# sndr Documentation

- Paris Smaragdis,

sndr is a command-line tool which allows one to learn sound models and then use these models to classify new audio content and find specific kinds of sounds in it. This software runs in three modes:

1. Learning mode

2. Classification mode

3. Tracking mode (not quite finished yet)

The usual work pipeline involved two steps, learning and classifying. We describe these steps in the following two sections.

## Learning Sound Models

sndr uses either Gaussian Mixture Models (GMMs), or Hidden Markov Models (HMMs) to learn sound classes. Learning is performed by providing sound files that

exemplify the sound class one needs to model. For example, in order to learn a model for speech we need to gather sound files that include speech in them.

In order to learn a model of a sound class we need to go through two processes. First we need to perform *feature extraction* and subsequently estimate the model parameters.

Feature extraction is the process through which we represent the sound in a form that makes classification easy. A representation such as the waveform is not a useful representation for such processes, therefore we use time/frequency representations such as spectrograms, mel cepstra, etc. There are various parameters that define this transform which are controlled by the following flags. All of these flags are to be followed by a numerical value that will be used for the denoted parameter.

-l and -h : these two flags (each followed by a frequency value in Hz), define the lowest and highest frequency that we will use when representing the sound. Be careful to not use an upper frequency value that is higher than half of the sampling rate used in the sound files involved. Depending on the sound you want to learn, you can use a higher low frequency to make sure that the sound representation won’t be disturbed by low frequency rumble. And likewise you can use a lower upper frequency to ignore frequencies that the sound at hand might not display as much. E.g. for speech you can use the pair: -l 100 -h 5000, which defines the frequency range in which speech is dominant. These values would be bad for learning models of different music genres since they would discard a lot of low and high frequency bands which are used in music.

-T : This is the threshold parameter which allows us to discard low-energy frames. Usually with many sounds there are long pauses that are not meant to be used for training. For example, speech recordings often include low energy regions that include background noise or silence that is counterproductive to training a speech model. This parameter specifies a sound level as a fraction of the loudest peak in the input. All input frames under that level will be discarded. The definition of that level is done as a number from 0 to 1. A setting of -T 0 will result in all frames to be used for training, whereas -T 0.5 will result in keeping everything which is at least half as loud as the loudest part of the recording. Values around 0.1 to 0.3 are good picks if one wishes to use this setting. Anything more than that will probably result in a severe decimation of the amount of training data and potentially bias the model to only recognize loud cases of a sound class (which of course might be the intended effect - e.g. when making an agitated speech detector). By default this value is set to 0, which is fine for most purposes.

-t : This the frame size expressed in seconds. The sound files used are read in a frame-by-frame manner, and this parameter defined how long these frames are. For example a value of -t 0.1 will result in frames that are one tenth of a second long. This is an important parameter that can vary a lot depending on the goals of the task at hand. If you are trying to detect sound classes which last for a while, e.g. music in a movie, then you can afford to have a long frame which can be one or two seconds long. If you are trying to find an car crash, which is a sound that takes only a fraction of a second you need to have a frame size that is comparable to that (e.g. a tenth of a second). If the frame is too big, during most of the frame’s duration we would have another sound dominating (such as background music) and the crash could go unnoticed. That said, too short a frame means less sound data used to make an informed decision that can result in unreliable classifications. This value is also used to define the length of the FFT used for the feature transformation (closest power of two samples to that frame size), so it also influences the frequency resolution of the transform (large frames result in better frequency resolution). Values between 0.01 and 1 are reasonable values.

-H : This is the frame *hop size*. This defines how fast we sample frames in time. This is tied to the frame size parameter. A setting of -H *x* means that we want to have a new frame every *t*/*x* seconds where *t* is the size if the frame. So for -t 0.5 -H 1, we will have a half second long frame every half second, whereas -t 0.5 -H 2 will result in just as long frames, but twice as frequently. Traditionally the hop size is a power of two, and values higher than 8 are usually excessive. A higher hop size means that we take more frames per second and thus we have a finer-grained representation of the input (so if you want to track fast changing sound classes a large value for the hop size is a good idea). That does however mean that we have to perform more computations and consume more memory. A value of 2 is usually a good choice. The default value is 1.

-a : This is the feature averaging flag. In most cases you do not need to set this to a value other than the default of 1. For -a *x* every set of consecutive *x* frames is averaged and the classification is based on that averaged value as opposed to each individual frame. This is rarely useful and can potentially obscure fast changing sounds due to the averaging.

-F : This is the options control string that defines what steps to take for the complete feature transformation. This includes a series of characters that can be the following:

n : Normalize the energy of each FFT frame.

l : Take log of the transform.

b : Use Bark scale frequency warping.

m : Use Mel scale frequency warping.

c : Use a DCT transform.

0 : Use DCT’s 0th coefficient.

e : Append the energy value for each frame.

d : Use delta features.

D : Use delta-delta features.

The default is -F cdm, which uses a standard representation used a lot in the speech processing world. Quite often it will make sense to also use the flags 0 and e which take into account the loudness of a sound. Without them the trained model can learn a sound regardless of how loud it can be, but it will be likely to “hallucinate” is very soft sections and think that it “hears” sounds which aren’t there (this is technically correct because in silence you can claim to have any sound you want at an inaudible level). Adding the aforementioned flags makes the representation take note of usual levels so that silent parts are treated as being different.

The Bark and Mel scales are good for learning coarse models of sounds, whereas not using them will result in a spectrogram representation which can be good at learning specific things (e.g. a specific note), but it won’t generalize well.

The DCT transform is used to clan up the Mel or Bark scaling and often results in sound representations which are easier to learn.

The delta and delta-delta options attach in each frame the first and second derivative of the features respectively. This is good if one wants to have a rudimentary encoding of the temporal characteristics of a sound. The delta is often used, the delta-delta is usually overkill.

-b : This is the number of coefficients to use for each frame. This sets the size of the DCT transform, if that is specified in the above flags. Typical values are from 10 to 30. The default value is 13.

-n : This is the number of filterbanks to use for the Mel frequency warping. This only used when Mel warping is enabled. You rarely will have to change this value.

To define the models that we train, and how we do so, we use another set of flags:

-K : This is the number of Gaussian components. More components result in a more accurate model, but also in a very flexible model that can potentially describe other sources. Too few components make a very general model that might not be accurate. Reasonable values are between 4 and 20. The default value is 8.

-S : If using an HMM model, these are the number of states for the HMM (each having as many Gaussians as the -K flag specifies). An HMM model takes more care to model the temporal structure of sounds, whereas a GMM model is essentially modeling the spectral character of a sound. In most cases the GMM is enough, but there are situations in which the sound class has a lot of temporal information and an HMM model is better suited. For example the sound of rain and a bomb explosion are spectrally very similar (they are both wideband sounds) and mostly differ in their temporal evolution; rain being a constant stationary sound and the explosion consisting of a sharp onset and a slow release time. An HMM would do better in this case. If we however were to compare speech with rain, the two sounds are different enough in their spectral structure that we do not need to resort to HMMs. HMMs are slower and more complex so whenever possible the GMMs would be a more appropriate choice. If you use the -S flag the HMM learning will be enabled, otherwise GMMs will be used. \*\*\* currently HMMs are enabled by compiling with the \_\_HMM\_TRAIN flag, the flag-based model switching is not enabled yet.

-it : This is the number of training iterations that the model will train on. The more the better, but the improvement comes asymptotically, so too many iterations are not going to be a lot of help. During training keep your eye on the likelihood value. If it doesn’t look like it has started converging to a compact area you can try extending learning by a few more iterations. The default iterations value is 50.

Finally there are the file flags:

-i : Input sound file(s). This flag defines the name of the input files that one wishes to process. If we want to perform learning then for -i foo.wav bar.wav, this utility will learn a model for the sounds in the soundfiles foo.wav and bar.wav. Make sure that the input sounds are good representation of what you want to learn. If you want to learn a model of a particular person’s voice, make sure that you pick samples that do not include a lot of ambient noise, long pauses with background sounds, etc. You need to find representative examples. You can’t learn Obama’s voice from a TV interview and then recognize it in a live large crowd gathering because these two cases of his voice are very different. Make sure that your training data is representative of what you want to model.

-M : Filename for output model. Upon training the learned model is saved to disk and named according to this flag.

Using these flags we can now train a set of sound models from training sound files. For the Friends data we can do:

sndr -K4 -S 8 -t 0.1 -H 2 -Fcdb0e \\

-i ross.wav joey.wav chandler.wav -M guys

which will learn a model of the male speakers in the cast. Likewise we can lean the female cast, the music breaks and the laughter track:

sndr -K4 -S 8 -t 0.1 -H 2 -Fcdb0e \\

-i monica.wav rachel.wav phoebe.wav -M gals

sndr -K4 -S 8 -t 0.1 -H 2 -Fcdb0e \\

-i music.wav -M music

sndr -K4 -S 8 -t 0.1 -H 2 -Fcdb0e \\

-i laugh.wav -M laughter

This will be the only action we need to perform to train classifiers for these four sound classes.

## Classifying a new input

Once we have a set of models for some sound classes we can use them to track the appearance of these classes in a new recording. This process consists of two parts again, the feature extraction and the actual classification. The feature extractions are the same as before. Make sure you use the same feature parameters for both training and classification. If the features are different across runs then the classifiers will not function properly.

For each input file, each feature frame will be classified as belonging to any of the learned classes. This is done by measuring the likelihood of each frame for each model, and then assigning that frame to the highest likelihood class. This process can be noisy and having rapid changes between successive frames. For example when one person speaks, the voiced parts would be recognized as voice, but the short pauses between words might not. In order to deal with that we define a temporal persistence parameter which acts as a deterrent for this behavior.

The file flags used for classification are:

-i : As before this flag specifies the input file(s). This time these files are analyzed according to the previously learned models and each input frame is classified as belonging to one of the learned sound classes.

-m : Filename for input models. The use of -m music speech loads the pre-learned models music and speech and uses them for the classification. Note that the classification will only return an estimate of whether a sound is music or speech. If there is another class of sounds in the input file then it will be assigned to whichever class it is most similar to. This can result in counterintuitive results, such as wind noise being classified as music, or car horns being classified as speech. If you don’t have a (good) model for a class of sounds, they won’t be classified properly. That said, it is best to minimize the number of sound classes you use. Use too many classes and classification will be slower and with a higher potential for errors.

-d : Filename suffix for classification sound files dump. When doing recognition or tracking this option allows the generation of a set of soundfiles that include only the detected sounds from each class respectively. The flag here will be the prefix used for these files. A setting of -d foo for an input file named bar, will result in a set of files bar.wav.foo.**x**.wav, where **x** = 0, 1, 2, … for each class.

-D : EDL filename prefix. This acts the same way as above, and creates a set of EDL files which describe at which times each sound class is active.

*-g : Target file. Not used currently*

The parameter flags used for classification define how the classification results will be interpreted, and are the following:

-p : This is the temporal persistence parameter that we described above. It is encoded as the probability to transition from one class to another. If we use -p 1, then the probability of repeating the classification of the previous frame is 1 (and in this extreme case we will never change classification labels). A slightly lower value of -p 0.999, which is the default, is usually a good bet even though it might sound excessive. Values less than 0.9 are probably going to result in too many classification jumps.

-f : Length of state output filter. In addition to the persistence parameter we can use a median filter that smoothes out the classification labels across time. The integer value used in this option is the length of the median filter. A value of -f 0 is the default and means that there will be no filter imposed. The length of the filter is in terms of frames, so there is an implied time window that the median filter operates over. If one wants to have sound labels that don’t change much within, say, a second then an appropriate value for the filter would be that of the number of frames that fit in one second (make sure to factor the hop size in this calculation).

*-w : Bias of state output filter. This is a test flag not to be used officially.*

*-r : State transition bias**. Test option not to be used officially.*

*-B : Likelihood bias. Test option not to be used officially.*

Given all of the above and going back to the Friends example we can use the following command to classify a Friends episode using the models we learned in the previous section:

sndr –K 4 –S 8 –t 0.1 –H 2 –Fcdb0e –p 0.999 \\

-i 11471733.wav -m guys gals laughter music \\

-d friend-seg -D friends-seg-edl

This will analyze the soundfile 11471733.wav using the models guys, gals, laughter and music and a persistence parameter of 0.999. Note how we use the same analysis parameters as in the learning phase. The output of this operation will be eight files. Four of them will be soundfiles named:

11471733.wav.friend-seg.0.wav

11471733.wav.friend-seg.1.wav

11471733.wav.friend-seg.2.wav

11471733.wav.friend-seg.3.wav

Each containing the sound portions what were classified as belonging to classes guys, gals, laughter and music respectively.

Similarly there would be four EDL files named:

11471733.wav.friends-seg-edl.0.edl

11471733.wav.friends-seg-edl.1.edl

11471733.wav.friends-seg-edl.2.edl

11471733.wav.friends-seg-edl.3.edl

which denote the time segments where each class appears.