Lab 1 - "Playing with audio slices"

Sunday, July 20, 2008
11:37 PM

PURPOSE

This lab will introduce you to the practice of analyzing, segmenting, feature extracting, and applying basic classifications to audio files. Our future labs will build upon this essential work - but will use more sophisticated training sets, features, and classifiers.

We'll first need to setup some additional Matlab folders, toolboxes, and scripts that we'll use later.

SETUP

1. Download the **MIR Toolbox** to your local Matlab folder and add to your path (including all subfolders).
2. Download the **MIR Toolbox user guide** to your local Matlab folder.
3. Download the **audio.zip** files, which contain many of the audio files referenced in our labs.
4. Download the folder **Lab 1.zip** which contains the file(s): **ccrma_onset_detector.m**, and **zcr.m**.

SECTION 1

Play around with the different features for known frames… and see if you can build a heuristic understanding of what they are doing.

1. Load the audio file SimpleLoop.wav into Matlab, storing it in the variable ***x*** and sampling rate in ***fs***.

2. Run an onset detector to determine the approximate onsets in the audio file.
```
[onsets, numonsets] = ccrma_onset_detector(x,fs)
```

3. The onset values are displayed in samples. How do you display them in seconds?

One of Matlab's greatest features is its rich and easy visualization functions. Visualizing your data at every possible step in the algorithm development process not only builds a practical understanding of the variables, parameters and results, but it greatly aids debugging.

4. Plot the audio file in a figure window.

5. Now, add a marker showing the positive of each onset on top of the waveforms.
```
plot(x); hold on; plot(onsets,0.2,'rx')
```

6. Adding text markers to your plots can further aid in debugging or visualizing problems. Label each onset with it's respective onset number with the following simple loop:
```
for i=1:numonsets
   text(onsets(i),0.2,num2str(i));  % num2st converts an number to a string for display purposes
end
```

   Labeling the data is crucial. Add a title and axis to the figures. (**ylabel**, **xlabel**, **title**.)

7. Now that we can view the various onsets, try out the onset detector on a variety of other examples. Continue to load the various audio files and run the onset detector - does it seem like it works well?

Segmenting audio in Frames

As we learned in lecture, it's common to chop up the audio into fixed-frames. These frames are then further analyzed, processed, or feature extracted. Now that our onset detector is working, let's carve up the audio into frames 100 ms frames starting at each onset.

8. Create a loop which carves up the audio in fixed-size frames (100ms), starting at the onsets.
9. Inside of your loop, plot each frame, and play the audio for each frame. (The **pause** command is quite useful.)

Feature extract your frames

10. Create a loop which extracts the Zero Crossing Rate ***for each frame***, and stores it in an array.
    The command **[z] = zcr(x)** returns the number of zero crossings for a vector x.

    You should now have a feature array of 5 x 1 - which is the 5 frames (one at each detected onset) and 1 feature (zcr) for each frame.

11. Open up the file zcr.m in the editor and look at how it works. This is one of the simplest of feature extractors.

Sort the audio file by it's feature array.

Let's test out how well our features characterize the underlying audio signal.

12. First, sort the feature vector from low to highest.
    [y,index] = sort(features);

    If we play the audio that corresponds with these sorted frames, what do you think it will sound like? (e.g., same order as the loop, reverse order of the loop, snares followed by kicks, quiet notes followed by loud notes, or ???) Pause and think about this.

13. Now, create a loop which plays these sorted audio frames, from lowest to highest. (The **pause** command will be quite useful here, too.) How does it sound? Does it sort them how you expect them to be sorted?

You'll notice how trivial this drum loop is - but this is a good tip - always use familiar and predictable audio files when you're developing your algorithms! No sense in testing out overly complex examples at this stage.

14. Now, repeat the entire above process (onset detector, feature extraction, sorting and reordering) with the audio files "Conga Groove 01.wav", "125BOUNC-mono.WAV" , or "58BPM.WAV".

15. Do the slices seem to be sorted in an order that you matches your perceptual expectations? After trying out these audio files, try out this process with a few other audio files at random…

For your listening pleasure, I sorted all of Stairway to Heaven using a few additional features - it's located in the audio folder : "Stairway To HeavenMEAPED.mp3".

SECTION 2: More Cowbell!

Heuristics are experimentally derived rules; they are commonly used in a variety of applications. In this case, let's create some heuristics to detect when a snare drum is played. You will receive less direction in how to implement your techniques in this and further sections - if you need help, or don't know the proper Matlab commands for what you want to do, don't be afraid to ask.

1. Load the audio file "SimpleLoop.wav".
2. Looking at the zero crossing rate of this file, can you tell which frames contain snare drums? If so, which ones?

Sonifying events by inserting a special tone or marker into the audio stream is another great way of developing and debugging your algorithms. By listening to the sonified and marked-up audio files, you will be able to easily hear if you are getting good results. Looking at an array of floats is one thing - but listening to it is another animal entirely.

3. Create a loop which implements your heuristics by playing a cowbell at the same time as the snare drum of a given loop. The audio file "cowbell.wav" can be found in the audio files folder.

    Play the mixed audio file with the cowbell inserted on top of the snare drum. (Rock on!)

4. Now try on 125BOUNC-mono.WAV. Try some other files, too. What do you notice? Does your snare drum heuristic remain accurate the same for all of the other drum loop files?

    Notice what happens when the snares is hit harder, is pitched differently, has variation in its attack tone, has simultaneous hits with other instruments, or unseen events occur!

BONUS [only if you finish the other sections] : My First Transcription

1. Using the audio file "SimpleLoop.wav", now create a simple transcription heuristic for determining kick and snare.

2. Re-synthesize your transcription using some samples. (I placed a "kick.wav" and "snare.wav" in the audio files folder for you.)

3. Now, try out your transcription with 125BOUNC-mono.WAV. Now try it on some drum loop examples. How well does it hold up?

   Hmm… looks like we'll need some additional intelligence to classify the full range of kicks and snares out there.

NEED HELP?

Loading audio into Matlab
To load from the command line, you can load use the **wavread** command, such as [x,fs]=wavread('foo.wav').

Listening to an audio file in Matlab
sound(x,fs)
To stop listening to an audio file, press Control-C.

Audio snippets less than ~8000 samples will not play out Matlab.

Tricks of the trade
Select code in Matlab editor and then press F9. This will execute the currently selected code.
To run a Matlab "cell" (multiline block of code), press Control-Enter with the text cursor in the current cell.

The **clear** command re-initializes a variable. To avoid confusion, you mind find it helpful to clear arrays and structures at the beginning of your scripts.

Common Errors
>??? Index exceeds matrix dimensions.
Are you trying to access, display, plot, or play past the end of the file / frame?
For example, if an audio file is 10,000 samples long, make sure that the index is not greater than this maximum value. If the value is > than the length of your file, use an **if** statement to catch the problem.

Why are the Paste / Save keys different? Why does Paste default to Control-Y?
On Linux, Matlab defaults to using Emacs key bindings. To fix this, go:
    File menu > Preferences > Keyboard
    Switch the Editor/Debugger key bindings to "Windows"

Cowbell
If you don't understand the joke behind "more cowbell", check out :
http://webfeedcentral.com/2005/01/21/more-cowbell-video/