An analysis and storage system for music research datasets

Alastair Porter
Music Technology Group
Universitat Pompeu Fabra
Barcelona, Spain
alastair.porter@upf.edu

Xavier Serra
Music Technology Group
Universitat Pompeu Fabra
Barcelona, Spain
xavier.serra@upf.edu

ABSTRACT

We present a workflow processing and data storage system that has been developed to store the computational analysis of large databases of music-related documents. Documents can consist of any music-related data, for example, audio files or symbolic scores. The system can be used to run feature extraction algorithms to compute new data based on these input documents. The feature extraction process can be distributed over many networked computers to reduce calculation time. Researchers can develop and run their own algorithms on a collection of music-related documents using any programming system without needing system administrator knowledge or direct access to the files, using a web based interface. Computed features are stored in the system along with a reference to the version of the code that was used to compute it. Audio files and computed features can be accessed over the internet. This system has been developed as a supporting part of the CompMusic project, allowing collaborating researchers to develop new feature extraction algorithms and share them with other researchers. It has been released under an open-source licence.

1. INTRODUCTION

Many tasks in music information retrieval (MIR) and signal processing involve taking some input and computing descriptive features that describe it. Often this input is audio, and the resulting output is some kind of numeric or class value, varying from a single value such as the key of a song, to a time series of values such as the instantaneous pitch of the lead musician. Keeping audio and extracted features organised is something that is often neglected in MIR research, for example, haphazardly naming results of different versions of algorithms and audio in directories suffixed with _v1 and _v2. We believe that it is important to keep better track of audio, algorithms, and results. This includes referring to audio with stable identifiers which can be used by other researchers, and being able to easily identify the version of an algorithm used to compute a given feature. The creation, then, of a well organised library of audio and related extracted features is an important aspect of MIR research.

Copyright licensing issues pose a strong problem for music researchers. It is difficult and expensive to duplicate a collection of audio that another institution may have created in order to replicate their results. We believe that it would help the research community to be able to share the use of their audio collections without actually copying audio between different institutions.

In this paper we present a system we have developed to compute features from a large database of audio files and other related musical files. This system is a part of the Dunya application [5] being developed in the CompMusic project [6] to showcase the research being produced by this project. The computation system is used to extract high level features from files and make the results available to researchers. Computation of features can be distributed over many machines. Extracted features are stored on disk and associated metadata is stored in a database. An API allows software applications to programmatically access the extracted features of a file using the file's unique identifier. A web-based user interface lets researchers easily update feature extractors and run them over input files, distributing the computation over many machines if necessary. Physical console access to these machines is not needed, meaning that researchers do not need to follow complex instructions to install and run the algorithms, and means that access can be given to researchers from other institutions without needing to give them full access to the database or other systems.

Dunya and the analysis system described in this paper are free software, released under the Affero general public license, and are available at `https://github.com/MTG/dunya`.

2. MOTIVATION AND RELATED WORK

Distributed processing of MIR features has been proposed before. For example, Marsyas has a mode to distribute processing of audio over a cluster of machines [2]. However, this system is concerned with the distributed processing of a single input. While an interesting problem, for us the execution time of an algorithm on any single file is not a problem. We are concerned with the slow-down encountered when running our algorithms on thousands of input files.

Omen [3] is a system to compute low-level audio features on audio distributed over a collection of workstations. Omen

computes low-level features on demand which can be used as building blocks for more complex feature extractors. It is designed to be distributed over idle workstations, such as those in a library. Omen uses the jAudio metadata extraction framework [4]. While this system is a useful tool in the MIR community, it prevents researchers who use different frameworks from using the tool to help with feature computation. NEMA [7] is a more complete workflow system, including a graphical workflow editor to choose datasets and algorithms to run, and has been used in the past for evaluating MIREX contests.

Our project has some specific requirements that existing systems did not completely fulfill and so instead of trying to modify one of these systems to meet our requirements we developed our own. Our primary requirement was to store and easily access data, including both original documents and extracted and generated data. Our existing algorithms have been written in a number of different programming languages and using a variety of existing research libraries. We needed a system that was agnostic to development language. Because the work is part of an ongoing research project, it was important that we could change an algorithm, quickly run it on the input again with minimal effort, and evaluate the result of the new changes. Due to the size of our audio collection it was impractical to run the analysis one file at a time when an algorithm changed, so we needed a system where we could distribute file processing over many machines. We wanted to reference audio files, other inputs, and the generated output files with an industry standard identifier that could be shared between researchers even if the audio could not be. We had added all of the audio in our data sets to the MusicBrainz database[1] and so chose to use MusicBrainz identifiers to refer to items in the database. Finally, we needed to be able to easily share the audio and the extracted features, both within our institution and to other collaborating institutions. Our collaborating institutions needed to use our audio collection without breaking the licensing conditions restricting us from copying the audio out of our institution.

3. ANALYSIS SYSTEM

Our analysis system consists of three components. First, a database contains metadata about available source files. Source files make up a data set and are used as input to feature extraction algorithms. A job execution system executes these algorithms on the source files and stores the results to disk. Finally, an API allows access to metadata about the source files and generated files, and controls access to the files by external users.

The framework is written in the Python programming language, using the django framework[2] for the web interface and database access, and celery[3] to distribute work over many worker machines. We allow researchers to write implementations of their algorithms in Python or in a compiled language such as C, which can be called from Python.

[1] `https://musicbrainz.org`
[2] `http://djangoproject.com`
[3] `http://celeryproject.org`

Our current data set consists of 2,500 audio files. Each audio file is referenced by its MusicBrainz identifier. We also store non-audio files in our database, such as musical scores, which are referenced by a work (i.e., composition) MusicBrainz identifier. We have three main feature extraction algorithms covering tonic identification, rhythm identification, and predominant melody extraction. We currently write most of our algorithms using the essentia framework [1]. Essentia provides researchers with a foundation of high quality low level feature algorithms, which are also available through Python bindings. Algorithms currently in development as part of our project are being developed in Python, C, and Matlab. We use six networked machines running four processes on each giving us the ability to process 30 files at once. This lets us run some of our more processing-intensive algorithms over the data set in one or two days, compared to the 30 or more that it would take if we were only processing one file at a time. Each machine in the cluster has access to the same data set, mounted over NFS in the same location.

3.1 Computation and distribution

Algorithms are defined by writing a Python class which inherits from a common base class. This definition includes information such as a descriptive name for the algorithm (its *type*), its version, the type of input file that it takes, and a list of outputs that the algorithm produces (called *subtypes*). For example, an algorithm could return two parts of information: a length of the processed audio, and a list of time varied outputs. Multiple versions of the same algorithm can exist at the same time. This lets researchers run many variations of the same algorithm and then compare the output of each version. Computation is performed in a method named `run`. The `run` method is passed the unique identifier of the input file and its file name. The `run` method performs whatever computation is necessary and then returns a dictionary of data representing the defined subtypes. No restriction is put on the type or format of data that is returned by the extractors.

A file is processed by using a queue system to distribute work between worker machines. A message consists of an algorithm and version, and the id of a file to be processed. A worker thread on one of the worker machines pops the message off the queue, resolves the file id to a path on disk, and creates a feature extractor object. The `run` method is called which returns the result of the computation. The output of the data is saved to disk in an NFS location which is the same on all worker machines. The worker machines have access to the same database and add an entry indicating the file, version of the extractor, and git version of the source code.

We have a web interface that lets people easily see algorithms that are registered in the system. For each algorithm the interface shows the number of files that have been processed and are waiting. You can view a list of versions of the algorithm and a list of the files that have been processed. For each subtype, a link is given to download each file.

The web interface also shows the realtime status of each worker machine in the cluster, including the number of jobs it is currently processing, and the jobs waiting in queue. Individual worker machines can be restarted or updated by

people with sufficient permissions, directly from this interface. Old versions of an algorithm can also be deleted, including the extracted files and all metadata in the database.

3.2 Updating algorithms

Algorithms are stored using the git version control system. To update an algorithm, a researcher makes their changes and increases the version number of the algorithm, before committing the code to a source code repository. A link in the web interface causes each worker machine to download an updated copy of the source repository and check which algorithms have changed. The web interface indicates that a new version of an algorithm has been installed and prompts the user to run it. When the results of the computation are written to the database, a reference to the git commit identifier is also stored, allowing inspection of the exact version of code that was used to compute a feature.

At present, dependencies of algorithms are not currently installed automatically. For now, a system administrator must install these additional requirements manually. We have started to work on an automated installation system whereby researchers can install new versions of essentia directly from the web interface.

3.3 Downloading files

We provide an HTTP REST API that allows external clients to download either the source files, or the derived files that have been created from these files. The API lets you select a file by its MusicBrainz identifier, and will return a list of all feature types and their subtypes that are available for the file. By specifying a specific type and subtype in the request a user can download a specific output file. An optional version identifier can be added to download an old version of the features. Access to the files via the API requires an authentication token, which corresponds to a user account in the system. We can use the user accounts framework to restrict access to certain files in the system. For example, we can make source files (such as audio) available only to users of a particular institution, while making derived files available to any logged in user.

4. CONCLUSIONS AND FUTURE WORK

We have presented an in-progress report of a system that we have designed to assist us in computing high-level features of audio files. The system automatically distributes processing work over a number of networked worker machines. It can store output from different versions of an algorithm and has the capability to download specific versions of features so that they can be compared. A web interface lets researchers write, update, and run algorithms without needing physical access to the worker machines and files, or needing to know how to distribute their jobs over many machines at once. Source files and processed files are referenced with a stable industry standard identifier so that they can be unambiguously referenced.

Our immediate future work involves two tasks. The first is to make it easier to write feature extraction algorithms in programming languages other than Python, such as Matlab or C. We want to provide helper methods that require researchers to only fill in a few fields, such as the path to a binary or source file. The second feature is to reference items that do not have a known Musicbrainz identifier. While our hope is that eventually all music related information could have such an identifier, we have some items in our database that currently cannot be stored in the MusicBrainz database. For now we are using randomly generated UUID to reference these items, but would like to publish these UUIDs somewhere that would let other researchers get information about an item by its identifier, or use another well known stable identifier type.

A final feature that could be useful is to use our system to distribute feature computation over servers at different institutions. Worker machines in an institution could have access to only audio which can be accessed on the site of the institution. The interface can connect to these workers over the Internet and let researchers load and run new algorithms without physical access to the machines or the audio files. Researchers could run their algorithms on far greater audio collections than if they only had their own collections. Some verification would need to be done to ensure that researchers do not abuse the system to gain access to the raw audio or a recreatable version of it. A quota system could be implement in order to ensure that institutions with large libraries are not unfairly treated due to many other institutions wanting to use their audio and resources without reciprocation.

5. ACKNOWLEDGMENTS

The CompMusic project is funded by the European Research Council under the European Union's Seventh Framework Program (ERC grant agreement 267583).

6. REFERENCES

[1] D. Bogdanov, N. Wack, E. Gómez, S. Gulati, P. Herrera, O. Mayor, G. Roma, J. Salamon, J. Zapata, and X. Serra. Essentia: an open-source library for sound and music analysis. In *Proceedings of the 21st ACM international conference on Multimedia*, pages 855–858. ACM, 2013.

[2] S. Bray and G. Tzanetakis. Distributed audio feature extraction for music. In *Proceedings of the International Conference on Music Information Retrieval*, pages 434–437, 2005.

[3] D. McEnnis, C. McKay, and I. Fujinaga. Overview of omen. In *Proceedings of the International Conference on Music Information Retrieval*, pages 7–12, 2006.

[4] D. McEnnis, C. McKay, I. Fujinaga, and P. Depalle. jaudio: Additions and improvements. In *Proceedings of the International Conference on Music Information Retrieval*, pages 385–386, 2006.

[5] A. Porter, M. Sordo, and X. Serra. Dunya: A system for browsing audio music collections exploiting cultural context. In *Proceedings of International Society for Music Information Retrieval Conference*, pages 101–106. PPGIa, PUCPR Curitiba, Brazil, 2013.

[6] X. Serra. A multicultural approach in music information research. In *Proceedings of the International Society for Music Information Retrieval Conference*, pages 151–156, 2011.

[7] K. West, A. Kumar, A. Shirk, G. Zhu, J. S. Downie, A. Ehmann, and M. Bay. The networked environment for music analysis (NEMA). In *6th World Congress on Services*, pages 314–317. IEEE, 2010.