Pronunciation Checking with ASR

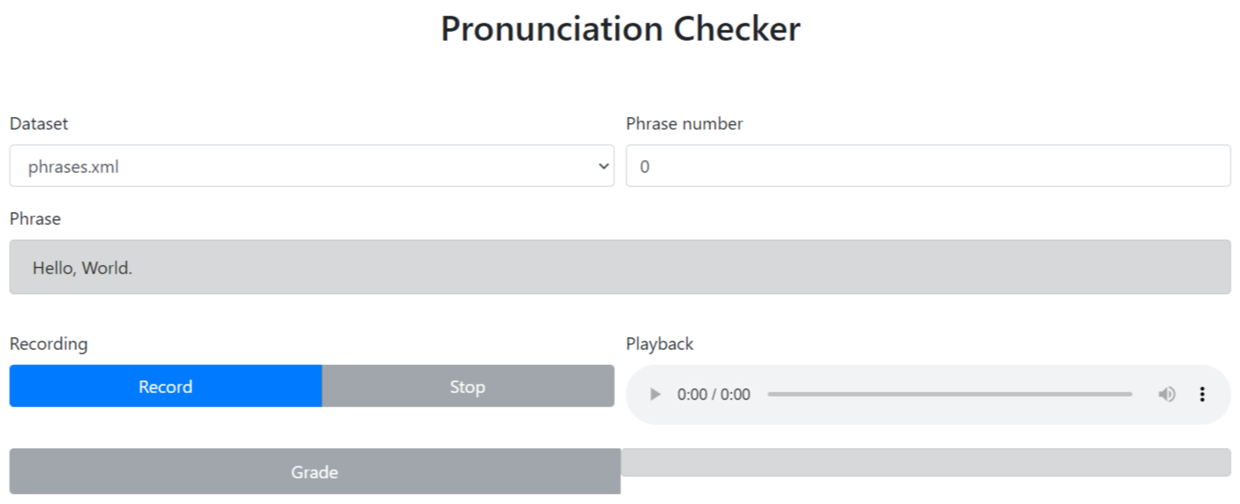
User Specification

Evan Nichols

MFF 2021/2022

What Is It?

This application allows the user to test their pronunciation of various English sentence prompts. It supports multiple concurrent users. Users record themselves, via the web page, and then wait for their pronunciation to be evaluated. After a short delay, they may pull up their score and review, or try again. It uses a neural network module built for automatic speech recognition to evaluate the speech. The module compares the users' spoken phonemes to those of the text prompt (converted from the text), and compares the differences. The application is built using Flask, a web framework for Python, with a web page run in JavaScript.



How to Use this Appplication

Upon loading the web page, the user is presented with a simple interface of a few fields, and will be prompted by most browsers for microphone access. Immediately, the **Record** button may be pressed. This will begin recording, and the user should read the **Phrase** text out loud. After speaking the phrase and pressing **Stop**, the recorded audio is sent to the server-side along with the text prompt and processed and packaged into a data set. This data set is then processed by an automatic speech recognition module and the user's speech audio is graded against the prompt phrase. A score, marking how well the user pronounced the prompt, is stored in a text file and once when the user presses **Grade** this score is output to them. To add varying challenge, the user may choose from a number of phrases, take from several data sets.

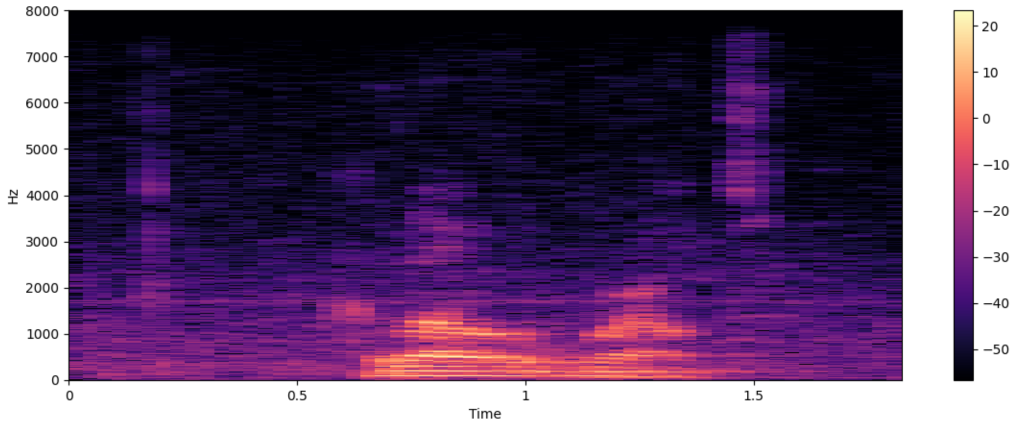
Phrase Prompts

The phrases can be any English words and sentences. The prompts do not NEED to be legitimate English, but for the sake of pronouncing words well it makes more sense for them to be actual language. Special characters are irrelevant in the grading, as the program strips items such as '!' and ':' before passing the prompt key to the ASR module. The phrases included in the app as of this writing are taken from various books and poems.

Input Audio

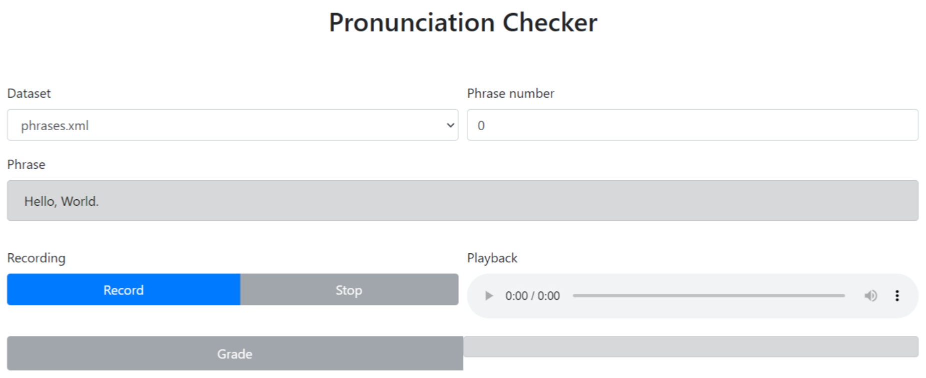
The user's speech speech is recorded, the raw data is sent to the back-end and then reformatted to a *.wav* file with a sampling rate of 16000 and 1 audio channel. A **spectrogram** is also created which displays some information to the user. The spectrogram measures frequency (y-axis) over time (x-axis), and the intensity of the color marks the amplitude of the frequency at that moment.

Figure 1: Spectrogram generated for input speech "hello world"



User Interface

- **Dataset** is a drop-down list of the available phrase libraries. Each phrase is an item in an XML file with a single id, which denotes the items position in the file (1, 2, 3, etc.
- **Phrase number** is a simple selector, selecting up or down pulls the corresponding phrase from the currently selected library. The numbers will loop around the first and last items for seamless selections.
- **Phrase** is a text box that displays the currently selected phrase. It is this text the user is expected to read out loud. The phrase is sent along with the audio to the ASR module, but before that it is converted to its phonetic representation (phonemes) and stripped of any non-alphabetic characters, eg. '-', or '!'.
- **Record** is a button that begins recording, assuming the user has given microphone access to the browser when prompted. The user should read the text in **Phrase** out loud, and hit **Stop** when they are finished speaking. Hitting **Record** again (after **Stop**) will overwrite any previous input, allowing the user to adjust their submission.
- **Stop** closes the recording and, behind the scenes, sends the audio and prompt to the back-end. It also activates the button **Grade** and the **Playback** box, which were deactivated by default upon loading the page.
- **Playback** allows the user to listen to whatever was recorded, and they may record again if they are not satisfied.
- **Grade** pulls the results from '*uuid*_graded.txt', and displays them to the right. The display will list the recognized phonemes (what the user submitted), the correct phonemes (from the text prompt), and the percentage score marking how well the user pronounced the prompt. Since the ASR module will run in variable time, based on the size of the input data, a slight delay might occur between the user pressing **Stop** and **Grade** returning the appropriate results. A simple message will tell the user to try again if their first press was too soon.



Program Files

app.js index.html

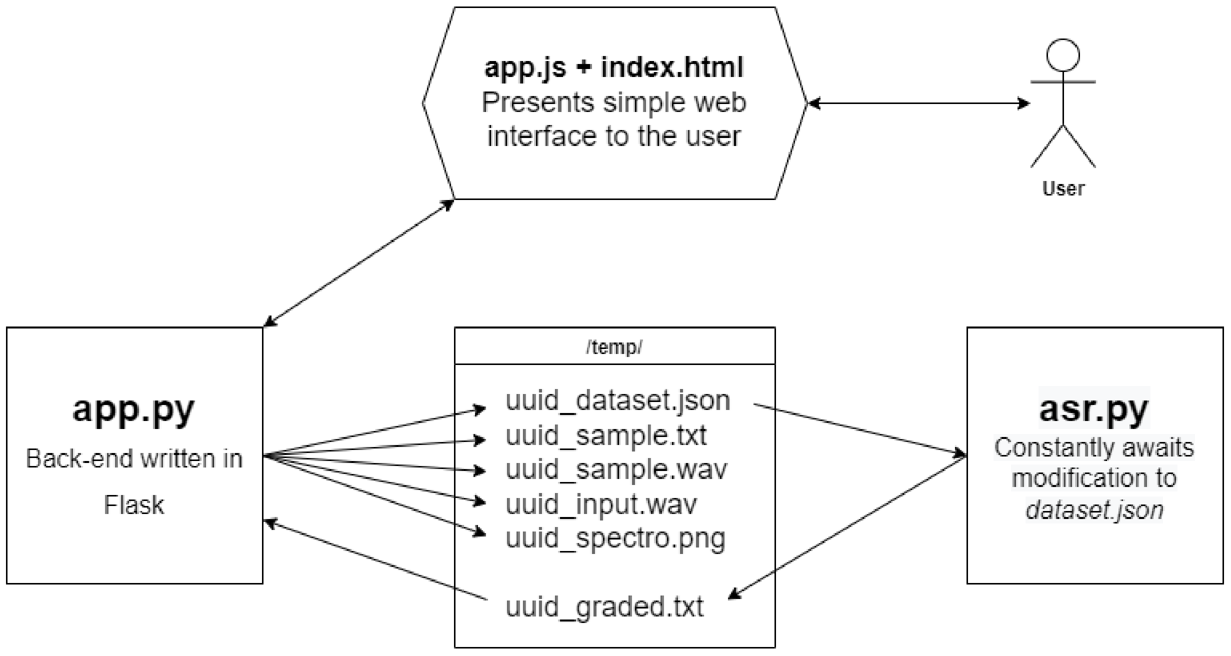
This JavaScript file powers the web page and handles sending requests to the back-end. A **MediaRecorder** object is used to record and playback the user's speech. Ajax POST requests are sent to **app.py** with the text prompt and the recorded audio. The program processes this input and feeds back the desired output: the graded speech results, the spectrogram, pronunciation key, and selected phrases. 'index.html' is an HTML layout for the web page with three buttons, an audio playback box, and two selectors for choosing a speech prompt. The user is able to listen to their own recorded audio and to re-record, as well, overwriting any previous input.

app.py

This is written with Flask, a web application framework. It receives and handles requests from **app.js**, returns the appropriate responses, and manages the folder '/temp/'. It reformats the input audio using **ffmpeg**, and turns the text prompt into a pronunciation key with **Phonemizer**. The spectrogram is generated with **Librosa**, and several small functions pull phrases for prompts from XML libraries.

asr.py

This Python file contains a watchdog loop that waits for any modification to '/temp/*uuid*_dataset.json' and upon detecting a change it feeds the fresh data set to the ASR module. The module takes the reformatted input audio and phoneme key and evaluates the user's pronunciation, calculating a scored based off the word error rate. The results are stored in '*uuid*_graded.txt'.



app.js + index.html
Presents simple web interface to the user
User
app.py
Back-end written in Flask
/temp/
uuid_dataset.json
uuid_sample.txt
uuid_sample.wav
uuid_input.wav
uuid_spectro.png
uuid_graded.txt
asr.py
Constantly awaits modification to dataset.json