The 3AD Framework:

Adaptive, Automated, Audio Description.

Anthony Theocharis

Matthew Pierce

Abstract:

There have been many attempts to collect and organize data about audio files for cataloguing, searching, and recommendation systems. Most existing systems are limited in scope or require a large amount of human input. This report describes a system that will use adaptive learning to generate metadata based on trends of user-inputted metadata, thereby reducing the need for human input as the dataset grows. This system will make the process of cataloguing audio files more efficient.

Introduction and Motivation:

Current audio classification systems such as last.fm and Pandora have provided a reasonably reliable method of cataloguing music. These programs use social-network style "tagging" methods, to add meaningful metadata to musical works, in order for the works to be referenced by the programs in useful ways.

While this method can prove effective over time, the reliance on human generated descriptors means that any new piece added to a collection is unusable until metadata is manually entered.

The motivation, therefore, behind 3AD is to create an automated computer system that generates human-usable audio descriptors. These descriptors could be used to serve the same function as the human generated tags, however would increase the usefulness of the system because each item in the collection would contain useful descriptors immediately upon entry into the collection.

Related Work:

Richard Jones' Audioscrobbler/Last.fm - works by taking user input and tracking the preferences of individual users. A user is presented with a song (via a custom playlist, or on a last.fm radio station) and they rate how much they like the song. The user's profile of likes and dislikes is then compared to other users, and suggestions are made. The Last.fm system allows users to add descriptive tags to tracks.

Music Genome Project / Pandora - Also takes user input. It attempts to create a "genomic" fingerprint of each track. The track is evaluated on 150 to 500 characteristics or "genes", each rated from 1 to 5. Example genes include “Breathy Female Vocal”, “Breathy Male Lead Vocalist”, “Brisk Swing Feel”, “Bumpin' Kick Sound”, “Busy Acoustic Hihat”. Each song is thus separated from other songs in 500-space, and the calculated distance represents the similarity of two pieces of music. All genes are evaluated by one or more expert listeners in a 20-30 minute process.

In the Mood (Charlotte Curtis) – a rhythmbox plug-in that uses Marsyas to create vectors in the same vein as Pandora’s genomes.

Marsyas – provides a generic, open platform for music information retrieval. It is completely open source, and currently under development by groups at a number of universities. Marsyas is aimed at MIR specialists, but has been instrumental in the creation of many end-user tools for applications involving MIR.

Problem Formulation:

There currently exist tools that compare pieces of music based on user-supplied characteristics, and tools that analyze pieces of music for arbitrary attributes. Systems like last.fm make it easy for end-users to categorize audio. The system improves with the number of users and the number of categorizations made by those users, but new audio files cannot be used by the system until a user has specifically categorized them.

Systems like Pandora make it easy for end-users to use descriptions provided by music experts. This system, too, however, has the drawback that new audio cannot be used until an expert has spent valuable time to categorize them. At 20 minutes per audio file, at most one person could categorize 24 files in an eight-hour work day.

Systems like Marsyas make it easy for individuals who are experts in both music and programming to decipher characteristics of audio. However, the system is completely inaccessible to the layperson, and, as yet, any practical application involves the development of an analysis technique and implementation of a new interface that uses that technique.

Systems like In The Mood provide a friendly user interface for systems like Marsyas, enabling end users to easily make use of automatically categorized audio, but such systems are few, and largely undeveloped. In The Mood, for example, has no official release, and uses only a few rudimentary analysis techniques.

There is an unfilled need for a system that brings these concepts together.

The desired system would allow developers to implement analysis techniques independent of music experts; music experts to categorize audio independent of software developers; end users to enter audio into the system and have it automatically categorized.

The ideal system would use multiple, pluggable analysis techniques to create abstract analyses of music, and would perform and store the results of these analyses when an end user enters audio into the system. It would then match the expert-supplied categorizations to trends in the automatic analyses, so that categorizations could be automatically generated in future.

The ideal system would become more accurate as more users added categorizations, and as more developers added analysis techniques. The need for expert input would thus wane as the number of analyses and expert categorizations grew.

Description:

The 3AD project will be divided into five basic phases:

1. Construct the framework and Heuristic Generator
2. Collect simple audio samples
3. Categorize audio samples
4. Collect a suite of analysis methods
5. Test the Heuristic Generator and improve it.

The framework will be constructed in modules:

- Analysis Framework
  This will support and include an arbitrary number of plug-ins, each of which will be responsible for performing one form of analysis on the audio files. The initial plug-ins will be wrappers for Marsyas functions/scripts. Plug-ins will be responsible for deciphering their own data formats. Information about plug-in data will be provided to the rest of the application by the plug-in via a standardized plug-in API.
- Database Wrapper
  The project will use an SQL database to store the information, but individual components need not bother with those details. The database wrapper will provide an object oriented description and relationship map of analyses, audio files and descriptors.
- Meta-Analysis Framework
  This will take the information from the Analysis Framework, and the information from the User Descriptors and find trends. From these it will generate heuristics to generate tags for new audio. This is the brain of the project.
- Audio Descriptor Generation System
  This will use the information from the meta-analysis framework to generate tags for audio files.
- User Interface
  Eventually the UI will be developed to include, in a user friendly format:
    - Tagging/Categorizing Audio
    - Entering new audio files
    - Selecting which plug-ins to use
    - Searching/retrieving audio

The language of choice for this project is Python. There are already frameworks such as Marsyas and Echonest that provide Python APIs, and it is a reasonably accessible and portable language.

Categorization of audio samples may make use of online services such as last.fm as well as human generated descriptions. In this instance this base data will be from the developers, but in future more experts could be involved in creating categorizations.

We will collect our audio samples from free music databases, as well as our own personal collections.

The first iteration of this software aims to provide the basic framework, and a few simple plug-ins and heuristics.

Timeline:

Nov 1

- Initial plug-ins developed (2)
- Work begun on Database Wrapper

Nov 15

- Analysis Framework laid out (API)
- Database Wrapper finalized

Nov 30

- Command-line interface for loading audio files complete
- Analysis Framework can run all plug-ins over all audio files, and save the results to the database.
- Meta-Analysis Framework laid out; can calculate rudimentary distance maps between various plug-in data and human audio descriptors.
- Audio Descriptor Generation System laid out; can suggest audio descriptors based on the results of the Meta-Analysis Framework.

Work for the Future:

- GUI / Web front-end
- Improved heuristic generation
- More plug-ins
- Accumulate database of user-entered audio and audio descriptors

We will be working closely together on all components. Within each component, implementations will be divided by feature. Anthony will guide the development, and Matt will take care of most of the resource and tag acquisition and generation. Otherwise development will be split evenly on a feature-by-feature basis.

Online Resources

http://developer.echonest.com/

http://undamped.blogspot.com/2008/07/version-01.html

http://code.google.com/p/rhythmbox-predictive-playback/

http://www.last.fm/

http://www.pandora.com/

http://en.wikipedia.org/wiki/Music_Genome_Project

http://marsyas.sness.net/

Project Proposal Phase II

Progress Report:

At this point in the development, we're just about where we expected to be.

Algorithm

The framework will make use of a variable number of plug-ins in order to characterize audio. Each plug-in will have a compute() method that takes a single WAV file and outputs a list of floats (vector in M-space).

Each audio file will then have a vector in N-space (the concatenation of all plug-in vectors for that file) that defines its characteristics.

We will seed the system with a number of starting tags. When a tag is applied to a file, an N-space vector is calculated for that tag, based on the average position of all the audio files with that tag.

When the system is later asked to identify audio, it will compare the N-space location of the new file to the N-space location of each tag, and classify it accordingly.

These last two items have yet to be implemented.

User Interface

We have currently implemented the script to add a new plug-in file to the database, so that the application can use it, and implemented a number of tests, to ensure that the components work together properly.

It still remains to create a script that imports an entire directory of audio files.

It still remains to create a script that applies pre-defined tags to all audio files (though this is trivial, given the currently developed API).

It still remains to create a script that, on demand, generates tags based on their associated vectors.

It still remains to create a script that displays the associated (generated and pre-defined) tags for each audio file.

Project Layout

We have laid out the Object Relational Map for the database, using MySQL5 and SQLAlchemy 0.4. The "model" module contains the mapping code for this, as well as the class definitions for Plugin, PluginOutput, AudioFile, Tag.

A Plugin instance has a name and a module name. It is responsible for parsing the module name and importing the appropriate plug-in module. After creation, the most important method on a Plugin object is its createVector() method. This takes as input an AudioFile instance, and returns a new PluginOutput object.

PluginOutput is a glorified result vector. Its purpose is to relate the output vector of a Plugin to the AudioFile and Plugin it is associated with.

AudioFile objects have one important property: the name of the file they are associated with. The path is relative to the top-level project path.

Tag objects are composed of a single string and a position vector. They serve to relate "textual audio descriptors" to AudioFiles. A Tag is one of the key elements of our system. It is generated by the framework, and represents the mean of the vectors of all of the AudioFiles associated with that Tag.

Plugins are one-to-many with PluginOutput.
AudioFiles are one-to-many with PluginOutput.
Tags are many-to-many with AudioFiles.

We've also written a Borg (pseudo-Singleton) class for the Database wrapper that is responsible for generating the schema and for performing queries.

We've implemented test cases to add a list of files to the database, add a list of tags to the database, add a list of plug-ins to the database, tag the files with predefined tags, and iterate over all files/plugins, saving the result vectors to the database.

Plug-ins

We decided to use our first plug-in as an introduction to Marsyas. We adapted Charlotte Curtis' GPL'd "In the Mood" plug-in for Rhythmbox to be our first plug-in. Her C++ feature extraction code was, in turn, based on code by George Tzanetakis, the author of Marsyas.

The algorithm ported fairly cleanly to Python. Due to some inconsistencies between Marsyas versions, and lacking functionality in the Python bindings (especially concerning the realvec vector class and the lack of method overloading), the resulting code is slightly more verbose. We've made use of the Numpy library to wrap the vectors generated by Marsyas. This allows us to use a more Pythonic syntax when handling them.

We plan for the next plug-in to use the algorithm from Marsyas' bextract script.

Sales Pitch:

Motivation

In the current digital audio landscape, there are many problems still to be solved, and standards yet to be developed. One important area that will certainly help shape the future for digital audio lies in audio sharing. Sharing encompasses a wide range of problems, such as where to find audio files, how to make music easily accessible to others, and what resources can be used to make audio searching and classification simple, efficient, and meaningful. That's where the 3AD Framework comes in.

Automated Audio Descriptor Generation

3AD stands for Adaptive Automated Audio Descriptor. So what does this mean? An audio descriptor is the simple, yet powerful idea behind audio search engines that are already in use today. Specifically, an audio descriptor is a textual description (usually a single word) of a piece of audio, whether music, spoken word, or otherwise, that is "attached" to the file (this is called "audio metadata"). This attached metadata allows the audio to be easily located by searching a database containing the file, using the text descriptor as the keyword. While this may not seem like a very revolutionary concept, keep in mind that with all the different ways of classifying music (by genre, instrumentation, tempo, etc.) it can become very difficult to determine what descriptions will be more meaningful than others. The beauty of the 3AD Framework is that it does all the legwork for you. Automated audio descriptors means that you feed a digital audio file into the framework to be processed, and what the framework returns is a comprehensive list of meaningful descriptors to guide the audio classification and searching process. These descriptors are generated from the underlying digital signal processing functionality, which analyzes specific characteristics of the audio itself, such as frequency spectra, rhythmic structure, timbral features, and more. Whereas other music classification and searching systems are reliant on human generated descriptors, 3AD comes up with descriptors all on its own.

So what makes the 3AD framework any better at generating meaningful descriptors than you or I? Can't anyone listen to a piece of audio and describe it to others with one or two words? Of course, this is true, however, unlike you or I, the framework has built-in knowledge of the most commonly used audio descriptions, based on a compilation of descriptors from current audio search systems (such as LastFM: www.last.fm). This means that the framework will return the most commonly used descriptors, which is key to optimizing an audio file's search potential. As an added bonus, 3AD allows you to insert your own audio descriptors, if you are unsatisfied with the list provided, or don't think it covered quite a broad enough description range.

Adaptive Framework

This brings us to the first 'A' of 3AD: Adaptive. What makes the framework adaptive, and why is that an advantage? The term comes from the adaptive-learning software embedded inside the framework. This software is the core of the descriptor generation, in

that it takes the audio analysis information and compares it against its own database of audio files and descriptors, which were used to "train" the adaptive algorithm. Not only does it use the analysis information to obtain the meaningful descriptions, but it puts each *new* piece of audio into its database with the descriptors it generates, thereby "learning" more about what types of audio are classified with what types of descriptors. The adaptive algorithm underlying this process can therefore become more robust with each new file it processes!

Applications

So who can use the 3AD Framework? Clearly it could be a useful tool for companies involved in audio searching and classification, but what about artists who want to get their music out into the digital world. 3AD bridges the gap between the two worlds so that both software professionals and musicians can begin to see standards developed in the way audio is classified between applications and networks. As an example, imagine a small web-based company that specializes in recommending music to its subscribers based on other music they have searched for (companies like this already exist). Now, imagine an artist wants to upload her music to this company's search database. The artist has given some general descriptions of her music to the company, however they aren't very highly searched terms. The company agrees to host her songs, however, she never seems to come up in any recommendation results and therefore cannot get her music out to a larger audience. This scenario, which actually exists, is called the "cold-start" scenario. Eventually it could be resolved by having dedicated subscribers to the company, who search for obscure songs and give them meaningful descriptions on a regular basis, come across the artist's music and add their descriptions. With 3AD, however, this scenario could be avoided completely. Having the software within the company means that whenever a new file is uploaded, whether it has descriptor metadata embedded already or not, it can be run through the framework and be given more standardized, highly searchable descriptors. From the artist's perspective, if they are hosting their own searchable database, hosting it with a company not running 3AD, or simply wishing to have more control over the metadata content attached to their music, they can run their entire audio collection through the framework. This may even give them better ideas of how to promote their music outside of the digital domain.

Extensibility

One can already see how the structure of the 3AD Framework facilitates greater community interaction and support amongst musicians and software/web-developers. In furthering this idea, the framework uses an open-source plug-in based design that allows anyone to become a 3AD developer. Because of the uncertainty of the current digital audio world, 3AD was built as an open framework that provides the foundations for automated audio descriptor generation, but allows for collaborative design efforts via an open-ended plug-in library. There are countless ways with which to describe pieces of audio, and we probably haven't thought of them all. Not only will creating new plug-ins expand the range of descriptors that can be generated, it will enhance the adaptive-learning software with each new plug-in. Furthermore, this type of software lends itself to

maintaining a strong development community, which will enhance the framework even further, with future versions.

Summary

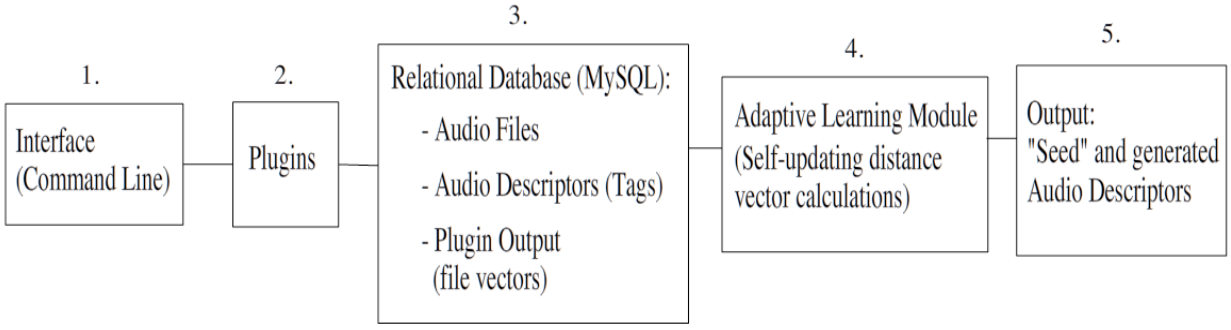
The 3AD Framework is lightweight, adaptable, and extensible. These aspects make it a great resource for companies and individuals, working with audio classification or search-based systems, who want to move forward in the uncertain field of digital audio. With a backend that utilizes a number of valuable digital signal processing techniques and adaptive learning algorithms, the framework is a powerful analysis tool that will just get better with time.

Phase III: Final Report

This report outlines the functionality of the current system. Though the framework is designed to be extensible, so as to encourage future development, we will discuss first the contributions made by the two original developers (Pierce and Theocharis), at the time of submission (December 2008).

Framework Overview

The 3AD Framework is conceptually divided into a number of modules, each tasked with its own, well-defined functionality. A block diagram of the system is shown below:



1. Currently, the 3AD Framework can be run on the command line via a number of Python scripts (detailed below). The interface provides a way to:
    a. create the database tables (init_database.py)
    b. initialize the starting plug-ins, audio files and tags (init_database.py)
    c. input single files to be processed and/or given descriptors (add_file.py)
    d. calculate (and print to the terminal) generated tags for one or all files within a user-defined tolerance zone (generate_tags.py)

2. Plugins are used to generate vectors for audio files in the framework's database. A plugin's calculations are intended to be based on a single audio processing function, but, by specification, a plugin need only provide a createVector() method that returns a list of floats. The final vector of an audio file is a concatenation of all plugin vectors for that file.

    The current version of 3AD has three plugins:
    a. charlotte(), a timbral feature extractor based on "In the Mood", a Rhythm Box plugin developed by Charlotte Curtis
    b. centroid_plugin(), a centroid calculation plugin which takes the average over the entire piece of audio
    c. bextract_plugin(), which extracts both timbral and textural features.

3. The MySQL database stores the filenames of all audio files and the audio descriptors and the current mappings between them. The database also stores the most recent output vectors for all audio files.

4. The 3AD Framework calculates audio descriptor vectors by taking the mean vector of all audio files tagged with that descriptor. These values are updated every time a new file vector is generated, so that a distance calculation between the file and the updated audio descriptor vectors can be made.

   As new descriptors and new file vectors are added into the system (by the addition of plugins or by the addition of new files) the descriptor vector calculation becomes more and more accurate, thereby adaptively improving the framework.

5. The framework generates audio descriptors for a given audio file by calculating the Euclidean distance between the audio file vector and each tag vector. If the difference is within a given tolerance range for a descriptor, then that file is "tagged" with that descriptor. The output of the 3AD Framework displays the "seed" (user provided) audio descriptors, as well as the generated audio descriptors for all processed files.

Files

The 3AD Framework was written in Python (2.5) and uses SQLAlchemy (0.4) for interacting with the database. All three of the current plugins use the Marsyas SWIG Python bindings.

The current distribution package contains the following files structure:

- controller.py: all of the main controller methods (used by the interface scripts)
- init_database.py: database initialization script
- init_vectors.py: database vector initialization script
- generate_tags.py: tag generation/regeneration script for single or multiple files that allows for filename (–f), pre-defined tag (-t), and classification tolerance value (-r) command-line flags
- model.py: model classes for interacting with the database as a collection of objects
- plugins/charlotte.py: timbral feature extraction plugin
- plugins/centroid_plugin.py: average centroid calculation plugin
- plugins/bextract.py: timbral and textural feature extraction

Summary and Auxiliary findings

The 3AD framework, in its current state, is useful for categorizing (with some level of inaccuracy) audio files from the command line.

It should run on any system that supports Marsyas, Swig, Python 2.5, NumPy, and SqlAlchemy 0.4. The underlying database (thanks to SqlAlchemy) can trivially be exchanged for any SQL based engine.

Results, somewhat unfortunately, are currently most accurate when using fewer plugins. We believe this to be a reflection of the current method of vector concatenation (it is much too simple, relying on the plugins themselves to normalize their data). Future work on this project should include framework-enforced vector normalization for plugin outputs.

In its current state, the framework's "adaptive learning module" consists of a simple Euclidean distance calculation between audio file vectors and constantly updated tag vectors. This method is simple (by design), and allowed us to more easily integrate each functional component of this initial version. Future versions of the framework, however, would benefit from using independent implementations of the adaptive learning module, each based on a separate machine-learning model, so as to allow even greater extensibility.

Working with Python was, again, an absolute joy. List comprehensions and other useful language features make expressing algorithms that work with datasets very straightforward and concise. The language's compatibility with easy, powerful GUI tools such as PyGTK could also prove very useful for future versions.

We began work on a GTK based GUI, but gave up before getting any functional results. We've designed the controller and models such that any view should be relatively trivial to implement (evidenced by the simple command line scripts we created), given a familiarity with the paradigm. While Python's GTK bindings are fairly straightforward, we just didn't have time to summit that final learning curve. We are still talking, however, about implementing a front-end as part of a Django based web application.

Some of the biggest frustrations we ran into while building 3AD involved 3rd party products. Marsyas is a daunting system in and of itself. While powerful, bugs in the compilation process followed by a steep learning curve made porting and writing the plugins a challenging, at times quite frustrating experience. Incomplete implementations of the iterator bindings of realvec objects also posed a problem. You'll see that we worked around this by wrapping Marsyas realvec objects in NumPy arrays before using them.

In general, we feel the initial version of the 3AD Framework was a success. Each component of the framework is functional, albeit at a somewhat rudimentary level. Useful results can be obtained, given the right conditions, and the framework is properly structured such that additional, more complex functionality can easily be integrated. We realized early on that the ambitious nature of the project would lead to frustration, changing expectations, and a large workload, however we are satisfied with the final results and feel we have met all our initial goals. It was certainly a good learning opportunity and something we can both draw from, moving forward.