JAUDIO: A FEATURE EXTRACTION LIBRARY

Daniel McEnnis
Faculty of Music
McGill University
Montreal Canada
daniel.mcennis
@mail.mcgill.ca

Cory McKay
Faculty of Music
McGill University
Montreal Canada
cory.mckay
@mail.mcgill.ca

Ichiro Fujinaga
Faculty of Music
McGill University
Montreal Canada
ich
@music.mcgill.ca

Philippe Depalle
Faculty of Music
McGill University
Montreal Canada
depalle
@music.mcgill.ca

ABSTRACT

jAudio is a new framework for feature extraction designed to eliminate the duplication of effort in calculating features from an audio signal. This system meets the needs of MIR researchers by providing a library of analysis algorithms that are suitable for a wide array of MIR tasks. In order to provide these features with a minimal learning curve, the system implements a GUI that makes the process of selecting desired features straight forward. A command-line interface is also provided to manipulate jAudio via scripting. Furthermore, jAudio provides a unique method of handling multidimensional features and a new mechanism for dependency handling to prevent duplicate calculations.

The system takes a sequence of audio files as input. In the GUI, users select the features that they wish to have extracted—letting jAudio take care of all dependency problems—and either execute directly from the GUI or save the settings for batch processing. The output is either an ACE XML file or an ARFF file depending on the user's preference.

Keywords: Java Audio Environment, Audio Feature Extraction, Music Information Retrieval.

1 INTRODUCTION

jAudio is a feature extraction system designed to meet the needs of MIR researchers by providing a collection of feature extraction algorithms bundled with both an easy-to-use GUI and a command-line interface. The system accepts audio files as input and produces either ACE XML files (McKay et al. 2005) or ARFF files. Furthermore, the system includes multidimensional features and a new way to handle dependencies between features.

Extracting high-quality features is of critical importance for many MIR projects (Fujinaga 1998; Jensen 1999). Since these features are the only information that a classifier or other interpretive construct has about the original data, a failure to capture information and patterns inherent in the signal will result in poor performance, no matter how good the interpretive layer is.

One such problem is the difficulty in extracting perceptual features such as meter or pitch from a signal. These features, though useful, are typically not used because they are generally too complicated to create. This is particularly true if the main focus of the research is in other areas, not the creation of new pitch or meter detection algorithms.

Since features are the sole mechanism by which interpretive layers can gain access to the latent information of the original data source, having many different features is desirable. Especially if feature selection or feature weighting is used, having a multitude of features permits the interpretive layer to have as many perspectives on the incoming data as possible. However, especially if the interpretive layer is the main focus of the researchers' efforts, creating and maintaining a large array of features is a significant effort that may not be feasible.

The current state of feature extraction techniques has some additional difficulties. There is no central repository of algorithms dedicated to extracting features. This means that researchers are dependent on often sparse descriptions from conference proceedings to identify the best features to obtain for a given topic. Unfortunately, due to space constraints, these descriptions tend to be terse to the point of obscurity, greatly increasing the chance that an algorithm may be implemented incorrectly due to a misunderstanding of a source. jAudio alleviates this problem by providing a central repository for placing features.

There is also a problem in how the feature extractors communicate with their interpreters. There has been some progress made in this field as Weka's ARFF format has become a de facto standard (Witten and Frank 1999). Yet, with the exception of Marsyas (Tzanetakis and Cook 2000), no existing feature extraction system provides their output data in a standard output format.

More critically, feature extraction source code is either not made available or is tightly coupled to the classification or analysis code. This prevents the reuse of this code in other contexts, limiting the ability of researchers to exchange feature extraction algorithms.

Furthermore, many algorithms for feature extraction

are implemented in a platform dependent way that does not necessarily function properly on computers of another architecture or sometimes on systems of the same architecture but different configuration.

Another concern that needs to be addressed is how easily the feature extraction platform can be extended. A complicated setup means that few features will be added by anyone but the maintainers, drastically limiting the usefulness of the project.

2 RELATED RESEARCH

Efforts to extract a large number of features in a single experiment have been done before. Papers such as Fujinaga (1998), Hong (2000), and Jensen (1999) are known for their large feature collection. However, the creation of libraries to avoid duplication of the effort of writing feature extraction algorithms is a relatively new phenomenon.

2.1 Marsyas

Marsyas by George Tzanetakis is a pioneer in this area. His system is implemented in C++. The system is both efficient and open source. Despite being integrated into a general classification system, Marsyas retains the ability to output feature extraction data to Weka's ARFF format. One drawback is the complicated interface for controlling the features selected for extraction in the extraction subsystem (Tzanetakis and Cook 2000).

2.2 CLAM

CLAM is produced by the Music Technology Group at Pompeu Fabra University (Amatrain et al. 2002). The system is an analysis/synthesis system and is implemented in C++. While a good general system with a good GUI user interface, the system was not intended for extracting features for classification problems.

2.3 M2K

M2K is built upon the D2K data mining software developed at NCSA at the University of Illinois (Downie et al. 2004). The system utilizes the GUI architecture of D2K patches—a very intuitive method for building large hierarchies of feature sets. Unfortunately, the system is currently in an alpha state. Further complicating this difficulty is the commercial license of the underlying D2K system. While M2K is available under a free license, the D2K system is only available for academic use. This makes the system more difficult to obtain by researchers outside the United States and raises the possibility of licensing problems for those researchers whose work with M2K blurs the edge between a research tool and a commercial open-source application.

2.4 Maaate

Maaate is produced by the Commonwealth Scientific and Industrial Research Organization. It was built primarily to extract features from MPEG 1 audio rather than uncompressed audio. While it has a GUI front end, the GUI is geared towards visualization rather than controlling the feature extraction process (Pfeiffer et al. 2005).

3 DESIGN DECISIONS

In order to address the issues introduced in Section 1, it was necessary to make a number of design decisions that shaped jAudio.

3.1 Java based

jAudio was written in Java in order to capitalize on Java's cross-platform portability and design advantages. A custom low-level audio layer was implemented in order to supplement Java's limited core audio support and allow those writing jAudio features to deal directly with arrays of sample values rather than needing to concern themselves directly with low-level issues such as buffering and format conversions.

3.2 XML and Weka output

jAudio supports multiple output formats, including both the native XML format of the ACE (Autonomous Classifier Engine) system, which is a framework for optimizing classifiers (McKay et al. 2005), and the ARFF format used by the popular Weka analysis toolkit (Witten and Frank 1999). This permits utilizing both the ACE environment for classification problems and providing a more established format. Export to ARFF is accomplished by treating all multidimensional features as collections of individual features.

3.3 Handling dependencies

In order to reduce the complexity of calculations, it is often advantageous to reuse the results of an earlier calculation in other modules. jAudio provides a simple way for a feature class to declare which features it requires in order to be calculated. An example is the magnitude spectrum of a signal. It is used by a number of features, but only needs to be calculated once. Just before execution begins, jAudio reorders the execution of feature calculations such that every feature's calculation is executed only after all of its dependencies have been executed. Furthermore, unlike any other system, the user need not know the dependencies of the features selected. Any feature selected for output that has dependencies will automatically and silently calculate dependent features as needed without replication.

3.4 Support for multidimensional features

jAudio has the capacity to accept features that provide an arbitrary number of dimensions. This is an extremely useful way to group related features calculated at once such as MFCC. This is in contrast to the ARFF format from Weka where all features are unidimensional. Furthermore, the dimensionality of each feature is exported. This permits derivatives and other metafeatures to have the same number of dimensions as the feature they are calculated from, even though the same code is used for all features.

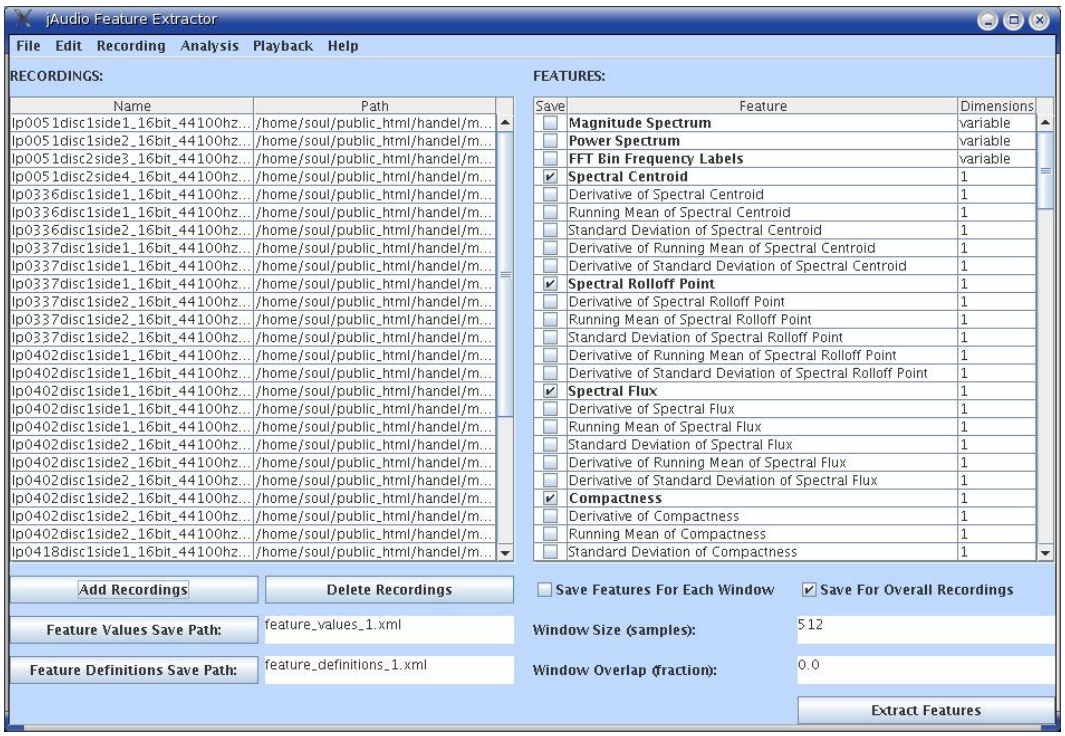

Figure 1: Screenshot of jAudio GUI.

3.5 Intuitive interface

jAudio permits control of downsampling of the input signal, signal normalization, window size, window overlap, and control of which features are extracted and saved with an easy to use GUI (See Figure 1). The GUI also permits users to configure settings and save them for batch processing.

3.6 License

All source code is publicly available on the Internet (http://coltrane.music.mcgill.ca/ACE) under the Lesser GNU Public License (LGPL).

3.7 Extensibility

Effort was taken to make it as easy as possible to add new features and associated documentation to the system. An abstract class is provided that includes all the features needed to implement a feature.

3.8 Metafeatures

Metafeatures are templates that can be applied against any feature to create new features. Examples of metafeatures include Derivative, Mean, and Standard Deviation. Each of these metafeatures are automatically applied to all features without the user needing to explicitly create these derivative features.

4 IMPLEMENTED FEATURES

There are 27 distinct features implemented in jAudio. The following is a non-exhaustive list.

- Zero Crossing

  Zero Crossing is calculated by counting the number of times that the time domain signal crosses zero within a given window. 'Crossing zero' is defined as ($x_{n-1} < 0$ and $x_n > 0$) or ($x_{n-1} > 0$ and $x_n < 0$) or ($x_{n-1} \neq 0$ and $x_n = 0$) .

- RMS

  RMS is calculated on a per window basis. It is defined by the equation:

$$RMS = \sqrt{\frac{\sum_n^N x_n^2}{N}} \quad (1)$$

  where N is the total number of samples provided in the time domain. RMS is used to calculate the amplitude of a window.

- Fraction of Low Amplitude Frames

  This feature is defined as the fraction of previous windows whose RMS is less than the mean RMS. This gives an indication of the variability of the amplitude of windows.

- Spectral Flux

  Spectral Flux is defined as the spectral correlation between adjacent windows (McAdams 1999). It is often used as an indication of the degree of change of the spectrum between windows.

- Spectral Rolloff

  Spectral rolloff is defined as the frequency where 85% of the energy in the spectrum is below this point.

It is often used as an indicator of the skew of the frequencies present in a window.

- Compactness

  Compactness is closely related to Spectral Smoothness as defined by McAdams (1999). The difference is that instead of summing over partials, compactness sums over frequency bins of an FFT. This provides an indication of the noisiness of the signal.

- Method of Moments

  This feature consists of the first five statistical moments of the spectrograph. This includes the area (zeroth order), mean (first order), Power Spectrum Density (second order), Spectral Skew (third order), and Spectral Kurtosis (fourth order). These features describe the shape of the spectrograph of a given window (Fujinaga 1997).

- 2D Method of Moments

  This feature treats a series of frames of spectral data as a two dimensional image which are then analyzed using two-dimensional method of moments (Fujinaga 1997). This gives a description of the spectrograph, including its changes, over a relatively short time frame.

- MFCC

  Mel-Frequency Cepstral Coefficients (MFCCs) are calculated according to the formula by Bogert et al. (1963). The calculations are implemented here using the code taken from the Orange Cow voice recognition project (Su et al. 2005). This is useful for describing a spectrum window.

- Beat Histogram

  This feature autocorrelates the RMS for each bin in order to construct a histogram representing rhythmic regularities. This is used as a base feature for determining best tempo match (Scheirer and Slaney 1997).

5 CONCLUSIONS

jAudio provides a comprehensive solution to the problem of the duplication of work in programming feature extraction. This system permits general use of a large number of features in a fashion that is both easy to use and extensible. The GUI permits the system to be easily configured with minimal effort and the command-line interface permits easy batch processing. The system also provides a central repository for the storing of feature algorithms with an unambiguous meaning with output that can be read by either ACE or Weka.

6 FUTURE WORK

The set of features provided by jAudio is by no means comprehensive. Numerous additional features remain to be added. In particular, the system needs an implementation of LPC and the ability to process multiple window sizes concurrently.

Also, if the license issues can be resolved, we would like to merge our development efforts into the M2K project. This would allow the project to take advantage of the extensive GUI support while maintaining the existing benefits of jAudio.

REFERENCES

X. Amatrain, P. Arumi, and M. Ramirez. Clam: Yet another library for audio and music processing? In *Proceedings of the ACM Conference on Object Oriented Programming, Systems, and Applications*, 2002. 22–3.

B. Bogert, M. Healy, and M. Healy. The quefrency alanysis of time series for echoes: cepstrum, pseudo-autocovariance, cross-cepstrum, and saphe-cracking. In *Proceedings of the Symposium Time Series Analysis*, 1963. 209–43.

S. Downie, J. Futrelle, and D. Tcheng. The international music information retrieval systems evaluation laboratory. *International Conference on Music Information Retrieval*, 2004. 9–14.

I. Fujinaga. Machine recognition of timbre using steady-state tone of acoustic musical instruments. In *Proceedings of the International Computer Music Conference*, 1998. 207–10.

I. Fujinaga. *Adaptive optical music recognition*. PhD thesis, McGill University, 1997.

T. Hong. Salient feature extraction of musical instrument signals. Master's thesis, Dartmouth College, 2000.

K. Jensen. *Timbre models of musical sounds*. PhD thesis, Kobenhavens Universitet, 1999.

S. McAdams. Perspectives on the contribution of timbre to musical structure. *Computer Music Journal*, 23:85–102, 1999.

C. McKay, D. McEnnis, R. Fiebrink, and I. Fujinaga. Ace: A framework for optomizing music classification. *International Conference on Music Information Retrieval*, 2005.

S. Pfeiffer, C. Parker, and T. Vincent. Maate, 2005. URL `http://www.cmis.csiro.au/maaate/`. [Accessed April 14, 2005].

E. Scheirer and M. Slaney. Construction and evaluation of a robust multi-feature speech/music discriminator. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, 1997.

C. Su, K. Fung, and A. Leonov. Oc volume, 2005. URL `http://ocvolume.sourceforge.net/`. [Accessed April 14, 2005].

G. Tzanetakis and P. Cook. Marsyas: A framework for audio analysis. *Organized Sound*, 10:293–302, 2000.

I. Witten and E. Frank. *Data mining: Practical machine learning tools and techniques with Java implementations*. San Fransisco: Morgan Kaufmann, 1999.