Fingerprinting for Real-Time Audio Stream Alignment

Jianchao Geng, Min Soo Kim
Prof. Jun Yang, Ph.D.

ABSTRACT

We consider the problem of the aligning two audio stream in real time, which arises in our project aimed at building a system for "pop-up" fact-checking. A backend server processes live streams of speeches and debates, identify check-worthy claims, and finds matching fact-checks in real time. By listening to the microphone on a user device, a frontend app detects which stream and where within this stream the user is currently listening to, and queries the backend server to display the matching fact-checks accordingly. The user's consumption of the audio stream may not occur at the actual time of the event (even if the user is supposedly listening to a live stream), and the user may pause/forward/rewind the stream. Furthermore, the audio signal received by the frontend app may be subject to considerable background noise. To address these challenges, we apply landmark-based audio fingerprinting techniques to develop a solution for the real-time audio stream alignment problem. This poster discusses the background, our progress, and future goals as Summer 2019.

BACKGROUND

An algorithm had to be created such that it can recognize audio samples that are subject to heavy background noise and lossy audio compression. Such an algorithm had to perform relatively quickly and be able to identify the short audio query within a large database of 1-to-2 hour long audio files (mainly political speeches). Not only does this algorithm have to be done quickly, but it also must return few false positives while also having a high-enough accuracy rate.

Dan Ellis’ Landmark-based Audio Fingerprinting algorithm, which is an open-source codebase, was found to be robust and accurate enough for our purposes.

GENERAL IMPLEMENTATION

The fingerprints hashes will be calculated both on the front-end server and on the back-end server. This is done so that, instead of giant audio files being communicated between the front-end and the backend, only the hashes are relayed. This, in turn, will not only decrease bandwidth but also decrease the time it takes for the fact-checks to be verified and sent to the user.

PORTING TO JAVASCRIPT

The audio fingerprinting library, by Dan Ellis, is written entirely in Python. While this works for our backend server, we would need an implementation in a different language for our frontend app. We chose to port the library to JavaScript because its ubiquity in client apps. To begin, we developed a browser app that listens to client’s microphone and computes fingerprints in a streaming fashion to be matched and aligned with server-computed fingerprints.

PORTING TO JAVASCRIPT

Calculating hashes:

Like mentioned in the abstract, the audio fingerprinting algorithm that we have decided to use is the landmark-based audio fingerprinting technique. Here are the basic, general outline to our algorithm:

1. Collect the amplitudes of a given audio sample, which have a sampling rate of 11,025 amplitudes per second.
2. Run the amplitudes through a reshaping, which will create overlapping windows of certain length (hop length). This is to ensure increased audio resolution and sensitivity.
3. The reshaped amplitudes are run through a Fourier Transformation

Matching Hashes:

To match a sample query to a speech in the database, we first compute hashes for both. For each matching hash found both in the database and the query, the offsets from the hashes at the timestamps to both the beginning of the query and database files are associated into time pairs. The time pairs are distributed into bins according to the track ID associated with the speech.

Within each bin the set of time pairs represent a scatterplot. If there is a match, there will be a significant cluster of points forming a diagonal line within the scatterplot. This can be detected by creating a histogram, which inputs the difference in offset times between the query and the speech as the X-axis and the number of such offsets as the Y-axis. If there is a significant diagonal line, then there will be a relatively tall bin and if that histogram bin is taller than a certain threshold or frequency, then the time pairs’ bin, which represents a track ID or speech, is considered a match.

Audio Fingerprinting

Record: App, Source: Speech
Format: start recording to see sample rate
Recordings

Only put in one audio file at a time
Choose File: No file chosen

(Main) Difficulty in Implementation:

• The JavaScript library is not well-suited for mathematical and scientific use as the Python library is. One very important distinction is the NumPy library for Python, which has an incredible variety of mathematical and statistical functions. JavaScript, unfortunately does not have a native version of NumPy. Although there is a similarly-named NumJs library that has been created, it does not have nearly all of the functions that are necessary for audio-fingerprinting.

REFERENCES


ACKNOWLEDGEMENTS

This work was supported by the Duke Summer Projects in Computer Science Program. The authors thank Professor Dan Ellis of Columbia for allowing us to use his landmark-based audio fingerprinting code. With help from East Chapel Hill High School Students, Jonathan Xu and Andrew Mu, the audio fingerprinting project progressed very well. Additionally, with the incredible help from Stavros Sintos and Jun Yang, the authors were able to understand the audio fingerprinting code.

Student Showcase: DATA+, CODE+, Summer Projects in Computer Science
August 2, 2019; Duke University, Gross Hall