EE 443 Design and Application of Digital Signal Processors
Homework Assignment #3 Spring 2018

Report Due: May 1, 2018

Demo: May 1, 2018

Problem 1: (Matlab) MFCC

MFCCs are commonly used as features of audio signals. Plot the MFCC coefficients of three Bird sounds (Bluejay, Dove, and Ducks) provided in Canvas webpage using Matlab.

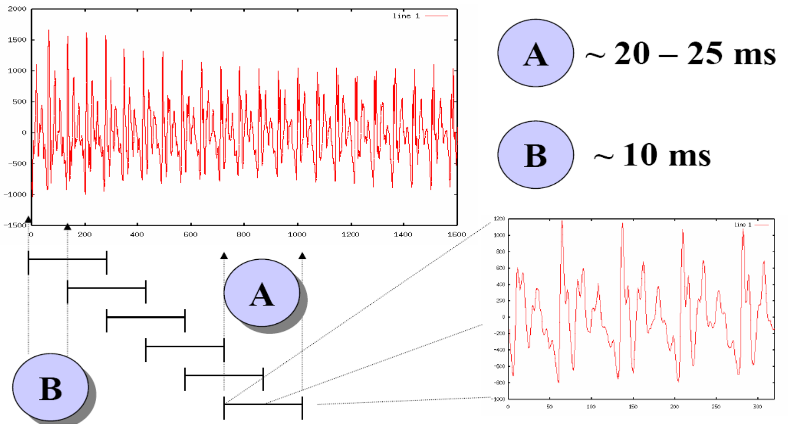

Fig. 1. Framing

1. Do the framing of Bird sounds using A=21ms windows, B=10ms, and A-B=11ms of overlapping. 4 seconds of Bird sounds will generate 380 frames.
2. Generate 13 MFCC coefficients for every frame. Every 4 sec of Bird sound will have 380x13 MFCC coefficients matrix as a result. You can use existing Matlab MFCC library.
3. Plot the 380x13 MFCC coefficients for all three Bird sounds in Matlab.

(Matlab MFCC library: https://www.mathworks.com/matlabcentral/fileexchange/32849-htk-mfcc-matlab?focused=5199998&tab=function)

Problem 2: (Matlab) Training GMM using MFCC features

Gaussian Mixture Model (GMM) helps to cluster the features. Find the GMM parameters of the features found in Problem 1. Since there are three Bird sound, there will be three clusters. You can use existing GMM function provided from Matlab.

`GMModel = fitgmdist(X,k)` returns a Gaussian mixture distribution model (`GMModel`) with k components fitted to data (X).

where X is a MFCC feature matrix found in Problem 1. X has 1140x13 matrix size since it includes all three feature matrices together. k is 3, which is the number of clusters.

(Matlab GMM library: https://www.mathworks.com/help/stats/fitgmdist.html)

Problem 3: (LCDK) Classification of Bird sound using GMM

Use GMM model trained in Problem 2. to classify the Bird sound received through Line input of LCDK.

1. Add library files in a project: fft.h, libmfcc.c, libmfcc.h, gmm.c, gmm.h
2. Write a C program to classify Bird sounds
    a. Receive the trained GMM model data from Matlab through UART, and store the GMM model in GMM gmm[1];
    b. Frame data collection in ISRs.c (21ms corresponds to 256 samples with 12kHz sampling frequency)
    c. FFT of a collected frame data in main.c while() loop
    d. MFCC coefficients calculation (13 coefficients, coeff = 0~12)
        i. `Function: mfcc_result[coeff] = GetCoefficient(spectrum, fs, numFilters, FFTsize, coeff);`
        ii. `spectrum: FFT results`
        iii. `fs: sampling frequency`
        iv. `numFilters: 48`
        v. `FFTsize: Size of FFT (256)`
        vi. `coeff: MFCC coefficient index`
    e. GMM score of captured data using trained GMM parameters
        i. `Function: llh = gmm_score(gmm, mfcc_result, N);`
        ii. `gmm: GMM model trained in Matlab`
        iii. `mfcc_result: a feature vector of frame data`
        iv. `N: Number of frames`
        v. `gmm_score() function prints the probabilities of the input sound to be classified into the clusters`
3. Download Bird sounds (blujay, dove, and mallard duck) from Canvas webpage
4. Play one of the Bird sounds in a PC (or any audio player) in a repeating mode to make sure the LCDK captures enough data.
5. Connect the audio input to line input of LCDK board
6. Run the C program
7. Collect feature vector (mfcc_result).
8. Print the probabilities. The gmm_score() function prints the K probabilities, where K is the number of clusters. It shows the probability that the Bird sound is classified into k-th cluster.

Problem 4: (Matlab) Training SVM for Bird sound classification

Support Vector Machine (SVM) helps to classification of the data. Train the SVM model using the features found in Problem 1. You can use existing SVM function provided from Matlab.

`Mdl = fitcecoc(X,Y)` returns a full, trained ECOC model using the predictors `X` and the class labels `Y`.

where X is a MFCC feature matrix found in Problem 1. X has 1140x13 matrix size. Y is the label vector that has 1140 labels that indicate which frame is trained by which Brid sound.

(Matlab SVM library: https://www.mathworks.com/help/stats/fitcecoc.html)

Problem 5: (LCDK) Classification of Bird sound using SVM

SVM helps to classify the features. Use MFCC features to classify Bird sounds.

1. Add library files in a project: fft.h, libmfcc.c, libmfcc.h, svm.cpp, svm.h
2. Write a C program to classify Bird sounds
   a. Receive the trained SVM model data from Matlab through UART, and store the SVM model in svm_model model[1]; as a global variable.
   b. Frame data collection in ISRs.c (21ms corresponds to 256 samples in 12kHz sampling frequency)
   c. FFT of a collected frame data in main.c while() loop
   d. MFCC coefficients calculation (13 coefficients, coeff = 0~12)
      i. `Function: mfcc_result[coeff] = GetCoefficient(spectrum, fs, numFilters, FFTsize, coeff);`
      ii. `spectrum: FFT results`
      iii. `fs: sampling frequency`
      iv. `numFilters: 48`
      v. `FFTsize: Size of FFT (256)`
      vi. `coeff: MFCC coefficient index`
   e. Print the classification result using SVM model
      i. `Function: predict(mfcc_result, N, D);`
      ii. `mfcc_result: a feature vector of training data`
      iii. `N: Number of frames`
      iv. `D: Number of features`
      v. `predict() function prints the predicted label`
3. Download Bird sounds from Canvas webpage
4. Play one of the Bird sounds in a PC (or any audio player) in a repeating mode to make sure the LCDK captures enough data
5. Connect the audio input to line input of LCDK board
6. Run the C program and print the classification result.

Appendix A. Sample code for MFCC and Classifications

Following code shows the flow of the speech recognition algorithm in main.c and ISRs.c.

<Main.c>

```
#include "DSP_Config.h"
#include <stdio.h>
#include "fft.h"
#include "gmm.h"
#include "libmfcc.h"
#include <math.h>

//#define PI 3.141592
#define BUFFERSIZE 256
int M=BUFFERSIZE;

int kk=0;
int startflag = 0;
int training = 1;

short X[BUFFERSIZE];
COMPLEX w[BUFFERSIZE];
COMPLEX B[BUFFERSIZE];
float feature;
float denominator = 0;
float numerator = 0;
float magnitude = 0;
float avg = 0;

int coeff=0;
double spectrum[BUFFERSIZE];
double mfcc_result[13]={0};
float llh;

GMM gmm[1]; // create GMM model

int main()
{
  int K = 3;  // Number of Classes
  int N = 1;  // Number of Blocks
  int D = 13;  // Number of Features

  DSP_Init();

  int ii, mm, bb, ll;

  // Initialize GMM model
  gmm_new(gmm, K, D, "diagonal");
  gmm_set_convergence_tol(gmm, 1e-6);
  gmm_set_regularization_value(gmm, 1e-6);
  gmm_set_initialization_method(gmm, "random");
  // (P3). Update GMM parameters (gmm[0]) through UART communication with Matlab

  // Twiddle factor
  for(ii=0; ii<M; ii++){
        w[ii].real = cos((float)ii/(float)M*PI);
        w[ii].imag = sin((float)ii/(float)M*PI);
  }

  // main stalls here, interrupts drive operation
  while(1) {
      if(startflag){
         // Remove bias (DC offset)
         avg = 0;
          for(mm=0; mm<M; mm++){
                avg = avg + X[mm];
          }
          // Measure the Magnitude of the input to find starting point
          avg = avg/M;
         magnitude = 0;
          for(mm=0; mm<M; mm++){
                magnitude = magnitude + abs(X[mm]-avg);
          }
          if(magnitude>30000){
               // N-blocks
               for(bb=0;bb<N;bb++){
                     for(ll=0; ll<M; ll++){
                           B[ll].real = X[ll];
                           B[ll].imag = 0;
                     }
                     // (P3). FFT: B is input and output, w is twiddle factors

                     // (P3). Find 13 MFCC coefficients

               }
               // (P3). Print GMM score using gmm model and mfcc_result

          }
        startflag = 0;
      }
   }
}
```

<ISRs.c>

```
#include "DSP_Config.h"

// Data is received as 2 16-bit words (left/right) packed into one
// 32-bit word.  The union allows the data to be accessed as a single
// entity when transferring to and from the serial port, but still be
// able to manipulate the left and right channels independently.

#define LEFT  0
#define RIGHT 1

volatile union {
       Uint32 UINT;
       Int16 Channel[2];
} CodecDataIn, CodecDataOut;

struct cmpx                          //complex data structure used by FFT
    {
    float real;
    float imag;
    };
typedef struct cmpx COMPLEX;


/* add any global variables here */
extern int startflag;
extern int kk;
extern int M;

extern short X[256];

interrupt void Codec_ISR()
///////////////////////////////////////////////////////////////////////
// Purpose:   Codec interface interrupt service routine
```

```
//
// Input:     None
//
// Returns:   Nothing
//
// Calls:     CheckForOverrun, ReadCodecData, WriteCodecData
//
// Notes:     None
///////////////////////////////////////////////////////////////////////
{
	/* add any local variables here */

	if(CheckForOverrun())		// overrun error occurred (i.e. halted DSP)
		return;			// so serial port is reset to recover

	CodecDataIn.UINT = ReadCodecData();		// get input data samples

	/* add your code starting here */
	// I added my code here
	if(kk>M-1){
	  /* (P3). Initialize index kk                                              */

	  /* (P3). Change startflag to start processing in while loop in main()    */

	}

	if(!startflag){
	  /* (P3). Put a new data to the buffer X    */

	  /* (P3). Update index kk                   */

	 }
	// end of my code
	/* end your code here */

	WriteCodecData(CodecDataIn.UINT);		// send output data to  port
}
```