Logistic Regression Report

Name: Rahul Govindkumar

SID: 801204373

Logistic Regression is a statistical technique. Here the input is a continuous variables and the output variable (dependent variable) is a binary variable.

Logistic Regression introduces the concept of the Log-Likelihood of the Bernoulli distribution, and covers a neat transformation called the sigmoid function.

Cross-entropy is commonly used in machine learning as a loss function.

$$\text{Cross Entropy Loss} = -\frac{1}{|D|}[\sum_{(y^i,\mathbf{x}^i)\in D} y^i \log p(y=1|\mathbf{x}^i;\mathbf{w},b) + (1-y^i)\log(1-p(y=1|\mathbf{x}^i;\mathbf{w},b))] + \frac{\lambda}{2}\|\mathbf{w}\|^2$$

It is a measure of the difference between two probability distributions for a given random variable or set of events.

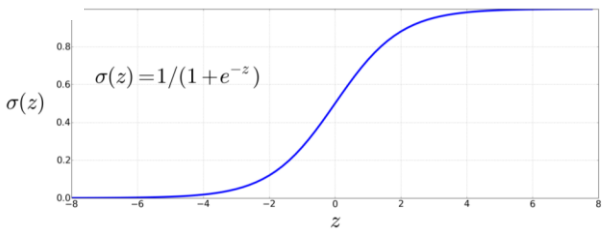
The goal of binary logistic regression is to train a classifier that can make a binary decision about the class of a new input observation. We have used the sigmoid classifier that will help us make this decision.

Consider a single input observation x, which we will represent by a vector of features $[x_1, x_2, ..., x_n]$

The classifier output y can be 1 or 0. Logistic regression solves the task by learning, from a training set, a vector of **weights** and a **bias term**.

$$z = \left(\sum_{i=1}^{n} w_i x_i\right) + b$$

The sigmoid function $\sigma(z) = \frac{1}{1+e^{-z}}$ takes a real value and maps it to the range [0,1]



$$x^{(i)} \text{ in input } [x^{(1)}, x^{(2)}, ..., x^{(m)}]$$
$$y^{(i)} = \sigma(\mathbf{w}\cdot\mathbf{x^{(i)}} + b)$$

$$P(y^{(1)}=1|x^{(1)}) = \sigma(\mathbf{w}\cdot\mathbf{x^{(1)}} + b)$$
$$P(y^{(2)}=1|x^{(2)}) = \sigma(\mathbf{w}\cdot\mathbf{x^{(2)}} + b)$$
$$P(y^{(3)}=1|x^{(3)}) = \sigma(\mathbf{w}\cdot\mathbf{x^{(3)}} + b)$$

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^{(1)} & \mathbf{x}_2^{(1)} & \dots & \mathbf{x}_f^{(1)} \\ \mathbf{x}_1^{(2)} & \mathbf{x}_2^{(2)} & \dots & \mathbf{x}_f^{(2)} \\ \mathbf{x}_1^{(3)} & \mathbf{x}_2^{(3)} & \dots & \mathbf{x}_f^{(3)} \\ \dots & & & \end{bmatrix}$$

$$\hat{y}^{(1)} = [\mathbf{x}_1^{(1)}, \mathbf{x}_2^{(1)}, ..., \mathbf{x}_f^{(1)}] \cdot [\mathbf{w}_1, \mathbf{w}_2, ..., \mathbf{w}_f] + b$$

Logistic Regression with Gradient Descent

The standard gradient descent algorithm is defined as follows where η is the learning rate.

$$\theta_t = \theta_{t-1} - \eta_t \triangle(\theta_{t-1})$$
$$\text{if } ||\theta_t - \theta_{t-1}|| < \epsilon \text{ then break;}$$

After the last iteration the above algorithm gives the best values of ϑ for which the function J is minimum.

$$L(\theta) = \prod_{i=1}^{n} p(y_i|x_i;\theta)$$

$$L(\theta) = \prod_{i=1}^{n} h_\theta(x_i)^{y_i}(1 - h_\theta(x_i))^{1-y_i}$$

$$\ell(\theta) = log\ L(\theta)$$

$$\ell(\theta) = \sum_{i=1}^{n} y_i \log(h_\theta(x_i)) + (1 - y_i) \log(1 - h_\theta(x_i))$$

The hypothesis function penalizes bad predictions by generating small values, so we want to maximize the log-likelihood.

Logistic Regression with Newton's Method

Cost fn of Logistic Regression.

$$J(\theta) = -\left[\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1-y^{(i)}) \log(1-h_\theta(x^{(i)}))\right]$$

$\min_\theta J(\theta)$

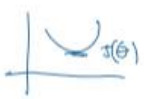
Derivative

$\frac{\partial}{\partial\theta} J(\theta) \overset{\text{Set}}{=} 0$

Find θ s.t

Use $\theta \in \mathbb{R}$ (rather than $\theta \in \mathbb{R}^{n+1}$ vector)

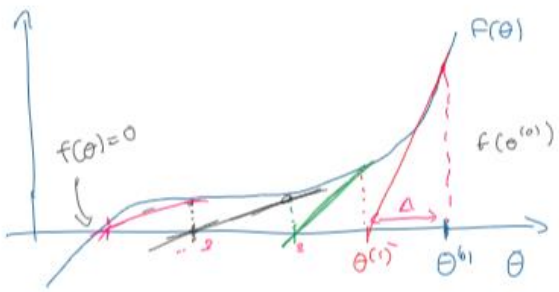
$\therefore f(\theta) = \frac{d}{d\theta} J(\theta)$



Goal: find θ s.t $f(\theta) = 0$

$\left[\because \frac{d}{d\theta}\theta^k \quad \frac{\partial}{\partial\theta}\theta^k \quad \theta \in \mathbb{R}^{n+1}\right]$

We have some fn f



Goal: θ s.t. $f(\theta) = 0 \left(f(\theta) = \frac{d}{d\theta} J(\theta)\right)$

$\theta^{(1)} = \theta^{(0)} - \Delta$

$\theta^{(0)} \to \theta^{(1)}$ $\quad \theta^{(1)} = \theta^{(0)} - \dfrac{f(\theta^{(0)})}{f'(\theta^{(0)})}$

$\vdots$

$\dfrac{d}{d\theta} f(\theta^{(0)}) = f'(\theta^{(0)})$

$= \dfrac{f(\theta^{(0)})}{\Delta}$ (height / width)

$\therefore \Delta = \dfrac{f(\theta^{(0)})}{f'(\theta^{(0)})}$

$\therefore \theta^{(t+1)} = \theta^{(t)} - \dfrac{f(\theta^{(t)})}{f'(\theta^{(t)})}$

We know

$f(\theta) = \dfrac{d}{d\theta} J(\theta) = J'(\theta)$

$$\therefore \theta^{(t+1)} = \theta^{(t)} - \frac{J'(\theta^{(t)})}{J''(\theta^{(t)})}$$

$J'(\theta) = \dfrac{d^2}{d\theta^2} J(\theta)$ $\quad \theta \in \mathbb{R}$

If $\theta \in \mathbb{R}^{n+1}$

divide 2nd derivative by 1st derivative

$\theta^{(t+1)} = \theta^{(t)} - \underbrace{H^{-1}}_{\mathbb{R}^{n+1 \times n+1}} \underbrace{\nabla_\theta J}_{\mathbb{R}^{n+1}}$

"Hessian"

$H \in \mathbb{R}^{n+1 \times n+1}$

$H_{ij} = \dfrac{d^2 J}{\partial\theta_i \partial\theta_j}$

$$\begin{bmatrix} H_{11} & H_{12} & \\ H_{21} & \cdots & \\ & & H_{ij} \end{bmatrix}$$

$$\nabla_\theta J = \begin{bmatrix} \frac{\partial}{\partial\theta_0} J(\theta) \\ \frac{\partial}{\partial\theta_1} J(\theta) \\ \vdots \\ \frac{\partial}{\partial\theta_n} J(\theta) \end{bmatrix} \in \mathbb{R}^{n+1}$$

$$\nabla_\theta J = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right) \cdot x^{(i)} \quad \in \mathbb{R}^{n+1}$$

$$H = \frac{1}{m} \sum_{i=1}^{m} \Big[\underbrace{h(x^{(i)})}_{\mathbb{R}} \underbrace{(1 - h(x^{(i)}))}_{\mathbb{R}} \cdot \underbrace{(x^{(i)})}_{\mathbb{R}^{n+1 \times 1}} \underbrace{(x^{(i)})^T}_{\mathbb{R}^{1 \times n+1}} \Big]$$

$[\ ][\ ]$

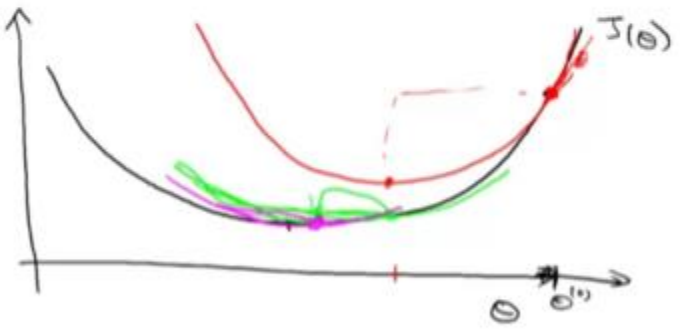
$[\ ]_{n+1 \times n+1}$

$$H_{ij} = \frac{\partial^2 J}{\partial \theta_i \partial \theta_j}$$

$$\theta = \theta - H^{-1}(\nabla_\theta J)$$

This is one iteration of Newton's method

- Newton's method converges very quickly



Newton method fits a quadratic fn that matches the fn

The Primary Differences

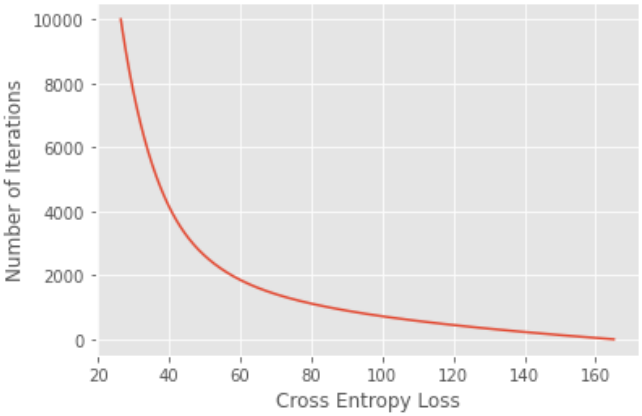
The two methods aren't equivalent, and as a general rule, we can't replace one with the other.

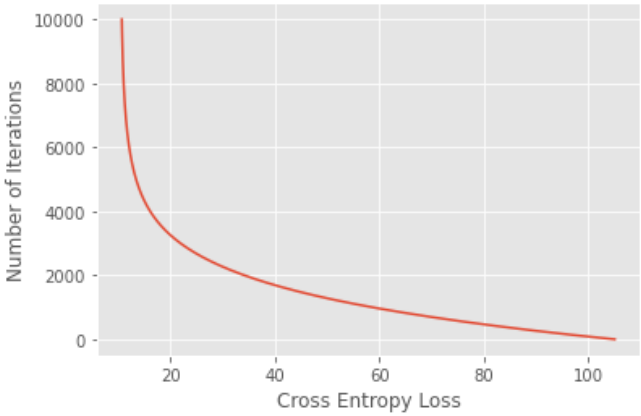
The first difference lies in the fact that gradient descent is parametric according to the learning rate α . Newton's method isn't parametric, which means that we can apply it without worrying for hyperparameter optimization. A parametric version of Newton's method also exists, in truth, but it only applies in cases for which we operate with a polynomial function with multiple roots.

The second difference has to do with the cost function on which we apply the algorithm.
Newton's method has stronger constraints in terms of the differentiability of the function than gradient descent.
If the second derivative of the function is undefined in the function's root, then we can apply gradient descent on it but not Newton's method.

The third difference consists of the behavior around stationary points.
If gradient descent encounters a stationary point during iteration, the program continues to run, albeit the parameters don't update.

Newton's method, however, requires to compute $\frac{f'(x)}{f''(x)}$ for $f'(x) = f''(x) = 0$. .
The program that runs it would therefore terminate with a division by zero error.





Gradient Descent

Newtons Method

We can see the Difference in the Curves from both the outputs.

In the Newtons Method the Loss function converges Quickly w.r.t to the Gradient Descent