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:

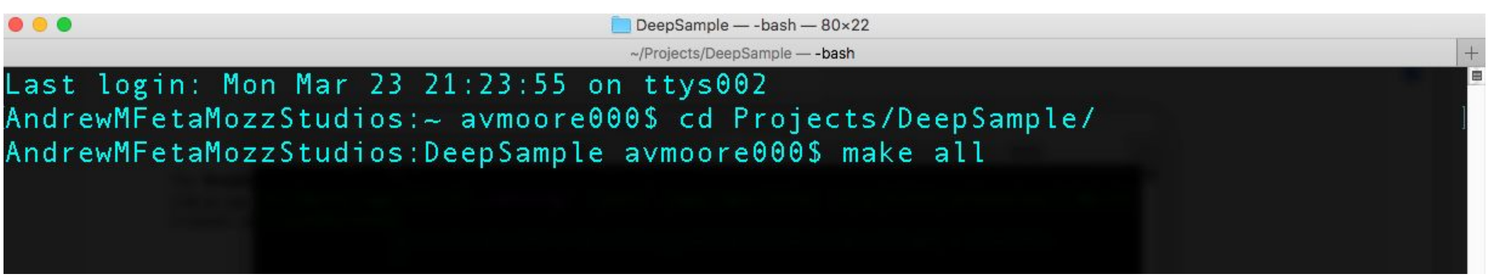
make all:
This command will generate the binaries **DeepSample** and **SampleGenerator.**

make DeepSample:
This command will only create the **DeepSample** 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:



This will make all of the binaries, and place **DeepSample** 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 segmentation 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()
{
     vector<complex<double> > leftChannel;
     vector<complex<double> > rightChannel;
     string inputFile;
     string path;
     string audioDir;
     string sanName;
     int channels;
     bool debug;

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

     loadAudio(inputFile,leftChannel,rightChannel,channels,
               Debug, path, audioDir, sanName);
     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

DeepSample

The **DeepSample** 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. **DeepSample** had built in help that can be accessed by running it without arguments:

./DeepSample

This will output a list of commands that can be given to **DeepSample**. I will also be listing those options here and going into a bit more detail.

Program Use:

./DeepSample [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 monoral and stereo sound properly.

debugMode:

Used to toggle debug mode on and off. 1 to enable debug, 2 to disable.

tests:

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

0 - Runs all available tests.
1 - Run only the zero-cross test.
2 - Run only the spectrum flux test.
3 - Run only the cepstrum test.
4 - Run only the ANNI 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 **DeepSample**, **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

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

Function Reference

audioHandler

function **loadAudio**(fileName, &leftChannel, &rightChannel,
channels, debug, path, audioDir, sanName)

Wrapper function for convertSound

Parameters

- fileName - A string containing the name of the audio file to convert
- leftChannel - A vector of complex doubles to store the left channel of the audio file. It is passed by reference.
- rightChannel - A vector of complex doubles to store the right channel of the audio file. It is passed by reference.
- channels - An integer describing the number of channels
- debug - A boolean flag that controls debug output.
- path - A string containing the path for the debug output file.
- audioDir - A string containing a path for saving individual debug data for the audio file.
- sanName - A string containing the name of the audio file with all path information removed.

Returns:

Return Type: void

function **convertSound**(fileName, &leftChannel, &rightChannel, Channels, debug, path, audioDir, sanName)

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

Parameters

- fileName - A string containing the name of the audio file to convert
- leftChannel - A vector of complex doubles to store the left channel of the audio file. It is passed by reference.
- rightChannel - A vector of complex doubles to store the right channel of the audio file. It is passed by reference.
- channels - An integer describing the number of channels
- debug - A boolean flag that controls debug output.
- path - A string containing the path for the debug output file.
- audioDir - A string containing a path for saving individual debug data for the audio file.
- sanName - A string containing the name of the audio file with all path information removed.

Returns:

Return Type: void

FourierTransform

function **fft**(x, debug, resultDirectory)

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

- x - A vector of complex doubles describing the audio signal
- debug - A boolean flag that controls the debug output.
- resultDirectory - A string containing the path for output files.

Returns:

Return Type: void

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

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.
- debug - A boolean flag that controls the debug output
- fileName - A string containing the name of the output file.

Returns:

Return Type: void

cepstrum

function **cCepstrum**(x)

Performs the cepstrum audio segmentation algorithm on a given input.

Parameters

- x - A vector of complex doubles describing an audio wave.

Returns: *vector* containing the results.
Return Type: vector<complex<double> >

function **realCepstrum**(x)

Filters only the real numbers of the input to a vector.

Parameters

- x - A vector of complex doubles describing an audio wave.

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> >

<u>spectrumFlux</u>

function **spectralFlux**(leftChannel, rightChannel, spectralFlux[],
channels, debug, path)

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

Parameters

- leftChannel - A vector of complex doubles describing the left channel of the audio wave.
- rightChannel - A vector of complex doubles describing the right channel of the audio wave.
- spectralFlux[] - An array of doubles that will hold the result of the spectral flux algorithm.
- channels - An integer describing the number of channels in the audio file.
- debug - A boolean flag that controls the debug output.
- path - A string containing the path for output files.

Returns:
Return Type: void

zeroCross

Function **zeroCross**(leftChannel, rightChannel, &zeroCross, channels, debug, path)

Calculates the zero cross of a given audio file.

Parameters

- leftChannel - A vector of complex doubles describing the left channel of the audio wave.
- rightChannel - A vector of complex doubles describing the right channel of the audio wave.
- &zeroCross - A 2D vector of doubles that will contain the results of the zero cross algorithm. The vector must be passed by reference.
- channels - An integer describing the number of channels in the audio file.
- debug - A boolean flag that controls the debug output.
- path - A string containing the path for output files.

Returns:

Return Type: void

ANN

function **ANNI**(zeroCrossResults, spectrumFluxResults[], path,
audioName, channels, debug)

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

Parameters

- zeroCrossResults - An n-dimensional vector containing the zero cross results of the audio file being analyzed.
- spectrumFluxResults - An array of doubles containing the spectral flux of the audio file
- path - A string containing the path for output files.
- audioName - A string containing the name of the current audio file
- channels - An integer describing the number of channels in the audio file.
- debug - A boolean flag that controls the debug output.

Returns:
Return Type: void

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

Calculates the euclidean distance between row1 and row2.

Parameters

- row1 - A vector of floats containing the first row
- row2 - A vector of floats containing the second row
- path - A string containing the path for output files

- debug - A boolean flag that controls the debug output.

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

Return Type: double

function **getBestMatch**(knownData, testRow, path, channels, 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.
- Path - A string containing the path for the output files
- channels - An integer describing the number of channels in the audio file.
- debug - A boolean flag that controls the debug output.

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

Return Type: int

function **trainCodeBooks**(database, trainSet, nBooks, lRate, epochs, channels, 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

- database - A vector containing known datapoints for generating the training set.

- trainSet - A vector that will contain a subset of the training data for comparisons.
- 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
- 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

Utilities

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 **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 **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 **plotter**(sourceFile, plotFileName, path, graphType, channels, alg)

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

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 **timestamp**()

Returns the current system time

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

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

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

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.
- debug - A boolean flag that controls the debug output
- path - A string containing the path for output files.

Returns:

Return Type: void