pyAudioAnalysis/**utilities.py**

def isfloat(x):
"""
Check if argument is float

def isint(x):
"""
Check if argument is int

def isNum(x):
"""
Check if string argument is numerical
"""

def peakdet(v, delta, x = None):
"""
Converted from MATLAB script at http://billauer.co.il/peakdet.html

Returns two arrays

function [maxtab, mintab]=peakdet(v, delta, x)
%PEAKDET Detect peaks in a vector
% [MAXTAB, MINTAB] = PEAKDET(V, DELTA) finds the local
% maxima and minima ("peaks") in the vector V.
% MAXTAB and MINTAB consists of two columns. Column 1
% contains indices in V, and column 2 the found values.
%
% With [MAXTAB, MINTAB] = PEAKDET(V, DELTA, X) the indices
% in MAXTAB and MINTAB are replaced with the corresponding
% X-values.
%
% A point is considered a maximum peak if it has the maximal
% value, and was preceded (to the left) by a value lower by
% DELTA.

pyAudioAnalysis/**audioVisualization.py**

def generateColorMap():
'''
This function generates a 256 jet colormap of HTML-like hex string colors (e.g. FF88AA)

'''

def levenshtein(str1, s2):
'''
Distance between two strings
'''

def textListToColors(names):
'''
Generates a list of colors based on a list of names (strings). Similar strings correspond to similar colors.
'''

def textListToColorsSimple(names):
'''
Generates a list of colors based on a list of names (strings). Similar strings correspond to similar colors.
'''

def chordialDiagram(fileStr, SM, Threshold, names, namesCategories):
'''
Generates a d3js chordial diagram that illustrates similarites
'''

def visualizeFeaturesFolder(folder, dimReductionMethod, priorKnowledge = "none"):
'''
This function generates a chordial visualization for the recordings of the provided path.
ARGUMENTS:
- folder: path of the folder that contains the WAV files to be processed
- dimReductionMethod: method used to reduce the dimension of the initial feature space before computing the similarity. "pca"
- priorKnowledge: if this is set equal to "artist"
'''

pyAudioAnalysis/**audioBasicIO.py**

def convertDirMP3ToWav(dirName, Fs, nC, useMp3TagsAsName = False):
'''
This function converts the MP3 files stored in a folder to WAV. If required, the output names of the

WAV files are based on MP3 tags, otherwise the same names are used.
ARGUMENTS:
- dirName: the path of the folder where the MP3s are stored
- Fs: the sampling rate of the generated WAV files
- nC: the number of channesl of the generated WAV files
- useMp3TagsAsName: True if the WAV filename is generated on MP3 tags
'''

def convertFsDirWavToWav(dirName, Fs, nC):
'''
This function converts the WAV files stored in a folder to WAV using a different sampling freq and number of channels.
ARGUMENTS:
- dirName: the path of the folder where the WAVs are stored
- Fs: the sampling rate of the generated WAV files
- nC: the number of channesl of the generated WAV files
'''

def readAudioFile(path):
'''
This function returns a numpy array that stores the audio samples of a specified WAV of AIFF file
'''

def stereo2mono(x):
'''
This function converts the input signal (stored in a numpy array) to MONO (if it is STEREO)
'''

pyAudioAnalysis/**audioAnalysisRecordAlsa.py**

def **recordAudioSegments**(RecordPath, BLOCKSIZE):
# This function is used for recording audio segments (until ctr+c is pressed)
# ARGUMENTS:
# - RecordPath: the path where the wav segments will be stored
# - BLOCKSIZE: segment recording size (in seconds)
#
NOTE: filenames are based on clock() value

def **recordAnalyzeAudio**(duration, outputWavFile, midTermBufferSizeSec, modelName, modelType):
'''
recordAnalyzeAudio(duration, outputWavFile, midTermBufferSizeSec, modelName, modelType)

This function is used to record and analyze audio segments, in a fix window basis.

ARGUMENTS:
- duration total recording duration
- outputWavFile path of the output WAV file
- midTermBufferSizeSec (fix)segment length in seconds
- modelName classification model name
- modelType classification model type

```
def main(argv):
if argv[1] == '-recordSegments': # record input
if (len(argv)==4): # record segments (until ctrl+c pressed)
recordAudioSegments(argv[2], float(argv[3]))
else:
print "Error.\nSyntax: " + argv[0] + " -recordSegments <recordingPath> <segmentDuration>"
if argv[1] == '-recordAndClassifySegments': # record input
if (len(argv)==6): # recording + audio analysis
duration = int(argv[2])
outputWavFile = argv[3]
modelName = argv[4]
modelType = argv[5]
if modelType not in ["svm", "knn"]:
raise Exception("ModelType has to be either svm or knn!")
if not os.path.isfile(modelName):
raise Exception("Input modelName not found!")
recordAnalyzeAudio(duration, outputWavFile, 2.0, modelName, modelType)
```

pyAudioAnalysis/**analyzeMovieSound.py**

```
def classifyFolderWrapper(inputFolder, modelType, modelName, outputMode=False):
# modelType=='svm' or 'knn':
# Read files from a folder and classify
# Print class histogram
```

```
def getMusicSegmentsFromFile(inputFile):
modelType = "svm"
modelName = "data/svmMovies8classes"
```

```
def analyzeDir(dirPath):
for i,f in enumerate(glob.glob(dirPath + os.sep + '*.wav')): # for each WAV file
getMusicSegmentsFromFile(f)
[c, P]= classifyFolderWrapper(f[0:-4] + "_musicSegments", "svm", "data/svmMusicGenre8", False)
```

def main(argv):

if argv[1]=="--file":
elif argv[1]=="--dir":
elif argv[1]=="--sim":
elif argv[1]=="--loadsim":
elif argv[1]=="--loadsim":
elif argv[1]=="--audio-event-dir":

pyAudioAnalysis/**audioAnalysis.py**

def dirMp3toWavWrapper(directory, samplerate, channels):

def dirWAVChangeFs(directory, samplerate, channels):

def featureExtractionFileWrapper(wavFileName, outFile, mtWin, mtStep, stWin, stStep):

def beatExtractionWrapper(wavFileName, plot):

def featureExtractionDirWrapper(directory, mtWin, mtStep, stWin, stStep):

def featureVisualizationDirWrapper(directory):

def fileSpectrogramWrapper(wavFileName):

def fileChromagramWrapper(wavFileName):

def trainClassifierWrapper(method, beatFeatures, directories, modelName):

def trainRegressionWrapper(method, beatFeatures, dirName, modelName):

def classifyFileWrapper(inputFile, modelType, modelName):

def regressionFileWrapper(inputFile, modelType, modelName):

def classifyFolderWrapper(inputFolder, modelType, modelName, outputMode=False):

def regressionFolderWrapper(inputFolder, modelType, modelName):
def trainHMMsegmenter_fromfile(wavFile, gtFile, hmmModelName, mtWin, mtStep):
def trainHMMsegmenter_fromdir(directory, hmmModelName, mtWin, mtStep):
def segmentclassifyFileWrapper(inputWavFile, modelName, modelType):
def segmentclassifyFileWrapperHMM(wavFile, hmmModelName):
def segmentationEvaluation(dirName, modelName, methodName):
def silenceRemovalWrapper(inputFile, smoothingWindow, weight):
def speakerDiarizationWrapper(inputFile, numSpeakers, useLDA):
def thumbnailWrapper(inputFile, thumbnailWrapperSize):
def parse_arguments():
def trainHMMsegmenter_fromdir(directory, hmmModelName, mtWin, mtStep):
def segmentclassifyFileWrapperHMM(wavFile, hmmModelName):

pyAudioAnalysis/**audioFeatureExtraction.py**

```
def stZCR(frame):
"""Computes zero crossing rate of frame"""

def stEnergy(frame):
"""Computes signal energy of frame"""

def stEnergyEntropy(frame, numOfShortBlocks=10):
"""Computes entropy of energy"""

def stSpectralCentroidAndSpread(X, fs):
"""Computes spectral centroid of frame (given abs(FFT))"""

def stSpectralEntropy(X, numOfShortBlocks=10):
"""Computes the spectral entropy"""

def stSpectralFlux(X, Xprev):
"""
Computes the spectral flux feature of the current frame
ARGUMENTS:
X: the abs(fft) of the current frame
Xpre: the abs(fft) of the previous frame

def stSpectralFlux(X, Xprev):
"""
Computes the spectral flux feature of the current frame
ARGUMENTS:
X: the abs(fft) of the current frame
Xpre: the abs(fft) of the previous frame


def stSpectralRollOff(X, c, fs):
"""Computes spectral roll-off"""

def stHarmonic(frame, fs):
"""
Computes harmonic ratio and pitch
"""

def mfccInitFilterBanks(fs, nfft):
"""
Computes the triangular filterbank for MFCC computation (used in the stFeatureExtraction function before the stMFCC function call)
This function is taken from the scikits.talkbox library (MIT Licence):
https://pypi.python.org/pypi/scikits.talkbox
"""
```

def stMFCC(X, fbank, nceps):
"""
Computes the MFCCs of a frame, given the fft mag

ARGUMENTS:
X: fft magnitude abs(FFT)
fbank: filter bank (see mfccInitFilterBanks)
RETURN
ceps: MFCCs (13 element vector)

Note: MFCC calculation is, in general, taken from the scikits.talkbox library (MIT Licence),
with a small number of modifications to make it more compact and suitable for the pyAudioAnalysis Lib
"""
def stChromaFeaturesInit(nfft, fs):
"""
This function initializes the chroma matrices used in the calculation of the chroma features
"""

def stChromaFeatures(X, fs, nChroma, nFreqsPerChroma):
#TODO: 1 complexity
#TODO: 2 bug with large windows

def stChromagram(signal, Fs, Win, Step, PLOT=False):
"""
Short-term FFT mag for spectogram estimation:
Returns:
a numpy array (nFFT x numOfShortTermWindows)
ARGUMENTS:
signal: the input signal samples
Fs: the sampling freq (in Hz)
Win: the short-term window size (in samples)
Step: the short-term window step (in samples)
PLOT: flag, 1 if results are to be ploted
RETURNS:
"""
def phormants(x, Fs):

def beatExtraction(stFeatures, winSize, PLOT=False):
"""
This function extracts an estimate of the beat rate for a musical signal.
ARGUMENTS:
- stFeatures: a numpy array (numOfFeatures x numOfShortTermWindows)
- winSize: window size in seconds

RETURNS:
- BPM: estimates of beats per minute
- Ratio: a confidence measure
"""
def stSpectogram(signal, Fs, Win, Step, PLOT=False):
"""
Short-term FFT mag for spectogram estimation:
Returns:
a numpy array (nFFT x numOfShortTermWindows)
ARGUMENTS:
signal: the input signal samples
Fs: the sampling freq (in Hz)
Win: the short-term window size (in samples)
Step: the short-term window step (in samples)
PLOT: flag, 1 if results are to be ploted
RETURNS:
"""

def stFeatureExtraction(signal, Fs, Win, Step):
"""
This function implements the shor-term windowing process. For each short-term window a set of features is extracted.
This results to a sequence of feature vectors, stored in a numpy matrix.

ARGUMENTS
signal: the input signal samples
Fs: the sampling freq (in Hz)
Win: the short-term window size (in samples)
Step: the short-term window step (in samples)
RETURNS
stFeatures: a numpy array (numOfFeatures x numOfShortTermWindows)
"""
def mtFeatureExtraction(signal, Fs, mtWin, mtStep, stWin, stStep):
"""
Mid-term feature extraction
"""

def stFeatureSpeed(signal, Fs, Win, Step):

""" Feature Extraction Wrappers

- The first two feature extraction wrappers are used to extract long-term averaged
audio features for a list of WAV files stored in a given category.

It is important to note that, one single feature is extracted per WAV file (not the whole sequence of feature vectors)

"""

def dirWavFeatureExtraction(dirName, mtWin, mtStep, stWin, stStep, computeBEAT=False):
"""

This function extracts the mid-term features of the WAVE files of a particular folder.

The resulting feature vector is extracted by long-term averaging the mid-term features.
Therefore ONE FEATURE VECTOR is extracted for each WAV file.

ARGUMENTS:
- dirName: the path of the WAVE directory
- mtWin, mtStep: mid-term window and step (in seconds)
- stWin, stStep: short-term window and step (in seconds)
"""

def dirsWavFeatureExtraction(dirNames, mtWin, mtStep, stWin, stStep, computeBEAT=False):
'''

Same as dirWavFeatureExtraction, but instead of a single dir it takes a list of paths as input and returns a list of feature matrices.
EXAMPLE:
[features, classNames] =
a.dirsWavFeatureExtraction(['audioData/classSegmentsRec/noise','audioData/classSegmentsRec/speech',
'audioData/classSegmentsRec/brush-teeth','audioData/classSegmentsRec/shower'], 1, 1, 0.02, 0.02);

It can be used during the training process of a classification model ,
in order to get feature matrices from various audio classes (each stored in a seperate path)
'''
def dirWavFeatureExtractionNoAveraging(dirName, mtWin, mtStep, stWin, stStep):
"""

This function extracts the mid-term features of the WAVE files of a particular folder without averaging each file.

ARGUMENTS:
- dirName: the path of the WAVE directory
- mtWin, mtStep: mid-term window and step (in seconds)
- stWin, stStep: short-term window and step (in seconds)
RETURNS:
- X: A feature matrix
- Y: A matrix of file labels
- filenames:
"""

The following two feature extraction wrappers extract features for given audio files, however
NO LONG-TERM AVERAGING is performed. Therefore, the output for each audio file is NOT A SINGLE FEATURE VECTOR
but a whole feature matrix.
#
Also, another difference between the following two wrappers and the previous is that they NO LONG-TERM AVERAGING IS PERFORMED.
In other words, the WAV files in these functions are not used as uniform samples that need to be averaged but as sequences

def mtFeatureExtractionToFile(fileName, midTermSize, midTermStep, shortTermSize, shortTermStep, outPutFile,

storeStFeatures=False, storeToCSV=False, PLOT=False):

""""

This function is used as a wrapper to:

a) read the content of a WAV file

b) perform mid-term feature extraction on that signal

c) write the mid-term feature sequences to a numpy file

""""

def mtFeatureExtractionToFileDir(dirName, midTermSize, midTermStep, shortTermSize, shortTermStep, storeStFeatures=False, storeToCSV=False, PLOT=False):

pyAudioAnalysis/**audioSegmentation.py**

def smoothMovingAvg(inputSignal, windowLen=11):

def selfSimilarityMatrix(featureVectors):

'''

This function computes the self-similarity matrix for a sequence of feature vectors.

ARGUMENTS:

- featureVectors: a numpy matrix (nDims x nVectors) whose i-th column corresponds to the i-th feature vector

RETURNS:

- S: the self-similarity matrix (nVectors x nVectors)

'''

def flags2segs(Flags, window):

'''

ARGUMENTS:

- Flags: a sequence of class flags (per time window)

- window: window duration (in seconds)

RETURNS:

- segs: a sequence of segment's limits: segs[i,0] is start and segs[i,1] are start and end point of segment i
- classes: a sequence of class flags: class[i] is the class ID of the i-th segment

'''

def segs2flags(segStart, segEnd, segLabel, winSize):

'''

This function converts segment endpoints and respective segment labels to fix-sized class labels.

ARGUMENTS:

- segStart: segment start points (in seconds)
- segEnd: segment endpoints (in seconds)
- segLabel: segment labels
- winSize: fix-sized window (in seconds)

RETURNS:

- flags: numpy array of class indices
- classNames: list of classnames (strings)

'''

def readSegmentGT(gtFile):

'''

This function reads a segmentation ground truth file, following a simple CSV format with the following columns:

<segment start>,<segment end>,<class label>

ARGUMENTS:

- gtFile: the path of the CSV segment file

RETURNS:

- segStart: a numpy array of segments' start positions
- segEnd: a numpy array of segments' ending positions
- segLabel: a list of respective class labels (strings)

'''

def plotSegmentationResults(flagsInd, flagsIndGT, classNames, mtStep, ONLY_EVALUATE=False):

'''

This function plots statistics on the classification-segmentation results produced either by the fix-sized supervised method or the HMM method.

It also computes the overall accuracy achieved by the respective method if ground-truth is available.

'''

def evaluateSpeakerDiarization(flags, flagsGT):

def trainHMM_computeStatistics(features, labels):
'''
This function computes the statistics used to train an HMM joint segmentation-classification model using a sequence of sequential features and respective labels

ARGUMENTS:
- features: a numpy matrix of feature vectors (numOfDimensions x numOfWindows)
- labels: a numpy array of class indices (numOfWindows x 1)
RETURNS:
- startprob: matrix of prior class probabilities (numOfClasses x 1)
- transmat: transition matrix (numOfClasses x numOfClasses)
- means: means matrix (numOfDimensions x 1)
- cov: deviation matrix (numOfDimensions x 1)
'''
def trainHMM_fromFile(wavFile, gtFile, hmmModelName, mtWin, mtStep):
'''
This function trains a HMM model for segmentation-classification using a single annotated audio file
ARGUMENTS:
- wavFile: the path of the audio filename
- gtFile: the path of the ground truth filename
(a csv file of the form <segment start in seconds>,<segment end in seconds>,<segment label> in each row
- hmmModelName: the name of the HMM model to be stored
- mtWin: mid-term window size
- mtStep: mid-term window step
RETURNS:
- hmm: an object to the resulting HMM
- classNames: a list of classNames

After training, hmm, classNames, along with the mtWin and mtStep values are stored in the hmmModelName file
'''

def trainHMM_fromDir(dirPath, hmmModelName, mtWin, mtStep):
'''
This function trains a HMM model for segmentation-classification using a where WAV files and .segment (ground-truth files) are stored
ARGUMENTS:

- dirPath: the path of the data diretory
- hmmModelName: the name of the HMM model to be stored
- mtWin: mid-term window size
- mtStep: mid-term window step

RETURNS:
- hmm: an object to the resulting HMM
- classNames: a list of classNames

After training, hmm, classNames, along with the mtWin and mtStep values are stored in the hmmModelName file
'''

def hmmSegmentation(wavFileName, hmmModelName, PLOT=False, gtFileName=""):

def mtFileClassification(inputFile, modelName, modelType, plotResults=False, gtFile=""):
'''
This function performs mid-term classification of an audio stream.
Towards this end, supervised knowledge is used, i.e. a pre-trained classifier.
ARGUMENTS:
- inputFile: path of the input WAV file
- modelName: name of the classification model
- modelType: svm or knn depending on the classifier type
- plotResults: True if results are to be plotted using matplotlib along with a set of statistics

RETURNS:
- segs: a sequence of segment's endpoints: segs[i] is the endpoint of the i-th segment (in seconds)
- classes: a sequence of class flags: class[i] is the class ID of the i-th segment

'''

def evaluateSegmentationClassificationDir(dirName, modelName, methodName):

def silenceRemoval(x, Fs, stWin, stStep, smoothWindow=0.5, Weight=0.5, plot=False):
'''
Event Detection (silence removal)
ARGUMENTS:
- x: the input audio signal
- Fs: sampling freq
- stWin, stStep: window size and step in seconds
- smoothWindow: (optinal) smooth window (in seconds)
- Weight: (optinal) weight factor (0 < Weight < 1) the higher, the more strict
- plot: (optinal) True if results are to be plotted

RETURNS:

- segmentLimits: list of segment limits in seconds (e.g [[0.1, 0.9], [1.4, 3.0]] means that the resulting segments are (0.1 - 0.9) seconds and (1.4, 3.0) seconds
'''

def speakerDiarization(fileName, numOfSpeakers, mtSize=2.0, mtStep=0.2, stWin=0.05, LDAdim=35, PLOT=False):
'''
ARGUMENTS:
- fileName: the name of the WAV file to be analyzed
- numOfSpeakers the number of speakers (clusters) in the recording (<=0 for unknown)
- mtSize (opt) mid-term window size
- mtStep (opt) mid-term window step
- stWin (opt) short-term window size
- LDAdim (opt) LDA dimension (0 for no LDA)
- PLOT (opt) 0 for not plotting the results 1 for plottingy
'''

def speakerDiarizationEvaluateScript(folderName, LDAs):
'''
This function prints the cluster purity and speaker purity for each WAV file stored in a provided directory (.SEGMENT files are needed as ground-truth)
ARGUMENTS:
- folderName: the full path of the folder where the WAV and SEGMENT (ground-truth) files are stored
- LDAs: a list of LDA dimensions (0 for no LDA)
'''

def musicThumbnailing(x, Fs, shortTermSize=1.0, shortTermStep=0.5, thumbnailSize=10.0):
'''
This function detects instances of the most representative part of a music recording, also called "music thumbnails".
A technique similar to the one proposed in [1], however a wider set of audio features is used instead of chroma features.
In particular the following steps are followed:
- Extract short-term audio features. Typical short-term window size: 1 second
- Compute the self-silimarity matrix, i.e. all pairwise similarities between feature vectors
- Apply a diagonal mask is as a moving average filter on the values of the self-similarty matrix.
The size of the mask is equal to the desirable thumbnail length.
- Find the position of the maximum value of the new (filtered) self-similarity matrix.
The audio segments that correspond to the diagonial around that position are the selected thumbnails

ARGUMENTS:
- x: input signal

- Fs: sampling frequency
- shortTermSize: window size (in seconds)
- shortTermStep: window step (in seconds)
- thumbnailSize: desider thumbnail size (in seconds)

RETURNS:
- A1: beginning of 1st thumbnail (in seconds)
- A2: ending of 1st thumbnail (in seconds)
- B1: beginning of 2nd thumbnail (in seconds)
- B2: ending of 2nd thumbnail (in seconds)

USAGE EXAMPLE:
import audioFeatureExtraction as aF
[Fs, x] = basicIO.readAudioFile(inputFile)
[A1, A2, B1, B2] = musicThumbnailing(x, Fs)

[1] Bartsch, M. A., & Wakefield, G. H. (2005). Audio thumbnailing of popular music using chroma-based representations.
Multimedia, IEEE Transactions on, 7(1), 96-104.
'''

pyAudioAnalysis/**audioTrainTest.py**

def signal_handler(signal, frame):

def classifierWrapper(classifier, classifierType, testSample):
'''
This function is used as a wrapper to pattern classification.
ARGUMENTS:
- classifier: a classifier object of type mlpy.LibSvm or kNN (defined in this library)
- classifierType: "svm" or "knn"
- testSample: a feature vector (numpy array)
RETURNS:
- R: class ID
- P: probability estimate

EXAMPLE (for some audio signal stored in array x):
import audioFeatureExtraction as aF
import audioTrainTest as aT
# load the classifier (here SVM, for kNN use loadKNNModel instead):
[Classifier, MEAN, STD, classNames, mtWin, mtStep, stWin, stStep] = aT.loadSVModel(modelName)
mid-term feature extraction:
[MidTermFeatures, _] = aF.mtFeatureExtraction(x, Fs, mtWin * Fs, mtStep * Fs, round(Fs*stWin), round(Fs*stStep));

```
# feature normalization:
curFV = (MidTermFeatures[:, i] - MEAN) / STD;
# classification
[Result, P] = classifierWrapper(Classifier, modelType, curFV)
'''
def regressionWrapper(model, modelType, testSample):
'''
This function is used as a wrapper to pattern classification.
ARGUMENTS:
- model: regression model
- modelType: "svm" or "knn" (TODO)
- testSample: a feature vector (numpy array)
RETURNS:
- R: regression result (estimated value)

EXAMPLE (for some audio signal stored in array x):
TODO
'''
def randSplitFeatures(features, partTrain):
'''
def randSplitFeatures(features):

This function splits a feature set for training and testing.

ARGUMENTS:
- features: a list ([numOfClasses x 1]) whose elements containt numpy matrices of features.
each matrix features[i] of class i is [numOfSamples x numOfDimensions]
- partTrain: percentage
RETURNS:
- featuresTrains: a list of training data for each class
- featuresTest: a list of testing data for each class
'''

def trainKNN(features, K):
'''
Train a kNN classifier.
ARGUMENTS:
- features: a list ([numOfClasses x 1]) whose elements containt numpy matrices of features.
each matrix features[i] of class i is [numOfSamples x numOfDimensions]
- K: parameter K
RETURNS:
- kNN: the trained kNN variable

'''
```

def trainSVM(features, Cparam):
'''
Train a multi-class probabilitistic SVM classifier.
Note: This function is simply a wrapper to the mlpy-LibSVM functionality for SVM training
See function trainSVM_feature() to use a wrapper on both the feature extraction and the SVM training (and parameter tuning) processes.
ARGUMENTS:
- features: a list ([numOfClasses x 1]) whose elements containt numpy matrices of features
each matrix features[i] of class i is [numOfSamples x numOfDimensions]
- Cparam: SVM parameter C (cost of constraints violation)
RETURNS:
- svm: the trained SVM variable

NOTE:
This function trains a linear-kernel SVM for a given C value. For a different kernel, other types of parameters should be provided.
For example, gamma for a polynomial, rbf or sigmoid kernel. Furthermore, Nu should be provided for a nu_SVM classifier.
See MLPY documentation for more details (http://mlpy.sourceforge.net/docs/3.4/svm.html)
'''

def trainSVMregression(Features, Y, C):

def featureAndTrain(listOfDirs, mtWin, mtStep, stWin, stStep, classifierType, modelName, computeBEAT=False, perTrain=0.90):
'''
This function is used as a wrapper to segment-based audio feature extraction and classifier training.
ARGUMENTS:
listOfDirs: list of paths of directories. Each directory contains a signle audio class whose samples are stored in seperate WAV files.
mtWin, mtStep: mid-term window length and step
stWin, stStep: short-term window and step
classifierType: "svm" or "knn"
modelName: name of the model to be saved
RETURNS:
None. Resulting classifier along with the respective model parameters are saved on files.
''

def featureAndTrainRegression(dirName, mtWin, mtStep, stWin, stStep, modelType, modelName, computeBEAT=False):
'''
This function is used as a wrapper to segment-based audio feature extraction and classifier training.

ARGUMENTS:
dirName: path of directory containing the WAV files and Regression CSVs
mtWin, mtStep: mid-term window length and step
stWin, stStep: short-term window and step
modelType: "svm" or "knn"
modelName: name of the model to be saved
RETURNS:
None. Resulting regression model along with the respective model parameters are saved on files.
'''

def loadKNNModel(kNNModelName, isRegression=False):

def loadSVModel(SVMmodelName, isRegression=False):
'''
This function loads an SVM model either for classification or training.
ARGMUMENTS:
- SVMmodelName: the path of the model to be loaded
- isRegression: a flag indigating whereas this model is regression or not
''

def evaluateClassifier(features, ClassNames, nExp, ClassifierName, Params, parameterMode, perTrain=0.90):
'''
ARGUMENTS:
features: a list ([numOfClasses x 1]) whose elements containt numpy matrices of features.
each matrix features[i] of class i is [numOfSamples x numOfDimensions]
ClassNames: list of class names (strings)
nExp: number of cross-validation experiments
ClassifierName: svm or knn
Params: list of classifier parameters (for parameter tuning during cross-validation)
parameterMode: 0: choose parameters that lead to maximum overall classification ACCURACY
1: choose parameters that lead to maximum overall F1 MEASURE
RETURNS:
bestParam: the value of the input parameter that optimizes the selected performance measure
''

def evaluateRegression(features, labels, nExp, MethodName, Params):
'''
ARGUMENTS:

features: numpy matrices of features [numOfSamples x numOfDimensions]
labels: list of sample labels
nExp: number of cross-validation experiments
MethodName: svm or knn
Params: list of classifier params to be evaluated
RETURNS:
bestParam: the value of the input parameter that optimizes the selected performance measure
'''

def printConfusionMatrix(CM, ClassNames):
'''
This function prints a confusion matrix for a particular classification task.
ARGUMENTS:
CM: a 2-D numpy array of the confusion matrix
(CM[i,j] is the number of times a sample from class i was classified in class j)
ClassNames: a list that contains the names of the classes
'''

def normalizeFeatures(features):
'''
This function normalizes a feature set to 0-mean and 1-std.
Used in most classifier trainning cases.

ARGUMENTS:
- features: list of feature matrices (each one of them is a numpy matrix)
RETURNS:
- featuresNorm: list of NORMALIZED feature matrices
- MEAN: mean vector
- STD: std vector
'''

def listOfFeatures2Matrix(features):
'''
listOfFeatures2Matrix(features)

This function takes a list of feature matrices as argument and returns a single concatenated feature matrix and the respective class labels.

ARGUMENTS:
- features: a list of feature matrices

RETURNS:
- X: a concatenated matrix of features
- Y: a vector of class indeces
''

def pcaDimRed(features, nDims):

def fileClassification(inputFile, modelName, modelType):

def fileRegression(inputFile, modelName, modelType):
# Load classifier:

def lda(data, labels, redDim):

def writeTrainDataToARFF(modelName, features, classNames, featureNames):

def trainSpeakerModelsScript():
'''
This script is used to train the speaker-related models (NOTE: data paths are hard-coded and NOT included in the library, the models are, however included)
import audioTrainTest as aT
aT.trainSpeakerModelsScript()
'''