Statlearn - homework II

# Part I - Song genre classification

## Installing and importing libraries

# this part is to be executed only once to install libraries we need   
# i kindly suggest you run this on windows OS  
# But if you feel like solving R dependencies hell on linux... give it a try .  
# about macOS , don't really know  
#   
#   
# install.packages('signal')  
# install.packages('audio')  
# install.packages('wrassp')  
# install.packages('warbleR')  
# install.packages('tuneR')  
# install.packages('audiolyzR')

# then we import all libraries needed here  
suppressMessages(require(signal, quietly = T))  
library(signal)  
  
suppressMessages(require(audio, quietly = T))   
library(audio)  
  
suppressMessages(require(wrassp, quietly = T))  
library(wrassp)  
  
  
library(warbleR)

## Loading required package: maps

## Loading required package: tuneR

##   
## Attaching package: 'tuneR'

## The following object is masked from 'package:audio':  
##   
## play

## Loading required package: seewave

##   
## Attaching package: 'seewave'

## The following object is masked from 'package:signal':  
##   
## unwrap

## Loading required package: NatureSounds

##   
## NOTE: functions are being renamed (run 'print(new\_function\_names)' to see new names). Both old and new names are available in this version   
## Please see citation('warbleR') for use in publication

library(tuneR)  
library(audiolyzR)

## Loading required package: hexbin

## Loading required package: RJSONIO

## Loading required package: plotrix

##   
## Attaching package: 'plotrix'

## The following object is masked from 'package:seewave':  
##   
## rescale

## Reading and describing Data

# My path to the data   
auPath <- "data\_example"  
labelsFile <- paste0(auPath,'/labels.txt')  
labelsFile

## [1] "data\_example/labels.txt"

# List the .au files  
auFiles <- list.files(auPath, pattern=glob2rx('\*.au'), full.names=TRUE)  
auFiles

## [1] "data\_example/f1.au" "data\_example/f2.au"

# Number of files   
N <- length(auFiles)

we have a total of {N} songs in our dataset .

## Let's try to get the files in order   
ord = c(1:N)  
ordFileList = paste0(rep(paste0(auPath,'/f')),paste0(ord,rep('.au',N)))  
ordFileList

## [1] "data\_example/f1.au" "data\_example/f2.au"

# let's get also labels for each files  
labels <- read.table(file=labelsFile, header=TRUE, sep=" ",col.names = c('id','type'))

## Warning in read.table(file = labelsFile, header = TRUE, sep = " ",  
## col.names = c("id", : incomplete final line found by readTableHeader on  
## 'data\_example/labels.txt'

## Warning in read.table(file = labelsFile, header = TRUE, sep = " ",  
## col.names = c("id", : header and 'col.names' are of different lengths

labels

## id type  
## 1 1 country  
## 2 2 country

?read.csv

## starting httpd help server ... done

# Load an audio file, e.g. the first one in the list above  
x <- read.AsspDataObj(ordFileList[1])  
str(attributes(x))

## List of 10  
## $ names : chr "audio"  
## $ trackFormats: chr "INT16"  
## $ sampleRate : num 22050  
## $ filePath : chr "data\_example/f1.au"  
## $ origFreq : num 0  
## $ startTime : num 0  
## $ startRecord : int 1  
## $ endRecord : int 666820  
## $ class : chr "AsspDataObj"  
## $ fileInfo : int [1:2] 15 2

#Then we set a fixed samples length for all files  
# as the minimum lenght of all of them   
fixedLength = 22050 \* 30 # default length  
  
  
for (i in 1:N) {  
 x <- read.AsspDataObj(auFiles[i])  
 min = attributes(x)$endRecord # the samples length of the current file  
 fixedLength <- ifelse(fixedLength<=min, fixedLength, min) # we take the minimum  
}

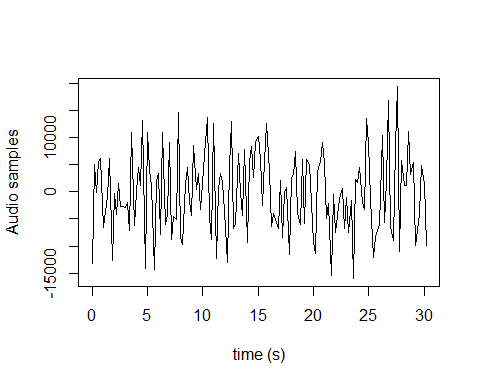
We can see from the output that The records were made at a sample rate of 22050hz for a duration of 30 seconds and therefore contains around 22050 \* 30 = 661500 samples

Now lets’ plot rthe first file samples to geta general idea

# We   
# (only plot every 10th element to accelerate plotting)  
  
x = read.AsspDataObj(ordFileList[1])  
x

## Assp Data Object of file data\_example/f1.au.  
## Format: SND (binary)  
## 666820 records at 22050 Hz  
## Duration: 30.241270 s  
## Number of tracks: 1   
## audio (1 fields)

ith = 22050 /5 # ith element to plot . basically we are plotting elements each 0.2s  
  
x\_axe = seq(0,numRecs.AsspDataObj(x) - 1, ith) / rate.AsspDataObj(x)  
  
y\_axe = x$audio[c(TRUE, rep(FALSE,ith-1))]  
  
plot(x\_axe,  
 y\_axe,  
 type='l',  
 xlab='time (s)',  
 ylab='Audio samples')



## Features Extractions

### Features to extract

#### Zero crossing rate

The zero-crossing rate is the rate of sign-changes along a signal, i.e., the rate at which the signal changes from positive to zero to negative or from negative to zero to positive.[1] This feature has been used heavily in both speech recognition and music information retrieval, being a key feature to classify percussive sounds.[2]

ZCR is defined formally as

{zcr={}*{t=1}^{T-1}* { *{<0}}(s*{t}s\_{t-1})} {zcr={}*{t=1}^{T-1}* { *{<0}}(s*{t}s\_{t-1})} where {s} s is a signal of length {T} T and { *{* {<0}}} { *{* {<0}}}

is an indicator function.

#### Spectral properties

the spectral prperties are a set of statistics computed on the spectrum of an audio signal, sucha as - Spectral Centroid ( most important one) - Spectral mean or median - spectral quartiles

The spectral centroid is a measure used in digital signal processing to characterise a spectrum. It indicates where the center of mass of the spectrum is located. Perceptually, it has a robust connection with the impression of brightness of a sound. We basically loop on on audio files, and compute some features using functions defined above .

It is calculated as the weighted mean of the frequencies present in the signal, determined using a Fourier transform, with their magnitudes as the weights

#### Spectral roll Off

The roll-off frequency is defined as the frequency under which some percentage (cutoff) of the total energy of the spectrum is contained. The roll-off frequency can be used to distinguish between harmonic (below roll-off) and noisy sounds (above roll-off).

#### Mel Frequency Cepstral Coefficients

Mel Frequency Cepstral Coefficients (MFCC) for an object of class Wave. In speech recognition MFCCs are used to extract the stimulus of the vocal tract from speech

#### Chroma frequencies

Chroma features are an interesting and powerful representation for music audio in which the entire spectrum is projected onto 12 bins representing the 12 distinct semitones (or chroma) of the musical octave. Since, in music, notes exactly one octave apart are perceived as particularly similar, knowing the distribution of chroma even without the absolute frequency (i.e. the original octave) can give useful musical information about the audio – and may even reveal perceived musical similarity that is not apparent in the original spectra.

### Implementation with R

In R we used principally package - Seawave - SoundGen - TuneR

#### Convert audio data to wave

# Transform  
  
  
# x : array to transform   
# rate : the sample rate of x   
# bit :   
# reduceRate : wether reduce the sample rate or not   
# newRate # if down Sample is TRUE, new sample rate to use   
transformToWave <- function(x, rate, bit = 16,reduceRate = FALSE, newRate = 11025 ){  
 xwv = Wave( as.numeric(x), samp.rate = rate, bit = bit)  
 if(reduceRate){  
 xwv = downsample(xwv, samp.rate = newRate)  
 }  
 #transformedWave <- ifelse(reduceRate, downsample(xwv, samp.rate = newRate),xwv)  
 return( xwv)  
}

#### Spectrum analysis , power spectrum and energy band

# Compute the powerspectrum of the input signal  
  
# x : audio samples array   
# rate : samples rate   
# The output is a matrix, where each column represents a power spectrum   
# for a given time frame and each row represents a frequency.  
powerSpectrum <- function(x , rate ){  
 out = powspec( as.numeric(x), rate)  
 return(out)  
  
}  
  
  
# Spectral info ------------------------------------------------------------  
  
# calculate the fundamental frequency contour  
# name : name of the file input  
spectralInfo <- function(name){  
 f0vals = ksvF0("data\_example/f1.au", toFile=F)  
 return(f0vals)  
}  
  
  
  
# --------------------------------------------------------------------  
  
# Get the spectogram of a wave   
# x : wave array   
# winsize : Fourier transform window size  
# fs : rate   
# overlap : overlap with previous window, defaults to half the window length.  
getSpecgram <- function (x, winsize, fs ,overlap){  
 sp <- specgram(x, n = winsize, Fs = fs, overlap = overlap)  
 return(sp)  
   
}  
  
#---------------------------------------------------------------------  
  
#Frequency spectrum of a time wave  
  
# x : an R object.  
  
# fs : sampling frequency of wave (in Hz). Does not need to be specified if embedded in wave.  
  
# wl : if at is not null, length of the window for the analysis (by default = 512).  
  
# wn : window name, see ftwindow (by default "hanning").  
  
# fftw : if TRUE calls the function FFT of the library fftw for faster computation. See Notes of the function spectro.  
  
# norm #if TRUE the spectrum is normalised by its maximum.  
  
getSpec <- function (x, winsize, fs ){  
 sp <- spec(x, fs = fs, wl = winsize, fftw = TRUE,norm =TRUE)  
 return(sp)  
}  
  
#-----------------------------------------------------------------------------------  
# To get spectral properties   
  
# spec : a data set resulting of a spectral analysis obtained with spec or meanspec (not in dB).  
  
# f :sampling frequency of spec (in Hz).  
  
# str :logical, if TRUE returns the results in a structured table.  
  
# flim :a vector of length 2 to specifgy the frequency limits of the analysis (in kHz)  
  
# mel #a logical, if TRUE the (htk-)mel scale is used.  
  
GetSpecProps <- function(x, fs){  
 specProps = specprop(x, f= fs, mel = TRUE)  
 return(specProps)  
}  
  
#---------------------------------------------------------------  
  
# compute the zero crossing rate  
  
# x : R wave object   
  
  
#f :sampling frequency of wave (in Hz). Does not need to be specified if embedded in wave.  
  
#wl: length of the window for the analysis (even number of points, by default = 512). If NULL the zero-crossing rate is computed of the complete signal.  
  
#overlap : overlap between two successive analysis windows (in %) if wl is not NULL.  
 zeroCrossingRate <- function (x , fs, wl , overlap ){  
   
 cr = zcr(x,f= fs, wl = wl, ovlp = overlap)  
 return(zcr)  
   
 }  
  
# Computation of MFCCs (Mel Frequency Cepstral Coefficients) for a Wave object  
  
# x : Object of class Wave.  
  
getMfccs <- function(x){  
 mfccs = MFCC(x, a = 0.1, HW.width = 0.025, HW.overlapping = 0.25,   
 T.number = 24, T.overlapping = 0.5, K = 12)  
   
 return(mfcccs)  
}  
  
  
# Energy bands ----------------------------------------------------------------  
  
# x : audio spectogram  
# winsize : Fourier transform window size  
# fs : rate   
# nb : number of bands to select  
# lowB :  
# eps : default minimum energy value   
  
  
  
energyBands <- function(x,fs,nb, lowB,eps,winsize){  
ntm <- ncol(x$S) # number of (overlapping) time segments  
fco <- round( c(0, lowB\*(fs/2/lowB)^((0:(nb-1))/(nb-1)))/fs\*winsize )  
energy <- matrix(0, nb, ntm)  
for (tm in 1:ntm){  
 for (i in 1:nb){  
 lower\_bound <- 1 + fco[i]  
 upper\_bound <- min( c( 1 + fco[i + 1], nrow(x$S) ) )  
 energy[i, tm] <- sum( abs(x$S[ lower\_bound:upper\_bound, tm ])^2 )  
 }  
}  
energy[energy < eps] <- eps  
energy = 10\*log10(energy)  
  
return(energy)  
   
   
}

### Dataset Creation

Basically we loop over the audio file, extracting all features and saving in in text file

# first we define the general parameters  
  
rate = 22050  
newrate = 11050  
reduceRate = FALSE  
  
  
# STFT  
winsize <- 2048  
nfft <- 2048  
hopsize <- 512  
overlap <- winsize - hopsize  
  
# Frequency bands selection  
nb <- 2^3  
lowB <- 100  
eps <- .Machine$double.eps  
# Number of seconds of the analyzed window  
corrtime <- 15  
  
# the file list to use is our Ordered file list   
  
data = list()  
  
for( file in ordFileList){  
   
 # we only take the fixed length sample to make equal for all files   
 x = read.AsspDataObj(file)$audio[1:fixedLength]  
 xWave = transformToWave(x,rate)  
 xPoweSpec = powerSpectrum(x,rate)  
 xSpecgram = getSpecgram(x,fs = rate,winsize = winsize,overlap = overlap)  
 xSpectralInfo = spectralInfo(file)  
   
   
 # we save all the values   
   
 #xData = data.frame(xWave,xPoweSpec,xSpecgram,xSpectralInfo)  
 #data <- c(data,xData)  
   
}

## Models to apply

### List of models to use and reasons (limit to models seen in class)

### Models Implementation

## classification

### Classification performance for each model

### Chose the best model

### Determine each feature contribution to model

### Maintain only the most important ones

### Final classification

## Map

# Part II - Theory