

MARSYAS SUBMISSIONS TO MIREX 2009

George Tzanetakis
University of Victoria
Computer Science
gtzan@cs.uvic.ca

ABSTRACT

Marsyas is an open source software framework for audio analysis, synthesis and retrieval with specific emphasis on Music Information Retrieval. It is developed by an international team of programmers and researchers led by George Tzanetakis. In MIREX 2009 the Marsyas team participated in the following tasks: Audio Classical Composer Identification, Audio Genre Classification (Latin and Mixed), Audio Music Mood Classification, Audio Beat Tracking, Audio Onset Detection, Audio Music Similarity and Retrieval and Audio Tagging Tasks. In this abstract we describe the specific algorithmic details of our submission and provide information about how researchers can use our system using the MIREX input/output conventions on their own datasets. Also some comments on the results are provided especially highlighting the excellent running time performance of our system (an order of magnitude faster than any other submission while remaining competitive in task performance).

1 INTRODUCTION

Marsyas is an open source software framework for audio processing with specific emphasis on Music Information Retrieval (MIR). It has been around since 1999 and in 2002-2003 underwent a major restructure/rewrite (version 0.2) [9]. This version has now matured and has been progressing nicely in 2006-2009 with the addition of several new developers and finally some decent documentation. We have participated in several tasks mostly related to classification and similarity since the Music Information Retrieval Evaluation Exchange (MIREX) in 2007. This year we also submitted algorithms for automatic onset detection, beat tracking and automatic music tag annotation.

There are two main advantages of *Marsyas* compared to other solutions for building MIR systems:

- **Integration:**

Marsyas strives to support all the necessary algorithmic and software building blocks required to build full MIR systems. Frequently MIR researchers use a variety of different software systems to achieve their goal. For example MATLAB might be used

for feature extraction and WEKA might be used for machine learning/classification. There are two main problems with such non-integrated approaches. The first is reduced performance due to communication bottlenecks between each part of the process. The second which is more deep but not really utilized in our submission this year is the ability of integrated systems to combine signal processing and machine learning on several different abstraction layers and with both bottom-up and top-down processing. In contrast typically the use of non-integrated approaches follows the classic bottom-up sequential approach of feature extraction followed by classification.

- **Runtime performance:**

As most practitioners of MIR for audio signals know, it takes a lot of computation time. One of the major goals of *Marsyas* is to reduce this computation time as much as possible. Unlike many other computer applications, computation time differences in audio MIR can play an important role in the ability to conduct experiments especially over large audio collections. An experiment that completes in 30 minutes is much easier to handle compared to one that completes in 8 hours. Fast computations means that the experiment can be repeated several times to tune different parameters. Being able to process a million sound clips can result in better statistics for feature extraction than processing 100 sound clips and so on. *Marsyas* achieves fast run-time performance using a variety of different means which include: 1) a dataflow architecture that minimizes the need for memory allocation and can process audio files using large networks of computation blocks with a fixed memory footprint 2) fast, optimized C++ code for all operations 3) the ability to process large collections of audio files in one run with fixed memory footprint. Frequently other approaches to MIR operate on one file at a time adding significant computation time to start/stop a process, allocate memory etc every time a file is processed.

The main goal of our MIREX submission was to highlight these characteristics of *Marsyas* and hopefully motivate more researchers to explore the framework and contribute to it. Anyone can download the software framework, look at the corresponding code and run experiments

on their own datasets. In fact the source distribution of Marsyas includes a subdirectory named MIREX with specific detailed instructions of how to compile and run the MIREX tasks so that researchers can easily perform their own experiments on datasets as long as they follow they MIREX conventions.

For the classification, tag and retrieval tasks the selected set of features and classification approach we choose to utilize was straight-forward, well-known and most important fast to compute. Moreover, we have significant experience using these features over a large number of various audio datasets so we felt more confident about their robustness dealing with unknown audio collections. More complicated feature extractors for example based on rhythmic, pitch, and stereo information are supported at various levels of completeness in *Marsyas* but unfortunately will have to wait for next MIREX.

This year we introduced new submissions to the following tasks: audio tag annotation, audio onset detection and audio beat tracking.

2 TEAM

George Tzanetakis is the author of the abstract but several Marsyas developers participated in the development of the algorithms. Steven Ness (University of Victoria, Canada), Anthony Theocharis (University of Victoria, Canada) and Luis Gustavo Martins (Catolica University, Portugal) worked on various aspects of the automatic tag annotation submission. Fabien Gouyon (INESC Porto, Portugal), Joao Lobato Oliveira (INESC Porto, Portugal) and Luis Gustavo Martins (Catolica University, Portugal) worked on automatic beat detection while Luis Gustavo Martins worked on the audio onset detection. In addition other submissions by different teams also utilized Marsyas as part of their systems. These include the classification submissions of Hasegawa, Nishimoto, Ono, Sagayama (HNOS1-4), Rao and Kini (RK1) and Tsunoo, Tzanetakis, Ono, Sagayma (TTOS). These submissions were not coordinated by the Marsyas team and therefore are not described in the abstract.

3 SYSTEM DESCRIPTION

For all the tasks we participated we decided to represent each audio clip as a single feature vector. Even though much more elaborate audio clip representations have been proposed in the literature we like the simplicity of machine learning and similarity calculation using single feature vectors per audio clip. Coupled with a decent classifier this approach worked reasonably well compared to other much more complex ones.

The features used are Spectral Centroid, Rolloff, Flux and Mel-Frequency Cepstral Coefficients (MFCC). To capture the feature we compute a running mean and standard deviation over the past M frames:

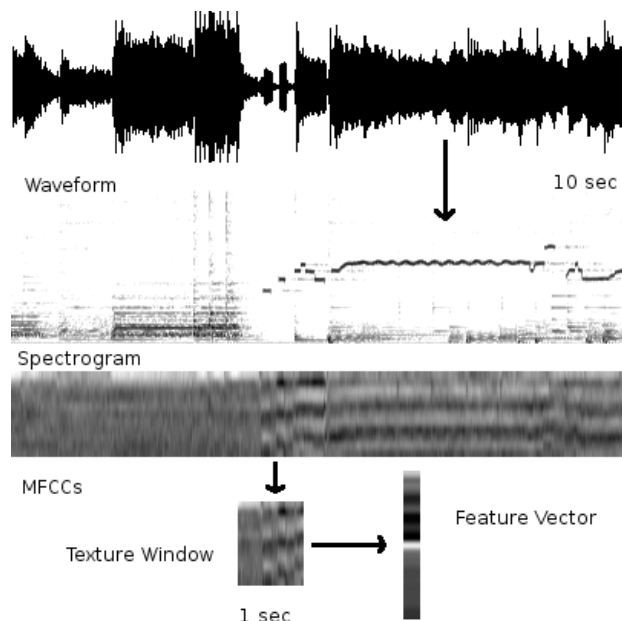


Figure 1. Feature extraction and texture window

$$m\Phi(t) = \text{mean}[\Phi(t - M + 1), \dots, \Phi(t)] \quad (1)$$

$$s\Phi(t) = \text{std}[\Phi(t - M + 1), \dots, \Phi(t)] \quad (2)$$

where $\Phi(t)$ is the original feature vector. Notice that the dynamics features are computed at the same rate as the original feature vector but depend on the past M frames (40 in our case corresponding to approximately a so called “texture window” of 1 second). This results in a feature vector of 32 dimensions at the same rate as the original 16-dimensional one. This process is illustrated in Figure 1. The sequence of feature vectors is collapsed into a single feature vector representing the entire audio clip by taking again the mean and standard deviation across the 30 seconds (the sequence of dynamics features) resulting in the final 64-dimensional feature vector per audio clip. A more detailed description of the features can be found in Tzanetakis and Cook [7].

We also submitted an algorithm that utilizes stereo panning information to improve classification tasks for which the recording/mixing process is different for the individual classes. The features are based on calculating a Stereo Panning Spectrum that has a panning value between -1 (full left) and +1 (full right) for every FFT bin.

We describe a set of features that summarize the information contained in the Stereo Panning Spectrum that can be used for automatic music classification. The main idea is to capture the amount of panning in different frequency bands as well as how it changes over time.

We define the Panning Root Mean Square for a particular frequency band as:

$$P_{l,h} = \sqrt{\frac{1}{h-l} \sum_{k=l}^h [SPS(k)]^2} \quad (3)$$

where l is the lower frequency of the band, h is the high frequency of the band, and N is the number of frequency bins. By using RMS we only consider the amount of panning without taking into account whether it is to the left or right. We consider the following 4-dimensional feature vector corresponding to an analysis window t :

$$\Phi(t) = [P_{total}(t), P_{low}(t), P_{medium}(t), P_{high}(t)] \quad (4)$$

The PRMS values correspond to overall panning (0–22050 Hz), and panning for low (0–250 Hz), medium (250–2500 Hz) and high frequencies (2500–22050 Hz) respectively.

To capture the dynamics of panning information we compute a running mean and standard deviation over the past M frames:

$$m\Phi(t) = \text{mean}[\Phi(t - M + 1), \dots, \Phi(t)] \quad (5)$$

$$s\Phi(t) = \text{std}[\Phi(t - M + 1), \dots, \Phi(t)] \quad (6)$$

This results in a 8-dimensional feature vector at the same rate as the original 4-dimensional one. For the experiments M is set to 40 corresponding to approximately 0.5 seconds. To avoid any duration effects on classification we only consider approximately the first 30 seconds of each track, resulting in a sequence of 1000 8-dimensional feature vectors for each track. The tracks are stereo, 16-bit, 44100 Hz sampling rate audio files and the STFT window size is set to 1024 samples. The sequence of feature vectors is collapsed to a single feature vector representing the entire track by taking again the mean and standard deviation across the first 30 seconds resulting in the final 16-dimensional feature vector per track. In addition the usual audio features described above are calculated separately for each channel and concatenated. More details can be found in Tzanetakis et al [8].

For the audio similarity and retrieval task once all the feature vectors (one per audio clip) have been computed the features are normalized so that the minimum of each feature is 0 and the maximum is 1 (Max/Min Normalization) and Euclidean distance over the normalized features is used for the distance matrix.

For all the classification tasks (audio classical composer identification, audio genre classification, audio music mood classification) a linear support vector machine classifier was used.

4 AUDIO TAG CLASSIFICATION

Audio tag annotation can be viewed as a problem of multi-label classification [6]. More details about our approach can be found in a recent ACM Multimedia paper [4]. Our approach is to use a distribution classifier (a linear SVM with probabilistic outputs) that can output a distribution of affinities (or probabilities) for each tag. This affinity vector can either be used directly for indexing and retrieval, or thresholded to obtain a binary vector with predicted tag associations for the particular track. The resulting affinity vector is fed into a second stage SVM classifier in order to better capture the relations between tags.

This approach is a specialized case of stacking generalization [10], a method for the combination of multiple classifiers. Similar ideas have appeared in the literature under other terms such as anchor-based classification, and semantic space retrieval, but not necessarily in a multi-label tag annotation context. The general idea is to map the content-based features to a more semantically meaningful space, frequently utilizing external information such as web resources. Stacked generalization has been used for discriminative methods for multi-label classification in text retrieval [3] but using a vector of binary predictions for each label to model dependencies between them. The most closely relevant work is applied in improving multi-label analysis of music titles again using a second stage classifier on the binary predictions of the first stage classifiers which the authors term the correction approach [5].

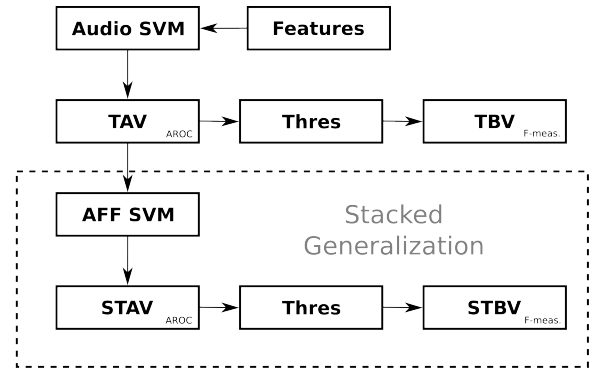


Figure 2. System flow diagram

Figure 4 shows the flow of information for our proposed audio annotation system. For each track in the audio collection a feature vector is calculated based on the audio content. As each track might be annotated by multiple tags the feature vector is fed into the multi-class Audio SVM several times with different tags. Once all tracks have been processed, the linear SVM is trained and a tag affinity output vector (TAV) is calculated. The TAV can be used directly for retrieval and storage or converted to a tag binary vector (TBV) by some thresholding method. When stacked generalization is used, the tag affinity vector (TAV) is used as a semantic feature vector for a second round of training over the tracks using an affinity SVM which produces a stacked tag affinity vector (STAV) and a stacked tag binary vector (STBV). The resulting predicted affinity and binary vector can be used to evaluate the effectiveness of the retrieval system using metrics such as Area under Receiver Operating Characteristic Curve (AROC) for the TAV and information retrieval measures for the TBV.

5 AUDIO ONSET DETECTION

The onset detection algorithm is based on a recent tutorial article [2], where a number of onset detection algorithms

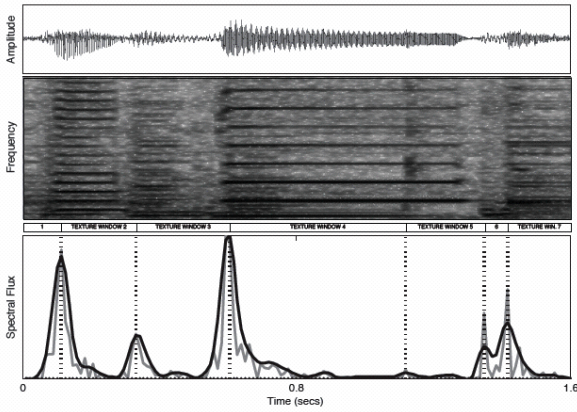


Figure 3. The top panel depicts the time domain representation of a fragment of a polyphonic jazz recording, below which is displayed its corresponding spectrogram. The bottom panel plots both the onset detection function $SF(n)$ (gray line), as well as its filtered version (black line). The automatically identified onsets are represented as vertical dotted lines.

were reviewed and compared on two datasets. Dixon concluded that the use of a spectral flux detection function for onset detection resulted in the best performance versus complexity ratio.

Following these findings our approach is based on the use of the spectral flux as the onset detection function, defined as:

$$SF(n) = \sum_{k=0}^{N/2} H(|X(n, k)| - |X(n-1, k)|) \quad (7)$$

where $H(x) = \frac{x+|x|}{2}$ is the half-wave rectifier function, $X(n, k)$ represents the k -th frequency bin of the n -th frame of the power magnitude (in dB) of the short time Fourier Transform, and N is the corresponding Hamming window size. For the experiments performed in this work a window size of 46 ms (i.e. $N = 2048$ at a sampling rate $f_s = 44100$ Hz) and a hop size of about 11ms (i.e. 512 samples at $f_s = 44100$ Hz) are used. The bottom panel of Figure 4 plots the values over time of the onset detection function $SF(n)$ for an jazz excerpt example.

The onsets are subsequently detected from the spectral flux values by a causal peak picking algorithm, where it attempts to find local maxima as follows. A peak at time $t = \frac{nH}{f_s}$ is selected as an onset if it satisfies the following conditions:

$$SF(n) \geq SF(k) \quad \forall k : n-w \leq k \leq n+w \quad (8)$$

$$SF(n) > \frac{\sum_{k=n-w}^{k=n+w} SF(k)}{mw + w + 1} \times thres + \delta \quad (9)$$

where $w = 6$ is the size of the window used to find a local maximum, $m = 4$ is a multiplier so that the mean is calculated over a larger range before the peak, $thres = 2.0$

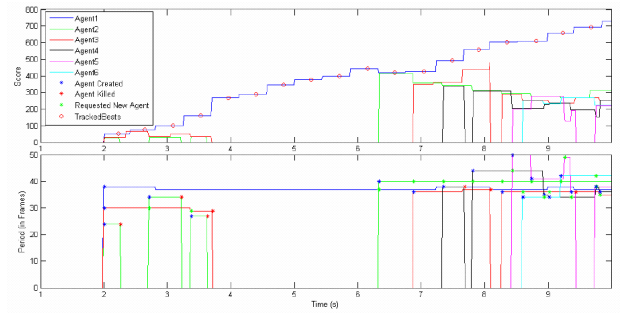


Figure 4. Agent-based Causal Beat Tracking: Graphical evolutionary view of the scores and periods of 6 BeatAgents along a 10sec. musical piece

is a threshold relative to the local mean that a peak must reach in order to be sufficiently prominent to be selected as an onset, and $\delta = 10^{-20}$ is a residual value to avoid false detections on silence regions of the signal. All these parameter values were derived from preliminary experiments using a collection of music signals with varying onset characteristics.

As a way to reduce the false detection rate, the onset detection function $SF(n)$ is smoothed (see bottom panel of Figure 4), using a Butterworth filter defined as:

$$H(z) = \frac{0.1173 + 0.2347z^{-1} + 0.1174z^{-2}}{1 - 0.8252z^{-1} + 0.2946z^{-2}} \quad (10)$$

In order to avoid phase distortion (which would shift the detected onset time away from the $SF(n)$ peak) the input data is filtered in both the forward and reverse directions. The result has precisely zero-phase distortion, the magnitude is the square of the filter's magnitude response, and the filter order is double the order of the filter specified in the equation above.

6 AUDIO BEAT TRACKING

The developed beat tracking system follows a line of two state-of-the-art algorithms: 1) a multi-agent system where different agents track beats at distinct metrical levels (periods) to find the most meaningful tempo hypothesis 2) an evaluation and guiding system to account for eventual variations along a musical piece, handling pulse period changes and short-term timing deviations. The designed approach uses a set of competitive agents to perform a causal and real-time rhythm analysis of any real musical piece. For such, an initial induction step, based on the autocorrelation (ACF) of a spectral flux window correspondent to the beginning of the music, feeds the first set of agents with their initial period, phase hypotheses pairs. Then an evaluation function guides the tracking process, by selecting the hypothesis which better fits the most relevant metrical structure (tactus) imposed by the musical events (spectral flux function), at each time. Along all procedure, the agents hypotheses are adjusted as needed, and new agents are generated in response to the evalua-

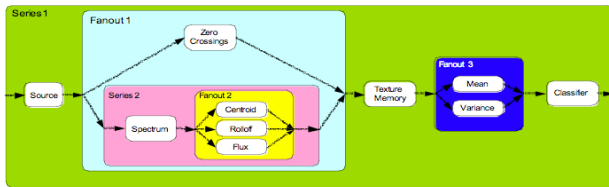


Figure 5. Feature extraction data flow network in Marsyas 0.2

tion of each agents local behaviour. Figure 4 shows the evolution of 6 beat agents for a 10-second clip.

7 IMPLEMENTATION

In this section we provide information about how to download *Marsyas* and find information for installing and using the framework as well as specific information for running the tasks we participated using the MIREX 2009 input/output conventions. We hope that providing this information will help other researchers and practitioners run our system on their own datasets and motivate them to participate in the *Marsyas* developer and user communities. *Marsyas* can be compiled under Linux, OS X (Intel and PPC), and Windows (Visual Studio, Visual Studio Express, Cygwin and MinGW).

To download *Marsyas* use the following url:

<http://www.sourceforge.net/projects/marsyas>

For information and documentation use the following url:

<http://marsyas.sourceforge.net>

System specific installation instructions are provided in the documentation. Once compiled it is straightforward to run the MIREX 2009 tasks we participated. The file MIREX/README in the Marsyas source tree contains all the necessary instructions including the exact SVN revision numbers that were used for the MIREX 2009 submission.

For classification the *Marsyas* MIREX submissions utilized a linear support vector machine trained using libsvm [1] which is directly integrated into the source code. Figure 5 shows the *Marsyas* dataflow diagram for the feature extraction that is common among all tasks.

7.1 Quick Instructions for compiling Marsyas

Quick instructions for compiling Marsyas (more detailed instructions can be found in the manual which is online at <http://marsyas.sness.net> - the instructions assume that subversion and cmake are available in the system the revision number is provided separately for each task). The last command enters the subdirectory where all the Marsyas executables reside.

```
> svn -r REVNUM co SVNPATH marsyas
> cd marsyas
> mkdir build
> cd build
```

	Rank	Marsyas	Best
Composer	15/29	51.48	62.05
Latin	22/32	52.82	74.66
Mixed	12/31	67.87	73.30
Mood	14/28	59.33	65.67

Table 1. Rank and classification accuracy for Marsyas submissions

```
> cmake ../src
> make
> cd build/bin
```

where each task has a different revision number (REVNUM).

Typically the latest revision should work for all tasks however to ensure accurate replication we record the revision used for the MIREX submission. The SVNPATH is <https://marsyas.svn.sourceforge.net/svnroot/marsyas/trunk>. See the Appendix for detailed instructions for each task.

8 DISCUSSION OF RESULTS

Overall we were pleased with the performance of the *Marsyas* submissions to MIREX 2009. In all tasks the Marsyas submissions performed reasonably well. The detailed results are available from the MIREX 2009 webpage¹ so in this section we only briefly highlight some of the evaluation results that are specific to Marsyas. The run-time results are only available for some tasks so we can not comment in detail about the superior run-time performance of Marsyas. We expect that the Marsyas submissions are significantly faster especially for the classification and similarity tasks. For example for the audio music similarity and retrieval the Marsyas submission is the only one that completes under 1 hour. Another important highlight is the excellent performance of our first entry to the automatic music tag annotation were the Marsyas submission was ranked 2nd.

9 FUTURE WORK

There is plenty of interesting future work to be explored. Now that we have the MIREX Input/Output conventions fully supported we are very excited about participating in MIREX in the future. Our submissions this year can be considered a baseline and we can only improve in the future. In no particular order here are some of the directions we would like to explore for the tasks we participated this year: more complex audio clip representations and similarities than the single vector approach, additional features (rhythm-based, pitch/chroma based, stereo panning), and better utilization of domain knowledge such as hierarchies. In addition we hope to participate in more tasks in the following years.

¹ http://www.music-ir.org/mirex2009/index.php/Main_Page

	MusicMiner	Mood
Marsyas F-Measure	0.293	0.211
Best F-Measure	0.311	0.219
Marsyas ROC	0.786	0.649
Best ROC	0.807	0.701

Table 2. Average Tag F-Measure and ROC for tag annotation tasks

	Rank	Marsyas	Best
Onsets	12/12	0.595	0.796
McKinney	8/11	0.415	0.548
Mazurka	9/11	0.321	0.678

Table 3. F-measure results for onset detection and beat tracking

Table 3 shows classification accuracy results and rankings of the best Marsyas submission for all the MIREX classification tasks. Table 2 shows the tag annotation results as average tag F-measure and average ROC for Marsyas and the best submission for this task. In addition the Marsyas tag annotation system performed quite well (2/5) in the special tagatune evaluation (68.60% compared to the best score of 70.10%).

Finally we would like to encourage other practitioners to explore and hopefully contribute to *Marsyas*. We are also happy to offer assistance to anyone interested in porting their existing systems into *Marsyas*.

10 REFERENCES

- [1] Chih-Chung Chang and Chih-Jen Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at <http://www.csie.ntu.edu.tw/~cjlin/libsvm>.
- [2] S. Dixon. Onset detection revisited. In *Proc. Int. Conf. on Digital Audio Effects (DAFx)*, 2006.
- [3] Shantanu Godbole and Sunita Sarawagi. Discriminative methods for multi-labeled classification. In *Proc. Pacific-Asia Conf. on Knowledge Discovery and Data Mining*, 2004.
- [4] S. Ness, A. Theocharis, G. Martins, L., and G. Tzanetakis. Improving automatic music tag annotation using stacked generalization of probabilistic svm outputs. In *Proc. ACM Multimedia*, 2009.
- [5] F. Pachet and P. Roy. Improving multilabel analysis of music titles: A large-scale validation of the correction approach. *Audio, Speech, and Language Processing, IEEE Transactions on*, 17(2):335–343, 2009.
- [6] G. Tsoumakas and I. Katakis. Multi label classification: An overview. *Int. Journal of Data Warehouse and Mining*, 3(3):1–13, 2007.
- [7] G. Tzanetakis and P. Cook. Musical Genre Classification of Audio Signals. *IEEE Trans. on Speech and Audio Processing*, 10(5), July 2002.
- [8] G. Tzanetakis, R. Jones, and K. McNally. Stereo panning features for classifying recording production style. In *Proc. Int. Conf. on Music Information Retrieval (ISMIR)*, 2007.
- [9] George Tzanetakis. Marsyas-0.2: a case study in implementing music information retrieval systems. In *Intelligent Music Information Systems*. IGI Global, 2007. to appear.
- [10] David H. Wolpert. Stacked generalization. *Neural Networks*, 5:241–259, 1992.

11 APPENDIX

11.1 Audio Tagging (rev 3691)

Assumes train.txt is a training list file (files and tags) and test.txt is a testing list file (just files)

Step 1) Extract features for both lists

```
> bextract -ws 1024 -as 400 -sv -fe train.txt -w train.arff -od /path/to/workdir
> bextract -ws 1024 -as 400 -sv -fe test.txt -w test.arff -od /path/to/workdir
```

These two commands will generate two files in Weka .arff format that will be placed in the working_directory specified.

Step 2) First stage automatic tag annotation

The kea command can take up to 30-50 minutes to compute and does not show any progress output until the full model is trained.

```
> kea -m tags -id /path/to_working_dir -od /path/to/workdir
-w train.arff -tc test.arff -pr stage1_affinities.txt
```

stage1_affinities.txt should contain the predicted tag affinities for the test.txt collection and stage1_predictions.txt should contain the predicted tag binary relevance file. Although the output of this stage can be directly evaluated we have found that a second stage of stacked generalization where the tag affinities of each song are used as feature vectors improves the results in most cases. In addition it generates the files stacked_train.arff and stacked_test.arff which are used for the second stage describe below.

Step3) Second stage (stacked generalization) for automatic tag annotation

```
> kea -m tags -id /path/to_workdir -od /path/to/workdir
-w stacked_train.arff -tc stacked_test.arff -pr stage2_affinities.txt
> ../../scripts/Ruby/threshold_binarization.rb train.txt
stage2_affinities.txt > stage2_predictions.txt
```

11.2 Audio Similarity(rev 3691)

Extract features:

```
> bextract -fe -sv filelist.txt -od /path/to/workdir -w marsyas_features.arff
```

(OPTIONAL) If stereo files are available and filelist contains only stereo files then additional stereo panning features can be calculated as follows:

```
> bextract -fe -sv -st filelist.txt -od /path/to/workdir -w filelist.arff
> kea -m distance_matrix -id /path/to/workdir -od /path/to/workdir -w
filelist.arff -dm filelist_matrix.txt
```

The generated filelist_matrix.txt contains the full distance matrix of all songs to all songs of filelist.txt.

11.3 Audio Classification Tasks (rev 3691)

Extract features, train classifier and predict for 1 fold:

```
./bextract -sv train.txt -tc test.txt -pr test_predicted.txt
-od /path/to/workdir -w features.arff
```

(the .arff file contains the calculated features in case anyone is interested) (OPTIONAL) If stereo files are available use the following command to extract additional features based on stereo panning information:

```
./bextract -st -sv train.txt -tc test.txt -pr test_predicted.txt
-od /path/to/workdir -w stereo_features.arff
```

The executables can be launched in parallel for each fold to take advantage of multiple cores without a problem as long as there are different scratch directories for each fold.

11.4 Audio Onset Detection (rev 3700)

```
./onsets soundExample.wav
```

Resulting soundExample.output will be saved in the same directory of the onsets executable.

11.5 Audio Beat Tracking (rev 3719)

This implementation contains two distinct versions, differentiated by its functional heuristics. The application outputs two text files: one with the tempo measure (median IBI, in BPMs) - XXX_medianTempo.txt; and other with the beat times, in seconds - XXX.txt. For running each of them just use the following commands:

```
1-> ./ibt -s "squareCorr" input.wav outputDir/output.txt
2-> ./ibt -s "regular" input.wav outputDir/output.txt
```

NOTE: If no output directory is assigned, the executable dir and the audio filename will be assumed. If only the output directory is assigned, the audio filename will be assumed.