Artificial Intelligence 4 Assessed Exercise

Naive Bayes Classifier Report

Task

The task at hand was to create an intelligent agent that could analyse a series of audio files, and from this analyse then classify further audio files into 2 categories either speech or silence. This was to be achieved by monitoring 3 attributes of waves that we had to extract from the raw data of the audio files. These 3 attributes were Energy, Magnitude and Zero-Crossing Rate, which will be referred to throughout the report and code as EMZ.

Design

Given the nature of the way the work was assigned i.e. via a series of labs with objectives to fulfil my design was largely pushed by this form of assignment. Firstly I created a program to analyse a single file and extract EMZ for all points along the audio wave, then plot the features. This included mathematical functions to perform a convolution of a wave with a rectangular window, as part of the extraction of EMZ.

This was the first half of the assessment given and was therefore a stand alone program that could be expanded on to achieve a greater task. The next step was to loop through a series of 100 files and extract the EMZ properties for each file. Given that we only needed these properties to assess the entire file as opposed to any section of the file it was possible to condense the relevant information by taking the mean average of the EMZ values for each file. This could then be plotted, but unlike before the instruction was to plot 3 graphs as EvsM, EvsZ and MvsZ.

Again this program worked as a stand alone system that could be further expanded upon by adding he classifier and therefore seemed like a logical stage to create the system in. This was done according to the specification using a K-Fold cross validation Naive Bayes Classifier where K = 10. Part of this done in general times by using some of the files to train the system what the different classifications look like and then the rest of the files are used to test the systems accuracy this is then done on a repeating cycle until all the files have been used to test the system.

Performance Measure

In terms of classifier the performance measure would be most strongly represented by the number of successful classifications. This calculation can be done by dividing the total number of successful classifications by the total number of tests performed.

Environment

The environment in this scenario is simply the collection of sound files available to the system.

Actuator

In terms of actuation for this system it would have to be the printing out of success or failure of the tests, potentially also the plotting of the graphs, but this is could be considered not actuation as it is not a response from the intelligent agent merely a feeding back of information in graphical form.

Sensors

In this situation the sensor could be considered to the be considered to be the function pulling information from the files. So the reading of data from the environment, which would be the usual way in which sensors are defined.

Theory

Artificial Intelligence 4 Assessed Exercise

Completing this project there were 2 main concepts that I had previously not been familiar with which were signal processing elements to obtain these specific qualities of a wave and the classifier.

Signal Analysis

Having studied communication systems for electronics I was familiar with the concept of convolution in use of wave analysis, but had never truly understood exactly what information was being gained by this process.

The convolution of a wave with a window or rectangle function provides for each sample of the wave a view of the closely surrounding data i.e. a short term analysis. This is because a window function unless otherwise specified is 0 for all points except for a series of sequential points of a certain size; for this exercise we used a window of 30ms which was equal to 240samples. This window function can the sample-wise multiplied by the wave which gives values for everything inside the window and 0 for everything out of the window. These values are then summed to create a value that represents all the data in a 30ms window in a single point. This window moves the entire length of the array giving this representation for each point in the array.

To calculation energy here is a simple calculation once a convolution function has been established, that in maths notation is:

$$E[n] = \sum_{k=-\infty}^{\infty} s^2[k] \cdot w[n-k]$$

Where “E” is energy, “n” is the current index, “s” is in this case the audio wave and “w” is the window function.

The equation for magnitude is very similar except instead of squaring “s” it is simply made absolute i.e. all values the positive representation of their value, hence magnitude.

$$M[n] = \sum_{k=-\infty}^{\infty} |s[k]| \cdot w[n-k]$$

Where “M” is magnitude, “n” is the current index, “s” is in this case the audio wave and “w” is the window function.

Non-Zero Crossing Rate is a slightly different equation is the measure of how often the wave changes sign from positive to negative and vice versa hence crossing zero. When convolved with a window function this shows how often the signal changes sign within this window and the divided by the number of samples in window thereby forming a mean average.

$$Z[n] = \frac{1}{2N} \sum_{k=-\infty}^{\infty} |sign(s[k]) - sign(s[k-1])| \cdot w[n-k]$$

Where “Z” is the zero-crossing rate, “n” is the index, “N” is the number of samples in the window

(in our case 480), “s” is the wave (in this case the audio file) and “w” is the window.

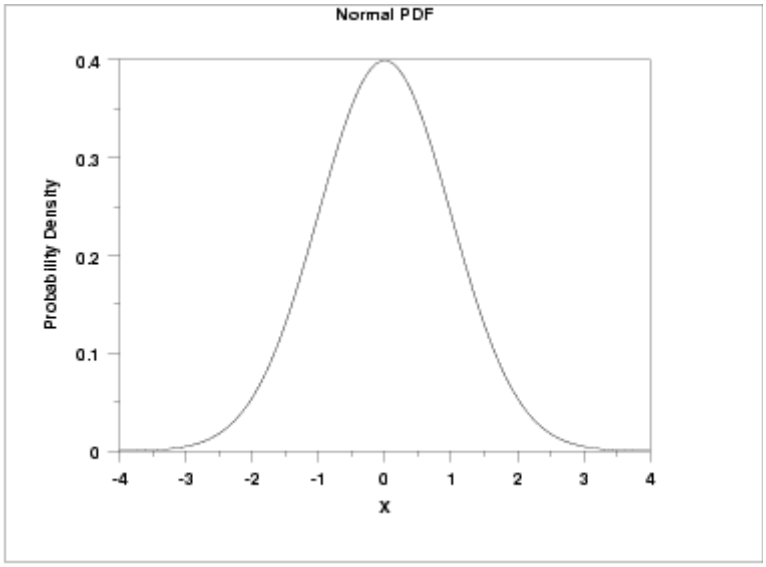
Classification

A classifier is a system that can place items in a classification via analysis of the attributes of said items. Obviously for a human this is a very simple process the notice the difference between silence(of course not truly silent so still has a wave form readable by a computer) and speech, but because the system merely sees a wave for each it needs to use statistical analysis to determine which classification an item falls under.

This Classification is done by giving the system a view of what an average sample of each classification will look like. In our example this was done by feeding it a series of samples that it knows the classification of already and it then uses this information to form a set of averages and compare that to a test sample that it is attempting to classify.

Given that we are dealing with continuous values in this scenario, each new value has an infinitely small chance of matching a previous value observed (though on a computer this is not strictly true because there are limited values available, but still a very small chance).

This means that we need to form a probability density of our data, it generates a kind of field of probability that gives a likely hood of any attribute belonging to a class.



This graph represents a normal distribution which is used to provide this, so in reference to the graph if say an element has attribute 'x' and you want to see how likely it is that this element belongs to the classification that has generated this curve you follow 'x' across the graph and observe its probability. In this instance if x is 0 this would increase the probability density that it belongs to this classification whereas as 'x' moves towards -4 or +4 probability dramatically decreases.

This graph is generated by looking at the variance and mean of an array, so in our scenario for each attribute EMZ for each classification a mathematical representation of this graph is created. The variance represents how much values stray from the mean value which is represented on the graph by the steepness of the slope and the mean which in terms of the graph represents the most likely position so in the situation on the graph the mean would be 0.

So in our example we have a selection of training data that we are using to teach the system what

each of the classifications look like and from this training data the mean and variance is calculated for each attribute. Any test file then checks each of its own attributes against this mathematical model to determine its probability density for each classification.

$$P(x = v|c) = \frac{1}{\sqrt{2\pi\sigma_c^2}} e^{-\frac{(v-\mu_c)^2}{2\sigma_c^2}}$$

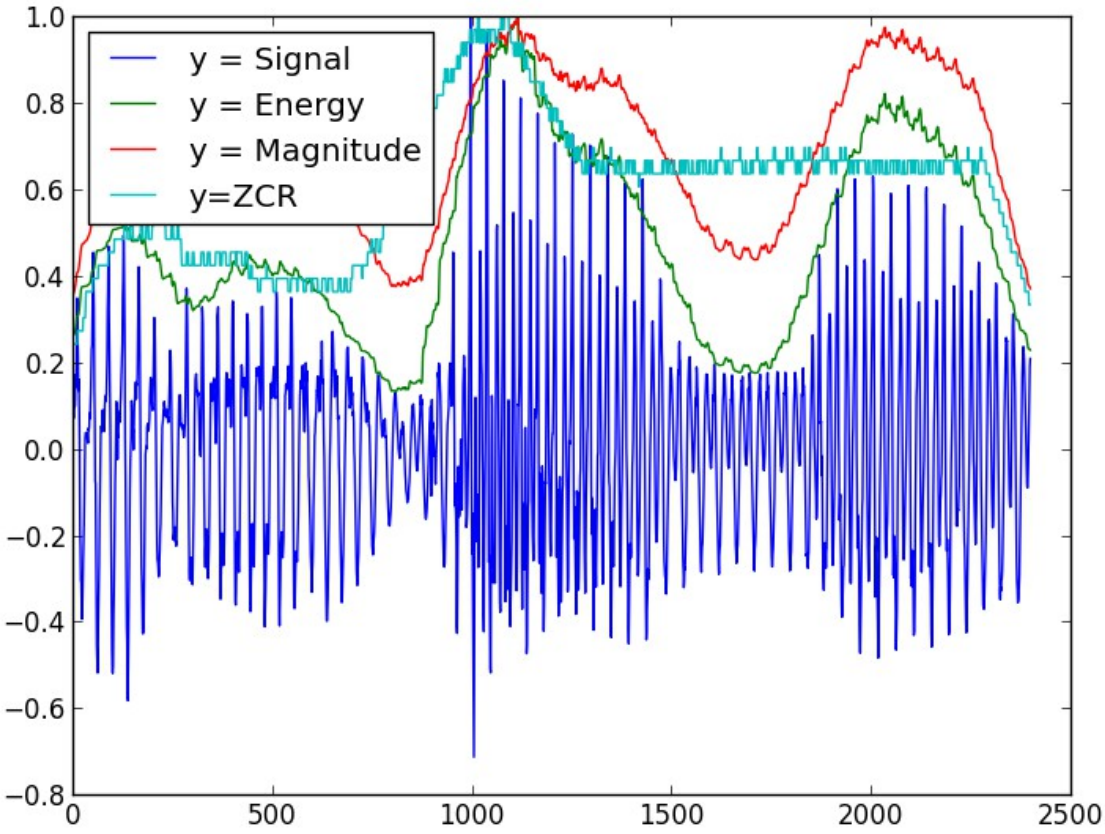
Where x is the continuous attribute, μ_c is the mean of the values in x associated with class *c* and σ_c^2 is the variance of the values in x associated with class *c*.

For testing purposes we used a K-Fold cross-validation procedure this is a procedure that divides the entire set of files into K smaller sets, K-1 of which are used for training the system and 1 of which is used for testing the system this is then done in a loop so that all sets are each used once for testing. The assignment of a sample to a set is done randomly bar the fact that there is even distribution of speech and silence so each set has fileNumer/2K random speech files and the same number of silence files, in this case that is 100 files/ 20 as 10 was chosen as K.

Experiments

As described in the design this exercise was completed in 3 stages, single file, multiple files and the classifier therefore experiments were done in this nature. I wrote the entirety of my code in python, it is very easy to read, write and grow upon. All of my plots are created at runtime in a python library called matplotlib which tends to follow the the MATLAB style plotting and syntax.

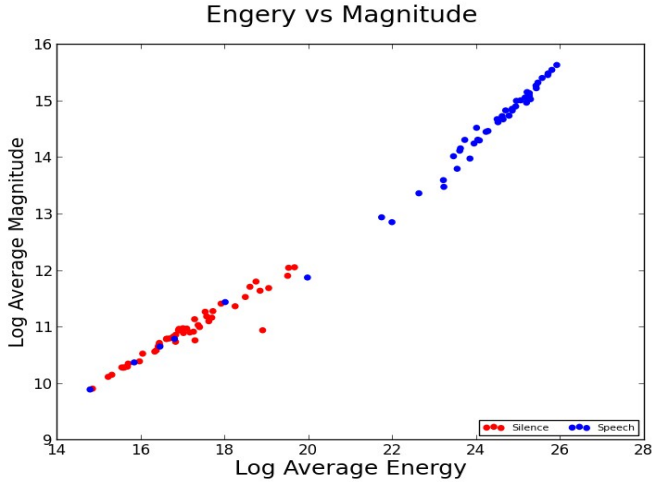
The single file analysis was done by reading into an array a series of integers from a ".dat" file. Performing all the necessary functions on the data to extract the desired attributes and then plotting each point of each attribute. There should be 2400 points of each of the EMZ arrays and can therefore be plotted on he same graph (though normalisation is required so that all of the values are visible).



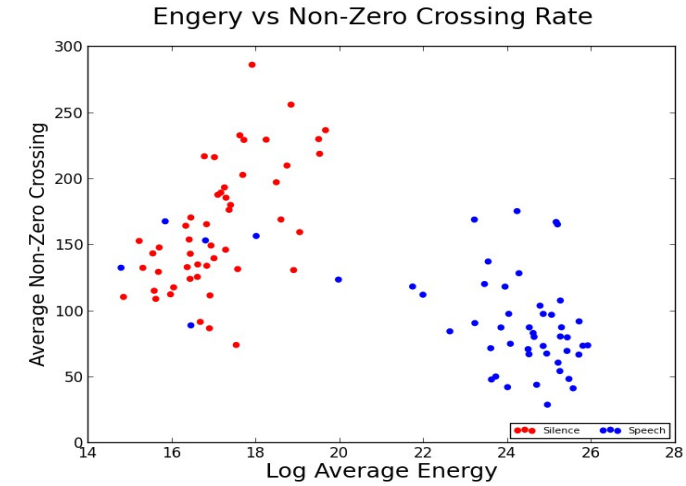
For the array of files, instead of storing an array of EMZ as we did with a single file we stored a mean average of each array. In my implementation I stored this as a an array of triples, a triple for each file being read in.

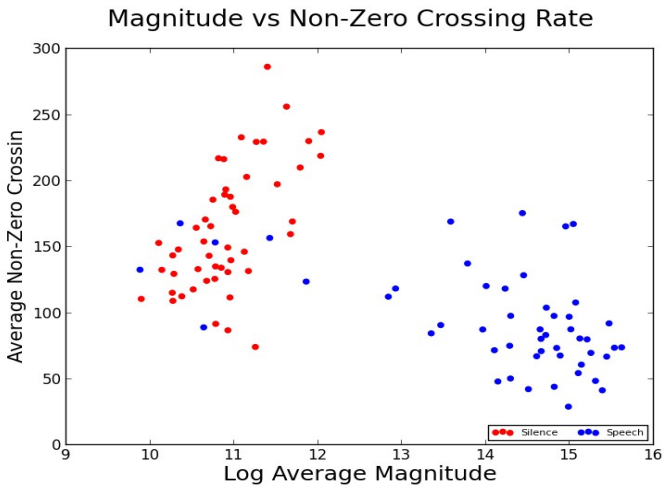
This was simply calling my previous single file implementation in 2 loops (one for silence, one for speech), then averaging out the EMZs (also taking logs of E and M to scale them down).

For this experiment instead of plotting a plotting the attributes through time as we had before (because its impossible after the averaging), we plotted EvsM, EvsZ and ZvsM as can be seen in the diagram below



Artificial Intelligence 4 Assessed Exercise





As can be see in the images there are 6 speech files that look very similar to silence files via human visual inspection, this is probably where the classifier will fail to classify some of the samples.

Classifier

Given that there is no real output from the classifier bar the success and failure rates, there are little results. After running the program 10 times my results were an extremely consistent 95% with 0 runs resulting in a different outcome.

Looking at the diagrams it might look like there would be 6 failures given how similar those 6 files look to silence files on the graphs, but as it is very likely that when testing the one of those 6 closest to the speech files the other 5 have been used to train the system. With the system knowing that 5 speech files have values that are very far away from the mean of the speech files attributes, it can determine that an outlier is speech rather than silence because silence does not have any samples that have their attributes so far removed from the average. This is where the variance in the probability density equation is useful.

Appendix

Code

AI4 – python main – all code is in orange

```
import mathsFunctions as mf
import matplotlib.pyplot as plt
import math as math
import random as random

def readFile(fileName):
    f = open(fileName,'r')
    array = f.readlines()
    array = [int(numStr) for numStr in array]
    f.close()
    return array

def values(name, i):
    number = str(i)
    if (i < 10):
        number = "0" + str(i)
    Rect = []
    for i in range (0,240):
        Rect.append(1)
    DataIn = readFile("./silence_speech/" + name + "_" + number + ".dat")

    sampleRate = len(DataIn)/0.3
    Datasign = []

    for j in range (1,len(DataIn)):
        Datasign.append(mf.sign(DataIn[j]) - mf.sign(DataIn[j-1]))

    Datafft = mf.square(DataIn)
    Datapos = mf.absolute(DataIn)
    Datasign = mf.absolute(Datasign)

    Energy = mf.convolve(Datafft, Rect)
    Magn = mf.convolve(Datapos, Rect)
    ZCR = mf.convolve(Datasign, Rect)
```

```
    E = mf.logBase(mf.mean(Energy), math.exp(1))
    M = mf.logBase(mf.mean(Magn), math.exp(1))
    Z = mf.mean(ZCR)
    return E, M, Z

def setEMZ(array):
    E = []
    M = []
    Z = []
    for i in range (0,len(array)):
        E.append(array[i][0])
        M.append(array[i][1])
        Z.append(array[i][2])
    return E, M, Z

def postSil(sample):
    global trainESil, trainMSil, trainZSil
    return 0.5 * mf.probDens(trainESil,sample[0]) * mf.probDens(trainMSil, sample[1]) *
mf.probDens(trainZSil, sample[2])

def postSpe(sample):
    global trainESpe, trainMSpe, trainZSpe
    return  0.5 * mf.probDens(trainESpe,sample[0]) * mf.probDens(trainMSpe, sample[1]) *
mf.probDens(trainZSpe, sample[2])

def genKSet(Sil, Spe):
    KSets = []
    random.shuffle(Sil)
    random.shuffle(Spe)
    for i in range (0,10):
        KSets.append([Sil[i*5:(i+1)*5], Spe[i*5:(i+1)*5]])
    return KSets

def reform(Kset):
    Sil = []
```

```
    Spe = []
    for i in range (0, len(Kset)):
        Sil.append(Kset[i][0])
        Spe.append(Kset[i][1])
    Sil = dechunk(Sil)
    Spe = dechunk(Spe)
    return Sil, Spe

def dechunk(array):
    dechunked = []
    for i in range (0,len(array)):
      for j in range (0, len(array[i])):
          dechunked.append(array[i][j])
    return dechunked

def testSample(sample):
    result = postSil(sample) < postSpe(sample)
    return result



def writeFile(fileName, array1, array2):
    array = []
    array3 = array1.ca
    f = open(fileName, 'w')
    f.write(array1)

    f.close()
    return array

Sil = []
Spe = []

for i in range (1,51):
    Sil.append(values("silence", i))
for i in range (1,51):
```

```
    Spe.append(values("speech", i))


KSet = genKSet(Sil, Spe)
correctCount = 0
errorCount = 0
for i in range(0,10):
    trainingSet = []
    for j in range (0,10):
        if (j != i):
            trainingSet.append(KSet[j])
    testSet = [KSet[i]]
    trainSil, trainSpe = reform(trainingSet)

    trainESil, trainMSil, trainZSil = setEMZ(trainSil)
    trainESpe, trainMSpe, trainZSpe = setEMZ(trainSpe)

    testSil, testSpe = reform(testSet)

    for k in range (0, len(testSil)):
        if not (testSample(testSil[k])):
            correctCount += 1
        else:
            errorCount += 1
    for k in range (0, len(testSpe)):
        if (testSample(testSpe[k])):
            correctCount += 1
        else:
            errorCount += 1


ESil, MSil, ZSil = setEMZ(Sil)
ESpe, MSpe, ZSpe = setEMZ(Spe)

print correctCount
```

```
print errorCount



fig1 = plt.figure(1)
EM1 = plt.scatter(ESil, MSil, color = 'red')
EM2 = plt.scatter(ESpe, MSpe, color = 'blue')
fig1.suptitle("Engery vs Magnitude", fontsize=20)
plt.xlabel('Log Average Energy', fontsize=18)
plt.ylabel('Log Average Magnitude', fontsize=16)
plt.legend(( EM1, EM2), ("Silence", "Speech"),loc='lower right', ncol=3, fontsize=8)
fig2 = plt.figure(2)
EZ1 = plt.scatter(ESil, ZSil, color = 'red')
EZ2 = plt.scatter(ESpe, ZSpe, color = 'blue')
fig2.suptitle("Engery vs Non-Zero Crossing Rate", fontsize=20)
plt.xlabel('Log Average Energy', fontsize=18)
plt.ylabel('Average Non-Zero Crossing', fontsize=16)
plt.legend((EZ1, EZ2), ("Silence", "Speech"),loc='lower right', ncol=3, fontsize=8)
fig3 = plt.figure(3)
MZ1 = plt.scatter(MSil, ZSil, color = 'red')
MZ2 = plt.scatter(MSpe, ZSpe, color = 'blue')
fig3.suptitle("Magnitude vs Non-Zero Crossing Rate", fontsize=20)
plt.xlabel('Log Average Magnitude', fontsize=18)
plt.ylabel('Average Non-Zero Crossin', fontsize=16)
plt.legend((MZ1, MZ2),( "Silence", "Speech"), loc='lower right', ncol=3, fontsize=8)
plt.show()
```

AI4 – mathsFunctions – All code in blue

```
from math import *

def convolve(array1, window):
    array2 = []

    for i in range (0, len(array1)):
        low = max(i-len(window)/2, 0)
```

```
        high = min(i+len(window)/2, len(array1))
        total = sum(array1[low:high])
        array2.append(total)
    return array2

def divide(array1, x):
    array2 = [(num/x) for num in array1]
    return array2

def multiply(array1, x):
    array2 = [(num*x) for num in array1]
    return array2

def absolute(array1):
    array2 = []
    for i in range(0, len(array1)):
        if (array1[i] >= 0):
            array2.append(array1[i])
        else:
            array2.append(array1[i] * -1)
    return array2

def square(array1):
    array2 = [sqr**2 for sqr in array1]
    return array2

def mean(array):
    total = 0.0
    for i in range (0, len(array)):
        total += array[i]
    mean = total/len(array)
    return mean

def logBase(x, base):
    return log(x,base)
```

Artificial Intelligence 4 Assessed Exercise

```
def sign(value):
    if (value < 0):
        x = -1
    if (value == 0):
        x = 0
    if (value > 0):
        x = 1
    return x

def variance(array):
    array2 = []
    m = mean(array)
    array2 = [(num-m)**2 for num in array]
    return mean(array2)

def probDens(array, quality):
    m = mean(array)
    var = variance(array)
    first = (sqrt(2*pi*var))
    second = exp(-(quality-m)**2/(2*var))
    return second/first
```

AI – Lab1 code – in Green

```
from numpy import *
from mathsFunctions import *
import mathsFunctions as mf
import numpy as np
import matplotlib.pyplot as plt

DataIn = loadtxt("./laboratory.dat");

Rect = []
Datasign = []
```

```
for i in range (0,240):
    Rect.append(1)

for i in range (1,len(DataIn)):
    Datasign.append(np.sign(DataIn[i]) - np.sign(DataIn[i-1]))

Datafft = mf.square(DataIn)

Datapos = mf.absolute(DataIn)

Datasign = mf.absolute(Datasign)

Energy = mf.convolve(Datafft, Rect)
Magn = mf.convolve(Datapos, Rect)
ZCR = mf.convolve(Datasign, Rect)

DataIn = mf.divide(DataIn, max(DataIn))
Energy = mf.divide(Energy, max(Energy))
Magn = mf.divide(Magn, max(Magn))
ZCR = mf.divide(ZCR, max(ZCR))

x1 = np.linspace(0, len(DataIn), len(DataIn))
x2 = np.linspace(0, len(Energy), len(Energy))
x3 = np.linspace(0, len(Magn), len(Magn))
x4 = np.linspace(0, len(ZCR), len(ZCR))


plt.figure(1)

plt.plot(x1,DataIn)

plt.plot(x2, Energy)
plt.plot(x3, Magn)
plt.plot(x4, ZCR)
```

```
plt.legend(['y = Signal', 'y = Energy', 'y = Magnitude', 'y=ZCR'], loc='upper left')

plt.show()
```