4

a) Expressions

① Gradient:

Let our model be

$$\boxed{P_i = \frac{1}{1+e^{-(B_0 + B_1 X_{i1} + B_2 X_{i2} + \cdots + B_n X_{in})}}}$$

Loss function for Logestic Regression is

$$L(B) = \sum_{i=1}^{N} \left[y_i \log(P_i) + (1-y_i)\log(1-P_i) \right]$$

where:

N = No of Samples

P_i = Probability that i^{th} sample having $Y = 1$

$\rightarrow$ Predicted probability

y_i = observed outcome of i^{th} sample.

Now gradient will be $\dfrac{\partial L(B)}{\partial B_j}$

$$\Rightarrow \frac{\partial L(B)}{\partial B_j} = \sum_{i=1}^{N} \left(\frac{y_i}{P_i} - \frac{(1-y_i)}{(1-P_i)} \right) \times \frac{\partial P_i}{\partial B_j}$$

$$\frac{\partial P_i}{\partial \beta_j} = P_i(1-P_i)\, x_{ij}$$

$\Rightarrow$ Gradient $$\frac{\partial L(\beta)}{\partial \beta_j} = \sum_{i=1}^{N} \left(y_i(1-P_i) - (1-y_i)P_i \right) x_{ij}$$

$$\boxed{\text{Gradient} = \sum_{i=1}^{N} (y_i - P_i)\, x_{ij}}$$

② <u>Hessian</u>:

$$H_{ij} = -\frac{\partial^2 L(\beta)}{(\partial \beta_i)(\partial \beta_j)} = -\frac{\partial}{\partial \beta_i}\left[\frac{\partial L(\beta)}{\partial \beta_j}\right]$$

(or)

$$H_{jk} = \frac{-\partial^2 L(\beta)}{(\partial \beta_j)(\partial \beta_k)} = \frac{-\partial}{\partial \beta_j}\left[\frac{\partial (L(\beta))}{\partial \beta_k}\right]$$

$$= \frac{-\partial}{\partial \beta_j}\left[\sum_{i=1}^{N} (y_i - P_i)\, x_{ik}\right]$$

$$= \cancel{\sum_{i=1}^{N} (y_i - 1) x} - \sum_{i=1}^{N} (-1)\, x_{ik} \frac{\partial P_i}{\partial \beta_j}$$

$$\boxed{H_{jk} = \sum_{i=1}^{N} P_i(1-P_i)\, x_{ij} x_{ik}}$$

$\Rightarrow$ Hessian Matrix is (Size $M \times M$) (M = No of parameters)

$$\begin{bmatrix}
\sum_{i=1}^{N} P_i(1-P_i)(x_{i1})^2 & \sum_{i=1}^{N} P_i(1-P_i)\, x_{i1} x_{i2} & \cdots & \sum_{i=1}^{N} P_i(1-P_i)\, x_{i1} x_{iM} \\
\sum_{i=1}^{N} P_i(1-P_i)\, x_{i2} x_{i1} & \sum_{i=1}^{N} P_i(1-P_i)(x_{i2})^2 & \cdots & \\
\vdots & \vdots & & \vdots \\
\sum_{i=1}^{N} P_i(1-P_i)\, x_{iM} x_{i1} & \cdots & & \sum_{i=1}^{N} P_i(1-P_i)(x_{iM})^2
\end{bmatrix}$$

③ update rules

$$\boxed{(\beta_i)_{n+1} = (\beta_j)_n - \frac{\partial(L(\beta))}{\partial \beta_j}}$$ → should do this for all β_j (j = 1 to m)

For ~~log~~ gradient descent

Let 'β' be the vector of all co-efficients

β_{n+1} → updated for n^{th} time

β_n → updated for $(n-1)^{th}$ time.

$$\boxed{\beta_{n+1} = \beta_n - [H^{-1} \cdot \nabla L(\beta_n)]}$$

All the coefficients will be updated

Newton-Raphson method

H^{-1} → Inverse of Hessian Matrix

$\nabla L(\beta)$ → Gradient vector.

④ Algorithm :

ⓑ (Psuedo code)

Let $z = 10^{-6}$

while ($\beta_{n+1} - \beta_n > z$):

$\beta_{n+1} = \beta_n - [H^{-1} \cdot \nabla L(\beta_n)]$

↳ More Mathematical

Lets see a more code related algo

```
import numpy as np

def sigmond(z):
    (Returns probablitiy, i.e 1/1+ np.exp(-z))

def gradient (X, y, beta):
    (Returns the gradient updated value, i.e Bn+1)

def hessian (X, beta):

    (Returns the Hessian matrix)
```

```
def newton_raphson(X, y):

    beta = np.zeroes(N)

    while True:

        gr = gradient(X, y, beta)
        he = hessian (X, beta)

        beta_new = beta - np.linalg.solve(he, gr)

        If (beta_new - beta < 10^-6):
            Break.

    beta_new = beta
    Return beta_new

# X is the feature matrix
# y is the binary outcome vector
```