Logistic Regression:

$$X = \underbrace{\begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix}}_{m = \#\text{ training examples}} \Big\}\, n_x \quad = \#R * \#G * \#B * \#\text{ channels}$$

$\Rightarrow$ We write X as $n_x \times m$ not $\underbrace{m \times n_x}$

In some notation

because this will be more convenient when we implement neural network.

$$z = w^T x + b \quad , \quad w \in \mathbb{R}^{n_x}, \; b \in \mathbb{R}$$

$$\hat{y} = \sigma(z) \quad , \quad \sigma(z) = \frac{1}{1+e^{-z}} \quad (\text{sigmoid function})$$

△ Note that we use w and b seperately.

No (like some notations : $z = \theta \cdot x$, $\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_{n_x} \end{bmatrix}_{n_x+1}$ ⇒ b

$$X = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \end{bmatrix}_{(n_x+1) \times m}$$

$$\underbrace{\mathcal{L}(\hat{y}, y)}_{\text{loss function}} = -(y \cdot \log \hat{y} + (1-y) \cdot \log(1-\hat{y}))$$

$$\underbrace{J(w, b)}_{\text{cost function}} = \frac{1}{m} \cdot \sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$$

$\Rightarrow$ The loss function computes the error for a single training example, while the cost function computes the average of the loss function of the entire set

dvar notation in Python code:

$$\text{dvar} \Rightarrow \frac{dJ \text{ (or other outputs you care about)}}{d\,\text{var}}$$

Whenever possible, avoid explicit for-loops.

$\Rightarrow$ Use built-in functions provided by numpy

will make use of vectorization technique, which saves you a lot of time.

Do not use "rank 1" array (e.g (5,)), this is actually an array, not a vector.

Instead, you should use

a = np.random.randn(5,1) ⇒ have a is (5,1)

⇒ Another tips: Try to use assert to help you determine the shape. assert(a.shape = (5,1))