Linear Regression

Algorithm

1 Input:
Dataset with one independent variable (x) and one dependent variable (y)

2 Initialize:
~~coefficients~~ (m) = 0 m = slope
~~slope~~ (b) = 0 b = intercept.

3. Training.
for each iteration
calculate predicted value : $y = mx + b$
calculate error value : $mx + b - y$ (Predicted value - actual value)
update coefficients.
$$m = m + \alpha \cdot \frac{1}{n} \Sigma(\text{error} * x)$$
$$b = b + \alpha \cdot \frac{1}{n} \Sigma(\text{error})$$

Pseudo code

```
function LR (x, y, alpha, iterations):
    Initialize m=0, b=0
    n = length(x)
    for i from 1 to iterations:
        y_pred = m * X + b
        error = y - y_pred
        m = m + alpha * (1/n) * sum(error * X)
        b = b + alpha * (1/n) * sum(error)
        ~~return~~
    return m, b.
```

Multiple Regression

Algorithm

1. Input:
   A dataset with multiple independent variables $(x_1, x_2, \ldots x_n)$ and one dependent variable

2. Initialise:
   set coffecients $(b_0, b_1 \ldots b_2)$ to 0

3. Training:
   for each iteration:
   calculate the predicted value: $(b_0 + b_1 x_1 + b_2 x_2 + \ldots b_n x_n)$
   calculat error $= y - \hat{y}$
   update the coefficients:
   for each (j) from 0 to (n):
   $b_j = b_j + \alpha \cdot \frac{1}{m} \sum (\text{error} \cdot x_j)$

Pseudo code

```
function MR (X, y, alpha, iterations)
    Initialize b = [0, 0 ... 0]
    m = length(X)
    for i from 1 to iterations:
        y_pred = b[0] + sum(b[j] * X[j] for j in 1 to n)
        error = y - y_pred
        for j from 0 to n:
            b[j] = b[j] + alpha * (1/m) * sum(error * X[j])
    return b
```

Logistic Regression

1. Initialize:
   set all small random values to zeros.

2. Compute the weighted sum:

$$z = w_1 x_1 + w_2 x_2 + \dots w_n x_n + b$$

3. apply the sigmoid function:

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

4. Compute the loss

$$Loss = -\frac{1}{m} \sum_{i=1}^{m} \left[y_i \log(\hat{y}_i) + (1-y_i) \log(1-\hat{y}_i) \right]$$

5. Update weights.

$$w = w - \alpha \frac{\partial Loss}{\partial w}$$

$$b = b - \alpha \frac{\partial Loss}{\partial b}$$

Pseudocode

```
for iter in range(num_epoch):
    z = w * X + b
    y_pred = sigmoid (z)
    loss = - mean(y * log(y_pred))
    gradients:
    dw = mean((y_pred - y) * x)
    db = mean (y_pred - y)
```

· weights and bias:

$w = w - learning\_rate * dw$

$b = b - learning\_rate * db$

return final weights

24/3