- **Machine learning** : field of study that gives computers ability to learn.
  → Grew out of work in AI. We wanted to build machines which can work as humans. But finding an algorithm that works perfectly is very difficult, so we give the task of learning to machines itself, thus emerged Machine learning from AI.

- A computer program is said to learn from experience E with respect to some Task T and some performance measure P if its performance on T measured by P improves with experience E.
  → Eg: An email program watches which email we do or do not mark as spam. here
  ☹ T is classifying email as spam or not spam
  ☹ E is program watching us label emails
  ☹ P is number of mails correctly labeled as spam/not spam by program
  → Eg: Chess engine learning to play chess
  ☹ T is the task of playing chess
  ☹ E is the experience computer gets from playing chess
  ☹ P is the probability of computer winning next game.

- **Types of learning :→**
  ⓐ Supervised learning
  ⓑ unsupervised learning

ⓐ **Supervised learning :→** (learns from 'right answers')
→ here right answers are already given
→ i.e we feed a dataset to the computer and according to it we find the answer
→ for eg :→ selling a 750 sq feet house.
> collect dataset i.e sq feet and selling value of houses
> plot these on graph
> choose the best fit { this is an important & tricky task }
> according to the fit find value of house.

here we already had dataset i.e correct answer

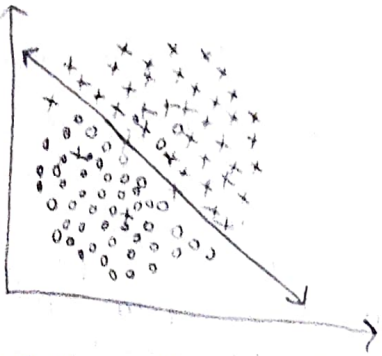
→ This type of problem can also be called **Regression problem** where we predict continuous valued outputs (price)

→ eg 2: finding out if Breast cancer is malignant or benign i.e serious or normal.
> This type of problem is different from above Regression Problem
> Here there are only 2 possibilities unlike Regression where we have a continuous range of possibilities
(Note) > number of possibilities being 2 is not necessary, it can be 3,4 etc. The only criteria is that possibilities has to be discrete.
> Here we use multiple features like Tumor size and age

> Based on these features plot a graph.

> now we use our machine learning algorithm to create a line that partitions between these plot.

> now we find where does our points lie in this grap. Based on this we predict type of Breast cancer.



here O ⇒ benine cancer datapoint
X ⇒ malignant cancer datapoint

- we plot both of these on graph
- create the line with algorithm
- then plot our point on graph & if it is below then cancer is benine else it is malignant

> In this case we used only 2 features Age & Tumor size but what if we want to use more features, likely infinite features; This is a problem coz ∞ features = ∞ datapoints, but we dont have ∞ memory to store these points. Here we can use Support Vectors machine.

Q) Difference between Regression & Classification?

Regression
- → Predicts a number
- → Infinitely many possibilities

Classification
- → Predicts categories
- → Smaller number of possible outputs

⑤ Unsupervised learning

→ We have unlabeled dataset
→ We plot this unlabeled dataset and try to find clusters in them thus it is also called clustering algorithm.
→ Eg: google news ; collects all of the news regarding a particular topic. Here in the begining there is nothing given to us ; but based on the similarities between multiple news we cluster them into same groups. involves clustering
→ "I Don't know what is the data but can you find structure in this data"

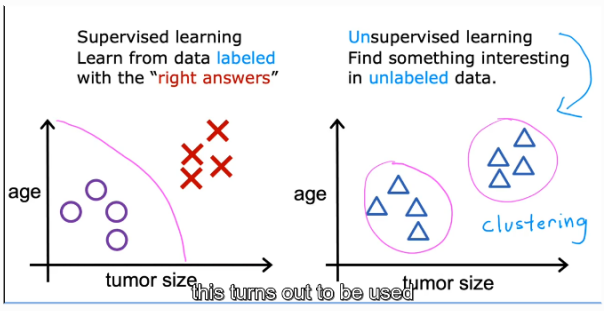
Cocktail party problem: (A type of unsupervised learning problem)

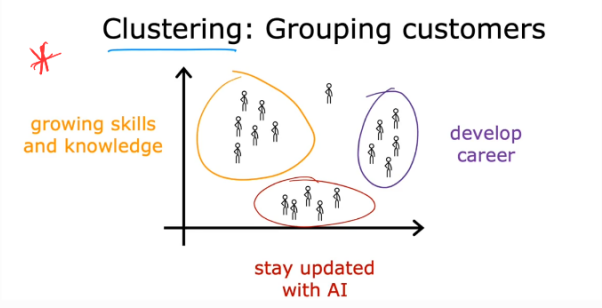
multiple people talking at the same time and it is difficult to understand anything. taking example with 2 people & 2 Mics
∴ This does not involve clustering
∴ Here we find structure in chaotic environment

These mic M_1 & M_2 will collect voices of both P_1 & P_2. but when both of these Audios are used together and based on the distance we can filter out the voices of P_1 & P_2 individually

* Data only comes with input x but not output labels y (like in supervised learning)
* Algorithm has to find structure in the data.
* Clustering → Grouping similar data points together
* Dimensionality reduction → Compress data using fewer numbers
* Anamoly Detection → Find unusual data points

Clustering Example :-





* All the news have the three words common - eg of clustering

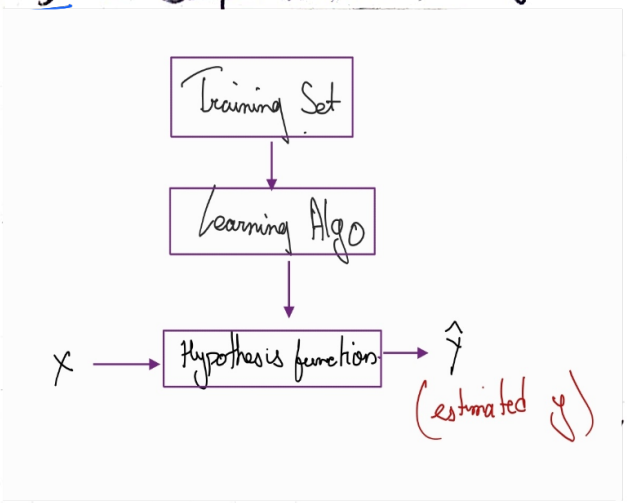
* Unsupervised learning Algorithms are used at →

⇒ Given a set of news articles found on the web, group them into sets of articles about the same story.

Given a database of customer data, automatically discover market segments and group customers into different market segments.

Linear Regression with One Variable (Univariate Linear Regression) (2)

- **Model Representation**
  - → It is a type of supervised learning
  - → Regression i.e predicting Real-valued output (predicts number)
  - → Here we use Training set where :
    - m is total number of training example
    - x is input variable / features
    - y is output variable / target



→ This training data set is passed through a learning Algorithm which gives out h. h is a hypothesis function.

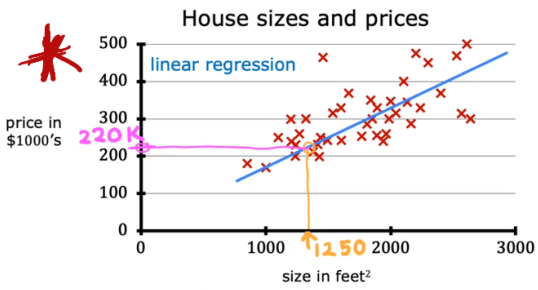
→ For the example of housing if we put in number of square feet of our house in h it will give predicted price of it.

→ we represent h as :→

$$h(x) = \theta_0 + \theta_1 x$$ { i.e $y = mx + c$ }

(linear equation m is slope & C is constant)

→ we are simplifying and sticking to only straight lines for model-fitting. thus a linear equation is used.



Supervised learning model
Data has "right answers"

Corresponding Data Set

Data table

size in feet²	price in $1000's
2104	400
1416	232
1534	315
852	178
...	...
3210	870

Terminology

Training set: Data used to train the model

	x: size in feet²	y: price in $1000's
(1)	2104	400
(2)	1416	232
(3)	1534	315
(4)	852	178
...	...	...
(47)	3210	870

m = 47

$x^{(1)} = 2104$ $y^{(1)} = 400$

$(x^{(1)}, y^{(1)}) = (2104, 400)$

$x^{(2)} = 1416$ $x^{(2)} \neq x^2$ not exponent

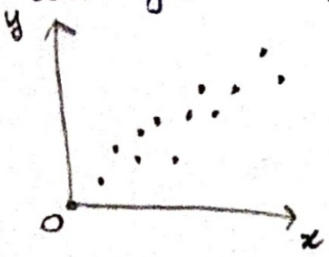
Notation:
- x = "input" variable / feature
- y = "output" variable / "target" variable
- m = number of training examples

(x, y) = single training example

$(x^{(i)}, y^{(i)})$
$(x^{(i)}, y^{(i)})$ = i^{th} training example
index (1st, 2nd, 3rd ...)

- **Cost function**

→ Suppose we get the plotted graph as below :→



→ Here it is impossible to choose a Value of θ_1 & θ_0 in a way that all points passes perfectly.

→ In this case we construct a line which will create least error in all cases.

→ Turns out that using squared error function is the best way to deal with Regression problems. thus we get

Cost function

Denoted by J

$$\underset{\theta_0 \theta_1}{\text{minimize}} \; \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^i) - y^i \right)^2$$

→ $h_\theta(x^i)$ is the value we get from our hypothesis function

→ y is the actual value

∴ Thus if we minimize this value and get a value of θ_0 & θ_1 we get a hypothesis function that creates least average error possible

→ $\frac{1}{2m}$ is for average

Cost function: Squared error cost function

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

$\hat{y}$ ↑ error

m = number of training examples

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

Comparision between hypothesis & cost function :

→ $h(x) = \theta_0 + \theta_1 x$ — i.e function of x

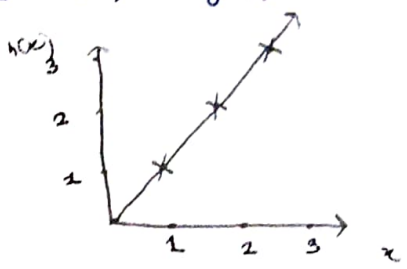
$J(\theta_0, \theta_1) = \frac{1}{2M} \sum_{i=0}^{n} (h(x)^i - y^i)^2$ — i.e function of θ_0 & θ_1

→ now for each value of θ_0 & θ_1 gives us a different fit in the graph. and thus we get a value of J corresponding to this chosen θ_0 & θ_1.

→ for the simplicity of calculation let $\theta_0 = 0$, {i.e all fits pass through origin.

and the dataset is

x	y
1	1
2	2
3	3



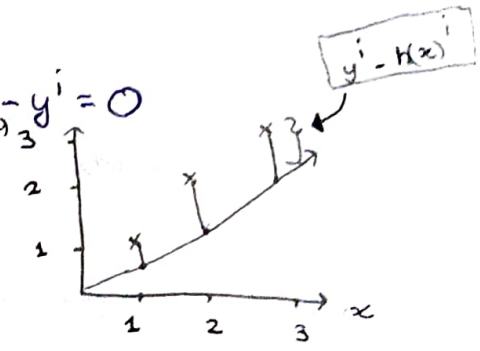
→ for $\theta_1 = 1$ we get a graph as :

for $\theta_1 = 1$ & $\theta_0 = 0$ we get J as

$$J(0,1) = \frac{1}{2*3} \sum_{i=0}^{3} (h(x)^i - y^i)^2$$

∴ here for all cases $h(x)^i = y^i \Rightarrow h(x)^i - y^i = 0$

$\Rightarrow J(0,1) = 0$



→ for $\theta_1 = 0.5$ we get a graph as :

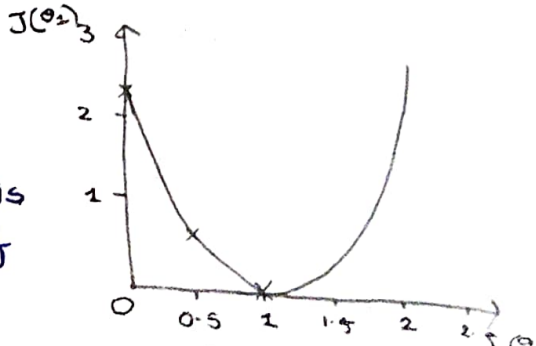
$$J(0,0.5) = \frac{1}{6}\left[(0.5)^2 + (1)^2 + (1.5)^2\right]$$

$$= \frac{1}{6}[0.25 + 1 + 2.25]$$

$$= \frac{3.5}{6}$$

$$= 0.58$$

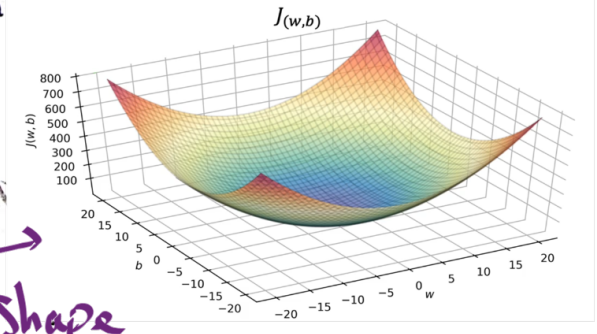
→ similarly if we repeat this process is others values we get the graph of cost function as :→



Conclusion :→ we get a parabolic graph ; by studying this graph we get that at $\theta_1 = 1$ we get $J(\theta_1) = 0$ i.e least possible cost for our scenario & thus we choose $\theta_1 = 1$.

(★) If we do not neglect θ_0 i.e $\theta_0 \neq 0$ then we get a 3-D graph of cost function which is similar to the above graph but just 3-D.

here the bottom point of this curve have least value of J and thus we choose it.



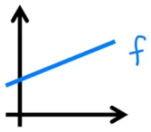
Soup bowl or Hammock Shape

model:

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

parameters:

w, b



cost function:

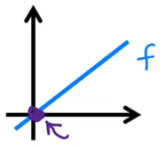
$$J(w,b) = \frac{1}{2m}\sum_{i=1}^{m}(f_{w,b}(x^{(i)}) - y^{(i)})^2$$

goal:

$$\underset{w,b}{\text{minimize}}\, J(w,b)$$

simplified

$f_w(x) = wx$ $\quad b = \emptyset$

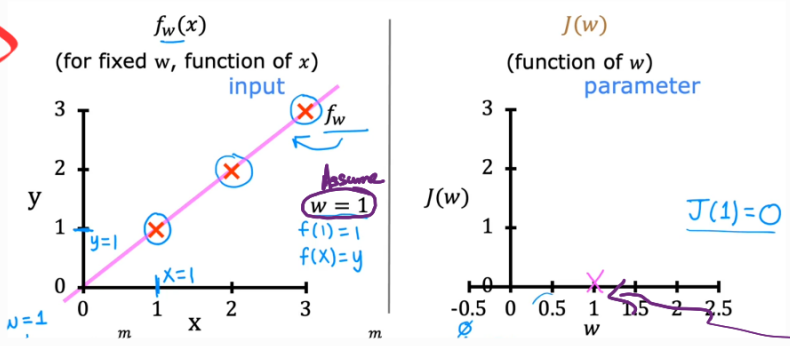


$$J(w) = \frac{1}{2m}\sum_{i=1}^{m}(f_w(x^{(i)}) - y^{(i)})^2$$

$wx^{(i)}$

$$\underset{w}{\text{minimize}}\, J(w)$$

⇒ for simplicity of calculation

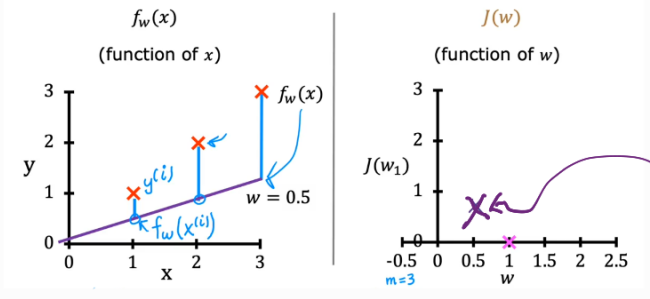


$$J(w) = \frac{1}{2m}\sum_{i=1}^{m}(f_w(x^{(i)}) - y^{(i)})^2$$

$$= \frac{1}{2m}\left((1-1)^2 + (2-2)^2 + (3-3)^2\right)$$

$$= \frac{1}{2m} \times 0 = 0$$

⇒ Similarly for $w = 0.5$,

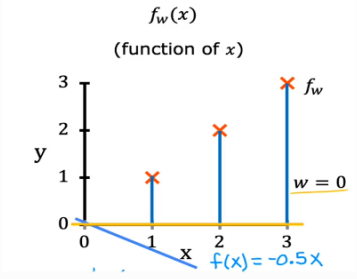


$$J(w) = \frac{1}{2m}\sum_{i=1}^{m}(f_w(x^{(i)}) - y^{(i)})^2$$

$$= \frac{1}{2m}\left[(0.5-1)^2 + (1-2)^2 + (1.5-3)^2\right]$$

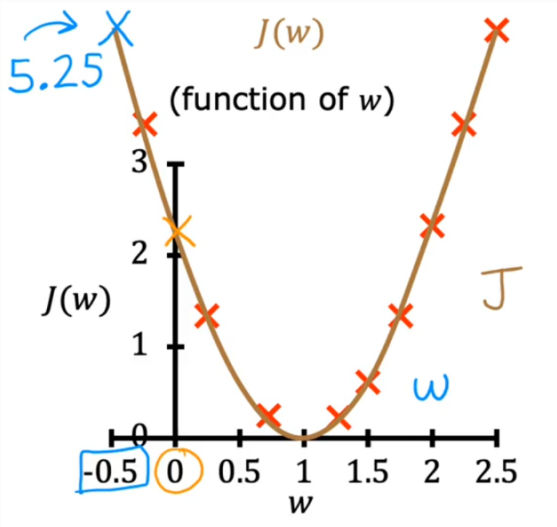
$$= \frac{1}{2\times 3} \times 3.5 \approx \boxed{0.58}$$

⇒ Similarly if we do for $w = 0$,



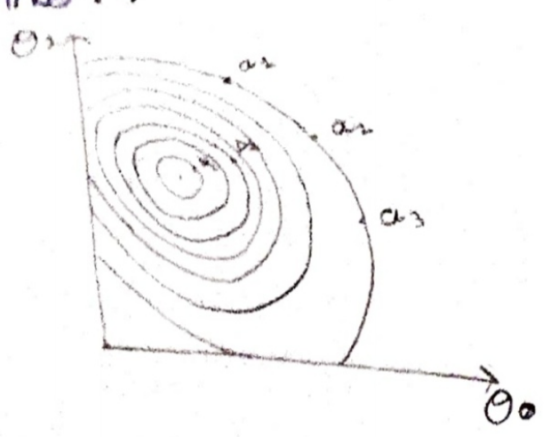
⇒ we get $J(w) = 2.3$

After completely plotting J(w) we get a parabolic curve.



Our final goal is to minimize the difference, From the above plotted graph we can see that the minimum differencence (J(w)) is for w = 1. Hence we choose w = 1

Contours Plots :→ Traying to find min J in a 3-D graph is diff. so we use a Contour plot rather then the 3 D graph. it looks like :→



∴ It is the projection of the 3-D graph on the θ_0-θ_1 plane. ③

∴ for all points a_1, a_2 a_3 value of J is same

∴ Value of J at b_2 < Value of J at b_1

∴ Thus more the center we go, both or value of θ_0 & θ_1 is found i.e Values yielding to lower J.

Gradient Descent

→ just doesn't apply to cost f^n of Lin Reg

→ Up until now all the methods we saw required us to plot the graph for cost function and observe its lowest point for value of θ's.

→ Gradient Descent does not require construction of graph

→ Procedure :

1. Assume a 3-D graph; here choose any random point.
2. Height of this graph shows cost. Thus our objective is to reach to the least height possible.
3. At this random point scan 360° around us and choose a direction which leads us to a smaller height. take some steps
4. Repeat till we reach to a position where no direction can decrease height.

Here

- := is an assignment operator, i.e A := B means update value of A such that A = B happens.
- α is learning rate i.e defines how big step are we taking after fixing direction
- $\frac{\partial}{\partial \theta_j}$ Partial derivation ; defines which direction to move.

Mathematically,

$$\Rightarrow X = X - \alpha \frac{\partial}{\partial w} J(X, Y)$$

Generalizing it for us,

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

→ we know we have two variables here and the update for both is necessary

→ We need to keep in mind that before w gets updated we also change b. i.e, simultaneously update w & b

Correct: Simultaneous update

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

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

$$w = tmp\_w$$

$$b = tmp\_b$$

Incorrect

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

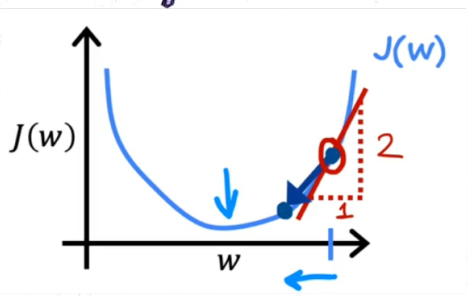
$$w = tmp\_w$$

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

$$b = tmp\_b$$

w is getting updated before J(w,b) can be calculated for b

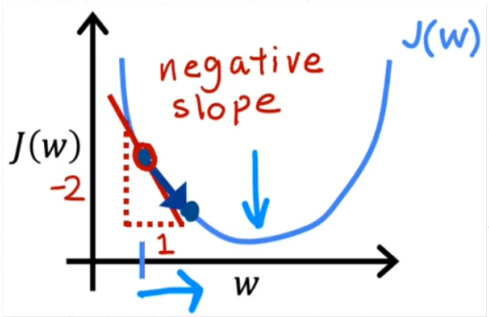
→ As we know $\frac{d f(x)}{dx}$ will give us slope of x, and if we assume $\theta_0 = 0$, we get a 2-D graph and $\frac{\partial}{\partial \theta_1} = \frac{d}{d\theta_1}$ i.e for only 1 variable partial diff. & normal diff. are same thus



∴ here $\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1)$

and at this point slop is positive thus $\frac{\partial}{\partial \theta_1} J(\theta_1)$ is a positive number

∴ Thus updated value of θ_1 is smaller then Prev value and that is what we want.

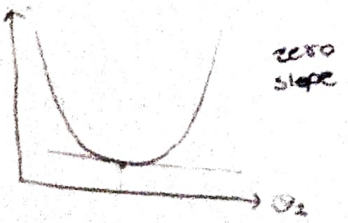


∴ here slope is -ve

⇒ $\frac{\partial}{\partial \theta_1}$ is -ve ; thus new θ_1 > Prev θ_1

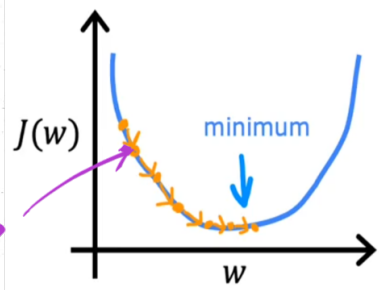
$(\omega - \alpha(-\text{ve No}) = \omega + \alpha(+\text{ve No})$ ⇓ As old new Value greater!

and that is what we want to reach at θ^1



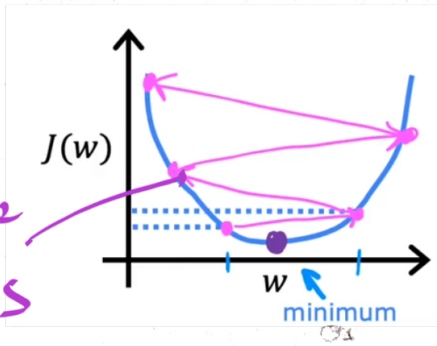
∴ slop is 0

thus min already reach and value of θ_1 will not change now.



Small Steps

∴ very small value of α means we are taking very small steps ; thus gradient decent will be slow and take very long to reach to final point



Large steps

∴ IF we choose a value of α too large then each step will lead us to a value of J even larger than before. Thus we will never reach to final point.

→ Thus we will want to take larger steps initially and decrease step size as we get closer to the answer. (4)

for this we do not have to change α after every point.

As with each step value of Θ_1 changes which will lead to a smaller value of $J(\Theta_1)$ and thus automatically decreasing step size in next iteration

∴ so we are good as long as value of α is not too small nor too big

Can reach local minimum with fixed learning rate? Yes !!

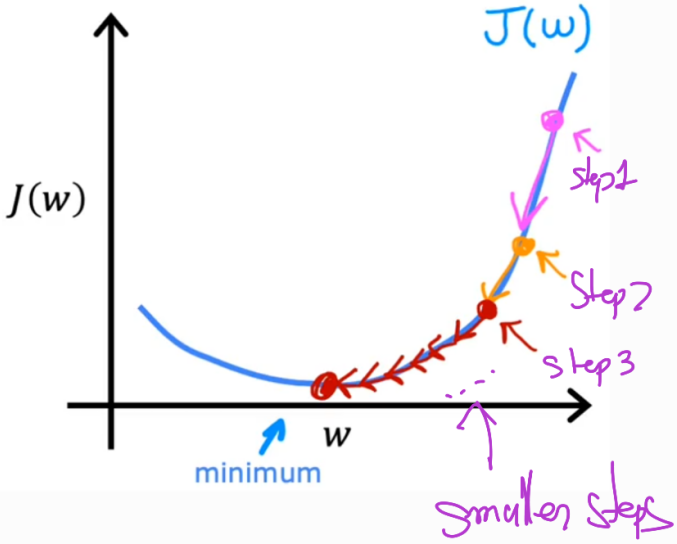
α

$$w = w - \alpha \frac{d}{dw} J(w)$$

smaller

not as large

large



Near a local minimum,
- Derivative becomes smaller
- Update steps become smaller

Can reach minimum without decreasing learning rate α

Linear regression model

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

Cost function

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

Gradient descent algorithm

repeat until convergence {

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

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

}

The derivation of these forms are optional

(Optional)

$$\frac{\partial}{\partial w}J(w,b) = \frac{d}{dw}\frac{1}{2m}\sum_{i=1}^{m}(f_{w,b}(x^{(i)}) - y^{(i)})^2 = \frac{d}{dw}\frac{1}{2m}\sum_{i=1}^{m}(wx^{(i)} + b - y^{(i)})^2$$

$$= \frac{1}{2m}\sum_{i=1}^{m}(wx^{(i)} + b - y^{(i)})2x^{(i)} = \frac{1}{m}\sum_{i=1}^{m}(f_{w,b}(x^{(i)}) - y^{(i)})x^{(i)}$$

$$\frac{\partial}{\partial b}J(w,b) = \frac{d}{db}\frac{1}{2m}\sum_{i=1}^{m}(f_{w,b}(x^{(i)}) - y^{(i)})^2 = \frac{d}{db}\frac{1}{2m}\sum_{i=1}^{m}(wx^{(i)} + b - y^{(i)})^2$$

$$= \frac{1}{2m}\sum_{i=1}^{m}(wx^{(i)} + b - y^{(i)})2 = \frac{1}{m}\sum_{i=1}^{m}(f_{w,b}(x^{(i)}) - y^{(i)})$$

no $x^{(i)}$

Gradient descent algorithm

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

repeat until convergence {

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

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

}

(update w & b simultaneously)

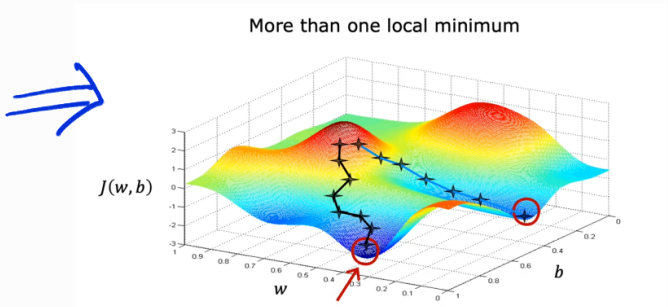
⟹ final Algorithm

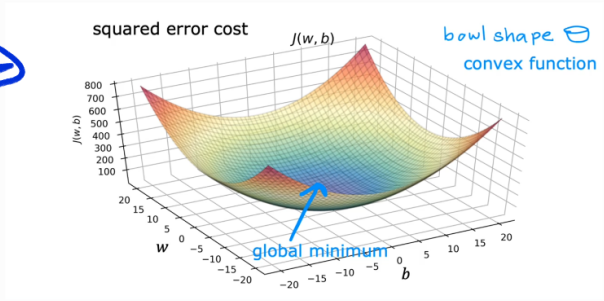
→ This type of gradient descent is called batch gradient descent coz entire range of i=0 to i=m is considered single batch and processed.

Each step of gradient descend uses all the training examples.

→ Disadvantage :→ For a 3-D graph with multiple local minima & a global minima; this process will give the local minima nearest to starting point; not the global minima

But this is not a problem in linear regression coz graph are Convex in nature i.e only 1 global minima.





Matrices and Vectors

→ Matrix : $\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$ size : Row x Column
i.e 3x3

→ If Rows = Column ⇒ square matrix
→ If only 1 column i.e (n x 1) then it is called Vector.
→ $\begin{bmatrix} a & b \\ c & d \end{bmatrix} + \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} a+e & b+f \\ c+g & d+h \end{bmatrix}$ i.e if size same then normal addition possible else invalid
→ $4 \times \begin{bmatrix} a & b \\ c & D \end{bmatrix} = \begin{bmatrix} 4a & 4b \\ 4c & 4D \end{bmatrix}$ i.e irrespective of size multiplication with scalar is possible

→ $\begin{bmatrix} a & b \\ c & d \\ e & f \end{bmatrix} \times \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} ax + by \\ cx + dy \\ ex + fy \end{bmatrix}$ i.e multiplication possible only if no of column in 1st = no. of Rows in 2nd.
(3x2) (2x1) (3x1)

→ $\begin{bmatrix} a & b \\ c & d \\ e & f \end{bmatrix} \times \begin{bmatrix} w & x \\ y & z \end{bmatrix} = \begin{bmatrix} aw + by & ax + bz \\ cw + dy & cx + dz \\ ew + fy & ex + fz \end{bmatrix}$
(3x2) (2x2) (3x2)

→ $A * B \neq B * A$ {i.e not commutative}
→ $A * (B * C) = (A * B) * C$ {i.e associative}
→ Identity matrix : Square matrix with all elements 0 except diagonal which is 1 i.e.
a 4 dimension identity matrix : $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

→ Identity matrix denoted by I. i.e :→ $A * I = A$
→ If $A * x = I$ we say x is the inverse of A ie $x = A^{-1}$
→ Octave has 2 commands inv & Pinv to find inverses. inv finds inverses but gives error if the matrix is irreversible. Pinv means Pseudo-inverse. It gives us values despite the fact that matrix is irreversible.
→ Transpose function : If $A = \begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix}$

then $A^T = \begin{bmatrix} a & d \\ b & e \\ c & f \end{bmatrix}$

Multiple Features

$f_{w,b}(x) = b + wx$; here $h(x) = \theta_0 + \theta_1 x$

→ Till now we used only 2 features i.e x_0 & x are 2 features where $x_0 = 1$

$$\Rightarrow f_{\vec{w},b}(\vec{x}) = w_1x_1 + w_2x_2 + w_3x_3 + \dots w_nx_n + b$$

$\vec{w} = [w_1 \; w_2 \; w_3 \dots w_n]$ ⇒ Row Vector (parameters of the model)

b is a number.

$\vec{x} = [x_1 \; x_2 \; x_3 \dots x_n]$ ⇒ Row Vector of multiple features

$$\therefore f_{\vec{w},b}(\vec{x}) = \vec{w} \cdot \vec{x} = w_1x_1 + w_2x_2 + w_3x_3 \dots w_nx_n + b$$

(dot product)

MULTIPLE LINEAR REGRESSION

Multiple features (variables)

	Size in feet² x_1	Number of bedrooms x_2	Number of floors x_3	Age of home in years x_4	Price ($) in $1000's
	2104	5	1	45	460
$i=2$	1416	3	2	40	232
	1534	3	2	30	315
	852	2	1	36	178
	...	...	...	...	...

$j = 1 \dots 4$

$n = 4$

x_j = j^{th} feature

n = number of features

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

$x_j^{(i)}$ = value of feature j in i^{th} training example

$\vec{x}^{(2)} = [1416 \; 3 \; 2 \; 40]$

$x_3^{(2)} = 2$

Vectorization..

Parametes:-

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

b is a number

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

Python Implementation :—

```
w = np.array([1.0, -2.5, -3.3])
b = 4
x = np.array([10, 20, 30])
```

Without Vectorization :-

$f_{\vec{w},b}(\vec{x}) = w_1x_1 + w_2x_2 + w_3x_3 + b$

```
f = w[0] * x[0] +
    w[1] * x[1] +
    w[2] * x[2] + b
```

Without Vectorization :- (loop)

$f_{\vec{w}b}(\vec{x}) = \left(\sum_{j=1}^{n} w_jx_j\right) + b$

∴ for j in range (n)

```
f = f + (w[j] * x[j])
f = f + b
```

Using Vectorization :-

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

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

Why do Vectorized Code faster then loops ?

→ Vectorized code usually use parallel processing in the hardware making the calculations faster. whereas loops are a serial implementation and thus slower relatively.

Gradient Descent for multiple Variables

→ now we have $\theta_0, \theta_1, \theta_2 \dots \theta_n$, update all of these simultaneously till convergence

Gradient descent

One feature

repeat {

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

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

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

simultaneously update w, b

}

n features ($n \geq 2$)

repeat {

$j = 1$

$$w_1 = w_1 - \alpha \frac{1}{m}\sum_{i=1}^{m}\left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)x_1^{(i)}$$

$\frac{\partial}{\partial w_1} J(\vec{w},b)$

$\vdots$

$j = n$

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

}

Feature & Parameter Scaling :-

⇒ Price = $w_1x_1 + w_2x_2 + b$

(x_1: Size, x_2: #bedrooms)

x_1 range → 300 – 2000 ft^2

x_2 range → 0 – 5

House: $x_1 = 2000$, $x_2 = 5$, price = 500K ← taking one training data for example

⇒ According to the above what should be the value of w_1, w_2 ?

Ans

$w_1 = 50,\ w_2 = 0.1,\ b = 50$

$\widehat{price} = \underbrace{50 * 2000}_{100{,}000K} + \underbrace{0.1 * 5}_{0.5K} + \underbrace{50}_{50K}$

$\widehat{price} = \$100{,}050.5k = \$100{,}050{,}500$

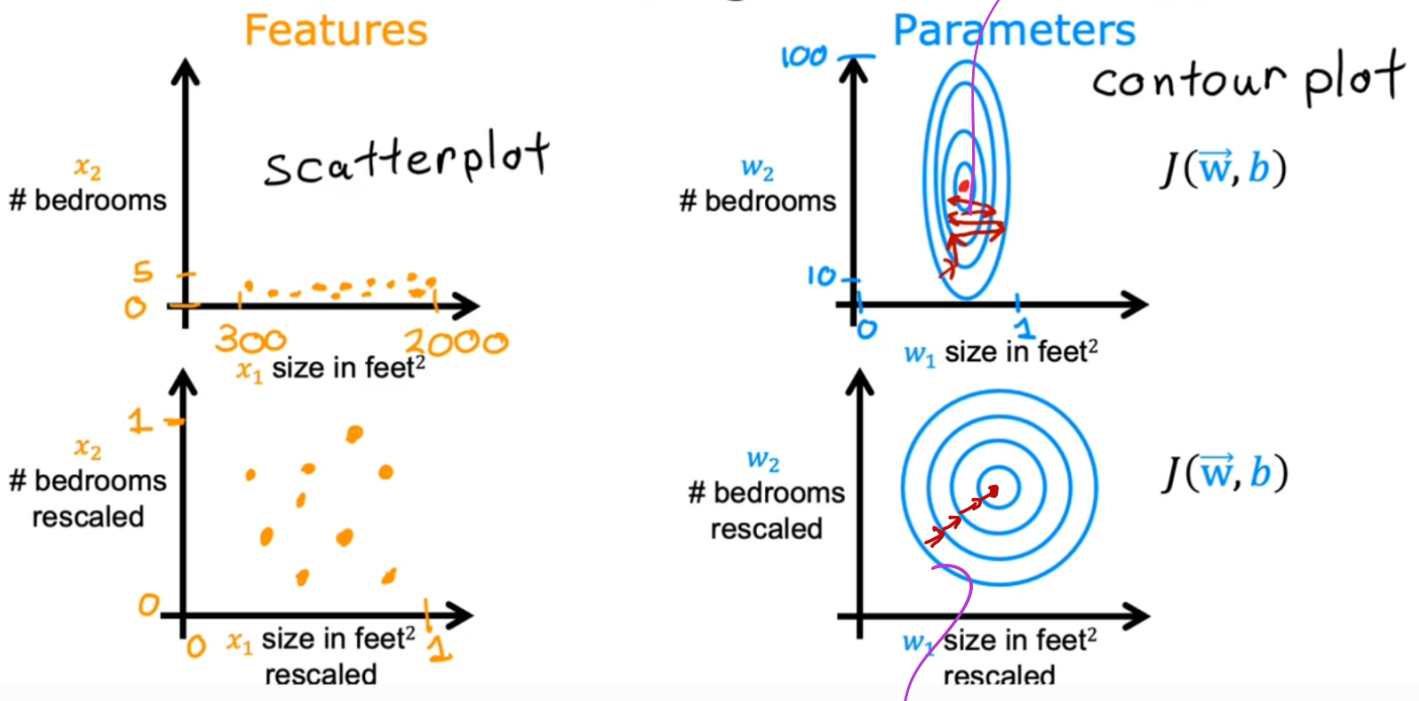
$w_1 = 0.1$ (small), $w_2 = 50$ (large), $b = 50$

$\widehat{price} = \underbrace{0.1 * 2000k}_{200K} + \underbrace{50 * 5}_{250K} + \underbrace{50}_{50K}$

$\widehat{price} = \$500k$ more reasonable

The values keeps jumping to both sides before reaching the global minimum

Feature size and gradient descent



The value directly reaches the global minimum.

When Feature Scaling is Used :-

aim for about $-1 \leq x_j \leq 1$ for each feature x_j

$-3 \leq x_j \leq 3$ } acceptable ranges

$-0.3 \leq x_j \leq 0.3$ }

$0 \leq x_1 \leq 3$ okay, no rescaling

$-2 \leq x_2 \leq 0.5$ okay, no rescaling

$-100 \leq x_3 \leq 100$ too large → rescale

* Anything close to -1 to 1 is acceptable else rescale it !

How to choose feature Scaling :-

1) Divide by Maximum :—

$300 \leq x_1 \leq 2000$ $\quad$ $0 \leq x_2 \leq 5$

$x_{1\,scaled} = \frac{x_1}{2000}$ $\quad$ $x_{2\,scaled} = \frac{x_2}{5}$

$0.15 \leq x_{1\,scaled} \leq 1$ $\quad$ $0 \leq x_{2\,scaled} \leq 1$

2) Mean Normalization :—

Usually scaled answer ranges from -1 to 1. We center everything around zero.

eg, Assuming $\mu_1 (\text{Mean } x_1) = 600$, $\mu_2 (\text{Mean } x_2) = 2.3$

$300 \leq x_1 \leq 2000$ $\quad$ $0 \leq x_2 \leq 5$

$x_1 = \frac{x_1 - \mu_1}{\text{Max} - \text{Min}}$ $\quad$ $x_2 = \frac{x_2 - \mu_2}{\text{Max} - \text{Min}}$

$\therefore -0.18 \leq x_1 \leq 0.82$ $\quad$ $-0.46 \leq x_2 \leq 0.54$

3) Z-Score Normalization :-

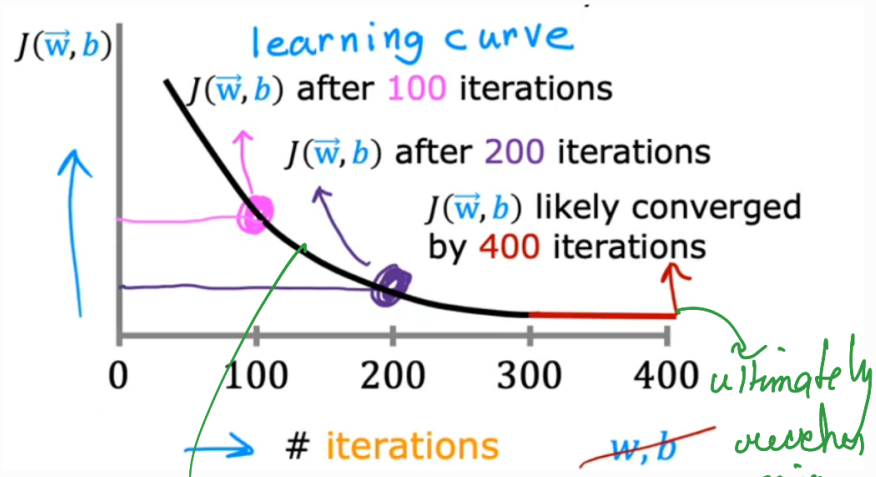
We calculate standard deviation for this

$$x = \frac{x - \mu}{\sigma}$$

$-0.67 \leq x_1 \leq 3.1$ $\quad$ $-1.6 \leq x_2 \leq 1.9$

Making Sure Gradient Descent is Working :—

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



~~w, b~~ ultimately reaches min. (convergence)

According to Gradient Descent $J(\vec{w}, b)$ should **decrease** after every iteration. Hence the downward curve.

Automatic Convergence test :-

Let ε (epsilon) be 10^{-3}

If $J(\vec{w}, b)$ decreases $\leq \varepsilon$ declare convergence.

$\Downarrow$

i.e., $J(\vec{w}, b)$ has reached the flat line part (red part) of the graph.

∴ We have found parameter $\vec{w}, b$ to reach global minimum.

Learning Rate

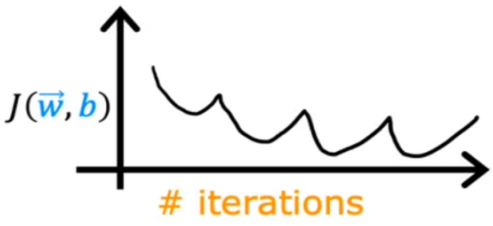
→ To Confirm that we have chosen a perfect α; plot $J(\theta)$ vs no of iteration

→ if everything is right value of $J(\theta)$ decreases after every iteration

→ else value of α is too large

→ $J(\theta)$ can be declared as converged if the decrease in $J(\theta)$ after every iteration is less than 10^{-3}.

Identify problem with gradient descent



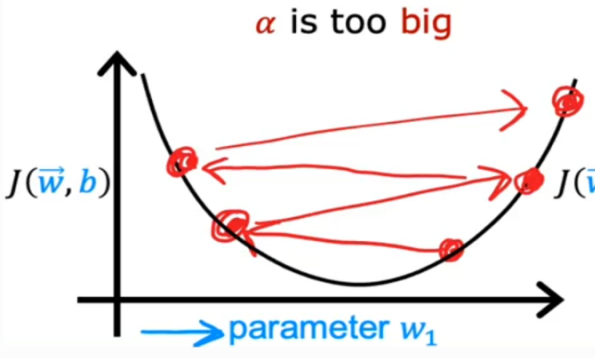
or learning rate is too large

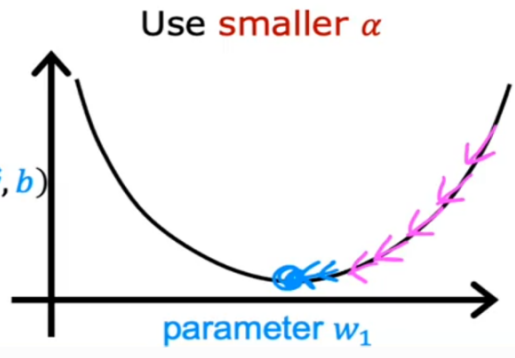
$w_1 = w_1 + \alpha d_1$

use a minus sign

$w_1 = w_1 - \alpha d_1$

Adjust learning rate





With a small enough α, $J(\vec{w}, b)$ should decrease on every iteration

If α is too small, gradient descent takes a lot more iterations to converge

Normal Equation → Only to solve linear Regression (Usually not recommended)

→ another method to minimize J. unlike gradient descent it is a non-iterative approach.

usually used in ML libraries

$$\theta = (X^T X)^{-1} X^T y$$

→ eg: →

x_0	Size x_1	no of bedroom x_2	no. of floors x_3	Age of house x_4	Price y
1	2104	5	1	45	460
1	1416	3	2	40	232
1	1534	3	2	30	315
1	852	2	1	36	178

here $X = \begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{bmatrix}$ $y = \begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \end{bmatrix}$

→ Here the data set is small so matrix multiplication & inverse is computationally inexpensive ; but for larger dataset gradient descent is preferable

→ feature scaling does not affect the Normal Equation procedure

∴ Gradient descent vs Normal equation :

Gradient descent	Normal equation
• Need to choose α	• no need of α
• Iterative process	• no iteration
• Complexity : $O(kn^2)$	• $O(n^3)$ {coz need to find inverse}
suited when n is large	• suited when n is small preferably < 10,000

→ in Octave Use pinv for inverse so that we get value of θ if $(X^T X)$ is non invertable.

→ reasons due to which $X^T X$ can become non-invertable :→

- redudant features {i.e 2 or more feature linearly related}
- Too many features {$m \leq n$}

Feature engineering

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

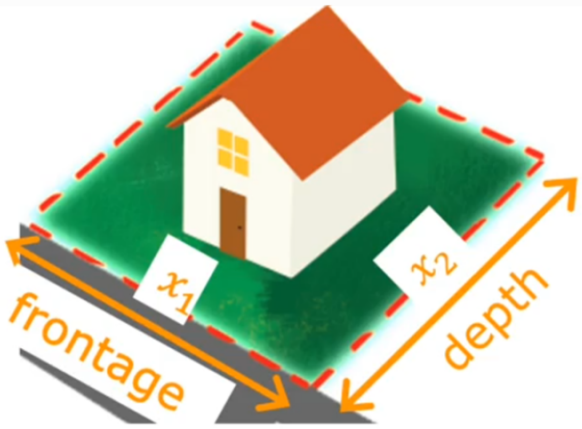
frontage depth

$$area = frontage \times depth$$

$$x_3 = x_1 x_2$$

new feature

$$f_{\vec{w},b}(\vec{x}) = w_1 x_1 + w_2 x_2 + w_3 x_3 + b$$



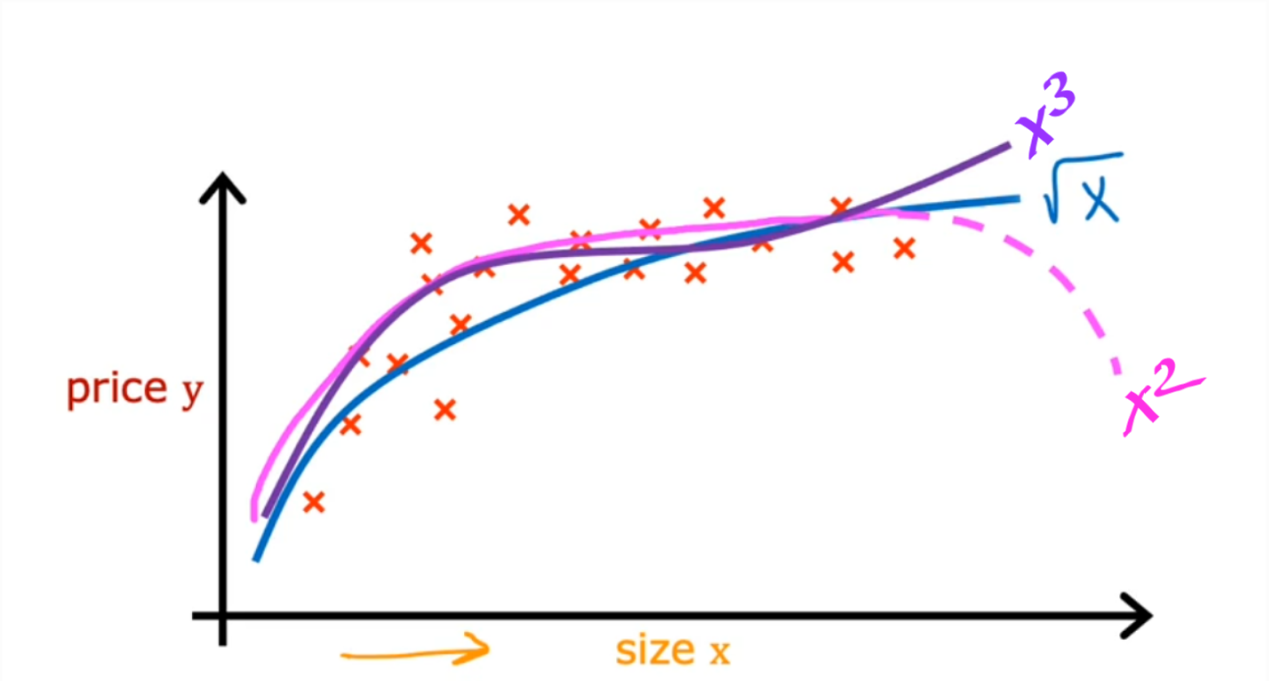
Feature engineering: Using intuition to design new features, by transforming or combining original features.

* Sometimes the features we start with doesn't help us predict correctly. Hence we need to take certain new features while working. Here x_3.

* Feature engineering can be used to fit not just straight lines but also curves to a set of data.

Polynomial Regression :-

-> Helps us fit curve, non linear function to the data.



$$f_{\vec{w},b}(x) = w_1 x + w_2 x^2 + b$$

(size, size^2)

$\Rightarrow$ We can't use this as x^2 comes down in the graph plotting ∩

$$f_{\vec{w},b}(x) = w_1 x + w_2 x^2 + w_3 x^3 + b$$

(size, size^2, size^3)

We are taking different values of x → Feature Scaling.

$$f_{\vec{w},b}(x) = w_1 x + w_2 \sqrt{x} + b$$

(size, $\sqrt{\text{size}}$)

CLASSIFICATION

⇒ output is discreet

⇒ The answer can only have 2 values → Yes or No.

Email is spam → Y or N

Tumour is Malignant → Y or N

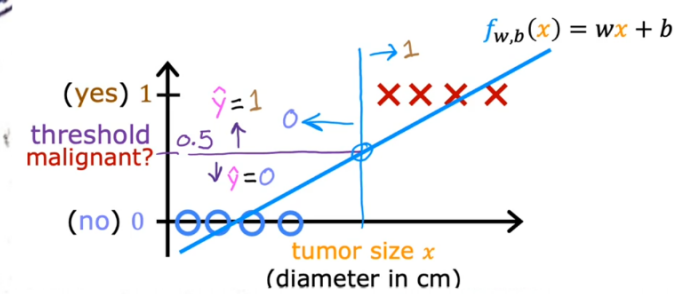
⇒ Also known as binary classification

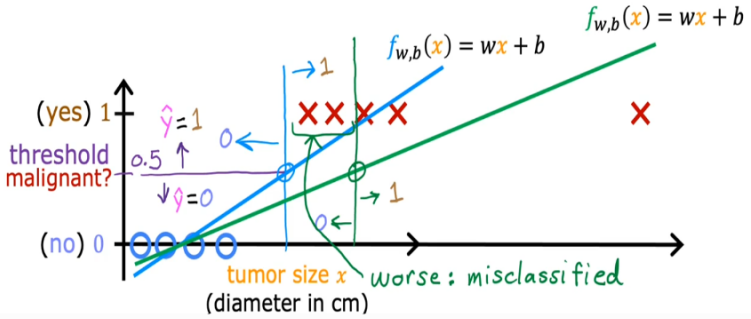
⇒

False	True
0	1
Negative Class	Positive Class

⇒ If we try using linear Regression, we fail miserably

eg 1





- Any value > 0.5 is positive (1) & others are negative (0).
- Seems to work fine with this example.
- It is obvious that adding a training data set with very large Tumor size should be a Malignant tumor.
- But here our hypothesis does not work fine, starting results does not match!!

Sigmoid Function :-

=> It is one of the most used function in Machine/Deep learning.

=> Also known as squashing function.

=> Takes input and returns values between 0 and 1.

=> S-Shaped Curve (Sigmoid Curve)

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

Domain → R

Range → (0,1)

=> As x approaches +ve ∞, $\sigma(x)$ approaches 1

" " -ve ∞, $\sigma(x)$ " 0

$$\sigma'(x) = \sigma(x)(1-\sigma(x))$$

*

Let $y = \sigma(x) = \frac{1}{1+e^{-x}}$

Let u = 1 + e^{-x}. Thus, y = 1/u.

First, find the derivative of u with respect to x :

du/dx = $-e^{-x}$.

Then, find the derivative of y with respect to u:

$\frac{dy}{du} = -\frac{1}{u^2}$

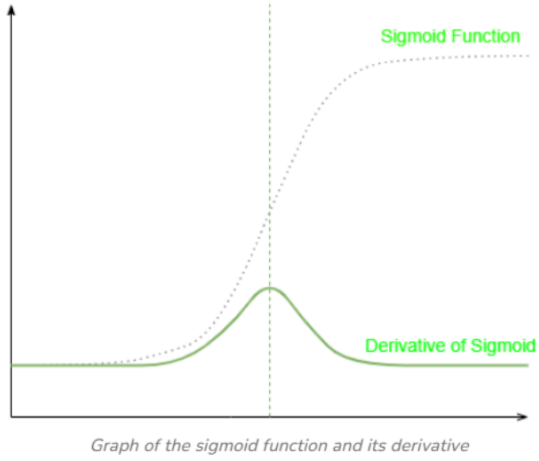
Apply the chain rule:

$\frac{dy}{dx} = \frac{dy}{du} \cdot \frac{du}{dx} = -\frac{1}{u^2} \cdot (-e^{-x}) = \frac{e^{-x}}{(1+e^{-x})^2}$

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

Thus, $\sigma'(x) = \frac{e^{-x}}{(1+e^{-x})^2} = \left(\frac{1}{1+e^{-x}}\right)\left(\frac{e^{-x}}{1+e^{-x}}\right) = \sigma(x)\,(1-\sigma(x))$

*



Graph of the sigmoid function and its derivative

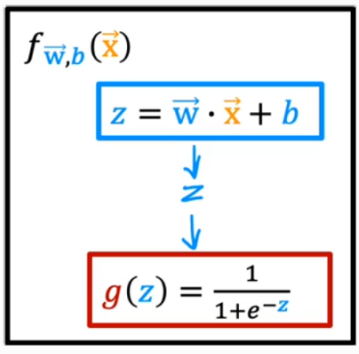
Using Sigmoid for logistic Regression :-

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

We have seen in linear Regression

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

Now we put this in the sigmoid function,



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

$\therefore$ Logistic Regression,

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

Interpretation of logistic regression output :-

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

"probability" that class is 1

P(1)

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

Probability that y is 1, given input $\vec{x}$, parameters $\vec{w}, b$

Example:

x is "tumor size"

y is 0 (not malignant) or 1 (malignant)

$$P(y = 0) + P(y = 1) = 1$$

$f_{\vec{w},b}(\vec{x}) = 0.7$

70% chance that y is 1

Decision Boundary :-

⇒ Decision Boundary is the line on the map which helps us in classification.

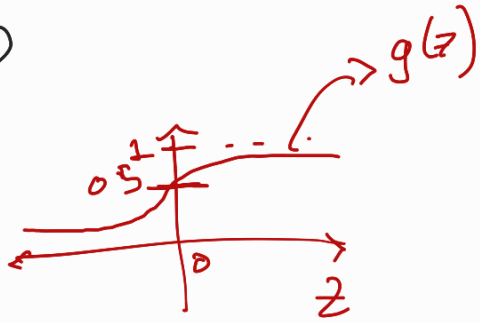
⇒ It is the threshold line (0.5)

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

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

Is $f_{\vec{w},b}(\vec{x}) \geq 0.5$,

Yes: $\hat{y}=1$ No : $\hat{y}=0$



when ≥ 0.5

$g(z) \geq 0.5$

when $z \geq 0$ → we can see from the graph

$\vec{w}\vec{x}+b \geq 0$	$\vec{w}\vec{x}+b < 0$
$\therefore \hat{y}=1$	$\hat{y}=0$

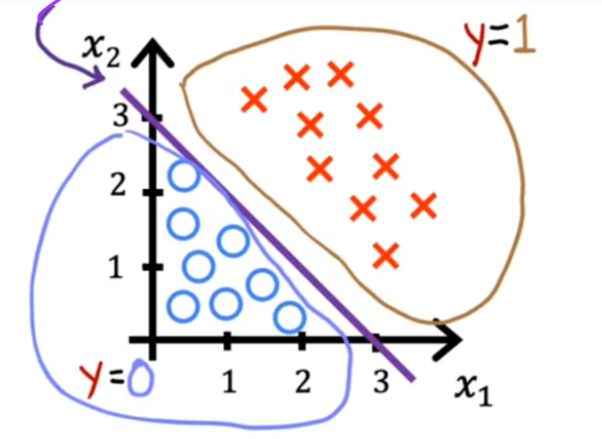
Different example for Decision Boundary :-

$\Rightarrow f_{\vec{w},b}(\vec{x}) = g(z) = g(\underset{1}{w_1}x_1 + \underset{1}{w_2}x_2 + \underset{3}{b})$ → Assume

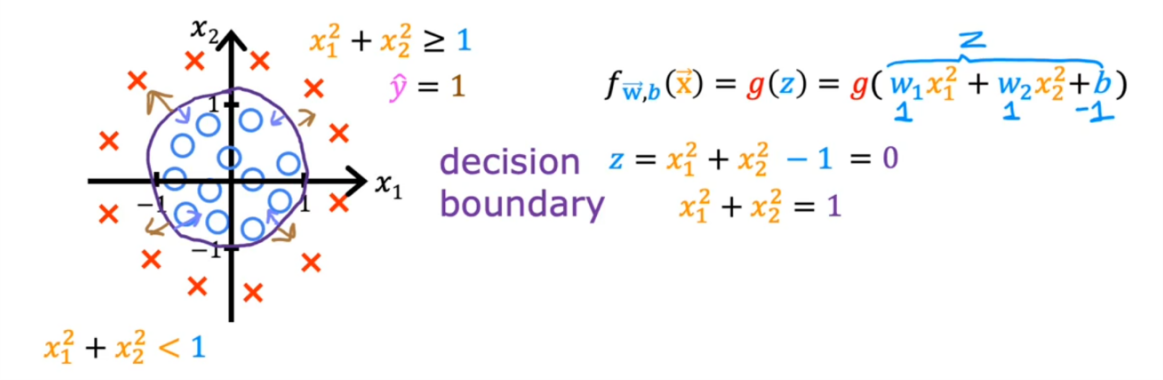
Decision Boundary, $z = \vec{w} \cdot \vec{x} + b = 0$

$z = x_1 + x_2 - 3 = 0$

$x_1 + x_2 = 3$

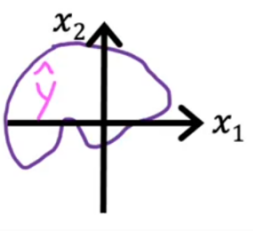


⇒ Non Linear Decision Boundary —



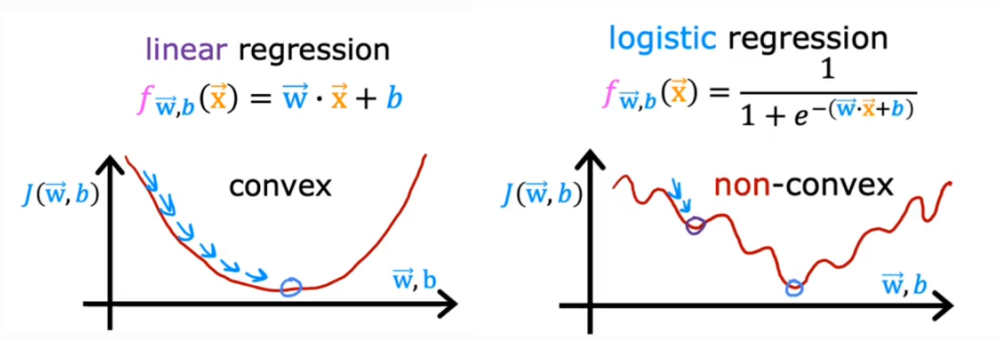
⇒ Decision Boundary can be a even more complex shape. It depends on the no. of features and their degree.

$$f_{\vec{w},b}(\vec{x}) = g(z) = g(w_1x_1 + w_2x_2 + w_3x_1^2 + w_4x_1x_2 + w_5x_2^2 + w_6x_1^3 + \cdots + b)$$



Cost Function :—

* The cost function that we used in linear regression → Squared Error Cost cannot be used for Regression.
* If we use the Squared Error Cost we get a non-convex curve that has a lot of local minima and we keep getting sucked in it.



Squared Error Cost → $\frac{1}{m} \sum_{i=1}^{m} \frac{1}{2} \left(f_{\vec{w},b}(\vec{x}_i) - y_i \right)^2$

⇓ we take this as a new function

loss → $L\left(f_{\vec{w},b}(\vec{x}_i) , y^{(i)} \right)$

Logistic loss function :—

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

cost

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

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

convex
↳ can reach a global minimum

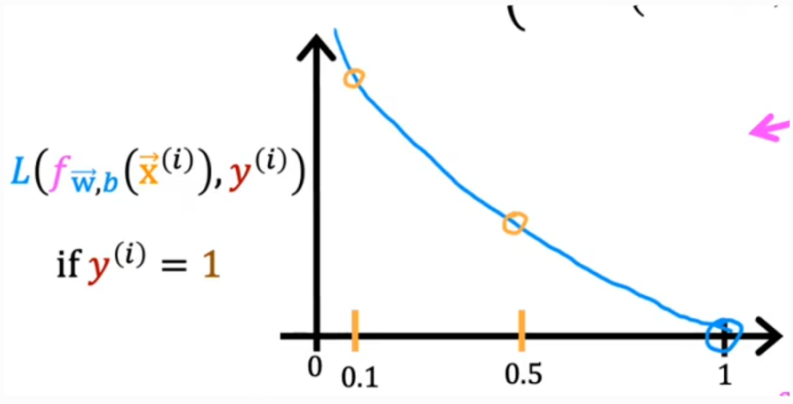
⇒ We know log(f), log(6), 0, 1, −log(6)

⇒ In logistic Regression answers can be only between 0 and 1 and hence we will only focus on that part.

⇒ If $y^{(i)} = 1$,

If $f_{\vec{w},b}(\vec{x}_i) \to 1$ then,
loss $\to 0$

If $f_{\vec{w},b}(\vec{x}_i) \to 0$ then
loss $\to \infty$.

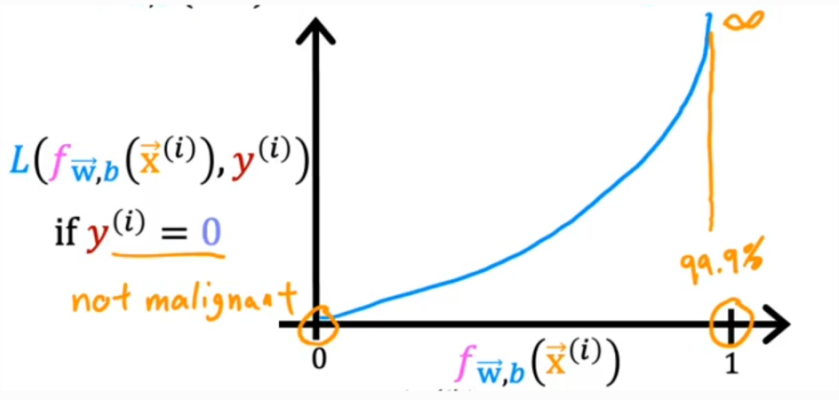


∴ loss is lowest when $f_{\vec{w},b}(\vec{x}_i)$ predicts closest to the true value of $y^{(i)}$

⇒ If $y^{(i)} = 0$,

If $f_{\vec{w},b}(\vec{x}_i) \to 1$ then,
loss $\to \infty$

If $f_{\vec{w},b}(\vec{x}_i) \to 0$ then
loss $\to 1$



∴ The further the prediction $f_{\vec{w},b}(\vec{x}^{(i)})$ is from target $y^{(i)}$, the higher the loss.

Simplified loss function :-

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

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

$\Rightarrow$ if $y^{(i)} = 1$:

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

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

$(1-1) = 0$; $y^{(i)} \to 1$; $\left(1 - y^{(i)}\right) \to 0$

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

$$L\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right), y^{(i)}\right) = -(1-0)\log(1 - f(\vec{x}))$$

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

$y^{(i)} \to 0$; $\left(1 - y^{(i)}\right) \to 0$

Simplified cost function :-

loss

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

convex

cost

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

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

Gradient descent

cost

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

repeat {

$j = 1 \dots n$

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

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

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

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

} simultaneous updates

Gradient descent for logistic regression

repeat {

$$w_j = w_j - \alpha \left[\frac{1}{m}\sum_{i=1}^{m} (f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}) x_j^{(i)}\right]$$

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

} simultaneous updates

We might think that this looks similar to Linear Regression but the $f_{\vec{w},b}(\vec{x}^{(i)})$ is different for both.

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

Same concepts:

- Monitor gradient descent (learning curve)
- Vectorized implementation
- Feature scaling

→ These are same as Linear Regression

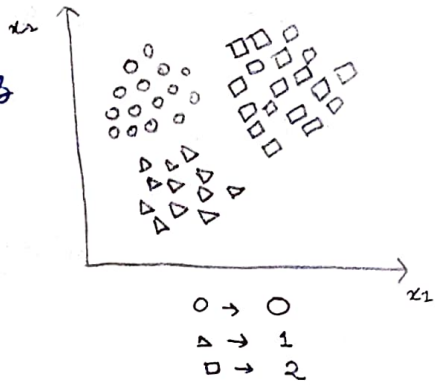
Multiclass Classification

→ Approach for multiple classes ; i.e $y = 0, 1, 2, 3 \ldots n$ { multiple classes but still discrete }

→ For $y = 0, 1, 2 \ldots n$ we divide the problem into smaller $n+1$ binary classifications.

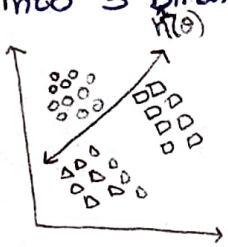
eg :→ If $y = 0, 1, 2$ then :→

we divide this into 3 binary classifications



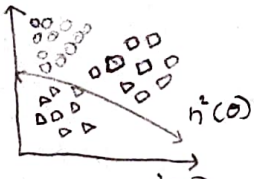
① For class 0 :

from this we get $h^1(\theta)$



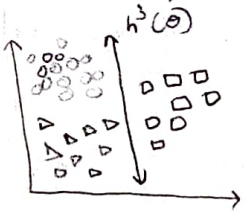
② For class 1 :

from this we get $h^2(\theta)$



③ For class 2 :

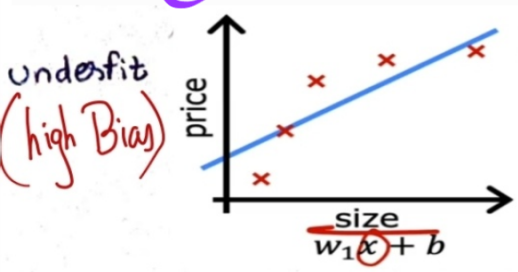
from this we get $h^3(\theta)$



☺ By combining $h^1(\theta)$, $h^2(\theta)$ and $h^3(\theta)$ we can classify this multi class classification

Prediction = $\max_i \left(h_\theta^i(x)\right)$

OverFitting :—

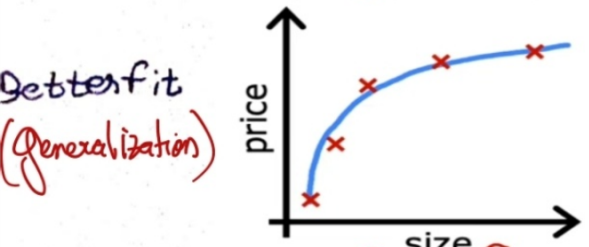


underfit (high Bias)

$w_1x + b$

☹ using $y = \theta_0 + \theta_1 x$ does not pass through any points of trainingset

errors on training set
errors in realtime values



Bettersfit (Generalization)

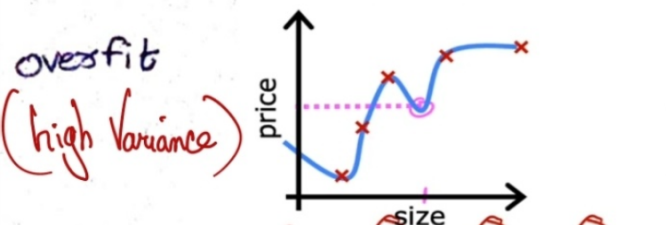
$w_1x + w_2x^2 + b$

☹ adding an extra x^2 feature

$y = \theta_0 + \theta_1 x + \theta_2 x^2$

Decent fit ; Passes through almost all points.

Decent on training set
Decent on realtime values



oversfit (high Variance)

$w_1x + w_2x^2 + w_3x^3 + w_4x^4 + b$

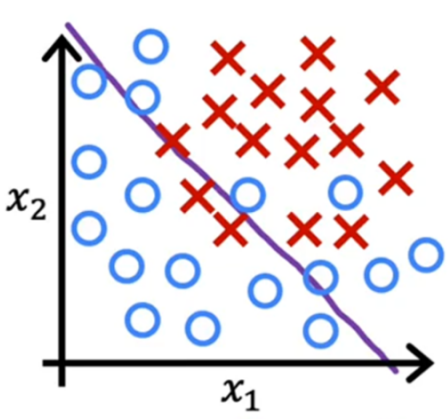
☹ adding too many features

$y = \theta_0 + \theta_1 x_1 + \theta_2 x_2^2 + \theta_3 x_1 x_2^2 + \theta_4 x_1^2 x_2$

Perfect fit ; Cost = 0.

Perfect on training set
errors in real time values

Classification

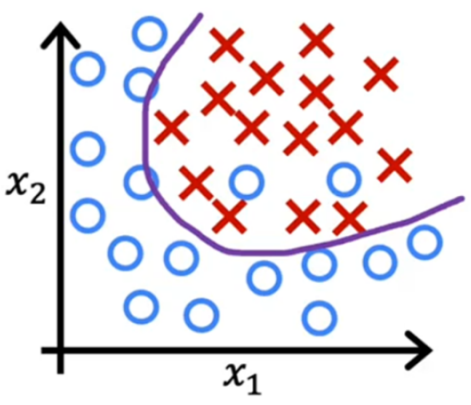


$z = w_1x_1 + w_2x_2 + b$

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

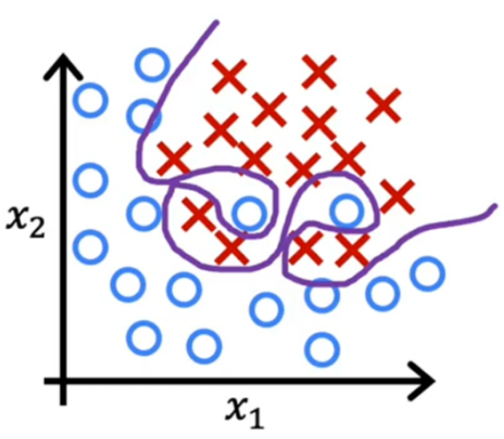
g is the sigmoid function

underfit high bias



$z = w_1x_1 + w_2x_2 + w_3x_1^2 + w_4x_2^2 + w_5x_1x_2 + b$

just right



$z = w_1x_1 + w_2x_2 + w_3x_1^2x_2 + w_4x_1^2x_2^2 + w_5x_1^2x_2^3 + w_6x_1^3x_2 + \cdots + b$

overfit

☹ The problem with overfitting is we tryhard to fit the training dataset due to which we loose the generalized behaviour of curve. so we have cost 0 for training set but while calculating realtime values it shows very high cost.

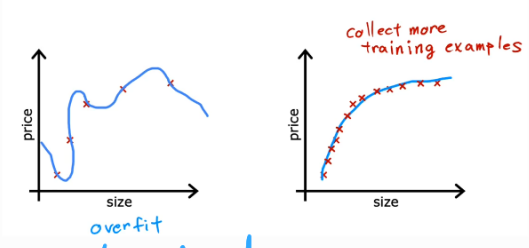
Loose generalized behaviour

Reducing Overfitting :-

1) Collect more Data.
2) Reduce the no. of features.
3) Regularize the features.

Collect More Data :-

→ We can always collect more data to convert an overfit data to a normal Data.
→ The only problem is that sometimes more data is just not available

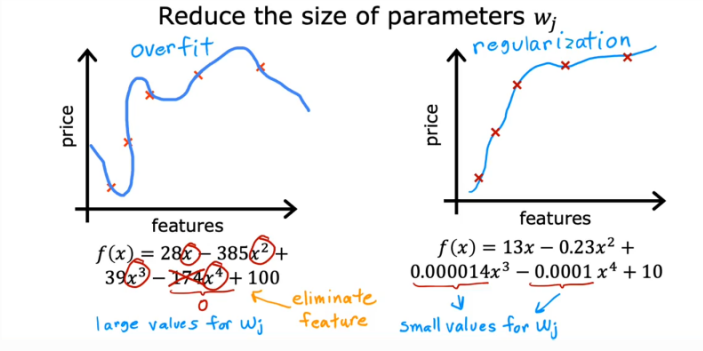


Reduce the no. of features :-

→ If there are too many feature and we have insufficient data then it will result in Overfitting.
→ We can reduce it by taking a subset of the features
→ Disadv → As we are taking a limited no. of features and not all of them, we may lose out on important features.

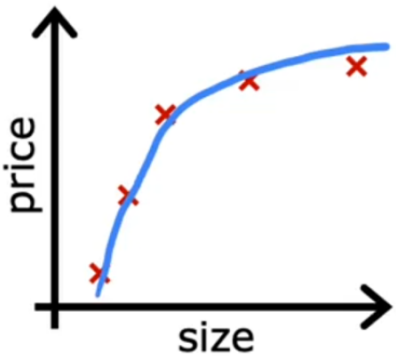
Regularization :-

→ Instead of just simply omitting a feature we can take a smaller (Regularized value of the parameter.

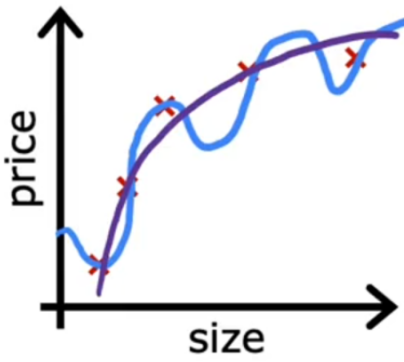


Regularization :—

Intuition



$w_1x + w_2x^2 + b$



$w_1x + w_2x^2 + \underbrace{w_3x^3}_{\approx 0} + \underbrace{w_4x^4}_{\approx 0} + b$

make w_3, w_4 really small (≈ 0)

$$\min_{\vec{w},b} \frac{1}{2m}\sum_{i=1}^{m}\left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)^2 + 1000\underbrace{w_3^2}_{0.001} + 1000\underbrace{w_4^2}_{0.002}$$

regularization term

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

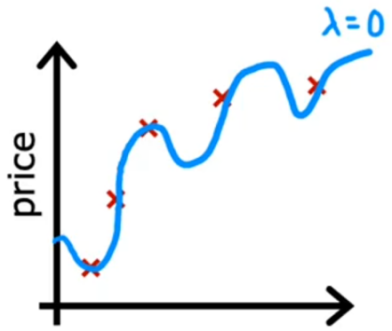
can include or exclude

$\lambda > 0$

Choosing λ :—

If $\lambda = 0$, the regularization term is zero. and we end up with the same overfitted data.

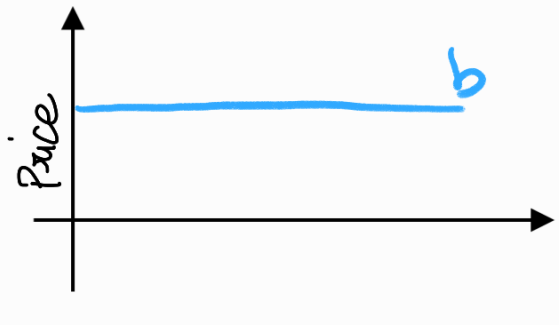
$$f_{\vec{w},b}(\vec{x}) = w_1x + w_2x^2 + w_3x^3 + w_4x^4 + b$$



If λ = very high (10^{10}), then the parameters have to be very very small and tends to 0.

$$f_{\vec{w},b}(\vec{x}) = \underbrace{w_1}_{\approx 0}x + \underbrace{w_2}_{\approx 0}x^2 + \underbrace{w_3}_{\approx 0}x^3 + \underbrace{w_4}_{\approx 0}x^4 + b$$

$\therefore f(x) = b$



Regularized linear regression

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

Gradient descent

repeat {

$$w_j = w_j - \alpha \frac{\partial}{\partial w_j} J(\vec{w},b) \qquad j = 1,\ldots,n$$

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

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

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

} simultaneous update

don't have to regularize b (as it was optional)

Implementing gradient descent

repeat {

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

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

} simultaneous update

Regularized logistic regression

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

$$\min_{\vec{w},b}$$

Gradient descent

Looks same as for linear regression!

repeat {

$$w_j = w_j - \alpha \frac{\partial}{\partial w_j} J(\vec{w},b) \qquad j = 1\ldots n$$

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

logistic regression

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

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

}

don't have to regularize