YAAFE, AN EASY TO USE AND EFFICIENT AUDIO FEATURE EXTRACTION SOFTWARE

Benoit Mathieu, Slim Essid, Thomas Fillon, Jacques Prado, Gaël Richard
Institut Telecom, Telecom ParisTech, CNRS/LTCI
firstname.lastname@telecom-paristech.fr

ABSTRACT

Music Information Retrieval systems are commonly built on a feature extraction stage. For applications involving automatic classification (e.g. speech/music discrimination, music genre or mood recognition, ...), traditional approaches will consider a large set of audio features to be extracted on a large dataset. In some cases, this will lead to computationally intensive systems and there is, therefore, a strong need for efficient feature extraction.

In this paper, a new audio feature extraction software, YAAFE [1], is presented and compared to widely used libraries. The main advantage of YAAFE is a significantly lower complexity due to the appropriate exploitation of redundancy in the feature calculation. YAAFE remains easy to configure and each feature can be parameterized independently. Finally, the YAAFE framework and most of its core feature library are released in source code under the GNU Lesser General Public License.

1. INTRODUCTION AND RELATED WORK

Most Musical Information Retrieval (MIR) systems include an initial low-level or mid-level audio feature extraction stage. For applications involving automatic classification (e.g. speech/music discrimination, music genre or mood recognition,...), traditional approaches consider a large set of audio features to be extracted on a large dataset, possibly combined with early temporal integration [2]. The importance of the feature extraction stage therefore justifies the increasing effort of the community in this domain and a number of initiatives related to audio features extraction have emerged in the last ten years, with various objectives.

For example, Marsyas is a software framework for audio processing [1], written in C++. It is designed as a dataflow processing framework, with the advantage of efficiency and low memory usage. Various building blocks are available to build real-time applications for audio analysis, synthesis, segmentation, and classification. Marsyas is widely and successfully used for various tasks.

Note however, that the audio feature extraction (*bextract* program) is only a small component of the whole Marsyas's framework. Extracted features are written in ARFF format, and can be directly reused with the WEKA [6] machine learning toolkit. Some classic features are available out-of-the-box. The user can select which features to extract, but parameters like frame size and overlaps are global. The user also has low control upon temporal integration.

VAMP Plugins [3] is the specification of a C++ Application Programming Interface (API) for plugins allowing extraction of low level features on audio signals. The very permissive BSD-style license permits the user to develop his own plugin or application that uses existing plugins. Several plugin libraries have been developed by various research labs. VAMP Plugins comes with the Sonic Visualizer [2] application, a tool for viewing contents of music audio files together with extracted features.

Batch feature extraction using VAMP Plugins can be done with the command line tool Sonic annotator [4]. Users can declare features to extract in RDF files [5] with precise control over each feature parameter. Output can be written to CSV [6] or RDF files. Early temporal integration is limited to predefined segment summaries, and it is not possible to perform temporal integration over overlapping texture windows. VAMP Plugins API allows the development of independent libraries, but prevents the development of new plugins that would depend on already existing plugins.

Another example, the MIR toolbox, is a Matlab toolbox dedicated to musical feature extraction [3]. Algorithms are decomposed into stages, that the user can parameterize. Functions are provided with a simple and adaptive syntax. The MIR toolbox relies on the Matlab environment and therefore benefits from already existing toolboxes and built-in visualization capabilities, but suffers from memory management limitations.

Other projects also exist. jAudio [5] is a java-based audio feature extractor library, whose results are written in

[1] http://yaafe.sourceforge.net

[2] Temporal integration is the process of summarizing features values over a segment or a texture window by computing mean, standard deviation, and/or any relevant statistical function. The term *early* refers to an integration performed before the classification step.

[3] http://vamp-plugins.org/, *Queen Mary, University of London*.

[4] http://www.omras2.org/SonicAnnotator

[5] Resource Description Framework is a semantic web standard.

[6] Comma Separated Values

XML format. Maaate is a C++ toolkit that has been developed to analyze audio in the compressed frequency domain. FEAPI [4] is a plugin API similar to VAMP. MPEG7 also provides Matlab and C codes for feature extraction. Lately, MIR web services have surfaced. For instance, the Echo Nest [7] provides a web service API for audio feature extraction. Input files are submitted through the web, and the user receives a XML description.

Whatever the objectives are, the computational efficiency of the feature extraction process remains of utmost interest. It is also clear that many features share common intermediate representations, such as spectrum magnitude, signal envelope and constant-Q transform. As already observed for the VAMP plugins with the Fast Fourier Transform (FFT), performances can be drastically improved if those representations are computed only once and this especially when large feature sets are extracted. Note also that this philosophy can be extended to the different transformations (such as derivatives) of a given feature.

YAAFE has therefore been created both to get the best of the previous tools and to address their main limitations in situations where a large feature set needs to be extracted from large audio collections with different parameterizations. In particular, YAAFE has been designed with the following requirements in mind:

- Computational efficiency with an appropriate exploitation of feature calculation redundancies.
- Usage simplicity with a particular attention to the feature declaration syntax.
- Capability to process very long audio files.
- Storage efficiency and simplicity.

The paper is organized as follows: the architecture of YAAFE is detailed in section 2. A detailed benchmark is then proposed in section 3. Finally, we suggest some conclusions and future work in section 4.

2. YAAFE

2.1 Overview

YAAFE is a command line program. Figure 1 describes how YAAFE handles feature extraction. The user has to provide the audio files and a feature extraction plan. The feature extraction plan is a text file where the user declares the features to extract, their parameters and transformations (see section 2.2).

To take advantage of feature computation redundancy, YAAFE proceeds in two main stages. In a first stage, a parser analyzes the feature extraction plan in order to find common computational steps (implemented in C++ components), and a reduced dataflow graph is produced. Then in a second stage, feature extraction is applied to the given audio files according to the reduced dataflow graph and results are stored in HDF5 files (see section 2.6).

[7] `http://echonest.com/`

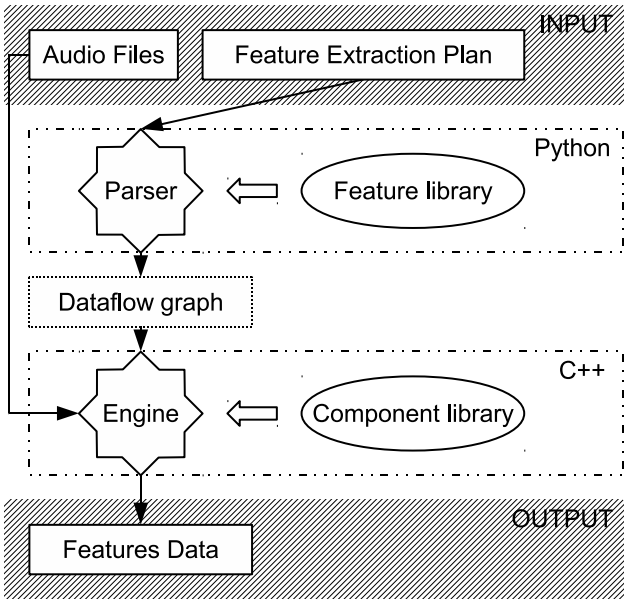

Figure 1. YAAFE internals overview.

Python is preferred to C++ for implementing the feature library and the parser, because the Python object model and reflection allow more concise and readable code to be written. The dataflow engine and the component library have been developed in the C++ language for performance.

YAAFE can be extended. Anyone can create their own extension which consists of a feature library and a component library. Provided extensions are loaded at runtime.

2.2 Feature extraction plan

2.2.1 Features

YAAFE feature extraction plan is a text file that describes the features to extract. Each line defines one feature, with the following syntax:

```
name: Feature param=value param=value
```

An example:

```
m: MFCC blockSize=1024 stepSize=512
z: ZCR blockSize=1024 stepSize=512
l: LPC LPCNbCoeffs=10
ss: SpectralSlope
```

The example above will produce 4 output datasets (see section 2.6) named *m*, *z*, *l* and *ss*, which will hold features MFCC [8], ZCR [9], LPC [10], SpectralSlope with given parameters. Missing parameters are automatically set to a predefined default value.

2.2.2 Transformations and temporal integration

One can also use spatial or temporal feature transforms, such as Derivate [11], StatisticalIntegrator [12], or SlopeInte-

[8] Mel-Frequency Cepstral Coefficients
[9] Zero Crossing Rate
[10] Linear Prediction Coefficients
[11] Derivate computes first and/or second derivatives.
[12] StatisticalIntegrator computes mean and standard deviation over the given frames.

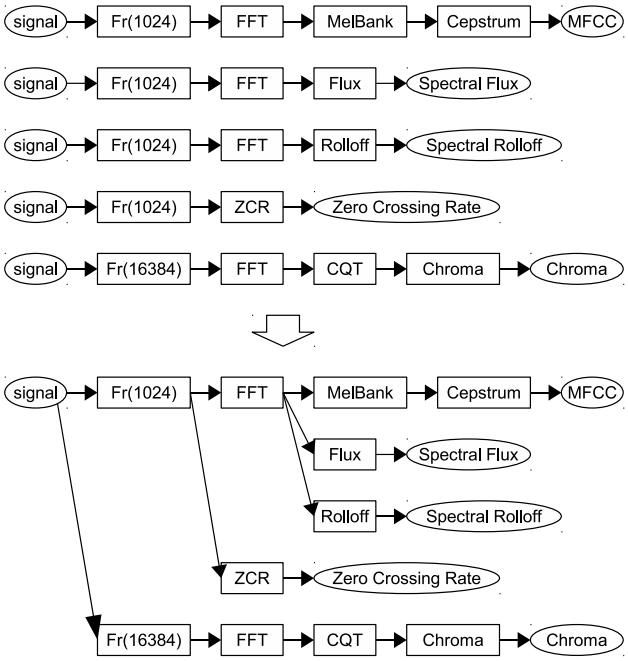

Figure 2. Automatic redundancy removal performed when parsing feature extraction plan. Fr(N) boxes are decompositions into analysis frames of size N. Step size is omitted but assumed equal.

grator [13] to enrich his feature extraction plan. For example, a plan to extract MFCCs along with derivatives and perform early integration over 60 frames will look like this:

```
m: MFCC > StatisticalIntegrator NbFrames=60
m1: MFCC > Derivate DOrder=1 ...
    > StatisticalIntegrator NbFrames=60
m2: MFCC > Derivate DOrder=2 ...
    > StatisticalIntegrator NbFrames=60
```

Obviously, *m, m1, m2* are all based on *MFCC* computation which should be computed only once. This is discussed in the next section.

2.3 Feature plan parser

Within YAAFE, each feature is defined as a sequence of computational steps. For example, MFCC is the succession of steps: Frames, FFT, MelFilterBank, Cepstrum. The same applies to feature transforms and temporal integrators.

As shown in Figure 2, the feature plan parser decomposes each declared feature into steps and groups together identical steps which have the same input into a reduced directed graph of computational steps.

The reduced graph can be dumped into a *dot* file, so an advanced user can discern how the features are really computed.

2.4 Dataflow engine

Each computational step is implemented in a C++ component which performs computation on a data block. Specific components manage audio file reading and output file writing.

The *dataflow engine* loads components, links them according to the given dataflow graph, and manages computations and data blocks. Reading, computations and writing is done block by block, so that arbitrarily long files can be processed with a low memory occupation.

[13] SlopeIntegrator computes the slope over the given frames.

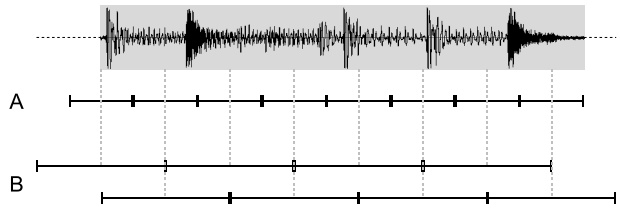

Figure 3. Temporally aligned frame decomposition for different frame sizes A and B, with same step sizes.

2.5 Feature timestamps alignment

In a feature extraction plan, each feature may have its own analysis frame size and step size. Some features require longer analysis frame sizes than others. As we intended to use YAAFE as input for classification systems, we have ensured that extracted features are temporally aligned. This is especially important with operations like the Constant-Q Transform (CQT) that may have very large analysis frames.

YAAFE addresses this issue as follows. We assume that when a feature is computed over an analysis frame, the resulting value corresponds to the time of the analysis frame center. Then, beginning with a frame centered on the signal start (left padded with zeros) ensures that all features with the same step size will be temporally aligned (see Figure 3).

A feature may also have an intrinsic time-delay. For example, when applying a derivative filter, we want the output value to be aligned with the center of the derivative filter. The design of YAAFE ensures that this is handled properly and that output features will be temporally aligned.

YAAFE only deals with equidistantly sampled features. However, some features like onsets have a natural representation which is event-based. In the current version, event-based features are represented as equidistantly sampled features for which the first dimension is a boolean value denoting the presence of an event.

2.6 Output format

YAAFE outputs results in HDF5 files [14] . Other output formats will be added in the future. The choice of the HDF5 format has initially been motivated by storage size and I/O performance. HDF5 allows for on-the-fly compression. Results are stored as double precision floating point numbers hence with no precision loss.

HDF5 is a binary format designed for efficient storage of large amounts of scientific data. HDF5 files can be read

[14] Hierarchical Data Format, `http://www.hdfgroup.org/HDF5/`

in the Matlab environment through built-in functions [15], and in the Python environment with the h5py package [16]. HDF5 files are platform independent, so they can be easily shared.

A HDF5 file can hold several datasets organized into a hierarchical structure. A dataset can be a table with several columns (or fields) of different data types, or simply a 2-D matrix of a specific data type. Attributes can be attached to datasets, an attribute has a name and a value of any data type.

YAAFE creates one HDF5 file for each input audio file. For each feature declared in the feature extraction plan, one dataset is created, with some attributes attached such as the feature definition, the frame size, the step size and the sample rate.

2.7 Availability and License

The YAAFE framework and a core feature library are released together under the GNU Lesser General Public License, so that it can freely be reused as a component of a bigger system. The core feature library contains several spectral features, Mel-Frequencies Cepstrum Coefficients, Loudness, Autocorrelation, Linear Prediction Coefficients, Octave Band Signal Intensities, OBSI ratios, amplitude modulation (tremolo and graininess description), complex domain onset detection [7], Zero Crossing Rate. Derivative and Cepstral transforms as well as statistical, slope and histogram early integrators are also provided. YAAFE is available for Linux platforms, source code can be downloaded [17].

A separate feature library will be available in binary version and for non commercial use only. It will provide Constant-Q Transform, Chromas [8], Chord detection [9], Onset detection [10], Beat histogram summary [11]. An implementation of CQT with normalization and kernels temporal synchronicity improvements [12] from reference implementation [18] is proposed.

3. BENCHMARK

We have run a small benchmark to compare YAAFE with Marsyas' bextract and Sonic Annotator. The objective is to compare the design of the three system, and not the algorithms used to compute feature. We chose few similar and well-defined features, available for the three systems for which we compared CPU time, memory occupation and output size when extracting those features on the same audio collection.

3.1 Protocol

We chose to extract the following features: MFCC (13 coefficients), spectral centroid, spectral rolloff, spectral crest factor, spectral flatness, and zero crossing rate. Features like chroma or beat detection have been avoided because the associated algorithms can be very different. In the case of Sonic Annotator, all features are available in the *Vamp libxtract plugins* [19] [13]. Early temporal integration is not computed.

We ran the feature extraction over about 40 hours of 32 KHz mono wav files (8.7 Gbs). The collection is composed of 80 radio excerpts of about 30mn each. The measures have been done on a Intel Core 2 Duo 3GHz machine, with 4 Gbs of RAM, under the Debian Lenny operating system. We checked that all systems used one core only. The RAM used has been measured with the `ps_mem.py` script [20].

We first ran the benchmark measuring the extraction of all features simultaneously. Then we ran the benchmark a second time measuring the extraction of each feature independently.

	S.A.	Marsyas	YAAFE
CPU time	52m05s	24m21s	6m34s
RAM used	14.0 Mbs	10.6 Mbs	15.5 Mbs
Output format	CSV	ARFF	HDF5
Output size	1.74 Gbs	2.7 Gbs	1.22 Gbs
Feature dim.	16	16 (32)	19

Table 1. Feature extraction with Sonic Annotator with VAMP libxtract plugins, Marsyas's bextract and YAAFE. All features are extracted simultaneously. Audio collection is 40 hours of 32 KHz mono wav files.

Feature	S.A.	Marsyas	YAAFE
MFCC	25m06s	19m28s	2m22s
Centroid	12m04s	15m42s	3m55s
Rolloff	12m11s	15m51s	3m14s
ZCR	3m41s	10m20s	0m57s
Total	53m02s	61m21s	10m28s

Table 2. CPU times for single feature extraction on the same collection as Table 1.

3.2 Results

The results are described in Table 1 and Table 2. It is important to note some differences between the 3 systems that influence the results. Firstly, we could not prevent Marsyas from performing temporal integration, so we reduced integration length to 1. Consequently, the output generated by Marsyas has 32 columns: 16 columns of feature data (mean) and 16 columns of zeros (standard deviation). This explains why Marsyas has a larger output size. Secondly, YAAFE extracts spectral spread, skewness and kurtosis together with the spectral centroid. This explains why output feature dimension is 19 for YAAFE and 16 for other systems.

Due to those differences the measures must be taken with caution. We can say that all systems performed well.

[15] See the *hdf5info*, *hdf5read* and *hdf5write* functions. YAAFE also provide useful scripts to directly load feature data into a matrix.

[16] `http://code.google.com/p/h5py/`

[17] `http://yaafe.sourceforge.net`

[18] B.Blankertz, "The Constant Q Transform", http://wwwmath.uni-muenster.de/logik/Personen/blankertz/constQ/constQ.html

[19] The VAMP libxtract plugins rely on the libxtract library: `http://libxtract.sourceforge.net/`

[20] `http://www.pixelbeat.org/scripts/ps_mem.py`

	YAAFE
CPU time	11m15s
RAM used	30.3 Mbs
Output format	HDF5
Output size	0.64 Gbs
Feature dim.	288

Table 3. Large feature set extraction with YAAFE. Audio collection is 40 hours of 32 KHz mono wav files.

They all succeed at extracting features over audio files of 30 minutes length and with low memory occupation.

The sum of single extraction times in Table 2 compared to the extraction time in Table 1 shows that Sonic Annotator does not exploit computation redundancy. The VAMP plugin API allows for computing feature in the frequency domain, but this is not done by Vamp libxtract plugins. That explains why Sonic Annotator requires more CPU time than others.

Marsyas performance clearly suffers from writing 16 column of zeros. For the evaluated task, the CPU times in Table 1 show that YAAFE tends to be faster than Marsyas.

As Sonic Annotator stored the timestamp in each output files (one per feature), and half of Marsyas' output is additional zeros, we can say that Sonic Annotator and Marsyas outputs are roughly equivalent in space. This is not a surprise as both CSV and ARFF format are text formats. Using HDF5 format, YAAFE stores more feature data, with no precision loss, using less space.

3.3 Extracting many features

YAAFE is designed for extracting a large number of features simultaneously. To check how it performs in such situation we ran YAAFE a second time under the same conditions but with a larger feature extraction plan.

In this run, we extracted MFCCs, various spectral features, loudness, loudness-based sharpness and spread, and zero crossing rate. For each feature except zero crossing rate, we added first and second derivatives. Then we performed early temporal integration by computing mean and standard deviation over sliding windows of 1 second with a step of 0.5 second. The total output dimension is 288.

The results are presented in Table 3. It should be emphasized that temporal integration is done, so the output size is much smaller than in the previous run. As a larger feature set is extracted, the dataflow graph is larger and uses more RAM. The CPU time shows that YAAFE remains very efficient in this situation.

4. CONCLUSIONS AND FUTURE WORK

In this paper, a new audio feature extraction software, YAAFE, is introduced. YAAFE is especially efficient in situations where many features are simultaneously extracted over large audio collections. To achieve this, the feature computation redundancies are appropriately exploited in a two step extraction process. First, the feature extraction plan is analyzed, each feature is decomposed into computational steps and a reduced dataflow graph is produced. Then, a dataflow engine processes computations block by block over the given audio files.

YAAFE remains easy to use. The feature extraction plan is a text file where the user can declare features to extract, transformations and early temporal integration according to a very simple syntax. YAAFE has already been used in Quaero project internal evaluation campaigns for the music/speech discrimination and musical genre recognition tasks.

Future plans include the extension of the toolbox with additional high level features such as fundamental frequency estimator, melody detection and tempo estimator and the extension to alternative output formats.

5. ACKNOWLEDGMENT

This work was done as part of the Quaero Programme, funded by OSEO, French State agency for innovation.

6. REFERENCES

[1] G. Tzanetakis, and P. Cook: "MARSYAS: A framework for audio analysis," *Org. Sound*, Vol. 4, No. 3, pp. 169-175, 1999.

[2] C. Cannam, C. Landone, M. Sandler, and J. Bello: "The Sonic Visualiser: A Visualisation Platform For Semantic Descriptiors From Musical Signals," *Proceedings of International Conference on Musical Information Retrieval*, Victoria, Canada, 2006.

[3] O. Lartillot, and P. Toiviainen: "A MATLAB TOOLBOX FOR MUSICAL FEATURE EXTRACTION FROM AUDIO," *Proceedings of the International Conference on Digital Audio Effects (DAFx'07)*, 2007.

[4] A. Lerch, G. Eisenberg, and K. Tanghe: "FEAPI, A LOW LEVEL FEATURES EXTRACTION PLUGIN API," *Proceedings of the International Conference on Digital Audio Effects*, (DAFx'05), 2005.

[5] D. McEnnis, C. McKay, I. Fujinaga, and P. Depalle: "jAudio: A feature extraction library," *Proceedings of the International Conference on Music Information Retrieval*, pp. 600–603, 2005.

[6] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, Ian H. Witten: "The WEKA Data Mining Software: An Update", *SIGKDD Explorations*, Vol. 11, Issue 1, 2009

[7] C.Duxbury et al., "Complex domain onset detection for musical signals", *Proceedings of the International Conference on Digital Audio Effects*, (DAFx'03), 2003.

[8] J.P. Bello and J. Pickens: "A Robust Mid-level Representation for Harmonic Content in Music Signals.", *Proceedings of the 6th International Conference on Music Information Retrieval*, (ISMIR-05), 2005.

[9] L.Oudre, Y.Grenier, C.Fevotte: "TEMPLATE-BASED CHORD RECOGNITION : INFLUENCE OF THE CHORD TYPES", *Proceedings of the International Conference on Music Information Retrieval*, 2009

[10] M.Alonso, G.Richard. B.David: "EXTRACTING NOTE ONSETS FROM MUSICAL RECORDINGS", *International Conference on Multimedia and Expo* (IEEE-ICME'05), 2005.

[11] G. Tzanetakis, "Musical Genre Classification of Audio Signals", *IEEE Transactions on speech and audio processing*, vol. 10, No. 5, 2002.

[12] J.Prado, "Transformée à Q constant", *technical report 2010D004*, http://service.tsi.telecom-paristech.fr/cgi-bin/valipub_download.cgi?dId=185, Institut TELECOM, TELECOM ParisTech, CNRS LTCI, 2010.

[13] J. Bullock, "Libxtract: A lightweight library for audio feature extraction," in Proceedings of the International Computer Music Conference, 2007.