Simple Audio Analysis with pyAudioAnalysis

I had a need to do some classification of sets of environmental audio files so started looking for an easy to use Python library that was up for the task. Here are my notes on setting up the pyAudioAnalysis Library for simple audio classification task.

Installation

This library relies on Python 2, so do a quick *python –version* to make sure you are not on Python 3. If both are installed on your machine they may be accessible as *python* and *python3*

Dependencies

You will need to make sure the following modules are installed:

- numpy
- matplotlib
- scipy
- gsl
- mlpy
- scikit-learn (v 0.16.1)
- Simplejson

Note that the Anaconda Python 2.7 distribution has many of these packages already included, but you need *scikit-learn (v 0.16.1)* and the latest *Anaconda* distribution delivers the *0.17.1* version. If you get an error message referencing a missing *hmm* component, you likely have the *0.17.** version of scikit-learn, as *hmm* was deprecated in that version.

When the dependencies are all verified or installed, simply clone the repo into a directory you want to use for development.

```
mkdir ~/audioClassification
cd audioClassification
mkdir trainingData
mkdir sampleData
git clone https://github.com/tyiannak/pyAudioAnalysis.git
```

Training Data

The simplest way to get started it to create sub directories under *trainingData* for each sound category you wish to recognize. For this example lets use these

```
mkdir trainingData/breaking_glass
mkdir trainingData/bubbles
mkdir trainingData/vacuum
```

Next do a Google search looking for free audio files to add to the *trainingData* directory: e.g. *breaking glass audio wav free*, then for bubble sounds try *bubbles audio wav free*, finally for vacuum sounds try *vacuum audio wav free*. Save a dozen or so examples of each sound into the corresponding sub-directory under *trainingData*. The *pyAudioAnalysis* library requires wav files, so make sure any files you save to *trainingData* are wav files.

Sample Data

Next add some audio samples that can be used to test the training. The search is the same as above, but just choose different sample files, so you can test how well the classification model works. You can save them directly under the *sampleData* directory for this example. It is also good to choose a few random other sounds to see how they are interpreted by the classification model.

Scripts

I wrote two quick scripts to do the classification and test a file against the classification model.

createClassifierModel Script

From inside the *audioClassification* directory

```
touch createClassifierModel
chmod +x createClassifierModel
```

In your favorite editor open *createClassifierModel* and add the following code:

```
#!/usr/local/bin/python2
from pyAudioAnalysis import audioTrainTest as aT
import os
from sys import argv
script, dirname = argv

subdirectories = os.listdir(dirname)[:4]
subdirectories.pop(0)
```

```
subdirectories = [dirname + "/" + subDirName for subDirName in subdirectories]

print(subdirectories)
aT.featureAndTrain(subdirectories, 1.0, 1.0, aT.shortTermWindow, aT.shortTermStep, "svm", "svmModel", False)
```

testClassifierModel Script

From inside the *audioClassification* directory

```
touch testClassifierModel
chmod +x testClassifierModel
```

In your favorite editor open *testClassifierModel* and add the following code:

```
#!/usr/local/bin/python2
from sys import argv
import numpy as np
from pyAudioAnalysis import audioTrainTest as aT
script, filename = argv
isSignificant = 0.8 #try different values.

# P: list of probabilities
Result, P, classNames = aT.fileClassification(filename, "svmModel", "svm")
winner = np.argmax(P) #pick the result with the highest probability value.

# is the highest value found above the isSignificant threshhold?
if P[winner] > isSignificant :
  print("File: " +filename + " is in category: " + classNames[winner] + ", with probability: " + str(P[winner]))
else :
  print("Can't classify sound: " + str(P))
```

Run the createClassifierModel Script

To create a classification model simply run the *createClassifierMode* script and pass in the name of the training directory as an argument to the script.

```
createClassifierModel trainingData
```

Below is an example output of the script.

```
        breaking_glass              bubbles             vacuum         OVERALL
C      PRE    REC    F1     PRE    REC    F1     PRE    REC    F1     ACC    F1
0.001  58.1   100.0  73.5   100.0  56.0   71.8   33.3   0.0    0.0    64.0   48.4
0.010  83.7   100.0  91.1   100.0  95.0   97.4   98.5   65.0   78.3   90.0   89.0    best F1        best Acc
0.500  80.0   94.0   86.4   100.0  99.0   99.5   81.8   54.0   65.1   85.2   83.7
1.000  79.8   98.5   88.1   100.0  91.0   95.3   91.9   57.0   70.4   86.2   84.6
```

```
      5.000  79.4   94.5   86.3   100.0  93.0   96.4   81.2   56.0   66.3   84.5   83.0
      10.000 77.7   97.5   86.5   100.0  89.0   94.2   90.0   54.0   67.5   84.5   82.7
Confusion Matrix:
      bre    bub    vac
bre    50.0   0.0    0.0
bub    1.0    23.7   0.3
vac    8.8    0.0    16.2
Selected params: 0.01000
```

Run the testClassifierModel Script

To categorize an audio file simply pass the file name as an argument to the *testClassifierModel* script. In the example below a sample was named *bubbles1.wav*

```
testClassifierModel sampleData/bubbles1.wav
```

Below is an example output of the script.

```
File: sampleData/bubbles1.wav is in category: bubbles, with probability: 0.95486693161
```

Potential GOTCHAS

One issue is that not all *wav* files will be understood by the *wavfile.py* library. If the classification hangs or aborts then it is possible one of the downloaded wav files is the issue. I found the simple solution was to add a line to *wavefile.py* that printed the filename currently being worked on so I could see what was happening the first time using a new set of training files.

```
for me this is the location
/usr/local/lib/python2.7/site-packages/scipy/io/wavfile.py

datadef read(filename, mmap=False):
   """
   Return the sample rate (in samples/sec) and data from a WAV file

   Parameters
   ----------
   filename : string or open file handle
      Input wav file.
   mmap : bool, optional
      Whether to read data as memory mapped.
      Only to be used on real files (Default: False)

      .. versionadded:: 0.12.0

   Returns
```

```
-------
rate : int
    Sample rate of wav file
data : numpy array
    Data read from wav file

Notes
-----
* The file can be an open file or a filename.
* The returned sample rate is a Python integer.
* The data is returned as a numpy array with a data-type determined
  from the file.
* This function cannot read wav files with 24 bit data.

"""
print("FILENAME: " + filename)
```

Conclusion

This was a basic intro to help you get started with pyAudioAnalysis library, check out the project wiki page for more use case for the library including other classification model options, as well as feature extraction, segmentation and visualization capabilities.

April 08, 2016

Tweet

« Org Export Configurations

Design, development, productivity -- notes and explorations.

- Home
- Archives
- About
- Feed

- Twitter
- LinkedIn

Search

Recent Posts

- Simple Audio Analysis with pyAudioAnalysis
- Org Export Configurations
- Deft + Org for Notes
- Org Mode ES2015+ Code Blocks (updated)
- ANCS Example on BLE Nano
- ANCS Message Display
- Org Mode ES2015+ Code Blocks
- ESP8266 Initial Notes
- Self Updating Edison Apps
- Workbench Replaces My Desk

Popular Posts

- Simple Audio Analysis with pyAudioAnalysis
- Getting RKLogConfigureByName working in RubyMotion
- Screen capture goodness...
- CoreData.SQLDebug for RubyMotion
- Wiki + searchable notes.
- Spreadsheet Driven Web Apps
- Treetop and Parsing Expression Grammars (PEGs)
- Fun with Literate CoffeeScript
- User Stories and Activity Diagrams
- d3 charts - wrapping 'NVD3 Charts' in a web component