Third Year Project

Generated by Doxygen 1.9.4

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Data Structure Documentation

4.1 Boundary Struct Reference

Data Fields

- float **min_best_ste**
- float **min_worst_ste**
- float **min_best_zc**
- float **min_worst_zc**
- float **max_best_ste**
- float **max_worst_ste**
- float **max_best_zc**
- float **max_worst_zc**
- float **avg_best_ste**
- float **avg_worst_ste**
- float **avg_best_zc**
- float **avg_worst_zc**
- int **count**

The documentation for this struct was generated from the following files:

- /home/bruh/Documents/GitHub/cleaned_and_commented/Seperation/ctest.c/ctest.c
- /home/bruh/Documents/GitHub/cleaned_and_commented/Seperation/ctest.c/ctest_ma_scnd.c

4.2 Bounds Class Reference

Inheritance diagram for Bounds:

Collaboration diagram for Bounds:

The documentation for this class was generated from the following file:

- /home/bruh/Documents/GitHub/cleaned_and_commented/Dynamic_Time_Warping/main.py

4.3 Cluster Struct Reference

Cluster structure :: used to hold the generated 1D Jenks Natural Breaks cluster @centroids can be one or more centroids, these are generated from the @clust function @values these are the values from the original data after it has been sorted and duplicates have been removed the i'th array is associated with the i'th centroid in @centroids @sizes is the length of each @values array with the corresponding index @count is the number of centroids in @centroids @gfv is the Goodness of Variance Fit and is a descriptor of how well the centroids describe the dataset.

```
#include <cluster.h>
```

Data Fields

- float ∗ **centroids**
- float ∗∗ **values**
- int ∗ **sizes**
- int **count**
- float **gfv**

4.3.1 Detailed Description

Cluster structure :: used to hold the generated 1D Jenks Natural Breaks cluster @centroids can be one or more centroids, these are generated from the @clust function @values these are the values from the original data after it has been sorted and duplicates have been removed the i'th array is associated with the i'th centroid in @centroids @sizes is the length of each @values array with the corresponding index @count is the number of centroids in @centroids @gfv is the Goodness of Variance Fit and is a descriptor of how well the centroids describe the dataset.

The documentation for this struct was generated from the following file:

- /home/bruh/Documents/GitHub/cleaned_and_commented/Clustering/cluster.h

4.4 codes Struct Reference

Data Fields

- int **amount**

The documentation for this struct was generated from the following file:

- /home/bruh/Documents/GitHub/cleaned_and_commented/Testing/test.c

4.5 Dataset Class Reference

Inheritance diagram for Dataset:

Collaboration diagram for Dataset:

The documentation for this class was generated from the following file:

- /home/bruh/Documents/GitHub/cleaned_and_commented/Dynamic_Time_Warping/main.py

4.6 Feature_Set Struct Reference

Data Fields

- float ∗ **entropy**
- float ∗ **kurtosis**
- float ∗ **zc**
- float ∗ **ste**
- int **coeffs**

The documentation for this struct was generated from the following file:

- /home/bruh/Documents/GitHub/cleaned_and_commented/Dynamic_Time_Warping/dtw.h

4.7 Guess Struct Reference

the Guess struct is used to store a guess in KNN for sorting and deciding the result @guess index guessed, can be the phoneme, group or voice @diff the difference result used for sorting @ref_indx the MFCC index used to produce the result - used for storing the amount of correct and incorrect guesses for the MFCC

```
#include <knn.h>
```

Collaboration diagram for Guess:

Data Fields

- int **guess**
- double **diff**
- int **ref_indx**
- struct Phoneme ∗ **ref**
- long double **diff**

4.7.1 Detailed Description

the Guess struct is used to store a guess in KNN for sorting and deciding the result @guess index guessed, can be the phoneme, group or voice @diff the difference result used for sorting @ref_indx the MFCC index used to produce the result - used for storing the amount of correct and incorrect guesses for the MFCC

The documentation for this struct was generated from the following files:

- /home/bruh/Documents/GitHub/cleaned_and_commented/Clustering/knn.h
- /home/bruh/Documents/GitHub/cleaned_and_commented/Real_Time/KNN/knn.h

4.8 guess Struct Reference

Data Fields

- long **start**
- long **end**
- char ∗ **name**

The documentation for this struct was generated from the following file:

- /home/bruh/Documents/GitHub/cleaned_and_commented/Testing/test.c

4.9 KNN Class Reference

Inheritance diagram for KNN:

Collaboration diagram for KNN:

The documentation for this class was generated from the following file:

- /home/bruh/Documents/GitHub/cleaned_and_commented/Dynamic_Time_Warping/main.py

4.10 MyApp Class Reference

Inheritance diagram for MyApp:

Collaboration diagram for MyApp:

Public Member Functions

- def **build** (self)

The documentation for this class was generated from the following file:

- /home/bruh/Documents/GitHub/cleaned_and_commented/Dynamic_Time_Warping/main.py

4.11 MyFrame Class Reference

Inheritance diagram for MyFrame:

Collaboration diagram for MyFrame:

Public Member Functions

- def **runProcess** (self)
- def **update_out** (self, text)
- def **run_dtw** (self)
- def **start_dtw_thread** (self)
- def **change** (self, ∗args)

The documentation for this class was generated from the following file:

- /home/bruh/Documents/GitHub/cleaned_and_commented/Dynamic_Time_Warping/main.py

4.12 MyGrid Class Reference

Inheritance diagram for MyGrid:

Collaboration diagram for MyGrid:

Public Member Functions

- def **update_values** (self)
- def **reset_values** (self)
- def **change** (self, ∗args)

The documentation for this class was generated from the following file:

- /home/bruh/Documents/GitHub/cleaned_and_commented/Dynamic_Time_Warping/main.py

4.13 Ph_index Struct Reference

Data Fields

- int **i**
- char **group** [5]
- int **group_i**
- char ∗ **name**
- int **voice**

The documentation for this struct was generated from the following files:

- /home/bruh/Documents/GitHub/cleaned_and_commented/Dynamic_Time_Warping/dtw.h
- /home/bruh/Documents/GitHub/cleaned_and_commented/Real_Time/DTW/dtw.h

4.14 Phoneme Struct Reference

Collaboration diagram for Phoneme:

Data Fields

- struct Ph_index ∗ **index**
- struct Cluster ∗∗ **clust**
- struct Feature_Set ∗∗ **feats**
- float ∗∗ **mfcc_delta**
- float ∗∗ **mfcc_delta_delta**
- float ∗∗ **norm_mfcc**
- double **score**
- float ∗∗ **mfcc**
- long double ∗∗ **sequence**
- int ∗ **size**
- int ∗ **used**
- int ∗ **amounts**
- int **size_count**
- int **reduced_count**
- long double **trained**
- float ∗ **correct**
- float ∗ **error**
- float ∗∗ **raw_time**
- int ∗ **raw_sizes**
- int **raw_count**
- int ∗ **count**
- int **use_count**

The documentation for this struct was generated from the following files:

- /home/bruh/Documents/GitHub/cleaned_and_commented/Dynamic_Time_Warping/dtw.h
- /home/bruh/Documents/GitHub/cleaned_and_commented/Real_Time/DTW/dtw.h

4.15 wav_file Struct Reference

Data Fields

- short ∗ **seq**
- int **length**
- int **offset**

4.15.1 Detailed Description

data and length of data

The documentation for this struct was generated from the following file:

- /home/bruh/Documents/GitHub/cleaned_and_commented/Training/train.h

Chapter 5

File Documentation

5.1 /home/bruh/Documents/GitHub/cleaned_and_commented/↩ Clustering/cluster.c File Reference

A naive Jenks clustering class -.

```
#include "cluster.h"
```

Include dependency graph for cluster.c:

Functions

- float ∗ **breaks** (float ∗data, int length, int amount, int offset)
- float **mean** (float ∗data, int length)
- int **remove_dups** (float ∗data, int length)
- int **remove_zeros** (float ∗data, int length)
- int **fltcomp** (const void ∗a, const void ∗b)
- void ∗ **threadproc** (void ∗arg)
- struct Cluster ∗ **clust** (float ∗data, int length, int amount, int offset, int num)
- float **SDCM** (float ∗data, int length)

Variables

- int **tries** [64] = {0}
- float **EX** [64] = {0}
- float **tries_mult** [64] = {1}

5.1.1 Detailed Description

A naive Jenks clustering class -.

5.2 cluster.h

```
#ifndef CLUSTER_H
#define CLUSTER_H

#include <memory.h>
#include <stdlib.h>
#include <stdio.h>
#include <float.h>
#include <math.h>
#include <time.h>
#include <pthread.h>
#include <unistd.h>

#include "../Misc/realloc.h"
#include "../Dynamic_Time_Warping/dtw.h"

struct Cluster* clust(float* data, int length, int amount, int offset, int num);
float SDCM(float* data, int length);

struct Cluster {
    float* centroids;
    float** values;
    int* sizes;
    int count;
    float gfv;
} Cluster;

#endif
```

5.3 /home/bruh/Documents/GitHub/cleaned_and_commented/↩ Clustering/knn.c File Reference

Contains the functions comparing phonemes using KNN.

`#include "knn.h"`

Include dependency graph for knn.c:

5.4 knn.h

```
#ifndef KNN_H
#define KNN_H

#include <memory.h>
#include <float.h>
#include <stdlib.h>
#include <stdio.h>

struct Guess {
    int guess;
    double diff;
    int ref_indx;
    struct Phoneme* ref;
};

#include "../Dynamic_Time_Warping/dtw.h"
#include "cluster.h"

int knn_mfccs(float** test, int test_length, int k, char* ph);
int knn_mfccs_size(float** total_test, int test_length, int k, char* ph);
int k_means(float* test, int test_length, int k);
int knn_mfccs_size_noref(float** test, int test_length, int k, char* ph);
int knn_mfccs_voice_time(float* test, int test_length, int k, char* ph);
int knn_mfccs_group_time(float* test, int test_length, int k, char* ph);

#endif
```

5.5 knn.h

```
#ifndef KNN_H
#define KNN_H

#include "../Misc/includes.h"

struct Guess {
    int guess;
    long double diff;
    int ref_indx;
     struct Phoneme* ref;
};

int knn_mfccs_size(float* test, int test_length, int k);

#endif
```

5.6 /home/bruh/Documents/GitHub/cleaned_and_commented/Dynamic↩ _Time_Warping/dtw.c File Reference

The main control file of the program which also contains the DTW functions.

`#include "dtw.h"`

Include dependency graph for dtw.c:

Functions

- int **max** (int a, int b)
- int **min** (int a, int b)
- int **compar** (const void ∗a, const void ∗b)
- void **export_phones** (void)
- double ∗∗ init_dtw_matrix (int signal_length, int phone_length, int w)

    Initalises a zeroed matrix for use in DTW.
- void ∗ create_mfcc (void ∗argv)

    Creates all MFCCs for a given phoneme.
- void ∗ **create_clusters** (void ∗argv)
- void **clean** (void)

    Cleans up any used memory before closing the program.
- void **gen_aoo** (void)
- void **export_mfccs** (void)

    Exports all MFCCs in use to files in the main directory.
- void **export_clusters** (void)

    Exports all clusters in use.
- void average_mfccs (int num)

    Averages the MFCC coefficients for each phoneme.
- void **export_for_jenks** (char ∗p, int coef, float ∗data, int data_size)
- void **read_jenks** (void)
- void **export_for_pca** (void)
- void **normal_all** (void)

    Normalises all MFCCs and deltas.
- void **std_test_output** (void)

    Produces standard (non-boundary) tests output to the user.
- void handle_argv (int argc, char ∗argv[ ])

Handles all input arguments provided by the user from the command line.
- void **cluster** (void)
- void **pca** (void)
- void **threaded_mfccs** (void)

  Generates a thread for each phoneme to produce all MFCCs for that phoneme.
- void **min_max_values** (void)

  Assigns the minimum and maximum values found in all MFCCs.
- void mean_size_mfccs (struct Phoneme ∗phone)

  Averages all MFCCs of the same size by phoneme.
- void **frame_variance** (void)

  Calculates and outputs the variance among frames of all MFCCs.
- void export_device (void)

  Exports all MFCCs and config files for use with the physical device.
- void **non_threaded_mfccs** (void)

  Generates a thread for each phoneme to produce all MFCCs for that phoneme, only one thread is used.
- int **seco** (long time)
- int **minu** (long time)
- int **hour** (long time)
- int **main** (int argc, char ∗argv[ ])
- void **dtw_init** (void)
- void **dtw_clust** (float ∗∗signal, int signal_length, struct Phoneme ∗phoneme, short limith)
- double **dtw_clust_result** (float ∗signal, int signal_length, struct Phoneme ∗phoneme, int t, int clust, short limit)
- void dtw_frame (float ∗∗signal, int signal_length, struct Phoneme ∗phoneme, short limit)

  Performs DTW frame-by-frame for MFCCs.
- double **dtw_frame_result** (float ∗signal, int signal_length, struct Phoneme ∗phoneme, int p, short limit)
- double **dtw_frame_result_group** (float ∗∗signal, int signal_length, struct Phoneme ∗phoneme, int p, short limit)
- double **dtw_frame_result_group_time** (float ∗signal, int signal_length, struct Phoneme ∗phoneme, int p, short limit)
- size_t length (short ∗array)

  Returns the length of an array terminated with SHRT_MAX.
- int **is_sil_mean** (short ∗array, int length)
- int **is_sil** (short ∗array, int length)
- int **is_sil_ste_chunk** (short ∗array, int length)
- int **is_sil_ste** (short ∗array, int length)
- int **is_sil_zc** (short ∗array, int length)
- int **is_sil_flat** (short ∗array, int length)
- int **is_sil_db** (short ∗array, int length)
- void **cancel_threads** (void)

  Closes all threads in use.
- void interrupt_handler (int signo)

  Handles a signal interrupt from the user.
- void **mask_sig** (void)

  Masks any signals for additional threads other than main.

Variables

- int **glbl_banks** = 40
- int **glbl_window_width** = 128
- int **glbl_paa_op** = 1
- int glbl_paa = 2

  Applies PAA to the input sequence.
- int **glbl_dtw_window** = 200
- int **glbl_interval_div** = 2
- int **glbl_nfft** = 512
- int **glbl_mfcc_num** = 25
- int **glbl_clust_num** = 0
- int **glbl_k** = 7
- float **glbl_test_trunc** = 24
- int **no_window** = 0
- int **glbl_test_iter** = 1
- int **glbl_group_k** = 7
- int **glbl_voice_k** = 7
- int **glbl_frame_limit** = INT_MAX
- int **glbl_zc_incr**
- int **glbl_ste_incr**
- int **glbl_entr_incr**
- int **glbl_neg_incr**
- int **glbl_larg_incr**
- int **glbl_larg_entr_incr**
- int **glbl_larg_ste_incr**
- int **largest_mfcc** = 0
- int **largest_index** = 0
- float **largest_value** = 0
- float **smallest_value** = FLT_MAX
- short **failed** = 0
- int **DTW_ERROR**
- int **DELTA** = 0
- int **DELTA_DELTA** = 0
- int **NORM** = 0
- int **SIMPLE** = 0
- int **EXTRA** = 0
- int **CLUST** = 0
- int **BOUNDS** = 0
- int **LOG_E** = 0
- int **AVG** = 0
- int **KNN** = 0
- int **GROUP** = 0
- int **VOICED** = 0
- int **GRAM** = 0
- int **PCA** = 0
- int **ZC** = 0
- int **STE** = 0
- int **MEAN_SIZE** = 0
- int **KURT** = 0
- int **ENTR** = 0
- int **SPKR1** = 0
- int **SPKR1_NOSIL** = 0
- int **WHITE** = 0
- int **STRT** = 0

- int **CAR** = 0
- int **CAFE** = 0
- int **SPLIT_DATA** = 0
- int **SVM** = 0
- int **EXPORT** = 0
- int **THREAD** = 0
- int **MALE** = 0
- int **FEMALE** = 0
- int **trained** = 0
- int **tested** = 0
- int **mfcced** = 0
- int **clustered** = 0
- int **exported** = 0
- double **best_so_far** = DBL_MAX
- time_t **avg_test_time** = 0
- long double **total_test_time** = 0
- long double **total_dtw_tests** = 0
- time_t **start** = 0
- pthread_t ∗ **threads** = NULL
- int **thread_count** = 0
- int **clusters_left** = 0

5.6.1 Detailed Description

The main control file of the program which also contains the DTW functions.

Author

T. Buckingham

Date

Wed May 4 15:41:32 2022

This file contains program dependant configuration variables along with all required dynamic time warping files. This file was decided to contain these and

5.6.2 Function Documentation

5.6.2.1 average_mfccs()

```
void average_mfccs (
            int num )
```

Averages the MFCC coefficients for each phoneme.

Parameters

num	The phoneme index.

5.6.2.2 create_mfcc()

```
void * create_mfcc (
             void * argv )
```

Creates all MFCCs for a given phoneme.

Parameters

argv	The phoneme index to create MFCCs for

Returns

Not used

Creates all MFCCs from all raw data signals for the phoneme index provided.

5.6.2.3 dtw_frame()

```
void dtw_frame (
             float ** signal,
             int signal_length,
             struct Phoneme * phoneme,
             short limit )
```

Performs DTW frame-by-frame for MFCCs.

Parameters

signal	The input signal MFCC to be compared
signal_length	The length of
signal	in frames
phoneme	The phoneme to compare to
limit	The DTW window limit

5.6.2.4 export_device()

```
void export_device (
             void )
```

Exports all MFCCs and config files for use with the physical device.

This method produces a folder to be copied onto the device's SD card, all required information will be loaded by the device

5.6.2.5 handle_argv()

```
void handle_argv (
             int argc,
             char * argv[] )
```

Handles all input arguments provided by the user from the command line.

Parameters

argc	The number of arguments provided
argv	The arguments as strings

5.6.2.6 init_dtw_matrix()

```
double ** init_dtw_matrix (
             int signal_length,
             int phone_length,
             int w )
```

Initalises a zeroed matrix for use in DTW.

Parameters

signal_length	The length of the test's sequence MFCC in frames
phone_length	The length of the phoneme's sequence MFCC in frames
w	The windowing limit

Returns

The zero initialised matrix

5.6.2.7 interrupt_handler()

```
void interrupt_handler (
             int signo )
```

Handles a signal interrupt from the user.

Parameters

signo	The signal flag

When a CTRL+C (SIGINT) is sent to the program the used memory is cleaned and the current stats are displayed before exiting

5.6.2.8 length()

```
size_t length (
             short * array )
```

Returns the length of an array terminated with SHRT_MAX.

Parameters

array	The array to find the length of

Returns

The length of

Parameters

array	

No longer used in any function as the size is passed instead of finding it.

5.6.2.9 mean_size_mfccs()

```
void mean_size_mfccs (
             struct Phoneme * phone )
```

Averages all MFCCs of the same size by phoneme.

Parameters

phone	The phoneme to average MFCCs for

5.6.3 Variable Documentation

5.6.3.1 glbl_paa

```
length var glbl_paa = 2
```

Applies PAA to the input sequence.

Parameters

sequence	The input signal to be processed
length	The length of
input	

Returns

The new processed signal of size

The input signal is divided by the PAA amount set by the user, with each grouping being averaged into one value

Parameters

sequence	The input signal to be processed
length	The length of
input	

Returns

The new processed signal of size

The input signal is divided by the PAA amount set by the user, with each grouping being averaged into one value. The same as

5.7 dtw.h

```
#ifndef DTW_H
#define DTW_H

#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
#include <float.h>
#include <string.h>
#include <math.h>
#include <time.h>
#include <signal.h>
#include <fenv.h>
#include <errno.h>
#include <pthread.h>

#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <ftw.h>

#include "../Clustering/cluster.h"

struct Ph_index {
    int i;
    char group[5];
    int group_i;
    char* name;
    int voice;
};

struct Feature_Set {
    float* entropy;
    float* kurtosis;
    float* zc;
    float* ste;
    int coeffs;
    // float flatness;
    // float skewness;
    // float spread;
};

struct Phoneme {
    struct Ph_index* index;
    struct Cluster** clust;
    struct Feature_Set** feats;
    float** mfcc_delta;
    float** mfcc_delta_delta;
    float** norm_mfcc;
    double score;
    float** mfcc;
    long double** sequence;
    // int* coeffs;
    int* size;
    int* used;
    int* amounts;
    int size_count;
    int reduced_count;
    long double trained;
    float* correct;
    float* error;
    float** raw_time;
    int* raw_sizes;
    int raw_count;
} ph;

#include "../Training/train.h"
#include "../Misc/realloc.h"
#include "../Testing/test.h"
#include "../Feature_Extraction/paa.h"
#include "../Feature_Extraction/mfcc.h"
#include "../Feature_Extraction/delta.h"

#define PTHREAD_CANCELED ((void *) -1)

// Global options - may be best to make them local
extern int glbl_banks;
extern int glbl_window_width;
extern int glbl_paa_op;
extern int glbl_paa;
extern int glbl_dtw_window;
extern int glbl_interval_div;
extern int glbl_nfft;
extern int glbl_mfcc_num;
extern int no_window;
extern int glbl_clust_num;
extern int glbl_k;
extern int glbl_test_iter;
extern int glbl_group_k;
extern int glbl_voice_k;
extern int glbl_frame_limit;

extern int glbl_zc_incr;
extern int glbl_ste_incr;
extern int glbl_entr_incr;
extern int glbl_neg_incr;
extern int glbl_larg_incr;
extern int glbl_larg_entr_incr;
extern int glbl_larg_ste_incr;

extern int DELTA;
extern int DELTA_DELTA;
extern float glbl_test_trunc;

#define NO_SIG_LEN 1
extern int DTW_ERROR;
extern int SIMPLE;
extern int MALE;
extern int FEMALE;
extern int EXTRA;
extern int CLUST;
extern int BOUNDS;
extern int Z_ZC;
extern int ONE;
extern int LOG_E;
extern int KNN;
extern int GROUP;
extern int TO_RUN;
extern int GRAM;
extern int NORM;
extern int ZC;
extern int STE;
extern int VOICED;
extern int ENTR;
extern int KURT;
extern int SPKR1;
extern int SPKR1_NOSIL;
extern int WHITE;
extern int STRT;
extern int CAR;
extern int CAFE;
extern int SPLIT_DATA;
extern int SVM;

// Testing externs
extern float* aaomfcc;
extern int aaomfcc_length;
extern double against_all[];

// Output globals
extern double best_so_far;
extern short failed;
extern int trained;
extern int tested;
extern time_t start;

extern long double total_test_time;
extern long double total_dtw_tests;

extern pthread_t* threads;
extern int thread_count;

void dtw(float* signal, int signal_length, struct Phoneme* phoneme, short limit);
void dtw_init();
size_t length(short* array);
void export_phone(struct Phoneme* phoneme);
void dtw_test(float* signal, float* sequence, int signal_length, int signal_length2, short limit);
int seco(long time);
int minu(long time);
int hour(long time);
int is_sil(short* array, int length);
void dtw_frame(float** signal, int signal_length, struct Phoneme* phoneme, short limit);
int min(int a, int b);
int max(int a, int b);
void interrupt_handler (int signo);
int is_sil_db(short* array, int length);
int is_sil_mean(short* array, int length);
void dtw_clust(float** signal, int signal_length, struct Phoneme* phoneme, short limit);
void mask_sig(void);
double dtw_frame_result(float* signal, int signal_length, struct Phoneme* phoneme, int p, short limit);
double dtw_clust_result(float* signal, int signal_length, struct Phoneme* phoneme, int t, int clust,
    short limit);
double dtw_frame_result_group(float** signal, int signal_length, struct Phoneme* phoneme, int p, short
    limit);
double dtw_frame_result_group_time(float* signal, int signal_length, struct Phoneme* phoneme, int p,
    short limit);
int is_sil_zc(short* array, int length);
int is_sil_ste(short* array, int length);
int is_sil_flat(short* array, int length);
int is_sil_ste_chunk(short* array, int length);


#endif
```

5.8 dtw.h

```
#ifndef DTW_H
#define DTW_H

#include "../Misc/includes.h"

struct Ph_index {
    int i;
    char group[5];
    char* name;
};

struct Phoneme {
    struct Ph_index* index;
    double score;
    float** mfcc;
    int* size;
    int* count;
    int size_count;
    int use_count;
} ph;

long double dtw_frame_result(float* signal, int signal_length, struct Phoneme* phoneme, int p);

#endif
```

5.9 /home/bruh/Documents/GitHub/cleaned_and_commented/Dynamic↵ _Time_Warping/jenks.py File Reference

Creates clusters for each phoneme.

Functions

- def **main** (argv)

Variables

- string **rootdir** = "../Jenks/"

5.9.1 Detailed Description

Creates clusters for each phoneme.

Produces clusters for MFCCs using.

Exports all MFCCs to files so that can process them.

Reads in the Jenks centroids generated by.

A K-means function for use with.

is now used instead

Parameters

argv	Not used

Returns

Not used

This function is mostly unused due to the use of

Parameters

signal	The signal to be classified
signal_length	The length of

Parameters

signal	
phoneme	The phoneme to be compared to
limit	The window limit for DTW
p	The phoneme code to be exported
coef	The coefficient number to be exported
data	The MFCC data for
p	and
coef	
data_size	The length of the
data	array

5.10 /home/bruh/Documents/GitHub/cleaned_and_commented/↩ Dynamic_Time_Warping/pca.py File Reference

Exports all MFCCs in use for use with.

Functions

- def **main** (mean, plot)

Variables

- string **rootdir** = "../PCA_Data/"

5.10.1 Detailed Description

Exports all MFCCs in use for use with.

Exports the MFCCs for use with then cleans and exits.

5.11 /home/bruh/Documents/GitHub/cleaned_and_commented/Feature↩ _Extraction/delta.c File Reference

Functions for creating delta and detla-delta coefficients of MFCCs.

```
#include "delta.h"
```
Include dependency graph for delta.c:

Functions

- void normalise_delta (float ∗delta, int length)

  Normalises a delta coefficient array.
- void normalise_delta_delta (float ∗delta, int length)

  Normalises a detal-delta coefficient array.
- void update_deltas_norm (float ∗delta, int length)

  Update the minimum and maximum values of the delta coefficients.
- void update_delta_deltas_norm (float ∗delta_delta, int length)

  Update the minimum and maximum values of the delta-delta coefficients.
- float ∗ delta (float ∗mfcc, int size)

  Calculate the delta coefficients of from an MFCC.
- float ∗ delta_delta (float ∗mfcc_delta, int size)

  Calculate the delta-delta coefficients of from a delta coefficient array.

Variables

- float **min_delta** = FLT_MAX
- float **max_delta** = FLT_MIN
- float **min_delta_delta** = FLT_MAX
- float **max_delta_delta** = FLT_MIN

5.11.1 Detailed Description

Functions for creating delta and detla-delta coefficients of MFCCs.

Author

T. Buckingham

Date

Wed May 4 23:15:48 2022

5.11.2 Function Documentation

5.11.2.1 delta()

```
float ∗ delta (
                float ∗ mfcc,
                int size )
```

Calculate the delta coefficients of from an MFCC.

Parameters

mfcc	The MFCC to process
size	The length of
MFCC	

Returns

The delta coefficient array

5.11.2.2 delta_delta()

```
float * delta_delta (
            float * mfcc_delta,
            int size )
```

Calculate the delta-delta coefficients of from a delta coefficient array.

Parameters

mfcc	The delta coefficient array to process
size	The length of
mfcc_delta	

Returns

The delta coefficient array

5.11.2.3 normalise_delta()

```
void normalise_delta (
            float * delta,
            int length )
```

Normalises a delta coefficient array.

Parameters

delta	The delta array to be normalised
length	The length of @delta

5.11.2.4 normalise_delta_delta()

```
void normalise_delta_delta (
            float * delta,
            int length )
```

Normalises a detal-delta coefficient array.

Parameters

delta	The delta-delta coefficients array to be normalised
length	The length of
delta	

5.11.2.5 update_delta_deltas_norm()

```
void update_delta_deltas_norm (
            float * delta_delta,
            int length )
```

Update the minimum and maximum values of the delta-delta coefficients.

Parameters

delta	The delta-delta sequence to process
length	The length of
delta-delta	

5.11.2.6 update_deltas_norm()

```
void update_deltas_norm (
            float * delta,
            int length )
```

Update the minimum and maximum values of the delta coefficients.

Parameters

delta	The delta sequence to process
length	The length of
delta	

5.12 delta.h

```
#ifndef DELTA_H
#define DELTA_H

#include <memory.h>
#include <stdio.h>
#include <stdlib.h>
#include <float.h>

#include "../Clustering/cluster.h"
#include "../Dynamic_Time_Warping/dtw.h"

float* delta(float* mfcc, int size);
float* delta_delta(float* mfcc_delta, int size);
void update_deltas_norm(float* delta, int length);
void update_delta_deltas_norm(float* delta, int length);
void normalise_delta_delta(float* delta, int length);
void normalise_delta(float* delta, int length);

#endif
```

5.13 /home/bruh/Documents/GitHub/cleaned_and_commented/Feature↩ _Extraction/fft.c File Reference

Fast Fourier transform and Discrete Cosine tranform functions.

`#include "fft.h"`
Include dependency graph for fft.c:

Functions

- float ∗ dct (float ∗array, int width)

  DCT function used during the MFCC process.
- float ∗∗ fft_chunks (float ∗∗chunks, int n, int length)

  Produces an 2 dimensional array of FFT magnitudes.
- float complex ∗ fft (float complex ∗chunk, int length)

  Performs an FFT on a given array.

5.13.1 Detailed Description

Fast Fourier transform and Discrete Cosine tranform functions.

Author

T. Buckingham

Date

Thu May 5 14:48:03 2022

5.13.2 Function Documentation

5.13.2.1 dct()

```
float * dct (
             float * array,
             int width )
```

DCT function used during the MFCC process.

Parameters

array	The MFCC to be processed
width	The length of
array	

Returns

5.13.2.2 fft()

```
float complex * fft (
            float complex * chunk,
            int length )
```

Performs an FFT on a given array.

Parameters

chunk	The complex array to be processed
length	The length of
chunk	

Returns

The FFT in the form of a complex number, no magnitude taken

Before processing the input must be in complex form and the return will be also be in complex form and so the magnitude will need to be taken for further MFCC processing.

5.13.2.3 fft_chunks()

```
float ** fft_chunks (
            float ** chunks,
            int n,
            int length )
```

Produces an 2 dimensional array of FFT magnitudes.

Parameters

chunks	The Hanning window sequences from an audio signal
n	The number of Hanning windows to be processed
length	The length of each Hanning window

Returns

A 2D array of FFT arrays

Each 'chunk' passed will have an FFT applied and the magnitude extracted, the same amount of arrays will be returned however, they will all be of length (

Parameters

length	/ 2)

5.14 fft.h

```
#ifndef FFT_H
#define FFT_H

#include <stdlib.h>
#include <stdio.h>
#include <memory.h>
#include <math.h>
#include <complex.h>

#ifndef M_PI
#define M_PI (3.14159265358979323846)
#endif


float** fft_chunks(float** chunks,int n, int length);
float complex* fft(float complex* chunk, int length);
float* dct(float* array, int width);

#endif
```

5.15 /home/bruh/Documents/GitHub/cleaned_and_commented/Feature←↩ _Extraction/hanning.c File Reference

Functions for creating and applying a Hanning window.

```
#include "hanning.h"
```
Include dependency graph for hanning.c:

Functions

- float ∗ hanning_window (int num)

  Produces a Hanning window of the given length.
- float ∗∗ hanning_chunks (float ∗input, int length, int incr)

  Applies a Hanning window to each given chunk of an audio input.
- float ∗∗ hanning_chunks_no_overlap (float ∗input, int length, int incr)

  Applies a Hanning window to each given chunk of an audio input with no overlaping of the windows.
- float ∗ hanning (float ∗array, int num)

  Applies a Hanning function to a given signal.

5.15.1 Detailed Description

Functions for creating and applying a Hanning window.

Author

T. Buckingham

Date

Thu May 5 17:41:08 2022

5.15.2 Function Documentation

5.15.2.1 hanning()

```
float * hanning (
              float * array,
              int num )
```

Applies a Hanning function to a given signal.

Parameters

array	The input signal to apply the Hanning function to
num	The width of the window

Returns

The result of applying the Hanning function to the given input signal

5.15.2.2 hanning_chunks()

```
float ** hanning_chunks (
              float * input,
              int length,
              int incr )
```

Applies a Hanning window to each given chunk of an audio input.

Parameters

input	The audio signal to be processed
length	The length of
length	
incr	The window width to use

Returns

The array of Hanning windows

The input signal will be blocked into chunks of size

Parameters

incr	then each chunk will have a Hanning window applied.

5.15.2.3 hanning_chunks_no_overlap()

```
float ** hanning_chunks_no_overlap (
              float * input,
              int length,
              int incr )
```

Applies a Hanning window to each given chunk of an audio input with no overlaping of the windows.

Parameters

input	The audio signal to be processed
length	The length of
length	
incr	The window width to use

Returns

The array of Hanning windows

The input signal will be blocked into chunks of size

Parameters

incr	then each chunk will have a Hanning window applied.

5.15.2.4 hanning_window()

```
float * hanning_window (
              int num )
```

Produces a Hanning window of the given length.

Parameters

num	The length of the Hanning window to produce

Returns

The Hanning window of length

Parameters

num	

5.16 hanning.h

```
#ifndef HANNING_H
#define HANNING_H

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <memory.h>
#include "../Training/train.h"
#include "../Seperation/cross_rate.h"

#ifndef M_PI
#define M_PI (3.14159265358979323846)
#endif

float* hanning(float* array, int num);
float* hanning_window(int num);
float** hanning_chunks_no_overlap(float* input, int length, int incr);
float** hanning_chunks(float* input, int length, int incr);

#endif
```

5.17 /home/bruh/Documents/GitHub/cleaned_and_commented/Feature↩_Extraction/mel.c File Reference

Functions for generating a Mel filter bank and applying it to an FFT'ed sequence.

`#include "mel.h"`
Include dependency graph for mel.c:

Functions

- float ∗∗ **mel_fb** (int width, int banks)
- float ∗ mel (float ∗array, int width, int banks)

  Applies a generated Mel filter bank to the input array.

5.17.1 Detailed Description

Functions for generating a Mel filter bank and applying it to an FFT'ed sequence.

Author

T. Buckingham

Date

Thu May 5 14:56:25 2022

5.17.2 Function Documentation

5.17.2.1 mel()

```
float * mel (
              float * array,
              int width,
              int banks )
```

Applies a generated Mel filter bank to the input array.

Parameters

array	The FFT'ed sequence the filter bank is to be applied to
width	The length of
array	
banks	The number of desired filter banks

Returns

The result of applying the Mel filter bank to the input FFT'ed sequence.

5.18 mel.h

```
#ifndef MEL_H
#define MEL_H

#include <memory.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <float.h>

#include "../Clustering/cluster.h"
#include "../Dynamic_Time_Warping/dtw.h"

float** mel_fb(int width, int banks);
float* mel(float* array, int width, int banks);

#endif
```

5.19 /home/bruh/Documents/GitHub/cleaned_and_commented/Feature←_Extraction/mfcc.c File Reference

Functions for creating MFCCs and additional MFCC features, and normalising MFCCs.

`#include "mfcc.h"`
Include dependency graph for mfcc.c:

Functions

- float log_entropy (float ∗sequence, int inc)
  Calculates the log entropy of a given signal.
- float log_energy (float ∗chunk, int length)
  Calculates the log energy of a given signal.
- float st_energy (float ∗chunk, int length)
  Calculates the short time energy of a given signal.
- float kurtosis (float ∗chunk, int length)
  Calculates the kurtosis of a given signal.
- void **normalise_mfcc** (float ∗mfcc, int length)
- void **update_mfcc_norm** (float ∗mfcc, int length)
- float ∗ mfcc (float ∗sequence, int width, int incr, int banks, int paa)
  The control function that generates an MFCC from the given input audio sequence.

Variables

- float **min_mfcc** = FLT_MAX
- float max_mfcc = FLT_MIN
  Normalises an MFCC using.

5.19.1 Detailed Description

Functions for creating MFCCs and additional MFCC features, and normalising MFCCs.

Author

T. Buckingham

Date

Thu May 5 16:06:24 2022

5.19.2 Function Documentation

5.19.2.1 kurtosis()

```
float kurtosis (
            float ∗ chunk,
            int length )
```

Calculates the kurtosis of a given signal.

Parameters

sequence	The signal to be processed.
inc	The length of
sequence.	

Returns

The log kurtosis value.

5.19.2.2 log_energy()

```
float log_energy (
            float * chunk,
            int length )
```

Calculates the log energy of a given signal.

Parameters

sequence	The signal to be processed.
inc	The length of
sequence.	

Returns

The log energy value.

5.19.2.3 log_entropy()

```
float log_entropy (
            float * sequence,
            int inc )
```

Calculates the log entropy of a given signal.

Parameters

sequence	The signal to be processed.
inc	The length of
sequence.	

Returns

The log entropy value.

5.19.2.4 mfcc()

```
float * mfcc (
            float * sequence,
            int width,
            int incr,
            int banks,
            int paa )
```

The control function that generates an MFCC from the given input audio sequence.

Parameters

sequence	The audio sequence to be processed
width	The length of
sequence	
incr	The window width used for Hanning
banks	The number of Mel filter banks to use
paa	The Piece-wise aggregation divisor

Returns

The generated MFCC.

5.19.2.5 st_energy()

```
float st_energy (
            float * chunk,
            int length )
```

Calculates the short time energy of a given signal.

Parameters

sequence	The signal to be processed.
inc	The length of
sequence.	

Returns

The short time energy value.

5.19.3 Variable Documentation

5.19.3.1 max_mfcc

```
min_mfcc and var max_mfcc = FLT_MIN
```

Normalises an MFCC using.

Parameters

mfcc	The MFCC sequence to be normalised.
length	The length of
mfcc	

5.20 mfcc.h

```c
#ifndef MFCC_H
#define MFCC_H

#include "mel.h"
#include "paa.h"
#include "hanning.h"
#include "fft.h"

#include "../Clustering/cluster.h"

float* mfcc(float* sequence, int size, int window, int banks, int paa);
float log_energy(float* chunk, int length);
void update_mfcc_norm(float* mfcc, int length);
void normalise_mfcc(float* mfcc, int length);
float kurtosis(float* chunk, int length);
float log_entropy(float* sequence, int inc);

#endif
```

5.21 mfcc.h

```c
#ifndef MFCC_H
#define MFCC_H

#include "../Misc/includes.h"

int frame_amount(int signal_length);
int mfcc_size(int signal_length);
float* mfcc_quick(float* sequence, int width, int incr, int banks, int paa);

#endif
```

5.22 /home/bruh/Documents/GitHub/cleaned_and_commented/Feature↩ _Extraction/paa.c File Reference

Functions to apply Piece-wise aggregation to an input signal.

```
#include "paa.h"
```

Include dependency graph for paa.c:

Functions

- short ∗ **paa** (short ∗sequence, int length)
- float ∗ **f_paa** (float ∗sequence, int length)

5.22.1 Detailed Description

Functions to apply Piece-wise aggregation to an input signal.

Author

T. Buckingham

Date

Thu May 5 15:50:21 2022

5.23 paa.h

```
#ifndef PAA_H
#define PAA_H

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <memory.h>
#include "../Training/train.h"
#include "../Misc/realloc.h"

short* paa(short* sequence, int length);
float* f_paa(float* sequence, int length);

#endif
```

5.24 /home/bruh/Documents/GitHub/cleaned_and_commented/↩ Misc/realloc.c File Reference

Safe reallocation functions for arrays.

`#include "realloc.h"`

Include dependency graph for realloc.c:

Functions

- short ∗ s_realloc (short ∗input, int length)

  Reallocates @input to size @length.
- float ∗ **f_realloc** (float ∗input, int length)
- int ∗ **i_realloc** (int ∗input, int length)

5.24.1 Detailed Description

Safe reallocation functions for arrays.

Author

T. Buckingham

Date

Fri Apr 29 01:07:49 2022

A range of reallocation functions which contain checks before return the new array. Contains a int, short and float variant.

5.24.2 Function Documentation

5.24.2.1 s_realloc()

```
short * s_realloc (
             short * input,
             int length )
```

Reallocates @input to size @length.

Parameters

input	The array to reallocate
length	The resulting size of the new array

Returns

An array of size @length

5.25 realloc.h

```
#ifndef REALLOC_H
#define REALLOC_H

#include <memory.h>
#include <stdio.h>
#include <stdlib.h>

short* s_realloc(short* input, int length);
float* f_realloc(float* input, int length);
int* i_realloc(int* input, int length);

#endif
```

5.26 mfccs.h

```
#ifndef MFCCS_H
#define MFCCS_H

#include "../Misc/includes.h"

struct Phoneme* get_mfcc(struct Phoneme* phone, int length);
void dtw_init(void);

extern int truncation;
extern struct Phoneme** phones;
extern char* p_codes[];

#endif

```

5.27 includes.h

```
#include <memory.h>
#include <stdlib.h>
#include <math.h>
#include <float.h>
#include <stdlib.h>
#include <limits.h>
#include <string.h>
#include <complex.h>
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>
#include <dirent.h>

#include "../MFCCs/mfccs.h"
#include "../Misc/mfcc_vars.h"
#include "../Test/test.h"
#include "../DTW/dtw.h"
#include "../Feature/mfcc.h"
#include "../Boundary/bounds.h"
#include "../KNN/knn.h"
```

5.28 mfcc_vars.h

```
#ifndef MFCC_VARS_H
#define MFCC_VARS_H

#include "includes.h"

extern int num_ph;
extern int glbl_paa;
extern int glbl_paa_op;
extern int glbl_window_width;
extern int glbl_banks;
extern float glbl_dtw_window;
extern int glbl_interval_div;
extern int glbl_nfft;
extern float glbl_test_trunc;

#endif
```

5.29 rt.h

```
#ifndef RT_H
#define RT_H

#include "./Misc/includes.h"

void record(void);
int check_in(void);

#endif
```

5.30 /home/bruh/Documents/GitHub/cleaned_and_commented/↩ Seperation/bounds.c File Reference

Functions for finding phoneme boundaries in an audio signal.

```
#include "bounds.h"
```

Include dependency graph for bounds.c:

Functions

- float get_entropy (short ∗sequence, int inc)

    Calculates the entropy value of a given audio chunk.
- short ∗ shift_and_reduce (short ∗sequence, int length, int shift)

    Creates a new array and moves the remaining values to the new array.
- int next_boundary (short ∗sequence, int length)

    Finds and returns the next boundary in audio signal.

5.30.1 Detailed Description

Functions for finding phoneme boundaries in an audio signal.

Author

T. Buckingham

Date

Thu May 5 18:05:53 2022

5.30.2 Function Documentation

5.30.2.1 get_entropy()

```
float get_entropy (
          short * sequence,
          int inc )
```

Calculates the entropy value of a given audio chunk.

Parameters

sequence	The audio chunk to process
inc	The length of
sequence	

Returns

The entropy value of

Parameters

sequence	

5.30.2.2 next_boundary()

```
int next_boundary (
             short * sequence,
             int length )
```

Finds and returns the next boundary in audio signal.

Parameters

sequence	The audio signal to process
length	The length of
sequence	

Returns

The next boundary found in the audio sequence

The boundary is found using entropy, zero cross and short time energy change thresholds which are defined by the user when running the program.

5.30.2.3 shift_and_reduce()

```
short * shift_and_reduce (
             short * sequence,
             int length,
             int shift )
```

Creates a new array and moves the remaining values to the new array.

Parameters

sequence	The sequence to procces
length	The length of
sequence	
shift	The amount of data indexed from 0 to remove

Returns

The new array with the data from 0 ->

Parameters

shift	

Once a boundary has been found and processed this data needs to be removed from the array, this removes this data and returns a new array to save memory and prevent a memory leak.

5.31 bounds.h

```c
#ifndef BOUNDS_H
#define BOUNDS_H

#include "../Misc/includes.h"

int shift_and_reduce(short* sequence, int length, int shift);
int next_boundary(short* sequence, int length);
float get_entropy(short* sequence, int inc);
int next_boundary(short* sequence, int length);
int is_positive(short num);
int f_is_positive(float num);

#endif
```

5.32 bounds.h

```c
#ifndef BOUNDS_H
#define BOUNDS_H

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <memory.h>
#include <float.h>
#include "cross_rate.h"
#include "ste.h"
#include "../Feature_Extraction/fft.h"

int next_boundary(short* sequence, int length);
short* shift_and_reduce(short* sequence, int length, int shift);

#endif
```

5.33 /home/bruh/Documents/GitHub/cleaned_and_commented/↩ Seperation/cross_rate.c File Reference

Zero crossing rate functions, primarily used in boundary detection and voice classification.

`#include "cross_rate.h"`

Include dependency graph for cross_rate.c:

Functions

- int is_positive (short num)

    Check if a short is postivie.
- int **f_is_positive** (float num)
- int cross_rate (short ∗signal, int signal_length)

    Calculates the crossing rate of the given signal.
- float favg_cross_rate (short ∗signal, int signal_length)

    Calculates the average cross rate of the given signal.
- float **f_cross_rate** (float ∗signal, int signal_length)
- float **stavg_cross_rate_no_overlap** (float ∗signal, int signal_length)
- float stavg_cross_rate (float ∗signal, int signal_length)

    Calculates the short time zero cross average of a signal signal.

5.33.1 Detailed Description

Zero crossing rate functions, primarily used in boundary detection and voice classification.

Author

T. Buckingham

Date

Tue May 17 00:23:35 2022

5.33.2 Function Documentation

5.33.2.1 cross_rate()

```
int cross_rate (
            short * signal,
            int signal_length )
```

Calculates the crossing rate of the given signal.

Parameters

signal	The signal to process
signal_length	The length of @signal

Returns

The total crossing rate of the signal

5.33.2.2 favg_cross_rate()

```
float favg_cross_rate (
            short * signal,
            int signal_length )
```

Calculates the average cross rate of the given signal.

Parameters

signal	The signal to process
signal_length	The length of @signal

Returns

The averaged crossing rate as a floating point

5.33.2.3 is_positive()

```
int is_positive (
            short num )
```

Check if a short is postivie.

Parameters

num	The number to check

Returns

1 if positive, 0 if negative or zero.

5.33.2.4 stavg_cross_rate()

```
float stavg_cross_rate (
            float * signal,
            int signal_length )
```

Calculates the short time zero cross average of a signal signal.

Parameters

signal	The signal to process
signal_length	The length of @signal

Returns

The short time averaged crossing rate as a floating point

The signal is windowed using a Hanning window, the result from each window is summed then the average of these windows is taken.

5.34 cross_rate.h

```
#ifndef CROSS_RATE_H
#define CROSS_RATE_H

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <memory.h>

#include "../Feature_Extraction/fft.h"
#include "../Feature_Extraction/hanning.h"
#include "../Dynamic_Time_Warping/dtw.h"

int cross_rate(short* signal, int signal_length);
float f_cross_rate(float* signal, int signal_length);
float stavg_cross_rate(float* signal, int signal_length);
float stavg_cross_rate_no_overlap(float* signal, int signal_length);
float favg_cross_rate(short* signal, int signal_length);

#endif

```

5.35 d.h

```
#ifndef D_H
#define D_H

extern int glbl_banks;
extern int glbl_window_width;
extern int glbl_paa_op;
extern int glbl_paa;
extern int glbl_dtw_window;
extern int glbl_interval_div;

#endif
```

5.36 /home/bruh/Documents/GitHub/cleaned_and_commented/↩ Seperation/ste.c File Reference

Functions for calculating the short time energy of a signal.

`#include "ste.h"`
Include dependency graph for ste.c:

Functions

- float short_time_energy (short ∗signal, int signal_length, int window_length)

  Calculates the short time energy of a signal.
- float f_short_time_energy (float ∗signal, int signal_length, int window_length)

  Calculates the short time energy of a signal; this version is for a float input signal.

5.36.1 Detailed Description

Functions for calculating the short time energy of a signal.

Author

T. Buckingham

Date

Thu May 5 17:59:44 2022

5.36.2 Function Documentation

5.36.2.1 f_short_time_energy()

```
float f_short_time_energy (
            float * signal,
            int signal_length,
            int window_length )
```

Calculates the short time energy of a signal; this version is for a float input signal.

Parameters

signal	The signal to process
signal_length	The length of
signal	
window_length	The length of the window to use

Returns

The calculated short time energy

5.36.2.2 short_time_energy()

```
float short_time_energy (
            short * signal,
            int signal_length,
            int window_length )
```

Calculates the short time energy of a signal.

Parameters

signal	The signal to process
signal_length	The length of
signal	
window_length	The length of the window to use

Returns

The calculated short time energy

5.37 ste.h

```
#ifndef STE_H
#define STE_H

#include <stdio.h>
#include <stdlib.h>
#include <memory.h>
#include <float.h>

#include "../Feature_Extraction/hanning.h"

float short_time_energy(short* signal, int signal_length, int window_length);
float f_short_time_energy(float* signal, int signal_length, int window_length);

#endif
```

5.38 /home/bruh/Documents/GitHub/cleaned_and_commented/↩ Testing/test.c File Reference

Contains the functions for classifying test data with the trained model.

`#include "test.h"`
Include dependency graph for test.c:

Data Structures

- struct guess
- struct codes

Functions

- void **export_results_pca** (char ∗ph_code)
- int test_phoneme_utterance (short ∗h, int signal_length)

  Tests files without knowing the boundaries, will use the boundary detection to find them.
- void minimum_edit_distance (char ∗res_filename, char ∗phn_filename)

  Determines the minimum edit distance between a reference and result file.
- void merge_silences (char ∗filename)

  Merges any contiguous silence classifications into one.
- struct codes ∗ get_codes (char ∗filename)

Extracts the labels (start, end and code) from a given file.

- void reset (void)

  Resets all values used for determining accuracy and outputting results.
- void test_phoneme (short ∗h, int signal_length, char ∗p)

  Tests files knowing the boundary.
- void export_results (char ∗ph_code)

  Exports the results of a test. Used for one-to-one testing, not boundary testing.
- void utterance_test (char ∗filename, short ∗sequence, int length, int offset)

  Tests a found phoneme using boundary detection.
- float data_size (void)

  Calculates the size of all MFCCs currently in use.
- float reduce_data (void)
- void bounds_output (FILE ∗m_fp)

  Outputs the boundary test results to the screen, and the results with which paramters to a file.
- void matrix_output (FILE ∗m_fp)

  Outputs the confusion matrix and percentage correct files for use with the given Python scripts.
- void test_output (FILE ∗m_fp)

  Outputs the one-to-one testing to the user.
- struct wav_file ∗ trim_silence (FILE ∗fp, short ∗sequence)

  Trims the silence data from a given .wav file.
- void test (void)

  The main control function of the testing process.

Variables

- int ∗∗ **matrix**
- float ∗ **per_correct**
- int **group_matrix** [7][7] = {0}
- int **voice_matrix** [3][3] = {0}
- int **sil_v_matrix** [3][3]
- int **sil_corr**
- int ∗ **shorted**
- int ∗ **removed**
- int **correct** = 0
- int **fails** = 0
- int **group** = 0
- int **done** = 0
- int **zero_fails** = 0
- long double **min_edit** = 0
- long double **ins** = 0
- long double **delts** = 0
- long double **subs** = 0
- int **tested_files** = 0
- long double **corr** = 0
- long double **WER** = 0
- long double **refs** = 0
- long double **avg_wer** = 0.0
- long double **low_wer** = 0.0
- long double **high_wer** = 0
- long double **worst**
- long double **total_lengths** = 0
- long double **total_dists** = 0

- long double **total_edits** = 0
- long double **worst_case** = 0
- int **iteration_limit** = 0
- int **current_chunk** = 0
- char ∗ **best_file**
- char ∗ **worst_file**

5.38.1 Detailed Description

Contains the functions for classifying test data with the trained model.

Author

T. Buckingham

Date

Wed May 4 20:20:52 2022

5.38.2 Function Documentation

5.38.2.1 bounds_output()

```
void bounds_output (
             FILE * m_fp )
```

Outputs the boundary test results to the screen, and the results with which paramters to a file.

Parameters

m↩ _fp	The file to output the results to.

5.38.2.2 data_size()

```
float data_size (
             void  )
```

Calculates the size of all MFCCs currently in use.

Returns

The current size of all MFCCs

5.38.2.3 export_results()

```
void export_results (
            char * ph_code )
```

Exports the results of a test. Used for one-to-one testing, not boundary testing.

Parameters

ph_code	The code of the phoneme which has been tested.

A result may be exported to correct, group or fail. If the classification is correct then 'correct' will be used, if it is was within the same group then 'group' will be used, if neither then 'fail' will be used. This export will contain what the correct phoneme's score was and what the classifications score was at the top with a list of all tested phonemes with their scores below.

5.38.2.4 get_codes()

```
struct codes * get_codes (
            char * filename )
```

Extracts the labels (start, end and code) from a given file.

Parameters

filename	The name of the filename to extract the labels from

Returns

A codes structure containing a list of phoneme codes in the same order as written in the file.

5.38.2.5 matrix_output()

```
void matrix_output (
            FILE * m_fp )
```

Outputs the confusion matrix and percentage correct files for use with the given Python scripts.

Parameters

m↩_fp	The file to output the matrix and percentages to.

The percentages correct are defined as the amount correct over the amount observed.

5.38.2.6 merge_silences()

```
void merge_silences (
            char * filename )
```

Merges any contiguous silence classifications into one.

Parameters

filename	The name of the file to find and merge silences

Any classifications of subsequent silences results in the same outcome for the user but a different WER and so it is decided that as one or more silences acts the same then they can be combined together.

5.38.2.7 minimum_edit_distance()

```
void minimum_edit_distance (
            char * res_filename,
            char * phn_filename )
```

Determines the minimum edit distance between a reference and result file.

Parameters

res_filename	The result file produced by the testing process
phn_filename	The reference file given by the user in the test data

5.38.2.8 reduce_data()

```
float reduce_data (
            void )
```

Removes any sil MFCCs above a certain frame length as they do not vary much among frames, removes coefficients with minimal variance among phonemes and reduces coefficients with low, but not minimal, variance to a short rather than a float to save space.

Returns

The amount of data reduced by the function

5.38.2.9 reset()

```
void reset (
            void )
```

Resets all values used for determining accuracy and outputting results.

This function's primary use is to reset the values when SPLIT_DATA is in use as the accuracy values and confusions matrices should be per test loop

5.38.2.10 test()

```
void test (
            void )
```

The main control function of the testing process.

Determines and sets the test dataset file path, iterates of the files in the dataset file path, produces and outputs the results of the testing process.

5.38.2.11 test_output()

```
void test_output (
            FILE * m_fp )
```

Outputs the one-to-one testing to the user.

Parameters

m↩ _fp	The file to output the results to.

5.38.2.12 test_phoneme()

```
void test_phoneme (
            short * h,
            int signal_length,
            char * p )
```

Tests files knowing the boundary.

Parameters

h	The signal to be classified
signal_length	The length of
h	
p	The previous phoneme, used for the grammar functions

This is the main one-to-one testing function and can test phonemes using basic DTW, KNN, K-means, etc. A 'complete_signal' can be produced which stores the STE and ZC values for each frame to be used in voice type KNN classification

5.38.2.13 test_phoneme_utterance()

```
int test_phoneme_utterance (
            short * h,
            int signal_length )
```

Tests files without knowing the boundaries, will use the boundary detection to find them.

Parameters

h	The audio signal to test
signal_length	The length of
h	

Returns

The classified phoneme's index

5.38.2.14 trim_silence()

```
struct wav_file * trim_silence (
             FILE * fp,
             short * sequence )
```

Trims the silence data from a given .wav file.

Parameters

fp	The reference file associated with the .wav file
sequence	The associated .wav file in an array

Returns

The new wav structure with silence removed, the size updated and the offset

When using a NOSIL dataset this function returns the structure with the silence removed and provides the offset so that the boundary .res files are correct.

5.38.2.15 utterance_test()

```
void utterance_test (
             char * filename,
             short * sequence,
             int length,
             int offset )
```

Tests a found phoneme using boundary detection.

Parameters

filename	The filename of the reference file
sequence	The found phoneme audio sequence
length	The length of
sequence	
offset	The offset of the file, used when NOSIL is used as the start of testing is no longer the start of the .wav file due to the removal of the leading and trailing silence.

5.39 test.h

```
#ifndef TEST_H
#define TEST_H

#include "../Misc/includes.h"

extern struct Phoneme** phones;

int test_phoneme_utterance(short* h, int signal_length);

#endif
```

5.40 test.h

```
#ifndef TEST_H
#define TEST_H

#include <math.h>
#include "../Clustering/cluster.h"
#include "../Dynamic_Time_Warping/dtw.h"
#include "../Training/train.h"
#include "../Seperation/cross_rate.h"
#include "../Seperation/bounds.h"
#include "../Clustering/knn.h"

void export_results(char* ph_code);
void export_results_aao(char* ph_code);
void test(void);
void test_phoneme(short* h, int signal_length, char* p);
void test_phoneme_aao(short* h, long start_end[2], char* p);
void test_phoneme_pca(short* h, long start_end[2], char* p);
float reduce_data(void);
float data_size(void);
void means(void);

extern int correct;
extern int fails;
extern int group;
extern int sil_corr;
extern int zero_fails;

extern float* per_correct;

extern int* shorted;
extern int* removed;
extern int shted;
extern int rmved;

extern long double refs;
extern long double ins;
extern long double delts;
extern long double subs;

extern int group_matrix[7][7];
extern int voice_matrix[3][3];
extern int sil_v_matrix[3][3];

#endif

```

5.41 /home/bruh/Documents/GitHub/cleaned_and_commented/↩ Training/train.c File Reference

`#include "train.h"`

Include dependency graph for train.c:

Functions

- void **update_zc** (float ∗signal, int signal_length, int n)

- void **update_ste** (short ∗signal, int signal_length, int n)
- void **update_sil** (short ∗signal, int signal_length)
- void **update_sil_db** (short ∗signal, int signal_length)
- void **update_sil_mean** (short ∗signal, int signal_length)
- void **update_sil_flat** (short ∗signal, int signal_length, char ∗filename)
- void **update_sil_ste_frame** (short ∗array, int length, char ∗filename)
- int **mfcc_size** (int signal_length)
- int **frame_amount** (int signal_length)
- double **cubic_interpolate** (short y0, short y1, short y2, short y3, double mu)
- long double **ld_cubic_interpolate** (long double y0, long double y1, long double y2, long double y3, long double mu)
- void train (void)

  is the main control function for training
- short ∗ **resize** (short ∗shorter, size_t s, size_t l)
- long double ∗ **ld_resize** (long double ∗shorter, size_t s, size_t l)
- short ∗ **train_ph_mfcc** (int new, short ∗sequence, struct Phoneme ∗phone)
- void **allocate_ph** (FILE ∗fp, short ∗wav, unsigned char t_t)
- struct wav_file ∗ **read_wav** (FILE ∗fp)
- void **update_sil_ste** (short ∗array, int length, char ∗filename)
- void **update_sil_zc** (short ∗array, int length, char ∗filename)
- float **flatness** (float ∗chunk, int length)

Variables

- struct Phoneme ∗∗ **phones**
- char ∗ p_codes [ ]

  Initliases all phonemes found in.
- char ∗ p_group [ ] = {"STOP", "AFRI", "FRIC", "NASL", "SEMV", "VOWL", "OTHR"}
- int OBSTR = 0
- int **SONOR** = 1
- int **OTHER** = 2
- float ∗ ph_zc_max
- float ∗ **ph_zc_min**
- float ∗ **ste_min**
- float ∗ **ste_max**
- int **limit_changed** = 0
- int **num_tt** = 0
- int **num_ph** = 0
- int max_sil = 0

  used when testing numerous silence detection methods
- int **min_sil** = INT_MAX
- float max_sil_dB = 0

  used when testing numerous silence detection methods
- float **min_sil_dB** = INT_MAX
- float max_sil_zc = 0

  used when testing numerous silence detection methods
- float **min_sil_zc** = INT_MAX
- float max_sil_flt = 0

  used when testing numerous silence detection methods
- float **min_sil_flt** = INT_MAX
- float max_sil_ste = 0

  used when testing numerous silence detection methods
- float **min_sil_ste** = INT_MAX
- float max_sil_mean = 0

  used when testing numerous silence detection methods
- float **min_sil_mean** = INT_MAX

5.41.1 Detailed Description

\Handles training of the phonemes and reading of .wav files

5.41.2 Function Documentation

5.41.2.1 train()

```
train (
            void )
```

is the main control function for training

.PHN file found. These are passed to

If training then the signal is passed to

5.41.3 Variable Documentation

5.41.3.1 max_sil

```
int max_sil = 0
```

used when testing numerous silence detection methods

@max_sil @min_sil are values that store the maximum and minimum raw signal values found in testing

5.41.3.2 max_sil_dB

```
float max_sil_dB = 0
```

used when testing numerous silence detection methods

@max_sil_dB @min_sil_dB are values that store the maximum and minimum raw signal values in decibels found in testing

5.41.3.3 max_sil_flt

```
float max_sil_flt = 0
```

used when testing numerous silence detection methods

@max_sil_flt @min_sil_flt are values that store the maximum and minimum flatness values found in testing

5.41.3.4 max_sil_mean

```
float max_sil_mean = 0
```

used when testing numerous silence detection methods

@max_sil_mean @min_sil_mean are values that store the maximum and minimum mean values found in testing

5.41.3.5 max_sil_ste

```
float max_sil_ste = 0
```

used when testing numerous silence detection methods

@max_sil_ste @min_sil_ste are values that store the maximum and minimum short time energy values found in testing

5.41.3.6 max_sil_zc

```
float max_sil_zc = 0
```

used when testing numerous silence detection methods

@max_sil_zc @min_sil_zc are values that store the maximum and minimum zero cross values found in testing

5.41.3.7 OBSTR

```
int OBSTR = 0
```

@OBSTR @SONOR @OTHER are types of voiceness a phoneme (or sil) can be

5.41.3.8 p_codes

```
char* p_codes[]
```

Initial value:
```
= {"\0", "b", "d", "k", "p", "t", "g",
                  "jh", "ch",
           "s", "sh", "th", "v", "f", "dh", "z",
           "m", "n", "ng",
           "l", "r", "hh", "w", "y",
                  "aa", "ae", "ah", "aw", "er", "ay", "eh", "ey", "ih", "iy", "ow", "oy", "uh", "uw",
           "sil",
             "\0"}
```

Initliases all phonemes found in.

All phonemes are initialised with the index, string code, MFCC array, raw data array, size array and MFCC count.

5.41.3.9 p_group

```
char* p_group[] = {"STOP", "AFRI", "FRIC", "NASL", "SEMV", "VOWL", "OTHR"}
```

@p_group contains a list of phoneme groups

5.41.3.10 ph_zc_max

```
float* ph_zc_max
```

@ph_zc_max @ph_zc_min @ste_min @ste_max All were used to determine if a phoneme should be tested using DTW or KNN if the test's values were not within the range of the phoneme's values then no test would be done

5.42 train.h

```
#ifndef TRAIN_H
#define TRAIN_H

#include <dirent.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <limits.h>
#include <math.h>
#include <sys/types.h>

#include "../Clustering/cluster.h"
#include "../Dynamic_Time_Warping/dtw.h"
#include "../Misc/realloc.h"
#include "../Testing/test.h"
#include "../Feature_Extraction/paa.h"
#include "../Feature_Extraction/mfcc.h"
#include "../Seperation/cross_rate.h"
#include "../Seperation/ste.h"

#define TRAIN 0
#define TEST  1

struct wav_file {
    short* seq;
    int length;
    int offset;
};

double cubic_interpolate(short y0, short y1, short y2, short y3, double mu);
void train(void);
struct wav_file* read_wav(FILE* fp);
void allocate_ph(FILE* fp, short* wav, unsigned char t_t);
short* train_ph(int new, short* sequence, struct Phoneme* ph);
short* init_new_phone(struct Phoneme* phone, short* sequence, int new);
short* resize(short* shorter, size_t s, size_t l);
int mfcc_size(int signal_length);
int frame_amount(int signal_length);
long double* ld_resize(long double* shorter, size_t s, size_t l);
void update_sil_zc(short* array, int length, char* filename);
void update_sil_ste(short* array, int length, char* filename);
float flatness(float* chunk, int length);

extern struct Phoneme** phones;
extern char* p_codes[];
extern char* p_group[];
extern int num_ph;
extern int prev_ph;

extern int limit_changed;

extern float* ph_zc_max;
extern float* ph_zc_min;
extern long double* zc_st_avg_no_oc;
extern float* ste_min;
extern float* ste_max;

extern int max_sil;
extern int min_sil;
extern float max_sil_dB;
extern float min_sil_dB;
extern float max_sil_zc;
extern float min_sil_zc;
extern float max_sil_ste;
extern float min_sil_ste;
extern float max_sil_flt;
extern float min_sil_flt;
extern float max_sil_mean;
extern float min_sil_mean;

#endif
```

Index