AviPulse

Project Report

Autumn 2014

Web Bird Detector

Kumar Ayush

October 21, 2014

Contents

1 Task

The task of this project was to create a working web-based tool which can input a bird call and implement the developed sound processing algorithm to identify the bird species with probability.

2 Methodology and Plan

The original code that trains a Naive Bayes learner and calculates bird probability for an unknown sample after deriving parameters from the sound based using Short Time Fourier Transform was implemented in MATLAB. To develop a web application, we would need to invoke MATLAB CGI which is not known for being efficient or easy to implement. Rather, we could try to convert the MATLAB code to Python.

2.1 Pros and Cons

- The conversion requires extra effort. This is justifiable given better compatibility of Python with web technologies and more developer support for libraries, especially in the field of computational sciences.
- Python is infamous for its poor ability to handle floating points. This difficulty is tackled by using numpy, a mathematical package for Python, and the fact that we maintain variables upto 4 variables only in the original algorithm keeps the accuracy up to the mark.
- Python is a lot slower than MATLAB for numerical computations.
- The whole application will have heavy library dependencies. If we use a compiled Python script, this is not really a problem.

2.2 Keynotes

- Use Gaussian Naive Bayes class in Scilabs package to create machine learner
- Use scikits.audiolabs and numpy to process audio file for feature extraction
- Use xlrd to read excel files

3 The GNB Learner

3.1 Code

This section describes the code of Gaussian Naive Bayes learner and its usage. The code is supposed to be self-explanatory.

```
#!usr/bin/env/python

#Project: AviPulse
#Author: Riddhish et. al. (matlab)
#Translation: Kumar Ayush (python)
#Methodology: You need an excel file with all input parameters. The pr

#to read excel files
import xlrd

#numpy
import numpy as np

#Naive Bayes Learner Class
from sklearn.naive_bayes import GaussianNB as GNB

#to save model state into a file
from sklearn.externals import joblib

#open workbook and load a specific sheet by name
workbook = xlrd.open_workbook('data2.xlsx') #open excel file
datasheet = workbook.sheet_by_name('data without outliers manual')

tmp2 = []
tmp3 = []

#load values in arrays
for i in range(1,datasheet.nrows):
        tmp2.append(datasheet.cell_value(i,6))
        tmp3.append([datasheet.cell_value(i,j) for j in range(2,6)])

#numpify arrays, prepare for processing
data = np.array(tmp3)
target = np.array(tmp2)

```

```
#the test data
testsheet = workbook.sheet_by_name('test data')
tmp = []
for i in range(1,testsheet.nrows):
        tmp.append([testsheet.cell_value(i,j) for j in range(2,6)])

testdata = np.array(tmp)

#create a learner class
gnb = GNB()

#fit it
y_fit = gnb.fit(data,target)

#get prediction for testdata
y_pred = y_fit.predict(testdata)

#get model score
score = y_fit.score(data,target)

#Print output
print "Prediction is Bird Number %d and score is %f" %(y_pred[0],score

#save model state onto file
joblib.dump(y_fit,'model.pkl')
```

3.2 Usage

data2.xlsx is attached with this document. You can witness the format in which data is to be fed. The model is trained and the state of the model is stored in a pickle file called *model.pkl*. Generally, you would no longer need the database, but the model state only relates bird-ids and not their names. So, you would need *data2.xlsx* for bird identification to produce names from matched bird id. To train the model, just run the file on using

```
python preproc.py
```

IMP : Make sure that *data2.xlsx* is in the same folder as the python script

4 The Identifier

4.1 Code

This code is more complicated owing to the mathematical algorithms it uses. It takes multipart form data, extracts features from it, predicts a bird label and outputs the bird name. As in the previous case, the code is supposed to be self-explanatory, except during the STFT computation where textual knowledge of signal processing algorithms is expected.

```
#!/usr/bin/env python

#Project: AviPulse
#Author: Riddhish et. al. (matlab)
#Translation: Kumar Ayush (python)
#Methodology: You need a web form to post a syllable file in .wav form

#import CGI libraries
import cgi
import cgitb

#numpy
import numpy as np

#to read a wave
from scikits.audiolab import wavread

#Python math library
import math

#To read an excel file
import xlrd

#The Gaussian Naive Bayes Machine Learner Class
from sklearn.naive_bayes import GaussianNB as GNB

#To load the saved model state
from sklearn.externals import joblib

#Enable errors
#use log directory for logdir, set display=0 if don't want error displa
cgitb.enable(display=1, logdir="/home/cheeku/log")
```

```
#Define the cgi form object
form = cgi.FieldStorage()

#HTML headers
print "Content-type: text/html\n\n"

#Write the multipart sound data to a temporary file
tmpf = open("tmp.wav","wb")
tmpf.write(form['wav'].value)
tmpf.close()

#Variable initializations
scmed = 0        #Spectral Centroid Median
scmean = 0       #Spectral Centroid Mean
scmax = 0        #Spectral Centroid Max
scmin = 0        #Spectral Centroid Min
sfluxmed = 0     #Spectral Flux Median
sfluxmean = 0    #Spectral Flux Mean
intmode = 0
pitmean = 0      #Pitch Mean
pitmed = 0       #Pitch Median
flatmean = 0     #Spectral Flatness Mean
flatmed = 0      #Spectral Flatness Median

#Read the sound, stored in the temporary file
#'s' is data,'enc' is encoding
s,fs,enc = wavread("tmp.wav")
s = np.array(s)
s = s/max(abs(s))        #normalization, to imitate matlab's wavread

Frame_size=20.0          #Parameter of computation
Frame_shift = 10.0       #Parameter of computation

#Variable recasts and normalizers, prep for computation
y = s
reconsine = y
Frame_size = Frame_size/1000.0
Frame_shift = Frame_shift/1000.0

#Empty lists
```

```
secondpeak = []
scarr = []       #Spectral Centroid Array
spcroll = []
spflux = []      #for Spectral Flux calculaion
spcrest = []
spflatness = [] #for Spectral Flatness calculation
spspread = []
amplitude = []  #for Amplitude calculation
pitch = []
pitch1 = []      #for Pitch calculations
dip_bin = []
dip_amp = []
max_amp_bin = []
max_amp = []
peaks = []
peakpeak = []
max2_amp = []
max2_bin = []
atimbre = []

#Re-normalization
max_value = max(abs(y))
y = y/max_value

#Scale Frame size and length
Frame_length = Frame_size*fs
sample_shift = Frame_shift*fs

w = np.hamming(Frame_length)    #create a hamming window of given leng

dftylast = 0    #empty variable to store dft result for previous compu
for i in range(int(math.floor(len(y)/sample_shift)-math.ceil(Frame_len
        #Fourier Transform with data scaled using a sliding hamming wi
        k,jj = 0,0
        yy = []
        yyy = []
        for j in range(int(i*sample_shift),int(i*sample_shift+Frame_le
                yy.insert(k,y[j]*w[jj])
                yyy.insert(k,y[j])
                jj,k = jj+1,k+1
        dfty = abs(np.fft.fft(yy))
```

```
        yy = np.array(yy)
        dftyp = []
        for it in range(len(yy)):
                dftyp.append(math.atan2(yy[it].imag,yy[it].real))

        #computation, computation, computation
        scn,scd,add,sf,ismax = 0,0,0,0,0
        q,sctimbre,geo,jj = 0,0,1,0
        M = len(dfty)/2
        for p in range(M):
                scn = scn + (p+1)*dfty[p]*dfty[p]
                scd = scd + dfty[p]*dfty[p]
                add = add + dfty[p]
                geo = geo*dfty[p]
                if dfty[p]>ismax:
                        ismax = dfty[p]
                else:
                        ismax = ismax
                if p>0 and p<M-1:
                        if dfty[p]>dfty[p-1] and dfty[p]>dfty[p+1]:
                                peaks.insert(q,[])
                                peaks[q].insert(1,p)
                                peaks[q].insert(0,dfty[p])
                                peaks[q].insert(2,dftyp[p])
                                sctimbre = sctimbre + dftyp[p]
                                q = q+1
                if i>0:
                        sf = sf + (dfty[p]-dftylast[p])*(dfty[p]-dftyl
                else:
                        sf = 0

        #convolutions and computations
        s = 0
        length = len(yyy)
        acf_clip = np.correlate(yyy[:len(yyy)/2],yyy[:len(yyy)/2],"ful
        lenacf_clip = len(acf_clip)
        max_acfclip,max_acfclip_bin = max(acf_clip),np.argmax(acf_clip
        for acfclip_bin in range(max_acfclip_bin,lenacf_clip-1):
                if acf_clip[acfclip_bin-1]>=acf_clip[acfclip_bin] and
                        dip_bin = np.append(dip_bin,acfclip_bin)
                        dip_amp = np.append(dip_amp,acf_clip[acfclip_b
```

```
        dip_bin = np.array(dip_bin)[np.newaxis].T          #row to column
        dip_amp = np.array(dip_amp)[np.newaxis].T          #row to column
        dipMin_amp,Min_bin = min(dip_amp),np.argmin(dip_amp)
        dipMin_bin = dip_bin[Min_bin]
        max2_amp.insert(i,max(acf_clip[int(dipMin_bin):lenacf_clip]))
        max2_bin.insert(i,np.argmax(acf_clip[int(dipMin_bin):lenacf_cl
        max2_bin[i] = max2_bin[i] + dipMin_bin - 1 - max_acfclip_bin
        if i == 0:
                max2_bin[i] = max2_bin[i]
        else:
                max2_bin[i] = (max2_bin[i-1]+max2_bin[i])/2.0
        pitch1 = np.append(pitch1,fs*(1.0/(max2_bin[i]+1)))

        for r in range(1,q-1):
                if peaks[r][0]>peaks[r-1][0] and peaks[r][0]>peaks[r+
                        peakpeak.insert(s,[])
                        peakpeak[s].insert(1,peaks[r][1])
                        peakpeak[s].insert(0,peaks[r][0])
                        peakpeak[s].insert(2,peaks[r][2])
                        s = s+1

        np.sort(peakpeak,axis=0)          #sort ascending
        peakpeaksorted = np.flipud(peakpeak)     #reverse, effectively
        np.sort(peaks,axis=0)
        peakssorted = np.flipud(peaks)
        energy = float(add)/M
        sc = float(scn)/scd
        ssn = 0
        sctimbre = sctimbre - q*peaks[1][1]
        for p in range(M):
                ssn = ssn + (p+1-sc)*(p+1-sc)*dfty[p]
        ss = float(ssn)/scd
        ss = ss**0.5
        geo = geo**(1.0/M)

        sfl = float(geo)/energy
        spc = float(ismax)/energy
        add = 0.85*add
        scc = 0
        for p in range(M):
```

```
                    scc = scc + dfty[p]
                    if scc>=add:
                            break

        dftylast = dfty #current dfty to dftylast assignment

        #populate result arrays
        amplitude.insert(i,20*math.log10(add/0.85))
        pitch.insert(i,peakpeaksorted[0][1]/Frame_size)
        secondpeak.insert(i,peakpeaksorted[1][1]/Frame_size)
        spcroll.insert(i,scc)
        scarr.insert(i,sc)
        spflux.insert(i,sf)
        spcrest.insert(i,spc)
        spflatness.insert(i,sfl)
        spspread.insert(i,ss)
        atimbre.insert(i,sctimbre)

#Since we have no good way of seperating syllables in a sound file
#the following code gets the length of the file in seconds
#assuming it is a single syllable
import wave
import contextlib
fname = 'tmp.wav'
with contextlib.closing(wave.open(fname,'r')) as f:
        frames = f.getnframes()
        rate = f.getframerate()
        duration = frames / float(rate)

tmp = [duration,np.median(pitch1),np.median(scarr),np.median(spflux)]
testdata = np.array(tmp)        #prepare testdata
y_fit = joblib.load('model.pkl')        #load model from file
y_pred = y_fit.predict(testdata)        #get predicted label

#Print predicted label
print "Prediction: "
#print y_pred[0]

#open workbook and load relevant sheet
workbook = xlrd.open_workbook('data2.xlsx')
datasheet = workbook.sheet_by_name('data without outliers manual')
```

```
#print name of bird based on id
i = 0
while datasheet.cell_value(i,6) != int(y_pred[0]):
        i = i+1

print datasheet.cell_value(i,0)

#close the output
print "</html>"
```

4.2 Usage

Create a form that POSTS multipart form data in wave file format to the above python script. Make sure *data2.xlsx* is in the same folder as this file.

5 Dependencies

- **xlrd** - https://pypi.python.org/pypi/xlrd
- **scikits** - http://scikit-learn.org/stable/
- **numpy** - http://www.numpy.org/
- **scilabs** - http://www.scipy.org/topical-software.html

6 Limitations

The application currently supports single syllable sound files in .wav format.This is mainly because of failure in search for an algorithm that separates syllables efficiently and easily.

7 Learnings

- MATLAB indices start from 1, and Python indices start from 0, much like most of scripts and languages. This was responsible for most of the bugs in the program
- Gained a better understanding of Fourier Transforms and correlation algorithms.

- Naive Bayes is a simple algorithm. I was not initially convinced of its performance. The algorithm you use depends on the situation, and not on the complexity.
- Learnt basics of MATLAB. Being based out of an F77 compiler, it's not really friendly to the thought processes of a regular modern programmer, but it is useful for scientific purposes. It really excels most of other scripts when numerous numerical computations are a necessity.

8 Extending Remarks

Due to insufficient server permissions, it was not possible to install dependencies on the server. This problem has a possible solution. The python code can be compiled into an .exe file, which runs without any dependencies. This will be explored soon.