jAudio: Towards a standardized extensible audio music feature extraction system

Cory McKay
Faculty of Music, McGill University
555 Sherbrooke Street West
Montreal, Quebec, Canada H3A 1E3
cory.mckay@mail.mcgill.ca

ABSTRACT

Audio feature extraction play an essential role in automatic music classification. This paper explains the needed for a standardized audio feature extraction system, describes the most important attributes that such a system should possess and presents a prototype that has been developed to meet this need.

1.0 INTRODUCTION

Features, or characteristic pieces of information that can be used to describe objects or abstractions, play an essential role in any classification task Features are the percepts to classification system, and even a perfect classifier will not be able to make correct classifications if it does not have access to features that properly segment the objects to be classified.

Features can be particularly problematic with respect to audio signals. When asked how to distinguish between two different types of sound, a typical person might have a difficulty expressing a precise list of features, even if he or she can easily make the classification. Even in cases where one is able to vocalize audio features, the features are likely to be abstractions that are difficult to quantify and extract digitally from an audio signal. For example, a musician as kedto describe the differences between two genres of music is likely to use terms that require the ability to extract information based on pitches, rhythms and timbres.

Unfortunately, high-level information such as this is currently difficult to impossible to reliably extract from music signals in general. This difficulty means that one must at least start with low-level signal processing-based features. Determining which features are best-suited for a particular task can be difficult, as humans do not tend to perceive or think about sound in terms that are meaningful in a low-level signal processing sense.

Fortunately, there are successful approaches to dealing with these problems. Although low-level features are not usually intuitive to humans in a perceptual sense, an individual well-trained in signal processing and in auditory perception can use his or her expertise to gain insights into how certain low-level features can be useful.

Furthermore, one can take an iterative approach to feature extraction, where low-level features derived directly from audio signals are used to derive mid-level representations, which can in turn be used to derive high-level features that are perceptually meaningful to humans. For example, basic spectral features can currently be used to track note onsets and pitch in the special case of monophonic music, which can in turn be used to generate MIDI transcriptions, which can then be used to generate high-level features related to rhythmic patterns and melodies.

An additional important point is that, even though it may be the case that a particular low-level feature is not used by humans to perform a certain classification, this does not necessarily mean that this feature cannot effectively be used by a computer to perform the same classification. Feature selection techniques can be used to experimentally and statistically determine which features are useful and which are not in a given context.

Such an explorative approach is particularly important with respect to music classification. Although a great deal of research has been done on features that could be useful for speech recognition, music has received much less attention. Experimental research on which features work well with respect to different kinds of music in different contexts could be of theoretical interest in and of itself, in addition to improvements in

classification success rates that it might bring about.

In any case, it is clear that the acquisition of a large number of low-level features can be very useful, for the purpose of direct classification as well as for constructing more abstract higher-level features. Software that can easily do this is therefore essential for audio classification.

It is currently common in the field of music information retrieval (MIR) for individual research groups to each design their own feature extraction systems. Aside from the obvious inefficiency involved in this duplication of effort, this can also lead to inconsistencies and inaccuracies in extracted features. MIR researchers in general require expertise in both machine learning and music, with the result that it is not surprising that many of them have only a passing knowledge of signal processing. Time constraints related to the fact that a given feature extraction system is only part of a larger system that is being developed also makes it likely that the system will be less rigorously designed and tested than a specialized system might be. For all of these reasons, a good standardized audio feature extraction system could be of great benefit to the MIR community.

2.0 EXISTING SYSTEMS

A number of feature extractionsystems have been made publicly available over the last several years in order to address this need. Three of these in particular deserve special mention here, as they represent perhaps the most serious efforts towards creating reliable and widely applicable systems.

2.1 Marsyas

Marsyas has been an important pioneer in the MIR field, and is probably currently the most widely used feature extraction system. It is well maintained by its creator, George Tzanetakis, who is in the process of finalizing a new release. The software has been discussed in a number of publications, perhaps most famously with respect to genre classification (Tzanetakis, Essl, and Cook. 2001; Tzanetakis, G., and P. Cook. 2002).

The software is efficiently implemented in C++, but the code can be somewhat difficult for novice users to understand and extend, and the documentation on the details of the system can be limited. There have also been some problems with installation and ease of use in the past, although these may be corrected in the newest release.

2.2 CLAM

CLAM is a powerful and flexible audio analysis, synthesis and transformation system implemented in C++ that is currently in the final stages of development. Although the groundwork has clearly been laid for a powerful and easy-to-use interface, the current beta version still has some problems. In addition, installation can be particularly problematic. Furthermore, CLAM is not specifically designed for feature extraction, and its general power and flexibility complicate the system for those concerned solely with feature extraction.

So, although CLAM promises to be an excellent system, it is only tangentially suitable for a specialized feature extraction role, particularly in its current stage of development. More information on CLAM can be found in the soon to be published paper by Pau Arumi and Xavier Amatriain (2005).

2.3 M2K

M2K is a patch-based system implemented in Java that is currently in alpha development. This system promises to be a very powerful prototyping tool in the domain of MIR, for which it has been specifically designed. Downie, Futrelle and Tcheng have published the initial work on the system (2004).

M2K makes use of the D2K distributed processing framework. This means that M2K can process tasks simultaneously using several computers and can also take advantage of D2K's classification libraries and powerful GUI.

Unfortunately, D2K's licence not only sometimes makes it complicated for researchers outside the U.S.A. to obtain it, but forbids its use in commercial applications. This means that any system that uses D2K cannot itself be used for any non-research-based tasks. In addition, D2K itself still has a number of bugs and it can also be difficult to install. Furthermore, not all of D2K is open-source.

So, although M2K promises to be an excellent tool for MIR prototyping, its licensing situation and current early stage of development makes it inappropriate for developing a generally usable standardized feature extraction system.

3.0 DESIGN PRIORITIES

It can be seen from Section 2 that there are a number of powerful and useful tools that can be used for audio feature extraction. Unfortunately, none of them are without weaknesses with respect to the particular role of a standardized and general-purpose feature extractor.

This leaves a gap which needs to be filled. The system described in this paper is intended as a preliminary step towards filling this gap. Before describing the details of the system, however, it is important to explicitly state the qualities that are most essential in order for a general audio feature extraction system to be successful.

3.1 Extensibility

It must be easy to add new features to the system. A system must allow one to experiment with new features, not just extract existing ones.

In particular, it should be simple for new features to make use of existing features, as many features build upon one another. For example, extracting the RMS of windows allows one to perform an auto-correlation which can produce a beat histogram, which in turn can be used to derive information such as tempo or meter.

It is therefore essential that one be able to easily add new features that potentially take advantage of existing features. It should be possible to do this with only minimal knowledge of the design of the system as a whole. This implies a modular and well documented design. Furthermore, the software should be open source so that individual developers can expand on it and others can take advantage of their work.

3.2 Ease of use

It is essential that the software be easy to learn and use. Potential users could have a wide variety of backgrounds in both signal processing and compute science, and this variety must be taken into account. The details of signal processing should be hidden from users whenever possible, in order to accommodate those with little experience. Feature interdependencies in particular should be made invisible to users. At the same time, it must be possible for users skilled in signal processing to be able to access signal processing details so that they can take advantage of their expertise.

A dual implementation of GUI and command-line interfaces represents a good solution, as novice users can perform powerful tasks in a simple fashion using the GUI, and skilled users can perform specialized batch tasks using the command line. In either case, the interface(s) must be extensively and clearly documented.

3.3 Portability and ease of installation

If a system is to be widely usable, it must be possible to install it under any common operating system. It should also not require any proprietary commercial software, such as MatLab, or any software with a limited licence, such as D2K. The latter concern is particularly important for users who might wish to integrate the feature extraction system into larger systems.

Given these requirements, it is also very important that the software be effortless to install. This means that it might be wise to implement such a system in a portable language such as Java. Languages such as C++ should be avoided unless specialized and complete installation subsystems are made available for each potential operating system, as compilation and linking difficulties can discourage many users.

3.4 Interaction with classifiers

It is, of course, necessary that the feature extraction system be able to communicate extracted features to a classification system. The feature extraction system should be independent of any particular classification approach or software on, as too tight an integration compromises the generality and portability of the system. On the other hand, the system must be able to output features in formats that common classifiers are able to read.

There is, unfortunately, no standardized file format for storing features. The development of such a format would be of great use, as a single output file could then be used with arbitrary classifiers. Of course, this would require that the standardized format be recognized by existing classification systems, which could take some time to come about. So, although the development of a standardized format should be pursued, existing formats should not be neglected.

3.5 Variety of features

As mentioned above, a large library of features can be processed by feature selection algo-

rithms in order to choose the best features for a particular task. This makes the implementation of a wide variety of features valuable, as it makes more candidates available for feature selection. It also makes more basis features available which can be used to derive other features in the future.

Of course, it is important that the features be well tested in order to assure their accuracy and consistency with other feature extraction systems. Consistency can sometimes be difficult to achieve, unfortunately, as there are sometimes multiple interpretations of the same feature. For example, many spectral features contain versions based on magnitude spectra derived from Fourier analysis as well as versions based on power spectra.

3.6 User control of pre-processing and feature parameters

Users will wish to use features for many different tasks under many different conditions. The user must therefore be given control over the parameters of individual features, where appropriate, as well as over any pre-processing that is to occur. Possible parameters include downsampling, if any, window size and amount of window overlap.

Some users will wish to classify small segments of audio signals individually, and others will only wish to classify recordings or other large portions of a signal as a whole. A good system should therefore be able to extract and save features for individual windows as well as for sets of windows. Averages and standard deviations of window features could be particularly useful in the latter case.

4.0 IMPLEMENTED SYSTEM

The system implemented here, named jAudio, was designed with the concerns expressed in Section 3 in mind. Although jAudio cannot yet be said to perfectly meet all of these requirements, it does address many of them. The version of jAudio discussed here is a prototype that is intended to eventually be developed into a complete system that meets all of Section 3's requirements. This is not to say that jAudio is not fully functional, however, as the prototype does work fully and reliably.

4.1 Basic implementation

The system is implemented in Java, partly in order to take advantage of Java's portability and partly because Java's design facilitates an extensible implementation. Although there were some initial concerns about speed, it turns out that Java in its current implementation is faster than is commonly believed, and processing time was not more of an issue than it is with audio processing systems in general.

The disadvantage of using Java is that there is only limited support for audio in the core Java libraries. This is perhaps the reason that there was no previously implemented Java-based audio feature extraction system of any scale. It was therefore necessary to do a good deal of low-level work simply to read and process audio at all. Now that that work has been done, however, the jAudio system makes the level of audio involving signal streams, buffering, etc. invisible to future developers who wish to implement new features.

The software is entirely open source and freely distributable. Full Javadoc documentation is available on all classes. Installing the software consists only of copying two Jar files into one's Java extensions directory and then copying the jAudio Jar file into whatever home directory one desires. The software can then be run simply by double clicking on the jAudio Jar.

The software makes use of two pieces of third-party software, both of which are freely distributable. A Tritonus plug-in is used to perform sampling rate conversions and the Xerces XML file parser is used to process XML files.

4.2 Adding new features

One of the most important strengths of jAudio is the ease with which the design makes it possible to implement new features. If one wishes to do so, one need only write a single new class that extends an abstract class named FeatureExtractor. All that this new class must contain is a constructor and an implementation of a FeatureExtractor method called extractFeature.

This design makes it very easy to make use of other features in one's new feature, without requireing any knowledge of how they themselves are implemented. The constructor of the new class can contain an array of strings and an array of integers. Each string can represent the name of any other feature and each integer repre-

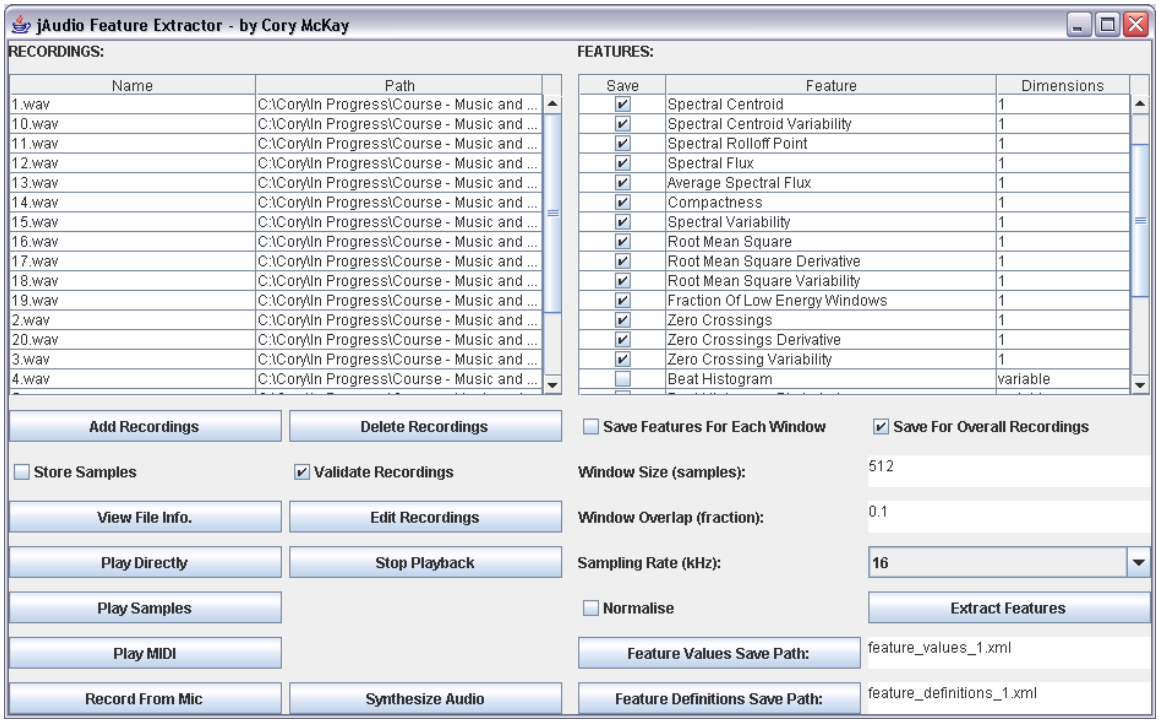

Figure 1: Basic jAudio interface.

sents the delay for the given string (e.g. 0 for the feature value from the current window, -1 for the feature value for one window previous to the current window, etc.). The extractFeature method will automatically be fed the requested feature values as doubles during execution, as well as an array of doubles representing the current window of samples and the sampling rate. No knowledge of how this is done is needed to implement a new feature.

This ability to implement feature interdependencies without any knowledge of how jAudio itself works or how any of the other features work is very valuable from the perspective of extensibility. jAudio automatically and invisibly schedules the order of feature extraction in order to ensure that all features that can be extracted are extracted and that no feature needs to be extracted more than once for a given window. This has powerful implications relating to efficiency as well as extensibility.

4.3 File formats

jAudio uses two novel file formats that are proposed as candidates for representing extracted features in a standardized format. Both of these files are based on XML, which makes them human readable and easy to debug.

One file contains feature definitions that explain the details of each feature and the other contains the actual feature values. The advantage of this separation is that the meta-data contained in the feature definitions can be accessed by classification systems, which therefore do not need any other connection with the feature extraction system. This also makes it possible to reuse the same definitions with multiple data sets that have the same features extracted from them.

These file formats have two important advantages over traditional ways in which feature values are stored. Firstly, a logical relationship is maintained between the components of multi-dimensional features, unlike most feature extraction systems, which tend to save each component of a multi-dimensional feature as a separate and unrelated value.

Secondly, the relationship between an overall audio recording or stream and its windows is maintained. Most existing systems tend to store each window as separate and independent instance.

This format also makes it possible to store certain features for overall recordings as well as separate features for each window.

These file formats are both used by the ACE classification system (McKay et al. 2005). Soft-

ware is also available to convert these XML files to the ARFF format used by the often-used and powerful Weka data mining framework (Witten & Frank 2000).

4.4 Interface

A simple and intuitive GUI is implemented (see Figure 1). The user must only select the files that s/he wishes to extract features from, check the desired features and presses the *Extract Features* button in order to extract and save features.

As mentioned above, jAudio automatically manages feature dependencies. This means that the user needs no knowledge of these dependencies in order to use the system. For example, if a user is interested in saving the zero-crossing variability but not the zero-crossing rate itself, s/he need only check the zero-crossing variability feature. The system will then automatically determine that the zero-crossing rate is needed to find the zero-crossing variability, extract it and use it in zero-crossing variability calculations, but not save it unless the zero-crossing rate box is checked.

This enables users with little knowledge of signal processing to still extract and save whatever feature values they want, as there is no need to manually match the inputs of any features to the outputs of the features that they need.

4.5 Pre-processing

Several basic pre-processing steps can be performed before feature extraction is commenced:

- **Sampling Rate Conversion:** All recordings can be downsampled (or upsampled, as the case may be) to a user-set sampling rate. This is done for two reasons. Firstly, the sampling rate affects the values of some features, so recordings with different sampling rates would have this reflected in their feature values. Secondly, much of the useful information in a recording is often at relatively low frequencies. Downsampling reduces the time needed to extract features and potentially (depending on window size) the amount of data that needs to be stored. Downsampling throws away relatively little useful information with respect to classification, and can improve classification accuracy by highlighting the particularly pertinent part of the spectrum.
- **Normalisation:** The option is available to normalise recordings based on dynamics. This could be very useful in some cases, such as when audio has been recorded at a variety of gains independent of class, and one does not wish this to affect feature values. Normalising can be undesirable in other cases, however, as differences in overall loudness may certainly be relevant to certain types of classification.
- **Channel Merging:** Stereo recordings are collapsed into mono in order to have a single set of features for each recording.
- **Windowing:** All recordings are divided into windows of a user-definable size. The user has the option of having overlapping windows, with a specifiable overlap.

The user may also choose whether to save the feature values of individual windows, the averages and standard deviations of these values over each recording or both.

4.6 Additional functionality

Extra functionality is provided in addition to basic feature extraction. This includes converting between different audio file formats, transcribing MIDI recordings to audio, recording audio from a microphone, playing audio, performing basic additive synthesis, viewing details about encoding schemes for different audio files and viewing plots of waveforms and Fourier analysis results. This functionality greatly facilitates the testing of new features.

5.0 FEATURES IMPLEMENTED

A good deal of research has been done on extracting features for the purpose of speech processing. Although the features that are the best for this task are not necessarily also the best for music classification, they do provide a good starting point. Scheirer and Slaney (1997) and Cary, Parris and Lloyd-Thomas (1999) have published useful sets of features with respect to speech/music discrimination. George Tzanetakis and his collaborators have used many of these features and expanded upon them with the particular needs of music in mind (Tzanetaks, Essl & Cook 2001; Tzanetakis & Cook 2002).

Researchers in musical instrument identification have made some important contributions. This includes the work of Kashino and Murase

(1997), Martin and Yim (1998), Eronen (2001) and Essed, Richard and David (2004).

Park (2000) has also done some very interesting work on feature extraction with respect to music, as have Fujinaga (1998), Kotek (1998), Jensen (1999), Herrera, Peeters and Dubnov (2003) and West and Cox (2004). Pope, Holm and Kouznetsov (2004) have published some good guidelines for when it is appropriate to use certain types of features.

The jAudio system currently has a total of 26 features implemented. These 26 features may be extracted for individual windows, and the averages and standard deviations of each of these features may be calculated for each recording as a whole.

There is not enough space here to describe each of the 26 features in detail. These features are well documented in the general literature, and precise and well-documented implementation details are available in the source code for each feature (see the AudioFeatures directory).

- **Average Spectral Flux:** The mean spectral flux over the last 100 windows.
- **Beat Histogram:** A histogram showing the relative strength of different rhythmic periodicities (tempi) in a signal. Found by calculating the auto-correlation of the RMS.
- **Beat Histogram Bin Labels:** The bin labels, in beats per minute, of each beat histogram bin. Not useful as a feature in itself, but useful for calculating other features from the beat histogram.
- **Beat Sum:** The sum of all bins in the beat histogram. This is a good measure of the importance of regular beats in a signal.
- **Compactness:** A measure of the noisiness of a recording. Found by comparing the components of a window's magnitude spectrum with the magnitude spectrum of its neighbouring windows.
- **FFT Bin Frequency Labels:** The bin label, in Hz, of each power spectrum or magnitude spectrum bin. Not useful as a feature in itself, but useful for calculating other features from the magnitude spectrum and power spectrum.
- **Fraction Of Low Energy Frames**: The fraction of the last 100 windows that has an RMS less than the mean RMS of the last 100 windows. This can indicate how much of a signal section is quiet relative to the rest of the signal section.
- **Magnitude Spectrum:** A measure of the strength of different frequency components. Derived directly from the FFT.
- **Power Spectrum:** A measure of the power of different frequency components. Derived directly from the FFT.
- **Root Mean Square (RMS):** A measure of the power of a signal over a window.
- **Root Mean Square Derivative:** The window to window change in RMS. An indication of change in signal power.
- **Root Mean Square Variability:** The standard deviation of the RMS of the last 100 windows.
- **Spectral Centroid:** The centre of mass of the power spectrum.
- **Spectral Centroid Variability:** The standard deviation of the spectral centroid over the last 100 windows.
- **Spectral Flux:** A measure of the amount of spectral change in a signal. Found by calculating the change in the magnitude spectrum from frame to frame.
- **Spectral Rolloff Point:** The fraction of bins in the power spectrum at which 85% of the power is at lower frequencies. This is a measure the right-skewedness of the power spectrum.
- **Spectral Variability:** The standard deviation of the magnitude spectrum. A measure of how varied the magnitude spectrum of a signal is.
- **Strength Of Strongest Beat:** How strong the strongest beat in the beat histogram is compared to other potential beats.
- **Strongest Beat:** The strongest beat in a signal, in beats per minute, found by finding the highest bin in the beat histogram.
- **Strongest Frequency Variability:** The standard deviation of the frequency of the power spectrum bin with the highest power over the last 100 windows.
- **Strongest Frequency Via FFT Maximum:** An estimate of the strongest frequency component of a signal, in Hz, found via finding the FFT bin with the highest power.
- **Strongest Frequency Via Spectral Centroid:** An estimate of the strongest fre-

quency component of a signal, in Hz, found via the spectral centroid.

- **Strongest Frequency Via Zero Crossings:** An estimate of the strongest frequency component of a signal, in Hz, found via the number of zero-crossings.
- **Zero Crossings:** The number of times the waveform changed sign in a window. An indication of frequency as well as noisiness.
- **Zero Crossings Derivative:** The absolute value of the window to window change in zero crossings. An indication of change of frequency as well as noisiness.
- **Zero Crossing Variability:** The standard deviation of the zero-crossings of the last 100 windows.

Several of the features described above extract standard deviations and averages over a short number of preceding windows. Although these features are redundant with respect to features calculated for recordings as a whole, they are useful for window-based classification, as they provide data on local history to classifiers classifying individual windows.

6.0 CONCLUSIONS

This paper has emphasized the need for a standardized audio feature extraction system for audio classification. An overview was presented of important qualities that such a system should possess, and the jAudio prototype system was presented.

7.0 FUTURE RESEARCH

There are a number of improvements that remain to be made to the jAudio system. First and foremost, more features need to be implemented, perhaps most urgently MFCC and LPC-based features. A command line based interface also remains to be made available. An on-line help system would prove helpful. The addition of basic filters for use in pre-processing as well as the ability to parse MP3 and SDIF files should be added. It could also be useful to improve the system so that it can extract features from live audio streams as well as saved files. Porting a version of jAudio to M2K could also be profitable, given M2K's potential and the speed advantages of distributed computing.

8.0 ACKNOWLEDGEMENTS

This system was designed and implemented under the supervision of Prof. Philippe Depalle. The idea of automatic scheduling of feature dependencies was proposed by Daniel McEnnis, who has also implemented a feature extraction system which will soon be merged with jAudio. The graphing and additive synthesis implementations were inspired by the excellent Java Sound on-line tutorials of Richard Baldwin.

9.0 BIBLIOGRAPHY

Arumi, P., and X. Amatriain. 2005. CLAM, an object oriented framework for audio and music. Accepted for publication at the *International Linux Audio Conference.*

Carey, M. J., E. S. Parris, and H. Lloyd-Thomas. 1999. A comparison of features for speech, music discrimination. *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*. 149–152.

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

Eronen, A. 2001. Comparison of features for musical instrument recognition. *Proceedings of the IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*. 753–6.

Essed, S., G. Richard, and B. David. 2004. Musical instrument recognition based on class pairwise feature selection. *Proceedings of the International Conference on Music Information Retrieval*. 560–8.

Fujinaga, I. 1998. Machine recognition of timbre using steady-state tone of acoustic musical instruments. *Proceedings of the International Computer Music Conference.* 207-210.

Herrera, P., G. Peeters, S. Dubnov. 2003. Automatic classification of musical instrument sounds. *Journal of New Music Research* 32(1): 3–21.

Jensen, K.. 1999. Timbre models of musical sounds. *Ph.D. dissertation*. Kobenhavns Universitet.

Kashino, K., and Murase, H. 1997. Sound source identification for ensemble music based on the music stream extraction. *Proceedings of the*

International Joint Conference on Artificial Intelligence. 1126–31.

Kotek. B. 1998. Soft computing-based recognition of musical sounds. In *Rough Sets in Knowledge Discovery*, eds. L. Polkowski and A. Skowron. Heidelberg: Physica-Verlag.

Martin, K., and Y. Kim. 1998. Musical instrument identification: A pattern recognition approach. *Proceedings of the Acoustical Society of America*.

McKay, C., R. Fiebrink, D. McEnnis, B. Li, and I. Fujinaga. 2005. ACE: A framework for optimizing music classification. Submitted to the *International Conference on Music Information Retrieval.*

Park, T. H. 2000. Salient feature extraction of musical instrument signals. *Master's thesis*. Dartmouth College, USA.

Pope, S. T., F. Holm, and A. Kouznetsov. 2004. Feature extraction and database design for music software. *Proceedings of the International Computer Music Conference*. 596–603.

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

Tzanetakis, G., G. Essl, and P. Cook. 2001. Automatic musical genre classification of audio signals. *Proceedings of the International Symposium on Music Information Retrieval*. 205–10.

Tzanetakis, G., and P. Cook. 2002. Musical genre classification of audio signals. *IEEE Transactions on Speech and Audio Processing* 10 (5): 293–302.

West, C., and S. Cox. 2004. Features and classifiers for the automatic classification of musical audio signals. *Proceedings of the International Conference on Music Information Retrieval*. 531-7.

Witten, I., and E. Frank. 2000. *Data mining: Practical machine learning tools and techniques with Java implementations*. San Francisco: Morgan Kaufmann Publishers.