Deliverable D5.8

Release of timbral characterisation tools for semantically annotating non-musical content

Grant agreement nr	688382
Project full title	Audio Commons: An Ecosystem for Creative Reuse of Audio Content
Project acronym	AudioCommons
Project duration	36 Months (February 2016 - January 2019)
Work package	WP5
Due date	31 January 2019 (M31)
Submission date	31 January 2019 (M31)
Report availability	Public (X), Confidential ( )
Deliverable type	Report ( ), Demonstrator ( ), Other (X)
Task leader	Surrey
Authors	Andy Pearce, Saeid Safavi, Tim Brookes, Russell Mason, Wenwu Wang, and Mark Plumbley
Document status	Draft ( ), Final (X)

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N° 688382

Table of contents

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N° 688382

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N° 688382

Executive Summary

This report describes the final release (v0.4) for the eight perceptual models that can predict the timbral characteristics of a recorded sound by analysis an audio file: *hardness*, *depth*, *brightness*, *roughness*, *warmth*, *sharpness*, *boominess*, and *reverb*. These models can be used to automatically generate metadata describing the timbral properties of recorded sounds, which can, in turn, be implemented into a search function, enabling users to filter search results based on the timbral properties.

The package has been improved with the inclusion of an extractor function that can generate all timbral metadata with a single function call. The package also contains a loudness normalisation function which does not require files to be pre-loudness normalised prior to analysis. This should make the metadata more accurate for the stimuli on freesound. Minor improvements have been made to the models to ensure that they work with a set of over 2,000 sounds from Freesound that caused crashes with the Version 0.3 implementations. All references to non-pip installable python libraries have been removed, making the distribution much easier to install.

A cut-down version of the timbral reverb model has been included as a pure python distribution, whose predictions are less accurate than those of the full model, available in MATLAB.

All current models have been made available in a public source code repository[1].

[1] https://github.com/AudioCommons/timbral_models

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N° 688382

1 Description of the models

The Audio Commons project aims to provide tools for the automatic annotation of audio content. The development of these tools is spread across work packages (WP) 4 and 5, depending on the type of properties to be annotated. WP5 aimed at providing annotation tools that describe non-musical properties; more specifically the timbral characteristics of sound effects.

This demonstrator package describes the final implementation of eight perceptual models for annotating the timbral attributes of *hardness*, *depth*, *brightness*, *roughness*, *warmth*, *sharpness*, *boominess*, and *reverb*. This distribution is in two parts: 1) the main python distribution containing all timbral attributes, with a cutdown version of the `timbral_reverb` model with poorer performance; and 2) a MATLAB and Weka distribution of the full `timbral_reverb` model.

Section 2 describes the python distribution, and Section 3 describes the MATLAB/Weka distribution.

2 Python distribution

All attributes are coded in python and are structured into a single `timbral_models` package. The `timbral_reverb` model's python implementation is a cutdown version with poorer performance due to the complexity of the full version which relies on pre-existing matlab code. The code described in this deliverable relates to Version 0.4 for all functions.

Section 2.1 below describes the installation, setup procedure, and dependencies required to run the timbral models. Section 2.2 introduces the timbral extractor function that can be used to extract all timbral attributes with a single function call. Sections 2.3 to 2.10 then describe each model in more detail, the required inputs, the outputs of the function, and provides some example code for running each model.

2.1 Installation

All python distribution models have been written to only use PyPI-based dependencies, and as such can be most simply be installed using pip installation from PyPI.

```
$ pip install timbral models
```

All Python-implemented models were tested in both Python 2.7 and 3.5. These timbral models rely on the `numpy`, `scipy`, `soundfile`, `sklearn`, `LibROSA`, `six`, and `PyLoudNorm` Python packages. If installing via pip, these dependencies should be installed automatically, or can be installed using the pip tool manually, e.g. `pip install numpy`

Models are also available from the project's GitHub page: https://github.com/AudioCommons/timbral_models. Editable versions of the functions can be installed by cloning this repository and using pip's local install function.

```
$ git clone https://github.com/AudioCommons/timbral_models
# Navigate to folder
$ pip install -e .
```

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N° 688382

All timbral models can be accessed by importing the `timbral_models` package into Python. The timbral metadata can then be calculated for specific timbral attributes or using the timbral extractor function.

Functions can either be called by giving a filename/path or given an array of audio samples. It is recommended to use the filename calling method.

2.1.2 Filename calling

The timbral extractor and all individual timbral functions can be called by giving a string that is the full path and filename of an audio file. An example of calling the timbral extractor could be:

```
import timbral_models
fname = '/Documents/Music/TestAudio.wav'
# get all timbral attributes
timbre = timbral_models.timbral_extractor(fname)

# get just the timbral depth
depth = timbral_models.timbral_depth(fname)
```

This package relies on the pysoundfile package for reading audio files. In the event that an audio file cannot be read with this package, users have the option to use their own audio reading functions and pass a numpy array to the functions as described below.

2.1.3 Audio sample calling

When calling a timbral function with audio samples, it is expected that the audio is a numpy array and is structured in the same way as would be read by pysoundfile (i.e. dimension 1 is the audio samples, and dimension 2 is the number of channels). With this calling method, the sample rate must be specified by setting the `fs` parameter. If no sample rate is given, the function will return an error.

```
import timbral_models
import soundfile as sf
fname = '/Documents/Music/TestAudio.wav'

# read audio files
audio_samples, fs = sf.read(fname)

# get all timbral attributes
timbre = timbral_models.timbral_extractor(audio_samples, fs=fs)
```

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N° 688382

2.1.4 Additional inputs

All timbral models (with the exception of `timbral_reverb`) have at least three optional inputs of `dev_output`, `phase_correction`, and `clip_output`, described in the table below.

Required Input	Input type	Description
`fname`	String/numpy array	String: Path and filename to the audio file to be analysed. numpy array: If presented with a numpy array, the model requires the samplerate be specified (fs).
`fs`	int/float	If giving a numpy array as an input, fs must be set to the sampling frequency.
Optional parameter		
`dev_output`	Bool, default to `False`	When set to `False`, the code returns the predicted timbral value as a single float. When set to `True` returns the value of each extracted feature without the regression model applied.
`phase_correction`	Bool, default to `False`	When set to True, the phase of the left and right channels are compared before summing to mono (for stereo signals only). If the correlation is less than -0.5, the phase of the right channel is inverted before summing. Note that this does not apply to timbral_reverb as the algorithm is calculated on both the left and right signals (or first two channels for higher channel counts).
`clip_output`	Bool, default to `False`	Limits the values returned from timbral models to between 0 and 100, the values used for training data. Note that this does not apply to the `timbral_reverb` model.

The following subsections give more detail about the use of each timbral model, including other optional parameters.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N° 688382

2.2 Timbral Extractor

The timbral extractor function (Version 0.4) has been designed to extract all timbral attributes with a single function call. The function can return results for each attribute as either a python dictionary, or list.

2.2.1 General use of the extractor

The `timbral_extractor` can be called from the `timbral_models` package. The function should be given a string that is the path and filename of the audio file to be analysed. For example:

```
import timbral_models
fname = '/Documents/Music/TestAudio.wav'
timbre = timbral_models.timbral_extractor(fname)
```

By default, the extractor will return the timbral attributes as a python dictionary. To return the results as a list, call the function with the optional argument `output_type='list'`. If returning a dictionary, each timbral attribute can be accessed with the names `hardness`, `depth`, `brightness`, `roughness`, `warmth`, `sharpness`, `boominess`, and `reverb`. For example:

For example:

```
# get the hardness and depth
Hardness = timbre['hardness']
Depth = timbre['depth']
```

When returning a list, the results will be in the order: `hardness`, `depth`, `brightness`, `roughness`, `warmth`, `sharpness`, `boominess`, and `reverb`.

2.2.2 Additional features of the timbral extractor

As well as the required filename, the timbral extractor has the following variable parameters.

Optional parameter	Parameter type	Description
`output_type`	String, default to `'dictionary'`	Defines the output type of the timbral extractor. Inputs can be `'dictionary'` or `'list'`.
`verbose`	Bool, default to `True`	Determines if the `timbral_extractor` prints current evaluation progress to the console. By default, this prints `'Calculating attribute...'` for each attribute. Set to `False` to not show output.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N° 688382

2.3 Timbral Hardness

The timbral hardness model (Version 0.4) is a Python implementation model. This model undergone minor improvements to performance from the previous version described in Deliverables D5.6 and D5.7. The model now includes automatic loudness normalisation, better error handling, and improved efficiency.

2.3.1 General use of the Hardness model

The `timbral_hardness` function can be called from the `timbral_models` package. The function should be given a string that is the path and filename of an audio file. For example:

```
import timbral_models
fname = '/Documents/Music/TestAudio.wav'
hardness = timbral_models.timbral_hardness(fname)
```

This will return the estimated hardness of the audio file as a single floating point value. Higher values represent the audio file as sounding more *hard*. Although the model was trained on subjective data ranging from 0 to 100, the output values can be beyond these scales due to the nature of the linear regression implemented. The `clip_output` optional parameter can be used to limit the returned values from 0 to 100.

2.3.2 Additional features of the Hardness model

As well as the required filename, the timbral hardness model has the following variable parameters. Each parameter has been set to provide the highest correlation with subjective ratings.

Optional parameter	Parameter type	Description
`max_attack_time`	Float, default to 0.1	Sets the maximum period after each onset to evaluate the attack time, in seconds.
`bandwidth_thresh_db`	Float, default to -75	Sets the threshold for estimating the bandwidth of the audio signal, in dB.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N° 688382

2.4 Timbral Depth

The `timbral_depth` model (Version 0.4) is a Python implemented model to predict the apparent depth of an audio file. Like with the Hardness model, this has undergone minor improvements from the previous version and now includes automatic loudness normalisation and updates to avoid crashes with extremely short audio files.

2.4.1 Using the Depth model

The `timbral_depth` function can be called from the `timbral_models` package. The function should be given a string that is the path and filename of an audio file. For example:

```
import timbral_models
fname = '/Documents/Music/TestAudio.wav'
depth = timbral_models.timbral_depth(fname)
```

This will return the estimated depth of the audio file as a single floating point value. Higher values represent the audio file as sounding more *deep*. Although the model was trained on subjective data ranging from 0 to 100, the output values can be beyond these scales due to the nature of the linear regression implemented. The `clip_output` optional parameter can be used to limit the returned values from 0 to 100.

2.4.2 Additional features of the Depth model

As well as the required filename, the `timbral_depth` model has the following optional parameters. Each parameter has been set to provide the highest correlation with subjective ratings.

Optional parameter	Parameter type	Description
`threshold_db`	Float, default to -60	Threshold for estimating the spectral centroid features for a given spectrogram frame, in dB.
`low_frequency_limit`	Float, default to -50	Frequency to high-pass filter the audio signal, in Hz.
`centroid_crossover_frequency`	Float, default to 2000	Frequency of the crossover for calculating the spectral centroid features, in Hz.
`ratio_crossover_frequency`	Float, default to 500	Frequency of the crossover for calculating the spectral ratio features, in Hz.
`db_decay_threshold`	Float, default to -40	Threshold for estimating the decay time of the signal, in dB.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N° 688382

2.5 Timbral Brightness

The timbral brightness model (Version 0.4) is a Python implemented model to predict the perceived brightness of an audio file. Like with the Hardness and Depth models, this has undergone minor improvements from the previous versions, including automatic loudness normalisation and updates to avoid crashes with extremely short audio files.

2.5.1 Using the Brightness model

The `timbral_brightness` function can be called from the `timbral_models` package. The function should be given a string that is the path and filename of an audio file. For example:

```
import timbral_models
fname = '/Documents/Music/TestAudio.wav'
brightness = timbral_models.timbral_brightness(fname)
```

This will return the estimated brightness of the audio file as a single floating point value. Higher values represent the audio file as sounding more *bright*. Although the model was trained on subjective data ranging from 0 to 100, the output values can be beyond these scales due to the nature of the linear regression implemented. The `clip_output` optional parameter can be used to limit the returned values from 0 to 100.

2.5.2 Additional features of the Brightness model

As well as the required filename, the timbral brightness model has the following variable parameters. Each parameter has been set to provide the highest correlation with subjective ratings.

Optional parameter	Parameter type	Description
`threshold`	Float, default to 0	Threshold below which to ignore the energy in a time window.
`ratio_crossover`	Float, default to 2000	Crossover frequency for calculating the HF energy ratio in Hz.
`centroid_crossover`	Float, default to 100	Highpass frequency for calculating the spectral centroid in Hz.
`stepSize`	Int, default to 1024	Step size for calculating spectrogram, in samples.
`blockSize`	Int, default to 2048	Block size (fft size) for calculating spectrogram.
`minFreq`	Float, default to 20	Frequency for high-pass filtering audio prior to all analysis in Hz.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N° 688382

2.6 Timbral Roughness

As with the other models, timbral roughness model (Version 0.4) has been updated from previous versions. This version includes automatic loudness normalisation, updates to avoid crashes with extremely short audio files, and linear regression to subjective ratings, meaning ratings now sit between 0 and 100.

2.6.1 Using the Roughness model

The `timbral_roughness` function can be called from the `timbral_models` package. The function should be given a string that is the path and filename of an audio file. For example:

```
import timbral_models
fname = '/Documents/Music/TestAudio.wav'
roughness = timbral_models.timbral_roughness(fname)
```

This will return the estimated roughness of the audio file as a single floating point value. Higher values represent the audio file as sounding more *rough*. Although the model was trained on subjective data ranging from 0 to 100, the output values can be beyond these scales due to the nature of the linear regression implemented. The `clip_output` optional parameter can be used to limit the returned values from 0 to 100.

2.6.2 Additional features of the Roughness model

As well as the required filename, the timbral roughness model has the following variable parameters. Each parameter has been set to provide the highest correlation with subjective ratings.

Optional parameter	Parameter type	Description
`peak_picking_threshold`	float, default to 0.01.	Sets the minimum dynamic range between peaks in a frequency spectrum for the peak picking algorithm.

2.7 Timbral Warmth

The `timbral_warmth` model (Version 0.4) has been updated from previous versions to have minor improvements of efficiency and to avoid crashes with extremely short stimuli.

2.7.1 Using the Warmth model

The `timbral_warmth` function can be called from the `timbral_models` package. The function should be given a string that is the path and filename of an audio file. For example:

```
import timbral_models
fname = '/Documents/Music/TestAudio.wav'
warmth = timbral_models.timbral_warmth(fname)
```

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N° 688382

This will return the estimated warmth of the audio file as a single floating point value. Higher values represent the audio file as sounding more '*warm*'. Although the model was trained on subjective data ranging from 0 to 100, the output values can be beyond these scales due to the nature of the linear regression implemented. The `clip_output` optional parameter can be used to limit the returned values from 0 to 100.

2.7.2 Additional features of the Warmth model

As well as the required filename, the timbral warmth model has the following variable parameters. Each parameter has been set to provide the highest correlation with subjective ratings.

Parameter name	Parameter type	Description
`max_fft_frame_size`	Int, default to 8192	Frame size for calculating spectrogram, in Hz.
`rolloff_ratio`	Float, default to 0.85	Ratio for calculating the spectral rolloff frequency, defaults to 0.85 meaning 85% of the energy sits below this frequency.
`max_WR`	Float, default to 12000	Maximum allowable warmth region frequency (3.5 times the fundamental frequency), in Hz.

2.8 Timbral Sharpness

The timbral sharpness model (Version 0.4) has been updated from the previous version to now include a linear regression to subjective ratings and automatic loudness normalisation.

2.8.1 Using the Sharpness model

The `timbral_sharpness` function can be called from the `timbral_models` package. The function should be given a string that is the path and filename of an audio file. For example:

```
import timbral_models
fname = '/Documents/Music/TestAudio.wav'
sharpness = timbral_models.timbral_sharpness(fname)
```

This will return the estimated sharpness of the audio file as a single floating point value. Higher values represent the audio file as sounding more '*sharp*'. Although the model was trained on subjective data ranging from 0 to 100, the output values can be beyond these scales due to the nature of the linear regression implemented. The `clip_output` optional parameter can be used to limit the returned values from 0 to 100.

The `timbral_sharpness` model does not have any optional parameters beyond those described in Section 1.1.2.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N° 688382

2.9 Timbral Booming

The timbral booming model (Version 0.4) has been updated from the previous version to now include a linear regression to subjective ratings and automatic loudness normalisation.

2.9.1 Using the Booming model

The `timbral_booming` function can be called from the `timbral_models` package. The function should be given a string that is the path and filename of an audio file. For example:

```
import timbral_models
fname = '/Documents/Music/TestAudio.wav'
boominess = timbral_models.timbral_booming(fname)
```

This will return the estimated boominess of the audio file as a single floating point value. Higher values represent the audio file as sounding more '*boomy*'. Although the model was trained on subjective data ranging from 0 to 100, the output values can be beyond these scales due to the nature of the linear regression implemented. The `clip_output` optional parameter can be used to limit the returned values from 0 to 100.

The `timbral_booming` model does not have any optional parameters beyond those described in Section 1.1.2.

2.10 Timbral Reverb

As described in Section 1, the python implementation of the `timbral_reverb` model is a cutdown version of the full MATLAB and Weka implementation (described in Section 3).

Version 0.4 is the first release of the python version of the timbral reverb model (version numbering is for consistency with other timbral models). Unlike all previously described models, this is a classification model based from a logistic regression. Therefore, the `clip_output` parameter does not affect this model.

For files with two or more channels, the algorithm is calculated on the first two audio channels and the mean taken. As such the `phase_correction` parameter does not affect the model.

1.10.1 Using the Reverb model

The `timbral_reverb` function can be called from the `timbral_models` package. The function should be given a string that is the path and filename of an audio file. For example:

```
import timbral_models
fname = '/Documents/Music/TestAudio.wav'
reverb = timbral_models.timbral_reverb(fname)
```

This will return the estimated classification of the audio file as either 0 (indicating the file does not sound reverberant), or 1 (indicating that the file sounds reverberant).

The `timbral_reverb` model does not have any optional parameters.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N° 688382

3 MATLAB/Weka implementation

The full version of the `timbral_reverb` model is implemented with feature extraction in MATLAB and modelling in Weka. The model is implemented as a classification model categorising the sound files into 'does not sound reverberant' and sounds 'reverberant' categories. The current model better fits with subjective ratings of the perceived level of reverberation compared to previous implementations and the python distribution version.

New features have been extracted from the audio files to predict the perceived level of reverberance. Like the Deliverable D5.2 report, one of extracted features is RT60, but the extraction algorithm has been improved from what was proposed initially in D5.2. The new approach for extracting RT60 is based on blind estimation, meaning that it uses an audio signal as an input, rather than a room impulse response. This approach is based on the Laplacian model and the detail of this algorithm are given in [Jan and Wang, 2012]. This is the algorithm the python distribution is based on.

In addition to RT60, the following features are also extracted and combined as an input for the machine learning modelling approaches:

- Level of foreground stream
- Level of background stream
- Interaural time difference (ITD) fluctuation in the foreground
- ITD fluctuation in the background
- Level of the low-frequency part of the spectrum
- Reverberance
- Clarity
- Apparent source width
- Listener Envelopment

These extra features are extracted as described in [Schuitman and Vries, 2013]. Schuitman et al. propose a method for derivation of signal based measures aiming at predicting aspects of room acoustic perception from content specific signal representations produced by a nonlinear model of the human auditory system. This approach simulates multiple aspects of human hearing to resemble human auditory perception.

3.1 Installation

MATLAB and Weka should both be installed according to the developers recommendations. The MATLAB functions required to run this version can be obtained from : https://github.com/AudioCommons/timbral_models. The folder 'MATLAB/Weka Timbral Reverb' should be downloaded and the matlab search path set to look in this directory and all sub directories.

3.2 Using the Reverb model

The reverb model can be run by running the script `Reverb_AC_SS_V4`. In this script, several parameters need to be changed prior to execution: the path to the audio files (Line 5); the filename and location to save the output .arff file containing the extracted features (line 8); location of the ".arff" file which contains features of test audio files; the location where Weka is installed (Line 39)); and finally the location and name of the trained reverberation model (line 40). For example:

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N° 688382

```
Run Reverb_AC_SS_V1.m
Line 5: cd('\\surrey.ac.uk/audiofiles\');  % location of the audio files
Line  8:  fid  =  fopen('ExtractedFeatures.txt','w'); % file name for storing extracted
features
Line 39: cd('C:\Program Files\Weka-3-8');  % location of Weka installation
Line 40: -l Logistic_classifier_Weka_D5.6.model -T
         C:\Users\Fred\Desktop\temp\outputfile.arff  % doc command for computing model
```

By running the main script an explorer will be opened in the specified audio file location (the directory defined at line 5 of the main script). The selected audio files (multiple audio files can be selected simultaneously) will then be analysed.

Extracted features for the selected audio files will be stored in the comma separated format and in a single ".arff" file, specified in line 8. Extracted features are then automatically aligned and reformatted for use in Weka. Figure 1 below shows an example of how the relationships between the attributes and data are defined: Line 1 defines the relation between the features; and lines 2 to 14 define the attributes. The extracted features for each audio file are stored from line 16 onwards. By running the main MATLAB script the ".arff" file is generated for all the selected audio test files automatically.

```
1  @relation RTlandRTRandBLandsREVandFLandpClarandLlowandITDfandpASWandITDbandpLEVandsLEV
2  @attribute RTL numeric
3  @attribute RTR numeric
4  @attribute BL numeric
5  @attribute sREV numeric
6  @attribute FL numeric
7  @attribute pClar numeric
8  @attribute Llow numeric
9  @attribute ITDf numeric
10 @attribute pASW numeric
11 @attribute ITDb numeric
12 @attribute pLEV numeric
13 @attribute sLEV numeric
14 @attribute reverb {L, H}
15
16 @data
17 9.263610e-01,9.233524e-01,2.082710e+01,4.703869e-01,2.721431e+01,1.306678e+00,4.192304e+01,3.446164e-01,6.126307e+00,3.866359e-01,5.994641e+00,9.883385e-01,?
18 8.894703e-01,8.394163e-01,2.376970e+01,6.202309e-01,1.425087e+01,5.995392e-01,4.186177e+01,2.866721e-01,6.045132e+00,4.147940e-01,6.106387e+00,9.897530e-01,?
19 9.663063e-01,9.627961e-01,1.450386e+01,1.934884e-01,1.630286e+01,1.124036e+00,2.844782e+01,3.460880e-01,5.858654e+00,3.816900e-01,5.814528e+00,9.856422e-01,?
20 6.765210e-01,9.013081e-01,1.583651e+01,2.401891e-01,3.550238e+01,2.241806e+00,2.942647e+01,3.201429e-01,5.844384e+00,3.913158e-01,5.862126e+00,9.864093e-01,?
```

Figure 1 - Example of the .arff file generated.

To identify the prediction output, two class labels were defined during the training phase: labelled with L or H. These labels are came from the human listening tests, in which L means the audio file has low level of reverberation and H means high level.

The Reverberation model is trained using 420 audio files from multiple source types, and simple logistic classifier (Weka logistic classifier with the default predefined parameters) is used for modelling. Details about this classifier could be found in [Le Cessie and van Houwelingen, 1992].

The pre-trained reverberation model and generated ".arff" files are then used by Weka to estimate the reverberation class. Weka could be used via its own graphic interface or from the Command Prompt environment. In this report, to make it simple for evaluation, the Weka prediction is integrated in the MATLAB script. The output of the script will have a format as shown in Figure 2.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N° 688382

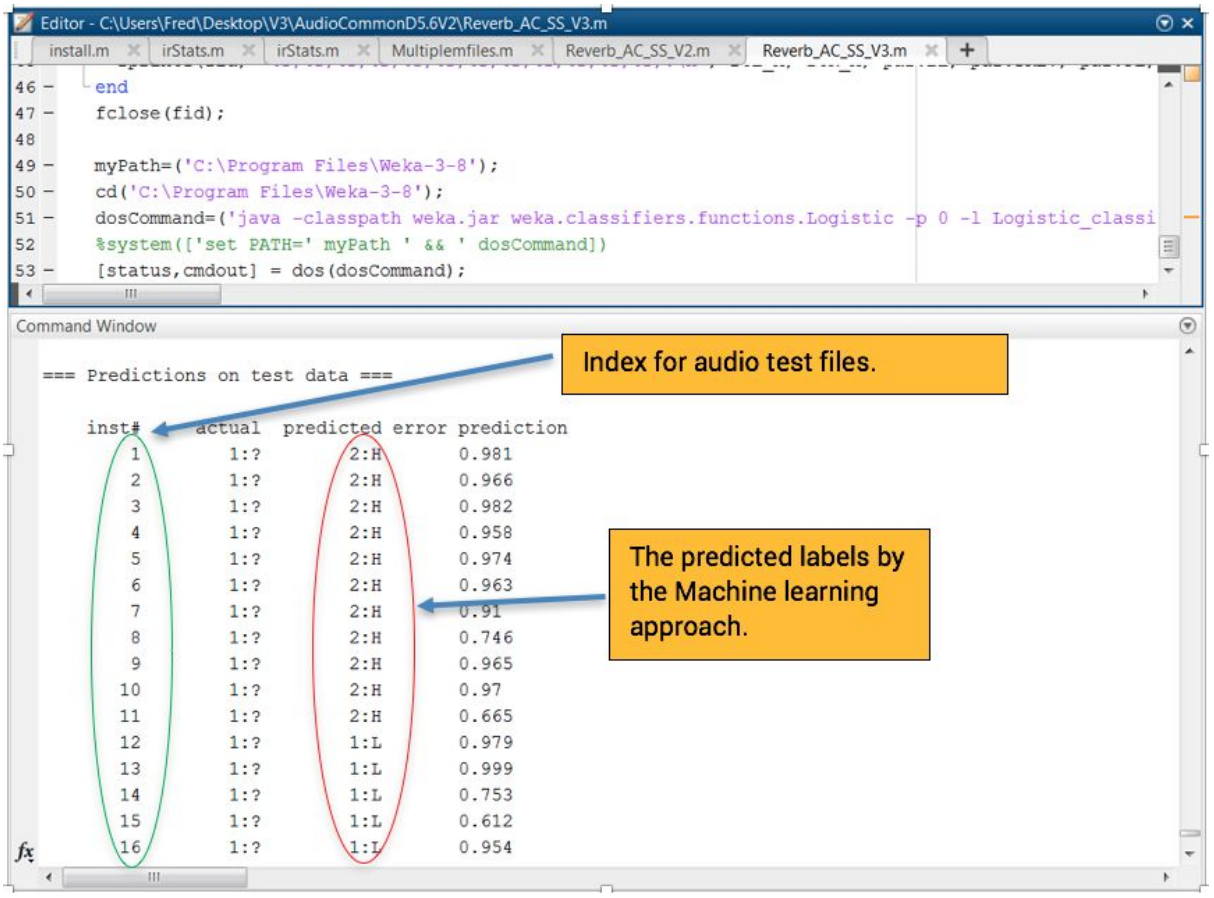

Figure 2 - Example of the output from the Reverb_AC_SS_V4.m script.

In addition to test index and predicted labels there also is a column which shows the probability of this prediction based on the predefined model. Full description of this approach and the obtained results described fully in [Safavi, Pearce, Wang, and Plumbley, 2018].

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N° 688382

4 Example of model use

To demonstrate a use case of these models, a tool called *timbral explorer* was developed. An example of the interface is shown in Figure 3 below, and can be accessed at www.iosr.uk/audiocommons.

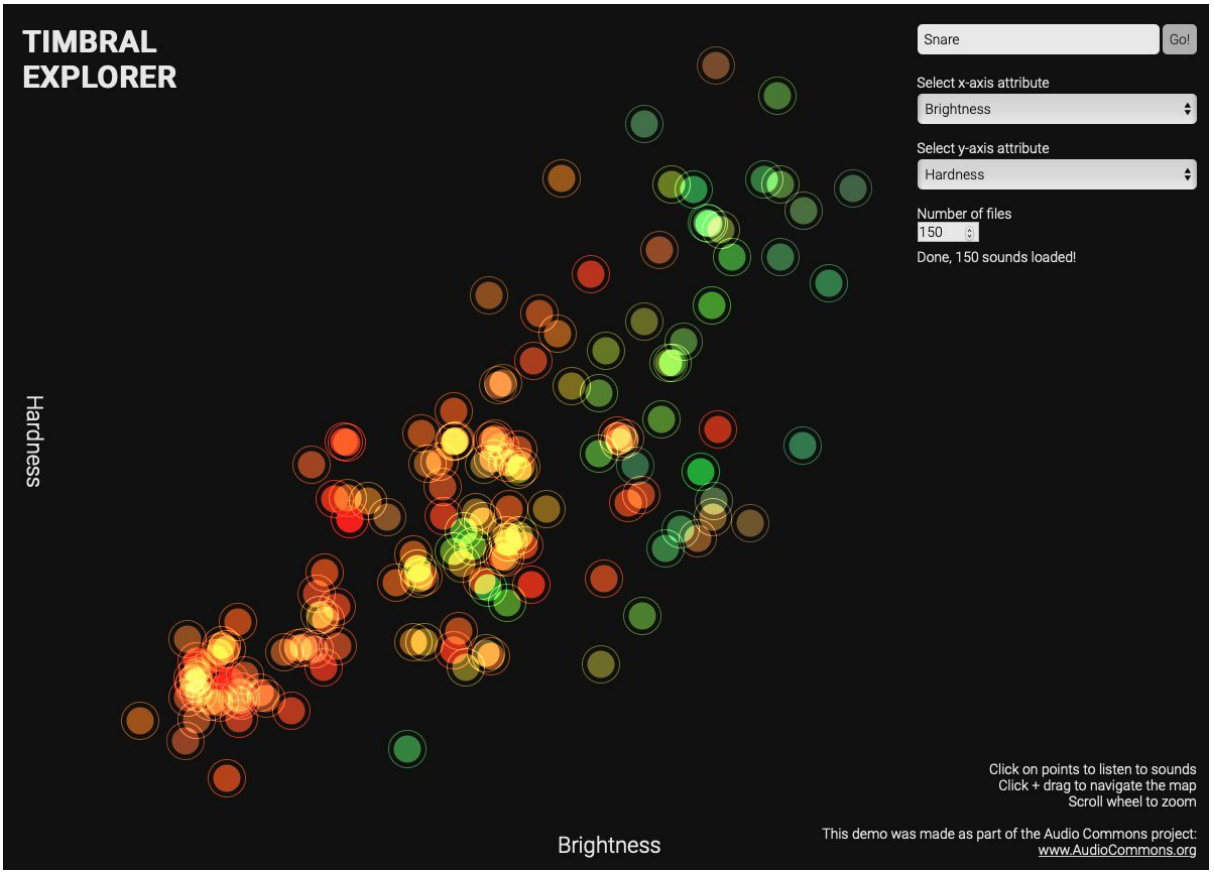

Figure 3 - Example of the Timbral Explorer interface.

This tool allows users to search Freesound for any source type (e.g. snare drum) in the search bar (top right). The results are then displayed as circles whose position depends on the outputs from the timbral models. The x and y axes can be set to any of the attributes timbral attributes. In Figure 3, the sounds are distributed according to their Hardness (on the y-axis) and Brightness (on the x-axis). Therefore sound towards the top of the screen will sound particularly hard, and those towards the bottom will sound soft. Likewise, sounds towards the right of the screen will sound particularly bright, and sounds towards the left will sound dull.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N° 688382

5 Conclusion

In this deliverable, the final implementations of timbral models of *hardness*, *depth*, *brightness*, *roughness*, *warmth*, *sharpness*, *boominess*, and *reverb* were discussed. Two distributions have been discussed: the main python distribution containing all models in a single package, and a MATLAB/Weka distribution containing the full reverb model.

The final python distribution has been updated to accept numpy arrays as inputs, updated with an automated loudness normalisation function, and have improved error handling capabilities. A `timbral_extractor` function has also been added that allows for the computation of all timbral features with a single function call.

The MATLAB/Weka distribution contains a more advanced version of the reverb classification model that outperforms the python distribution.

Potential future improvements to these timbral models could be made by improving the performance of the models to better match subjective data, or testing the applicability of the models for describing musical pieces. Alternatively, new models of additional timbral attributes could be developed, improving the users options for searching for sound effect in the ACE.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N° 688382

5 References

Jan, T., and Wang, W., 2012: "Blind reverberation time estimation based on Laplace distribution", *EUSIPCO*. pp. 2050-2054, Bucharest, Romania.

Le Cessie, S. and van Houwelingen, J., 1992: "Ridge estimators in logistic regression", Applied Statistics, Vol. 41, No. 1, pp. 191-201.

Safavi, S., Pearce, A., Wang, W., & Plumbley, M. (2018). Predicting the perceived level of reverberation using machine learning. In *Asilomar conference on signals, systems and computers*. Pacific Grove.

Schuitman, J., and Vries, D., 2013: "Deriving content-specific measures of room acoustic perception using a binaural, non linear auditory model". *Acoustical society of America*, 133(3), pp.1572-1585.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N° 688382