Data Analysis and Modelling

Probability and Statistics Exam

Joseph Pritchard - Adelaide University

June 19, 2020

Question I

a) We wish to find the sum of probabilities the test provides a true result. This is given by

$$\begin{aligned} P(T = true) =& P(T = true|terrorist = false)P(terrorist = false) \\ &+ P(T = true|terrorist = true)P(terrorist = true) \end{aligned} \tag{1}$$

Substituting our known values we then have

$$\begin{aligned} P(T = true) &= (0.1)(0.99999) + (0.9)(0.00001) \\ &= 0.100008 \\ &= 10.0008\% \end{aligned} \tag{2}$$

b) Observe the following, utilising Bayes rule

$$\begin{aligned} P(terrorist = false|T = false) &= \frac{P(terrorist = false, T = false)}{P(T = false)} \\ &= \frac{P(T = false|terrorist = false)P(terrorist = false)}{\sum_{terrorist} P(T = false|terrorist)P(terrorist)} \end{aligned} \tag{3}$$

Inserting our known values we then have

$$\begin{aligned} P(terrorist = false|T = false) &= \frac{(0.9)(0.99999)}{(0.1)(0.00001) + (0.9)(0.99999)} \\ &= 0.9999989 \end{aligned} \tag{4}$$

c) Same process as (c) setting *terrorist* and T values to *true* thus

$$P(terrorist = true|T = true) = \frac{(0.9)(0.00001)}{(0.9)(0.00001) + (0.1)(0.99999)} = 0.00008999 \quad (5)$$

d) Mis-classification occurs when either $terrorist = false$ AND $T = true$ OR $terrorist = true$ AND $T = false$. Note the following

$$P(terrorist = true, T = false) = P(T = false|terrorist = true)P(terrorist = true)$$

$$P(terrorist = false, T = true) = P(T = true|terrorist = false)P(terrorist = false)$$

Substituting our known values we have

$$P(terrorist = true, T = false) = (0.1)(0.99999) = 0.099999 \quad (6)$$

$$P(terrorist = false, T = true) = (0.1)(0.00001) = 0.000001 \quad (7)$$

Giving a total mis-classification probability of

$$0.099999 + 0.000001 = 0.1$$

Question II

Here we utilise the functions norm.cdf and norm.expect from the scipy.stats Python library. The results are

a) 0.841
b) 0.092
c) 5803.75

Question III

Here we use the function quad from the scipy.integrate Python package and act on sum of two gaussian distributions we define ourselves. The result is

$$P(x > 8) = 0.9313$$

Note that an upper bound of 100 is chosen as a limit of integration. Increasing the upper bound past this point provided negligible improvement.

Question IV

For (a), (b) and (c) the Python function *poisson* and associated methods are used. The resulting values are

```
4a: 5.9999999999999964
4b: 0.0024787521766663585
4c: 0.042620923582537995
```

Figure 1:

Question V

For both (a) and (b) the Python functions *numpy.cov* and *numpy.corrcoeff* are used. The resulting matrices are

```
5a :
[[  49.99        -91.12833333  138.41666667  391.04166667]
 [ -91.12833333  247.9225     -187.09166667 -485.1375    ]
 [ 138.41666667 -187.09166667  535.10333333 1287.32833333]
 [ 391.04166667 -485.1375     1287.32833333 3698.0225    ]]
5b :
[[ 1.         -0.81856635  0.84630711  0.90948741]
 [-0.81856635  1.         -0.51366224 -0.50666639]
 [ 0.84630711 -0.51366224  1.          0.91513618]
 [ 0.90948741 -0.50666639  0.91513618  1.        ]]
```

Figure 2:

Question VI

To find the global minima of the Rosenbrock function, a differential evolution algorithm is used. Here, n random points in the parameter space are selected to form the first generation of points. We then create a donor vector for each of our points, constructed from the other points within our current generation as

$$V_i = \lambda X_{best}(1-\lambda)X_{r1} + F(X_{r2} - X_{r3})$$

where X_{ri} are the i random points selected, F is a mixing function, X_{best} is the current best point and λ controls the mixing of the current best point. We then construct a trial vector U_i, where each each element is either the corresponding element in X_i or V_i. The probability of selection from either vector is controlled by a parameter C. To ensure $U_i \neq X_i$ we then randomly select one element of U_i and replace it by the corresponding value of V_i. We now evolve to the next generation of points by choosing the vector V_i or X_i which minimizes our target function.

This algorithm is implemented via the Python $differentialevolution$ function, where the paramter *recombination* detemines the mixing variable C. Both the mixing function F and best mixing value λ are set internally by the function. To ensure the minima found are indeed global, the function is rerun multiple times. Each time the 5D minima is unchanged and the 7D minima retains the same parameter values. The 7D function value changes with each re-seeding however all values are approximately zero. The results are

```
6a:
Function value:
0.0
Parameter values:
[1. 1. 1. 1. 1.]
6b:
Function value:
8.663664891589762e-28
Parameter values:
[1. 1. 1. 1. 1. 1. 1.]
```

Figure 3:

Question VII

a) To prevent the likelihood values from becoming too, in order to maximise the likelihood, we instead minimize the negative log likelihood.

Optimisation is performed by first selecting a set of parameters ϕ_i. Then, for every bin of data, we assume a Poisson spread and evaluate the probability of observing our measured data given the parameter set ϕ_i. We then average over all bin probabilities to provide an average negative log likelihood. The optimum result is selected using the genetic algorithm method outlined in question VI.

To provide guidance for parameter search bounds we observe the histogram plot of the gathered data. In analysing we note a bump in the plot around the value of 4 along the x axis, suggesting a search for the mean parameter around this value. Similarly the spread of the bump appears to be around 1 units on the x axis, suggesting a range for the sigma parameter. Values for the remaining parameters A and B are found by referencing the values provided in the course notes and setting bounds capturing these values. Initially optimum parameters for A and B lie at the boundaries but through boundary adjustment optimisation is achieved within the parameter bounds.

The resulting MLE values for the parameters A, B, μ and σ are

```
[167.14300069 156.59594146   4.63419107   1.3741325 ]
```

Figure 4:

An overlay of the background function, signal, background signal sum and raw data histogram is provided in figure . Note the good fit of the background signal sum in capturing the bump in the observed data (Figure 5).
b) To provide a confidence level on the rejection of a $B = 0$ hypothesis, we define a test statistic as the log ratio of likelihoods as

$$t = -2ln\left(\frac{L(a,b)}{L(\phi_i)}\right)$$

Here a is the value describing our test hypothesis $B = 0$, b is the set of parameters which maximise the likelihood for this test hypothesis and ϕ_i are the parameters calculated previously maximizing the likelihood function. Maximisation of $L(a,b)$ is achieved using the same genetic algorithm described above. Assuming Wilk's theorem, we know the test statistic follows a χ^2 distribution.

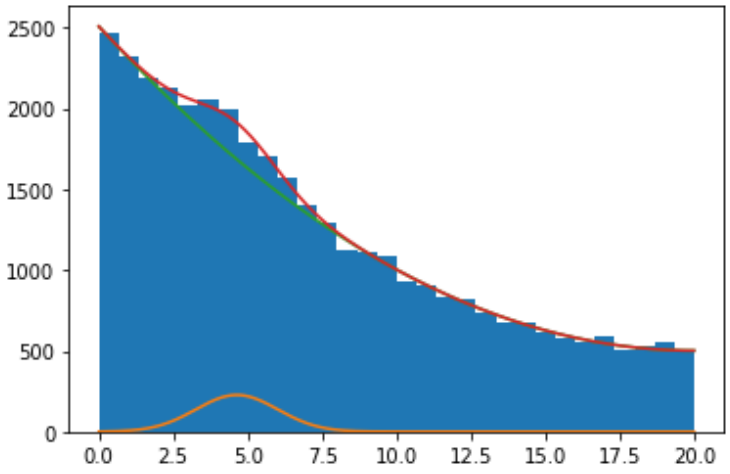

Figure 5:

As such we find the value of the test statistic required for 95% confidence in a χ^2 distribution and compare this to our t value calculated. The results are shown in figure 6.

```
Test statistic is: 79.04067274187514
Chi^2 95% value is: 3.841458820694124
```

Figure 6:

Appendix

Following is the Python code used in the generation of all plots included in this exam.

```
import numpy as np
from scipy.stats import norm
from scipy.stats import poisson
import scipy.integrate as integrate
import matplotlib.pyplot as plt
import scipy.optimize as opt
import pandas
from scipy.stats import chi2



#Question 1a
Pa = 0.1*0.99999 + 0.9*0.00001
print("1a: " + str(Pa))

#Question 1b
Pb = ( (0.9)*(0.99999) )/( (0.1)*(0.00001) + (0.9)*(0.99999) )
print("1b: " + str(Pb))

#Question 1c
Pc = ( (0.9)*(0.00001) )/( (0.9)*(0.00001) + (0.1)*(0.99999) )
print("1c: " + str(Pc))

#Question 1d
Pda = (0.1)*(0.99999)
print("1ca: " + str(Pda))
Pdb = (0.1)*(0.00001)
print("1cb: " + str(Pdb))
print("1c_Combined: " + str(Pda + Pdb))



####

#Question 2a
y = norm.cdf(100,70,30.1)
print("2a: " + str(y))

y = norm.cdf(110,70,30.1)
print("2b: " + str(1 - y))

def squared(x):
    return x**2
y = norm.expect(lambda x: x**2, loc = 70, scale = 30.1, lb=0, ub=200)
print("2c: " + str(y))



####

#Question 3
```

```
def DG(x):
    return 0.8*norm.pdf(x, 10, 1) + 0.2*norm.pdf(x, 10, 3)

y = integrate.quad(DG, 8, 1000)
print("3: " + str(y))



####

#Question 4a

G = poisson(6)
expect = G.dist.expect(lambda x: x, G.args, lb=0, ub=np.inf)
print("4a: " + str(expect))

b = poisson.pmf(0, 6)
print("4b: " + str(b))

c = 1 - poisson.cdf(10, 6)
print("4c: " + str(c))



####

#Question 5a

#Each row contains 1 variable from each dimension
x = np.array([ [ 3.1, 200.1, 42.6, 7.9 ], [10.2, 168.3, 39.2, 9.1 ], [5.7, 192.3, 23.4,
cov = np.cov(x)
print("5a :")
print(cov)

cor = np.corrcoef(x)
print("5b :")
print(cor)



####
print("")
#Question 6

results = opt.differential_evolution(opt.rosen, bounds = [(-2,2),(-2,2),(-2,2),(-2,2),(-
print("6a: ")
print("Function value: ")
print(opt.rosen(results.x))
print("Parameter values: ")
print(results.x)

results = opt.differential_evolution(opt.rosen, bounds = [(-2,2),(-2,2),(-2,2),(-2,2),(-
```

```
print("6b: ")
print("Function value: ")
print(opt.rosen(results.x))
print("Parameter values: ")
print(results.x)


####

#Question 7

#Import data
data = pandas.read_csv("data.csv")


#Define background function
def background(myy, A=1):
    a = 15
    b = -1.2
    c = 0.03

    result = A * (c*myy**2 + b*myy + a)
    return result

#Define signal function
def signal(myy, B, mean, sigma):

    result = B*5*norm.pdf(myy, mean, sigma)

    return result

#Define bin locations
x = np.arange(0, 20, 0.01)

#Extract data
lhc_data = np.genfromtxt('data.csv', delimiter = ',')

#Convert data into histogram form and store edge locations
entries, edges, _ = plt.hist(lhc_data, bins = 30, range=[0, 20])

#Calculate bin centers
bin_centers = 0.5 * (edges[:-1] + edges[1:])


#Define minus log likelihood function
def minus_log(params):
    A, B, mu, sigma = params
    result = 0
    entries, edges, _ = plt.hist(lhc_data, bins = 30, range=[0, 20])

    bin_centers = 0.5 * (edges[:-1] + edges[1:])
```

```
    n = bin_centers.size

    for i in range(0, n):
        b = background(bin_centers[i], A)
        s = signal(bin_centers[i], B, mu, sigma)
        lambda_i = s + b
        k_i = entries[i]

        result += poisson.logpmf(k_i, lambda_i)
    return -1*result


#plt.clf()


#Optimise minus log likelihood
results = opt.differential_evolution(minus_log, bounds = [(160,170),(150,170),(0,5),(0,2


print("7:")
print(results.x)
plt.plot(x, signal(x, results.x[1], results.x[2], results.x[3]))
plt.plot(x, background(x, results.x[0]))
plt.plot(x, background(x, results.x[0]) + signal(x, results.x[1], results.x[2], results.
plt.show()

plt.plot(x, signal(x, 163.2, 2.7, 1))
plt.plot(x, background(x, 166.5))
plt.plot(x, background(x, 166.5) + signal(x, 163.2, 4.5, 1))
plt.show()



mrl = opt.differential_evolution(minus_log, bounds = [(160,170),(0,0.01),(0,5),(0,2)])#


background_L = minus_log([mrl.x[0],0,mrl.x[2], mrl.x[3]])
new_model_L = minus_log([results.x[0],results.x[1],results.x[2], results.x[3]])

#Assuming wilks theorem, the test statistic
# t = -2( ln(background) - ln(new_model))
# will have a chi^2 distribution
# So, we compare the value of t with the
# value of 95% confidence for a chi^2
# distribution

t = -2*(-background_L + new_model_L)

print("7b: ")

print("Test statistic is: " + str(t))
```

```
crit = chi2.ppf(q = 0.95, df = 1)
print("Chi^2 95% value is: " + str(crit))
```