# ACE: A FRAMEWORK FOR OPTIMIZING MUSIC CLASSIFICATION


| Cory McKay | Rebecca Fiebrink | Daniel McEnnis | Beinan Li | Ichiro Fujinaga |
|---|---|---|---|---|
| Music Technology | Music Technology | Music Technology | Music Technology | Music Technology |
| McGill University | McGill University | McGill University | McGill University | McGill University |
| Montreal, Canada | Montreal, Canada | Montreal, Canada | Montreal, Canada | Montreal, Canada |
| cory.mckay@ mail.mcgill.ca | rfiebrink@ acm.org | daniel.mcennis@ mail.mcgill.ca | beinan.li@ mail.mcgill.ca | ich@ music.mcgill.ca |



## ABSTRACT

This paper presents ACE (Autonomous Classification Engine), a framework for using and optimizing classifiers. Given a set of feature vectors, ACE experiments with a variety of classifiers, classifier parameters, classifier ensembles and dimensionality reduction techniques in order to arrive at a good configuration for the problem at hand. In addition to evaluating classification methodologies in terms of success rates, functionality is also being incorporated into ACE allowing users to specify constraints on training and classification times as well as on the amount of time that ACE has to arrive at a solution.

ACE is designed to facilitate classification for those new to pattern recognition as well as provide flexibility for those with more experience. ACE is packaged with audio and MIDI feature extraction software, although it can certainly be used with existing feature extractors.

This paper includes a discussion of ways in which existing general-purpose classification software can be adapted to meet the needs of music researchers and shows how these ideas have been implemented in ACE. A standardized XML format for communicating features and other information to classifiers is proposed.

A special emphasis is placed on the potential of classifier ensembles, which have remained largely untapped by the MIR community to date. A brief theoretical discussion of ensemble classification is presented in order to promote this powerful approach.



**Keywords:** music classification, classifier ensembles, combining classifiers, optimization, MIR


## 1 INTRODUCTION

Classification techniques play an essential role in many MIR-related research areas. These include genre classification, similarity analysis, music recommendation, performer identification, composer identification and instrument identification, to name just a few. An examination of the MIREX evaluation topics clearly demonstrates the importance of classification in MIR.

Despite this importance, there has been relatively little work on developing standardized and easy-to-use classification software with the particular needs of music in mind. A survey of published MIR papers reveals that many researchers either implement their own custom-built systems or use off-the-shelf pattern recognition software that was developed for fields other than music.

The former approach results in time wasted through duplication of effort and, potentially, relatively limited software, as one only has so much time to devote to building classifiers if this is only a part of a larger research project. Using general pattern recognition frameworks can work well with some limited applications, but one inevitably encounters complications, limitations and difficulties due to the particularities of music.

Standardized classification software especially adapted to MIR could therefore be of significant benefit. Fortunately, some work has been done in this area. Marsyas (Tzanetakis and Cook 1999) in particular has been used effectively by many researchers, and M2K (Downie 2004) has great promise. ACE (Autonomous Classification Engine) is proposed here as a framework that builds upon these important systems and addresses a number of areas that remain to be dealt with.

Section 2 of this paper discusses the shortcomings of general-purpose pattern recognition frameworks with respect to music and proposes specific improvements. A particular emphasis is put on the importance of a standardized method of transmitting features from feature extractors to classification software. Several XML file formats are proposed in order to address this issue.

Section 3 of this paper concentrates on grouping classifiers into ensembles. Many MIR researchers perform experiments with a variety of classifiers in order to find the ones that are best suited to their particular tasks. Only a few experiments, such as the Bodhidharma genre classification system (McKay 2004), however, have been conducted on combining these classifiers into ensembles.

This is surprising, given the proven effectiveness of ensemble algorithms such as AdaBoost (Freund and Shapire 1996). Classifier ensembles have been gaining increasing attention in the machine learning and pattern recognition communities over the past decade, and the MIR community could certainly benefit from experimenting with the wide variety of potentially very powerful approaches that are available. This is particularly true considering the asymptotic behaviour that success rates appear to be demonstrating in a variety of MIR areas, as observed by Aucouturier and Pachet (2004).

Of course, classifier ensembles do come at the cost of added complexity, and the variety of approaches available can be daunting. Section 3 of this paper presents a brief survey of the field in order to encourage experimentation with classifier ensembles in MIR research.

Even when only one classifier is used, the variety and sophistication of classification techniques can make it difficult to decide which techniques and parameters to use. Even the most experienced pattern recognition researchers must often resort to experimentation. The ACE framework has been designed to deal with this problem automatically. ACE performs optimization experiments using different dimensionality reduction techniques, classifiers, classifier parameters and classifier ensemble architectures. Particular efforts have been made to investigate the power of feature weighting (Fiebrink, McKay, and Fujinaga 2005).

Functionality is also being built into ACE that allows it to analyze the effectiveness of different approaches not only in terms of classification accuracy, but also training time and classification time. This allows users to experimentally determine the best set of techniques to use for their particular priorities.

ACE may also be used directly as a classifier. Once users have selected the classifier(s) that they wish to use, whether through ACE optimization or using pre-existing knowledge, they need only provide ACE with feature vectors and model classifications. ACE then trains itself and presents users with trained classifier(s).

An important advantage of ACE is that it is open-source and freely distributable. ACE is implemented in Java, which means that the framework is portable and easy to install. ACE has also been built with a modular and extensible design philosophy. It is a simple and well-documented matter for users to build upon ACE.

Section 4 of this paper presents the ACE framework itself. The implementation and functionality of the software are discussed, as are the benefits that it offers the MIR community.

Any classification system is only as good as the features that it receives. ACE is packaged with easy-to-use, flexible and, perhaps most importantly, highly extensible feature extraction software for extracting features from both audio and MIDI files. There is no requirement to use ACE with these feature extractors, however, as ACE is designed to work well with any feature extractors that can generate appropriately formatted output. Section 5 presents a brief overview of the bundled feature extraction software.

Both feature extraction and classification can be computationally intensive. This is particularly true of a system such as ACE, given its exploratory approach. Section 6 details future plans for adapting ACE so that it can distribute its workload over multiple computers.

Section 7 shows the results of several test classification tasks performed with ACE in order to evaluate its effectiveness. These include tests on standard benchmarks from the UCI Machine Learning Repository as well as two MIR-specific tasks.

Section 8 summarizes this paper and Section 9 presents some ideas for future additions to ACE.

# 2 DEVELOPING A CLASSIFICATION FRAMEWORK SUITED TO MIR

## 2.1 Limitations of existing systems

The development of a general pattern recognition software package is not trivial. Each application domain has its own needs and peculiarities that might not occur to researchers in other fields. It is therefore no surprise that what general pattern recognition frameworks are available have important weaknesses with respect to MIR.

In general, it appears that PRTools (van der Heijden et al. 2004) and Weka (Witten and Frank 2000) are the two most often used general frameworks in MIR. PRTools is a Matlab toolbox and Weka is a Java application and code library. Both of these frameworks are very well-designed and powerful tools, but they do have several limitations when applied to MIR.

PRTools has the disadvantage that it is reliant upon Matlab, a proprietary software package. Although PRTools itself is free for academic use, one must still purchase Matlab in order to use it. Furthermore, one must pay for PRTools if one wishes to use it commercially, and its licence does not permit it to be redistributed. This means that any software that is developed using PRTools cannot be distributed without special permission, and it cannot be distributed with an open licence. So, although PRTools is certainly suitable for basic research and prototyping, it is problematic with respect to serious application development.

This introduces some of the important concerns with respect to MIR software. The general consensus in the MIR community appears to be supportive of free, open source and fully distributable software. This is important in ensuring research transparency and sharing of results, and it is essential in allowing researchers to build upon each other’s work.

Related to this is the importance of extensibility and modularity. In an open research community, not only should code be freely distributable, but it must be designed so that others can expand upon it easily.

Portability, documentation and ease of use and installation are also important considerations. Although lip service is often paid to these principles, they should be taken very seriously. It is not at all an uncommon experience for potential users to become discouraged by installation difficulties, such as linking errors, or by arcane code documentation.

Furthermore, good MIR software should be usable and understandable by users with a variety of skill levels. The MIR community is composed of experts in a wide variety of fields, and it is not reasonable to expect all of them to be highly knowledgeable about classification, even though it might be of benefit to their research.

The Weka data mining framework largely meets these requirements. It is freely distributable, open source, relatively well documented, implemented with all of Java’s platform-independence, beautifully designed and truly a pleasure to work with. It also includes a variety of interfaces for users with different needs and abilities. It is, however, as is inevitable with any general system, missing some important qualities with respect to MIR.

The most significant issues are related to the Weka ARFF file format that is used to store features and communicate them to classifiers. To begin with, there is no good way to assign more than one class to a given instance. One possible solution is to break one multi-class problem into many binary classification problems, so that there is a separate ARFF file for every class, with all instances classified as either belonging or not belonging to each class. Alternatively, one could create a separate class for every possible combination of classes, with a resulting exponential increase in the numbers of classes.

It is clear that neither of these solutions is ideal. Unfortunately, this is a problem with classification systems in general, not just Weka. This is understandable, as most pattern recognition tasks require classification into one and only one class. Unfortunately, a great deal of musicological research involves certain unavoidable ambiguities, and the imposition of only one class membership on each instance is unrealistic for tasks such as genre classification and many types of similarity-related classification, for example.

A second problem is that ARFF files do not permit any logical grouping of features. Each feature is treated as an independent quantity with no relation to any other feature. One often encounters multi-dimensional features in music, and it can be useful to maintain some logical relationship between the components of such features. Power spectra, MFCC's, bins of a beat histogram and instruments present are just a few examples. Maintaining a logical relationship between the values of multi-dimensional features allows one to perform classifications in particularly fruitful ways that take advantage of their interrelatedness, particularly with respect to classifier ensembles. Training one neural net on MFCC's, for example, and using another classifier for features such as RMS or spectral centroid could prove much more fruitful than mixing the MFCC's in with the other features.

A third problem is that ARFF files do not allow any labelling or structuring of instances. Each instance is stored only as a collection of feature values and a class identifier, with no identifying metadata. In music, it is often appropriate to extract features over a number of windows. Furthermore, some features may be extracted for each window, some only for some windows and some only for a recording as a whole. Weka and its ARFF files provide no way of associating the features of a window with the recording that it comes from, nor do they provide any means of identifying recordings or of storing time stamps associated with each window. This means that this information must be stored, organized and processed by some external software using some unspecified and non-standardized file format.

A fourth problem is that there is no way of imposing a structure on the class labels. One often encounters hierarchical structures in music, such as in the cases of genre categories or structural analyses. Weka treats each class as distinct and independent. This means that there is no native way to use classification techniques that make use of structured taxonomies.

These criticisms are not meant to denigrate Weka in any way. Quite to the contrary, in fact, as Weka is singled out here only because it is arguably the best framework available. One of the many positive aspects of Weka is that it is easy to write Java code that makes use of the excellent existing Weka code and adds functionality to it, which is precisely what ACE does.

Many of the issues discussed above apply to existing systems developed specifically with music in mind as well. As mentioned in Section 1, the two most well-known and powerful such systems are Marsyas and M2K.

Marsyas is a pioneering system that has been used very effectively in a number of research projects. Unfortunately, there can be some portability and installation issues with this C++ based system. Marsysas is also currently centred around audio classification, and does not currently include MIDI functionality.

It is also unfair to compare Marsyas to general classification systems such as Weka, as Marsyas was originally designed primarily as a feature extractor, and performs very well at this task. Marsyas is, however, regularly maintained by its creator, George Tzanetakis, and there are plans to extend its functionality and possibly port increasing amounts of Weka's functionality to it.

M2K is a graphical feature extraction and classification framework based on the D2K parallel data mining and machine learning system. Although still in alpha release, and therefore impossible to fairly evaluate, M2K promises to be an extremely powerful and flexible system for MIR prototyping.

Unfortunately, M2K does inherit several licensing problems from D2K that potentially limit its use beyond prototyping. D2K's licence can make it complicated for researchers outside the U.S.A. to obtain it, and forbids its use in commercial applications. This means that any system that uses D2K cannot itself be used for any non-research-based tasks. Furthermore, D2K is not open source.

### 2.2 Feature file formats

It is clear from Section 2.1 that there is an important need for a standardized and flexible file format for storing feature values and communicating them to classifiers. Existing formats such as ARFF, while certainly suitable for the types of tasks their designers had in mind, are insufficient for the particular needs of MIR researchers.

Several XML-based file formats are presented here in order to attempt to meet this need. XML is chosen because it is not only a standardized format for which parsers are widely available, but is also extremely flexible. It is a verbose format, with the result that it is less space efficient than formats such as ARFF, but this verbosity has the corresponding advantage that it allows humans to easily read the files. This is particularly useful when one is working on debugging feature extractors.

An important priority when developing a feature file format is to enforce a clear separation between the feature extraction and classification tasks, as particular researchers may have reasons for using particular feature extractors or particular classification systems. The file format should therefore make it possible to use any feature extractor to communicate any features of any type to any classification system. This portability makes it possible to use features generated with different extractors with the same classification system, or a given set of extracted features with multiple classification systems.

The reusability of files is another important consideration. For example, it could be useful to use the same set of extracted features for a variety of tasks, such as genre classification as well as artist identification. Similarly, it could be convenient to reuse the same model classifications with different sets of features. For example, one could classify a given corpus of audio recordings and then later perform the same task on symbolic recordings of the same corpus using the same model classifications. Unfortunately, most current feature file formats merge feature values and model classifications, making this kind of reusability difficult.

The use of two separate files is therefore proposed for what is traditionally contained in one file, namely one file for storing feature values and another for storing model classifications. Unique keys such as file names can be used to merge the two files. The model classification file can be omitted when using unsupervised learning or classifying unknown patterns.

We also propose the use of an additional optional file for specifying taxonomical structures. This enables one to specify the relationships between classes, information which can be very useful for tasks such as hierarchical classification. This file can be omitted if only flat classification is to be used.

One final optional file format is proposed for storing metadata about features, such as basic descriptions or details about the cardinality of multi-dimensional features. Although not strictly necessary, such a file helps solidify the potential for full independence between feature extractors and classifiers. A researcher with a classifier could be e-mailed a feature values file and a feature definitions file by other researchers, for example, and would need no additional information at all about the feature extractor used or the features it extracted.

The explicit Document Type Definitions (DTD's) of the four proposed ACE XML formats are shown in Figures 1 through 4. It can be seen from Figure 1 that features may be stored for overall instances, called data sets, which may or may not have sub-sections. This can correspond to a recording and its windows, for example. Each sub-section has its own features, and each data set may have overall features as well. Each sub-section may have start and stop stamps in order to indicate what portion of the data set it corresponds to. This makes it possible to have windows of arbitrary and varying sizes that can overlap. Each feature has a name identifying it, which makes it possible to omit features from some data sets or sub-sections if appropriate. Each feature may also have one or more values (denoted by the <v> element) in order to permit multi-dimensional features.

```
<!ELEMENT feature_vector_file (comments,
                               data_set+)>
<!ELEMENT comments (#PCDATA)>
<!ELEMENT data_set (data_set_id,
                    section*,
                    feature*)>
<!ELEMENT data_set_id (#PCDATA)>
<!ELEMENT section (feature+)>
<!ATTLIST section start CDATA ""
                  stop CDATA "">
<!ELEMENT feature (name, v+)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT v (#PCDATA)>
```

**Figure 1**. XML DTD of the ACE XML file format for storing feature values.

```
<!ELEMENT classifications_file(comments,
                               data_set+)>
<!ELEMENT comments (#PCDATA)>
<!ELEMENT data_set (data_set_id,
                   misc_info*,
                   role?,
                   classification)>
<!ELEMENT data_set_id (#PCDATA)>
<!ELEMENT misc_info (#PCDATA)>
<!ATTLIST misc_info info_type CDATA "">
<!ELEMENT role (#PCDATA)>
<!ELEMENT classification (section*,
                          class*)>
<!ELEMENT section (start,
                  stop,
                  class+)>
<!ELEMENT class (#PCDATA)>
<!ELEMENT start (#PCDATA)>
<!ELEMENT stop (#PCDATA)>
```

**Figure 2**. XML DTD of the proposed ACE XML file format for storing classifications.

```
<!ELEMENT taxonomy_file (comments,
                         parent_class+)>
<!ELEMENT comments (#PCDATA)>
<!ELEMENT parent_class (class_name,
                        sub_class*)>
<!ELEMENT class_name (#PCDATA)>
<!ELEMENT sub_class (class_name,
                     sub_class*)>
```

**Figure 3**. XML DTD of the optional ACE XML file format for storing class taxonomies.

```
<!ELEMENT feature_key_file (comments,
                            feature+)>
<!ELEMENT comments (#PCDATA)>
<!ELEMENT feature (name,
                   description?,
                   is_sequential,
                   parallel_dimensions)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT is_sequential (#PCDATA)>
<!ELEMENT parallel_dimensions (#PCDATA)>
```

**Figure 4**. XML DTD of the optional ACE XML file format for storing feature definitions.

Figure 2 shows the DTD for storing model classifications. This format may also be used to output classification results. Each data set may have optional metadata associated with it. Each data set can be broken into potentially overlapping sub-sections if desired, and each sub-section can be assigned one or more classes. Each data set may be assigned one or more overall classes as well. Each sub-section is given start and stop stamps to show the region of influence of particular classes.

The DTD for the optional taxonomy format is shown in Figure 3. This format allows the representation of hierarchically structured taxonomies of arbitrary depth.

The final optional file format, for storing feature definitions, is shown in Figure 4. This format enables one to store the name of each possible feature, a description of it, whether or not it can be applied to sub-sections or to overall data sets only and how many dimensions it has.

## 3 CLASSIFIER ENSEMBLES

### 3.1 Motivation for using classifier ensembles

As noted in Section 1, many MIR researchers have performed experiments with multiple classifiers to see which are best suited to particular tasks, but few have attempted to combine these classifiers into ensembles. This section provides justification for doing so.

The practice of combining classifiers into ensembles is inspired by the notion that the combined opinions of a number of experts is more likely to be correct than that of a single expert. Ideally, an ensemble will perform better than any of its individual component classifiers. Although this will often be the case, it is not necessarily guaranteed.

One might question whether it is worth the increases in computational demands and implementation complexity that often accompany ensemble classification if one is not guaranteed an increase in performance. Dietterich (2000) has proposed three reasons why classifier ensembles can be beneficial.

The first reason, referred to by Dietterich as the statistical reason, is as follows. Suppose one has a number of trained classifiers. One knows how well they each performed on the training, testing and potentially the validation data, but this is only an estimate of how well they will each generalize to the universe of all possible inputs. If all of the classifiers performed similarly on the testing and validation data, there is no way of knowing which is in fact the best classifier. If one chooses a single classifier, one runs the risk of accidentally choosing one of the poorer ones. The statistical argument is particularly strong in cases where only limited training and testing data is available, as the evaluation of individual classifiers using test sets is likely to have a high error.

The second reason, referred to as the computational reason, applies to classifiers that train using hill-climbing or random search techniques. Training multiple neural networks, for example, on the same training data can very well result in significantly different trained classifiers, depending on the randomly generated initial conditions. Aggregating such classifiers into an ensemble can take advantage of the multiplicity of solutions offered by the different classifiers. The computational argument highlights the particular appropriateness of instable classifiers for ensemble classification, as they can lead to a variety of useful solutions using only slightly modified training data.

The final reason, termed referential, is based on the fact that there is no guarantee that the types of classifiers that one is using for a particular problem could ever converge to a theoretically optimal solution. To provide a simplified example, say a researcher mistakenly believes that a given problem is linear, and decides to use only linear classifiers. In reality, the optimal classifier will be non-linear, so it is not possible that any of the linear classifiers under consideration could perform optimally individually. However, an ensemble of linear classifiers could approximate a non-linear decision boundary, and could therefore potentially perform better than any single linear classifier ever could.

An essential element in the effectiveness of classifier ensembles is their diversity. If all of the classifiers in an ensemble tend to misclassify the same instances, then combining their results will have little benefit. In contrast, a greater amount of independence between the classifiers can result in errors by individual classifiers being overlooked when the results of the ensemble are combined. Many of the most successful ensemble techniques, such as bagging and boosting (see Section 3.2), are based on increasing classifier diversity.

The well-known effectiveness of algorithms such as AdaBoost (Freund and Shapire 1996) provide convincing experimental evidence for the efficacy of classifier ensembles. It is therefore not surprising that many influential researchers, such as Josef Kittler (2000), continue to emphasize their value.

### 3.2 Overview of classifier ensemble techniques

Although an in-depth survey of ensemble classification is beyond the scope of the paper, a brief overview is presented here in order to promote the use of classifier ensembles in the MIR community. Kuncheva's book (2004) is an excellent resource for those looking for more information.

Methods for combining classifiers into ensembles are often divided among two groups. The first, classifier fusion, involves merging the results of all classifiers through a method such as voting. The second, classifier selection, involves using some system to dynamically select which specialist classifiers to use for each particular input pattern. The mixture of experts method, also called stacking, is an example of a hybrid method where a classifier is trained to weight the votes of other classifiers in the ensemble.

The way in which features and training data are divided up among the component classifiers can play an important role in the success or failure of ensembles.

Bagging and boosting are two powerful techniques that make use of this fact in order to attempt to maximize diversity and, correspondingly, ensemble effectiveness.

Bagging involves using bootstrapping to train the classifiers. This means that each classifier acquires a training set by sampling all available training instances with replacement.

Boosting involves iteratively training classifiers so that the instances that previous classifiers performed poorly on are emphasized in the training sets for subsequent classifiers. The AdaBoost approach, of which there are now many variants, is particularly well known for its success. Boosting tends to perform better than bagging given enough training data, but bagging is better with smaller training sets.

## 4 ACE

The ACE system is designed with the dual goals of increasing classification success rates and facilitating the process of classification for users of all skill levels.

ACE is implemented in Java using the Weka framework. As discussed in Section 2, Weka is powerful, flexible and well designed, but it has some limitations with respect to MIR research needs. A key aspect of ACE is that it adapts Weka to meet these needs, including multi-class membership, hierarchical taxonomies, multi-dimensional features, instance sub-sections, etc.

One of the most important ways in which this is done is through the implementation of the ACE XML file formats, presented in Section 2.2. Although conversion utilities are included to convert between ACE XML and Weka's ARFF format, arguably the current de facto standard in MIR, the use of ACE XML is encouraged because of its superior expressive power.

ACE's use of Weka makes it possible to take advantage of Weka's many classification tools. These include classifiers such as feedforward neural nets, support vector machines, nearest neighbour classifiers, decision tree classifiers and Bayesian classifiers, to name just a few. A variety of dimensionality reduction tools are also available, such as principle component analysis and feature selection through genetic algorithms, exhaustive comparisons and best first searches. Finally, a number of classifier combination techniques are available, including AdaBoost, bagging, majority voting and stacking.

One of the main features of ACE is that it automatically performs experiments with these approaches and their various parameters in order to find those that are well suited to each problem's particular needs. Different approaches often involve tradeoffs between classification success rates and processing times, and functionality is being built into ACE to make it possible to meet the needs of particular problems by allowing users to set training or testing time constraints.

Functionality is also being built into ACE that allows users to specify limits on how long the system has to arrive at a solution, with the result that ACE will initially pursue the most promising approaches, based on past experiments with similar data, and output the best approaches that it has found in the given time. This is accomplished by having ACE monitor its own performance.

ACE's incorporation of classifier ensembles has the potential to bring significantly improved classification rates to MIR research. Of course, it may be true in some cases that a pattern recognition expert could recommend a specialized solution to a given problem that is as good or better than one found experimentally by ACE. ACE is not intended to replace such experts, but rather to automatically provide good solutions relatively quickly and effortlessly to users with diverse skill levels.

ACE allows those with only a peripheral background in pattern recognition to easily perform high-quality classifications using a variety of methods. This is important, as pattern recognition experts rarely have specialized knowledge in applied fields such as music, and experts in applied fields rarely have expertise in pattern recognition. ACE makes sophisticated pattern recognition accessible to all MIR researchers. ACE also provides an excellent tool for those with more pattern recognition experience who wish to perform benchmarking comparisons of new approaches.

Much like Weka itself, ACE includes several interfaces for users with different needs. The first way to use ACE is through a GUI that allows users to build taxonomies, label and manage training and testing instances, manage features, control classifier settings, carry out comparisons of classification methodologies, train and use classifiers and view results of experiments and classifications.

The second way of using ACE is through a simple command-line interface. This interface is useful for users who already have the appropriate configuration files set up and would like a quick and easy method of performing tasks such as batch processing.

The final way of using ACE is for users to directly access the ACE Java classes from their own software. ACE is entirely open source, is well documented and is implemented in an intuitive manner.

## 5 FEATURE EXTRACTION

Feature extraction is a key part of any classification task. ACE is therefore packaged with two feature extraction applications, jAudio and jSymbolic, for extracting features from audio and symbolic recordings respectively. These feature extractors are powerful, flexible and, most importantly, extensible. They are designed with the same portability and ease of use of the ACE system itself. They have also been designed with an emphasis on the importance of the logical separation of feature extractors and classifiers, and could easily be used with classification frameworks other than ACE.

Similarly, ACE is designed to work with arbitrary existing feature extraction systems that can produce ARFF or, preferably, ACE XML files. Users are free to use whatever feature extraction software they wish, and they may take advantage of ACE's portability to install ACE on whatever platform their feature extraction software already runs on. jAudio and jSymbolic are provided for users who do not already have feature extraction software or who are interested in trying powerful new tools.

jSymbolic is based on the Bodhidharma symbolic feature library (McKay 2004), the most extensive such library currently available. McEnnis et al. (2005) have published further information on jAudio.

## 6 DISTRIBUTING THE WORKLOAD

Classification techniques can be computationally intensive, especially when many features are used or there are large training sets. This issue is amplified when multiple classifiers are used. Functionality is therefore being built into ACE to allow it run trials on multiple computers in parallel in order to achieve efficient and effective reductions in execution time.

Two distributed computing systems are currently being considered for use, namely Grid Weka (Khoussainov et al. 2004) and M2K/D2K. Grid Weka has the advantage of being built directly on Weka. D2K is a powerful and well-established environment, and M2K holds great promise, but there is the drawback that D2K has certain licensing issues, as discussed in Section 2.1.

Both Grid Weka and M2K/D2K allow computation to be distributed among either multi-purpose workstations or dedicated machines, and both are compatible with a range of hardware and operating system configurations. ACE's parallel capabilities could thus be exploited by anyone with access to a typical computing lab.

Once the distributed aspect of the system is complete, a server-based sub-system will be designed that contains a coordination system and database. Although not necessary for using ACE, users may choose to dedicate a computer to this server, allowing ACE to run perpetually. The server will keep a record of 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.

## 7 BENCHMARK TESTING

Two groups of tests were performed to verify ACE's effectiveness. The first group consisted of two MIR-related tasks, namely a beat-box recognition experiment and a reproduction of a previous seven-class percussion identification experiment (Tindale et al. 2004). ACE achieved a classification success rate of 95.6% with the five-class beat-box experiment using AdaBoost. Tindale's best success rate of 94.9% was improved to 96.3% by ACE, a reduction in error rate of 27.5%.

The second set of tests involved running ACE on ten UCI datasets (Blake and Merz 1998) from a variety of research domains. The results are shown in Table 1:

**Table 1**. ACE's classification success rate on ten UCI datasets using ten-fold cross-validation compared to a published baseline (Kotsiantis and Pintelas 2004).

| Data Set | ACE's Selected Classifier | Kotsiantis Success Rate | ACE Success Rate |
|---|---|---|---|
| anneal | AdaBoost | -- | 99.6% |
| audiology | AdaBoost | -- | 85.0% |
| autos | AdaBoost | 81.7% | 86.3% |
| balance scale | Naïve Bayes | -- | 91.4% |
| diabetes | Naïve Bayes | 76.6% | 78.0% |
| ionosphere | AdaBoost | 90.7% | 94.3% |
| iris | FF Neural Net | 95.6% | 97.3% |
| labor | k-NN | 93.4% | 93.0% |
| vote | Decision Tree | 96.2% | 96.3% |
| zoo | Decision Tree | -- | 97.0% |

It can be seen that ACE performed very well, particularly given the difficulty of some of these data sets. This is emphasized by ACE's excellent performance relative to a recently published algorithm, which was itself shown to be better than a wide variety of alternative algorithms (Kotsiantis and Pintelas 2004). Although statistical uncertainty makes it impossible to claim that ACE's results are inherently superior, it does show that ACE can certainly achieve results probably as good as or better than sophisticated state-of-the-art algorithms.

What is particularly impressive is that ACE was forced to restrict each of its learning schemes to one minute or less for both training and testing on a typical PC (2.8 GHz P4). This was done in order to investigate ACE's ability to rapidly evaluate a wide variety of classifiers. Although even higher success rates could likely have been achieved with more training time, the performance achieved by ACE in this limited time demonstrates its efficiency in exploratory research.

Table 1 is also revealing in that it demonstrates that a variety of classifiers will perform best given a variety of data sets. Furthermore, AdaBoost was selected by ACE 4 times out of 10, demonstrating the efficacy of ensemble classification. These results support the appropriateness of ACE's experimental approach as well as its utilization of ensemble classification.

The demonstrated effectiveness of ACE with respect to both musical and general data is particularly encouraging given that there are still many pattern recognition schemes to be incorporated into ACE and that the use of distributed computing in the future will make the allocation of increased training times justifiable.

## 8 CONCLUSIONS

The goals of ACE and this paper can be summarized as follows:

- Highlight the limitations of traditional pattern recognition software when applied to MIR and propose and implement a number of solutions.
- Encourage experimentation with classifier ensembles in the MIR community.
- Provide portable classification software as well as MIDI and audio feature extraction software that emphasize extensibility, ease of use and effectiveness.
- Provide software that allows users to automatically perform experiments with various classifiers, classifier parameters, data reduction techniques, ensemble architectures and ensemble parameters in order to find approaches well suited to particular problems.

## 9 FUTURE RESEARCH

Aside from the plans to incorporate distributed processing into ACE, as discussed in Section 6, there are a number of other future improvements planned. These include the implementation of learning schemes important to MIR that are currently missing from Weka, such as hidden Markov models. The inclusion of classifiers with memory (e.g., neural networks with feedback) is also an important area for expansion, as these can play an important role in music research.

There are also plans to implement modules for facilitating post-processing. The implementation of a tool for generating model classification files is another priority.

An additional goal is to strengthen ACE's support of Weka's unsupervised learning functionality. It would also be beneficial to include tools for constructing blackboard systems, in particular ones that integrate knowledge sources based on heuristics. This would complement ACE's machine learning approach nicely.

## ACKNOWLEDGEMENTS

The generous financial support from the *Social Sciences and Humanities Research Council of Canada*, the *Centre for Interdisciplinary Research in Music, Media and Technology (CIRMMT)* and the *McGill Alma Mater Fund* is greatly appreciated.

## REFERENCES

Aucouturier, J., and F. Pachet. 2004. Improving Timbre Similarity: How high is the sky? *Journal of Negative Results in Speech and Audio Sciences* 1(1).

Blake, C., and C. Merz. 1998. *UCI repository of machine learning databases*. Retrieved April 13, 2005, from www.ics.uci.edu/~mlearn/MLRepository.html. University of California, Irvine, Department of Information and Computer Sciences.

Dietterich, T. G. 2000. Ensemble methods in machine learning. In *Multiple classifier systems,* J. Kittler and F. Roli eds. New York: Springer.

Downie, J. S. 2004. International music information retrieval systems evaluation laboratory (IMIRSEL): Introducing D2K and M2K. *Demo Handout at the 2004 International Conference on Music Information Retrieval.*

Fiebrink, R., C. McKay and I. Fujinaga. 2005. Combining D2K and JGAP for efficient feature weighting for classification tasks in music information retrieval. *Proceedings of the 2005 International Conference on Music Information Retrieval.*

Freund, Y., and R. E. Schapire. 1996. Experiments with a new boosting algorithm. *Proceedings of the International Conference on Machine Learning*. 148–56.

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.

Khoussainov, R., X. Zuo, and N. Kushmerick. 2004. Grid-enabled Weka: A toolkit for machine learning on the grid. *ERCIM News* 59.

Kittler, J. 2000. A framework for classifier fusion: Is it still needed? *Proceedings of the Joint IAPR International Workshops on Advances in Pattern Recognition*. 45–56.

Kotsiantis, S., and P. Pintelas. 2004. Selective voting. *Proceedings of the International Conference on Intelligent Systems Design and Applications*. 397–402.

Kuncheva, L. 2004. *Combining pattern classifiers.* Hoboken, NJ: Wiley.

McEnnis, D., C. McKay, I. Fujinaga, and P. Depalle. 2005. jAudio: A feature extraction library. *Proceedings of the 2005 International Conference on Music Information Retrieval.*

McKay, C. 2004. Automatic genre classification of MIDI recordings. *M.A. Thesis*. McGill University, Canada.

Tindale, A., A. Kapur, G. Tzanetakis, and I. Fujinaga. 2004. Retrieval of percussion gestures using timbre classification techniques. *Proceedings of the International Conference on Music Information Retrieval.* 541–4.

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

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