ISOLATED DIGIT RECOGNITION DEMO

What does this demo do?
At a high level, this demo recognises isolated digits from an input utterance using the laptop’s internal microphone. In its current form, this will only recognise the words “one”, “two”, “three”, “four”, “five”, “six”, “seven”, “eight”, “nine”, and “zero”. It is not difficult, however, to modify the demo to recognise different (and more) words.

At a more detailed level it does the following:

TRAINING:
1) Automatically detects an isolated word from your speech (input utterance). It does this by calculating energy and the number of zero crossings on a frame-by-frame basis, and compares these values to a threshold.
2) For each frame in a word, it applies a window function followed by a pre-emphasis filter. It then calculates Mel Frequency Cepstral Coefficients (MFCC) and their delta and delta-delta coefficients for each frame, and uses these as feature vectors.
3) For each digit’s training data, we estimate the parameters of a Gaussian Mixture Model to fit the distribution of training vectors. We therefore train a model to represent each digit.

TESTING
1) Steps (1) and (2) are also applied to the input test utterance. We then calculate the posterior probabilities of the test vectors using each digits Gaussian mixture model, to obtain the negative log likelihood for each digit.
2) These likelihood values are used to classify the input speech.

REFERENCE
This approach is based upon the paper: “Robust Text-Independent Speaker Identification Using Gaussian Mixture Speaker Models”. Rather than speaker identification, however, I am doing word identification. A copy of this paper can be found at: http://www.ece.mcgill.ca/~rrose1//papers/reynolds_rose_sap95.pdf

FURTHER NOTES
This demo has only been tested in a speaker dependent scenario. In other words, I know that it works quite well when I train the demo to recognize my own voice. It should also work reasonably well for anyone else provided that you train the system for your own voice. Details on the training procedure are simple and outlined in this document.

I have not tried training the system with speech from a variety of speakers to make it speaker independent. It may work, I just have not tested it. This would require recording training utterances from many different people which will be quite time consuming.

This has been tested in a standard office environment and seems to work okay. I have not tested the robustness of this system.

I have not implemented a lower likelihood threshold that will cause the system to say “I don’t know”. So if you say something that the system has not been trained to recognise (e.g. “apples”), it will simply return a word from its dictionary that has the greatest likelihood.
Files
- digitrecgui.m: Main m-file to run the GUI
- digitrecgui.fig: Fig file for GUI
- trainmodels.m: Used for training the GMM’s for each digit
- trainscript.m: Script file used for training
- mfcc.m: Calculates MFCC feature vectors
- speechdetect.m: Simple utility function used when training
- Readme.pdf: This document
- gifs\<digit>.gif: Images for display when running the GUI.

Toolboxes Used
- Signal Processing Toolbox
- Statistics Toolbox
- Data Acquisition Toolbox

How does the demo work?
1) You first need to train the system to recognize your voice. This should take no more than 30 minutes of your time. See below for details.
2) Simply run the GUI titled ‘digitrecgui’ and click the “Start” button. As you speak, the GUI will display the digit that it recognises as well as the time domain plot of the isolated word that it was able to detect.

The rest of this document will help you to setup the isolated digit recognition demo to operate on your own voice. Note that this demo has been written to be run on Windows.

Laptop Settings
I noticed that my Lenovo T61 has some internal software settings that filter the acquired signals from the microphone input. If you have something similar on your laptop or PC, it’s best to turn these settings off. For example, on my laptop I go to Start -> Settings -> Control Panel. Double click on SoundMax and select “No filtering”.

![SoundMAX Control Panel](image-url)
You may also need to adjust your sound settings. From Control Panel, double click on “Sounds and Audio Devices”. From the “Audio” tab click the “Volume” button within the “Sound playback” panel and mute the microphone.

From the “Audio” tab click the “Volume” button within the “Sound recording” panel and change the microphone volume level to about halfway.
STEP 1: TRAINING

The first step that is necessary before running the IDR GUI is to train up a model for each digit to be used during classification. This requires multiple recordings of yourself speaking repeated utterances of each digit. So here is what you have to do …

1) Take your laptop, and power adaptor to a quiet office environment suitable for recordings. It must be quiet. It is also important that you have your laptop plugged into power while doing the recordings. For some reason, my T61 records differently when running off the battery which causes my speech detection algorithm to fall over. I have no idea why and have not investigated.

2) Fire up MATLAB and change to the directory where you have saved the IDR demo files.

3) Have a mouthful of water.

4) At the command prompt type:
   ```matlab
   >> y = wavrecord(30*8000,8000);
   ```
   This will record 30 seconds of sound at a sampling rate of 8000Hz. If you like, you can increase the time to capture more training data. While recording, repeat the same digit over and over again with a slight break between each spoken digit. Start at ‘one’.

5) Run the speech detection algorithm over your recording.
   ```matlab
   >> speechdetect(y);
   ```
   This does not do anything, it just checks to see if the training algorithm will be able to accurately isolate your spoken words from the background noise. Visually inspect the plot to make sure that the algorithm does not detect any false positives, i.e., make sure
that we do not mistake some noise for speech. This is not imperative, but too many false positives will corrupt the training data. Detected speech samples will be indicated by a red line plot with a value equal to one. The plot should look something like this:

![Speech detection plot](image)

The speech detection algorithm is quite crude so it will not be perfect. Pan across the plot and make sure that there are not too many false positives. If there are, you may need to repeat steps 4 and 5. You may run into trouble with digits with unvoiced sounds like ‘six’. The algorithm may split the word into two. For training this is not too big a deal, but during testing it will try to recognise two different utterances instead of one. I don’t run into many problems on my laptop with my voice - I’ve only had to make one or two repeat recordings.

If you do run into problems, you may want to adjust the following parameters at the top of the “speechdetect” script.

**std_zxings**: A gain factor used to define the minimum number of zero crossings required in a speech frame to signify speech activity. This speech detection algorithm calculates the number of zero crossings from the first 50 overlapped frames. The threshold is then determined by the maximum of 25 or \( \text{MEAN}(\text{zero crossings}) + \text{std_zxings} \times \text{STD}(\text{zero crossings}) \). Depending on the type of noise you get into your microphone, ‘std_zxings’ may need to be increased to reject noise. Be aware, however, if you increase this too high, you will also reject legitimate unvoiced speech. Provided that you record your training data in a quiet environment, the default value of 0.5 should be okay.

**std_energy**: A gain factor used to define the minimum energy required in a speech frame to signify speech activity. This speech detection algorithm calculates the energy from the first 50 overlapped frames. The threshold is then determined by \( \text{MEAN}(\text{energy}) + \text{std_gain} \times \text{STD}(\text{energy}) \). Provided that you record your training data in a quiet environment, the default value of 0.5 should be okay.
NOTE: THE PARAMETERS “std_zxings” AND “std_energy” SHOULD REMAIN CONSTANT ACROSS ALL RECORDINGS (OF EACH DIFFERENT DIGIT). IF YOU DO NEED TO ADJUST THESE PARAMETERS FOR YOUR SETUP, YOU WILL ALSO NEED TO ADJUST THE SAME PARAMETERS WITHIN ‘trainmodels.m’.

6) If everything is okay, write your recording out to a wave file.
   >> wavwrite(y,8000,’one.wav’);

7) Repeat steps 3-6 for the remaining digits “two”, “three”, “four”, “five”, “six”, “seven”, “eight”, “nine”, and “zero”. When naming your wave files in step 6, you must use these names for the training script (step 8) to run.

8) Once you have all your clean recordings, you can relocate back to your standard noisy working environment. If you needed to make any changes to the default parameters in ‘speechdetect.m’, make the corresponding changes in ‘trainmodels.m’.

   Run “trainscript” to begin the model training process. This will read in each of your wave files (assuming you have named them correctly) and estimate the parameters of a GMM for each digit. These parameters will be saved into a structure ‘models’ and saved to the file ‘MODELS.mat’. This should only take about a minute, if that.

9) That’s it. Training is complete.

STEP 2: TESTING

1) Depending on how noisy your testing environment is, you may need to make changes to the default ‘std_zxings’ and ‘std_energy’ parameters in ‘digitrecgui.m’. These parameters are defined in the ‘handles’ structure. For quiet environments, a value of 0.5 is usually okay. For noisy environments, a value of 1.5 can be used. The higher the value of these parameters, the higher the rejection rate of speech will be. So in noisy environments with higher valued parameters, you may need to speak more loudly into the laptop microphone, or even use an external microphone.

2) Run ‘digitrecgui’. Click on start and begin speaking into the microphone. With any luck, it should recognize your spoken digits. As I mentioned before, it will attempt to classify anything that is spoken. So it will not return an “I don’t know” or “please repeat” answer.

FINAL NOTE:

I have not done any major testing of this demo for robustness. Speech detection and recognition systems can be extremely sensitive to different laptops, different microphones, and different environments. Hopefully this will work for the majority of users. Good luck.