An Incomplete Guide to Audio Pipeline for AudioSense

Syed Shabih Hasan
syedshabih-hasan@uiowa.edu
Department of Computer Science
The University of Iowa
Iowa City, Iowa 52242

Contents

1 Overview

This document has been created to help researchers get a hold on the Mobile System Laboratory @ The U of I's Audio Processing Toolkit. Before we begin, we would like to acknowledge the freely available software that we have used [1]

- VoiceBox (link)

The audio pipeline allows the user to analyse the audio files we have collected through AudioSense. The pipeline also allows for validation and testing of certain aspects of the pipeline.

[1] In case you find that we have not acknowledged someone, please do let us know

2 Testing and Validation

–TODO–

3 Preprocessing

Preprocessing, in our case, consists of identifying buzzes and beeps from the input signal and removing them when computing features. This is done within the pipeline by using the function `buzzBeepFilter` in `preProcess`. The description of both these functions follows:

3.1 `buzzBeepFilter`

This function is used to remove the buzzes and beeps from the input audio file. The function accepts a single argument (the audio signal), identifies the positions (frame number) of the buzzes and beeps on the signal and returns them.

3.2 `preProcess`

This function is used to extract the signal from the input file and return the buzz and beep locations. The input should given in the form of a full file pathname.

3.3 Framing

An essential component of any pipeline that deals with any continuous signal is the breaking up the signal into frames. We achieve this using the `framing` function. This takes as input the audio signal, the sampling frequency, the size we want the frame to be in seconds (we use 0.02s for most of our calculations), location of buzzes, and location of beeps. The last two are obtained when the `preProcess` function is called with the proper arguments. It returns two binary masks with the same size as the number of frames, these two masks are indicators of which frames have buzzes and beeps present in them.

3.4 Usage

3.4.1 Locating buzzes and beeps

The usage of the `buzzBeepFilter` is encapsulated in the use of the `preProcess` function.

```
[locationOfBuzzes,locationOfBeeps,signal] = ···
preProcess('dummyPath/file.audio');
```

The variables `locationOfBuzzes`, and `locationOfBeeps` contain the location (as sample number) of buzzes and beeps respectively.

3.4.2 Framing the audio signal

The `framing` function works as follows:

```
[buzzMask, beepMask, frames] = ···
framing(signal, frequency, locationOfBuzzes, locationOfBeeps);
```

The outputs `buzzMask`, `beepMask` are the same length as the length of `frames`. For example if the audio signal consists of five frames and the second and fourth frame contain beep and buzz respectively, the `beepMask` and `buzzMask` would be `[0 1 0 0 0]` and `[0 0 0 1 0]` respectively.

4 Feature Extraction

There are several features we extract from the audio ranging from time domain features such as Root-Mean-Square (RMS) values to frequency domain features such as spectral roll-off. We describe them in the following sub sections. A point to note here is that each one of these feature extraction mechanisms work on a single frame.

4.1 Zero Crossing Rate (ZCR)

This is a time domain feature, it represents the number of times that the signal has changed its sign.

4.2 Root Mean Square (RMS)

This is also a time domain feature, it is representative of the energy that the signal carries.

4.3 Entropy

This is a frequency domain feature. The entropy is a representation of the amount of ambient noise present in the signal. A high entropy means that almost all frequencies contribute to the frequency spectrum which generally happens in the presence of ambient noise.

4.4 Spectral Roll-off

This is also a frequency domain feature. Spectral roll-off gives us the value of the frequency below which 93% of the signal lies. This is, of course, configurable by providing the third argument as the fraction in the range [0,1). For example,

```
[srf] = spectralRolloff(fftAbsScaled, freqRange, 0.56);
```

Here 0.56 is indicative of the fact that we want the frequency below which 56% of the signal lies. A point to note here is that `spectralRolloff` takes as input the real part of the FFT (`fftAbsScaled`) in the non-negative frequency range (`freqRange`).

4.5 Mel-Frequency Cepstral Coefficient

MFCC are computed in order to highlight the characteristic features of the audio signal. The number of MFCCs in addition to the 0^{th} coefficient is given as an argument to `extractFrameFeatures` when extracting features.

4.6 Indicative flags

The last three columns in the feature vector are binary, indicating whether the given frame (row) is a low-energy frame, contains a buzz, or contains a beep.

4.6.1 Low Energy

This is calculated by calculating the RMS value of each frame and comparing it with an empirically determined value (threshold). In case the frame has a lower RMS than the threshold, we declare the frame to be a low energy frame.

4.6.2 Buzz Indicator

See 3.4 and 3.3.

4.6.3 Beep Indicator

See 3.4 and 3.3.

4.7 Feature Vector

In case the user decides to use the default `extractFrameFeatures.m` function, the feature vector obtained is of the form
`[patient ID, condition ID, session ID, Zero Crossing Rate, Root Mean Squared Value, Entropy, Spectral Rolloff, Mel-Frequency Cepstral Coefficients, Low Energy Indicator, Buzz Indicator, Beep Indicator]`.

5 Machine Learning

The machine learning part of this pipeline is very open. We are testing with a particular ideology, in case a better method is found, we would include that as well. We shall now describe each of the functions associated with the machine learning portion of the pipeline.

5.1 `getTrueFeatures`

This function acts as a filter, it removes the low energy, buzz and beep containing frames and returns every other frame and the features associated with these frames. The only argument to this function is the matrix that contains all the features, each row is a frame. For feature extraction please see Section 4.

```
trueFeatureMatrix = getTrueFeatures(featureMatrix);
```

5.2 `createTestingTrainingSet`

This function takes as input the `trueFeatureMatrix` and outputs three matrices:

- trainingSet : this is created by random subsampling (without replacement). It is 90% the size of the feature matrix
- testingSet : this is the matrix that is left after elements of the training set have been removed from the original feature matrix
- GMMSet : this is a set created by randomly subsampling the training set with replacement. This would be used in creating the Gaussian Mixture Model object.

```
[trainingSet, testingSet, GMMSet] = ···
createTestingTrainingSet(trueFeatureMatrix);
```

5.3 `getGMMObject`

This is located in the HighLevelFeatures folder. This creates the Gaussian Mixture Model Object. It takes as input four arguments *viz.* `GMMSet` (obtained from Section 5.2), `startIndex`, `endIndex`, and `numberOfModels`. The

arguments `startIndex` and `endIndex` are used to specify the features that are to be used for creating the GMM object. For a list of the features please see Section 4.7.

```
GMMObj = getGMMObject(GMMSet, startIndex, endIndex, numberOfModels);
```

5.4 GMMHistorgram

`GMMHistogram` creates a histogram of the various gaussians per audio file. It looks at each frame and decides which gaussian it belongs to with the highest probability and then increments that model's count. This takes as input four arguments *viz.* `inputSet` (trainingSet or testingSet obtained from Section 5.2), `GMMObject` (obtained from Section 5.3), `startIndex`, `endIndex` (these should be the same as the ones from 5.3).

```
GMMHist = GMMHistogram(inputSet, GMMObject, startIndex, endIndex);
```

5.5 addTargetVariable

This adds a target variable to the GMM Histogram. This needs the survey dataset to map the GMM Histogram to the corresponding target variable in the survey. It takes as arguments the GMMHistogram obtained from Section 5.4, the survey dataset and the name of the variable to map to in the survey.

```
GMMHistogramWithTarget = ···
addTargetVariable(GMMHistogram, surveyDataset, variableName);
```

6 Setting up the pipeline

6.1 A simple feature extraction pipeline

A sample pipeline with explanation of each step in the `bareBonedPipeline/perFilePipeline.m`.