Last updated 7/11/12

Song Browser: Troyer Lab Software for Song Analysis

Table of contents

Introduction

This package is an attempt to gather, debug and document a suite of Matlab® functions used in the Troyer Lab to analyze, visualize, and categorize data from the spectral content of birdsong. Visuals aids and metrics associated with the dynamic time warping of categorized data are implemented to provide a more thorough look at the output. This feature of the package can also be used to investigate outliers in the data. Many of the procedures and algorithms are similar to or even identical with other common methods of song analysis. We are not attempting to replace or argue for any particular package or methodology. Eventually, however, we hope to compare and to benchmark the performance of different algorithms.

The package contains research level computer code. Debugging and documentation are not complete and functions should be used with a degree of skepticism. There has been no real attempt to optimize processing speed or memory usage. We hope to continually improve both the code and the documentation, but this is an ongoing process. We plan to gather this together into a more formal methods paper sometime in the near future. If you are reading this and attempt to use this code, please send an e-mail to todd.troyer@utsa.edu.

This manual first presents a components section in order to inform the researcher how the suite of functions is used. The following data structure setup section addresses how data must be formatted in order for the functions to work properly. The remaining sections of the manual provide a more in depth explanation of functionality and procedures for each component of the package.

We have included two small data sets with the minimal requirements needed to test out the procedures in this package. These are found in the bird and birdlarge directories. It is intended that the user could use the hand labeling procedures to hand label the songs in the bird directory, and then use this labeling as a 'seed' to perform automatic annotation of the songs in the birdlarge directory.

Components

The package is divided into five general functionalities that are commonly used in the analysis of birdsong data. Several of the components could be performed independent from one another.

Segmentation

As part of our original collection system, sound was segmented into small "clips" based on an amplitude threshold and duration requirement (Glaze & Troyer, 2006). Clips that are separated by less than 200 msec were glued together to form a recording (presumptive song) and stored as .wav files. Information about the clips and songs are stored as Matlab® structures and saved as matlab formatted data files. These data structures and the corresponding .wav files are the starting point for the use of our software. It will be assumed that data has been segmented and the required information stored in the proper format. This format will be described in the *data structure* section of this manual.

Hand labeling

The hand labeling process provides a method for manually classifying birdsong syllables into categories. The researcher may choose to label songs in order to identify specific stereotyped components or patterns of interest. During the labeling process, one can review their labeling scheme and make changes. The hand labeled data set can be used as is or to validate the performance of automated annotation algorithms. In our lab's case, the hand label data was used as part of supervised annotation algorithm (K Nearest Neighbors).

Calculate Features

All the segmented song data is converted to spectrograms, one spectrogram per syllable. From these spectrograms, we calculate song features that can be used to characterize song acoustics. Features are calculated for each time slice in the spectrogram. It is often useful to combine the feature values across slices within a syllable so that all syllables are described by the same number of feature values. To do so, for each feature we calculate the mean and standard deviation of the features values across slices. The package provides a way to visualize the syllable-level feature values through multiple scatter plots.

Annotation

Another purpose for feature calculations is syllable annotation, where different renditions of a syllable type are labeled with the same name (syllable A, syllable B, etc.). We calculate the mean and standard deviation of features values to represent each clip, and then use k nearest neighbors (KNN) clustering algorithm based on a normalized Euclidean distance to categorize syllables. By using this algorithm on an already hand-labeled set of songs, one can validate the expected accuracy of the algorithm. Also, the package includes visualization tools for manual inspection of the results of the automated annotation; these allow the package to be used in a semi-automated fashion in which a computer algorithm is used for initial classification, with a subsequent correction of borderline cases by a human observer.

Alignment

Syllables can be aligned in order to assess the spectral behavior within multiple sections. The package uses dynamic time warping in order to align classified syllables with an *averaged* template. As well, the alignment process allows for a general review of all syllables and their alignments against one another. The package has a graphical display available in order to visually see spectral behaviors within specified syllable categories. The software also offers distance metrics in order to assess outliers within categories. The alignment procedure gives a general sense of how well classification algorithms have performed for larger sets of information. The procedure is meant to act as another form of analysis, as well as a review process.

Data Structure Setup

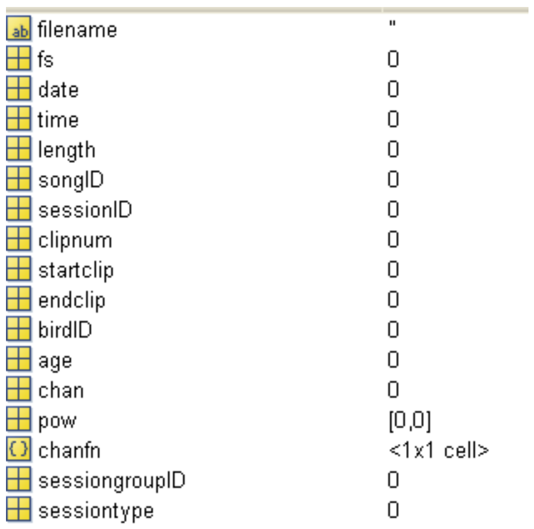
This section provides information on the data structures needed to make the software work. It is likely that some preprocessing will need to be done to convert your existing data structures into this structure to use SongBrowser. But before that, there are a couple of preliminaries to get things to work on your system. Most obviously, the programs directory and subdirectories should be added to the Matlab path. Also, some of the filename handling entails finding the full pathnames. To accommodate this code (which is likely to be replaced and updated), two functions need to be edited. First, **rootdirlist**.m holds strings that represent the primary drive/server to use; ie 'Y:', 'C:', '/Volumes/lab/', etc. **datadirlist**.m holds strings that represent the primary drive that holds the song data. Both of these functions, which are found in the utility subdirectory, should be edited to reflect the current user environment. Simply adding the drive name to both lists should be sufficient to get things to work.

The SongBrowser code is built around 'collections' of song data, thought of as a group of related songs. For example, these might be the songs recorded from a single bird on a given day during song development. All information about a collection of songs will be assumed to be stored in a single directory, with many of the files and subdirectories of this 'home' directory assumed to have the same name. While this is a bit restrictive, it enables the programs to find the relevant information without having to continually ask the user to provide information about the file or directory where that information can be found.

The two primary items contained in this core directory are a subdirectory containing the wav files that hold the raw data for the recorded songs, and a 'bookmark' or 'song' file that holds information about these songs at both the song and syllable levels of analysis. The 'bookmark' nomenclature arises for historical reasons, and these files have file extensions of .bmk, .sng. or .dbk (data bookmark file). We assume that a bookmark file will have name NAME.bmk and the subdirectory containing the wav files is named NAME_wav.

The bookmark file contains two variables holding information about the data at the song and syllable level: '*songs*' and '*clips*.' Listed below are some functions associated with the data structure setup process.

blanksongs.m—creates a 'song.a' structure which locates where and which clips are stored in .wav files. The *a* subfield stands for 'array' and is an array of structures holding the information about each song, so that songs.a(34) holds information about the 34th song in the collection, with subfields listed in the following figure:

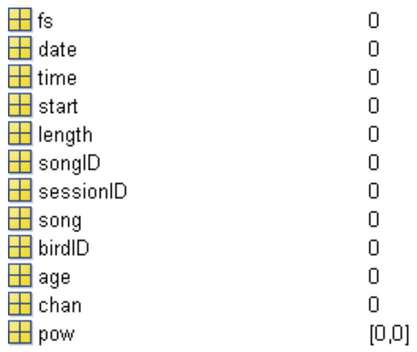


filename	''
fs	0
date	0
time	0
length	0
songID	0
sessionID	0
clipnum	0
startclip	0
endclip	0
birdID	0
age	0
chan	0
pow	[0,0]
chanfn	<1x1 cell>
sessiongroupID	0
sessiontype	0

The songs structure holds information about the songs within a given collection. Of these fields, *fs*(sample frequency), *length* (in samples), *filename*, *clipnum, startclip* and *endclip* are mandatory for the software package to work. The field *filename* holds the name of the wav file containing the raw data. The fields referring to clips will be explained below. The fields *birdID* (a number), *age* (days post hatch) and *date* (in Matlab date format) are optional but have obvious meanings. The other fields are specific to the song database in the Troyer lab and/or refer to the fact that much of our data is recorded in stereo from two birds housed in the same recording chamber (usually a live tutor and juvenile).

The package assumes that songs have already been segmented into syllables. In our previous recording system, sounds come pre-segmented into short sound 'clips.' Information about all clips within a collection of songs is stored in the clips variables.

blankclips.m—creates a 'clips.a' structure for segmented vocalization storage used throughout the package. To parallel the songs variable, clips has a single subfield called a, that is an array of structures with the following subfields:



fs	0
date	0
time	0
start	0
length	0
songID	0
sessionID	0
song	0
birdID	0
age	0
chan	0
pow	[0,0]

Of these fields, *fs* (sampling frequency), *start* (the number of samples within that song that marks the start of the clips), *length* (of the clip in samples), and *song* (explained below) are mandatory. As in the song variables, *date*, *birdID*, and *age* are optional, as are the other fields that are specific to the Troyer lab.

The *songs.a* array and the *clips.a* array list information about the songs/clips in the collection of songs in order. The fields *clips.a.song* holds the index of the song that clip belongs to and *songs.a.startclip* and *songs.a.endclip* hold the indices of the first and last clip in that song. For example, if the first three songs in the collection have 8, 12, and 11 clips respectively, songs.a(3). startclip=21, songs.a(3).endclip=31, and clips.a(25)=3.

The third critical file is a label file (with suffix .lbl) that holds the annotation information for all the clips in the clips variable. It holds a single variable labels, which has its own array subfield *labels.a* that is the same size as clips.a.

Hand labeling

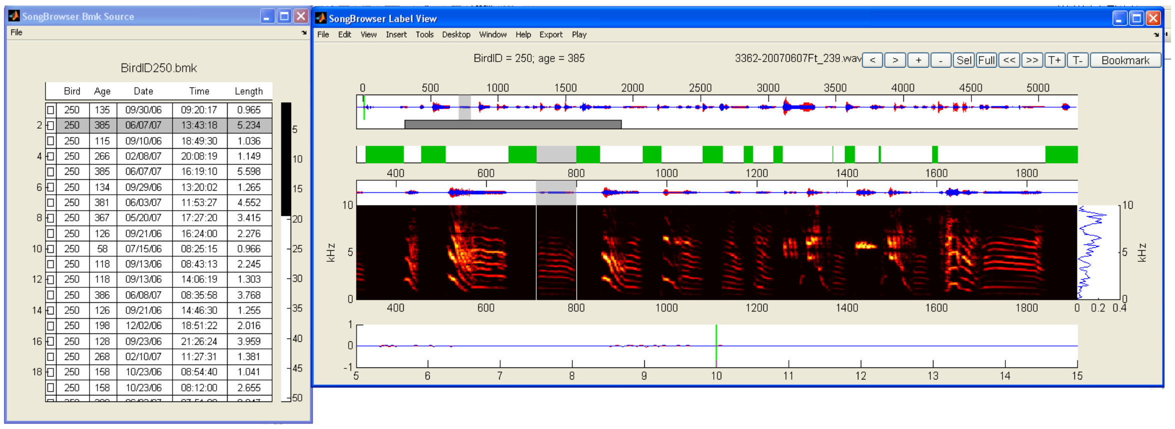
Goals

Hand labeling allows a researcher to evaluate the data set visually while labeling a sample of songs from a songbird. Hand labeling of smaller collections of songs can provide 'seed' information to guide the semi-automated labeling of much larger collections. The following procedure section introduces the functions used in the hand labeling process.

Procedures

sb3src_bmklbl .m – (UI SongBrowser Source, bookmark-label) -used to open and view a song file and its corresponding label structure if the label file exists in the selected song file directory. If label file does not exist, the function will request to create a label file for the hand labeling process. The figure provided below displays an example for the viewer:

Last updated 7/11/12



BirdID250.bmk

	Bird	Age	Date	Time	Length
	250	135	09/30/06	09:20:17	0.965
2	250	385	06/07/07	13:43:18	5.234
	250	115	09/10/06	18:49:30	1.036
4	250	266	02/08/07	20:08:19	1.149
	250	385	06/07/07	16:19:10	5.598
6	250	134	09/29/06	13:20:02	1.265
	250	381	06/03/07	11:53:27	4.552
8	250	367	05/20/07	17:27:20	3.415
	250	126	09/21/06	16:24:00	2.276
10	250	58	07/15/06	08:25:15	0.966
	250	118	09/13/06	08:43:13	2.245
12	250	118	09/13/06	14:06:19	1.303
	250	386	06/08/07	08:35:58	3.768
14	250	126	09/21/06	14:46:30	1.255
	250	198	12/02/06	18:51:22	2.016
16	250	128	09/23/06	21:26:24	3.959
	250	268	02/10/07	11:27:31	1.381
18	250	158	10/23/06	08:54:40	1.041
	250	158	10/23/06	08:12:00	2.655

Within the UI SongBrowser Label View, labels entered per clip can be any string format such as 'A','B','C', etc. The letter 'I' is reserved for introductory notes and there are several preset 'shortcut labels' for certain vocalizations and environmental recording phenomena. For example, a single press of the letter U will label that clip as being a call.

I	*Introductory notes*
T	*Tets*
U	*Call*
V	*Distance Call*
X	*Unknown*
Y	*Ch2*

'Ch2' refers to interference from the other bird in the recording chamber and is particular to the 2-bird, stereo recording done in our lab.

Using '=' prior to typing a set of letters allows the labeler to create an arbitrary label string such as 'ABC' or 'AB' or 'Fred' or whatever is needed.

Numbers from 0-9 can also be used in the labeling scheme. This can be useful for labeling different 'versions' of a syllable. Type the number first and then type the letter or string. The reverse order of typing the number first allows the program to automatically skip forward after typing a letter. Labels are assigned a number value of 1 by default and that value is not displayed.

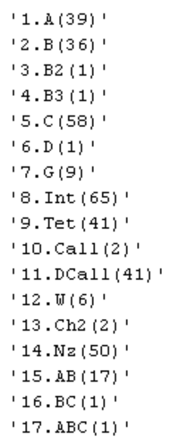
After selecting a label string, the labeler will progress through the song until it reaches the last segmentation. Arrow keys or the mouse can be used to navigate forward and backward. Within the UI SongBrowser Bmk Source, select the next song on the list and continue with the labeling process until all songs are labeled. Note that labels for the current song are not saved to the label (.lbl) file until the next song is loaded. So to save the labels for the last song, one must reopen an already labeled song using the UI SongBrowser Bmk Source.

Note that later routines in the package make a distinction between categories that contain a collection of clips where we expect a reasonable degree of similarity in the spectrograms (e.g. syllable 'A', intro note, etc.) and categories in which the elements won't necessarily have similar spectrograms (e.g. 'noise', 'Unknown'). We will call the former type of category a 'clustered' category and the latter a 'non-clustered' category. Depending on the bird and how finely one defines the categories, it may be appropriate to consider 'tets' and 'calls' to be 'clustered' or 'non-clustered.'

The package also includes a number of ancillary functions that can be used to adjust the label file without using the GUI.

makelabelkey.m—consolidates and writes the label key within a label file. If this is not performed, then the label structure may not be in the correct format. **Makelabelkey**.m is run as part of **calctemplates**.m (see below). One of these functions needs to be run after any changes to the label file.

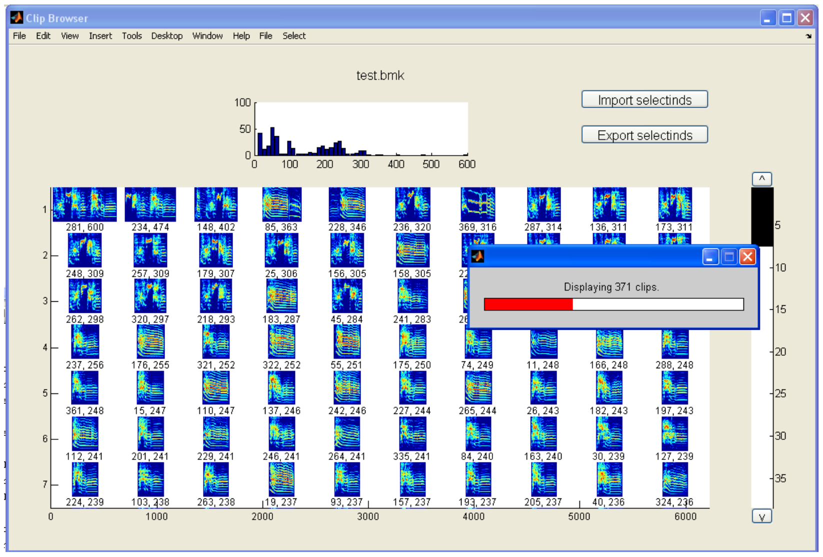
listlabels.m—while hand labeling uses strings to label each syllable category, much of the subsequent software relies on numeric category names (category '1', '2' etc. instead of category 'A','B' etc.). The numeric category name is known as a 'labelind'. listlabels will prompt the user to select a label file and will list the categories using in numeric order, showing the string label for each category and the number of clips in that category. Here's an example of the output:



```
'1.A(39)'
'2.B(36)'
'3.B2(1)'
'4.B3(1)'
'5.C(58)'
'6.D(1)'
'7.G(9)'
'8.Int(65)'
'9.Tet(41)'
'10.Call(2)'
'11.DCall(41)'
'12.W(6)'
'13.Ch2(2)'
'14.Nz(50)'
'15.AB(17)'
'16.BC(1)'
'17.ABC(1)'
```

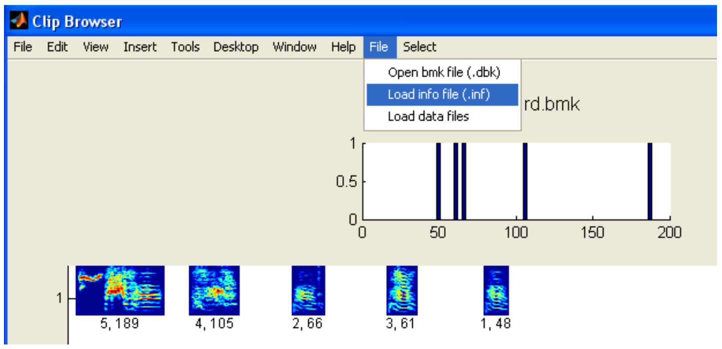
If the collection of hand labeled songs is relatively small, then the researcher might want to confirm that all syllables within a given category indeed have similar spectrograms. The package contains a general tool for viewing a collection of clips:

discliparr.m—used to inspect and view spectrograms for a collection of clips arranged in an array. Which clips are displayed is specified with the 'clipinds' parameter. For example, dispcliparr('clipinds',1:10) will prompt the user to select a bookmark file and then display the first 10 clips in that file. By default, the clips are sorted in descending order by length. However, the parameter 'values' can be used to specify a value to order the clips. For example, dispcliparr('clipinds',1:10,'values',1;10) will display the clips in reverse order; dispcliparr('clipinds',1:10,'values',1;10,'mode', 'ascend') will display the clips in numeric order. Often the value used to sort the clips has an intuitive meaning, and so the clip index and the value are shown below the displayed spectrogram. Also, the distribution of values is shown at the top of the figure:



Note that this function can be slow, particularly if the spectrograms have not already been calculated (by the calcspecs function described below).

In order to view a clip within a particular song, just double click on the spectrum. If you want to view the labels with the selected click you need to click on the second file drop down menu and select load info file (.inf); This will prompt a request for the song file. After selecting the song file, the function will prompt to save an info file which associates the viewer with a label file for the given song file. After doing so, when a spectrogram is chosen, the song viewer will return with the proper labels. The following figure shows the location for the above mention procedure:



Single clicking can be used to select a clip (outlined in green). Shift-clicking will add the current clip without deselecting other clips. To get the indices for the selected clips, click on the ‘Export selectinds’ button. This will make a variable selectinds in the workspace. If one wants to view a particular subset of clips within a larger set, load the larger set of clips into dispcliparr, set the variable selectinds to be the clips that you want to view, and click the button ‘Import Selectinds.’

movelbls.m—used to manually move clips from one category to a different category Ex: movelbls([372 250 100],6) ; move clips 372, 250, and 100 to category with labelind=6. (Clip numbers come from their position in clips.a and labels.a arrays.) For example, one could use dispcliparr to scan for mislabeled clips by selecting these clips with the mouse, exporting the indices to the workspace , and running movelbls(selectinds,10) , where category 1 is 'unknown' (for example).

newcat.m—used to create a new category for a given label file. Ex: newcat('AB') Will create a new category 'AB' which is empty. **Movelbls**.m can then be used to move existing clips into the created category. The function will request to save the new label file. After doing so, **makelabelkey**.m should be run in order to format the new label file correctly for further analysis.

calctemplates.m—calculates a 'template' for each category in a label file. The template for a given category is calculated as the average of properly shifted spectrograms of all clips in that category. We sometimes call the spectrogram of a single clip within a category 'an exemplar.' Templates are calculated by first centering all exemplars and then averaging. Next, each exemplar is shifted relative to the initial template until the best alignment is found, where best is determined by a mean absolute difference similarity measure (defined below). This process is performed on all exemplars and the template is re-averaged based on these shifted positions. This produces a new template. By default, this process terminates after 1 such iteration. (This default can be increased using the' iterations' parameter).

It really only makes sense to define a template for a clustered category. One can specify which categories have templates calculated by using the optional parameter/value pair 'cats'. For example, calctemplates('cats',[1:5 7 8]). By default, templates will be calculated for all categories.

Calculate Features

Goals

The package allows a researcher to calculate features from based on the spectrogram for each clip. By default, the following features are calculated for each time bin: amplitude, entropy, wiener entropy, frequency mean and standard deviation, fundamental frequency goodness, and amplitude difference (details explained below). Note that amplitude difference is based on our lab's specific recording setup involving two microphones.

General Procedures

The following procedures section states the functions used in the calculation procedure, and are ordered in a manner that represents the flow of the process.

calcspecs.m— this function is used to calculate and store the spectrograms for a given song file. The number of spectrograms will equal the number of clips in the song file (=length(clips.a)). To calculate each spectrogram, **calcspecs** calls the function **ftr_specgram.m,** which takes a spectrogram parameter structure as its second argument. By default, the spectrogram window size is 256 samples, nadvance is 64 samples (or amount overlapped is equal to 192), nfft is 256 (length of the fft), and sampling frequency is 24414.0625. The FFT settings may be adjusted in two ways. The function **defaultspecparams.m** sets the necessary spectrogram parameters:

```
params.window=256;
params.Nadvance=32;
params.NFFT=256;
params.fs = 24414.0625;
params.ampcut = .01;
params.specfloor = .05;
params.dt = 1000*params.Nadvance/params.fs; % width of each time bin in msec
df = params.fs/(params.NFFT*1000);
params.f = df*(0:floor(params.NFFT/2)); % centers of frequency bands in kHz
```

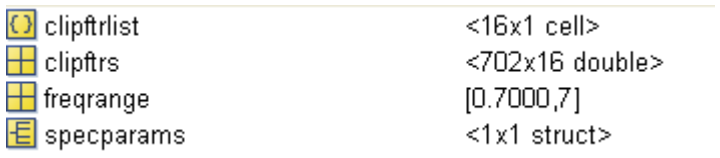
By editing this file, spectrogram parameters will be adjusted for all further calculations. One can also adjust the parameters one time by making a variable (e.g. p = defaultspecparams; p.Nadvance= 16), and then passing this to calcspecs: calcspecs('specparams',p).

Spectrograms are stored in the directory NAME_spec with filename NAME_spec_n.mat where n is the index of the clip. This file holds the variables spec, f and t, holding the spectrogram, the bin centers in frequency and bin centers in time respectively.

The data in our lab is segmented as part of a real time collection system that stores individual clips of sound. This is only a rough segmentation, and we perform an additional segmentation when running calcspecs. The amplitude variable *amp* is calculated by calculating the square root of the sum of the power (squared amplitude) in the frequency bins lying between .7 and 7 kHz. A further segmentation is accomplished by determining the first and last time bin that exceeds a fixed threshold *thresh* (=0.25 by default). These bin indices are stored in the variable *edges*. To bypass this further segmentation, use calcspec('thresh',0). For stereo recordings, the amplitude is calculated for both channels, with *amp2* being the amplitude from the directional microphone pointing away from the bird of interest. For mono recordings, *amp2* is set to be a vector of zeros. A single spec file in the spec directory holds the following variables: *spec, f, t, amp, amp2, edges, and thresh*.

calcftrslice.m—calculates the spectral features per window (or 'slice') for a given spectrogram. By default, the following variables are calculated: amplitude, entropy, wiener entropy, frequency mean, frequency standard deviation, fundamental frequency goodness, and amplitude difference (see below). The amplitude difference is used to separate birds in the stereo recording set up in the Troyer lab. This information is stored in the directory NAME_ftrs with filename NAME_ftr_n.mat where n is the clip index. This file stores 4 variables: sliceftrs, an MxN dimensional matrix where M is the number of time bins in the spectrogram for that clip and N is the number of features calculated; sliceftrlist, an N dimensional cell array holding the feature names; freqrange, a 2 dimensional vector holding the range of frequencies used to calculate the features (.7 and 7 by default); and specparams, the spectrogram parameter structure described above.

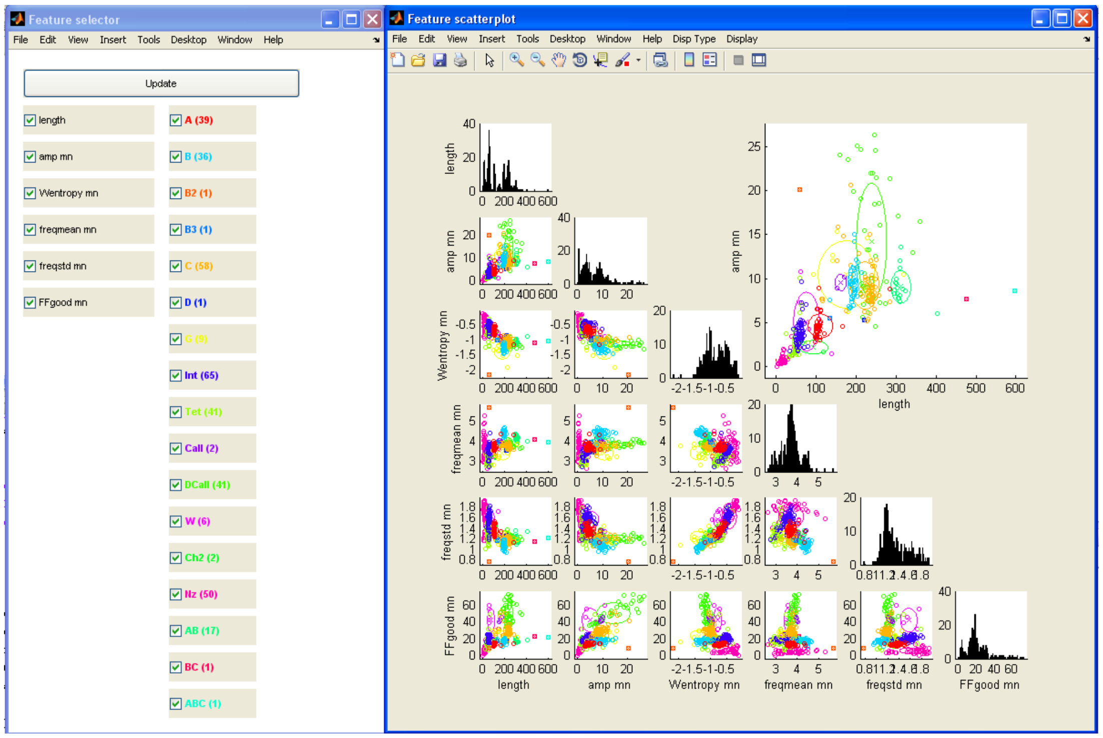
calcftrclips.m—for each clip, calculate the mean and standard deviation feature values across time bins. Also stores the clip length and a variable ampdiff, calculated for stereo data as the fraction of time bins in which the amplitude in the preferred channel is greater than the amplitude in the nonpreferred channel Note that since all clips now have the same number of features stored, the data is stored in a single file NAME.ftr:



clipftrlist	<16x1 cell>
clipftrs	<702x16 double>
freqrange	[0.7000,7]
specparams	<1x1 struct>

Clipftrs variable is the JxK dimensional matrix of feature values, where A is the number of clips and B is the number of clip features values, equal to 2 times the number of sliceftrs (mean and standard deviation) plus 2 (length and ampdiff). The *clipftrlist* variable holds the names of the feature values constructed as the slice feature name with the subscript '_mn' or '_std', plus length and ampdiff. *Freqrange* and *specparams* are the same as in sliceftrs.

Because all clips have the same number of clipftrs, they can be visualized as points in a high dimensional space. The function **ftrscatter**.m displays scatter plots of selected features pairs. Points are colored according to label (the function request sa labels file in order to run). The following is a figure for the viewer:


Feature selector
File Edit View Insert Tools Desktop Window Help
Update
Feature scatterplot
File Edit View Insert Tools Desktop Window Help Disp Type Display
length
amp mn
Wentropy mn
freqmean mn
freqstd mn
FFgood mn

✓ length	✓ A (39)
✓ amp mn	✓ B (36)
✓ Wentropy mn	✓ B2 (1)
✓ freqmean mn	✓ B3 (1)
✓ freqstd mn	✓ C (58)
✓ FFgood mn	✓ D (1)
	✓ G (9)
	✓ Int (65)
	✓ Tet (41)
	✓ Call (2)
	✓ DCall (41)
	✓ W (6)
	✓ Ch2 (2)
	✓ Nz (50)
	✓ AB (17)
	✓ BC (1)
	✓ ABC (1)

This view allows the user to assess the degree to which categories are clustered, and the features that do the best job of category segmentation.

Feature Calculations

Here are short descriptions of the feature calculations.

ftr_amp.m - square root of the total power (sum of squared amplitudes) for frequency bins within *freqrange*.

ftr_Wentropy.m - Wiener entropy is a measure of how similar a vector of positive values. It is defined as the log of the geometric mean divided by the arithmetic mean. The arithmetic mean of N numbers is the sum of the numbers divided by N. In analogy the geometric mean of N numbers is the Nth root of the product of those numbers. It's equivalent and better numerically to calculate the geometric mean as exp(mean(log(X)) where X is the vector of values. Since products are sensitive to very small values, one can show that the geometric mean is always smaller than the arithmetic mean. Therefore the ratio of the two is always smaller than one and the Wiener entropy is negative, with more negative values indicating the presence of entries very close to zero.

Several other features are defined by treating the amplitude spectrum (that falls within the limits given by *freqrange*) as a probability distribution. So first the amplitudes of the spectrogram are normalized by the sum of amplitudes. Below we will let the vector *p* be the vector of normalized amplitudes for a given time bin in the spectrogram, and let *f* be the vector of frequencies for those bins.

ftr_entropy.m – this is just the classical entropy of the amplitude spectrum, viewed as a probability distribution, i.e. -sum(p.*log(p)). Like the Wiener entropy, this captures how much the spectrum is spread equally across frequency bins.

ftr_freqmean.m – this is the mean (or centroid)of the amplitude distribution across frequencies, defined as the weighted sum of frequencies sum(f.*p). It gives some indication of whether power is concentrated in higher or lower frequencies.

ftr_freqstd.m – this is just the standard deviation of the amplitude spectrum, viewed as a probability distribution, and is equal to sqrt(sum(p.*(f-freqmean).^2)). This is one measure of how 'broadband' the spectrum is.

ftr_cepsFF.m – this function returns the fundamental frequency and goodness of fundamental frequency, determined by taking the peak of the cepstrogram. The cepstrogram is calculated by first taking the log of the amplitude spectrum. We apply a differencing filter to high pass filter this log spectrum, and then take the fourier transform to find periodicities in the spectrum. The fundamental frequency is defined as the periodicity in the spectrum with the peak amplitude (restricted to the range .5-2 kHz). The 'goodness' of this fundamental frequency (feature name 'goodFF') is defined as the magnitude of the cepstrum at this peak. We don't currently use the fundamental frequency as a feature since it is an unreliable measure in the case of sounds that are not strongly harmonic.

Note that we have essential 'thrown together' this handful of features. We have not worked hard on optimizing the parameters used to calculate features, or deciding on the 'best' collection of features to describe song. We plan to look at such issues, and add more features, in the future.

Annotation

Goals

It is relatively easy to record large collections of song. The first step in most analyses is 'annotation', putting a label on each clip of sound. Most methods of song annotation have two parts: the first entails calculating the similarity between clips, and the second entails clustering or grouping clips together based on this distance.

This package uses the k nearest neighbor (KNN) algorithm to perform clustering. This is a 'supervised' clustering algorithm that uses an already labeled set of clips as 'training data' for categorizing a new set of unlabeled data. To accomplish this, we use a small collection of 50-100 songs as training data and label these by hand. These data are then used to guide automated annotation of much larger sets of data. Since KNN implicitly assumes that clips will be labeled by being near a cluster of clips with a similar label, it only makes sense to measure the distance to clips in the training data that form a cluster, i.e.

Procedures: screening the data

Some categories of sound do not necessarily fall into a cluster, and some clips (such as cage noise or wing flaps) can reasonably separated from vocalizations based on certain parameters.

noisescreen.m—this function screens out artifacts that have extreme values for certain features. There are four variables used in this screening process:

1. Amplitude – very soft sounds are likely to be either cage noise or very soft 'meeps' or 'tets' that are so soft to analyze. Clips with low amplitude (< 1.0 by default) are assigned to the 'noise' category.
2. Length – very short sounds (< 30 msec by default) are unlikely to be syllables sung within a song and are also assigned to the 'noise' category.
3. Wiener entropy - vocalizations generally have power concentrated in a relatively narrow band of frequencies and/or contain strong harmonics, leading to rather large negative values for Wiener entropy. Sounds with low Wiener entropy (> -0.35) such as wing flaps and cage noise are assigned to the 'noise' category.
4. Ampdiff – only sounds that have greater power in the preferred stereo channel are likely to come from the bird of interest. Other sounds are likely corrupted by sounds from the other bird. So sounds with small values of ampdiff (<0.7 by default) are assigned to the 'Ch2' category.

The threshold values can be reset by using the ampthresh, lenthresh, WEthresh or ampdiffthresh parameters, e.g. noisescreen('lenthresh',32, 'ampthresh',0.75). The function will create ask to save a separate label file associated with the screening process. By default, 'unlabeled' will be category 1, 'Nz' will be category 2, and 'Ch2' will be category 3.

Procedures: calculating similarity

To calculate similarity, one needs a notion of ‘distance’ or ‘match.’ To accomplish song annotation, our package uses a distance measure similar to that shown in **ftrscatter**. However, different features have different units and so must be normalized in order to compare to one another. If x1 and x2 represent two vectors of feature values, our distance is defined as the normalized mean absolute deviation distance d=mean(abs((x1-x2)./k)). (This can be changed by setting the ‘metric’ parameter in the **calcdistftr** function described below).

calcmetric.m—used to calculate the store the normalizing factors with define a ‘metric’ for determining Euclidean distance. Our metric is defined based on a set of already labeled songs (the ‘traning data.’). First, the user must decide which of the categories in the training represent clustered categories (syllable A, intro, etc.) and which are not clustered (noise etc.). The function should be called with these categories as an argument, e.g. calcmetric(‘cats’,[1:5 7 8 10]). The normalizing factor for a given feature is defined by finding the median feature value for each category, then finding the distance between the feature value for each clip and the corresponding category median, and then averaging over all clips. Thus a distance of 1 means that feature value is at a typical distance away from the ‘center’ of each category.

calcdistftr.m—calculates the similarity between feature values of the clips in one song file and the feature values of the clips an a second song file. These values are stored as a matrix, where clips in the first file are indexed along the rows of the matrix, and the clips in the second file are indexed along the columns. Usually the first file, which we call the row file, will contain the already labeled training data, and the second file, called the column file, will contain the unlabeled data that needs annotation. To define the distance, the using a song file of interest and its accompanying feature file. The function can use two different song file inputs representing the rows (labeled) and columns (unlabeled) of the output matrix. The function also requests a metric file associated with the labeled song file (see **calcmetric**.m). Similarity measure used is Euclidean distance of normalized feature vectors. The user can list which features go into the calculation of the distance by using the ‘ftrlist’ parameter, followed by a list of clip features (the ones with the _mn and _std suffix).The following variables are outputted and saved in .mtch file: *colclips, colfile, colpath, ftrlist, m, metric, rowclips, rowfile, rowpath*. *Colclips/rowclips* are used to index the clips used, *col/rowfile* and *path* are used to store the file and path names, *metric* holds the weight values used per category, and the *m* variable represents the similarity matrix.

Procedures: clustering

knclust.m—the function performs K nearest neighbors clustering algorithm using the similarity matrix calculated from **calcdistftr**.m. For each clip in the ‘column’ file, the algorithm finds the k nearest clips from the ‘row’ file and labels the column clip with the most common label of its row neighbors. Ties (e.g. a clip is nearest to two ‘A’s, 2 ’B’s and an intro) are broken using a distance criterion. knclust.m defaults to k=5, but this can be changed,e.g. knclust(‘k’,10) for k=10, etc. The function requests the .mtch file associated with the row label file of interest and column (unlabeled file) of interest, saves its results in an .mlbl (match label) file in the directory of the column file (this is a labeling of the column clips).
Two Matlab structures are saved in the created file. The labels structure holds the same information

from a .lbl file, while the mtchlbls structure holds information about the matching process: labelinds, labeldist, labelnum, rowclips,colclips, and k. Labelinds indicates which categories the algorithm decided.

Generally, **knclust**.m does not include all clips in the row or column file. For example, an unlabeled data set is usually passed through **noisescreen.m** to detect outlier clips. These clips should retain their 'noise' labels and not be relabeled according to which clips in the row file happen to be close to them. The 'colcats' parameter can be used to specify which categories in the column files are considered for relabeling. Since **noisescreen.m** puts all clips that don't exceed one of the outlier thresholds into category 1 ('unlabeled'), knclust('colcats', 1) will apply k nearest neighbor clustering only to the unlabeled clips. Other clips will retain their labels. The rowcats parameter indicates which categorizes to use in assignment of unlabeled data. Since the KNN algorithm Implicitly assumes that the labeled data come in clusters, one should only consider the nearest row clips that belong to one of these clustered categories (and not e.g. the noise category). These categories can be specified with the rowcats parameter. For example, knclust('rowcats',[1,2,5,7,8,9,11,15],'colcats',1); will take all clips within category 1 in the column label file and assign them a category label that matches a category in the list [1,2,5,7,8,9,11,15] in the row file.

Procedures: validation

One method to assess the accuracy of annotation algorithm is to compare the results of the algorithm to manual annotation performed by a human observer. Since our KNN algorithm already relies on a set of hand labeled data, one can estimate the accuracy of the algorithm by 'leave-one-out cross-validation.' In short, one takes out a single clip and pretends that it does not have a label. Using the other clips in the labeled data set, the KNN algorithm is used to label that clip. Then one can compare the labeling that comes from the KNN algorithm to the original label. This can be performed by calling knclust with the 'selfclust' parameter set to 1:
knclust('selfclust',1,'rowcats',[1,2,5,7,8,9,11,15],'colcats',[1,2,5,7,8,9,11,15]). Note that rowcats and colcats should be set to the same categories since one is should apply the algorithm to the clips that are in the categories that are possible targets for matching. This will produce a .mlbl file that contains the results of the leave-one-out cross-validation process.

dispconfuse.m—displays the confusion matrix based two different labelings of the same set of clips. The function will prompt the user to load two label files. The first file is the row label file and is typically the original label file. The second file is the column file and is typically the .mlbl file obtained from the leave-one-out cross-validation. The figure below shows the graphic output for the confusion matrix:

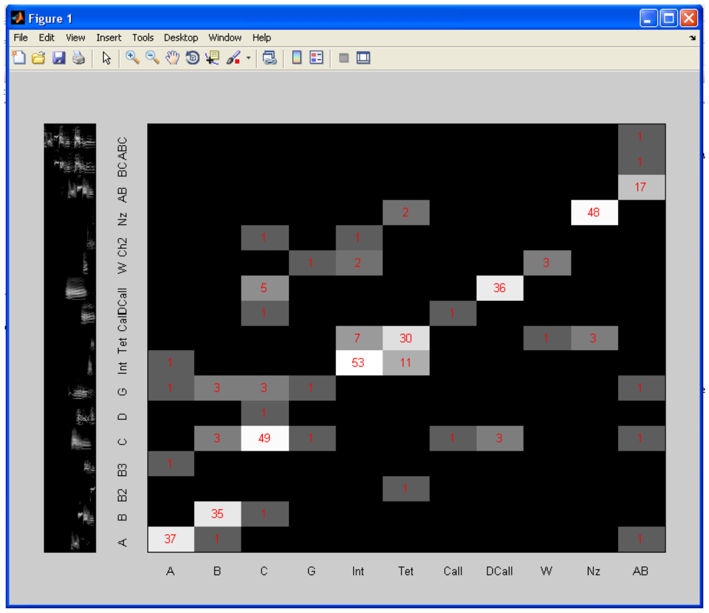


Figure 1
File Edit View Insert Tools Desktop Window Help
ABC BC AB Nz Ch2 W DCall Call Tet Int G D C B3 B2 B A
A B C G Int Tet Call DCall W Nz AB
1
1
17
2 48
1 1
1 2 3
5 36
1 1
7 30 1 3
1 53 11
1 3 3 1 1
1
3 49 1 1 3 1
1
1
35 1
37 1 1

The numbers and the grey scale represent how many clips have the corresponding row and column labels. Observing the confusion matrix allows the researcher insight into how the algorithm may be performing, and which categories are most easy to confuse. From another point of view, mismatches between the algorithm label and the hand label my be a human error.

To find out which clips fall in each bin one can run the function **gciconf**.m, which will retrieve the clip indices from any of the cells in the confusion matrix structure defined in dispconfuse.m.   The following is an example of this procedure:

d=dispconfuse;

gciconf(d,'d','c') % 'd' being the row, 'c' being the column;

ans   =

60

In this example, there is one clip that was labeled 'd' in the row label file and 'c' in the column label file, and the index of that clip was 60. Note that the clip indices that are output from gciconf can be used as inputs to the **dispcliparr** function, allowing the user to examine the spectrograms of those clips. Double clicking those spectrograms, will then bring up a view of the entire song in which that clip was sung.

Alignment

Goals

Previously described analyses compare clips by measuring distances between feature values that represent averages or standard deviations over the entire clip. These measures do not account for temporal structure within a syllable and do allow for the analysis of subsyllabic structure across syllables. Such analyses requires the alignment of subcomponents of the syllable. Alignment can also aid in syllble-to-syllable comparison by enabling a 'point-to-point' comparison of the 'same' parts of the syllable.

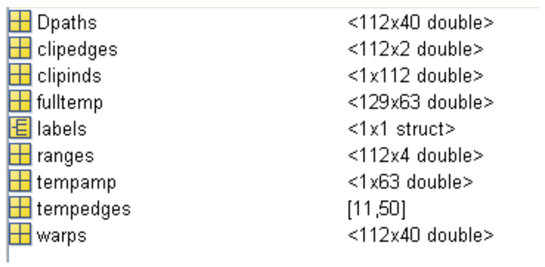
Procedures

Our package uses dynamic time warping (DTW) to address the temporal alignment problem. In short, dynamic time warping finds the 'best' alignment between two clips (e.g. Glaze and Troyer, 2006). This is accomplished by first calculating a matrix of values where the (i,j)th entry in the matrix holds the similarity between the ith time slice in the first clip and the jth time slice in the second clip. At this point our algorithm uses the mean absolute deviation between the slices of the spectrogram to calculate similarity (restricted to frequency bins within *freqrange*). Deciding on which of the many choices of similarity are optimal is an area of ongoing research.

calcdtw.m. Specifically, for each category we calculate an averaged template (see calctemplates.m described above), and then for each clip in that category we use DTW to align that clip to the template. After finding the alignment, one can locate points of interest in the template, and then find the corresponding time slice in every example of that category. **Calcdtw.m** requests a label file that holds the template information, including the averaged spectrogram. (Note the warping is done for the portion of the template and exemplar that exceed the amplitude threshold we set for more refined segmentation.) A second label file is selected that specifies which clips will be warped to the selected templates, and a spectrogram directory is requested that specifies where the location of the spectrograms of those clips. One generally only uses DTW on clustered categories; the 'cats' parameter can be used do specify these categories, e.g. calcdtw('cats',,[1,2,5,7,8,9,11,15]). Note that one might use a hand labeled subset of data to define a set of templates, and then warp syllable examples from a different (larger) song file that have been labeled by the KNN or another automated algorithm for annotation.

PROGRAMMING NOTE: **Calcdtw.m** relies on a mex file dtwmex.c. Finding the best alignment is calculation intense and so we relied on compiled code to speed processing. The use must run mex dtwmex.c on their system for **calcdtw** to work. See Matlab documentation on mex files.

calcdtw creates a _dtw directory which stores information on alignment:



Dpaths	<112x40 double>
clipedges	<112x2 double>
clipinds	<1x112 double>
fulltemp	<129x63 double>
labels	<1x1 struct>
ranges	<112x4 double>
tempamp	<1x63 double>
tempedges	[11,50]
warps	<112x40 double>

dtwalignftrs.m—calculates the aligned feature values resulting from dynamic time warping. Given the time alignment from DTW (which is based on the similarity of the spectrogram), one can then align the feature values for all exemplars that have been aligned. In our case, for each time bin in the template spectrogram, one finds the corresponding bin in each examplar, then copies the feature values into a matrix which has one row for each features and one column for each bin in the template (that exceeds threshold). Since we have one of these matrix for each feature, **dtwalignftrs.m** actually constructs a three dimensional matrix of size X x Y x Z, where X is the number of features, Y is the number of clips, and Z is the number of time bins in the template (above threshold). This is stored in the *ftrs* field in the variable *alignftrs*. *Sliceftrlist* variables lists the features calculated per time slice. *Clips* variable states the clip indices in the category from the song file. The function requires a 'cats' parameter indicating which categories to perform calculations for (likely the same categories as those in **calcdtw**). Function will also request for _dtw directory as well as a _ftrs directory to find the data needed for the calculations. For each category indicated, an *alignftrs* file will be created in the _dtw directory that holds information about the dynamic time warping feature calculation procedure.

The *alignftrs* variable also has a number of fields that are related to assessing the how similar the feature patterns are for exemplars within a category. These fields are preliminary and are the subject of ongoing research.

plotalignftrs.m—displays the aligned features traces per exemplar for a given category with its averaged template. This allows the user to assess the consistency of feature trajectories across a syllable, and to look for outliers in the data.