ELE 301, Fall 2011
Laboratory No. 5

It will behoove you to read this week's and next week's lab completely before you begin coding.

1 Background

In this lab we will begin to code a Shazam-like program to identify a short clip of music using a database of music. The basic procedure is:

1. Construct a database of features for each full-length song;
2. When a clip (hopefully part of one of the songs in the database) is to be identified, calculate the corresponding features of the clip;
3. Search the database for a match with the features of the clip.

Like Shazam, the features for each song (and clip) will be pairs of proximate peaks in the spectrogram of the song or clip. We start by finding the peaks in the (log) spectrogram; plotting the locations of these peaks gives a "constellation map". Each peak has a time-frequency location (t, f) and a magnitude A. We then form pairs of peaks that are within a pre-specified time and frequency distance of each other and record the details of these pairs in the database. For example, if we obtained a pair (f_1, t_1) and (f_2, t_2), we might record $(f_1, f_2, t_1, t_2 - t_1, A_1, A_2, \text{songid})$. We don't want to record all possible pairs of peaks; there are far too many. We will explain shortly how to select the pairs.

Instead of recording $(f_1, f_2, t_1, t_2 - t_1, A_1, A_2, \text{songid})$, we will discard the amplitudes and just record $(f_1, f_2, t_1, t_2 - t_1, \text{songid})$. The reason for this is explained in the paper: briefly, the amplitude of a peak may not be robust to gain changes across frequency.

Each song is summarized ("fingerprinted") by a (big) table of its extracted features, e.g.

$$\left|\begin{array}{c|c|c|c|c} f_1 & f_2 & t_1 & t_2 - t_1 & \text{songid} \\ & & & \vdots & \\ f_j & f_k & t_j & t_k - t_j & \text{songid} \\ & & & \vdots & \\ f_m & f_n & t_m & t_n - t_m & \text{songid} \end{array}\right|$$

When we are given a clip to identify, we do the same feature extraction for the clip. The result is a small table of clip features:

$$\left|\begin{array}{c|c|c|c} f_1 & f_2 & t_1 & t_2 - t_1 \\ & & & \vdots \\ f_j & f_k & t_j & t_k - t_j \\ & & & \vdots \\ f_m & f_n & t_m & t_n - t_m \end{array}\right|$$

We do not know the start time of the clip within its corresponding song. The times t_j that appear in the clip table are relative to the start of the clip. The clip itself starts at some unknown offset t_0 from the beginning of the music. Finding a match to the clip in the data base is a matter of matching the constellation map of the clip to the constellation maps of the songs by effectively sliding the former over the latter until a position is found in which a significant number of points match.

Using pairs of peaks as features gives us three quantities that we expect to be independent of the unknown offset time: $(f_1, f_2, t_2 - t_1)$. The song with the most triples $(f_1, f_2, t_2 - t_1)$ in common with the clip table is likely to be the source of the clip.

2 Lab Procedure - MyShazam Part I

This week you will write a script `make_table` that will take a song and construct a database of features for that song. The script will do the following:

1. Read in the song using `mp3read`.
2. Average the two channels, subtract the mean, and downsample.
3. Take the (log magnitude) spectrogram of the song using `spectrogram`.
4. Find the local peaks of the spectrogram by using `circshift` in a loop.
5. Adaptively (not manually) threshold the result of step 5 to end up with `peak_rate` peaks/sec.
6. For each peak, find `fanout` pairs in the target window and add them to the table.
7. Plot the constellation map and draw lines connecting the pairs.

These steps are now explained in detail.

2.1 Reading an mp3 file: mp3read.m

In order to read an .mp3 file into MATLAB we will use the function `mp3read.m` from Professor Dan Ellis, Columbia University. You need to put the files mp3read.m, mpg123.exe and mp3info.exe into the same folder on your computer. To read an .mp3 file into MATLAB, use the following command:

```
[y,fs]=mp3read('file_name');
```

This opens the file "file_name.mp3", decodes the file and returns the decoded signal in the vector `y`. The sampling rate is stored in `fs`.

2.2 Preprocessing

Read in the .mp3 file, viva.mp3. The signal is actually from two channels, but for our purposes we can combine them by taking the mean of the corresponding samples of the two channels. Do this via the command `y=mean(y,2)` to average along the second dimension. Now subtract off the mean of `y` to eliminate the potential horde of peaks at frequency $f = 0$.

Since `fs` is 44100 Hz, we have a lot more data than we need. Resample the signal at 8000 Hz using the command `resample`, as follows:

```
y=resample(y,new_smpl_rate,old_smpl_rate); % rates must each be integers.
```

This command performs an interpolation of the signal at the new sampling points and returns the result.

2.3 Spectrogram

Now we want to construct the spectrogram of this signal. To do this, use the MATLAB command `spectrogram`. It basically does what the spectrogram program you wrote in the previous lab does (with the extra addition of windowing, which you don't need to worry about). Call it as follows:

```
[S,F,T]=spectrogram(y,window,noverlap,nfft,fs);
```

`window` is an integer that indicates the length of the chunks you want to take the fft of. `noverlap` is the number of samples you would like to have as overlap between adjacent chunks. `nfft` is the length of the fft you would like to take, which in our case can be the same as `window`. Finally, `fs` is the sampling rate of the signal, in our case 8000 Hz. The function returns the spectrogram in the matrix `S` with just the positive frequencies. The

frequency vector for the vertical axis is returned in `F` and the time vector for the horizontal axis is returned in `T`.

Compute the spectrogram with window length 64 ms and an overlap of 32 ms. Be sure to specify both these parameters as an integral number of samples. Plot the magnitude of the spectrogram of the song with axes appropriately labeled.

It is common to study the log of the magnitude of the spectrogram. Why might this be a good idea?

Plot the log of the magnitude of the spectrogram with axes appropriately labeled. We will use the log magnitude spectrogram.

2.4 Spectrogram Local Peaks

Next, we find the local peaks of the spectrogram. A local peak has log magnitude greater than that of its neighbors. One way to find the local peaks is to iterate through each point in the spectrogram and compare the magnitude to the magnitude of each of the points in the surrounding `gs x gs` grid. This can be done for all point at the same time by using the command `circshift`. For example, if `S` denotes the log magnitude spectrogam, then the code

```
CS=circshift(S,[0,-1]);
P=((S-CS)>0);
```

returns a boolean matrix `P` with entries 1 for the positions in `S` that are greater than their neighbor immediately to the left (see `help circshift` for details). If you put this structure in a loop, you can select the points in `S` that are greater than all of their neighbors in the `gs x gs` grid, i.e. the local peaks. The location of these peaks are stored in a boolean matrix `P` of the same size as `S`. Plotting `P` as an image will display the constellation map. Change the colormap with the command:

```
colormap (1-gray);
```

which will display the entries where there are peaks as black pixels and the rest of the matrix as white pixels.

Try several values for `gs` and plot the constellation map. Note the effect of changing `gs`. Compute the constellation map for `gs=9`, i.e. 4 points in each direction. Calculate how many peaks there are and record your answer below. How many peaks are there per second on average?

2.5 Thresholding

We want to use only the larger peaks. Why? (Hint: think about the quality of the clip we would like to identify)?

To select the larger peaks and also to control the average rate of peak section (peaks per second), we have to do some sort of selection operation on the peaks. The simplest thing to do would be to apply a fixed threshold to the detected peaks and keep only those above the threshold. The threshold could be selected to yield (approximately) the desired number of peaks.

Assume that we want approximately 30 peaks per second. Find a threshold which yields that rate of peaks. Record the threshold value below along with the number of peaks kept. Write a routine which will find some optimal threshold automatically instead of manually doing it.

Again, display the constellation. Comment on the distribution of peaks. Is it uniform? Are they closely packed? If so, is this a good thing?

Alternatively, we can use an adaptive threshold, where we advance along the matrix in 1 second chunks of columns and adaptively threshold each of these chunks to approximately get our 30 peaks per second. Try this if time permits.

If we were concerned about having peaks that were too close together, how could we threshold so that we guarantee some separation between peaks? Concretely, outline a basic algorithm which would accomplish this (you don't have to code this).

2.6 Constructing the Table

As outlined in the preliminary material, we want to select pairs of peaks and record the frequency of each peak, the time of the first peak and the time difference between the two peaks.

A peak-pair must satisfy certain constraints: the second peak must fall within a given distance from the frequency of the first peak and the second peak must occur within a certain time interval after the first peak. We will also limit the number of pairs allowed to form from a given peak, say to 3 (this is called the fan-out). So a peak located at (t_1, f_1), can only be paired with peaks which have $t_1+\Delta_t^l < t_2 \leq t_1+\Delta_t^u$ and $f_1-\Delta_f \leq f_2 \leq f_1+\Delta_f$ for some Δ_t^l, Δ_t^u and Δ_f. See Figure 2.7 for an example, where the box encloses the area in which we are looking for peaks.

The command `find` is very useful.

Our objective is to select appropriate parameter values for Δ_t^l, Δ_t^u, Δf and `fanout`, and then record the 4-tuples $(f_1, f_2, t_1, t_2 - t_1)$ in a matrix. One can do this after selecting the larger peaks as in the previous section. However, another interesting possibility is to combine the selection of a subset of peaks with the selection of peak-pairs.

After you have produced your selection of pairs, display the constellation map and connect the pairs listed in the table with line segments (use the command `line`).

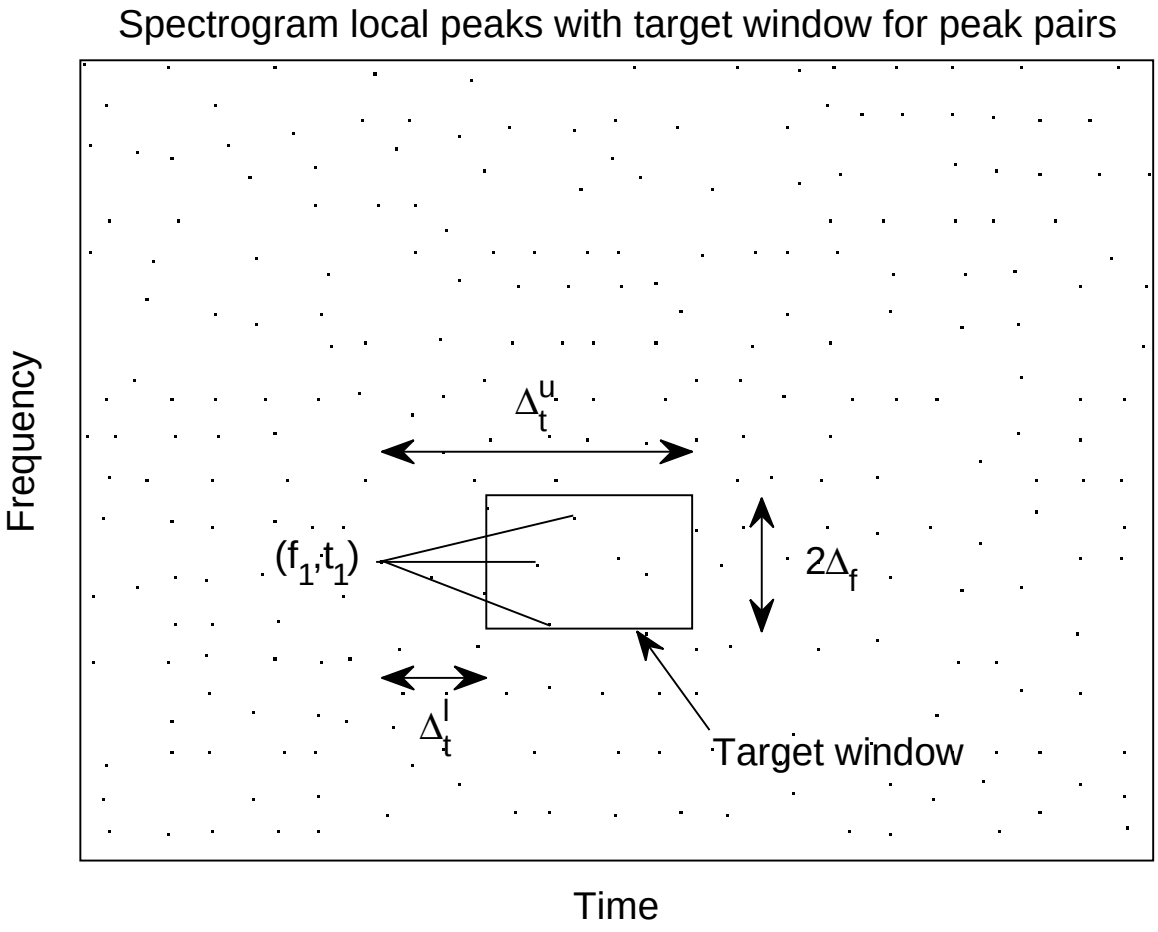
Experiment with changing the fan-out for each peak and with changing Δ_t^l, Δ_t^u and Δf. Also try different lengths for `window` in the spectrogram. Note any significant findings.

2.7 Final Function

Now, combine everything into a function `make_table` which puts all of these steps together. It will take as input the song signal and it will return an `npairs` $\times$ 4 matrix which contains in each row the 4-tuple corresponding to a peak pair:

$$\left|\begin{array}{c|c|c|c} f_1 & f_2 & t_1 & t_2 - t_1 \\ & & & \vdots \\ f_j & f_k & t_j & t_k - t_j \\ & & & \vdots \\ f_m & f_n & t_m & t_n - t_m \end{array}\right|$$

When you are constructing this table, use the indices of the spectrogram as the values for f and t instead of using the corresponding Hz and seconds values.


Spectrogram local peaks with target window for peak pairs
Frequency
Time
Δ_t^u
(f_1,t_1)
$2\Delta_f$
Δ_t^l
Target window