DeepSample

Developer Handbook

Andrew Moore, Alex Reno, Hue Truong

DRAFT

Contents

Making the Project

DeepSample uses a Makefile to simplify the compilation process. There are several options that can be used to generate different working binaries. First an overview of the commands:

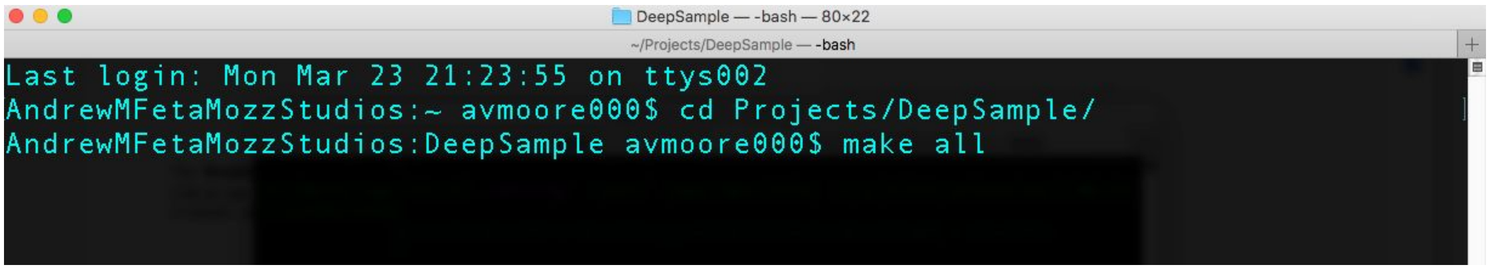
make all:
This command will generate the binaries **DeepSampleTests** and **SampleGenerator.**

make test:
This command will only create the **DeepSampleTests** binary.

make Samples:
This command will only create the **SampleGenerator**

make clean:
This command will clean up all binaries and text files, ignoring the user created results and plot Directories.

To make the project, first navigate to the DeepSample project directory in your terminal. In the root directory of the project ~/DeepSample, enter the desired make command. For example, make all the binaries:



```
Last login: Mon Mar 23 21:23:55 on ttys002
AndrewMFetaMozzStudios:~ avmoore000$ cd Projects/DeepSample/
AndrewMFetaMozzStudios:DeepSample avmoore000$ make all
```

This will make all of the binaries, and place **DeepSampleTests** and **SampleGenerator** in the root directory of the project. You can then proceed to use them as normal.

Programming with the DeepSample Library

DeepSample is a library of functions that allows the user to perform audio segmentatio“n tasks. Right now it is able to handle spectrum flux, zero crossing, and cepstrum algorithms. The audio formats currently supported are **OGG Vorbis**, **FLAC**, and **WAV** file formats.

The following snippet uses the DeepSample library to load and convert an audio file.

```
#include “DeepSample.h”
#include <string>
#include <vector>
#include <complex>
using namespace std;
void main()
{
     AudioWave wave(“test”, 2);
     string inputFile;
     string path;
     string audioDir;
     string sanName;
     int channels;
     bool debug, fullPrecision;

     inputFile = “sample.ogg”;
     path = “pathToOutputFiles”;
     audioDir = “pathToAudioOutputDirectory”;
     sanName = “sample.ogg”;
     channels = 2;
     debug = 1;
     fullPrecision = 1;

     loadAudio(wave, inputFile, audioDir, sanName,
               channels, fullPrecision, path, debug)
     return;
}
```

This is the simplest program that can be written using the DeepSample library. It merely takes in an audio file and converts it to a numerical representation, writing that representation to a file.

For more detailed information on the capabilities of DeepSample, please see the function reference.

Running the Prebuilt Binaries

DeepSampleTests

The **DeepSampleTests** binary contains a suite of test functions that can be used to verify the functionality of the DeepSample library, as well as to experiment around with the algorithms effects on different input files. **DeepSampleTests** has built in help that can be accessed by running it without arguments:

```
./DeepSampleTests
```

This will output a list of commands that can be given to **DeepSampleTests**. The following section goes into the various options in a bit more detail.

Program Use:

```
./DeepSampleTests [resultsDirectory] [inputFile]
                  [outputFile] [channels] [debugMode]
                  [tests]
```

resultsDirectory:

This is a user specified directory where output will be stored. If the directory does not exist it will be created. The directory will be placed within the directory your program is being run.

inputFile:

The audio file for analysis. As of this writing, DeepSample has support for **OGG Vorbis**, **FLAC,** and **WAV** format files.

outputFile:

The name of the file for the main output of the program. This will include all non-debug output.

channels:

The number of channels in the audio file. This is important for allowing the program to work with monaural and stereo sound properly.

fullPrecision:

Used to toggle full precision decimals on and off.

1 = Enable
0 = Disable

save:

Used to toggle the saving of data files on and off.

1 = enable
0 = disable

debug:

Used to toggle debug mode on and off.

1 = enable
0 = disable.

tests:

This number will tell DeepSample which tests you wish to run. The options are as follows:

0 - Run all available tests.
1 - Run Audio Test

```
2 - Run FFT Test
3 - Run Zero Cross Test
4 - Run Spectrum Flux Test
5 - Run Cepstrum Test
6 - Run Spectrum Centroid Test
7 - Run ANN Test
```

SampleGenerator

The **SampleGenerator** binary can be used to generate databases for training ANNI from a given set of audio files. It does not perform any testing of the functions and is meant as a utility allowing users to quickly create training sets for ANNI. Similar to **DeepSampleTests**, **SampleGenerator** has built in help functionality that is accessible by running the program without any arguments:

```
./SampleGenerator
```

This will output information on using the program. This information is described in more detail in the following section.

Program Use:

```
./SampleGenerator [resultsDirectory] [inputDirectory]
                  [outputFileName] [channels] [debugMode]
                  [plot]
```

resultsDirectory:

This is a user specified directory where output will be stored. If the directory does not exist it will be created. The directory will be placed within the directory your program is being run.

inputDirectory:

The directory containing the audio files for analysis. As of this writing, DeepSample has support for **OGG Vorbis**, **FLAC,** and **WAV** format files.

outputFileName:

A prefix that will be used for the output file. This will contain all non-debug general output of the main program.

channels:

The number of channels in the audio file. This is important for allowing the program to work with monaural and stereo sound properly.

1 = Monaural
2 = Stereo

fullPrecision:

Used to toggle full precision decimals on and off.

1 = Enable
0 = Disable

debugMode:

Used to toggle debug mode on and off.

1 = Enable
2 = Disable

plot:

Toggles graph plotting on and off.

```
1 = Plot graphs
2 = No graphing
```

save:

Used to toggle the saving of data files on and off.

```
1 = Enable
2 = Disable
```

Class Reference

AudioWave

Private Member Variables

string **fileName:**

A string indicating the file for data output.

vector<complex<double> > **leftChannel:**

A vector of complex doubles representing the left channel.

vector<complex<double> > **rightChannel:**

A vector of complex doubles representing the right channel.

vector<complex<double> > **leftFFT:**

A vector of complex doubles containing the fourier transform of the left channel.

vector<complex<double> > **rightFFT:**

A vector of complex doubles containing the fourier transform of the right channel.

vector<vector<double> > **cepstrumData:**

A 2D vector of doubles containing the cepstrum results for each channel.

vector<vector<double> > **zeroData:**

A 2D vector of doubles containing the zero cross results for each channel.

vector<double> **max:**

A vector of doubles containing the maximum values of each data vector.

vector<double> **min:**

A vector of doubles containing the minimum values of each data vector.

vector<double> **spectrumCData:**

A vector of doubles containing the spectrum centroid results for each channel.

vector<double> **spectrumFData:**

A vector of doubles containing the spectrum flux results for each channel.

vector<vector<string> > **sourceFiles;**

A vector of strings containing file names used for graphing data.

int **channels:**

An integer indicating the number of channels.

int **frames**:

An integer indicating the number of frames.

Member Functions

Object Manipulation Functions

function **AudioWave**(audioName, chan)

The constructor for an AudioWave object.

Parameters

- **audioName** - A string indicating the full path to an audio file.
- **chan** - An integer indicating the number of channels in the audio file.

Returns: wave - An AudioWave object.

Return Type: object

function ~**AudioWave**()

The destructor for an AudioWave object.

Parameters

Returns:

Return Type:

Initialization Functions

function **setCepstrumData**()

Initializes the cepstrumData member variable.

Parameters

Returns:

Return Type: void

function **setChannels**(chan)

Initializes the channels member variable.

Parameters

- **chan** - An integer indicating the number of channels in the audio file.

Returns:

Return Type: void

function **setFrames**(num)

Initializes the frames member variable.

Parameters

- **num** - An integer indicating the number of frames in the audio file.

Returns:

Return Type: void

function **setName**(audioName)

Initializes the fileName member variable.

Parameters

- **audioName** - A string indicating the full path to an audio file.

Returns:

Return Type: void

function **setLeftFFT**(fft)

Initializes the leftFFT member variable.

Parameters

- **fft** - A vector of complex doubles representing a FFT.

Returns:

Return Type: void

function **setRightFFT**(fft)

Initializes the rightFFT member variable.

Parameters

- **fft** - A vector of complex doubles representing a FFT.

Returns:

Return Type: void

function **setYMaximums()**

Sets the maximum values of the data vectors

Parameters

Returns:

Return Type: void

function **setYMinimums()**

Sets the minimum values of the data vectors.

Parameters

Returns

Return Type: void

function **setSourceFiles**()

Initializes the sourceFiles member variable.

Parameters

Returns:

Return Type: void

function **setZeroData**()

Initializes the zeroData member variable.

Parameters

Returns:

Return Type: void

Update Functions

function **pushCepstrum**(chan, data)

Add a value to cepstrumData member variable.

Parameters

- **chan** - An integer indicating the channel to add data to.
- **Data** - A double containing the data to add.

Returns:

Return Type: void

function **pushLeftChannel**(data)

Add a value to the leftChannel member variable.

Parameters

- **data** - A complex double containing the data to add.

Returns:

Return Type: void

function **pushRightChannel**(data)

Add a value to the rightChannel member variable.

Parameters

- **data** - A complex double containing the data to add.

Returns:

Return Type: void

function **pushSpectrumC**(data)

Add a value to the spectrumCData member variable.

Parameters

- **data** - A double containing the data to add.

Returns:

Return Type: void

function **pushSpectrumF**(data)

Add a value to the spectrumCData member variable.

Parameters

- **data** - A double containing the data to add.

Returns:

Return Type: void

function **pushZero**(chan, data)

Add a value to the zeroData member variable.

Parameters

- **chan** - An integer indicating the channel to add the data to.
- **data** - A double containing the data to add.

Returns:

Return Type: void

Get Functions

function **getFileName**()

Return fileName member variable.

Parameters

Returns: fileName

Return Type: string

function **getSourceFile**(chan, index)

Parameters

- **chan** - An integer indicating the channel to access.
- **index** - An integer indicating the source file to look up.

Returns:

Return Type: string

function **getLeftChannel**()

Return the leftChannel member variable.

Parameters

Returns: leftChannel

Return Type: vector<complex<double> >

function **getLeftFFT**()

Return the leftFFT member variable.

Parameters

Returns: leftFFT

Return Type: vector<complex<double> >

function **getRightChannel**()

Return the rightChannel member variable.

Parameters

Returns: rightChannel

Return Type: vector<complex<double> >

function **getRightFFT**()

Return the rightFFT member variable.

Parameters

Returns: rightFFT

Return Type: vector<complex<double> >

function **getChannelData**(chan, index)

Return a value from specified channel.

Parameters

- **chan** - An integer indicating the channel to access.
- **index** - An integer indicating which index to read from.

Returns: value - A complex double containing the data at the specified index.

Return Type: complex<double>

function **getCepstrumDataPoint**(chan, index)

Return a value from cepstrumData

Parameters

- **chan** - An integer indicating the channel to access.
- **index** - An integer indicating which index to read from.

Returns: dataPoint - A double containing the data at the specified index.

Return Type: double

function **getFFTDataPoint**(chan, index)

Return a value from an FFT vector.

Parameters

- **chan** - An integer indicating the channel to access.
- **index** - An integer indicating which index to read from.

Returns: value - A double containing the data at the specified index.

Return Type: double

function **getSpectrumCDataPoint(**chan)

Returns a value from the spectrumCData member variable.

Parameters

- **chan** - An integer indicating the channel to access.

Returns: dataPoint - A double containing the data at the specified index.

Return Type: double

function **getSpectrumFDataPoint**(chan)

Returns a value from the spectrumFData member variable.

Parameters

- **chan** - An integer indicating the channel to access.

Returns: dataPoint - A double containing the data at the specified index.

Return Type: double

function **getYMaximum(**alg, chan)

Parameters

- **alg** - An integer indicating the algorithm to look up
- **chan** - An integer indicating the channel to look up.

Returns: maxi - A double containing the maximum of the given dataset.

Return Type: double

function **getYMinimum(**alg, chan)

Parameters

- **alg** - An integer indicating the algorithm to look up
- **ahan** - An integer indicating the channel to look up.

Returns: mini - A double containing the minimum of the given dataset.

Return Type: double

function **getZeroDataPoint**(chan, index)

Returns a value from the zeroData member variable.

Parameters

- **chan** - An integer indicating the channel to access.
- **index** - An integer indicating which index to read from.

Returns: value - A double containing the data at the specified index.

Return Type: double

function **getChannels**()

Returns the channel member variable.

Parameters

Returns: channel - An integer indicating the number of channels.

Return Type: int

function **getChannelSize**(chan)

Returns the size of a specific channel.

Parameters

- **chan** - An integer indicating which channel’s size to look up.

Returns: cSize - An integer indicating the size of the specified channel.

Return Type: int

function **getFrames**()

Returns the frames member variable.

Parameters

Returns: frames - An integer indicating the number of Frames.

Return Type: int

function **getCSize**(chan)

Returns the size of the cepstrumData member variable by specific channel.

Parameters

- **chan** - An integer indicating the channel to look up

Returns: cSize - An integer indicating the size of the channel's cepstrumData.

Return Type: int

function **getLeftSize**()

Returns the size of the leftChannel member variable.

Parameters:

Returns: lSize - An integer indicating the size of the leftChannel member variable.

Return Type: int

function **getLeftFFTSize**()

Return the size of the left FFT.

Parameters:

Returns: size - An integer indicating the size of the left FFT

Return Type: int

function **getRightSize**()

Returns the size of the rightChannel member variable.

Parameters:

Returns: rSize - An integer indicating the size of the rightChannel member variable.

Return Type: int

function **getRightFFTSize**()

Return the size of the right FFT.

Parameters:

Returns: size - An integer indicating the size of the right FFT

Return Type: int

function **getSCSize**()

Returns the size of the spectrumCData member variable.

Parameters:

Returns: sSize - An integer indicating the size of the spectrumData member variable.

Return Type: int

function **getSFSize**()

Returns the size of the spectrumFData member variable.

Parameters:

Returns: sSize - An integer indicating the size of the spectrumFData member variable.

Return Type: int

function **getZSize**(chan)

Returns the size of the zeroCrossData member variable.

Parameters

- **chan** - An integer indicating the channel to access.

Returns: zSize - An integer indicating the size of the zeroCrossData member variable.

Return Type: int

Function Reference

audioHandler

function **convertSound**(&wave, fileName, audioDir, sanName,
channels, fullPrecision, path,
debug)

Takes an audio file and converts it to a numerical representation of the waves.

Parameters

- **&wave** - An AudioWave object.
- **fileName** - A string indicating the audio file to load.
- **audioDir** - A string indicating the path to the audio file directory.
- **sanName** - A string indicating the name of the audio file without path information.
- **channels** - An integer indicating the number of channels in the audio file.
- **fullPrecision** - A boolean flag specifying the precision of the output.
- **path** - A string indicating the path for output files.
- **debug** - A boolean flag that controls debug output.

Returns:

Return Type: void

function **loadAudio**(&wave, fileName, audioDir, sanName, channels, fullPrecision, path, debug)

Wrapper function for convertSound

Parameters

- **&wave** - An AudioWave object.
- **fileName** - A string indicating the audio file to load.
- **audioDir** - A string indicating the path to the audio file directory.
- **sanName** - A string indicating the name of the audio file without path information.
- **channels** - An integer indicating the number of channels in the audio file.
- **fullPrecision** - A boolean flag specifying the precision of the output.
- **path** - A string indicating the path for output files.
- **debug** - A boolean flag that controls debug output.

Returns:

Return Type: void

FourierTransform

function **fft**(&wave, save, fileName, path, debug)

A C++ implementation of the Cooley-Tukey Fast Fourier Transform (FFT) algorithm. Fourier transformations are used primarily in signal processing to indicate the frequency in a signal, and its proportion throughout said signal.

Parameters

- **&wave** - An AudioWave object.
- **save** - A boolean flag specifying whether to save data to file.
- **fileName** - A string indicating the file for data output.
- **path** - A string indicating the path for output files.
- **debug** - A boolean flag that controls the debug output.
-

Returns:
Return Type: void

function **inverseFT**(&x, fileName, debug)

Regenerates the audio file based on wave input.

Parameters

- **&x** - A vector of complex doubles representing the fft of an audio file. Must be passed by reference.
- **fileName** - A string containing the name of the output file.
- **debug** - A boolean flag that controls the debug output

Returns:
Return Type: void

cepstrum

Function **rCepstrum**(x)

Perform the cepstrum segmentation algorithm on the given input in accordance to the real cepstrum equation.

Parameters

- **x** - A vector of complex doubles describing an audio wave.
- **windowSize** - Size of hamming window.

Returns: *vector* containing the results

Return Type: vector<double>

function **windowHamming**(n)

Creates a hamming window to be used by the cepstrum algorithm.

Parameters

- **n** - A vector of numbers to be used for the window

Returns: windowSignal - A vector of numbers describing the Window

Return Type: vector<complex<double> >

spectrumCentroid

function **spectralCentroid**(&wave, fileName, path, debug)

Calculates the spectral centroid between each frame of a given wave.

Parameters

- **&wave** - An AudioWave object
- **fileName** - A string indicating the file for data output.
- **path** - A string indicating the path for output files.
- **debug** - A boolean flag that controls debug output.

Returns:

Return Type: void

spectrumFlux

function **spectralFlux**(&wave, fileName, path, debug)

Calculates the spectral flux between each frame of a given audio file.

Parameters

- **&wave** - An AudioWave object.
- **fileName** - A string indicating the file for data output.
- **path** - A string indicating the path for output files.
- **debug** - A boolean flag that controls debug output.

Returns:

Return Type: void

zeroCross

function **zeroCross**(&wave, fileName, path, debug)

Calculates the zero cross of a given audio file.

Parameters

- **&wave** - An AudioWave object.
- **fileName** - A string indicating the file for data output.
- **path** - A string indicating the path for output files.
- **debug** - A boolean flag that controls debug output.

Returns:

Return Type: void

ANN

Main Neural Net Functions

function **ANNI**(testFile, folds, learnRate, epochs, codebooks,
alg, channels, path, debug)

ANNI is the implementation of an artificial neural network (ANN) that is being used to analyze and classify audio files by musical genre.

Parameters

- **testFile** - A string indicating the file containing the samples to identify.
- **folds** - An integer indicating the number of folds to create from the wave object.
- **learnRate** - A double indicating the learning rate to apply to the algorithm.
- **epochs** - An integer indicating the number of epochs to train over.
- **codebooks** - An integer indicating the number of codebooks to use for training and analysis.
- **alg** - An integer indicating which algorithm to run ANNI on.
- **channels** - An integer describing the number of channels in the audio file.
- **path** - A string containing the path for output files.
- **debug** - A boolean flag that controls the debug output.

Returns:

Return Type: void

function **getBestMatch**(knownData, testRow, fileName, path, debug)

Finds the best matching genre for a new audio file by performing a comparison against a database of known files. This is an overloaded function.

Parameters

- **knownData** - Either a vector of floats or a vector of doubles containing the known dataset for use in the comparison.
- **testRow** - Either a vector of floats or a vector of doubles containing the data to be analyzed.
- **fileName** - A string containing the name of the output file
- **path** - A string containing the path for output files.
- **debug** - A boolean flag that controls the debug output.

Returns: match - An integer describing the category the testRow best matches.

Return Type: int

function **learningVectorQuantization**(trainSet, samples, BMUNames,
sampleNames, codeBooks, learnRate, epochs,
fileName, path, debug)

Determines the effectiveness of a training set, and makes a predictive match based on the trained data.

Parameters

- **trainSet** - An n-dimensional vector of doubles representing the training set.
- **samples** - An n-dimensional vector of doubles representing the sample set.
- **BMUNames** - A vector of strings containing the names for the BMU options
- **sampleNames** - A vector of strings containing the names of the samples
- **codebooks** - An integer indicating the number of codebooks to use for training and analysis.
- **learnRate** - A double indicating the learning rate to apply to the algorithm.
- **epochs** - An integer indicating the number of epochs to train over.
- **fileName** - A string indicating the name of the output file.
- **path** - A string containing the path for output files.
- **debug** - A boolean flag that controls the debug output.

Returns:

Return Type: void

function **trainCodeBooks**(trainSet, &codeBookSet, nBooks, lRate, epochs, fileName, path, debug)

Generates a user specified number of codebooks from a set of known data. These codebooks will be used in the matching algorithm.

Parameters

- **trainSet** - An n-dimensional vector of doubles containing the known data for generating the training set.
- **&codeBookSet** - An n-dimensional vector that will store the training set.
- **nBooks** - An integer describing the number of codebooks to generate.
- **lRate** - A double describing the learning rate to use during training.
- **epochs** - An integer describing the number of learning generations
- **fileName** - A string containing the name of the output file.
- **path** - A string containing the path to the output directory
- **debug** - A boolean flag that controls the debug output.

Returns:

Return Type: void

Helper Functions

function **euclideanDistance**(fileName, row1, row2, debug)

Calculates the euclidean distance between row1 and row2.

Parameters

- **fileName** - A string containing the name of the output file.
- **row1** - A vector of floats containing the first row
- **row2** - A vector of floats containing the second row
- **debug** - A boolean flag that controls the debug output.

Returns: distance - A double containing the euclidean distance between the rows.

Return Type: double

function **lvqHelper**(algorithm, testFile, resultsOutput, folds, learnRate, epochs, codeBooks, channels, path, debug)

Parameters

- **algorithm** - A string containing the name of the algorithm being run against.
- **testFile** - A string indicating the name of the file to test
- **resultsOutput** - A string containing the full path to the output file.
- **folds** - An integer indicating the number of folds to create from the wave object.
- **learnRate** - A double indicating the learning rate to apply to the algorithm.
- **epochs** - An integer indicating the number of epochs to train over.
- **codebooks** - An integer indicating the number of codebooks to use for training and analysis.
- **channels** - An integer describing the number of channels in the audio file.
- **path** - A string containing the path for output files.
- **debug** - A boolean flag that controls the debug output.

Returns:

Return Type: void

function **prepareFolds**(folding, folds, alg, currChan, channels, &folded, path, debug)

Breaks a given dataset into the correct number of folds for analysis.

Parameters:

- **folding** - A boolean flag indicating a fold is being prepared
- **folds** - An integer indicating the number of folds to create
- **alg** - An integer specifying the algorithm being worked on
- **currChan** - An integer specifying the current channel being manipulated.
- **channels** - An integer indicating the number of channels in the audio file.
- **&folded** - An n-dimensional vector of doubles that will hold the folded vector
- **path** - A string containing the path to the output directory
- **debug** - A boolean flag that controls the debug output.

Returns:

Return Type: void

function **randomDatabase**(database, &trainSet, path, debug)

Parameters

- **database** - An n-dimensional vector of doubles containing known data points for generating the training set.
- **&trainSet** - An n-dimensional vector of doubles that will hold the randomly generated training set.
- **path** - A string containing the path to the output directory
- **debug** - A boolean flag that controls the debug output.

Returns:

Return Type: void

Utilities

function **graphAlg**(wave. filePrefix, alg, fileName, path, debug)

Plots the results of an algorithm to file.

Parameters

- **wave** - An AudioWave object
- **filePrefix** - A string indicating the prefix for the plot file.
- **alg** - An integer indicating the algorithm to graph.
- **filName** - A string indicating the name of the output file.
- **path -** A string containing the path for output files
- **debug** - A boolean flag that controls debug output.

Returns:

Return Type: void

function **generateScript**(title, xlabel, ylabel, outFileName, sourceFile, channel)

Automates the generation of a gnuplot script file.

Parameters

- **title** - A string containing the title of the graph.
- **xlabel** - A string containing the label for the x-axis.
- **ylabel** - A string containing the label for the y-axis
- **outFileName** - A string specifying the name of the file to output the graph to.
- **sourceFile** - A string specifying the name of the source data file.
- **channel** - An integer specifying which audio channel is being graphed.

Returns:

Return Type: void

Function **genTrainSet**(source, &sink, exclude)

Parameters

- **source** - An n-dimensional vector of doubles containing the source for training
- **&sink** - An n-dimensional vector of doubles that will hold the training set
- **exclude** - An integer indicating the index to skip when creating the set.

Returns:

Return Type: void

function **normalize**(data, &normals, frames, channel, path, debug)

Normalizes a vector.

Parameters

- **data** - A vector of complex doubles describing the audio wave
- **&normals** - A vector of doubles that will contain the normalized vector. Must be passed by reference.
- **frames** - An integer specifying the number of frames to break the data into.
- **channel** - An integer specifying the channel that is being normalized.
- **path** - A string containing the path for output files.
- **debug** - A boolean flag that controls the debug output

Returns:

Return Type: void

function **plotter**(sourceFile, plotFileName, graphType, alg, channel, path)

Graph a given data file.

Parameters

- **sourceFile** - A string containing the name of the file to plot
- **plotFileName** - A string containing the name of the file to save the plot to.
- **graphType** - An integer denoting the type of graph to create.
- **alg** - An integer specifying the algorithm that called the plotter.
- **channel** - An integers specifying the channel being plotted.
- **path** - A string containing the path for output files.

Returns:

Return Type: void

function **printer**(fileName, value, algo, begin, end)

Formats and outputs text to a file.

Parameters

- **fileName** - A string containing the name of the output file.
- **value** - A string to be added to the output file
- **algo** - An integer specifying the algorithm that called the printer
- **begin** - An integer describing the beginning of the printed range
- **end** - An integer describing the end of the printed range

Returns:

Return Type: void

Function **realify**(wave, &reals, outputFile, path, debug)
Parameters

- **wave** - An AudioWave object containing the audiowave being analyzed
- **&reals** - A 2D vector of doubles that will contain the FFT magnitudes
- **outputFile** - A string describing the name of the file for output.
- **path** - A string containing the path for output files.
- **debug** - A boolean flag that controls the debug output

Returns:

Return Type: void

function **createString**(data, fieldWidth)

Generates a string from a given input. This function is Overloaded.

Parameters

- **data** - An integer, double, or boolean to be converted
- **fieldWidth** - An integer specifying the width of the data field.

Returns: newString - A string containing the converted data

Return Type: string

function **timestamp**()

Returns the current system time

Parameters:

Return: **currentTime** - A string containing the current system timestamp.

Return Type: string

function **fileExists**(fileName)

Determines the existence of a file.

Parameters

- **fileName** - A string containing the name of the file to check.

Returns: boolean value denoting existence of file

Return Type: bool

function **sortDist**(v1, v2)

Sorts a list of vectors from greatest to least euclidean Distance.

Parameters

- **v1** - The first vector to sort
- **v2** - The second vector to sort

Returns: isSorted - a boolean declaring the success of the function.

Return Type: bool

function **sign**(test)

Determines the sign of a given number.

Parameters

- **test** - A double containing the number to test.

Returns: result - An integer specifying the sign of the input.

Return Type: int