TR-CITI-15-001

2015/03/23

The AWtoolbox for Characterizing Audio Information

Chin-Chia Michael Yeh, Ping-Keng Jao, and Yi-Hsuan Yang

中央研究院 資訊科技創新研究中心
Reasearch Center for Information Technology Innovation, Academia Sinica
http://www.citi.sinica.edu.tw/HTML/TechReport/pdf/TR-CITI-15-001.pdf

The AWtoolbox for Characterizing Audio Information

Chin-Chia Michael Yeh, Ping-Keng Jao, and Yi-Hsuan Yang
Research Center for IT Innovation, Academia Sinica, Taiwan
{*mcyeh, nafraw, yang*}*@citi.sinica.edu.tw*

March 23, 2015

Abstract

AWtoolbox (Audio Word Toolbox) is an open-source software designed for extracting the audio word (AW) representation of audio signals. This document describes the usage of AWtoolbox for both basic users who are interested in extracting AW representation with the toolbox's graphical user interface and advanced users who are interested to learn about the details of audio word extraction process or to expend the functionality of the toolbox. Additionally, a preliminary benchmark on tweaking various components of audio word extraction process is presented. The preliminary benchmark shows how one can use AWtoolbox to procedurally search for the best AW representation for an audio recognition problem. AWtoolbox is available for download at `http://mac.citi.sinica.edu.tw/awtoolbox`.

Contents

1 Introduction

Audio word (AW) representation is characteristic of its ability of symbolizing any local audio event as a codeword within a pre-constructed dictionary. As dictionary constructing method is independent of the codeword assignment, AW representation has the flexibility of using an arbitrary large number of codewords learned from a corpus of audio data in an unsupervised fashion; thus, AW representation is capable of encompassing rich information from the corpus compactly. On top of that, AW representation only lightly dependents on domain knowledge for feature design. Therefore, it has been considered an powerful alternative to conventional hand-crafted audio features [22]. Due to the advancement of AW encoding and dictionary learning algorithms [2, 12, 23, 20, 14], it is not easy to keep track of different proposals of AW extraction algorithms and make comprehensive comparisons. For example, sparse coding (SC) has received considerable attention in recent years and its variants have been used extensively in various audio-related research [16, 15, 5]. Post-processing methods such as encoding result rectification and normalization [4, 9] have also been claimed important. Because of the lack of a standardized implementation of the related algorithms under a unified framework, it is difficult to compare the results reported in different works and gain insights. As a result, we have developed the AWtoolbox, an open-source graphic user interface (GUI) application intended to facilitate the implementation and development of various AW extraction pipelines. Also, a framework aims to standardize the modularization of the AW representation extraction is proposed. Aside from introducing the functionality of the AWtoolbox and the description of the framework, a preliminary benchmark is presented near the end of this document. In the benchmark section, we showcase the procedural for searching an adequate AW representation for audio recognition problems within the proposed framework.

2 Installation

This section provides a quick start guide for using the pre-compiled executable which is built for 64-bit Windows platform. For 32-bit Windows users, please follow Section 5 to compile AWtoolbox for 32-bit machines.

1. Download AWtoolbox from the BitBucket repository (`https://bitbucket.org/dnaoh/audio_word_toolbox`).
2. Download MATLAB Compiler Runtime 8.1 for 64-bit Windows from MathWorks (`http://www.mathworks.com/products/compiler/mcr/`).
3. Install MATLAB Compiler Runtime.
4. Run the pre-compiled executable at ".\release\audio_word_toolbox.exe" to start AWtoolbox. The GUI should show up as in Figure 1.

3 Use of the GUI

The GUI consists of a menu bar at the top, a design area for setting up the AW extraction process, a input area for setting up the paths for dictionary, a input area for setting up the directory paths for AW encoding, and an output area at the bottom for displaying relevant information. In the following section, a detail explanation is provided for each area.

3.1 Menu Bar

Figure 2 shows the menu items beneath "File" and "Setting". The "Save" and "Load" beneath "File" can be used to save current settings (including options within "Setting" and all control areas) and load pre-exist settings. For the three menu items beneath "Setting", "Output format..." can be used to set the output formant. Currently, the supported formats are comma-separated values (*.csv) and MATLAB MAT-file (*.mat). "File exist action..." can be used to set the response action when the output directory already contain an extracted AW for an audio clip. If "File exist action..." is set to "skip file", multiple instances of AWtoolbox can be launched and set to extracting the same AW from the same input directory to the same output directory because AWtoolbox process the audio clips in the input directory in a random order. Lastly, "Temporary Dir" can be used to set the temporary directory for

Figure 1: A screenshot of the AWtoolbox's GUI right after the toolbox is started.

dictionary learning. Depends on the size of the dictionary learning corpus and the type of representation before an encoding layer, the size of the temporary files could be huge; therefore, pleas make sure to set the temporary directory on a hard drive with sufficient space.

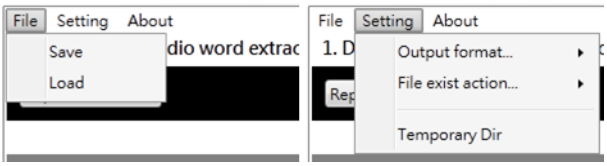

Figure 2: A closer look at the menu bar.

3.2 Design Area

We define five atomic functional layers of AW extraction: `input`, `encoding`, `rectification`, `pooling` and `other`, whose details are presented in Section 4. Different AW representations can be obtained by not only using different algorithms for each layer, but also cascading the functional layers in different ways. The same layer can be applied multiple times, using not necessarily the same algorithm each time. It is this versatility of the AW representation that makes it important to allow the users to define the number and order of these layers on their own. Users can graphically design the process by creating and arranging various kinds of layers for generating the desired AW representation. For visualization purpose, layers are color coded based on their types. For instance, the `input` layer is colored black and the `pooling` layer is colored light blue. Figure 3 provides a closer look at the designing area. The labeled control elements are:

1. drop down menu for selecting the desired function for input layer.
2. button for adding a new layer right after the input layer.
3. drop down menu for selecting the type of layer.

4. drop down menu and text box for setting options for the layer.
5. button for moving the layer up or down.
6. button for deleting the layer.
7. button for adding a new layer right after the last layer.

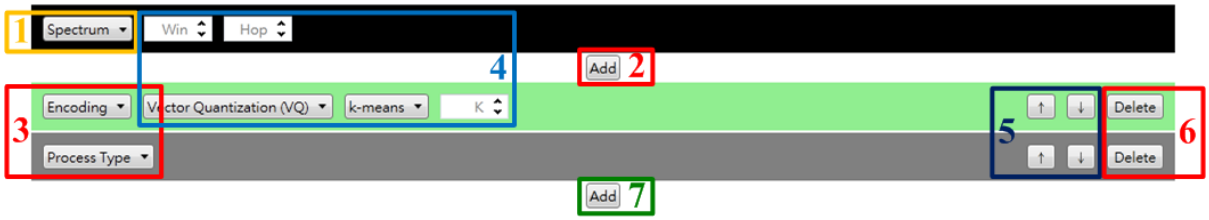

Figure 3: A closer look at the designing area.

3.3 Dictionary Generation

Users can either provide a previous built dictionary or prepare a corpus for constructing the dictionary. The dictionary and the corresponding user-specified design can be saved for later use. Dictionary generation process will generate temporary files, and the generated temporary files may occupy some amount of hard drive. Please make sure the hard drive which the temporary directory located has sufficient space.

3.4 Audio Word Encoding

When the desired dictionary is trained or selected, all the waveform under the input directory (Target Dir) will be encoded to generate the AW representation once the "Encode" button is pressed. The result AW representation will be saved in the output directory (Output Dir).

4 Functional Layer

4.1 Input Layer

The `input` layer is the first layer in any AW extraction pipeline, transforming an input audio stream into a series of $\mathbf{t}$ frame-level vector representation. The included representations are:

Time Series: The function simply reorganizes the audio stream into time-varying vector sequence based on the inputed window and hop size.
Spectrum: The function applies short-time Fourier transform on the input audio stream based on the inputed window and hop size.
Cepstrum: The function applies inverse short-time Fourier transform on the input audio stream's Spectrum. Such representation has been shown effective in guitar playing technique classification [17].
Mel-spectrum: The function apples Mel-scale triangular filters on the input audio stream's Spectrum. In addition to the window and hop size for Spectrum, the function also requires users to set the number of triangular filters.
MFCC: The function applies discrete cosine transform on the input audio stream's Mel-spectrum. The required inputs for this function are: window and hop size for Spectrum, number of triangular filters for Mel-spectrum, and number of cepstral coefficients for the cosine transform.

4.2 Encode Layer

The `encoding` layer is the core in AW extraction pipeline, it maps the input time-varying vectors $\mathbf{X}$ into another space based on the provided dictionary $\mathbf{D}$. Generally, $\boldsymbol{\alpha}$ is used to represent each vector in the output time-varying vector sequence. Since dictionary is always a required input for this layer,

AWtoolbox has provide three different methods for generating the dictionary. For all the dictionary generation methods, the only input is the size of dictionary k.

Encoding Methods

Vector Quantization (VQ): The function represents each vector in the input sequence $\mathbf{x}$ by a one-hot binary vector $\boldsymbol{\alpha}$ according to the nearest codeword $\mathbf{d}_j \in \mathbb{R}^m$ in $\mathbf{D}$. Namely, only an α_j is 1 and the rest of $\boldsymbol{\alpha}$ are 0, where $j = \text{argmin}_p\ z_p$ and $z_p = \|\mathbf{x} - \mathbf{d}_p\|_2^2$.
Triangle Coding (TC): This method is a 'soft' variant of VQ [13], obtains a real-valued $\boldsymbol{\alpha}$ by $\alpha_j = \max\{0, \mu(\mathbf{z}) - z_j\}, \forall j$, where $\mu(\mathbf{z}) = \frac{1}{k}\sum_{p=1}^{k} z_p$ is the mean of these distances.
Sparse Coding (SC): The function represents the input vector by a sparse combination of the dictionary codewords by solving the following LASSO problem [2],

$$\boldsymbol{\alpha}^* = \underset{\boldsymbol{\alpha}}{\text{argmin}} \frac{1}{2}\|\mathbf{x} - \mathbf{D}\boldsymbol{\alpha}\|_2^2 + \lambda\|\boldsymbol{\alpha}\|_1\,, \tag{1}$$

where λ controls the balance between the reconstruction error $\|\mathbf{x}-\mathbf{D}\boldsymbol{\alpha}\|_2^2$ and the sparsity $\|\boldsymbol{\alpha}\|_1 = \sum|\alpha_j|$, which is a convex relaxation of the l_0 norm $\|\boldsymbol{\alpha}\|_0 = \sum|\alpha_j|^0$. λ is set to $1/\sqrt{\min(m,k)}$ as recommended by [12]. For the case of $k \gg m$, it has been shown that SC outperforms VQ for audio classification problems [16].
Sparse Coding with Screening (SCS): This method is a variant of SC with much lower computational cost due to a theoretically-justified mechanism to filter out codewords not useful for reconstructing the input signal before solving Eq. 1 [20]. We adopt an algorithm tailored for audio signals proposed in [10] and employ clip-level rather than frame-level screening for better efficiency in time and memory usage. With SCS, we can afford using larger k for the dictionary. For this function, there is one input λ which is used to set the balance between correctness and rejection rate of the filtering. As higher rejection rate produces smaller filtered dictionary, the overall encoding efficiency is propositional to the rejection rate of filtering.

Dictionary Generation Methods

k-means: The dictionary is constructed by using each cluster center as a codeword after applying k-means clustering to the training corpus. This algorithm is usually used for VQ-based representation [13, 11].
Online Dictionary Learning (ODL): The dictionary is learned by optimizing the following equation using stochastic gradient descent [12],

$$\mathbf{D}^* = \underset{\mathbf{D}}{\text{argmin}}\ \frac{1}{N}\sum_{n=1}^{N}\left(\frac{1}{2}\|\mathbf{x}^{(n)} - \mathbf{D}\boldsymbol{\alpha}^{(n)}\|_2^2 + \lambda\|\boldsymbol{\alpha}^{(n)}\|_1\right), \tag{2}$$

where N denotes the number of vectors in the training corpus and n indexes the training instances. Variants of Eq. 2 that consider other cost functions such as non-negativity, group sparisty and structure sparsity have also been proposed [2], but not yet fully included in the AWtoolbox.
Random Samples (Rand): The function randomly extracts k vectors from the training corpus and directly uses the extracted examples as codewords for the dictionary. Therefore, it bypasses the computational cost involved in clustering or solving Eq. 2. It has been found that using such a random dictionary is effective when the dictionary size k is large [10].

4.3 Rectification Layer

The `rectification` layer applies rectifying non-linearity to the encoding result for improving representation power [4].

Absolute Value (Abs): The function simply applies the absolute value function to all the elements of the input to this layer.
Polar Split (Pol): The function splits the positive and negative elements of the input data into separate ones and concatenates them after changing the sign of the negative ones [4]. For example,

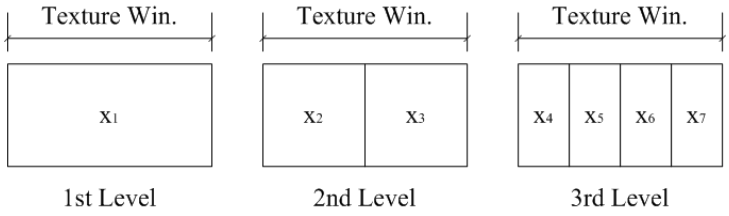

Figure 4: The three-level pyramid pooling partitioned a given segment in three different resolutions. Each of the seven partitions is then pooled with desired aggregation operator. The aggregated result are concatenated as $x = [x_1, x_2, x_3, \cdots, x_7]^T$ to form the output vector x.

when the input is the time-varying encoding result $\mathbf{A} \in \mathbb{R}^{k \times t}$, the output of polarity splitting would be $\hat{\mathbf{A}} \in \mathbb{R}^{2k \times t}$, $\hat{\mathbf{A}} = [\max\{0, \mathbf{A}\}^{\mathrm{T}},\ \max\{0, -\mathbf{A}\}^{\mathrm{T}}]^{\mathrm{T}}$.

4.4 Pooling Layer

The `pooling` layer summarizes a time-varying vector sequence by aggregation operators such as taking the mean or maximum or by other advanced multi-scale pooling techniques such as temporal pyramid pooling (Pyramid) [9]. Particularly, for each of the pooling method (plain or pyramid), there are two required inputs: the pooling function and the pooling level. As pooling can be performed with various aggregation functions, AWtoolbox has provided some of the most popular operators such as *sum*, *mean*, and *max*, and the users can choose from them based on the purpose of the AW. Additionally, since pooling can be done either in the clip-level or in the segment-level (a segment is a subset of a clip consisting of multiple consecutive frames), the users have to decide the level of pooling. For example, if segment-level pooling is applied before encoding layer, the result AW might be more robust against small temporal distortion. When segment-level pooling is chosen, the user also needs to provide the window size and hop size for the segmentation.

Plain: The function simply applies the aggregation operator across the time (within each segment for segment-level pooling) for each dimension in the input representation.
Pyramid: The main idea behind pyramid pooling is to approximate global geometric correspondence in an image by partitioning the image into increasingly fine sub-regions and pools local features found inside each sub-region. For a three level pyramid, the whole image's features are aggregated in the first level. Next, in the second level, the image is divided into 2×2 sub-region, and each sub-region's features are aggregated. For the third level, each sub-region is further divided into 2×2 sub-sub-region (i.e., 16 sub-sub-region in total), and features within each sub-sub-region are aggregated individually. Finally, all the aggregated result are concatenated to form the output feature vector. Unlike images, sounds are 1-D data. Therefore, the partition split the clip into 2 sub-segments instead of 2×2 sub-segments as shown in Fig. 4.

4.5 Other Layer

The `other` layer is added to accommodate other functions related to AW extraction but do not belong to the other four layers. We consider the following three types of functions:

Normalization: This type of functions is important for AW representations. The provided normalization methods are: Unit 2-norm, Sum-to-one, and nth Root normalization. All normalization function normalizes each vector in the time-varying vector sequence independently. Unit 2 norm divide each element within the vector with the vector's 2-norm, Sum-to-one divide each element within the vector with the sum of all the elements within the vector, and nth Root calculate the nth root of each element within the vector with the input degree n.
Random Sampling: The function exploits the repetitive nature of music signals and randomly samples (with replacement) the frame-level features of an audio clip to reduce the number of frames t to be

encoded [23]. The required input q is the percentage (between 0 and 1) of frames to be sampled.
Consecutive Frame (CF): The function concatenates multiple vectors to capture temporal information [14]; can be performed after the `input` or `encoding` layer. The required inputs are the window size for number of vectors to be concatenated and hop size for the number of vectors to skip between each concatenation.

5 Compilation

1. Compile the toolbox SPAMS (`http://spams-devel.gforge.inria.fr/`) under the instruction within the folder ".\MATLAB_code\toolbox\spams-matlab"
2. Compile the MATLAB codes into .dll by running ".\MATLAB_code\compile.m" in MATLAB. Please note MATLAB compiler is required for this step.
3. Compile the GUI by building ".\audio_word_toolbox.sln" with Microsoft Visual Studio.

6 Addition of New Method

This section gives an example to instruct users how to extend the AWtoolbox in case that users feel the included algorithm is insufficient for their own experiments/purposes.

Example:

Suppose you have a function named ⟨mf_encode.m⟩and would like to be added into ⟨Encoding layer⟩. Then you will need to complete five major steps. First, modify a XML file to extend the GUI, some variables are correlated to the second step. These variables are highlighted in red in the first step and second step. Second, modify an m-file so the program can correctly link to ⟨mf_encode.m⟩. Third, coded a wrapper for ⟨mf_encode.m⟩. Fourth, compile with MATLAB, and compile with C# for the last step. The detail is as follows:

Step 1: Modify a XML File

- Open "LayerSetup.xml" in the directory audio_word_toolbox_gui with a text editor.
- Find ⟨/EncodingLayer⟩and the line just before it will be ⟨/item#⟩where # is a number (by default it should be 4 if you simply download the source code with version 1.0).
- Add some lines between ⟨/item#⟩and ⟨/EncodingLayer⟩
  - Assume # = 4 and the ⟨mf_encode.m⟩to be added will be 5th item. So add:
    ⟨item5 itemName="the name you like" numberOfOption="3"⟩
    - "the name you like" will be displayed in the GUI, such as "SC w/ Screening (SCS)" in the figure below and will also be used in the second major step.
    - numberOfOption = "3" stands for 3 parameters (input) to be specified for the ⟨mf_encode.m⟩. For example, there are 3 input boxes (circled by red squares) in the figure below. Set a value that is exactly the same as the number of arguments of ⟨mf_encode.m⟩.
    - There are mainly two types of input box. Specify by value or specify by selecting fixed options. For example with the figure below again, the λ and K is specified by value and the Dictionary is specified by selecting fixed options.
  - Assume the first argument of ⟨mf_encode.m⟩is a double value, then add:
    ⟨option1 optionName="argument name 1" optionType="doubleUpDown" watermark="λ" maximum="1" minimum="0" increment="0.01"⟩⟨/option1⟩
    - "argument name 1" will be used in the second major step m files.
    - Use "doubleUpDown" for double or use "integerUpDown" if the input is an integer.

* The meaning of watermark is the same as its name, see the figure below that λ and K are watermarked when no value is specified.
* maximum="1" minimum="0" increment="0.01" are used for limiting the argument and the increment of pressing an arrow.

– Assume the second argument of ⟨mf_encode.m⟩need to be selected by 2 fixed options, then add:
⟨option2 optionName="argument name 2" optionType="comboBox" numberOfItem="2"⟩
⟨item1 itemName="option name 1"⟩⟨/item1⟩
⟨item2 itemName="option name 2"⟩⟨/item2⟩
⟨/option2⟩

* "argument name 2" will be displayed in GUI at first.
* always use optionType = "comboBox".
* the itemName will be displayed in GUI and feed into m-file.

- Finally, remember to add ⟨/item5⟩at the last line.

Step 2: Modify an M-file

- Open "en_encoding_layer.m" in the "\MATLAB_code\audio_word_encode" with a text editor.
- Add an elseif condition in the if block:
elseif strcmpi(process, 'the name you like')
- Add the body of the just added elseif condition with:
data = mf_encode_wrapper(data, dictionary, process_option);

Step 3: Code a Wrapper for ⟨mf_encode.m⟩

- Code for a wrapper that parse the argument "process_option" by adding a for loop:

```
for i = 1:length(process_option)
    if strcmpi(process_option{i}, 'argument name 1')
        arg1 = str2double(process_optioni+1);
    end
    if strcmpi(process_option{i}, 'argument name 2')
        case process_option{i+1}
        'option name 1'
            arg2 = 0;
        'option name 2'
            arg2 = 1;
        otherwise
            error;
        end
    end
end
```

- Then, call ⟨mf_encode.m⟩with parsed argument by:
data = mf_encode(arg1, arg2);

Step 4: Compile with MATLAB

Run ".\MATLAB_code\compile.m" with MATLAB. Please note MATLAB compiler is required for this step.

Step 5: Compile with C#

Open "audio_word_toolbox.sln" with Microsoft VisualStudio and compile.

Following the same spirit and syntax, you can add the code into any layer you like. There is one thing that is different for encoding layer. Users always have to add "Dictionary" and "Dictionary Size" as options (arguments), although the example did not show this. Users should simply copy and paste from the xml file there is no new dictionary learning algorithm used.

7 Preliminary Benchmark

In this section, preliminary benchmark of AW with different settings is presented. As the choice of input representation and encoding method is the core of the AW extraction pipeline, we have first exhaustively test all the possible combinations of varying these two components in a baseline AW extraction pipeline. The baseline AW extraction pipeline consists of the following five steps in sequence: 1) input representation extraction, 2) encoding with random sampled 2048 sized dictionary, 3) absolute value rectification, 4) plain clip level mean pooling, and 5) cube root normalization. Once the best input-encode combination is determined, we used the combination and varied the other settings one at a time in the baseline pipeline to figure out the effect of each components. Lastly, we identify five interesting setups and apply them to six different audio recognition problems to benchmark the performance of different setups.

7.1 Data Set

The following seven data sets are used in this section, and they are:

CAL10k is a subset of the data set collected by Tingle *et al.* [18] for music auto-tagging, which contains the genre and acoustic annotation of music labeled by professional music editors of the music service company Pandora (`http://www.pandora.com`). With the 7digital API (`http://www.7digital.com`), we collect the 30 sec audio previews of 7,799 songs, spanning 140 genre tags and 435 acoustic tags. We consider it as a multi-label problem and evaluate the precision for tag-based retrieval using the training/test splits (akin to five-fold CV) defined by [18]. The performance measures are area under the receiver operating characteristic curve (RAUC), mean average precision (MAP), precision at rank ten (P10) and precision at rank R (PR), R being the number of relevant clips [18].

MTG Instrument (MTG) consists of 2,500 polyphonic music clips, and each clip comes with a label of the predominant pitched instrument played in that duration. There are 11 instrument labels: `cello`, `clarinet`, `flute`, `acoustic guitar`, `electric guitar`, `Hammond organ`, `piano`, `saxophone`, `trumpet`, `violin` and `singing voice`. The clips are sampled from various genre of music and most music clip is shorter than 10 sec [7]. F-score and average classification accuracy (ACC) are reported for this data set.

FreeSound consists of 20,626 audio clips collected from Freesound (`http://www.freesound.org`). Each clip is manually labeled with one of the following five categories: `sound effect`, `soundscape`, `speech`, `instrument sample` and `complex music fragment` [6]. As the clips are annotated with mutually exclusive labels, we formulate the problem as a multi-class classification problem and report the F-score and ACC. We consider the first 30 sec for overly long clips for feature extraction and use ten-fold cross validation (CV) for evaluation.

MER31k contains 31,427 30 sec audio previews labeled with 190 music emotion tags [21] by the crowd of last.fm users (`http://www.last.fm`). These are the songs considered as most relevant to the emotion tags according to the `Tag.getTopTracks()` function of the last.fm API. We consider the subset of 43 emotion tags which appear in the Affective Norm for English Words [1], such as `sad`, `lazy`, `relaxed`, `happy`, `romantic`, `fun` and `angry`, and evaluate the accuracy for tag-based retrieval using the training/test split specified in [21]. The performance measures are RAUC, MAP, P10, and PR.

CAL500 contains 502 western popular music manually annotated with a lexicon of 174 pre-defined tags [19]. The length of a music audio clip ranges from 3 sec to over 22 min. According to the common protocol in the literature [14], we used a subset of 97 tags and evaluated the performance for both semantic *annotation* and *retrieval*. The accuracies of annotation (i.e., annotating a song with tags) were evaluated in terms of F-score and song-wise AUC (AAUC); the accuracies of

retrieval (i.e., retrieving relevant songs with respect to a tag query) were evaluated in terms of RAUC, MAP, P10 and PR [19].

USPOP is an external data set which only be used as a data source for unsupervised dictionary training. It contains nearly 7,000 contemporary pop music [3].

RWC Instrument (RWC) is another external data set for unsupervised dictionary training. It consists the monophonic sound of 150 musical instruments with various playing styles, pitches, and dynamics [8].

Table 1: The results of the CAL10k input versus encode benchmark, with the top two results for each performance metric highlighted.

		CAL10k Genre			CAL10k Acoustic		
		VQ	TC	SC	VQ	TC	SC
RAUC	Time series	0.7887	0.7933	0.8448	0.7590	0.7779	0.8311
	Spectrum	0.8350	0.8467	**0.8693**	0.7998	0.8255	**0.8458**
	Cepstrum	0.8279	**0.8674**	0.8638	0.7941	**0.8512**	0.8439
	Mel-spectrum	0.8076	0.8462	0.8470	0.7851	0.8228	0.8214
	MFCC	0.8015	0.8394	0.8352	0.7786	0.8175	0.8131
MAP	Time series	0.1239	0.1032	0.1483	0.1152	0.0994	0.1460
	Spectrum	0.1671	0.1437	**0.2012**	0.1511	0.1374	**0.1822**
	Cepstrum	0.1741	0.1748	**0.1939**	0.1513	0.1627	**0.1763**
	Mel-spectrum	0.1558	0.1429	0.1809	0.1400	0.1379	0.1653
	MFCC	0.1424	0.1360	0.1674	0.1324	0.1331	0.1555
P10	Time series	0.1667	0.1340	0.1824	0.1614	0.1277	0.1861
	Spectrum	0.2173	0.1759	**0.2444**	0.1988	0.1743	**0.2306**
	Cepstrum	0.2231	0.2163	**0.2394**	0.2027	0.2017	**0.2248**
	Mel-spectrum	0.2067	0.1786	0.2341	0.1965	0.1783	0.2194
	MFCC	0.1939	0.1727	0.2166	0.1870	0.1719	0.2110
PR	Time series	0.1330	0.1118	0.1539	0.1298	0.1105	0.1543
	Spectrum	0.1771	0.1505	**0.2085**	0.1661	0.1481	**0.1942**
	Cepstrum	0.1851	0.1841	**0.2045**	0.1694	0.1722	**0.1885**
	Mel-spectrum	0.1699	0.1524	0.1892	0.1579	0.1500	0.1801
	MFCC	0.1572	0.1433	0.1760	0.1514	0.1461	0.1718

Table 2: The results of the MTG input versus encode benchmark, with the top two results for each performance metric highlighted.

		VQ	TC	SC
Fscore	Time series	0.2386	0.2334	0.4006
	Spectrum	0.4036	0.1890	**0.5913**
	Cepstrum	0.4108	0.2602	0.5167
	Mel-spectrum	0.4620	0.3271	**0.5694**
	MFCC	0.4339	0.2902	0.5343
ACC	Time series	0.2628	0.2812	0.4094
	Spectrum	0.4196	0.2498	**0.5972**
	Cepstrum	0.4176	0.3094	0.5220
	Mel-spectrum	0.4711	0.3669	**0.5738**
	MFCC	0.4437	0.3387	0.5459

7.2 Input Versus Encode

The experiment in this section is performed on three different audio recognition tasks, and they are: CAL10k genre tag auto-tagging, CAL10k acoustic tag auto-tagging, and MTG instrument recognition. We have tested all five input representations combining with VQ, TC and SC with default settings.

The dictionaries are constructed from each corresponding data set. Because the dictionary construction is unsupervised, the label information is not accessed even if both training and test set are used for dictionary construction. Tables 1 and 2 shows the result of CAL10k and MTG respectively. Throughout the experiment, combining Spectrum with SC produces the best audio word representation. For CAL10k, combining Cepstrum with SC being the close second, and for MTG, combining Mel-spectrum with SC being the second. Because spectrum plus SC constantly produces the best audio word representation, the choice of input representation and encoding method is fixed to Spectrum and SC while varying other components in baseline pipeline in the next section.

Table 3: The alternative of each component for the baseline pipeline from the first step to the last.

	Baseline	Alternative	Associated Layer
Input representation	Spectrum	–	Input
Representation process	No process	Consecutive frame (CF)	Other
Encoding method	Sparse coding	–	Encode
Dictionary training method	Random samples	k-means, ODL	
Dictionary training data source	Self	CAL10k, MTG, USPOP, RWC	
Dictionary size	2,048	1,024, 4,096	
Rectification	Absolute value	Polar split	Rectification
Pooling	Plain	Pyramid	Pooling
Normalization	Cube Root	No normalization	Other

Table 4: The results of varying each component's setting, with the top results in each performance metric for each component highlighted. PR is not shown in this table because PR's trend is identical to P10's. We highlight the settings of the Baseline in the second column.

		CAL10k Genre			CAL10k Acoustic			MTG	
		RAUC	MAP	P10	RAUC	MAP	P10	Fscore	ACC
CF	**No**	**0.8693**	0.2012	0.2444	**0.8458**	**0.1822**	**0.2306**	**0.5913**	**0.5972**
	Yes	0.8668	**0.2029**	**0.2481**	0.8439	0.1813	0.2270	0.5700	0.5769
Dictionary method	**Rand**	0.8693	0.2012	0.2444	0.8458	0.1822	0.2306	0.5913	0.5972
	kmeans	0.8713	0.2059	0.2529	0.8496	0.1855	0.2355	0.6074	0.6125
	ODL	**0.8759**	**0.2112**	**0.2587**	**0.8525**	**0.1899**	**0.2394**	**0.6416**	**0.6464**
Dictionary source	CAL10k	0.8693	**0.2012**	0.2444	0.8458	**0.1822**	**0.2306**	**0.6515**	**0.6558**
	MTG	0.8560	0.1792	0.2217	0.8262	0.1636	0.2138	0.5913	0.5972
	USPOP	**0.8698**	0.2004	**0.2451**	**0.8463**	0.1798	0.2267	0.6493	0.6537
	RWC	0.8575	0.1678	0.2077	0.8369	0.1600	0.2038	0.5376	0.5449
Dictionary size	1,024	**0.8712**	0.1920	0.2329	**0.8503**	0.1765	0.2218	0.5522	0.5612
	2,048	0.8693	0.2012	0.2444	0.8458	**0.1822**	0.2306	0.5913	0.5972
	4,096	0.8636	**0.2064**	**0.2566**	0.8330	0.1812	**0.2331**	**0.6199**	**0.6250**
Rectify	**Abs**	0.8693	0.2012	0.2444	**0.8458**	0.1822	0.2306	0.5913	0.5972
	Pol	**0.8695**	**0.2065**	**0.2504**	0.8426	**0.1849**	**0.2331**	**0.6102**	**0.6156**
Pool	**Plain**	**0.8693**	**0.2012**	**0.2444**	**0.8458**	**0.1822**	**0.2306**	**0.5913**	**0.5972**
	Pyramid	0.8248	0.1730	0.2216	0.7894	0.1503	0.2043	0.5756	0.5854
Norm	No	0.8209	0.1207	0.1474	0.8026	0.1157	0.1511	0.4510	0.4787
	Cube	**0.8693**	**0.2012**	**0.2444**	**0.8458**	**0.1822**	**0.2306**	**0.5913**	**0.5972**

7.3 Other Settings

To further improve the power of the audio word representation generated by the baseline pipeline, each component is tested individually. The processes in baseline pipeline can be roughly categorized into three types of processes: 1) input representation extraction and processing, 2) encoding and dictionary training, and 3) post-processing. The three types of processes are applied sequentially as the presented

order. In the first type of processes, we have tested the addition of consecutive frames (CF). In the second type of processes, since the encoding method is fixed to SC, we mainly focused on varying settings associate with dictionary training, such as the dictionary training methods, dictionary training data set, and dictionary size. In the last type of processes, we examined the difference between absolute value and polar split rectification, difference between plain and pyramid pooling, and the removal of cube root normalization. A list of alternative settings for each components in the baseline pipeline is outlined in Table 3.

First of all, CF does not benefit the system. It is possible that local temporal information doest not play an important role in auto-tagging and instrument recognition problem. In terms of dictionary, the experiment result favors large, ODL trained, CAL10k/USPOP trained dictionary. Large dictionary is preferred in most cases because it contains more examples comparing to smaller dictionary as more examples means more information preserved. On the other hand, ODL trained dictionary incorporates the example compactly; thus, a ODL trained dictionary contains more information comparing to the naïve Rand dictionary. The main difference CAL10k/USPOP and MTG/RWC is total length of music audio clips; therefore, larger information source is desired for dictionary training. Note, in the MTG case, both CAL10k and USPOP trained dictionary outperform the MTG trained dictionary. This fact suggests that having a large training data set is more important than training with target data set. As a result, a large dictionary that is trained with a huge data set can work well in many different audio recognition problems. For the post-processing, polar split rectification, plain pooling, and cube root normalization produced the best result. Based on the experiment result, the best audio word representation is extracted by 1) Spectrum extraction 2) SC using 4,096 sized ODL dictionary trained from CAL10k/USPOP, 3) polar split rectification, 4) plain pooling, and 5) cube root normalization. Table 4 summarize the result for this set of experiment.

Table 5: The results of CAL10k, with the top two results in each performance metric highlighted.

	CAL10k Genre				CAL10k Acoustic			
	RAUC	MAP	P10	PR	RAUC	MAP	P10	PR
Random Guess	0.497	0.021	0.017	0.017	0.500	0.025	0.022	0.021
MFCC + VQ + k-means	0.803	0.144	0.191	0.159	0.500	0.025	0.022	0.021
Spectrum + SC + ODL	**0.869**	**0.214**	**0.257**	**0.222**	**0.838**	**0.184**	**0.236**	**0.199**
Spectrum + SC + Rand	0.864	**0.208**	**0.258**	**0.215**	0.833	**0.181**	0.232	**0.196**
Cepstrum + SC + ODL	**0.866**	0.201	0.248	0.207	**0.841**	**0.181**	**0.233**	0.194
Cepstrum + SC + Rand	0.857	0.199	0.248	0.204	0.830	0.175	0.231	0.193

Table 6: The results of MTG, FreeSound, and MER31k, with the top two results in each performance metric highlighted.

	MTG		FreeSound		MER31k			
	Fscore	ACC	Fscore	ACC	RAUC	MAP	P10	PR
Random Guess	0.091	0.091	0.183	0.200	0.503	0.007	0.005	0.005
MFCC + VQ + k-means	0.487	0.491	0.440	0.473	0.770	0.101	0.214	0.146
Spectrum + SC + ODL	**0.713**	**0.716**	0.537	**0.563**	**0.795**	**0.127**	**0.252**	**0.174**
Spectrum + SC + Rand	**0.695**	**0.698**	**0.540**	**0.563**	**0.793**	**0.122**	**0.238**	**0.168**
Cepstrum + SC + ODL	0.580	0.584	0.523	**0.564**	0.784	0.106	0.211	0.151
Cepstrum + SC + Rand	0.588	0.591	**0.540**	0.579	0.778	0.107	0.221	0.154

7.4 Presets

From previous experiments, we have identified five setups as *presets* for further examination. The five presets are variants of the suggested pipeline concluded in last section; the main differences between them are the input representation, encoding method, and dictionary training method. The five presets are: 1) MFCC + VQ+ k-means as this is the most commonly seen setup in the literature, 2) Spectrum + SC + ODL as the best performed setup from last section, 3) Spectrum + SC + Rand as the fast trained dictionary alternative, 4) Cepstrum + SC + ODL as one of the promising setup from previous

Table 7: The results of CAL500, with the top two results in each performance metric highlighted.

	AAUC	Fscore	RAUC	MAP	P10	PR
Random Guess	0.497	0.129	0.499	0.273	0.232	0.244
MFCC + VQ + k-means	0.772	0.206	0.696	0.439	0.456	0.406
Spectrum + SC + ODL	**0.807**	**0.228**	**0.740**	**0.482**	**0.502**	**0.447**
Spectrum + SC + Rand	0.804	**0.224**	**0.733**	**0.477**	0.491	0.438
Cepstrum + SC + ODL	**0.807**	0.221	0.731	**0.477**	**0.504**	**0.440**
Cepstrum + SC + Rand	0.806	0.219	0.730	0.474	0.496	**0.440**

section, and 5) Cepstrum + SC + Rand as the fast trained dictionary alternative. All the dictionary used in this section is trained with USPOP data set to simulate the real scenario where the dictionary is trained with an external large data set. The six included audio recognition tasks are: 1) CAL10k genre tag auto-tagging, 2) CAL10k acoustic tag auto-tagging, 3) MTG instrument classification, 4) FreeSound sound clip classification, 5) MER31k emotion tag auto-tagging, and 6) CAL500 auto-tagging. Based on the experiment results, all AW representation surpasses random guess, and Spectrum + SC + ODL is the best while Spectrum + SC + Rand is the close second. Also, both the Spectrum + SC + X and its Cepstrum variant outperform the MFCC + VQ+ k-means baseline. As a result, combining the baseline pipeline with Spectrum + SC + ODL is an appealing solution when dealing with the aforementioned audio recognition problems. We have empirically identified a preferred solution within the proposed framework. The settings and dictionaries can be downloaded at the project's website.

8 Conclusion

In this document, we have provided a guide on the basic and advanced functionality of the AWtoolbox. The GUI component of the toolbox provides a friendly interface for the basic users for quick feature extraction while the versatility of the toolbox assures the advanced users can further expend the toolbox at will. Additionally, we have shown how one can systematically improve a traditional VQ based AW representation within the proposed framework. The preferred AW representation extraction pipeline consists of 1) Spectrum extraction 2) SC using 4,096 sized ODL dictionary trained from USPOP, 3) polar split rectification, 4) plain pooling, and 5) cube root normalization; it outperforms the traditional VQ by a large margin in six different audio recognition tasks. All five setups along with the AWtoolbox can be access at the project's website: `http://mac.citi.sinica.edu.tw/awtoolbox/`.

9 Acknowledgement

This work was supported by the Academia Sinica Career Development Award 102-CDA-M09. The development of this toolbox is greatly benefited by the SPArse Modeling Software (SPAMS) developed by the Institut national de recherche en informatique et en automatique (INRIA), France.

10 Bibliography

References

[1] [Online] `http://csea.phhp.ufl.edu/media/`.

[2] F. Bach, R. Jenatton, J. Mairal, and G. Obozinski. Optimization with sparsity-inducing penalties. *Foundations and Trends in Machine Learning*, 2012.

[3] A. Berenzweig, B. Logan, D. P. W. Ellis, and B. Whitman. A large-scale evaluation of acoustic and subjective music similarity measures. In *Computer Music Journal*, 2003.

[4] A. Coates and A. Ng. The importance of encoding versus training with sparse coding and vector quantization. In *ICML*, pages 921–928, 2011.

[5] L. Deng and X. Li. Machine learning paradigms for speech recognition: An overview. *TASLP*, 21(5):1060–1089, 2013.

[6] F. Font et al. Audio clip classification using social tags and the effect of tag expansion. In *Semantic Audio*, 2014.

[7] F. Fuhrmann. *Automatic musical instrument recognition from polyphonic music audio signals*. PhD thesis, Universitat Pompeu Fabra, 2012.

[8] M. Goto and T. Nishimura. Rwc music database: Music genre database and musical instrument sound database. In *in ISMIR*, pages 229–230, 2003.

[9] P.-S. Huang, J. Yang, M. Hasegawa-Johnson, F. Liang, and T. S. Huang. Pooling robust shift-invariant sparse representations of acoustic signals. In *Interspeech*, 2012.

[10] P.-K. Jao, C.-C. M. Yeh, and Y.-H. Yang. Modified LASSO screening for audio word-based music classification using large-scale dictionary. In *ICASSP*, 2014.

[11] Y.-G. Jiang. SUPER: Towards real-time event recognition in internet video. In *ICMR*, 2012.

[12] J. Mairal, F. Bach, J. Ponce, and G. Sapiro. Online dictionary learning for sparse coding. In *ICML*, pages 689–696, 2009.

[13] B. McFee, L. Barrington, and G. R. G. Lanckriet. Learning content similarity for music recommendation. *TASLP*, 20(8):2207–2218, 2012.

[14] J. Nam, J. Herrera, M. Slaney, and J. Smith. Learning sparse feature representations for music annotation and retrieval. In *ISMIR*, 2012.

[15] E. C. Smith and M. S. Lewicki. Efficient auditory coding. *Nature*, 439(7079):978–982, 2006.

[16] L. Su, C.-C. M. Yeh, J.-Y. Liu, J.-C. Wang, and Y.-H. Yang. A systematic evaluation of the bag-of-frames representation for music information retrieval. *TMM*, 2014.

[17] L. Su, L.-F. Yu, and Y.-H. Yang. Sparse cepstral and phase codes for guitar playing technique classification. In *ISMIR*, 2014.

[18] D. Tingle et al. Exploring automatic music annotation with "acoustically-objective" tags. In *MIR*, 2010.

[19] D. Turnbull, L. Barrington, D. Torres, and G. Lanckriet. Towards musical query-by-semantic-description using the CAL500 data set. In *ACM SIGIR*, pages 439–446, 2007.

[20] Z. J. Xiang, H. Xu, and P. J. Ramadge. Learning sparse representations of high dimensional data on large scale dictionaries. In *NIPS*, 2011.

[21] Y.-H. Yang and J.-Y. Liu. Quantitative study of music listening behavior in a social and affective context. *TMM*, 15(6):1304–1315, Oct 2013.

[22] C.-C. M. Yeh, P.-K. Jao, and Y.-H. Yang. Awtoolbox: Characterizing audio information using audio words. In *ACM Multimedia*, 2014. `http://mac.citi.sinica.edu.tw/awtoolbox`.

[23] C.-C. M. Yeh, J.-C. Wang, Y.-H. Yang, and H.-M. Wang. Improving music auto-tagging by intra-song instance bagging. In *ICASSP*, 2014.