcess class for starting and communicating with external programs. In this part, we tried to using the class.function to load function first. But the problem is the program will print all the things of output after the whole function run completely. That means we cannot get the prompt message. And at out project, we cannot input our sound in step by step. So, we add the QApplication.processEvents() after the print event. This function handles the untreated events within 3 seconds or until no more events to process. It depends on which is more short. After using that we can print the output step by step.

**Screenshots**

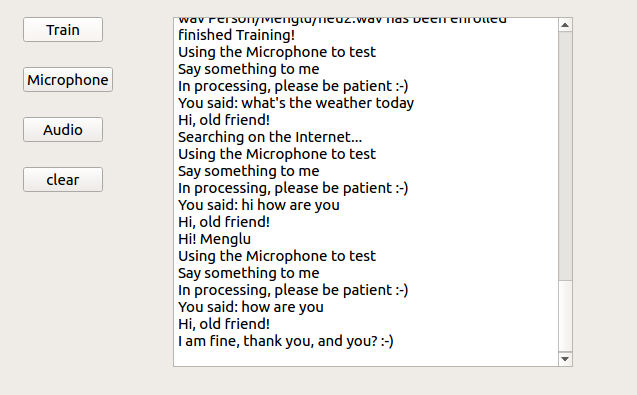
**Initial interface**

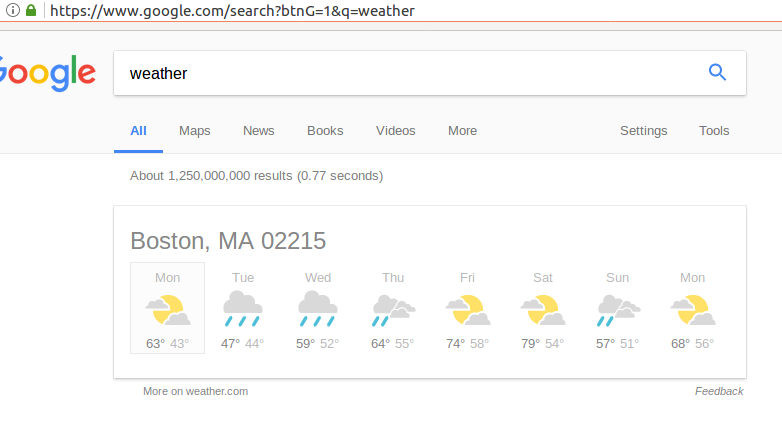


**Train interface**

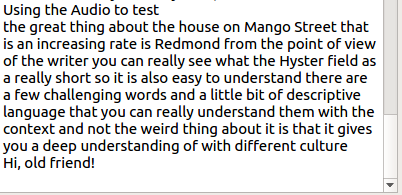


**Microphone Test interface**





**Audio Test interface**



**Future Scope**

Since we only write some simple key words and answer for the system, the Q & A system is not very intelligent. We had search for Q & A API for python. IBM Watson used to be a useful Q &A API, but it stopped it service a few years ago. It separated the original service into four other services and we need to combine the different methods and API to realize the Q & A function. We think our system could be more intelligent using the suitable methods.

Challenge: noise