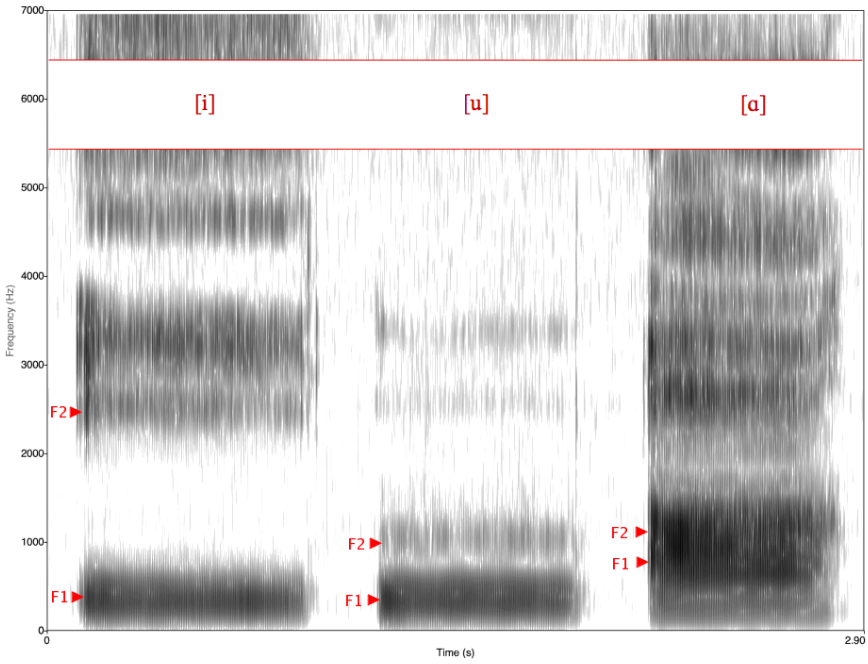
Wiki Article: Speech Signal Processing and Speech Recognition in Python

Introduction

Training your own speech-to-text model for the means of your project is a very daunting task and there are many factors to consider in speech signal processing — noise, reverb, volume, distance, etc., that will determine the performance of said model. Thankfully, the SpeechRecognition library for Python built by Anthony Zhang simplifies this task. In this article, we will learn how to conduct speech-to-text prediction in Python, and what to consider when creating commands for your project.

Background on Speech Signals

In theory, speech signals are simple signals that if we were to take the fourier transform and plot the frequency vs. time of the signal, we would be able to breakdown the constants, vowels, fricatives, and stops of the signal. With these considerations, creating your word cloud of speech commands should take word structure in terms of the fourier transforms into account. As shown below, we have a graph of the frequency vs. time of vowels.



Wikipedia. *Formant*

We are easily able to see the formants (thick cluster of frequencies marked by F1 and F2), and with this we are able to deduce what the sound is, and also what kind of stops are between words. Formants are directly related to the size and shape of your vocal tract, which creates a frequency response shown above. By analyzing these formants, we can gain information about the type of sound and speech being produced. As such, when thinking about your word cloud, find words that have very different structure.

Getting Started

- **Installing**
  - To install the library, type the following command in Pip:
  - pip install pyaudio
    - This is only required if you need microphone input
  - pip install SpeechRecognition
  - For further instructions on installation, see the PyPI documentation found here.

Using the Library

The script used in this section that I wrote will be included.

In this section, we will be writing a simple script on speech-to-text script that will conduct speech-to-text inferencing and also will be used in the next section on creating clean data. The entire script can be found at the bottom of the article.

To begin, we import the libraries used and create an instance of our recognizer.

Then, we decide on what recognizer to use. My recommendation would be to use the IBM speech-to-text recognizer or Google Web Speech API which we will set our recognizer instance to. There are 5 others, and all of these require some form of setup or account creation, but the Google one allows us to import it and quickly get going. It's very important to recognize that the recognizer we set requires an internet connection when moving forward.

```
# Imports
import speech_recognition as sr
import numpy as np
```

Next, we need to create a recognizer and microphone instance.

```
#create recognizer
r = sr.Recognizer()

#create mic
mic = sr.Microphone()
```

We are all setup to perform speech-to-text inferencing. The following function can be called at anytime to perform inferencing at a time of your choice.

```python
def inference(recognizer, microphone):
    with microphone as source:
        recognizer.adjust_for_ambient_noise(source)
        audio = recognizer.listen(source)

    response = {
        "success": True,
        "error": None,
        "transcription": None
    }

    try:
        response["transcription"] = recognizer.recognize_google(audio)
    except sr.RequestError:
        response["success"] = False
        response["error"] = "API Unavailable"
    except sr.UnknownValueError:
        response["error"] = "Unable to recognize speech"

    return response
```

We then can call the function as so:

```python
#inference using the function we built
words = inference(r, mic)

#produce word cloud
print(words["transcription"])
```

Here are the results. Now let's try a more difficult word to inference on.

```
result2:
{   'alternative': [{'confidence': 0.95001483, 'transcript': 'hello'}],
    'final': True}
hello
```

```
result2:
{   'alternative': [   {'confidence': 0.63342822, 'transcript': 'wood'},
                       {'transcript': 'would'}],
    'final': True}
wood
```

My intentions were “Would” and the recognizer produced would. In the next section, we will discuss ways to control what words we find and how we can manage this.

What to Consider in Your Project

As you have noticed already, running an inference on a microphone input does offer some form of error. In this section, we will find methods on reducing the noise and ensuring we are feeding clean audio signals into our model.

- Energy Thresholding
    - This is an included property in recognizer_instance. As you are aware, the lab is a loud place and successfully being able to conduct speech recognition may be difficult. Energy thresholding will allow you to set how strict the recognizer is when it starts recognition. This will be useful to tweak and test values (from 50 to 4000 according to the documentation) in order to see what minimizes the speech signals picked up by your microphone that we do not want.
- Confidence Value
    - When the inference is complete, you’ll see a list of words alongside their confidence value that the model produces. If we are attempting to make the prediction on the word: “Would,” we may see words with similar structure such as “wood,” “could,” “woot,” etc., with varying confidence. Because of this, you may find success in setting a specific threshold value for each word. Below is an example of an inference on the word “would.” Here, I would set a threshold value of 0.XX, and accept the command if “would” is above this confidence value. However, using words that have similar sounding words will be detrimental, and making sure each of your words have unique structure will create a phenomenal inference procedure.
- Noise Reduction
    - Although this is somewhat covered by energy thresholding, there are further methods by outside libraries that will help reduce noise. Energy thresholding will make sure the microphone picks up at the correct time, but it will not account for static microphone noise. A filter I can recommend that I’ve worked with before is the Savitzky-Golay filter, which is easily importable with the Scipy toolkit.

References:

1. Speech Recognition Documentation: https://pypi.org/project/SpeechRecognition/

2. Savitzky-Golay Filter: https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.savgol_filter.html
3. Wikipedia Formant: https://en.wikipedia.org/wiki/Formant