ADDITIONS AND IMPROVEMENTS TO THE ACE 2.0 MUSIC CLASSIFIER

Jessica Thompson
Music Technology
McGill University
jessica.thompson@mail.mcgill.ca

Cory McKay
CIRMMT
McGill University
cory.mckay@mail.mcgill.ca

John Ashley Burgoyne
CIRMMT
McGill University
ashley@music.mcgill.ca

Ichiro Fujinaga
CIRMMT
McGill University
ich@music.mcgill.ca

ABSTRACT

This paper presents additions and improvements to the Autonomous Classification Engine (ACE), a framework for using and optimizing classifiers. Given a set of feature values, ACE experiments with a variety of classifiers, classifier parameters, classifier ensembles and dimensionality-reduction techniques in order to arrive at a configuration that is well-suited to a given problem. Changes and additions have been made to ACE in order to increase its functionality as well as to make it easier to use and incorporate into other software frameworks. Details are provided on ACE's remodeled class structure and associated API, the improved command line and graphical user interfaces, a new ACE XML 2.0 ZIP file format and expanded statistical reporting associated with cross validation. The resulting improved processing and methods of operation are also discussed.

1. INTRODUCTION

Automatic classification techniques play an essential role in many music information retrieval (MIR) research areas. These include genre classification, mood classification, music recommendation, performer identification, composer identification, and instrument identification, to name just a few. Classification software especially adapted to MIR can be of significant benefit. Some important work has already been done in this area, as noted in Section 2. ACE, which is part of the jMIR software suite described below, is a framework that builds upon these systems and adds additional functionality that is generally lacking in both systems designed specifically for music classification as well as general classification systems. ACE 2.0, which is presented in this paper, has been significantly improved since the release in 2005 of ACE 1.1 [1]. Improvements include an entirely restructured and simplified API, a significantly improved command-line interface, a new GUI, new file formats, improved processing and significantly expanded statistical reporting.

1.1 jMIR

jMIR [2] is a suite of software tools developed for use in MIR research. The jMIR components can be used either independently or as an integrated suite. Although the components can read and write to common file formats such as Weka ARFF, jMIR also uses its own ACE XML file formats that offer a number of advantages over alternative data-mining formats [1, 3].

jMIR was designed to provide:

- a flexible set of tools that can easily be applied to a wide variety of MIR-oriented research tasks;
- a platform that can be used to combine research on symbolic, audio and/or cultural data;
- easy-to-use and accessible software with a minimal learning curve that can be used by researchers with little or no technological training;
- a modular and extensible framework for iteratively developing and sharing new feature extraction and classification technologies; and
- software that encourages collaboration between different research centers by facilitating the sharing of research data using powerful and flexible file formats [4].

jMIR is the only existing software suite that combines a meta-learning component (ACE) into an integrated framework with three different types of musical feature extractors, a metadata correction tool, and ground truth data. jMIR is also the only unified MIR research framework that combines all three of symbolic, audio, and cultural features.

1.2 ACE XML

ACE XML [1, 3] is a set of file formats developed to enable communication between the various jMIR software components, including ACE. These file formats have been designed to be very flexible and expressive. It is hoped that the MIR research community will eventually adopt them as multi-purpose standardized formats, beyond the limited scope of jMIR. ACE XML has recently been significantly revised and expanded in order to help make this possible [3].

1.3 ACE

ACE is a meta-learning classification system that can automatically experiment with a variety of different dimensionality-reduction and machine-learning algorithms in order to evaluate which ones are best suited to particular problems. ACE can also be used as a simple automatic classification system. ACE is open source and available for free. It is implemented entirely in Java in order to maximize portability.

ACE is built on the standardized Weka machine-learning infrastructure [5] and makes direct use of a variety of algorithms and data structures distributed with Weka. This means not only that new algorithms produced by the very active Weka community can be incorporated into ACE immediately, but also that new algorithms specifically designed for MIR research can be developed using the Weka framework. ACE can read features stored in either ACE XML or Weka ARFF files.

Two Weka data structures of particular interest that are used by ACE and referred to in this paper are the Weka `Instances` object, which stores a set of instances in a representation similar to the Weka ARFF file, and the Weka `Classifer` object, which classifies a set of Weka `Instances` with a specified classification algorithm.

2. RELATED WORK

There are a number of existing software packages that are often used for machine learning, including Weka [5], PRTools [6] and several other MATLAB [7] toolboxes. There are also several systems that offer meta-learning functionality, including RapidMiner (formerly Yale) [8] and METAL [9]. All of these are general purpose systems, however, and do not meet some of the special needs of MIR, as discussed in [1]. ACE and ACE XML make it possible to represent and use types of information that are particularly relevant to MIR but are not expressible or usable in most alternative systems. For example, jMIR and ACE XML have the ability to:

- maintain logical groupings between multi-dimensional features;
- represent class labels and feature values for potentially overlapping sub-sections of instances as well as for instances as a whole;
- represent structured class ontologies; and
- associate multiple classes with a single instance.

There are also several high-quality toolsets that have been designed specifically for MIR, but they tend to offer less sophisticated processing specifically with respect to machine learning. MIRtoolbox [10] is a powerful modular MATLAB toolbox for designing and extracting audio features. The well-known CLAM [11] and Marsyas [12] focus on audio-related tasks. The M2K [13] graphical patching interface can be used to connect a range of different MIR processing components in ways that can take advantage of distributed processing.

3. IMPROVEMENTS NEW TO ACE 2.0

3.1 Architectural Restructuring

ACE's class structure has been redesigned to be more flexible, extensible, and easy to understand. This redesign is intended to facilitate integration with other software. ACE's main functionality is accessed through an interface class called `Coordinator`. Figure 1 illustrates this organization: the GUI, command-line interface, and external software all only directly access this new `Coordinator` class, which then communicates with ACE's processing classes. This organization ensures that all processing is performed identically, regardless of the source of the request, and makes ACE easier to use. New users wishing to use the ACE API need only understand the `Coordinator` class in order to be able to use all of ACE's functionality. The remainder of this section presents ACE's main classes.

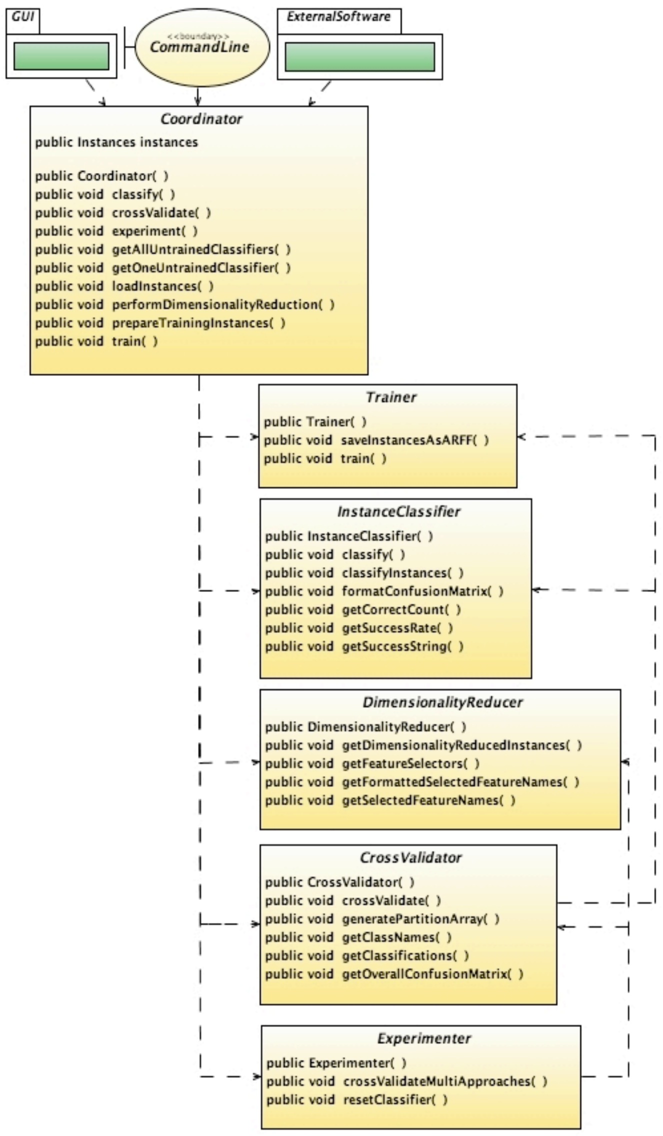

GUI
<<boundary>>
CommandLine
ExternalSoftware
Coordinator
public Instances instances
public Coordinator( )
public void classify( )
public void crossValidate( )
public void experiment( )
public void getAllUntrainedClassifiers( )
public void getOneUntrainedClassifier( )
public void loadInstances( )
public void performDimensionalityReduction( )
public void prepareTrainingInstances( )
public void train( )
Trainer
public Trainer( )
public void saveInstancesAsARFF( )
public void train( )
InstanceClassifier
public InstanceClassifier( )
public void classify( )
public void classifyInstances( )
public void formatConfusionMatrix( )
public void getCorrectCount( )
public void getSuccessRate( )
public void getSuccessString( )
DimensionalityReducer
public DimensionalityReducer( )
public void getDimensionalityReducedInstances( )
public void getFeatureSelectors( )
public void getFormattedSelectedFeatureNames( )
public void getSelectedFeatureNames( )
CrossValidator
public CrossValidator( )
public void crossValidate( )
public void generatePartitionArray( )
public void getClassNames( )
public void getClassifications( )
public void getOverallConfusionMatrix( )
Experimenter
public Experimenter( )
public void crossValidateMultiApproaches( )
public void resetClassifier( )

Figure 1. Structure of ACE's main processing classes. The `Coordinator` class provides an exclusive interface through which ACE's main functionality can be accessed. Arrows indicate interactions between classes. All public methods are listed, but parameters are omitted from method declarations to save space.

- `Coordinator`: The class provides the interface through which ACE's training, classification, cross validation, and experimentation functionality can be accessed. Only `Coordinator` calls the classes listed below; all other sources need only call the appropriate methods in `Coordinator` to access the functionality of all other processing classes. Loading and preparation of instances as well as dimensionality reduction is performed in this class prior to passing instances to processing classes.

- `Trainer`: Trains a specified type of Weka `Classifier` based on the given training instances. The trained Weka `Classifier` is stored and saved in an ACE `TrainedModel` object.

- `InstanceClassifier`: Classifies a set of instances using a trained Weka `Classifier`. This class reads the `TrainedModel` object from a specified file and uses it to classify the given instances. In the context of cross validation, a classified Weka `Instances` object is returned. Classifications can be written to a Weka ARFF file or an ACE XML Instance Label file.

- `DimensionalityReducer`: Reduces the dimensionality of the features extracted from a set of instances. This class is called by the `Coordinator` class to reduce the dimensionality of the training data prior to training and cross-validation in order to help avoid the "curse of dimensionality." This class is also called by the `Experimenter` class to create an array of multiple dimensionality-reduced versions of an original set of instances.

- `CrossValidator`: Cross-validates the given instances with the specified type of Weka `Classifier` using the specified number of partitions. Instances are partitioned randomly into training and testing data for each fold. `CrossValidator` makes calls to the `Trainer` and `InstanceClassifier` classes to evaluate the performance of a specific classification approach. The specified type of Weka `Classifier` is trained on the remaining training data and tested on the testing data for each partition. Statistics are stored for each partition and used to generate performance reports that provide much more statistical detail than Weka itself provides.

- `Experimenter`: Tests to find the best performing classification approach by making repeated calls to the `CrossValidator` class using different parameters each time. Different types of classifiers are tested with different types of dimensionality reduction. `Experimenter` calls `DimensionalityReducer` to get an array of Weka `Instances` objects, wherein each cell contains a different dimensionality-reduced version of the original instances.

  Each type of classifier is cross-validated with each set of dimensionality-reduced instances. A summary of the results for each cross-validation experiment for each dimensionality-reduction experiment is generated, as well as more detailed results when requested by the user. After the best classification methodology has been selected, validation is performed using a publication set put aside at the beginning of the experiment. A new Weka `Classifier` of the chosen type is created and trained on the chosen type of dimensionality-reduced instances (all instances are now available for use as training data, except for the publication set). The newly trained Weka `Classifier` is tested on the publication set and the results are saved.

3.2 Redesigned Cross Validation

ACE performs full meta-learning with training, testing, and publication data sets. Previously, cross-validation was performed using the Weka API, but now, ACE implements its own cross-validation that improves upon Weka's. This new implementation, contained in the `CrossValidator` class, includes output of additional statistics and more transparent data processing. Whereas previously Weka's cross-validation only allowed access to overall correctness statistics and confusion matrices, ACE's new implementation includes variances across partitions, individual instance classification results for each partition, confusion matrices for each partition, and data on running times.

3.3 ACE XML 2.0 ZIP and Project Files

ACE XML, the file format used to transmit information between the jMIR components, consists of four different file types for storing, respectively, extracted feature values, feature metadata, labeled instances and class ontologies. Although the separation of this data into four different types of files does have significant advantages [4], large projects consisting of multiple files can become unwieldy.

The new ACE XML 2.0 Project and ZIP files present solutions to this problem. The Project file allows users to associate ACE XML files together so that they may be automatically saved or loaded together, and the ZIP format makes it possible to package all files referred to in a Project file into a single compressed ZIP file.

The ACE XML ZIP file is implemented using an ACE XML Project file and a hidden text file with the extension ".sp". This file contains only one line of text that specifies the name of the single ACE XML Project file compressed within the ACE XML ZIP file. When ACE parses an ACE XML ZIP file, it looks for the .sp file first, and then parses the associated ACE XML Project file so that the other contents of the ACE XML ZIP file can be properly interpreted.

ACE includes utilities for creating, accessing, and managing ACE XML ZIP files. When ACE unzips an ACE XML ZIP file, it rewrites the ACE XML Project file to reflect the new path names of the newly unzipped files. ACE can also extract or add a single file from/to an ACE XML ZIP file. An ACE XML ZIP file can be used to load or save an ACE project via the ACE command-line interface, GUI or API.

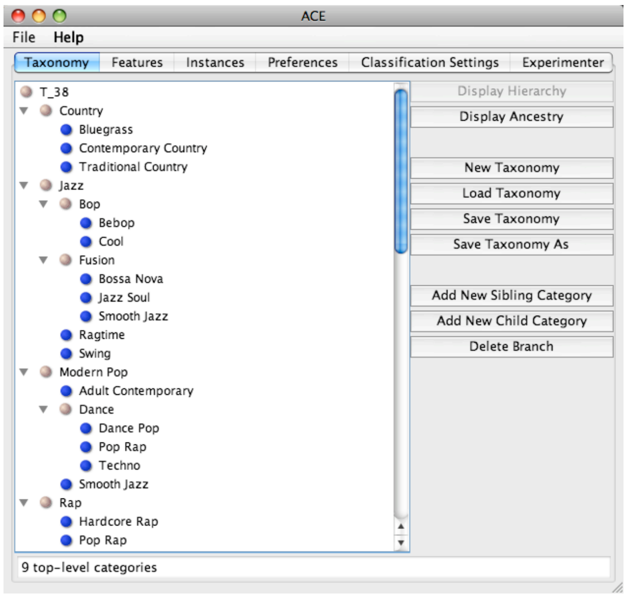
3.4 Improved Command-Line Interface

The previous ACE command-line interface has been entirely redesigned with clearer and more intuitive commands. The command-line interface of software such as ACE is particularly important, as it is often needed to perform batch processing that can last days or weeks. Running ACE from the command line has become easier with the addition of new functionality such as the ability to load an ACE project from an ACE XML Project file or an ACE ZIP file. The user can also now specify the type of classifier or dimensionality-reduction algorithm to be used as well as other options related to the distribution of datasets (e.g., randomization, maximum class membership, and maximum class spread). With the *verbose* option, the user also has the option of printing a more detailed report of the performed processing. These improvements to the command-line interface not only make ACE easier to use, but also provide more precise control of ACE's processing.

3.5 Graphical User Interface

ACE also now includes functionality for GUI-based viewing, editing, and saving of ACE XML files. This functionality is divided between three panes: the *Taxonomy* pane, which displays the contents of an ACE XML Class Ontology file; the *Features* pane, which displays the contents of an ACE XML Feature Description file; and the *Instances* pane, which displays the combined contents of both ACE XML Feature Value files and ACE XML Instance Label files.

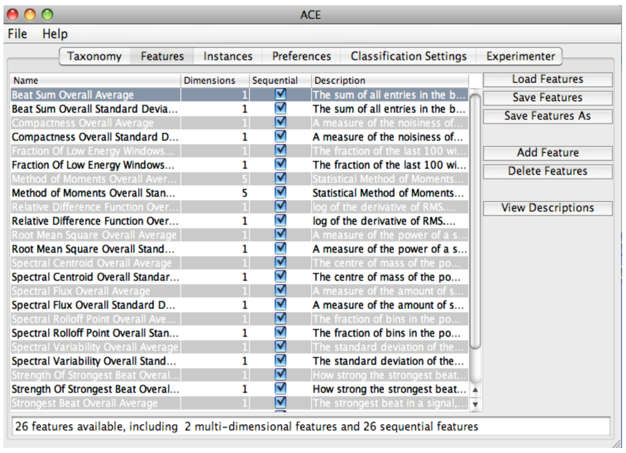
A screen shot of the *Taxonomy* pane is shown in Figure 2. The displayed structure indicates a genre taxonomy for use in an automatic genre classification task. If an ACE XML Instance Label file is loaded without explicitly specifying such a class ontology either manually or with an ACE XML Class Ontology file, then a flat ontology is automatically generated based on the labels used in the ACE XML Instance Label file, and is displayed in the *Taxonomy* pane. Figure 3 shows a *Features* pane displaying a list of audio features. If an ACE XML Feature Descriptions file is not loaded here prior to loading an ACE XML Feature Values file, feature descriptions are generated automatically based on the features present in the ACE XML Feature Values file. Figure 4 shows how the *Instances* pane can be used to display class labels that have been associated with particular instances.



- T_38
- Country
  - Bluegrass
  - Contemporary Country
  - Traditional Country
- Jazz
  - Bop
    - Bebop
    - Cool
  - Fusion
    - Bossa Nova
    - Jazz Soul
    - Smooth Jazz
  - Ragtime
  - Swing
- Modern Pop
  - Adult Contemporary
  - Dance
    - Dance Pop
    - Pop Rap
    - Techno
  - Smooth Jazz
- Rap
  - Hardcore Rap
  - Pop Rap

9 top-level categories

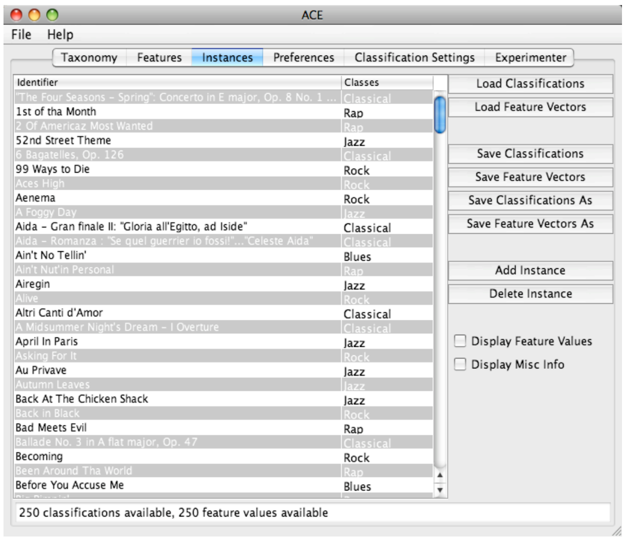
Figure 2. A sample genre ontology displayed in the *Taxonomy* pane of the ACE GUI.



Name	Dimensions	Sequential	Description
Beat Sum Overall Average	1	✓	The sum of all entries in the b...
Beat Sum Overall Standard Devia...	1	✓	The sum of all entries in the b...
Compactness Overall Average	1	✓	A measure of the noisiness of...
Compactness Overall Standard D...	1	✓	A measure of the noisiness of...
Fraction Of Low Energy Windows...	1	✓	The fraction of the last 100 wi...
Fraction Of Low Energy Windows...	1	✓	The fraction of the last 100 wi...
Method of Moments Overall Aver...	5	✓	Statistical Method of Moments...
Method of Moments Overall Stan...	5	✓	Statistical Method of Moments...
Relative Difference Function Over...	1	✓	log of the derivative of RMS....
Relative Difference Function Over...	1	✓	log of the derivative of RMS....
Root Mean Square Overall Average	1	✓	A measure of the power of a s...
Root Mean Square Overall Stand...	1	✓	A measure of the power of a s...
Spectral Centroid Overall Average	1	✓	The centre of mass of the po...
Spectral Centroid Overall Standar...	1	✓	The centre of mass of the po...
Spectral Flux Overall Average	1	✓	A measure of the amount of s...
Spectral Flux Overall Standard D...	1	✓	A measure of the amount of s...
Spectral Rolloff Point Overall Ave...	1	✓	The fraction of bins in the po...
Spectral Rolloff Point Overall Stan...	1	✓	The fraction of bins in the po...
Spectral Variability Overall Average	1	✓	The standard deviation of the...
Spectral Variability Overall Stand...	1	✓	The standard deviation of the...
Strength Of Strongest Beat Overal...	1	✓	How strong the strongest beat...
Strength Of Strongest Beat Overal...	1	✓	How strong the strongest beat...
Strongest Beat Overall Average	1	✓	The strongest beat in a signal,...

26 features available, including 2 multi-dimensional features and 26 sequential features

Figure 3. The *Features* pane of the ACE GUI displaying metadata about a set of audio features.



Identifier	Classes
"The Four Seasons – Spring": Concerto in E major, Op. 8 No. 1 ...	Classical
1st of tha Month	Rap
2 Of Americaz Most Wanted	Rap
52nd Street Theme	Jazz
6 Bagatelles, Op. 126	Classical
99 Ways to Die	Rock
Aces High	Rock
Aenema	Rock
A Foggy Day	Jazz
Aida – Gran finale II: "Gloria all'Egitto, ad Iside"	Classical
Aida – Romanza : "Se quel guerrier io fossi!"..."Celeste Aida"	Classical
Ain't No Tellin'	Blues
Ain't Nut'in Personal	Rap
Airegin	Jazz
Alive	Rock
Altri Canti d'Amor	Classical
A Midsummer Night's Dream – I Overture	Classical
April In Paris	Jazz
Asking For It	Rock
Au Privave	Jazz
Autumn Leaves	Jazz
Back At The Chicken Shack	Jazz
Back in Black	Rock
Bad Meets Evil	Rap
Ballade No. 3 in A flat major, Op. 47	Classical
Becoming	Rock
Been Around Tha World	Rap
Before You Accuse Me	Blues

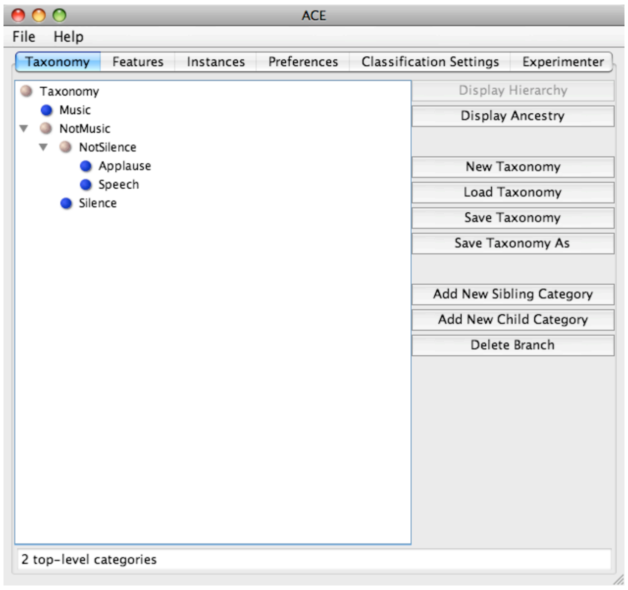
250 classifications available, 250 feature values available

Figure 4. The *Instances* pane of the ACE GUI. Song titles are associated in this example with particular genre

labels drawn from the genre ontology shown in Figure 2. Note that neither the *Display Feature Values* nor the *Display Misc Info* checkboxes are checked, so only instance identifiers and classes are displayed in this particular example.

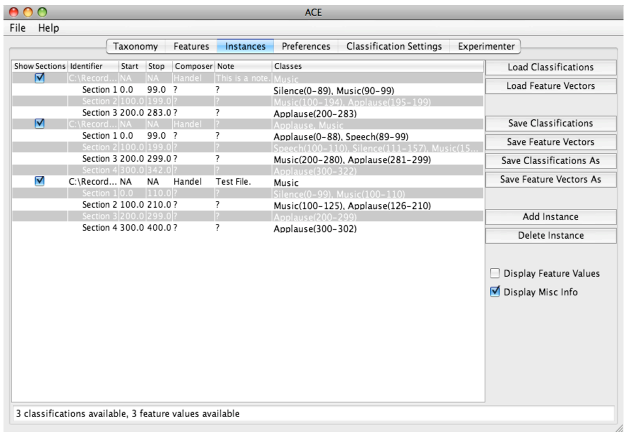
Figures 5, 6, and 7 illustrate a more complex example. Figure 5 establishes a new class ontology, in this case a hierarchical music–speech–applause–silence discriminator. Figure 6 shows how the *Instances* pane can display not only class labels, but also miscellaneous metadata. This figure also demonstrates that instances can be broken into separately-labeled subsections and that each instance or subsection may be associated with multiple class labels. Start and stop times indicate the boundaries of the subsections. Figure 7 demonstrates how feature values can also be displayed using the *Instances* pane. It shows the same data as Figure 6, except that feature values are displayed and miscellaneous metadata is not.

If feature arrays and class labels are loaded for the same subsection, all information for that instance is presented in one row. The specific time of the overlap of class labels within a subsection is indicated within parentheses after the class name. Subsection rows for any overall instance can be hidden by unchecking the *Show Sections* checkbox. The *Composer* and *Note* columns are metadata loaded from the particular Instance Label ACE XML file associated with this example and can be hidden by clicking on the *Display Misc Info* checkbox.



2 top-level categories

Figure 5. Another class ontology displayed in the *Taxonomy* pane of the ACE GUI.

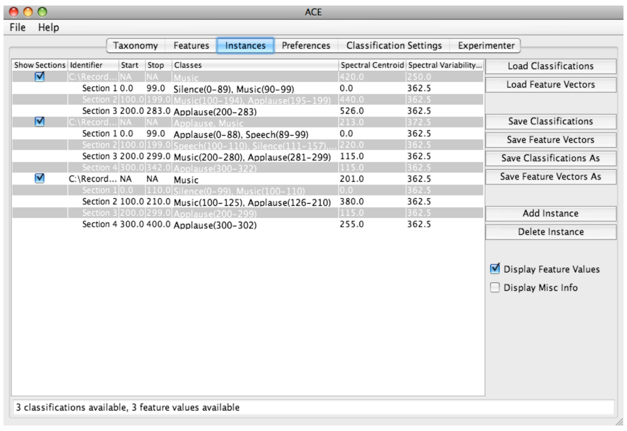


Show Sections	Identifier	Start	Stop	Composer	Note	Classes
✓	C:\Record...	NA	NA	Handel	This is a note.	Music
	Section 1	0.0	99.0	?	?	Silence(0-89), Music(90-99)
	Section 2	100.0	199.0	?	?	Music(100-194), Applause(195-199)
	Section 3	200.0	283.0	?	?	Applause(200-283)
✓	C:\Record...	NA	NA	Handel	?	Applause, Music
	Section 1	0.0	99.0	?	?	Applause(0-88), Speech(89-99)
	Section 2	100.0	199.0	?	?	Speech(100-110), Silence(111-157), Music(15...
	Section 3	200.0	299.0	?	?	Music(200-280), Applause(281-299)
	Section 4	300.0	342.0	?	?	Applause(300-322)
✓	C:\Record...	NA	NA	Handel	Test File.	Music
	Section 1	0.0	110.0	?	?	Silence(0-99), Music(100-110)
	Section 2	100.0	210.0	?	?	Music(100-125), Applause(126-210)
	Section 3	200.0	299.0	?	?	Applause(200-299)
	Section 4	300.0	400.0	?	?	Applause(300-302)

☐ Display Feature Values
☑ Display Misc Info

3 classifications available, 3 feature values available

Figure 6. Instances with subsections and metadata displayed in the *Instances* pane of the ACE GUI.



Show Sections	Identifier	Start	Stop	Classes	Spectral Centroid	Spectral Variability...
✓	C:\Record...	NA	NA	Music	420.0	250.0
	Section 1	0.0	99.0	Silence(0-89), Music(90-99)	0.0	362.5
	Section 2	100.0	199.0	Music(100-194), Applause(195-199)	440.0	362.5
	Section 3	200.0	283.0	Applause(200-283)	526.0	362.5
✓	C:\Record...	NA	NA	Applause, Music	213.0	372.5
	Section 1	0.0	99.0	Applause(0-88), Speech(89-99)	0.0	362.5
	Section 2	100.0	199.0	Speech(100-110), Silence(111-157),...	220.0	362.5
	Section 3	200.0	299.0	Music(200-280), Applause(281-299)	115.0	362.5
	Section 4	300.0	342.0	Applause(300-322)	115.0	362.5
✓	C:\Record...	NA	NA	Music	201.0	362.5
	Section 1	0.0	110.0	Silence(0-99), Music(100-110)	0.0	362.5
	Section 2	100.0	210.0	Music(100-125), Applause(126-210)	380.0	362.5
	Section 3	200.0	299.0	Applause(200-299)	115.0	362.5
	Section 4	300.0	400.0	Applause(300-302)	255.0	362.5

☑ Display Feature Values
☐ Display Misc Info

3 classifications available, 3 feature values available

Figure 7. Instances with subsections and feature values displayed in the *Instances* pane of the ACE GUI.

4. CONCLUSION AND FUTURE WORK

Improvements have been made to ACE since its original publication, including new capabilities that make it a more complete and easy-to-use meta-learning classification framework. ACE is an ongoing project, and further improvements will continue to be made.

4.1 Fully Functional GUI

The ACE GUI currently serves as a tool for viewing and editing ACE XML files. It will eventually be possible to also use the GUI to perform experiments on data sets, as can currently only be done with the command-line interface or API. This functionality will be accessible from two currently unfinished panes: the *Experimenter* pane and the *Preferences* pane. The *Experimenter* pane will allow full access to all of ACE's machine learning functionality. Several sub-panes will be used to display the same output that is printed or saved to files when running experiments from the ACE command-line interface. The *Preferences* pane of the ACE GUI will allow users to specify preferences related to both interface settings and machine learning parameters. User studies will also be performed in order to validate and improve the GUI design.

4.2 Distributed Work Load

Functionality is being built into ACE to allow it to run trials on multiple computers in parallel in order to reducte execution times. Once the distributed aspect of the system is complete, a server-based subsystem will be designed that contains a coordination system and database. Although not necessary for using ACE, users will be able to choose to dedicate computers to this server, allowing ACE to run continually. The server will keep a record of the performances of all ACE operations run on a particular user's cluster and generate statistics for self-evaluation and improvement. ACE will then make use of any idle time to attempt to improve solutions to previously encountered but currently inactive problems. Ultimately, the user would only be required to specify the total time available (typically days or weeks) for ACE to run its experiments and everything else, including the choice of learning algorithms and their parameters, would be automatically determined by ACE.

4.3 Expanded Machine Learning Algorithms

In the future, ACE will include learning schemes important to MIR that are currently missing from the Weka distribution, such as hidden Markov models and recurrent neural networks. Support for Weka's unsupervised learning functionality will also be incorporated. It would also be beneficial to include tools for constructing blackboard systems, in particular those that can integrate knowledge sources based on expert heuristics. Another potentially beneficial addition would be to implement modules for facilitating post-processing. All of these extensions would add to ACE's flexibility and breadth of processing.

5. ACKNOWLEDGEMENTS

The authors would like to thank the *Andrew W. Mellon Foundation* and the *Centre for Interdisciplinary Research in Music Media and Technology (CIRMMT)* for their generous financial support, as well as the members of the *Networked Environment for Music Analysis (NEMA)* group for their valuable critiques and suggestions.

6. REFERENCES

[1] McKay, C., R. Fiebrink, D. McEnnis, B. Li, and I. Fujinaga. 2005. ACE: A framework for optimizing music classification. *Proceedings of the International Conference on Music Information Retrieval.* 42–9.

[2] McKay, C., and I. Fujinaga. 2009. jMIR: Tools for automatic music classification. *Proceedings of the International Computer Music Conference*

[3] McKay, C., J. A. Burgoyne, J. Thompson, and I. Fujinaga. 2009. Using the ACE XML 2.0 file formats to store and share music classification data. *Proceedings of the International Conference on Music Information Retrieval.*

[4] McKay, C., and I. Fujinaga. 2008. Combining features extracted from audio, symbolic and cultural sources. *Proceedings of the International Conference on Music Information Retrieval*, 597–602.

[5] Witten, I., and E. Frank. 2005. *Data mining: Practical machine learning tools and techniques with Java implementations*. San Francisco: Morgan Kaufmann.

[6] van der Heijden, F., R. P. W. Duin, D. de Ridder, and D. M. J. Tax. 2004. *Classification, parameter estimation and state estimation: An engineering approach using MATLAB.* New York: Wiley.

[7] MathWorks 2008. *MATLAB version 7.6.0.* Natick, Massachusetts: The MathWorks.

[8] Mierswa, I., M. Wrust, R. Klinkenberg, M. Scholz, and T. Euler. 2006. YALE: Rapid prototyping for complex data mining tasks. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.*

[9] Brazdil, P., C. Soares, and J. Costa. 2003. Ranking learning algorithms. *Machine Learning* 50 (3): 251–77.

[10] Lartillot, O., P. Toiviainen, and T. Eerola. 2008. A Matlab toolbox for music information retrieval. In *Data analysis, machine learning and applications,* ed. C. Preisach, H. Burkhardt, L. Schmidt-Thieme, and R. Decker. New York: Springer. 261–8.

[11] Arumi, P., and X. Amatriain. 2005. CLAM: An object oriented framework for audio and music. *Proceedings of the International Linux Audio Conference.*

[12] Tzanetakis, G., and P. Cook. 1999. MARSYAS: A framework for audio analysis. *Organized Sound* 4 (3): 169–75.

[13] Downie, S., A. Ehmann, and D. Tcheng. 2005. Music-to-knowledge (M2K): A prototyping and evaluation environment for music information retrieval research. *Proceedings of the 28th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 676.