A TIMBRE ANALYSIS AND CLASSIFICATION TOOLKIT FOR PURE DATA

William Brent

University of California, San Diego
Center for Research in Computing and the Arts

ABSTRACT

This paper describes example applications of a timbre analysis and classification toolkit for pure data (Pd). The timbreID collection of Pd externals enable both systematic and casual exploration of sound characteristics via objects that are streamlined and easy to use. Details surrounding signal buffering, blocking, and windowing are performed independently by the objects, so that analyses can be obtained with very little patching. A modular design allows for adaptable configurations and many possible creative ends. The applications described here include vowel classification, target-driven concatenative synthesis, ordering sounds by timbre, and mapping of a sound set in timbre space.

1. INTRODUCTION

Several projects have been developed for the purpose of organizing sounds and/or querying an audio corpus based on timbral similarity. CataRT and Soundspotter are among the most widely recognized open source options [7][3]. The former is available as a Max/MSP implementation, while the latter is intended for multiple platforms—including Pd. Soundspotter's Pd realization is primarily designed for real time target-driven concatenative synthesis. More general tools for creative work centered on timbre similarity are limited in Pd.

timbreID is a Pd external collection developed by the author. It is composed of a group of objects for extracting timbral features, and a classification object that manages the resulting database of information. The objects are designed to be easy to use and adaptable for a number of purposes, including real-time timbre identification, ordering of sounds by timbre, target-driven concatenative synthesis, and plotting of sounds in a user-defined timbre space that can be auditioned interactively. This paper will summarize the most relevant features of the toolkit and describe its use in the four applications listed above.

2. FEATURE EXTRACTION OBJECTS

In general, timbreID's feature extraction objects have four important qualities. First, each object maintains its own signal buffer based on a user-specified window size. This eliminates the need for sub-patches in Pd to set window size using the block~ object. Second, Hann windowing is automatically applied within each object so that input signals do not need to be multiplied against a window table using the tabreceive~ object. Third, analysis timing is sample-accurate. Each object outputs analysis results upon receiving a bang, capturing the desired slice of audio regardless of Pd's default 64-sample block boundaries. Thus, there is no need to set overlap values with block~ in order to define a particular time resolution. Fourth, because the objects perform analysis on a per-request basis, the only computational overhead incurred during periods of analysis inactivity is that of buffering. Combined, these four qualities make signal analysis in Pd straightforward and accessible.

2.1. Available Features

The following external objects for measuring basic features are provided with timbreID: magSpec~, specBrightness~, specCentroid~, specFlatness~, specFlux~, specIrregularity~, specKurtosis~, specRolloff~, specSkewness~, specSpread~, and zeroCrossing~. The more processed features in the set (generated by barkSpec~, cepstrum~, mfcc~, and bfcc~) are generally the most powerful for classification. Mathematical definitions for many of these measurements are given in a previous paper, along with an evaluation of their effectiveness [1]. Detailed information on sound descriptors in general is available elsewhere [8][9]. Although an understanding of the various analysis techniques is not required for use, a general idea of what to expect can be very helpful. To that effect, a simple demonstration and straightforward explanation of each feature is given in its accompanying help file.

In order to facilitate as many types of usage as possible, non real-time versions of all feature externals are provided for analyzing samples directly from graphical arrays in Pd.

2.2. Open-ended analysis strategies

Independent, modular analysis objects allow for flexible analysis strategies. Each of the objects reports its results as either a single number or a list that can be further manipulated in Pd. Feature lists of any size can be packed together so that users can design a custom approach that best suits their

particular sound set. Figure 1 demonstrates how to generate a feature list composed of MFCCs, spectral centroid, and spectral brightness. Subsets of mel-frequency cepstral coefficients (MFCCs) are frequently used for economically representing spectral envelope, while spectral centroid and brightness provide information about the distribution of spectral energy in a signal. Each time the button in the upper right region of the patch is clicked, a multi-feature analysis snapshot composed of these features will be produced.

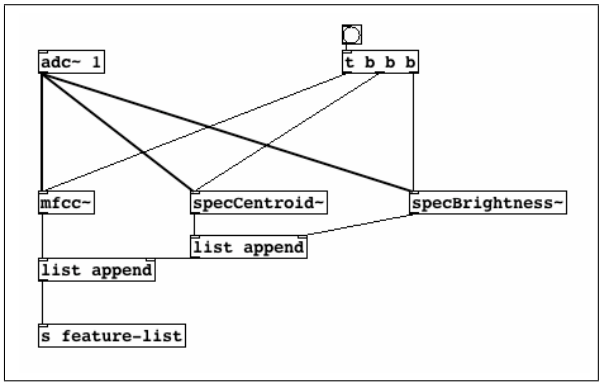

Figure 1. Generating a mixed feature list.

Capturing the temporal evolution of audio features requires some additional logic. In Figure 2, a single feature list is generated based on 5 successive analysis frames, spaced 50 milliseconds apart. The attack of a sound is reported by bonk~ [6], turning on a metro that fires once every 50 ms before turning off after almost a quarter second. Via list prepend, the initial moments of the sound's temporally-evolving MFCCs are accumulated to form a single list. By the time the fifth mel-frequency cepstrum measurement is added, the complete feature list is allowed to pass through a spigot for routing to timbreID, the classification object described below in section 3. Recording changes in MFCCs (or any combination of features) over time provides detailed information for the comparison of complex sounds.

These patches illustrate some key differences from the Pd implementation of libXtract, a well developed multi-platform feature extraction library described in [2]. Extracting features in Pd using the libXtract~ wrapper requires subpatch blocking, Hann windowing, and an understanding of the libXtract's order of operations. For instance, to generate MFCCs, it is necessary to generate magnitude spectrum with a separate object, then chain its output to a separate MFCC object. The advantage of libXtract's cascading architecture is that the spectrum calculation occurs only once, yet two or more features can be generated from the results.

While timbreID objects are wasteful in this sense (each object redundantly calculates its own spectrum), they are more efficient with respect to downtime. As mentioned above, features are not generated constantly, only when needed. Further, from a user's perspective, timbreID objects require

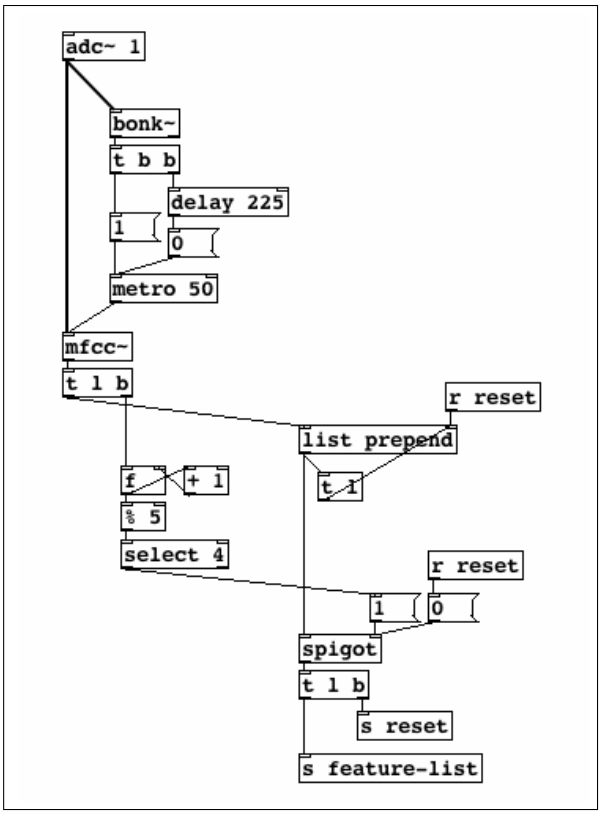

Figure 2. Generating a time-evolving feature list.

less knowledge about analysis techniques, and strip away layers of patching associated with blocking and windowing.

In order to have maximum control over algorithm details, all feature extraction and classification functions were written by the author, and timbreID has no non-standard library dependencies.

3. THE CLASSIFICATION OBJECT

Features generated with the objects described in section 2 can be used directly as control information in real-time performance. In order to extend functionality, however, a multi-purpose classification external is provided as well. This object, timbreID, functions as a storage and routing mechanism that can cluster and order the features it stores in memory, and classify new features relative to its database. Apart from the examples package described in the following section, an in-depth help patch accompanies timbreID, demonstrating how to provide it with training features and classify new sounds based on training. Figure 3 depicts the most basic network required for this task.

Training features go to the first inlet, and features intended for classification go to the second inlet. Suppose the patch in Figure 3 is to be used for percussive instrument classification. In order to train the system, each instrument

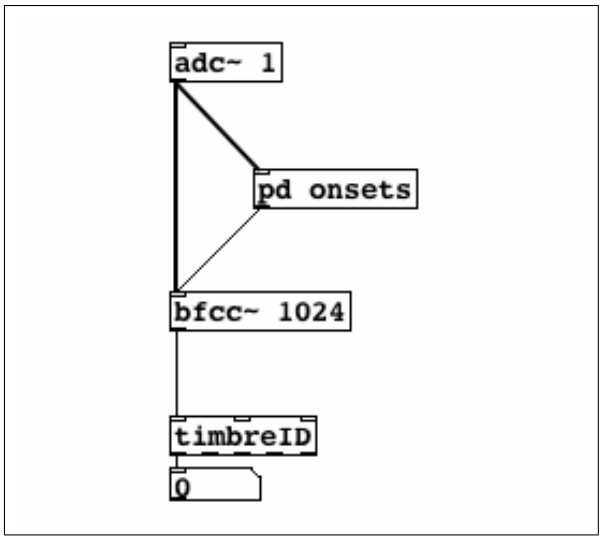

Figure 3. timbreID in a training configuration.

should be struck a few times at different dynamic levels. For each strike, an onset detector like bonk∼ will send a bang message to bfcc∼—the bark-frequency cepstral analysis object. Once a training database has been accumlated in this manner, bfcc∼'s output can be routed to timbreID's second inlet, so that any new instrument onsets will generate a nearest match report from the first outlet. A match result is given as the index of the nearest matching instance as assigned during training. For each match, the second outlet reports the distance between the input feature and its nearest match, and the third outlet produces a confidence measure based on the ratio of the first and second best match distances.

For many sound sets, timbreID's clustering function will automatically group features by instrument. A desired number of clusters corresponding to the number of instruments must be given with the "cluster" message, and an agglomerative hierarchical clustering algorithm will group instances according to current similarity metric settings. Afterward, timbreID will report the associated cluster index of the nearest match in response to classification requests.

Once training is complete, the resulting feature database can be saved to a file for future use. There are four file formats available: timbreID's binary .timid format, a text format for users who wish to inspect the database, ARFF format for use in WEKA[1], and .mat format for use in either MATLAB or GNU octave.

3.1. timbreID settings

Nearest match searches are performed with a k-nearest neighbor strategy, where K can be chosen by the user. Several other settings related to the matching process can also be specified. Four different similarity metrics are available: Euclidean, Manhattan (taxicab), Correlation, and Cosine Similarity. For feature databases composed of mixed features, feature attribute normalization can be activated so that features with large ranges do not inappropriately weight the distance calculation. Specific weights can be dynamically assigned to any attribute in the feature list in order to explore the effects of specific proportions of features during timbre classification or sound set ordering. Alternatively, the feature attributes used in nearest match calculations can be restricted to a specific range or subset. Or, the attribute columns of the feature database can be ordered by variance, so that match calculations will be based on the attributes with the highest variance.

Further aspects of timbreID's functionality are best illustrated in context. The following section describes four of the example patches that accompany the timbreID package.

4. APPLICATIONS

4.1. Vowel recognition

Identification of vowels articulated by a vocalist is a task best accomplished using the cepstrum∼ object. Under the right circumstances, cepstral analysis can achieve a rough deconvolution of two convolved signals. In the case of a sung voiced vowel, glottal impulses at a certain frequency are convolved with a filter corresponding to the shape of the vocalist's oral cavity. Depending on fundamental frequency, the cepstrum of such a signal will produce two distinctly identifiable regions: a compact representation of the filter component at the low end, and higher up, a peak associated with the pitch of the note being sung. The filter region of the cepstrum should hold its shape reasonably steady in spite of pitch changes, making it possible to identify vowels no matter which pitch the vocalist happens to be singing. As pitch moves higher, the cepstral peak actually moves *lower*, as the so-called "quefrency" axis corresponds to period—the inverse of frequency. If the pitch is very high, it will overlap with the region representing the filter component, and destroy the potential for recognizing vowels regardless of pitch[2].

Having acknowledged these limitations, a useful pitch-independent vowel recognition system can nevertheless be arranged using timbreID objects very easily. Figure 4 shows a simplified excerpt of an example patch where cepstral coefficients 2 through 40 are sent to timbreID's training inlet every time the red snapshot button is clicked. Although identical results could be achieved without splitting off a specific portion of the cepstrum[3], pre-processing the feature

[1] WEKA is a popular open source machine learning package described in [4]

[2] These qualities of cepstral analysis can be observed by sending cepstrum∼'s output list to an array and graphing the analysis continuously in real-time.

[3] The alternative would be to pass the entire cepstrum, but set timbreID's active attribute range to use only the 2nd through 40th coefficients in simi-

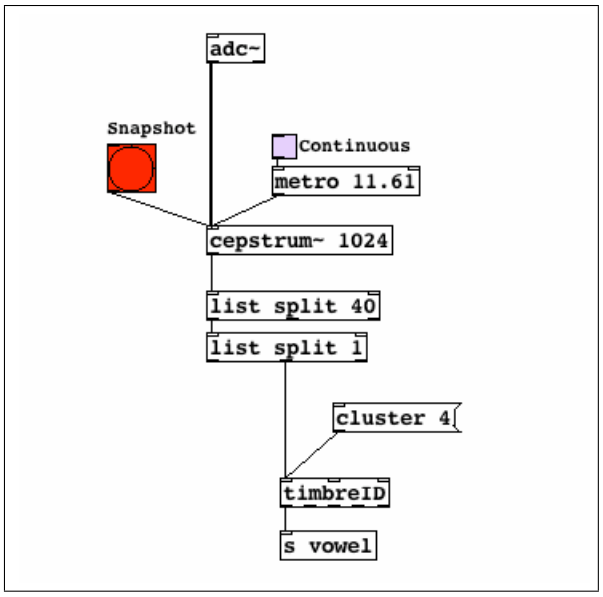

Figure 4. Sending training snapshots and continuous overlapping cepstral analyses to timbreID.

with two instances of Pd's list splitting object keeps timbreID's feature database more compact. The choice of cepstral coefficient range 2 through 40 is somewhat arbitrary, but it is very easy to experiment with different ranges by changing the arguments of the two list split objects.

In order to train the system on 3 vowels, about 5 snapshots must be captured during training examples of each sung vowel. In order to distinguish background noise, 5 additional snapshots should be taken while the vocalist is silent. Next, the "cluster" message is sent with an argument of 4, which automatically groups similar analyses so that the first vowel is represented by cluster 0, the second vowel by cluster 1, and so on. The cluster associated with background noise will end up as cluster 3. It is not necessary to ensure that each vowel receives the same number of analyses. If there were 7 training examples for the first vowel and only 5 for the others, the clustering algorithm should still group the analyses correctly. Clustering results can be verified by sending the "cluster_list" message, which sends a list of any particular cluster's members out of timbreID's fourth outlet.

To switch from training to classification, cepstrum~'s pre-processed output must be connected to timbreID's second inlet. The actual example patch contains a few routing objects to avoid this type of manual re-patching, but they are omitted here for clarity. Activating the metro in Figure 4 enables continuous overlapping analysis. If finer time resolution is desired for even faster response, the metro's rate can be set to a shorter duration. Here, the rate is set to half the duration of the analysis window size in milliseconds, which corresponds to an overlap of 2. As each analysis is passed from cepstrum~ to timbreID, a nearest match is identified and its associated cluster index is sent out timbreID's first outlet. The example patch animates vowel classifications as they occur.

larity calculations.

4.2. Target-based Concatenative Synthesis

Some new challenges arise in the case of comparing a constant stream of input features against a large database in real-time. The feature database in the vowel recognition example only requires about 20 instances. To obtain interesting results from target-based concatenative synthesis, the database must be much larger, with thousands rather than dozens of instances. This type of synthesis can be achieved using the systems mentioned in section 1, and is practiced live by the artist sCrAmBlEd?HaCkZ! using his own software design [5]. The technique is to analyze short, overlapping frames of an input signal, find the most similar sounding audio frame in a pre-analyzed corpus of unrelated audio, and output a stream of the best-matching frames at the same rate and overlap as the input.

The example included with timbreID provides an audio corpus consisting of 5 minutes of bowed string instrument samples. As an audio signal comes in, an attempt at reconstructing the signal using grains from the bowed string corpus is output in real time. Audio examples demonstrating the results can be accessed at www.williambrent.com.

In these types of applications, timbreID's third inlet can be used in order to search large feature databases. Classification requests sent to the third inlet are restricted by a few additional parameters. For instance, the search for a nearest match can be carried out on a specified subset of the database by setting the "search_center" and "neighborhood" parameters.

The concatenative synthesis example provides options for different grain sizes and analysis rates, but with default settings, the process of computing a BFCC feature for the input signal, comparing it with 2500 instances in the feature database, and playing back the best-matching grain occurs at a rate of 43 times per second. Using a 2.91 GHz Intel Core 2 Duo machine running Fedora 11 with 4 GB of RAM, the processor load is about 17%. By lowering the neighborhood setting, this load can be lowered. However, reducing processor load is not the only reason that restricted searches are useful. A performer may also wish to control which region of the audio corpus from which to synthesize.

A third parameter, "reorient" causes search_center to be continually updated to the current best match during active synthesis. With matches occuring 43 times per second, the search range adapts very quickly to changes in the input signal, finding an optimal region of sequential grains from which to draw.

4.3. Timbre ordering

The timbre ordering examples use two different approaches to sound segmentation: the first reads in pre-determined onset/offset times for each of 51 percussion instrument attacks, and the second automatically divides loaded samples into grains that are 4096 samples in length by default. Onset/offset labels for the first example were generated manually in Audacity, exported to a text file, then imported to a table in Pd. The percussive sound set included with this example is small, and is intended to provide a clear demonstration of timbreID's ordering capabilities. Figure 5 shows a region of the patch that includes the table where ordering information is stored and 5 sliders that control feature weighting.

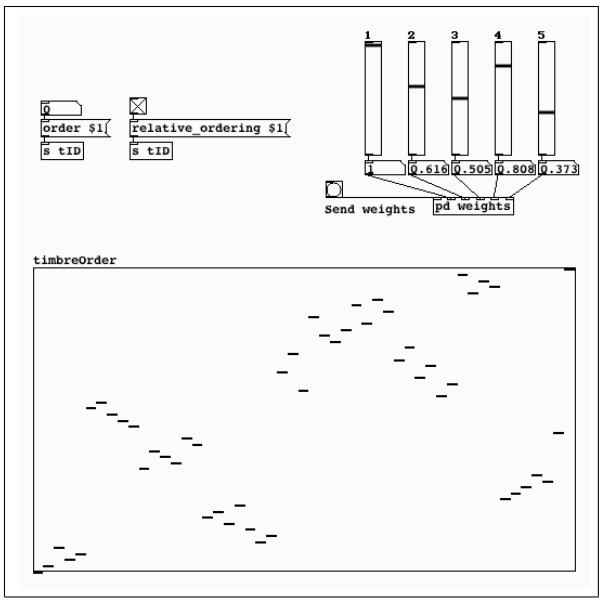

Figure 5. 51 percussion sounds ordered based on a user-specified weighting of 5 features.

Ordering is always performed relative to a user-specified starting point. With 51 instruments, when an instrument index between 0 and 50 is supplied along with the "order" message, timbreID will output the ordering list at its fourth outlet for graphing. Using the 5 feature weight sliders, it is possible to boost or cut back the influence of any particular feature in the ordering process. The features implemented in this patch are temporally evolving spectral centroid, spectral flatness, zero crossing rate, loudness, and BFCCs.

After hearing the results of a particular ordering, the levels of the feature weight sliders can be changed in order to produce a new ordering and gain an understanding of the effects of various features in the process. An ordering is shown in the graph of Figure 5, where the y axis represents instrument indices 0 through 50, and the x axis indicates each instrument's position in the ordering. It begins at instrument 0 with a drum and progresses through other drum strikes followed by snares, a sequence of cymbal strikes, and a sequence of wooden instruments. Ordering the set by starting with a wooden instrument will produce a different result that retains similarly grouped sequences. An expanded version of this patch could be useful as a compositional aid for exploring relationships between sounds in a much larger set, offering paths through the sounds that are smooth with respect to different sonic characteristics.

Two types of ordering are available: "raw" and "relative". The graph in Figure 5 was produced with relative ordering, which starts with the user-specified instrument, finds the nearest match in the set, then finds the nearest match to that match (without replacement), and so on. The point of reference is always shifting. Raw ordering begins with the given instrument, then finds the closest match, the second closest match, the third closest match (also without replacement), and so on. Orderings of this type start with a sequence of very similar sounds that slowly degrades into randomness, and usually finish with a sequence of similar sounds—those that are all roughly equal in distance from the initial sound, and hence, roughly similar to each other.

The second ordering example loads and segments arbitrary sound files. Loading a speech sample generates sequences of similar phonemes with a surprisingly continuous pitch contour. Audio generated from this and other ordering examples can be accessed at the author's website.

4.4. Mapping sounds in timbre space

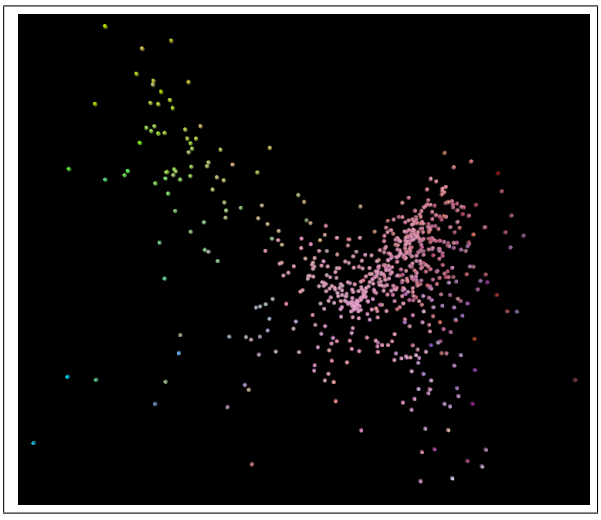

Figure 6. 847 speech grains mapped with respect to the 2nd and 3rd BFCC.

Another way to understand how the components of a sound set relate to one another is to plot them in a user-defined timbre space. CataRT is the most recognized and well developed system for this task; timbreID makes it pos-

sible within Pd using GEM for two- and three-dimensional plotting. In the provided example, the axes of the space can be assigned to a number of different spectral features, zero crossing rate, amplitude, frequency, or any of 47 Bark-frequency cepstral coefficients. By editing the analysis subpatch, additional features can be included. Figure 6 shows the speech grains described in the previous section plotted in a space where values of the second and third BFCCs are mapped to the x and y axes respectively. RGB color can be mapped to any available features as well.

Mousing over a point in the space plays back its appropriate grain, enabling exploration aimed at identifying regions of timbral similarity. The upper left region of figure 6 contains a grouping of "sh" sounds, while the central lower region contains a cluster of "k" and "ch" grains. Other phonemes can be located as well. In order to explore dense regions of the plot, keyboard navigation can be enabled to zoom with respect to either axis (or both simultaneously), and move up, down, left, or right in the space.

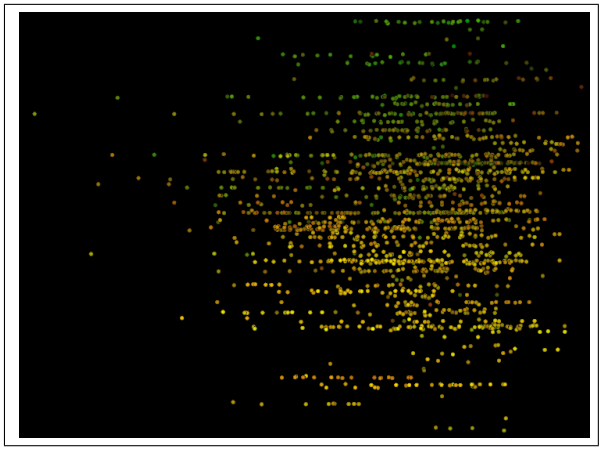

Figure 7. 2400 string grains mapped with respect to amplitude and fundamental frequency.

Figure 7 shows a plot of string sample grains mapped according to RMS amplitude and fundamental frequency. Because the frequencies in this particular sound file fall into discrete pitch classes, its grains are visibly stratified along the vertical dimension.

Mapping is achieved by recovering features from timbreID's database with the "feature_list" message, which is sent with a database index indicating which instance to report. The feature list for the specified instance is then sent out of timbreID's fifth outlet, and used to determine the instance's position in feature space.

5. CONCLUSION

This paper has introduced some important features of the timbreID analysis/classification toolkit for Pd, and demonstrated its adaptability to four unique tasks. Pd external source code, binaries, and the example patches described above are all available for download at the author's website: www.williambrent.com. The remaining patches in the example package—a cepstrogram plotting interface and a percussion classification system that identifies instruments immediately upon attack—were not described. The example patches are simple in some respects and are intended to be starting points that can be expanded upon by the user.

Future development will be focused on adding new features to the set of feature extraction objects, implementing a kD-tree for fast searching of large databases in order to make concatenative synthesis more efficient, and developing strategies for processing multiple-frame features of different lengths in order to compare sounds of various durations.

6. REFERENCES

[1] W. Brent, "Cepstral analysis tools for percussive timbre identification," in *Proceedings of the 3rd International Pure Data Convention*, São Paulo, Brazil, 2009.

[2] J. Bullock, "Libxtract: A lightweight library for audio feature extraction," in *Proceedings of the International Computer Music Conference*, 2007.

[3] M. Casey and M. Grierson, "Soundspotter/remix-tv: fast approximate matching for audio and video performance," in *Proceedings of the International Computer Music Conference*, Copenhagen, Denmark, 2007.

[4] G. Holmes, A. Donkin, and I. Witten, "Weka: a machine learning workbench," in *Proceedings of the second Australia and New Zealand Conference on Intelligent Information Systems*, Brisbane, Australia, 1994, pp. 357–361.

[5] S. König, http://www.popmodernism.org/scrambledhackz.

[6] M. Puckette, T. Apel, and D. Zicarelli, "Real-time audio analysis tools for pd and msp," in *Proceedings of the International Computer Music Conference*, 1998, pp. 109–112.

[7] D. Schwarz, G. Beller, B. Verbrugghe, and S. Britton, "Real-time corpus-based concatenative synthesis with catart," in *Proceedings of the COST-G6 Gonference on Digital Audio Effects (DAFx)*, Montreal, Canada, 2006, pp. 279–282.

[8] G. Tzanetakis and P. Cook, "Musical genre classification of audio signals," *IEEE Transactions on Speech and Audio Processing*, vol. 10, no. 5, pp. 293–302, 2002.

[9] X. Zhang and Z. Ras, "Analysis of sound features for music timbre recognition," in *Proceedings of the IEEE CS International Conference on Multimedia and Ubiquitous Engineering*, 2007, pp. 3–8.