Maximum Likelihood Estimation (MLE)

Anabel Yong

Abstract

In this report, the concepts of maximum likelihood estimation, and likelihood functions are introduced. These concepts will be explained and the different approaches utilized to calculate these functions. An example will be provided throughout the report, which is the ABO blood group example where we are estimating the parameters for p, q, r (the alleles according to Hardy-Weinberg principle).

Introduction

Likelihood

The concept of likelihood is important in both maximum likelihood estimation and Bayesian inference. Likelihood is not probability; however, it is proportional to probability. For likelihood, data is treated as a given and hypothesis varies whereas probability is the opposite of this. A prime example can be shown in the drug trial test.

Suppose you are running an experiment trial n times, x times a drug is successful are observed. As there are only two probable outcomes, the probability of getting x heads is given by the binomial distribution. Now, if this is a fair probability, the probability of getting a successful drug trial is 0.5. Binomial distribution is given by:

$$P(\chi = x|p) = \binom{n}{x} p^x (1-p)^{(n-x)} \tag{1}$$

If the probability was fair, the probability of getting 6 successful trial runs and 4 unsuccessful trials is given by:

$$P(\chi = 6|p = 0.50) = \frac{10!}{6! * 4!} * (0.5)^6 (0.5)^4 = 0.21 \tag{2}$$

If the probability of the drug was more likely to work, indicating this was a biased trial, probability of getting successful drug trial is p=0.75, the probability of getting 6 successful trial runs and 4 unsuccessful trials is given by:

$$P(\chi = 6|p = 0.75) = \frac{10!}{6! * 4!} * (0.75)^6 (0.25)^4 = 0.15 \tag{3}$$

With this example, the **likelihood** ratio of getting 6 successful drug trials in 10 experiment runs for a biased trial versus a fair trial would be denoted by:

$$\textit{Likelihood Ratio}(P = 0.5, 0.75) = \frac{P(\chi = 6|p = 0.50)}{P(\chi = 6|p = 0.75)} = 1.4 \tag{4}$$

The example above showcases how calculating likelihood determines whether we can trust the parameters in model based on the sample data observed. Calculating the likelihood helps us determine if the model parameter p=0.5 is defined correctly. This indicates, from the example above, the probability of a fair trial(p=0.5) is 1.4 times more likely than a biased trial (p=0.75).

Maximum Likelihood Estimation (MLE)

For the concept of MLE, the example of the ABO Blood group will be presented. From the data table below,

Table 1:

Phenotype	Genotype	Probability	Count
A	AA+AO	$p^2 + 2pr$	44
B	BB+BO	$q^2 + 2qr$	27
AB	AB	2pq	4
O	OO	r^2	88

As there are multiple outcomes from the table above, the probability is given by a multinomial distribution. Therefore, the likelihood function is given by:

$$L(p,q) = (p^2 + 2pr)^{nA} * (q^2 + 2qr)^{nB} * (2pq)^{nAB} * (r^2)^{nO} \tag{5}$$

or log-likelihood function:

$$l(p,q) = n_A * log(p^2 + 2pr) + n_B * log(q^2 + 2qr) + nAB * log(2pq) + nO * log(r^2) \tag{6}$$

As shown in equations 5 and 6 above, values of the parameters p, q, and r can change, which changes the value of the likelihood/ log likelihood function. Indicatively, in order to maximise probability of data fitting the model given in table, we need to calculate values of unknown parameters should achieve maximum probability through the likelihood function.

In statistics, maximum likelihood estimation (MLE) is a method of estimating the parameters of a statistical model given observations, by finding the parameter values that maximize the likelihood of making the observations given the parameters. This can be done in multiple ways, which includes setting the derivative of the function to 0 and or utilizing non-linear optimization methods.

Approaches to deriving Maximum Likelihood Estimation

Setting Derivative to 0

In simple calculus, usually gaining the derivative of an equation, which provides us the gradient will help us attain local minimum/ maximum. When setting the derivative to 0, this means the function is at critical or stationary point. This concept can theoretically be achieved to the likelihood function to attain the maximum likelihood.

However, calculating the derivative of likelihood function is quite hard to deal with due to the products and numerical precision is hard to maintain going into very tiny probabilities. This is not tractable analytically so the log-likelihood function is much preferred, which is the non-linear optimisation method.

Non-linear Optimization Method

Non-linear optimization method utilizes the log likelihood function. It is much easier to analyse equation 6 above when equating it to 0. In the ABO blood group example, this is a non-linear method as the objective function is the log likelihood function. In this method, to maximise likelihood, the objective function is:

$$max.f = n_A * log(p^2 + 2pr) + n_B * log(q^2 + 2qr) + nAB * log(2pq) + nO * log(r^2) \tag{7}$$

$$s.t. p + q < 1 \tag{8}$$

$$p, q, r > 0, \tag{9}$$

There are more constraints the objective function is subjected to but equation 8 and 9 are examples of the constraints. Using the non-linear optimisation method is computationally efficient. Using this method, the plots of log likelihood against p and q can be visualised with the initial observed data given in Table 1.

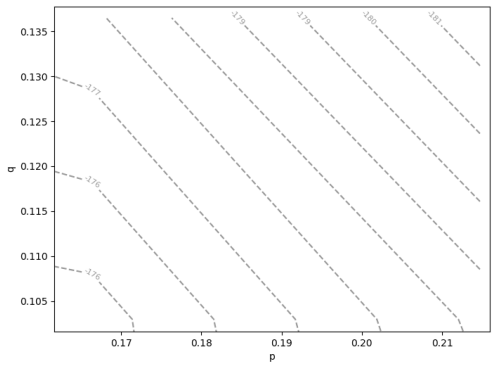

Figure 1: 2D Contour Plot of Log Likelihood against p and q

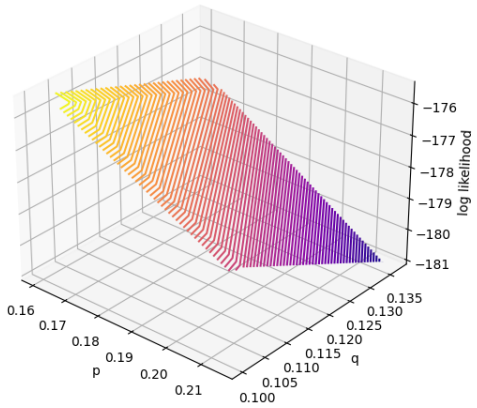

Figure 2: 3D Contour Plot of Log Likelihood against p and q

From both the 2D and 3D plots, the graphs represent the relationship of log likelihood against p and q parameters which we are trying to obtain the best parameter values in order for the observed data to be most probable. In the 2-dimensional plot, the highest likelihood value is around -176, where we can gain estimates of p and q values, which we can also derive r from the Hardy-Weinberg equilibrium, where the allele r is given by r=1-p-q. In the 3-dimensional plot, the yellow area provides the best parameter values as the likelihood function is the highest here.

Gene Counting Algorithm

The gene-counting algorithm is an example which demonstrates how the maximum likelihood function is used in the non-linear optimisation method. With constraints such as:

$$p = (\frac{1}{2n}) * (nAB + nA * (1 + hA)) \tag{10}$$

$$q = (\frac{1}{2n}) * (nAB + nB * (1 + hB)) \tag{11}$$

$$r = 1 - p - q \tag{12}$$

$$hA = \frac{p}{p+2r} \tag{13}$$

$$hB = \frac{q}{q+2r} \tag{14}$$

This is computationally carried out with Python. The number of optimization rounds is unknown. The gene counting function with constraints above is called in an infinite loop, and in every iteration, the difference between the new optimized parameter value (p, q, r) and old values before optimization is calculated. When the difference for each parameter is smaller than a significant constant, $10e^5$, the infinite loop is stopped where the results for log likelihood and parameter values are then plotted.

As shown in Figure 1 and 2, the gene counting algorithm is non-decreasing. Mathematically, this could be proven by the first derivative of log likelihood function where the derivative is always bigger than 0. In terms of visualisation from the 2D and 3D plots, when the gene counting algorithm is run multiple times, the function is always going up.

Appendix

Python Script/Code

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sat Sept 3 09:10:08 2022

@author: bananabelyong
"""
import numpy as np
import matplotlib.pyplot as plt

from functools import partial

def estimation_equations(nA, nB, nAB, nO, hA, hB): #packaged phenotypes and hA, hB
    """
    Function is called estimation equations to define equations 1.9 and 1.10 from ABO blood
    calculate p, q, r, and hA and hB in an infinite loop.
    """
    # initialize sample size
    n = nA + nB + nAB + nO

    # calculate p, q, r from given hA and hB from initilization.
    p = (1/(2 * n)) * (nAB + nA * (1 + hA))
    q = (1/(2 * n)) * (nAB + nB * (1 + hB))
    r = 1 - p - q

    hA = p / (p + 2*r)
    hB = q / (q + 2*r)

    return p, q, r, hA, hB

def log_likelihood(nA, nB, nAB, nO, p, q, r):
    """
    Code structure calculates the log likelihood according to ZiHeng's email comments
    """
    # +=adds another value with the variable's value and assigns the new value to the vari
```

```
    # calculate log likelihood from A, B, AB and O count respectively.
    lnL=nA * np.log(p**2 + 2*p*r)
    lnL+=nB * np.log(q**2 + 2*q*r)
    lnL+=nAB * np.log(2*p*q)
    lnL+=nO * np.log(r**2)

    return lnL

def gene_counting(): #run gene counting algorithm in a loop
    # initialize parameters
    nA = 44; nB = 27; nAB = 4; nO = 88 #sample given from Morton (1964)
    hA = 0.5; hB = 0.5  # initial guesses for hA and hB

    abs_criteria = 1e-5   #loop termination criteria

    #initialize parameters and log likelihood
    p, q, r, hA, hB = estimation_equations(nA, nB, nAB, nO, hA, hB)
    p_old = p
    q_old = q
    r_old = r

    plist = []; qlist = []; rlist = []
    plist.append(p)
    qlist.append(q)
    rlist.append(r)

    lnL_list = []
    lnL_list.append(log_likelihood(nA, nB, nAB, nO, p, q, r))

    # run algorithm
    while True: #infinite loop

        # estimate new parameter values
        p, q, r, hA, hB = estimation_equations(nA, nB, nAB, nO, hA, hB)
        plist.append(p)
        qlist.append(q)
        rlist.append(r)

        # calculate log likelihood
        lnL_list.append(log_likelihood(nA, nB, nAB, nO, p, q, r))

        # find absolute difference of old and new parameters p, q, r
        pdiff = np.abs(p_old - p)
        qdiff = np.abs(q_old - q)
        rdiff = np.abs(r_old - r)

        # termination condition
        if (pdiff < abs_criteria and qdiff < abs_criteria) and rdiff < abs_criteria:
            break

        # if termination condition fails, store new parameter values as old values
        p_old=p
        q_old=q
        r_old=r
```

```
    print("Log Likelihood Values")
    print(lnL_list)

    print('P values:')
    print(plist)
    print('Q values:')
    print(qlist)
    print('R values:')
    print(rlist)
    """
    # plot p vs q
    fig = plt.figure(figsize=(8,6))

    plt.plot(plist, qlist, 'r-')
    plt.xlabel('p')
    plt.ylabel('q')
    """
    # 2D contour figure
    fig2 = plt.figure(figsize=(8,6))
    ax   = fig2.add_subplot(111)

    P, Q = np.meshgrid(plist, qlist)

    lnL_partial = partial(log_likelihood, nA=nA, nB=nB, nAB=nAB, nO=nO)

    LnL = lnL_partial(p=P, q=Q, r=1-P-Q)
    contours =  ax.contour(P, Q, LnL, 10, colors='black', alpha=0.4)
    ax.set_xlabel('p')
    ax.set_ylabel('q')
    ax.clabel(contours, inline=True, fontsize=8, fmt='%.0f')

    ax.set_title('2D contour')

    # 3D contour figure
    fig3 = plt.figure(figsize=(8,6))

    P, Q = np.meshgrid(plist, qlist)

    ax2 = plt.axes(projection='3d')
    ax2.contour3D(P, Q, LnL, 100, cmap='plasma')
    ax2.set_xlabel('p')
    ax2.set_ylabel('q')
    ax2.set_zlabel('log likelihood')
    ax2.set_title('3D contour')

    plt.show()

if __name__=="__main__":
    gene_counting()
```