Lab 5: Shazam

ECE 2200 Fall 2014
Professor Lang Tong

November 17–20, 2014

1 Introduction

In this lab, you will combine what you learnt this semester. We will begin to code a Shazam-like program to identify short clip of music using a database of music. As shown in Fig. 1, 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, calculated 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 $(f_1, f_2, t_1, t_2 - t_1, \text{songid})$. We

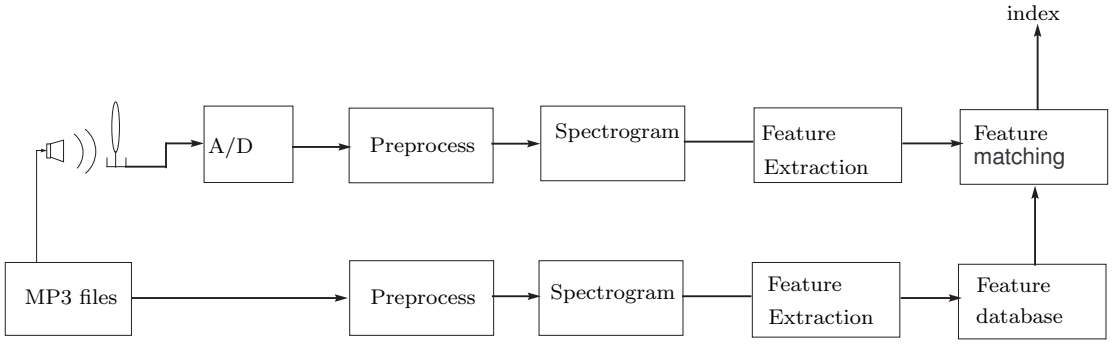

Figure 1: Shazam Diagram

will discard the amplitudes because the amplitude of a peak may not be robust. Detail can be found in the attached paper.

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

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

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:

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

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 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 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.

As shown in Fig. 1, in the pre-lab, we will learn how to preprocess the mp3 files (sec. 2.1), how to draw the spectrogram (sec. 2.2) and how to extract the features (sec. 2.3). In the in-lab, we will learn to build the database (sec. 4.2) and how to find the matching (sec. 4.3).

2 Pre-lab.

In the pre-lab, we will learn how to preprocess the mp3 files (sec. 2.1), how to draw the spectrogram (sec. 2.2) and how to extract the features (sec. 2.3).

Download the zip file and unzip it. In the folder named "songs", there are 52 .mat files of songs you uploaded. And in the folder "sample" there is one unknown clip. For pre-lab, we will use only one song in the folder "songs", ajt222_music.

For the pre-lab we will write a function `make_table`. A suggested function header could be

```
function table=make_table(songName,gs,deltaTU,deltaTL,deltaF)
```

where `songName` is the song name and others are parameters that will be specified. Given the name of the song, ajt222_music for example, `make_table` will take the name as input and construct a table of features for that song. The function will do the following: (details will be discussed in the following sub-section)

1. Load in the ajt222_music.mat file in the folder "database" and resample it if necessary.
2. Take the (log magnitude) spectrogram of the song using `spectrogram`.
3. Find the local peaks of the spectrogram.
4. Adaptively (not manually) threshold the the result of step 3 to end up with peak_rate peaks/sec, for example 30 peaks per second.
5. For each peak, find "fan-out" pairs in the "target window" and add then to the table.
6. Save the table into table.mat

   where gs is the target window size and deltaTU, deltaTL and deltaF describe the location of the window.

2.1 Preprocessing

Shazam read in the .mp3 file and convert it into .mat file. The signal contains two channels, but for our purposes it suffices to take the mean of the corresponding samples of the two channels. In the songs given in the folder named "songs", these steps are already done for each song and you can use those .mat files directly.

Since F_s is 44100Hz, we have a lot more data than we need. Resample the signal at 8000Hz using the command `y=resample(y,newFs,oldFs)`. This command performs an interpolation of the signal at the new sampling points and returns the result. And we will work on the resampled signal from now on.

What to do: Given a song name, ajt222_music for example, resample it at 8000 Hz and store the signal into y.

Prob. 3.1: What is the length after you resample the file ajt222_music.mat in the database?

2.2 Spectrogram.

Now we construct the spectrogram of the first song in the database, ajt222_music.mat. Call the spectrogram command as follows:

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

where:
`y` is the resampled signals.
`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, i.e. the resolution of frequencies, which in our case can be the same as window.
`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. Note that the number of samples in a window is simply the window length multiplied by our sampling rate. Be sure to specify both these parameters as an integral number of samples. Take the absolute value of S and take the log 10. We shall use the log magnitude spectrogram.

What to do: Given the resampled signal `y`, get the log 10 spectrogram `S` with window size 64ms and overlap 32ms.

Prob. 3.2: Plot the magnitude of the spectrogram of the song with axes appropriately labeled. Also plot the log of the magnitude of the spectrogram with axes appropriately labeled. Use command `imagesc` and we will get something similar to lab 2.

2.3 Feature Extraction.

2.3.1 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 $g_s \times g_s$ grid. This can be done for all points at the same tie by using the command `circshift`. For example, if S denotes the log magnitude spectrogram, 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 right (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 $g_s \times g_s$ grid, i.e. the local peaks. The location of these peaks are stored in a boolean matrix P of the same size as S. Next example may be helpful:

```
%%Code to find the peaks in matrix A with gs=5. Return the boolean matrix P.
clear all;
clc;
close all;

gs=5;%Consider the neighborhood of size 5x5
A=normrnd(0,1,5,5);%Create a random 5x5 matrix A
array=-floor(gs/2):floor(gs/2);
P=ones(size(A));
```

```
for i=1:gs
    for j=1:gs
        CA=circshift(A,[array(i),array(j)]);
        P=(A-CA>=0).*P;
    end
end
A
P
```

Try several values for g_s. Note the effect of changing g_s. In your code, set $g_s = 9$, i.e. 4 points in each direction.

What to do: Given the log spectrogram `S`, get the boolean matrix `P` in the same size, where if the entry in `S` is a local peak, the corresponding entry in `P` is 1, otherwise 0. Hint: there should be 4802 peaks.

Prob. 3.3: Calculate how many peaks there are and record your answer. How many peaks are there per second on average? Show the 1st column of P of the song ajt222_music.mat

2.4 Thresholding.

We want to use only the larger peaks. 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.

What to do: Find a threshold on the magnitude of the log spectrogram `S` such that on average there are 30 peaks per second. Apply the threshold to get a new `P`. Hint: After thresholding, the P matrix for ajt222_music should have 1800. You want to make this procedure automatic such that it will work for all songs in the database.

Prob. 3.3: What is the threshold of the log magnitude of S should we use if we want on average there are 30 peaks per second. How many nonzero items are there in the last column of P if we use this threshold.

2.4.1 Constructing the Table.

As outlined in the beginning of Sec. 3, 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). You can use the first three magnitude the `find` function returns to you.

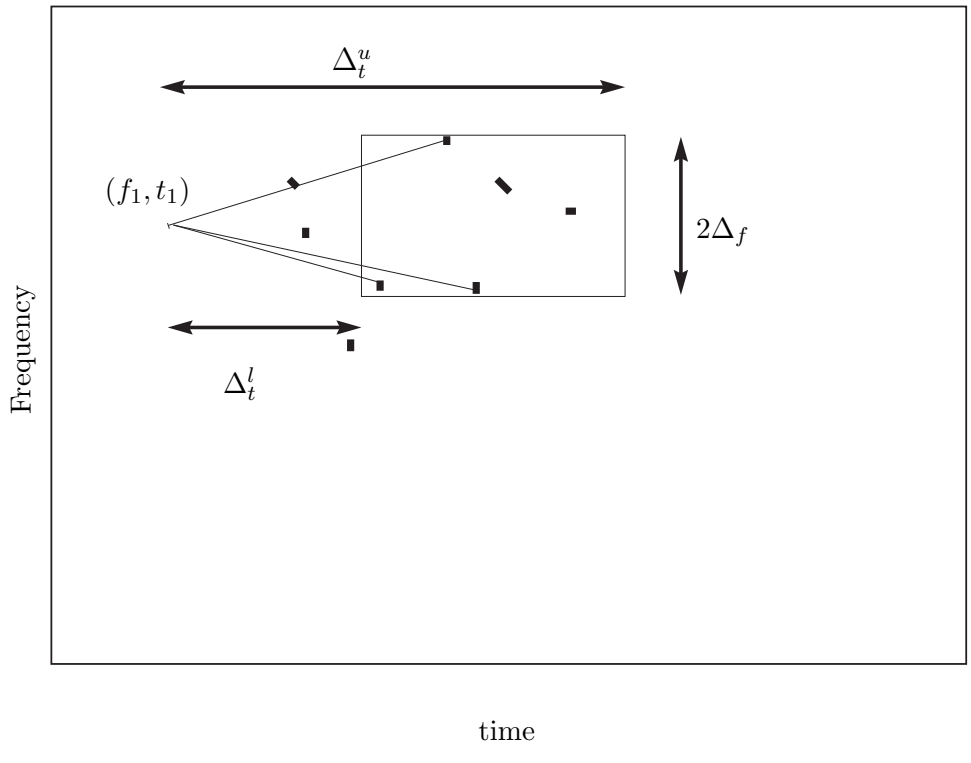

Figure 2: Peak pairs

So a peak located at (t_1, f_1), can only be paired with peaks which have $t_1 + \Delta_t^l \leq t_2 \leq t_1 + \Delta_t^u$ 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.

The commands `find` and `ind2sub` might come in handy.

Our objective is to select appropriate parameter values for Δ_t^l, Δ_t^u, Δ_f and fan-out, and then record the 4−tuples $(f_1, f_2, t_1, t_2 - t_1)$ in a matrix.

What to do: For the new `P`, find the 4-tuples $(f_1, f_2, t_1, t_2 - t_1)$ with $\Delta_t^l = 3$, $\Delta_t^u = 6$ and $\Delta_f = 9$. These parameters should be able to be set through `deltaTL`, `deltaTU` and `deltaF` outside the function `make_table`.

2.5 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 ×4 matrix which contains in each row the 4−tuple corresponding to a peak pair:

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

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.

What to do: Store all 4-tuples $(f_1, f_2, t_1, t_2 - t_1)$ in a matrix called `table`. You may want to iterate it in P column by column. You do not need to rotate when you are at the edge of the matrix P. And there should be 545 rows in the table.

Prob. 3.6: What are the first 10 **rows of the table?**

3 In-Lab instruction

In the in-lab, we will learn to build the database (sec. 4.2) and how to find the matching (sec. 4.3).

Now you have the function **make_table** that takes as input a clip and returns a table of peak pairs.

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

And you can run **make_table** on each song in the database and you can produce a large table.

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

If the clip comes from a particular song, we expect each entry in the clip table to have a corresponding entry in the song table. In particular, if $(f_1, f_2, t_1^c, t_2^c - t_1^c)$ is an entry in the clip table and $(g_1, g_2, t_1^s, t_2^s - t_1^s)$ is its corresponding entry in the song table, then we should have $f_1 = g_1$, $f_2 = g_2$, and $t_2^c - t_1^c = t_2^s - t_1^s$ because these features are time shift invariant. So using the triple $(f_1, f_2, t_2^c - t_1^c)$ from a clip table entry is a good way to search for a match from the set of song tables.

To make this process efficient, the song tables need to be combined to form our database of song "fingerprints".

3.1 Finding a Clip Match.

To identify a clip, we run the function **make_table** on the clip to generate a clip table of peak pairs. Then we search the database for matches to each entry in the clip table. We want to use the triples $(f_1, f_2, t_2^c - t_1^c)$ to search the database. What we need is a fast way to determine if $(f_1, f_2, t_2^c - t_1^c)$ matches something in the database and if so, extract what we know about that match.

Hash table will help to ensure fast database lookup. But for simplicity we will just combine entries of the table $(f_1, f_2, t_2^s - t_1^s)$ into a more compact one using a hash function $h(f_1, f_2, t_2^s - t_1^s)$ which will will be detailed described later. For each entry in the compact song table,we place the songid, the t_1^s, and the $h(f_1, f_2, t_2^s - t_1^s)$.

Then given a peak pair entry in the clip table, say $(f_1, f_2, t_2^c - t_1^c)$, we look up the database entry for $h(f_1, f_2, t_2^c - t_1^c)$. There will be either hit, or there is one hit with (songid,t_1^s,$h(f_1, f_2, t_2^s - t_1^s) = h(f_1, f_2, t_2^c - t_1^c)$) or multiple hits.

All the matches from the clip table to the correct song should occur with the same difference $t_o = t_1^s - t_1^c$, where t_1^s is the time of the pair in the song, and t_1^c is the time of the pair in the clip. The time $t_o = t_1^s - t_1^c$ is the offset in time we would need for the clip constellation to line ip with the song constellation. This suggests that we find the songid that has the most matches occurring with the same offset t_o and assert that our clip comes from that song, or rank the matches by this criterion.

4 In-Lab Procedure.

Complete the following tasks. You must have all of the TA checks completed by a TA in order to receive full credit for the lab. Label all figures.

You will need to write a few different scripts and functions. The following is the structure of scrips and functions. Some of them are provided, and some are partially completed.

Script `make_database.m`:

1. Apply `make_table` to each song in the folder "songs" so we have a large table as described in sec. 2.3.

2. Apply the `hash` function (described later) to the table to convert it from 4 columns to 2 and save it into the table named as `hashTable`. One example could be $[h(f_1, f_2, t_2 - t_1)\ t_1$ songID;...]. The songID is the order of the song in the database. Use `save` command to save the hash table into `hashTable.mat`.

3. Initialize `musicTable` and store the name of the music into it in order. For example: [ajt222_music;am859_music;.....]. Save the `musicTable` into `musicTable.mat`.

Function `match`: Take the file name of the clip and try to match it to one of the songs in the database and return the matching name. You need to write your own codes.

1. Load `hashTable` and `musicTable` that constructed in the previous

2. Run `make_table` and `hash` on the input clip to produce a hash table for it. Use the given clip "sample.mat".

3. For each $h(f_1^c, f_2^c, t_2^c - t_1^c)$, look for it in the `hashTable`. When one match is found, record the time offset $t_o = t_1^s - t_1^c$ and `songID` in a matrix `matchMatrix` like [t_0 songID; t_0 songID;...]

4. After going through the entire clip table, there will be a collection of $t_1^s - t_1^c$ values for each song in the database. The song that the clip matches will have a large mode, or a spike in the histogram of $t_1^s - t_1^c$ values. Determine a match by using `hist` or `mode`.

One possible function header could be

```
function songName=matching(clipName,hash,gs,deltaTU,deltaTL,deltaF)
```

Some steps are now discussed in detail:

4.1 Reading all files in a folder

We need to apply `make_table` for all the songs in the folder "songs". You may find these code useful:

```
files= what('/songs');
matFiles= files.mat;
fileName=matFiles{1};
toRead=['songs/' fileName];
```

All the names of files in the folder "songs" are stored in the cell `matFiles`. And the directory of the first song is now stored in `toRead`.

What to do: Apply `make_table` to all 52 songs in the folder "songs" and store the feature in a large 5 columns table, like [f_1,f_2,t_1,$t_2 - t_1$,songID;...].

4.2 Constructing the Hash Table

Since each frequency can be represented by a number from 0 to 255 (assuming for convenience that we drop the last (highest) frequency bin) we can represent each frequency with 8 bits. Depending on how large a time window we allow for our target box, we can also represent the time $t_2 - t_1$ by a n bit number, for some small n. Usually a hash function maps large data sets to smaller data sets, thus different inputs might be mapped to the same hash value. For this lab, to make things easy, we use a simple hash function of the form:

$$h(f_1, f_2, t_2 - t_1) = (t_2 - t_1)2^{16} + f_1 2^8 + f_2$$

(make sure that f_1 and f_2 range from 0 to 255, not 1 to 256. You should subtract f_1 and f_2 by 1.)

You can first run `make_table` to one song, apply hash function to it, add it to the `hashTable` then run `make_table` to the next. Or you can run `make_table` to each song to form a large table and then apply hash function to the large table.

What to do: Given a big table of songs peaks, convert it into the form of [$h(f_1, f_2, t_2 - t_1)$ t_1 songID;...] and save it as `hashTable`.

Prob. 6.1: What is the first 10 **rows of the hash table of the whole database.**

4.3 Finding a Match: matching

Next, write a function `matching` which takes in a clip and returns the song ID of the match, if there is a match. To find a match, the function will have to construct the peak pair table for the clip and then look up each entry in the hash table. The function should then find the song ID which has the most pairs with matching $t_o = t_1^s - t_1^c$, where t_1^s is t_1 for the pair in the song and t_1^c is t_1 for the pair in the clip.

You may find the next script useful:

```
%%Example of matching
%%A is our hash table of all the songs in the database. In this small
%%example, we have 3 songs. b is the hash table of the clip
%%[h t1 songid]
A=[1 2 1;
2 3 1;
1 2 2;
2 4 2;
3 5 2;
1 4 3;
2 3 3];
b=[1 1;
2 2];
%% matching terms of b in A
matchingMatrix=[];
for i=1:size(b,1)
index=find(A(:,1)==b(i,1));
matchingMatrix=[matchingMatrix; A(index,2)-b(i,2) A(index,3)];
%%record all the t_o and songid with matching hash function
end

%Build a table, column num = num of songs, row num = num of t_o=t_1^s-t_1^c.
%Each hit of t_o in a particular song, we add 1 to that entry corresponding
%to the song and t_o
recordMatrix=zeros(max(matchingMatrix(:,1))-min(matchingMatrix(:,1))+1,max(A(:,3)));
for i=1:size(matchingMatrix,1)
    recordMatrix(matchingMatrix(i,1)-min(matchingMatrix(:,1))+1,matchingMatrix(i,2))=...
        recordMatrix(matchingMatrix(i,1)-...
        min(matchingMatrix(:,1))+1,matchingMatrix(i,2))+1;
end
%%find which song has the most number of same t_o
[~,songID]=max(max(recordMatrix));
```

What to do: Given a clip, apply the `make_table` function and apply the hash function to the clip table. Look the clip table up in the database (hashTable) and return the song name.

4.4 Main Function

Write a main function so that you can test your program out using the "sample.mat" provided in the folder "sample".

Prob. 6.2: What is the name of this clips?

5 Summary of submission

1. Script of the main script, **make_database**, **matching** and all other code (on paper).
2. Answers to all the problems asked in bold face.