Assignment 2

CS146

Asmaa Aly

Fall, 2019

Full code https://drive.google.com/file/d/1-s7u9ejoC70vByoPXZ8Bf-b482Vm8kCi/view?usp=sharing

(1) Call center data modeling

(1)

```
lowerbound = []
higherbound = []
percentiles = []
for m in range(24):
    percentiles.append([compute_percentile(lambdas, posteriorlist[m], p)
                        for p in [0.01, 0.99]])
    lowerbound.append(percentiles[m][0])
    higherbound.append(percentiles[m][1])
```

Figure 1: the following figure shows the code I used to calculate the confidence intervals for the posterior.

```
for u in range(24):
    mean = sp.integrate.trapz(posteriorlist[u] * lambdas, lambdas)
    meanlist.append(mean)
```

Figure 2: the following figure shows the python code I used to calculate the confidence intervals for the posterior for each hour.

(2)

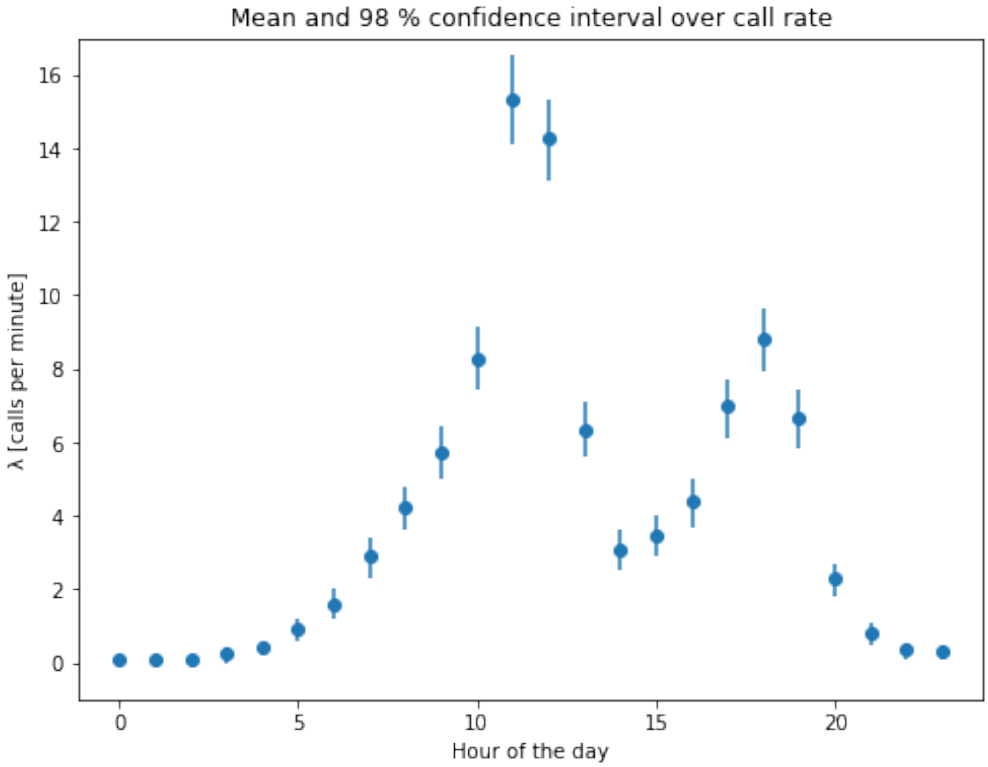


Figure 3: the following figure shows the mean values of the call rates per minutes over each hour of the day. The vertical lines show a 98 confidence intervals over these rates.

(3)

The plot starts with a small average of calls per minute, between zero and four, from midnight until 7 AM. There is a rapid rise in the average rate from 3 to 15 calls per minute between 7 and 11 AM. The plot reaches its peak at 11 AM and starts falling to reach 3.5 at 3 PM. It reaches a smaller peak at 6 PM with an average rate of 8.7 and falls to reach 0 at 11 PM. The vertical lines are indicators of the range of the values that have 98% chance of including where the correct average rate lies. As the number of calls increases, the vertical lines are longer, the estimated range is wider because the marginal error is also increasing. For

example, the peak values between 11 and 12 PM have a wider range as they have more calls per minute. On the other hand, the fewer number of the calls have a smaller range and smaller marginal error, as seen between midnight and 3 AM.

(2)

(1)

There are two reasons. First, from the lines in the function, the posterior is calculated as Maximum A Posterior Estimation such that we try to maximize the parameters values given data, the parameter value here is the lambda λ. Since the likelihood function is exponential, and since the exponential family are logarithmically concave,it is easier to work with a logarithmic form of the likelihood function to maximize it. The second reason is that taking a product of numbers less than 1, prior* the likelihood, would approach zero as the number of those numbers goes to infinity. Thus, they would be not practical to compute because of computation underflow. Hence, we will instead work in the logarithmic space.

$$\lambda_{MAP} = \arg\max_{\lambda} P(data|\lambda)P(\lambda) \tag{01}$$

By taking the log of the equation,

$$= \lambda \arg\max \log P(data|\lambda) + \log P(\lambda) \tag{02}$$

(2)

In order to get the likelihood distribution for all of the points in the data, we need to multiply each of the functions for each point x_i in the data. Thus,

we can write equation 2 as

$$= \lambda \arg\max \log \prod_i P\left(x_i|\lambda\right) + \log P(\lambda) \tag{03}$$

Because we are taking the log of this product, we can rewrite the equation as

$$= \lambda \arg\max \sum_i \log P\left(x_i|\lambda\right) + \log P(\lambda) \tag{04}$$

Thus, the numpy sum is doing the same process in the likelihood function that loops over each of the data points x_i in data or X.

(3)

It is an important step to ensure numerical stability while performing the calculations on Python as the exponential function grows very fast with positive exponents. Subtracting the maximum log unnormalized ensure posterior ensures that log unnormalized values would not cause overflow or underflow while computing the expression.

(4)

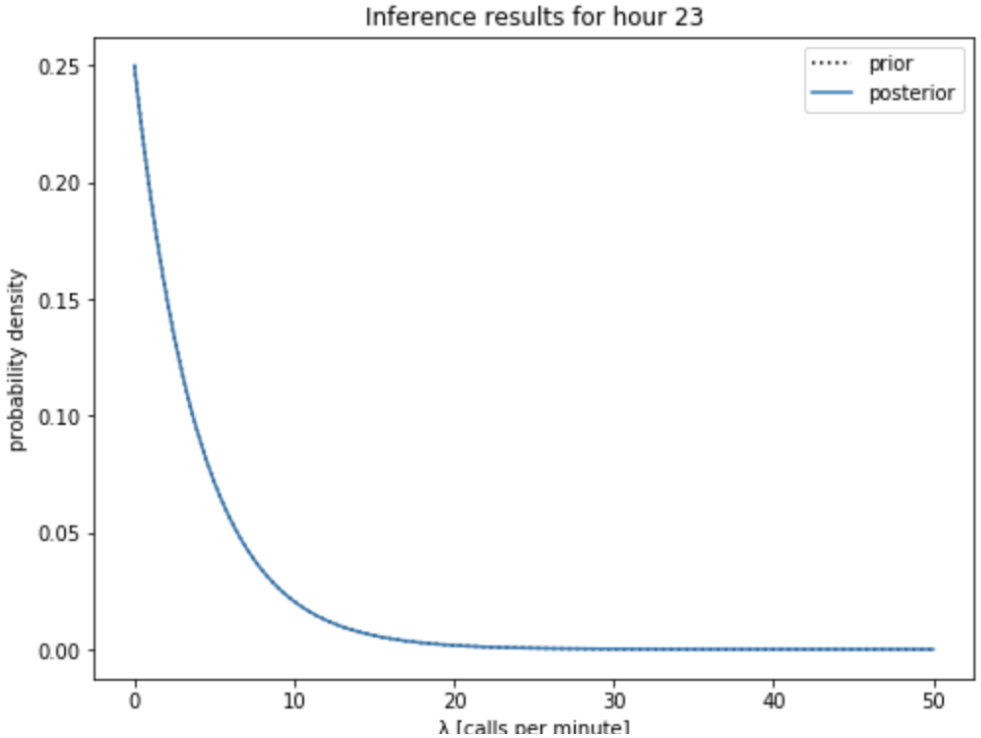
Since we need to make sure that the resultant posterior distribution has an area of one, we need to divide the unnormalized distribution by the area to normalize it.

(5)

For a likelihood function like the following:

$$\frac{1}{\sqrt{2\pi}} \exp \frac{-(x)^2}{4} \tag{05}$$

The python code using the log worked and plotted the following graph



However, the python code without the algorithm printed a run time error as the output exceeded the float limit, 64 bits, of python. Analytically, the optimization of a second degree normal distribution density function is harder without taking the log because of the rate of expansion of the function.