VAD Handoff - Vivek Chilakamarri

See metrics.ipynb for a complete example of how to use VAD.

Voice Activity Detection Overview

We'd like to be able to record audio data from environmental sensors without infringing on people's privacy. In order to be more flexible with our deployment of sensors, we use voice activity detection to automatically remove human speech from field recordings.

Vad Models

There are several models that already exist for voice activity detection ("Ok Google", and "Alexa" only work as commands due to VAD). Three popular VAD engines exist today that we consider for deployment: Silero, WebRTC, and Cobra.

We have a working implementation for each of these VAD engines. In vad_engines, a python script for each engine has a get_voice_timestamps() function, that you can call by passing in a path to the desired audio file. These calls will return a list of predicted intervals of human speech.

Some VAD models have parameters that we can tune for different results.

- Cobra
    - Speech Probability Threshold:
        - At each frame, cobra returns the PROBABILITY of human speech. I altered the script to return the intervals based on a given threshold of confidence.
- WebRTC
    - "Aggresiveness"
        - WebRTC takes in an "aggressiveness" parameter which is a more general probability threshold. From my experimentation, it seems best to leave this set to 3 (the maximum)

Scaper

Before we deploy any VAD to the field sensors, we need to understand how they will perform in the wild. However, in order to test, we need to have examples of audio data that a field recorder might hear. Moreover, this audio data must have human speech in it, labeled by timestamp.

This data is difficult to come by, so we can artificially create testing data by hand. In order to create testing data, we use Scaper. Scraper is a python package that allows you to mix random background noise with foreground samples determined by some probability distribution. This

means that we can create unique, random datasets of environmental noise (rain, wind, traffic noise, etc.) mixed with random speech recordings.

Scraper also produces an output text file that describes where in the generated soundscape text exists. We use this information to generate ground truth labels to test our models on.

Using Scaper:

- Scaper uses distribution tuples to define params
    - https://scaper.readthedocs.io/en/latest/tutorial.html#distribution-tuples
- This means that all parameters define probability distributions for creating soundscapes, not any hard-coded value. (powerful because we can create unique soundscapes with each run)
- Scraper will search for background samples in the audio/background directory. It searches for foreground samples in audio/foreground. Each folder within audio/background and audio/foreground are considered a “label” for Scaper to choose from. Populating these folders will result in rich soundscapes to choose from.
- Example.wav in the repository has an example of a generated soundscape.

Datasets

All of the data I collected is in the data directory. There are small samples from different datasets that I used to experiment with Scaper. In order to fully utilize the soundscape generation, we need to continue finding good background/foreground samples.

*There are still lots of good samples in the datasets below, but it may be good to find other datasets as well.

Raw Speech Datasets

RealVAD:
https://zenodo.org/record/3928151#.Yh64dBPMK3I

- RealVAD dataset is constructed from a YouTube video composed of a panel discussion lasting approx. 83 minutes. The audio is available from a single channel. There is one static camera capturing all panelists, the moderator and audience

AVA - Speech:
https://arxiv.org/pdf/1808.00606.pdf

- The choice of the (videos in the) dataset, for which we will provide YouTube video IDs (15 minute clips from 185 movies for 45 hours total) along with manually annotated dense labels indicating the presence or absence of speech activity.
- The labeling explicitly annotates segments containing active speech as one of three classes—clean speech, speech co-occurring with music or speech co-occurring with noise —and the ones that don’t contain speech

OpenSLR:
http://www.openslr.org/resources.php

- Collection of LOTS of audio datasets
- Notables
    - RWCP Sound Scene Database
    - https://www.openslr.org/51/ TED-LIUM Release 3
    - http://www.openslr.org/12/ LibriSpeech ASR Corpus

Mozilla Common Voice
https://commonvoice.mozilla.org/en/datasets

- More than 9000 hrs of speech data in over 60 different languages

Environmental Noise Datasets

ESC-50: Dataset for Environmental Sound Classification
https://github.com/karolpiczak/ESC-50

- The ESC-50 dataset is a labeled collection of 2000 environmental audio recordings suitable for benchmarking methods of environmental sound classification. The dataset consists of 5-second-long recordings organized into 50 semantical classes (with 40 examples per class) loosely arranged into 5 major categories:
    - Animals
    - Natural soundscapes & water sounds
    - Human, non-speech sounds
    - interior/domestic sounds
    - exterior/urban noises

Metrics

In metrics.ipynb, there is an example of a complete VAD workflow.

Setup

The first cell has code to install and import the necessary libraries for our VAD engines. If only a single VAD engine is being deployed to a sensor, not all of these installations/imports are required.

Below, there is a call to generate_soundscapes(), from the synthesize.py file. You can change any of these parameters to affect the probability distributions of soundscape generation.

Also, within soundscapes.py, you should adjust the below code to select from more specific background/foreground labels. (Replace `label=('choose', [])` with the desired distribution tuple

-
```
        # add background
        sc.add_background(label=('choose', []),
                          source_file=('choose', []),
                          source_time=('const', 0))

        # add random number of foreground events
        n_events = np.random.randint(min_events, max_events+1)
        for _ in range(n_events):
            sc.add_event(label=('choose', []),
                         source_file=('choose', []),
```

Setting up metrics:

```
with wave.open('./audio/soundscapes/soundscape_unimodal0.wav', 'rb') as wf:
        nchannels, sampwidth, framerate, nframes, comptype, compname = wf.getparams()

ground_truth_sum = np.zeros((n_soundscapes, nframes))
cobra_preds_sum = np.zeros((n_soundscapes, nframes))
silero_preds_sum = np.zeros((n_soundscapes, nframes))
webrtc_preds_sum = np.zeros((n_soundscapes, nframes))
```

These matrices are created to store the predicted speech activity for each soundscape. After all soundscapes have been predicted for, we can use this aggregated matrix to find the average performance across all samples.

Predicting Speech

The cell tagged with:

#ACTUALLY DO THE VAD PREDICTION AND GROUND TRUTH CALCULATIONS HERE

Shows how to actually predict speech activity on a sample. First, we need to convert the desired audio to a PCM_16k file type, in order to maintain compliance with all three VAD engines. Then, we use the associated ".txt" file to create the ground truth labels for that audio.

For example:

If a generated soundscape is 5 seconds long and has speech from seconds 1-2, the resulting label will look something like this.
An array (of length num_frames):
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
We find the predicted speech timestamps for each VAD engine by a call to their respective get_voice_timestamps() functions, and use the below logic to recreate a similar array for speech predictions:

```
for interval in cobra_voice_timestamps:
    start = interval['start']
    end = interval['end']
    cobra_preds[start:end] = 1
```

Metrics

Finally, after we have created all ground truth labels and respective speech predictions for all VAD engines, we can calculate the precision, recall, and f-score for that model.

See below for how to interpret these results:
https://towardsdatascience.com/accuracy-recall-precision-f-score-specificity-which-to-optimize-on-867d3f11124

For the most part, when the considered VAD models predict speech at a given timestamp, they tend to be correct. This means that looking only at precision (or the measure of how many predicted positive samples are actually true positive samples) is not comprehensive. For example, if speech is present from seconds 1-5, and our model predicts speech from seconds 3-5, then the precision of that model is 100%. This is because every instance of a model's predicted speech is correct.

In our case, recall is a very important metric. Recall is the measure of how many True Positive samples our model did not predict. For example, if speech is present from seconds 1-5, and our model predicts speech from seconds 3-5, recall captures the first two seconds of error.

Next steps

In order to fully validate models, the biggest next step is to populate the audio directory with many many background and foreground samples. The code to connect all pieces of the pipeline exists, we just need to create a large enough dataset to find meaningful results.

Also, there can be experimentation with different kinds of soundscapes (e.g. heavy rain, heavy traffic noise, sparse noise, etc.) to see which VAD model performs best under that use case.

After a model is deemed ready for deployment, you can follow the logic in metrics.ipynb to implement the desired model. Before logging audio data in the field, we simply remove the timestamps that the model predicts is human speech.

Depending on how accurate we need the models to be, we can adjust this speech removal to have a large/small window around predictions.

- For example, if a model predicts speech from 3-5 seconds, we may want to remove 2-6 seconds as a safety.
- Conversely, if models are over scrutinizing audio data, we can reduce the window to 3.5-4.5 seconds

I hope this is a good starting off point for Voice Activity Detection! You can email me at chilkvs@gmail.com for any questions!