Classification:

Ex
1. Is an email spam?
2. Is this transaction fraudulent?
3. Is this tumor malignant?

Answer is either yes or no → Binary Classification.

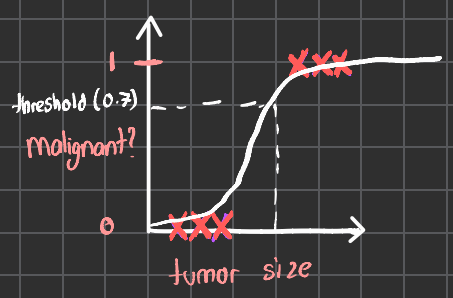
yes → 1 → positive class
no → 0 → negative class

Note: Class and category can be used interchangeably.

→ For some problems, linear regression may not be able to give a "best fit".

Logistic Regression: Algorithm used to give a better fit for classification problems.

Example

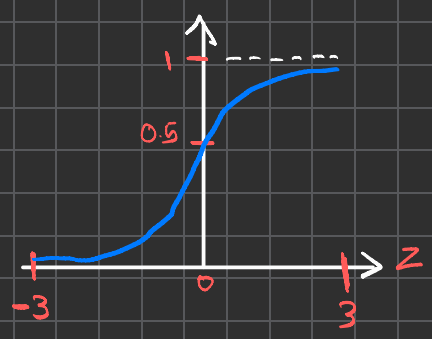


$f_{w,b}(x) = wx + b$

$\boxed{\vec{w} \cdot \vec{x} + b}$

$z \rightarrow \frac{1}{1+e^{-z}}$

Sigmoid Function:



- Sigmoid Function = Logistic Function.
- Outputs between 0 and 1.

Formula: $g(z) = \frac{1}{1+e^{-z}}$

$0 < g(z) < 1$

- $f_{\vec{w},b}(\vec{x}) = \boxed{\vec{w} \cdot \vec{x} + b}$

$\downarrow$

z

$\downarrow$

$$f_{\vec{w},b}(\vec{x}) = g(\underbrace{\vec{w} \cdot \vec{x} + b}_{z}) = \frac{1}{1+e^{-(\vec{w} \cdot \vec{x} + b)}}$$

" Logistic Regression "

Note: Essentially, with logistic regression, we use linear regression to output a value between 0 and 1, representing negative and positive.

Made with Goodnotes

Interpretation of LR output:

$$f_{\vec{w},b}(\vec{x}) = \frac{1}{1+e^{-(\vec{w}\cdot\vec{x}+b)}}$$

probability that class is 1.

Example

$x \rightarrow$ tumor size
$y \rightarrow$ 0 (non malignant)
$\rightarrow$ 1 (malignant)

$f_{\vec{w},b}(\vec{x}) = 0.7 \rightarrow$ 70% chance that y is 1.

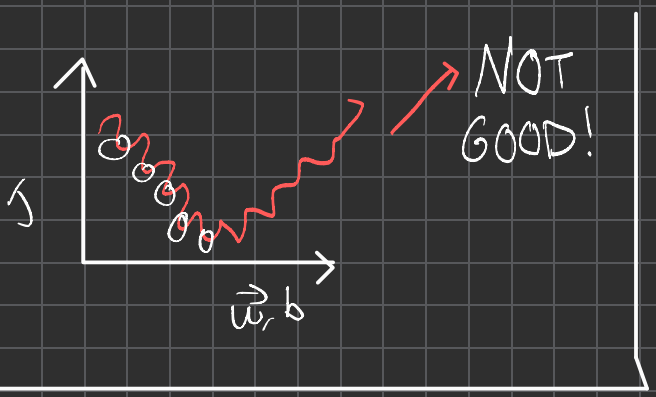
$$f_{\vec{w},b}(x) = P(y=1 \mid \vec{x}; \vec{w}, b)$$

$\rightarrow$ Probability of $y=1$, given input x, parameters w, b.

Cost Function For Logistic Regression:

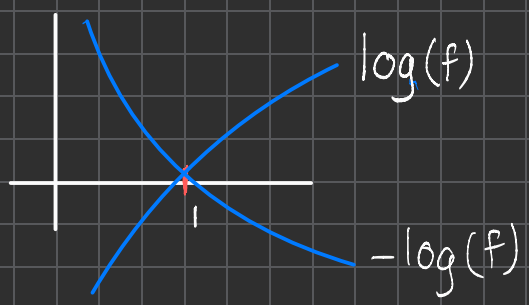
$\rightarrow$ Squared error cost function is not a good algorithm for logistic regression.

It leads to a non-convex function. (a lot of local minimas), which is not ideal.



Logistic Loss Function:

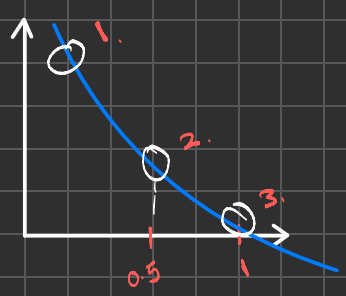
$$L(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)}) = \begin{cases} -\log(f_{\vec{w},b}(\vec{x}^{(i)})) & \text{if } y^{(i)} = 1 \\ -\log(1 - f_{\vec{w},b}(\vec{x}^{(i)})) & \text{if } y^{(i)} = 0 \end{cases}$$



Note: Since logistic regression can only between $0 < f < 1$
We can ignore the graph that exceeds it.

Ex Let's take a closer look at $\log(f)$ if $y^{(i)} = 1$

$$J(\vec{w}, b) = \frac{1}{m} \sum \underbrace{L(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)})}_{\text{loss}}$$



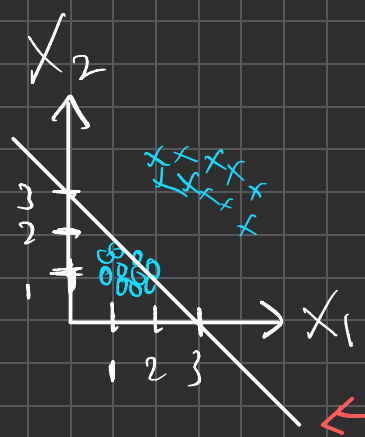
1. = A LOT of loss. ☹
2. = A little loss. 😐
3. = Almost no loss. ☺

Made with Goodnotes

Decision Boundary:

$$f_{\vec{w},b}(\vec{x}) = g(z) = g(w_1x_1 + w_2x_2 + b)$$

Hypothetically: Let's say our best fit is $w_1 = 1$, $w_2 = 1$, $b = -3$



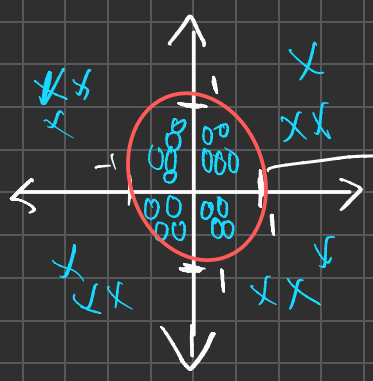
Decision Boundary: $z = \vec{w}.\vec{x} + b = 0$

$z = x_1 + x_2 - 3 = 0$

$x_1 + x_2 = 3$

Non-Linear Decision Boundary:

Decision Boundary: $f_{w,b}(x) = g(z) = g(\underset{1}{x_1^2} + \underset{1}{x_2^2} \underset{-1}{+b})$



$z = x_1^2 + x_2^2 - 1 = 0$

$x_1^2 + x_2^2 = 1$

Made with Goodnotes

Simplified Cost Function:

$$L(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)}) = -y^{(i)}\log(f_{w,b}(x^{(i)})) - (1-y^{(i)})\log(1-f_{w,b}(\vec{x}^{(i)}))$$ LOSS

$$J(w,b) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log(f_{w,b}(x^{(i)})) + (1-y^{(i)})\log(1-f_{w,b}(x^{(i)}))\right]$$ COST

Gradient Descent for Logistic Regression:

Find $\vec{w}, b$

$$J(w,b) = -\frac{1}{m}\sum\left[y^{(i)}\log(f_{w,b}(x^{(i)})) + (1-y^{(i)})\log(1-f_{w,b}(x^{(i)}))\right]$$

repeat {

$$w_j = w_j - \alpha\frac{d}{dw_j}J(\vec{w},b) \qquad \frac{d}{dw_j}J(\vec{w},b) = \frac{1}{m}\sum_{i=1}^{m}(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)})x_j^{(i)}$$

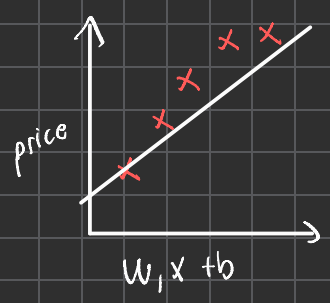
$$b = b - \alpha\frac{d}{db}J(\vec{w},b) \qquad \frac{d}{db}J(\vec{w},b) = \frac{1}{m}\sum_{i=1}^{m}(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)})$$

}

Linear Regression $= f_{\vec{w},b}(\vec{x}) = \vec{w}\cdot\vec{x} + b$

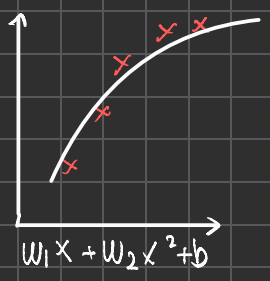
Logistic Regression $= f_{\vec{w},b}(\vec{x}) = \dfrac{1}{1+e^{-(\vec{w}\cdot\vec{x}+b)}}$

Overfitting:



→ Does not fit training set well

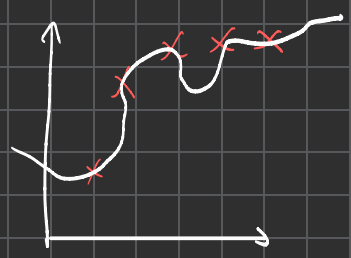
Underfit / High Bias



→ Fits pretty well.

Generalization

Best one!!!



→ Fits "too" well

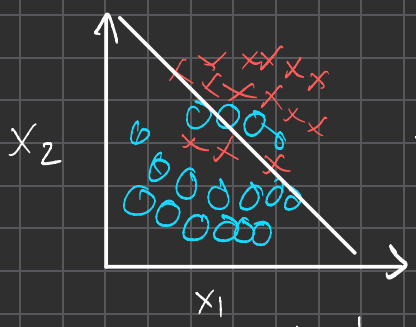
Overfit.

High Variance

Note: A slight change might cause trouble.

Made with Goodnotes

Underfitting / Overfitting in Classification:

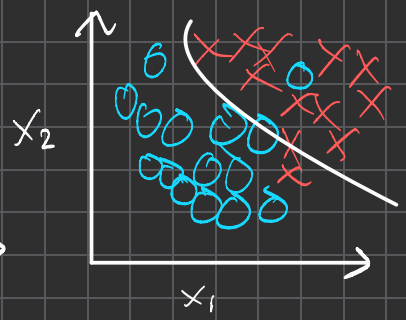


$z = w_1 x_1 + w_2 x_2 + b$

$f_{\vec{w},b}(\vec{x}) = g(z)$

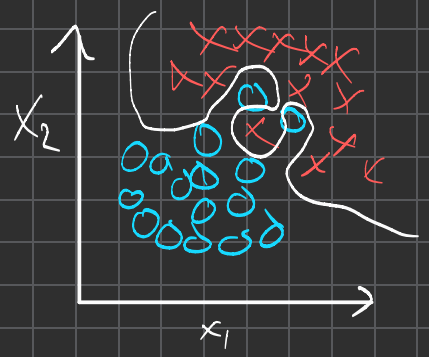
g is a Sigmoid function

Underfit / High Bias



$z = w_1 x_1 + w_2 x_2 + w_3 x_1^2 + w_4 x_2^2 + w_5 x_1 x_2 + b$

Generalization



$z =$ (A lot of complex polynomials)

Overfitting / High Variance

Note: In conclusion, even though overfitting fits the data "too" well. It has a hard time predicting outcomes correctly for new examples.

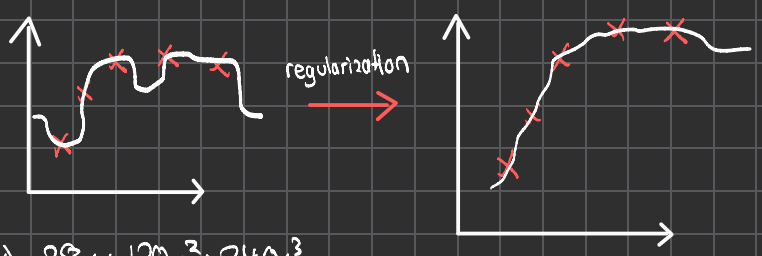
Addressing Overfitting:

- Collect more training examples. (if possible)
- Reduce the amount of features. = Feature Selection.

Disadvantage = Useful features may get lost!

A lot of Features	+	Insufficient Data	→	Overfitting.

- Regularization: Reduce the size of parameters instead of eliminating them entirely.



$f(x) = 28x + 120x^2 + 240x^3 \ldots$

$f(x) = 13x - 0.23x^2 + 0.000014x^3 + 0.00001x^4 \ldots$

Made with Goodnotes

Regularization to Reduce Overfitting:

→ Having smaller values to parameters $(w_1, w_2 \ldots w_n)$ leads to a simpler model.

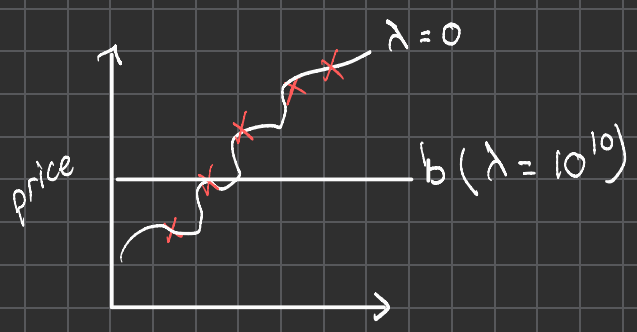
$$J(\vec{w}, b) = \underbrace{\frac{1}{2m}\sum_{i=1}^{m}\left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)^2}_{\text{Mean Squared Error (Fit data)}} + \underbrace{\frac{\lambda}{2m}\sum_{j=1}^{n} w_j^2}_{\text{Regularization (Keep } w_j \text{ small)}}$$

Lambda "Regularization Parameter"

$\lambda > 0$

λ balances both goals

Example



→ High λ
→ Parameters w too small

→ Low λ
→ Doesn't effect w.

Made with Goodnotes

Regularized Linear Regression:

$$\min J(w,b) = \min_{\vec{w},b} \left\{ \frac{1}{2m} \sum_{i=1}^{m} \left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)} \right)^2 + \frac{\lambda}{2m} \sum_{j=1}^{n} w_j^2 \right.$$

Gradient Descent:

repeat {

$$w_j = w_j - \alpha \boxed{\frac{d}{dw_j} J(w,b)} \longrightarrow \frac{1}{m} \sum_{i=1}^{m} \left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)} \right) x_j^{(i)} + \frac{\lambda}{m} w_j$$

$$b = b - \alpha \boxed{\frac{d}{db} J(w,b)} \longrightarrow \frac{1}{m} \sum_{i=1}^{m} \left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)} \right)$$

}

$$w_j = \underbrace{w_j \left(1 - \alpha \frac{\lambda}{m}\right)}_{\text{Shrink } w_j} \underbrace{- \alpha \frac{1}{m} \sum_{i=1}^{m} \left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)} \right) x_j^{(i)}}_{\text{Usual update}}$$

Ex $\alpha \frac{\lambda}{m} \rightarrow 0.01 \frac{1}{50} = 0.002$

$\underbrace{w_j (1 - 0.002)}_{0.9998}$ → decrease w_j a little bit on every iteration.

Made with Goodnotes