Purpose

The purpose of this project is to create an algorithm to approximate the time stamps of the pauses between phrases or sentences given some text and an audio recording of a person reading that text.

Threshold Algorithms

Deciding on a sound amplitude threshold to differentiate between speech and background noise is a crucial sub-problem in finding pauses and placing phrase breaks. You'll find several attempts at an effective and generalizable threshold algorithm in ./src/utils/findPauses.ts.

Every audio file will have a different threshold, so we really do need an algorithm that can dynamically find a threshold for any given example.

ClusterWithGMM

Cursory reading on others who have faced this problem before led to the idea that taking the logarithm of the amplitude of every audio sample in a sound file yields a bimodal distribution, with one peak representing speech and another representing background noise.

Attempting to take advantage of that insight, one of my first attempts at finding a threshold was to use a gaussian mixture model to find separate gaussian distributions for the amplitudes of audio samples representing speech, background noise, and silence.

Running the algorithm for a very small number of iterations usually yields promising results, but on most data I found that it would converge to a single gaussian distribution in less than 10 iterations. It's possible that I made some error in my implementation that results in a faulty convergence. It's also possible that the data I tested on didn't have distinct enough peaks for a correctly implemented GMM to differentiate them strongly.

ClusterWithKMeans

Using a simpler clustering method, I was usually able to find a pair of centers that roughly corresponded to the center of each bimodal distribution, as visualized by the “print histogram to prove bimodalness” function.

The “clusterWithKMeansAdjustToPeak” function looks at the histogram and moves each cluster’s centers to the nearest local maximum within a given percentage of the total distribution’s width.

Once those centers are found, the threshold is set at a point somewhere between them. in the top-level files in ./src/entryPoints, the parameter called FRACTION_OF_SPEECH specifies where the threshold should be set as a fraction of the distance from the background-noise peak to the speech peak. For example, FOS=0 sets the threshold at the background peak and FOS=1 sets the threshold at the speech peak. Straightforwardly enough, I generally got good results with FOS=0.5.

K refers to the number of clusters to sort the data into. I generally got the best results with K=3, then treating the two clusters with centers at the greatest amplitude as representing background noise and speech. It typically converged well within

ClusterByHistogramLocalMaxima

This threshold algorithm was a natural extension of clustering with K-means and then adjusting to the nearest local maxima. It iterates through the histogram of binned $\log_{10}$(amplitude) and finds all local maxima.

In order to really perform, it would need the addition of an algorithm to decide which local maxima represent the peaks for speech and background noise. This should be straightforwardly possible by choosing the tallest peaks that have the deepest valley between them.

Length-Based Algorithms

textLength

textLength is the foundation of several other algorithms. It doesn’t take the audio file into account except for reading its duration.

This algorithm assumes the fraction of the whole text's characters that fall into a sentence is the same as the fraction of the whole recording's duration corresponding to that sentence.

For example, if the whole text has 1000 characters and the audio recording is 20 seconds long, this algorithm would allocate 5 seconds to a 250 character sentence.

pauseAwareTextLength

In reality, textLength's assumption doesn't always hold true. In particular, it fails badly when the pauses in between phrases are significantly different in length. The parts of the audio recording that correspond to speech follow that assumption about proportionality, but the written text says nothing about how long each pause between sentences must be.

The goal of pauseAwareTextLength was to place the phrase breaks based on an assumed proportionality between its fraction of the full text and the fraction of the audio file's duration where the sound's amplitude is above a given threshold, as determined by a given threshold algorithm.

My implementation appeared to work well on most short examples, but on longer examples, the sentence breaks tended to cluster close to the beginning of the recording. I did not have time to investigate the root cause of this bug.

Pause-Based Algorithms

QuietestNearby

QuietestNearby takes the output of a length-based algorithm and looks for promising pause locations near each predicted split. It looks at a given fraction (BASELINE_SLOP_PERCENT) of the length of the segments to either side of it, divides them into sub-intervals of a given duration (BASELINE_INTERVAL_DURATION) and moves each split to the sub-interval with the smallest maximum amplitude.

PauseFinder

This algorithm finds a set of pauses by considering every audio sample with an amplitude less than a given threshold. It makes three passes on the data, first joining sets of

pauses separated by an amount of time less than MIN_GAP_PRE_DROP into one pause. Next, it deletes all pauses shorter than MIN_PAUSE_DURATION. Finally, it joins sets of pauses separated by gaps shorter than MIN_GAP_POST_DROP.

It returns the set of pauses that remain after all dropping and joining passes are complete.

DecodeWithPauses

DecodeWithPauses creates a mapping of every combination of sentence break in the text (as assigned by a length-based algorithm) and pause (as assigned by the pauseFinder algorithm) to a confidence value representing how likely they are to represent the same pause. Each sentence break is then assigned to the audio pause for which it has the highest confidence value.

To avoid assigning multiple sentence breaks to the same pause, we begin with the sentence break/pause combination with the overall highest confidence, and remove the listings for all pairings that use a given pause when any sentence is assigned to that pause.

The confidence values are determined by a function of the form:

$$(PAUSE\_WIDTH\_FACTOR)(pause\ width) + (DISTANCE\_FACTOR)(distance\ from\ pause\ to)^{DISTANCE\_POWER}$$