Page No.
Date : / /

Gradient descent Algo:

$$w = w - \alpha \frac{\partial}{\partial w} J(w,b)$$

update the parameter w by subtracting with the previous value of w with $\alpha \cdot \frac{\partial}{\partial w}$ of cost function $J(w,b)$

α = the learning rate (small positive number between 0 and 1)

$\alpha \uparrow$ indicates that the descent is high.
$\alpha \downarrow$ indicates that the descent is with baby steps (low)

$\frac{\partial}{\partial w} J(w,b)$ = derivative term of cost function J.

$$b = b - \alpha \frac{\partial}{\partial b} J(w,b)$$

continue these till there is no change.. ie till it reaches local minima

* Simultaneously update w and b.

Correct : Simultaneous way:

$$temp\_w = w - \alpha \frac{\partial}{\partial w} J(w,b)$$

$$temp\_b = b - \alpha \frac{\partial}{\partial b} J(w,b)$$

after this block get executed then store in w & b (update)

⚠ Do not update immediately

$w = temp\_w$
$b = temp\_b$

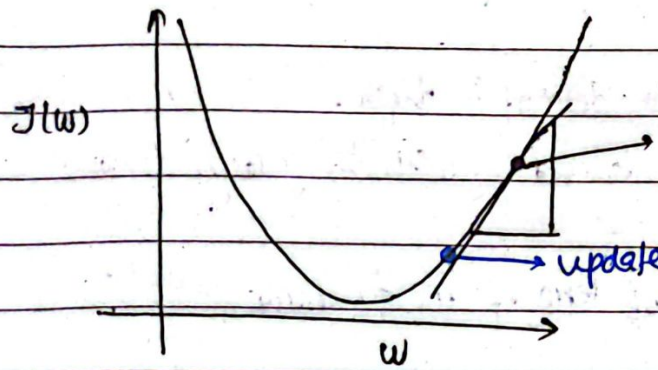
Page No.
Date : / /

Now,

Now for a cost function J of one parameter $J(w)$

$J(w)$

$$w = w - \alpha \cdot \frac{\partial}{\partial w} J(w)$$



Now,

$\frac{\partial}{\partial w} J(w)$ = a positive number

slope of tangent = +ve

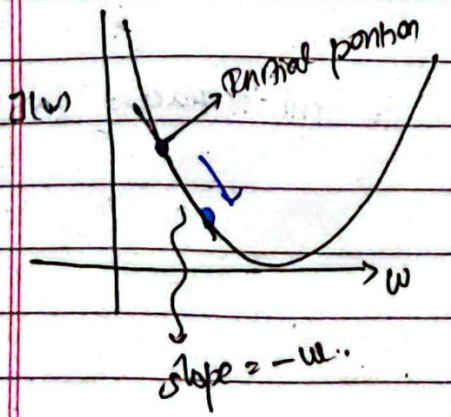
Similarly;

So,

$w = w - \alpha \cdot$ positive number

ie w decreases.

and updated w shifts towards left.



$w = w - \alpha \cdot (-ve \text{ slope})$

$= w + \alpha \cdot$ number.

$= w \uparrow$ and shifts towards right

Page No.
Date : / /

Learning Rate (α):

If $\alpha \rightarrow$ too small

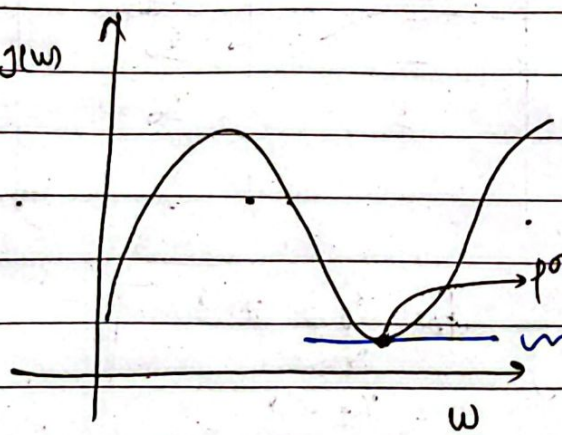
gradient descent may be slow.

If $\alpha \rightarrow$ too large

may take too huge step and cross the minimum which leads to never reaching minimum and may even diverge.

↳ gradient descent may overshoot.

What if the cost function is already at 'a local minimum'. ?



So, $w = w - \alpha \cdot \boxed{\frac{d}{dw} J(w)} = 0$

$\boxed{w \leftarrow w}$

Therefore no change ☺

Page No.
Date : / /

Multiple linear regression :

With multiple variables :

Example :

Size in feet^2	No of bedrooms	No of floors	Age of home	Price in $1000's
x_1	x_2	x_3	x_4	
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	200
852	2	1	36	150

x_j = j^{th} feature

n = number of features

} for $x_1 \Rightarrow n = 4$, x_2, x_3, x_4

$\vec{x}^{(i)}$ = features of i^{th} training example

$\vec{x}^{(2)}$

$x_j^{(i)} \rightarrow x_3^{(2)} = 2$

Model :

previously for single feature :

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

Page No.
Date : / /

Now, for multiple features:

$$f_{w,b}(x) = w_1 x_1 + w_2 x_2 + \dots + w_n x_n + b$$

$\vec{w} = [w_1 \; w_2 \; w_3 \dots w_n]$

b = a number

$\vec{x} = [x_1 \; x_2 \; x_3 \dots x_n]$

So, can be rewritten as:

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

$\vec{w} \cdot \vec{x} = w_1 x_1 + w_2 x_2 + w_3 x_3 \dots + w_n x_n$

dot product

aka: multiple linear regression.

(Note: Not a multivariate regression)

Vectorization:

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

$$= \sum_{j=1}^{n} w_j x_j + b$$

```
f = 0
for j in range (0, n):
    f = f + w[j] * x[j]
f = f + b
```

Code quality 😐 okay!

Page No.
Date : / /

But when, vectorization is used;

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

↳ import numpy as np

> f = np.dot(w, x) + b

☺ v. good!

Why vectorized codes run so fast?

Without vectorization	Vectorization:
> for j in range (0, 16): f = f + w[j] * x[j]	> np.dot(w, x)
t_0 f + w[0] * x[0]	to : w[0] w[1] ... w[15]
t_1 f + w[1] * x[1]	t_1 : x[0] x[1] ... x[15]
⋮	multiply in parallel ... and then with specialized hardware add together.
t_{15} f + w[15] * x[15]	w[0]x[0] + w[1]*x[1] + ... + w[15] x[15]
↓	↓
Cycles = 16	Cycles used = 2

Gradient Descent for multiple linear regression

Page No.
Date: / /

	prev. notn	Vector Notn
parameters :	$w_1, w_2 \cdots w_n$ b	$\vec{w} = [w_1, \ldots w_n]$
Model :	$f_{\vec{w},b}(\vec{x}) = w_1x_1 + \cdots + w_nx_n + b$	$f_{\vec{w},b}(\vec{x}) = \vec{w} \cdot \vec{x} + b$
cost fun :	$J(w_1, \ldots, w_n, b)$	$J(\vec{w}, b)$

gradient descent:

repeat {	repeat {
$w_j = w_j - \alpha \frac{\partial}{\partial w} J(w_1, \ldots w_n, b)$	$w_j = w_j - \alpha \frac{\partial J(\vec{w},b)}{\partial w}$
$b = b - \alpha \cdot \frac{\partial}{\partial b} J(w_1, \cdots w_n, b)$	$b = b - \alpha \frac{\partial}{\partial b} J(\vec{w},b)$
}	}
$\frac{\partial}{\partial w} J(w,b) = \frac{1}{m} \sum_{i=1}^{m} (f_{w,b}(x^{(i)}) - y^{(i)}) x^{(i)}$	$\frac{\partial}{\partial w} J(\vec{w},b) = \frac{1}{m} \sum_{i=1}^{m} (f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}) x_1^{(i)}$

$\Rightarrow$

$$\begin{bmatrix} x_0^0 & x_1^0 & \cdots & x_{n-1}^0 \\ \vdots & \vdots & & \vdots \\ x_0^{m-1} & x_1^{m-1} & & x_{n-1}^{m-1} \end{bmatrix}$$

Practical tips for Linear Regression

Page No.
Date: / /

* Feature scaling
  ↳ Enables gradient descent to run faster.

Feature & parameter values:

$\hat{price} = w_1 x_1 + w_2 x_2 + b$

w_1 → size

w_2 → bedrooms

x_1 : size (feet²) — range: 300 – 2000 (large)

x_2 : # bedrooms — range: 0–5 (small)

Now,

Let's take an example case where:

House : $x_1 = 2000$; $x_2 = 5$, price = $500k

So, what would be the parameters w_1 and w_2?

Suppose 50 and 0.1 are the numbers but we are not sure which one of them is w_1 and which is w_2

So, there are 2 cases

Case I

$w_1 = 50$; $w_2 = 0.1$; $b = 50$

$\hat{price} = 50 \times 2000 + 0.1 \times 5 + 50$

$= \$100,050.5k$

which is way higher

Case II :

$w_1 = 0.1$; $w_2 = 50$; $b = 50$

$\hat{price} = 50 \times 0.1 + 50 \times 5 + 50$

$= \$500k$

and is more reasonable case.

So, the conclusion :

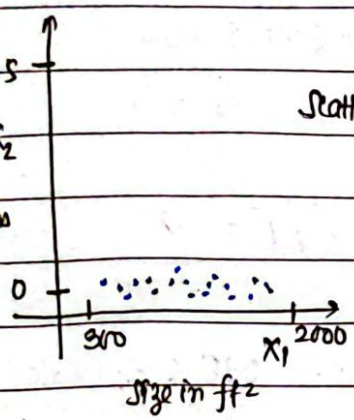
x_1 had large range. So, the good model should choose w_1 with less value

Likewise

x_2 has smaller range so, the good model should choose w_2 with greater value.

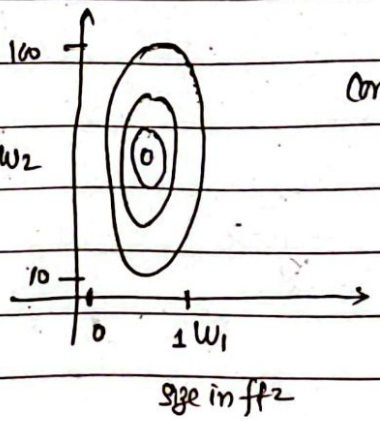
Page No.
Date: / /

Features



Scatter plot

Parameters

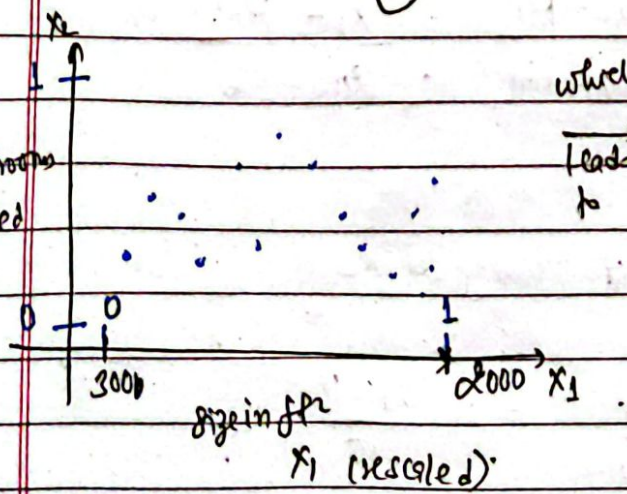


Contour plot

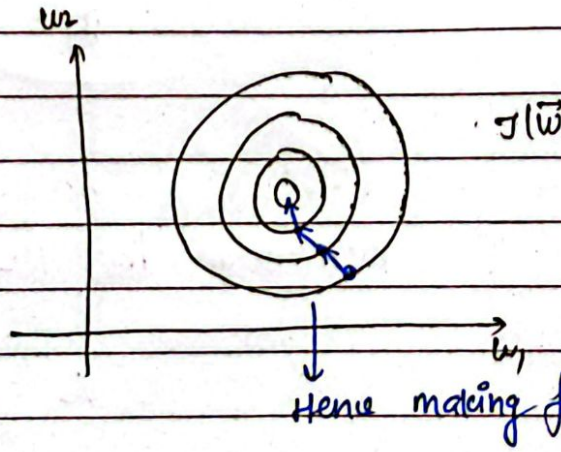
$J(\vec{w}, b)$

Now, if we were to run the gradient descent with the training data as it is, since the contour plot is so tall and skinny, gradient descent may end up bouncing back and forth and may take longer to reach the center.

In order to solve this, we simply scale the features. Leading to:



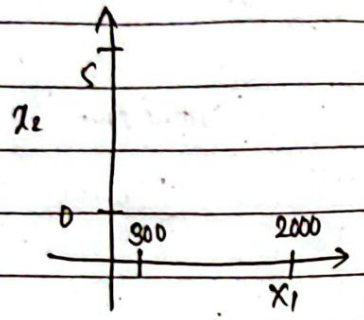
which leads to



Page No.
Date : / /

Q. How do you actually scale features?

— Method 1



Here, $800 \leq x_1 \leq 2000$

and $0 \leq x_2 \leq 5$

So, $x_1, \text{scaled} = \frac{x_1}{\max} = \frac{x_1}{2000}$

range: $0.15 < x_1, \text{scaled} < 1$

and

$x_2, \text{scaled} = \frac{x_2}{\max} = \frac{x_2}{5}$

range: $0 < x_2, \text{scaled} < 1$

— Method 2

We can also carry out mean normalization.

① First find out the average (μ_1) of traning set x_1

then,

$$x_1 = \frac{x_1 - \mu_1}{\max - \min} = \frac{x_1 - \mu_1}{2000 - 300}$$

② then same for μ_2 of traning set x_2

$$x_2 = \frac{x_2 - \mu_2}{\max - \min} = \frac{x_2 - \mu_2}{5}$$

The values range from -ve to +ve.

Page No.
Date: / /

- Method 3.

Z-score normalization

Find out (σ) of the data. (standard deviation).
and μ_1.

Then,

$$x_1 = \frac{x_1 - \mu_1}{\sigma_1}$$

Note: aim for about $-1 \le x_j \le 1$
$-3 \le x_j \le 3$
$-0.3 \le x_j \le 0.3$
} for each feature acceptable ranges

If $0 \le x_1 < 3 \rightarrow$ okay no need to rescale.

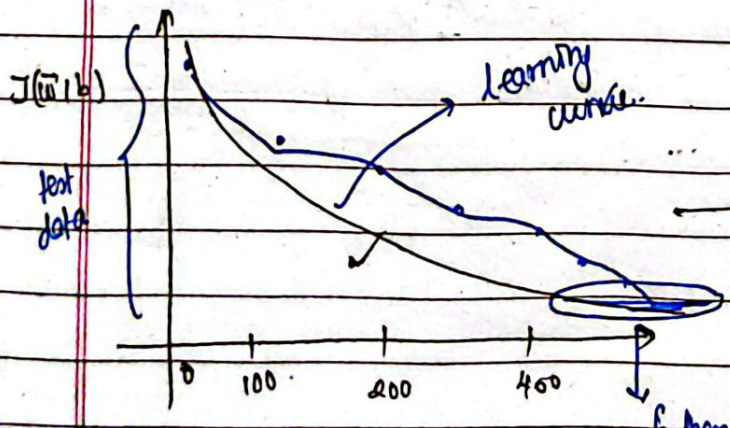
$-2 \le x_2 \le 0.5 \rightarrow$ okay no need to rescale

$-100 \le x_3 \le 100 \rightarrow$ too large $\rightarrow$ rescale

$-0.0001 \le x_4 \le 0.0001 \rightarrow$ too small $\rightarrow$ rescale

Checking Gradient Descent for Convergence

Objective: $\min_{\vec{w}, b} J(\vec{w}, b)$



→ if the graph is decreasing; the grad descent is working.

ie the cost is decreasing with increase in iterations.

∴ so, we can say it converged.

Page No.
Date: / /

Similarly,

Another way of testing gradient descent:-

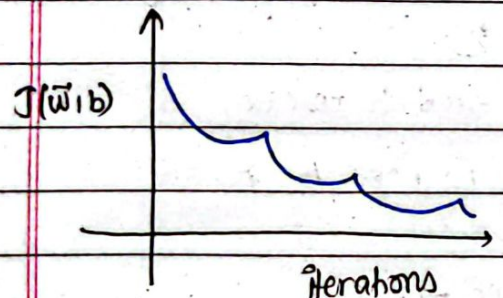
Automatic convergence test

Let "ε" {epsilon} be 10^{-3}.

If $J(\vec{w}, b)$ decreases by $\leq \varepsilon$ in one iteration, declare convergence

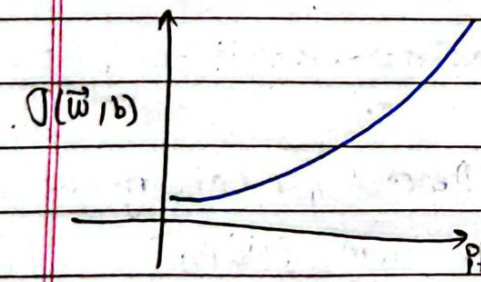
(But graphs could be better in use case)

Choosing the Learning Rate:



→ Conclusion: Either there is:

1. Bug in the code
2. the α is too high causing to overshoot.



→ α very very high.

or.

If bugs like

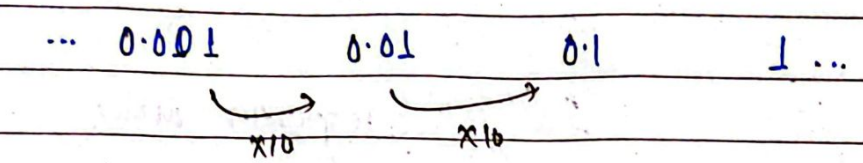
$w = w + \alpha d_1$ (should be '-')

exist

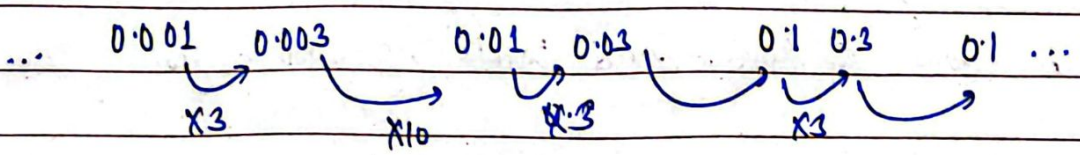
Page No.
Date : / /

Also, if α is too small, it takes too much of time.

Values of 'α' to try:



* for further tuning: once the rate is found near optimal:



and so on :)

~~Polynomial Regression~~

Polynomial Regression

→ It is a type of linear regression, but it differs in the way the relationship between independent variable and dependent variable is modeled.

- Model representation:

  Linear = r/ship between independent and dependent variable modeled as linear equation.
  - assumes straight line relationship

  polynomial = r/ship is modeled as n^{th} degree polynomial.

- Eq^n:

  Linear = $y = \beta_0 + \beta_1 x$
  (y: dependent, x: independent)

  polynomial = $\beta_0 + \beta_1 x + \beta_2 x^2 - \dots + \beta_n x^n$.

Page No.
Date: / /

Therefore, polynomial regression is the more flexible extension of linear regression that can capture non-linear relationships between variables.

Logistic regression Model.

— Single most likely used classification algorithm in the world.

Use cases of linear and logistic regression algo:

(1) Nature of output:
- Linear = Used when the output variable is continuous
- Logistic = Used when the output variable is binary or categorical.

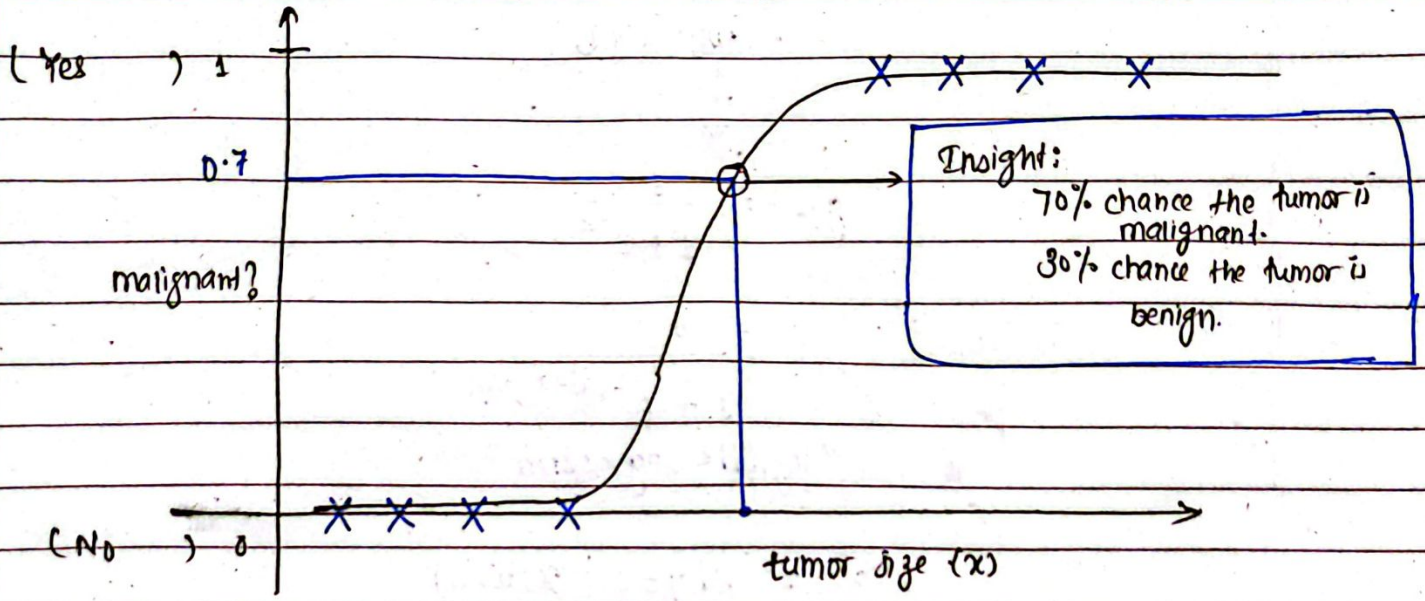
(2) · Model Representation:
- Linear = output is linear combination of input features fun^n to
- Logistic = Uses logistic model to probability that given input belongs to certain class.
  the output is transformed to fall between 0 and 1.

Example:

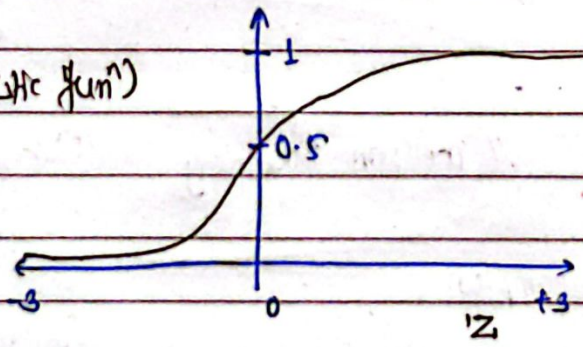
→

Page No.
Date : / /

Graph of classifying if the tumor is malignant.



Sigmoid fun (Logistic fun)



output between: 0 and 1

$$g(z) = \frac{1}{1+e^{-z}} \qquad (0 < g(z) < 1)$$

if z = +ve large number:

e^{-z} → negligible value which tends the curve to approach towards '1'

if z = -ve large number.

$e^{-z} \Rightarrow e^{-(-z)}$ or e^{+z}

and $\frac{1}{\text{large number}}$ → tends to zero

Likewise for z = 0:

$$g(z) = \frac{1}{1+1} = 1/2 = 0.5 \checkmark$$

Page No.
Date : / /

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

$$z = \vec{w} \cdot \vec{x} + b$$

$$\downarrow$$

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

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

"logistic regression"

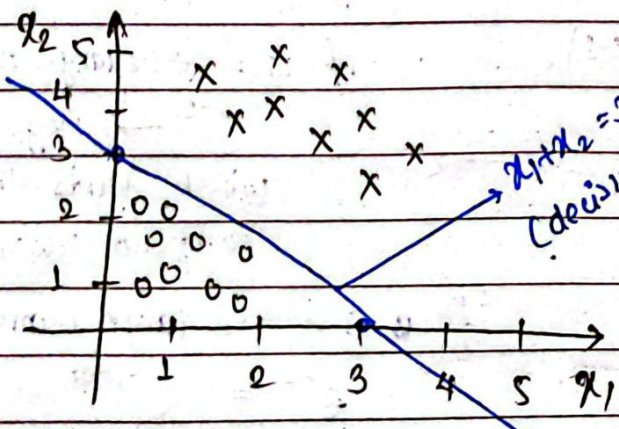
$$= P(y=1 \mid x; \vec{w}, b)$$

read as: probability of y equals to 1 given x with parameters w and b.

Decision Boundary (learnt previously in SVM)

Starting with examples:

Consider we have 2 ~~parameters~~ features 'x1' and 'x2'



X ⇒ y = 1 (+ve)
O ⇒ y = 0 (-ve)

The feature vector fun?

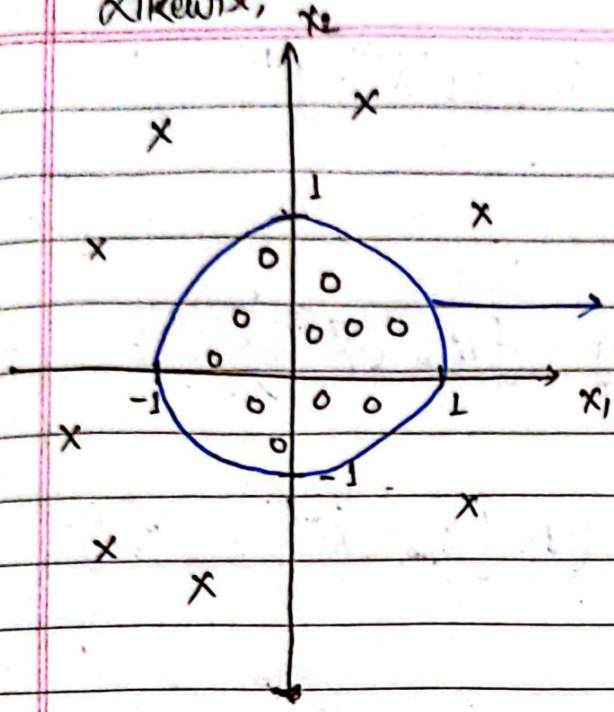
$$f_{\vec{w},b}(\vec{x}) = g(z)$$

$$= g(w_1x_1 + w_2x_2 + b)$$

assuming ↓ ↓ -3

Then, $x_1 + x_2 = 3$ will be the linear eqn aka. decision boundary

Likewise,



The fun^n : $f_{\vec{w},b}(\vec{x}) = g(z)$

$= g(wx_1^2 + wx_2^2 + b)$

Assume : 1 1 -1

Then,

$x_1^2 + x_2^2 = 1$

$x_1^2 + x_2^2 = 1$ decision boundary.

when

$x_1^2 + x_2^2 \geq 1$

$\hat{y} = 1$

and

$x_1^2 + x_2^2 < 1$

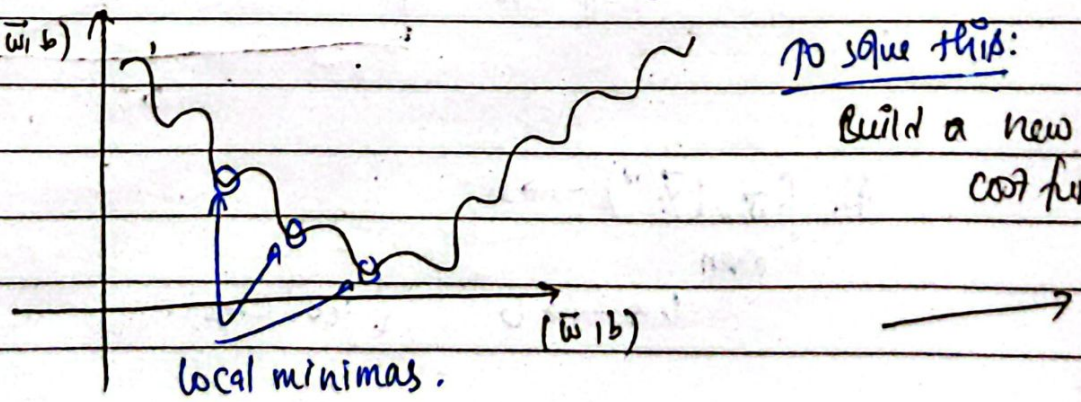
$\hat{y} = 0$

Cost fun^n for logistic regression:

→ for logistic eq^n the squared error cost fun^n ie.

$$J(\vec{w}, b) = \frac{1}{m} \sum_{i=1}^{m} \frac{1}{2} \left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)} \right)^2$$

is not an appropriate function. If used the graph becomes like.



To solve this:

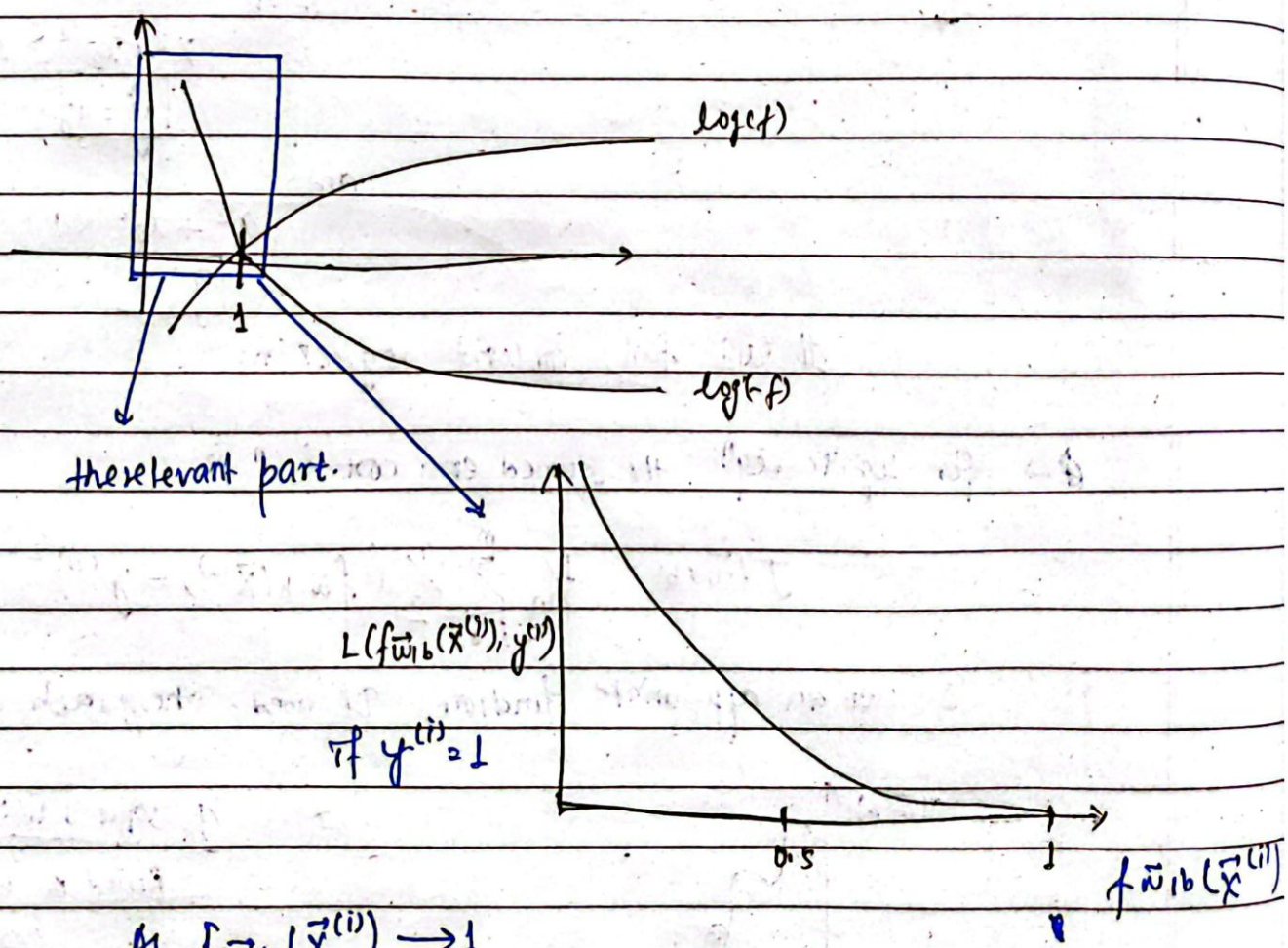
Build a new cost fun^n

Page No.
Date: / /

we will use loss fun^n: $L(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)})$

↳ Logistics loss function:

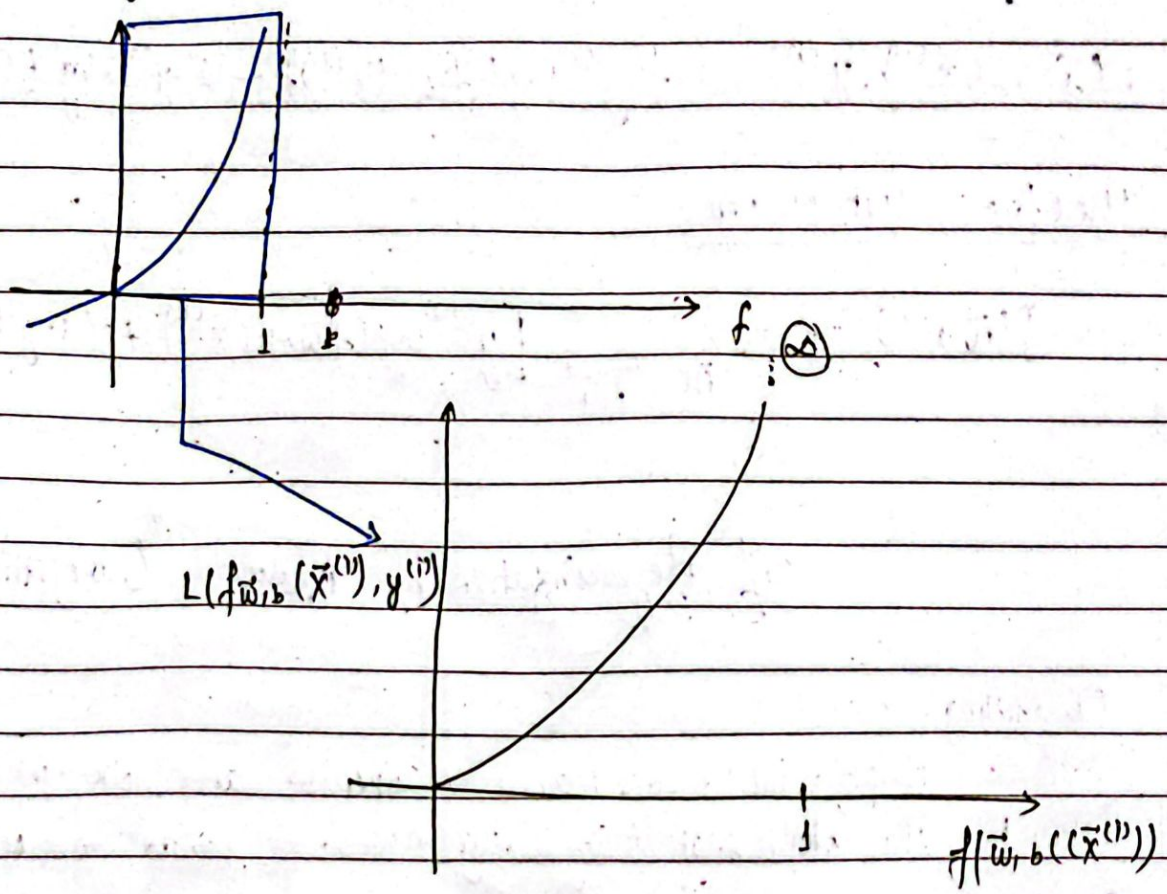
$$L(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)}) = \begin{cases} -\log\left(f_{\vec{w},b}(\vec{x}^{(i)})\right) & \text{if } y^{(i)}=1 \\ -\log\left(1-f_{\vec{w},b}(\vec{x}^{(i)})\right) & \text{if } y^{(i)}=0 \end{cases}$$



As $f_{\vec{w},b}(\vec{x}^{(i)}) \to 1$
then
loss $\to 0$ (good case).

Page No.
Date : / /

if $y^{(i)} = 0$:



As $f(\vec{w},b(\vec{x}^{(i)})) \longrightarrow 1$

loss $\rightarrow \infty$ (worst case)

$$J(\vec{w},b) = \frac{1}{m} \sum_{i=1}^{m} L(f_{w,b}(\vec{x}^{(i)}, y^{(i)})$$

Simplified loss function:

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

Therefore: cost function

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

Regularization to Reduce "Overfitting"

Overfitting:

When ML model learns the details and noise in the training data to the extent that it negatively impacts the performance of the model on new, unseen data.

↓

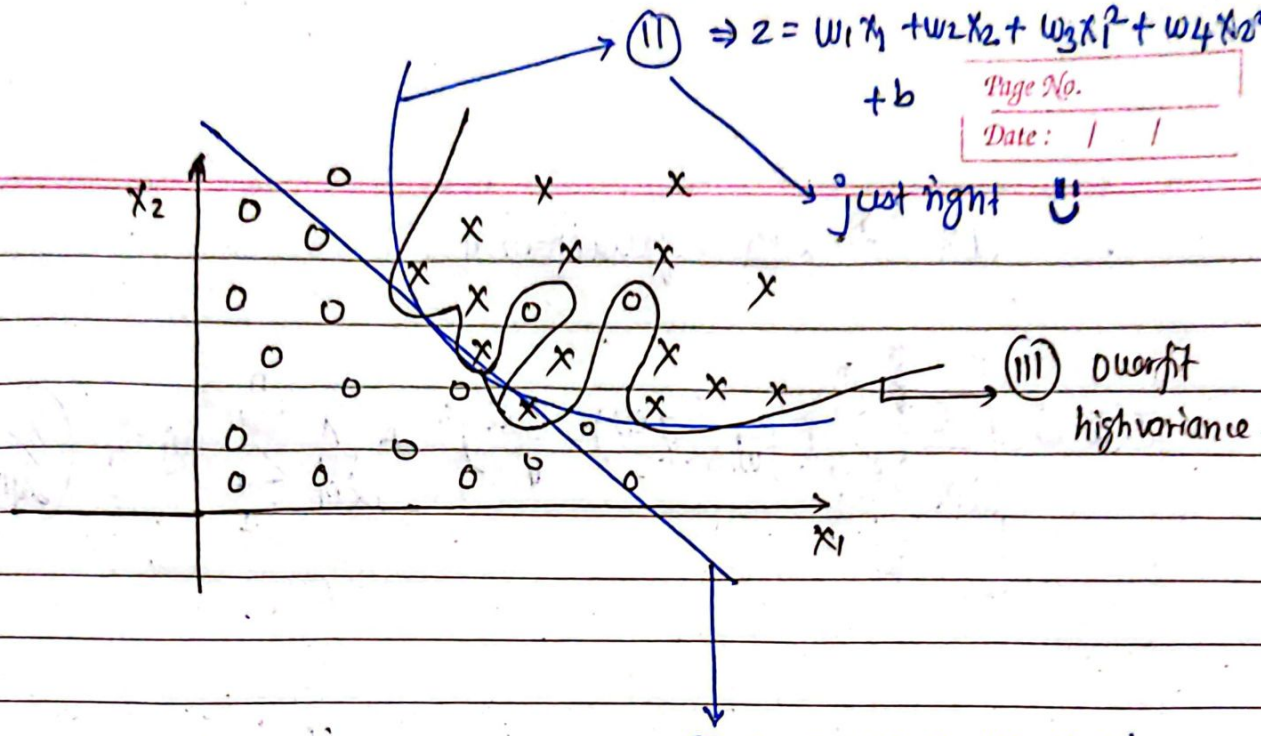
Can perform well on trained data but fails to "generalize" to new data.

Note:

Underfit data ⇒ high bias

overfit data ⇒ high variance

→



(I) ⇒ $z = w_1 x_1 + w_2 x_2 + b$

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

underfit : high bias

Addressing overfitting:

(1) Collect more training examples. (when abundant data available)

(2) Select features to include/exclude.

many features and less training data → leads to overfitting

So, go for "feature selection"

(3) Regularization:

· Reduce the size of parameters w_j.

ie

$$f(x) = 28x - 385x^2 + 39x^3 - 174x^4 + 100$$

$$= 13x - 0.23x^2 + 0.000014x^3 - 0.001x^4 + 10 \quad \text{(Regularized)}$$

Example Ca

Page No.
Date: / /

Cost fun^n with regularization:

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

regularization term.

where,
λ = regularization parameter

Can include or exclude 'b'.

- If $\lambda = 0$: model overfits
- If λ = very large : model underfits.

Regularized linear regression:

Gradient descent:

repeat {

$$w_j = w_j - \alpha \frac{\partial}{\partial w_j} \left(J(\vec{w}, b) \right)$$

$$b = b - \alpha \frac{\partial}{\partial b} J(\vec{w}, b)$$

}

Page No.
Date : / /

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

new term.
regularized

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

↓

don't have to regularize 'b' ☺

$$w_j = w_j - \alpha \left[\frac{1}{m} \sum_{i=1}^{m} \left(f_{\vec{w},b}(x^{(i)}) - y^{(i)} \right) x_j^{(i)} + \frac{\lambda}{m} w_j \right] \quad \text{for } j = 1 \dots n$$

Rearranged to be:

$$w_j = w_j \underbrace{\left(1 - \alpha \frac{\lambda}{m} \right)}_{\text{New part}} - \alpha \underbrace{\frac{1}{m} \sum_{i=1}^{m} \left(f_{\vec{w},b}(x^{(i)}) - y^{(i)} \right) x_j^{(i)}}_{\text{original part.}} \quad \text{for } j = 1 \dots n$$

↓ This decreases (new part) the value of 'w_j' each iteration by little bit.

Regularized Logistic regression

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

↓

and when applied gradient descent to this cost fun^n, it looks exactly like that of linear regression.

repeat {

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

$$b = b - \alpha \frac{\partial}{\partial b} J(\vec{w},b) \quad \rightarrow \quad \text{No change since regularization only applied on } w \text{ not } b \text{ !!}$$

$$\frac{1}{m}\sum_{i=1}^{m}\left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right).$$