BioMetrics

Assignment 1

2015

Jonathan O’Brien

10002051

3/11/2015

Table of Contents

1. Plot the score distributions for both groups 3

2. Compute and plot the resulting DET curve 4

3. Determine the EER 5

4. If the costs of a false accept is CFA euro and a false reject is CFR euro, estimate a suitable operating point on the DET curve that minimizes the overall cost. Assume equal apriori probabilities. 5

5. Python Source Code 6

# 1. Plot the score distributions for both groups

To plot the distributions of the score data files I received I first changed the score values from there scientific format to floating point as they were read in line by line. I then used the ‘matplotlib.pyplot’ library function in Python to generate a histogram plot, overlaying the score distributions on top of each other.

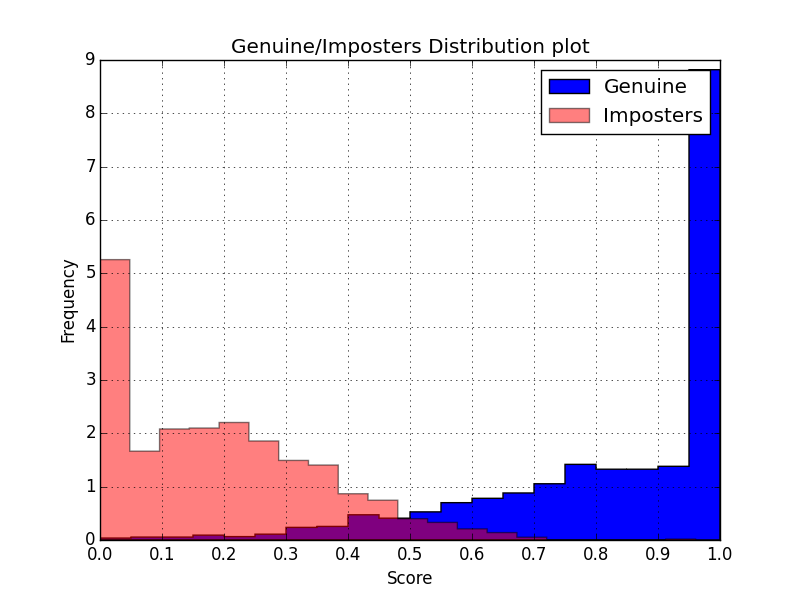


Figure 1

# 2. Compute and plot the resulting DET curve

To compute the DET Curve I first had to sort the Lists of Imposters and Genuine user scores, which I had imported earlier. I then created a loop that would generate a number of incrementing threshold values within a range of 10000 values. It would appear the bigger the range the most precise the results from the calculations become.

I then created a function that acted as a Confusion Matrix, calculating the True positive, False Positive, True Negative and False negative values for each threshold value that was passed into it. Using these value the function would also calculate the False accept and false reject rates for the threshold. Costs for each threshold are also calculated. These values are returned to a function call that is looping through the range of thresholds and which stores the results by appending them to lists.

When all the thresholds have been used to generate the error rates, I pass these Lists (arrays) of values to the python plotting function.

Below is a screenshot of the output with a resolution of 10000 thresholds in the range of 0-1. I plotted the false reject rate on the x-axis and the false accept rate on the y-axis. I also have the best operating point plotted as a red dot.

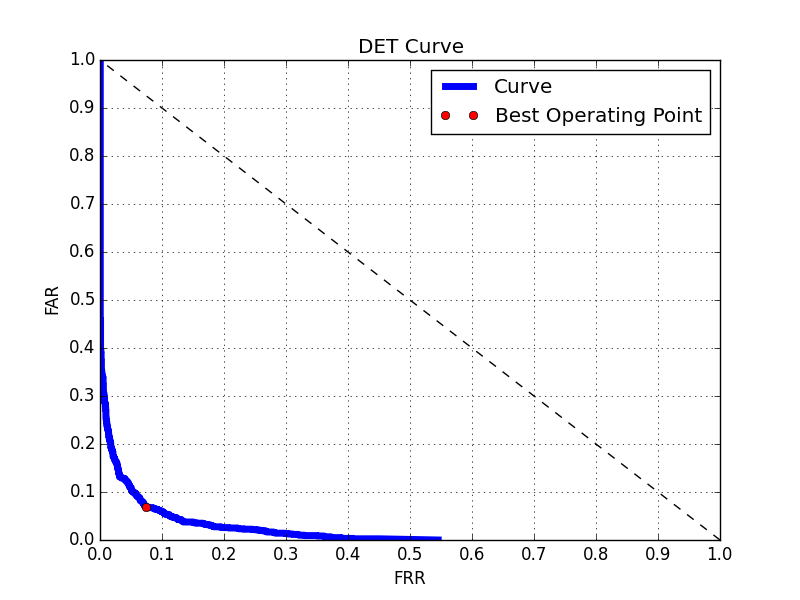


Figure 2

# 3. Determine the EER

To determine the equal error rate I added each FAR and FRR for each threshold sorted the output and selected the value that was closest to zero as being the equal error rate.

|  |  |
| --- | --- |
| ***Output from a resolution of 10000 Thresholds*** |  |
| EER (FRR + FAR) | 0.143 |
| FAR | 0.069 |
| FRR | 0.074 |
| Threshold | 0.382 |
| ERR Percentage | 7.4% |
| Performance Index | 92.6% |

Figure 3

# 4. If the costs of a false accept is CFA euro and a false reject is CFR euro, estimate a suitable operating point on the DET curve that minimizes the overall cost. Assume equal apriori probabilities.

To calculate the cost I used the formula from the notes and made it so a user could change the “apriori” probability values, but had a default value of equal with 0.5 for imposters and 0.5 for genuine users. I calculated this in the same loop as the FAR and FRR, as they used the same values. I stored these costs in a list and again found the smallest value and its accompanying rates and threshold. I then use these to plot a point on the curve as the best operating point. When using equal apriori probabilities the points generated are the same as the equal error rate.

|  |  |
| --- | --- |
| Best Cost Value | 1.071 Euro |
| FAR | 0.069 |
| FRR | 0.074 |
| Threshold | 0.382 |

Figure 4

# 5. Python Source Code

import sys

import pylab

import matplotlib**.**pyplot as plt

# The Range of Threshold values

Resolution **=** 0.0001

range\_ **=** 10000

# Hard Code defaults for A priori probabilities

# have the option to change probailites

# added this just to see what the different outcomes would be

Imposter\_priori\_probability **=** 0.5

Genuine\_priori\_probability **=** 0.5

loop **=** True

counter **=** 0

while**(**loop**):**

print "Please choose \"a priori\" Probabilties:"

choice1 **=** float**(**raw\_input**(**"Imposter :>"**))**

choice2 **=** float**(**raw\_input**(**"Genuine :>"**))**

if not **(**choice1 **>** float**(**0**)** and choice1 **<** float**(**1**))** and not **(**choice2 **>** float**(**0**)** and choice2 **<** float**(**1**)):**

print "Invalid Input - example (0.5 0.5) or (0.1 0.9)"

counter **=+** counter

elif not **(**choice1 **+** choice2**)** **==** float**(**1.0**):**

counter **=+** counter

print "please enter equal divisions"

elif counter **>=** 5**:**

print "Defaulting to equal a priori"

break

else**:**

loop **=** False

Imposter\_priori\_probability **=** choice1

Genuine\_priori\_probability **=** choice2

# load data from the dat files, converting to floating point numbers

dataI **=** **[**float**(**number**)** for line in open**(**'i.dat'**,** 'r'**)** for number in line**.**split**()]**

dataG **=** **[**float**(**number**)** for line in open**(**'g.dat'**,** 'r'**)** for number in line**.**split**()]**

# generate a distriction plot for imposters and genuine data

# using a histogram

plt**.**hist**(**dataG**,** bins **=**20**,** normed**=**True**,**histtype**=**'stepfilled'**,** color**=**'b'**,** label**=**'Genuine'**)**

plt**.**hist**(**dataI**,** bins**=**20**,** histtype**=**'stepfilled'**,** normed**=**True**,** color**=**'r'**,** alpha**=**0.5**,** label**=**'Imposters'**)**

plt**.**title**(**"Genuine/Imposters Distribution plot"**)**

plt**.**xlabel**(**"Score"**)**

plt**.**ylabel**(**"Frequency"**)**

# incease the density of ticks on the x axis

plt**.**xticks**(**pylab**.**arange**(**0**,**1.1**,**.1**))**

# add information

plt**.**legend**()**

# add a grid

plt**.**grid**(**True**)**

plt**.**show**()**

# List to store the Cost caculations assosiated with each threshold

COST\_CALC **=** **[]**

def **FARFRR(**neg**,** pos**,** threshold**):**

# Confusion Matrix

# Error

# False Negative Values when

# classified as negative but is actually over the Threshold

FN **=** 0

for i in neg**:**

if i **>=** threshold**:**

FN **=** FN **+** 1

# Error

# False Postive Values when

# classified as Postive but is actually under the Threshold

FP **=** 0

for i in pos**:**

if i **<** threshold**:**

FP **=** FP **+** 1

# True Postive when

# classified as postive and is over the Threshold

TP **=** 0

for i in pos**:**

if i **>=** threshold**:**

TP **=** TP **+** 1

# True Negative when

# claffied as negative and is under the threshold

TN **=** 0

for i in neg**:**

if i **<** threshold**:**

TN **=** TN **+** 1

# calculate the False accept rate and false reject rate

# far = float(FP)/float(len(neg))

# frr = float(FN)/float(len(pos))

far **=** float**(**FP**)/**float**(**len**(**neg**))**

frr **=** float**(**FN**)/**float**(**len**(**pos**))**

# Do this calcualtion to verify which to use

# TPR = float(TP)/float(len(pos))

# FNR = float(1) - float(TPR)

# verify this

# print "FNR", FNR, "FRR", frr

costList **=** None

# TN = len(neg) - FP

# FN = len(pos) - TP

# calculate the Best Cost

try**:**

imposterPredict **=** Imposter\_priori\_probability

genuinePredict **=** Genuine\_priori\_probability

# cost of a false accept and the cost of and false reject

CFA **=** 15

CFR **=** 15

#: Cost(T) = WFA \* FA(T) \* P(Impostor) + WFR \* FR(T) \* P(Genuine)

C **=** **(**CFA **\*** imposterPredict **\*** far**)** **+** **(**CFR **\*** genuinePredict **\*** frr**)**

print "C"**,** C**,**"threshold"**,** threshold

# store cost with the threshold and other values, for lookup

costList **=** **(**C**,**far**,** frr**,** threshold**)**

except**:**

costList **=** **(**0**,**0**,**0**,**0**)**

# return cacaultions to the caller

return far**,** frr**,** costList

# print "FARFRR TEST 0.5"

# print FARFRR(dataI, dataG, 0.5)

# incremental generation of steps for thresholds in a range

def **xfrange(**start**,** stop**,** step**):**

while start **<=** stop**:**

yield start

start **+=** step

def **EVAL(**negatives**,** positives**,** points**):**

Tarray **=** **[]**

costLIST **=** **[]**

# generate the array of threshold values

for i in xfrange**(**0**,** 1**,** points**):**

Tarray**.**append**(**i**)**

# get the number of points to calculate

points **=** len**(**Tarray**)**

far **=** **[]**

frr **=** **[]**

for i in range**(**points**):**

# generate the FRR FAR for this Threshold and store

# the results in lists

ret **=** FARFRR**(**negatives**,** positives**,** Tarray**[**i**])**

far**.**append**(**ret**[**0**])**

frr**.**append**(**ret**[**1**])**

costLIST**.**append**(**ret**[**2**])**

return far**,** frr**,** Tarray**,** costLIST

# Sort the data first

dataG**.**sort**()**

dataI**.**sort**()**

# evaluate the Genuine and Imposter data, with different number of threshold "points"

tfar**,** tfrr**,** Tarray**,** costLIST **=** EVAL**(**dataI**,** dataG**,** Resolution**)**

'''

Equal Error Rate (EER) : The equal error rate is computed as the point where

FAR = FRR for a given t. In practice, the score distributions are not continuous

and a crossover point might not exist. In this case, the EER

value is computed as follows :

http://svnext.it-sudparis.eu/svnview2-eph/ref\_syst/Tools/PerformanceEvaluation/doc/howTo.pdf

'''

print " "

# $$$$$$$$$$$$$$$$$$$$$$$$$

# EER Rate attempts

# create an array of combined error rates

# sort to find the smallest (closest to zero)

# then search for that again to print its values to the terminal

print "\n$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$"

print "----------Equal Error Rate Calculation----------"

print "$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$"

close **=** **[]**

CLOSE\_ERR\_ZERO **=** 0

ERR\_FAR **=** 0

ERR\_TRR **=** 0

ERR\_THRESHOLD **=** 0

# loop through the FAR and FRR lists (adding together) values for each

# threshold

for i in range**(**range\_**):**

close**.**append**(**tfar**[**i**]** **+** tfrr**[**i**])**

# sort the results to place the smallest at the first index

close**.**sort**()**

# do the calculations again and search for which index this occurs at

# then extract the relevant information

for i in range**(**range\_**):**

if tfar**[**i**]** **+** tfrr**[**i**]** **==** close**[**0**]:**

CLOSE\_ERR\_ZERO **=** close**[**0**]**

ERR\_FAR **=** tfar**[**i**]**

ERR\_TRR **=** tfrr**[**i**]**

ERR\_THRESHOLD **=** Tarray**[**i**]**

print "EER \t\t\t" **,** "%0.3f" **%** **(**CLOSE\_ERR\_ZERO**,)**

print "FAR \t\t\t" **,** "%0.3f" **%** **(**ERR\_FAR**,)** **,** "\nFRR\t\t\t"**,** "%0.3f" **%** **(**ERR\_TRR**,),** "\nThreshold \t\t"**,** "%0.3f" **%** **(**ERR\_THRESHOLD**,)**

print "EER Percentage =\t"**,** '{0:.2g}'**.**format**(**ERR\_TRR **\*** 100**),** "%"

print "Performance Index =\t"**,** **(**float**(**100**)** **-** float**((**'{0:.2g}'**.**format**(**ERR\_TRR **\*** 100**)))),** "%"

# attempt to get the best cost from the probally incorrectly pre-generated data

# sorting again to obtain the smallest and then again finding the index where

# this smallest values is.

array **=** **[]**

for i in costLIST**:**

array**.**append**(**i**[**0**])**

array**.**sort**()**

bestcost **=** array**[**1**]**

for i in costLIST**:**

if i**[**0**]** **==** bestcost**:**

bestcostDetails **=** i

print "\n$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$"

print "-Best Cost with apriori probabilties:"

print "--Imposter--"**,** Imposter\_priori\_probability**,**"%"

print "--Genuine--"**,** Genuine\_priori\_probability**,**"%"

print "$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$"

print "BEST\_COST\t"**,** "%0.3f" **%** **(**bestcostDetails**[**0**],)**#bestcostDetails[0]

print "FAR\t\t"**,** "%0.3f" **%** **(**bestcostDetails**[**1**],)**#bestcostDetails[1]

print "FRR\t\t"**,** "%0.3f" **%** **(**bestcostDetails**[**2**],)**#bestcostDetails[2]

print "Threshold\t"**,** "%0.3f" **%** **(**bestcostDetails**[**3**],)**#bestcostDetails[3]

# plot the DET curve data from the FAR FRR Lists

import matplotlib**.**pyplot as mpl

mpl**.**plot**(**tfrr**,** tfar**,** label**=**"Curve"**,** color**=**"blue"**,** linestyle**=**'-'**,**linewidth**=**5**)**

# add a grid

mpl**.**grid**(**True**)**

# use x and y limits

mpl**.**ylim**((**0**,**1**))**

mpl**.**xlim**((**0**,**1**))**

mpl**.**xticks**(**pylab**.**arange**(**0**,**1.1**,**.1**))**

mpl**.**yticks**(**pylab**.**arange**(**0**,**1.1**,**.1**))**

# add a division line

mpl**.**plot**([**1.0**,**0.0**],** **[**0.0**,**1.0**],**'k--'**)**

#plot the best operating point on the curve

mpl**.**plot**(**bestcostDetails**[**2**],**bestcostDetails**[**1**],**'ro'**,** label**=**"Best Operating Point"**)**

mpl**.**xlabel**(**'FRR'**)**

mpl**.**ylabel**(**'FAR'**)**

mpl**.**title**(**"DET Curve"**)**

mpl**.**legend**()**

mpl**.**show**()**