melody: Statistical Methods for the Quantitative Analysis of Song Spectrograms (Version 0.4.3)

David M. Schruth
dschruth@u.washington.edu

December 29, 2012

1 Licensing

This package is licensed under the Artistic License v2.0: it is therefore free to use and redistribute, however, we, the copyright holders, wish to maintain primary artistic control over any further development. Please be sure to cite us if you use this package in work leading to publication.

2 Installation

Building the *melody* package from source requires that you have the proper dependency packages, *caroline*, *network*, and *sna*, installed from CRAN. This can typically be accomplished via the following commands from within the R command line environment:

```
install.packages(c('caroline','network','sna'))
```

After a successful installation the *melody* package can be loaded in the normal way: by starting R and invoking the following `library` command:

```
> library(melody)
```

3 Introduction

The melody package is a suite of computer vision tools for processing vocal spectrograms. In addition to temporal partitioning, unit filtering, harmonic splitting, and background noise removal functionality, the package provides a collection of methods for assessing the higher order melodic content of vocalization spectrograms. Both unit level (tone and interval) as well as inter-unit measures (syllabic variation and repetition clustering) are also currently available.

4 Data Input

The basic unit of analysis in the *melody* package is the spectogram: an image representing the variation in spectral density of a (sound) signal over time. This can be represented as a two dimensional matrix of signal intensity values with rows corresponding to different frequency ranges and columns corresponding to different slices of time. One easy way to import a spectrographic image a matrix in R is to first convert the image into Portable Greymap [PGM] format, where each spectrographic intensity is coded as a greyscale value typically between 0 and 255 (the minimum and maximum values for each pixel). For an example: we read in an image in the PGM format below:

```
> pgm <- readPGM(system.file('extdata','calls','Tarsius-spectrum-Duet.female-Nietsch1998-2b2
> dim(pgm)
```

```
[1]   81 1106
```

```
>
```

5 The Spectrogram Object

The spectrogram object is essentially just a list composed of at least one element: our spectrogram matrix we just read in via `readPGM`. Additionally, after processing, this list also stores a list of matrices of the partitioned spectrogram units, vectors of various statistics on each of these units, and eventually, after clustering, a similarity matrix between different units. The object is instantiated using the `sg` function and requires only a single matrix as input. For an example, instantiate a spectrogram object and demonstrate the object structure and basic plotting functionality.

```
> s <- spectrogram.object <- sg(pgm)
> str(spectrogram.object)
```

```
Formal class 'sg' [package "melody"] with 1 slots
  ..@ .Data:List of 3
  .. ..$ : num [1:81, 1:1106] 255 255 255 255 255 255 255 255 255 255 ...
  .. ..$ : int 1106
  .. ..$ : int 81
```

```
>
```

```
> plot(spectrogram.object)
```

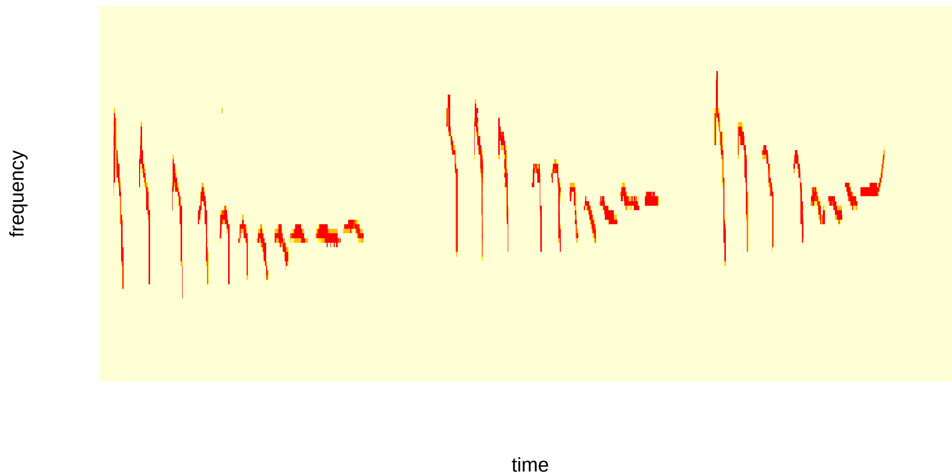

Figure 1: A spectrogram of the female component of a Tarsius spectrum duet (via the 'plot' S3 method)

5.1 Background vs Foreground Threshold Determination

The first somewhat trivial (but important) order of business when creating a spectrogram object is determining an absolute background/foreground threshold greyscale value. This is accomplished using the `threshold` function. In this example we use a 89 percent of the highest greyscale value as our threshold.

```
> s <- threshold(s, pct.max = 0.89)
> print(names(s))

[1] "x"             "width"         "height"
[4] "gray.dev"      "gray.mean"     "gray.max"
[7] "bg.threshold"

> s$bg.threshold

[1] 226.95
```

5.2 Partitioning

The first substantial step in processing a spectogram is actually breaking it up into separate units for downstream analysis and clustering. This is accomplished by looking for relatively quiet breakpoints in the spectrogram along

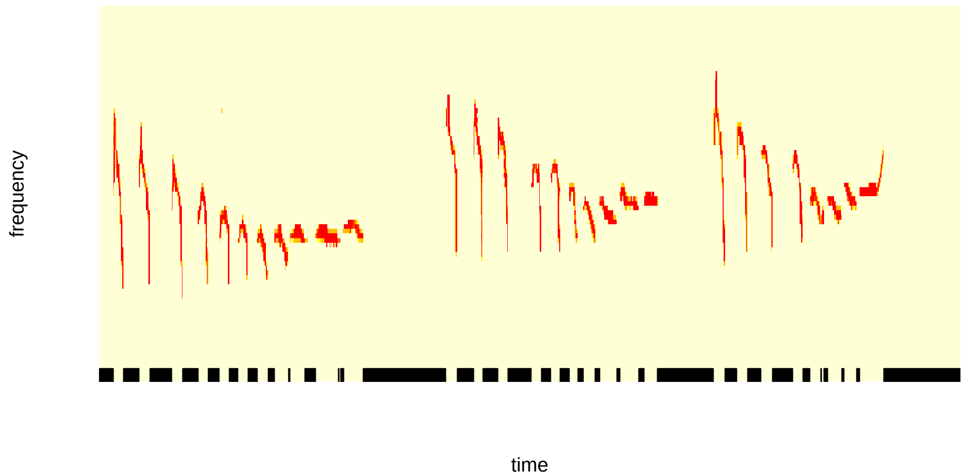

Figure 2: A partitioned spectrogram (black bands represent quiet breaks)

the time axis. Two variables (in addition to the threshold value) are used to guide this relativistic splitting process: the first is the average row value and the other is the variance in row values. The logic is that background noise will often appear as a light horizontal band punctuated by actual (higher amplitude) signal. A lambda weighting factor can additionally be used to determine how far down-biased (into the lower frequencies) this additional splitting equation carries.

```
> s <- partition(s, lambda=5)
> s$n

[1] 31

> length(s$units)

[1] 31

> plot(s)
```

5.3 Unit Statistics

The next important step in processing a spectrogram is calculating the simple statistics on the dimensions of each partitioned unit. By default the following statistics are collected: height (frequency difference), width (time difference), mean y-value (weighted mean frequency), mid x-value (median time value), mid y-value (median frequency).

```
> s <- unitstats(s)
> unit.param.names <- grep('^u.*[^sl]$', names(s), value=TRUE)
> print(sapply(unit.param.names, function(x) s[[x]]))

     u.width u.xavg u.ymin u.ymax u.ymid u.ylen  u.ymean
31        13   24.5     21     59   40.0     38 42.56846
65        14   58.0     22     56   39.0     34 43.16582
107       14  100.0     19     49   34.0     30 39.50602
140       13  133.5     22     43   32.5     21 36.07063
167       13  160.5     22     59   40.5     37 34.43226
191       13  184.5     23     36   29.5     13 32.13547
217       14  210.0     23     34   28.5     11 29.23475
243       18  234.0     25     34   29.5      9 30.47070
264       19  254.5     31     34   32.5      3 32.23043
307       29  292.5     30     34   32.0      4 32.06903
339       25  326.5     31     35   33.0      4 33.50951
460       15  452.5     28     62   45.0     34 51.01239
493       12  487.0     27     61   44.0     34 51.18855
525       13  518.5     29     57   43.0     28 48.87490
568       13  561.5     29     47   38.0     18 43.28964
592       12  586.0     29     48   38.5     19 43.66228
615       11  609.5     31     43   37.0     12 38.44107
637       15  629.5     31     40   35.5      9 36.45987
665       22  654.0     35     40   37.5      5 36.84981
693       24  681.0     38     43   40.5      5 39.60416
717       17  708.5     39     41   40.0      2 39.81058
804       15  796.5     26     67   46.5     41 50.86948
833       14  826.0     29     56   42.5     27 49.12722
865       15  857.5     29     51   40.0     22 46.12646
904       13  897.5     31     50   40.5     19 44.04279
927       14  920.0     35     42   38.5      7 39.29629
931        3  929.5     35     38   36.5      3 35.96113
954       18  945.0     35     40   37.5      5 37.99889
973       16  965.0     38     43   40.5      5 40.60740
1008      31  992.5     41     51   46.0     10 42.74411
     u.intensity
31         27711
65         23975
107        20514
140        16488
167        15823
191        10111
```

```
217         11370
243         14155
264         12353
307         21776
339         13371
460         23870
493         20535
525         21134
568         14079
592         13338
615          9530
637         10172
665         13548
693         15022
717         11337
804         26988
833         20902
865         14305
904         15245
927         10334
931          2205
954         11027
973         10170
1008        18524

>
```

5.4 Spectrogram Cleaning via Unit Filtering

Next we demonstrate how the `clean` function removes units depending on if it is too long or too brief temporally, too high or low frequency, or too small of a frequency difference (each unit must be at least 3 pixels high), or if it is too weak (each unit must be at least 1% of the total amplitude of the entire spectrogram).

```
> s <- clean(s)
```

6 Unit Level Scoring

Next we perform unit level melodic scoring for tone and interval. Tone is currently best calculated using the percentage of pixels below the background threshold ['pctbg'] or how 'white' the unit is: the whiter the unit the less noisy and the more tonal. Interval currently uses the ratio of the absolute change in per column mean-frequency over the difference between the max and min above-threshold frequencies ['ymRpuH'] (thus thick bands will not score higher interval scores than thin bands with the same slope). Each of these scores are averaged over the whole spectrogram and unit-level scores can optionally be weighted by unit-intensity.

```
> s <- tone(s)
> s$tone
```

```
[1] 0.9307188
```

```
> s <- interval(s)
> s$interval
```

```
[1] 0.7054849
```

7 Unit Clustering

The final processing step is to use all of our new unit level statistics to cluster the units into a mathematical graph where each unit is a node and a match between nodes is an edge. There are currently two ways to match nodes and both involve creating a list of distance matrices for the differences in each statistic between each of the units. The first method averages all of these difference matrices and has a single cut-off value to convert the single valued matrix into a binary adjacency matrix. The second method performs the binary (TRUE/FALSE value) determination first, using an array of limits (one limit per statistic), and then collapsing these binary matrices, using boolean logic, into a single binary adjacency matrix. The resulting matrix, from either method, forms the final clustering graph. Currently syllable count is merely a count of the different clusters (isolates are also syllables) and repetition is the average degree per cluster (isolates get a score of zero repetition).

```
> s <- cluster(s, method='limits', intensity.weighted=FALSE)
> s$syllable_ct
```

```
[1] 9
```

```
> s$repetition
```

```
[1] 0.8571429
```

```
> ## weighted by intensity
> s <- cluster(s, method='limits', intensity.weighted=TRUE)
> s$syllable_ct
```

```
[1] 5.358455
```

```
> s$repetition
```

```
[1] 0.8815676
```

8 Plotting and Graphing Units

The *melody* package also has two convenience functions to plot and graph the processed spectrogram and clustering graph (respectively). The `plot` function simply plots the spectrogram and adds annotations appropriately as the spectrogram object is modified. The `graph` function plots a mathematical graph showing the clustering patterns of the units and grouping different syllable types. Colored unit labels along the bottom of the spectrogram correspond to the colored labels in the clustering/repetition graph.

```
> par(mfrow=c(2,1))
> plot(s)
> graph(s)
```

References

Nietsch A, Niemitz A (1987) The Vocal-Acoustical Repertoire of Free-Ranging Tarsius-Spectrum. International Journal of Primatology 8, p 483.

Butts, Carter T. (2008). "network: a Package for Managing Relational Data in R." Journal of Statistical Software, 24(2).

Carter T. Butts (2010). sna: Tools for Social Network Analysis. R package version 2.2-0. http://CRAN.R-project.org/package=sna

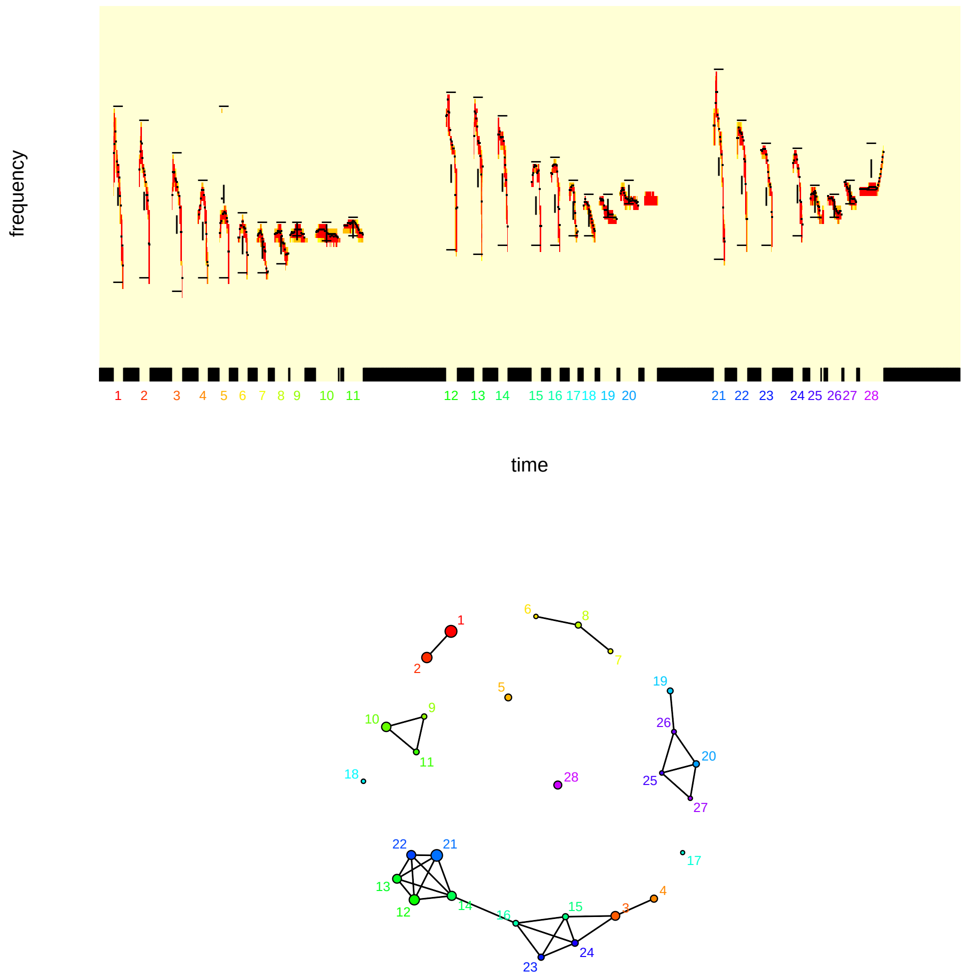

Figure 3: spectrogram and graph