Machine Learning-01

Video-01 **Human Learning**

- **Learning**
  - * ability to see patterns
  - * recognize patterns
  - * add some constraints
- **Knowledge**
  - * variability in constraints
- **Intelligence**
  - * invoke the knowledge for given test case

Machine Learning

E * T = P

It is the experience 'E' wrt to some task 'T' with some performance measure 'P'

A computer program that learns from experience 'E' wrt some task 'T' with some added performance measure 'P'

The performance on T as measured by P improves with E

Human Intelligence	Machine Intelligence
→ Based on biological neural network where large no. of neurons are connected and perform certain tasks on activation	→ It's based on von-neuman architecture
→ Thinks in parallel	→ Serial & non-ambiguous.
→ Process & Represent	→ Represent and then process

Artificial Intelligence: mimic human brain intelligence using ML

Machine learning: It is a field study that gives computers the ability to learn without being explicitly programmed

Artificial Intelligence: Technique which allows the machine to act like humans by replacing the behaviours and nature or mimic human brain

Machine learning: It is subset of AI in which it allows machines to learn and make predictions based on its experience or data

DL: Subset of AI and ML that achieves great power & flexibility to by learning to represent hierarchy of concepts or abstractions -OR- features generated automatically

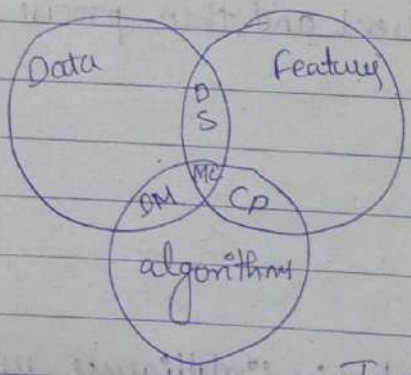
1950 - Turing's Test

1952 - First learning algorith
Samuel - Game of Checker's

1957 - First Neural Network
Frank Rissan

1981 - First logic based Desigu
Gerald

2006 - Coined the term DL
Jeffrey

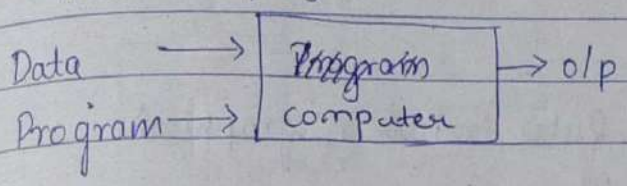


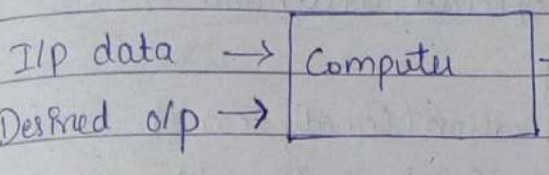
DS - data science
DM - data mining
CP - classical programming

Types
- Supervised learning: Data + label
- Unsupervised: Data - structures
- Semi
- Reinforcement
- Speech Analysis
- Internet content classifica

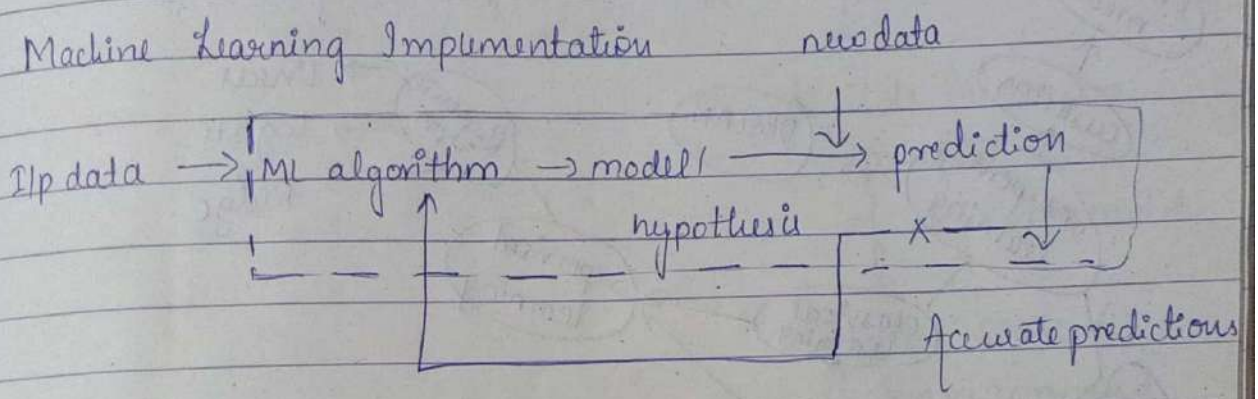
DBSCAN

Traditional programs:





Machine Learning Implementation



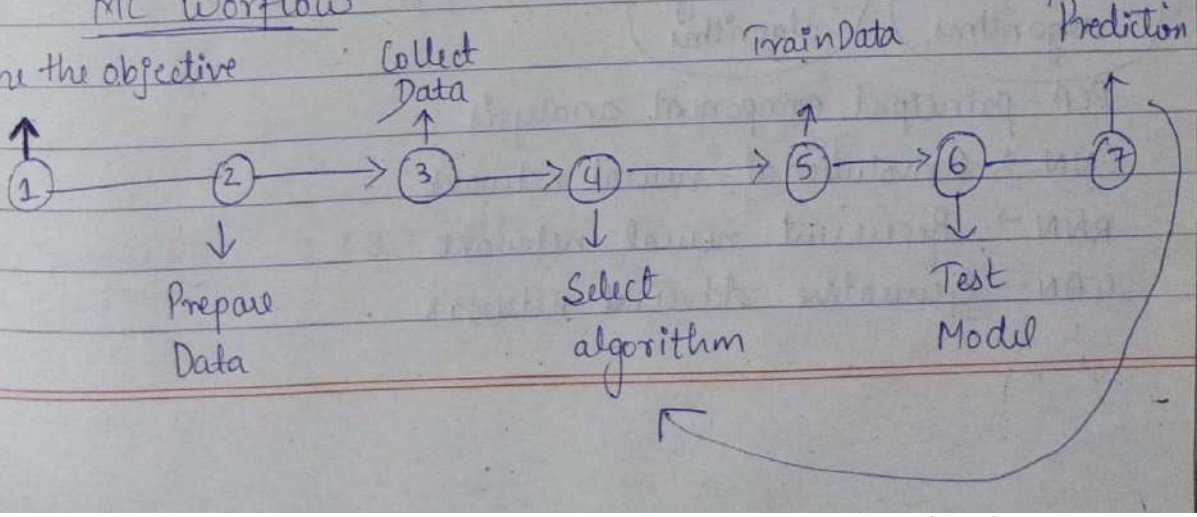
Application:

1) Voice / Speech Recognition
2) Search Engine
3) No. plate recognition
4) Dream Reader

Seed → Algo
Nutrients → Data
Gardener → Programmers
Plants → programs

Video-02

ML Workflow

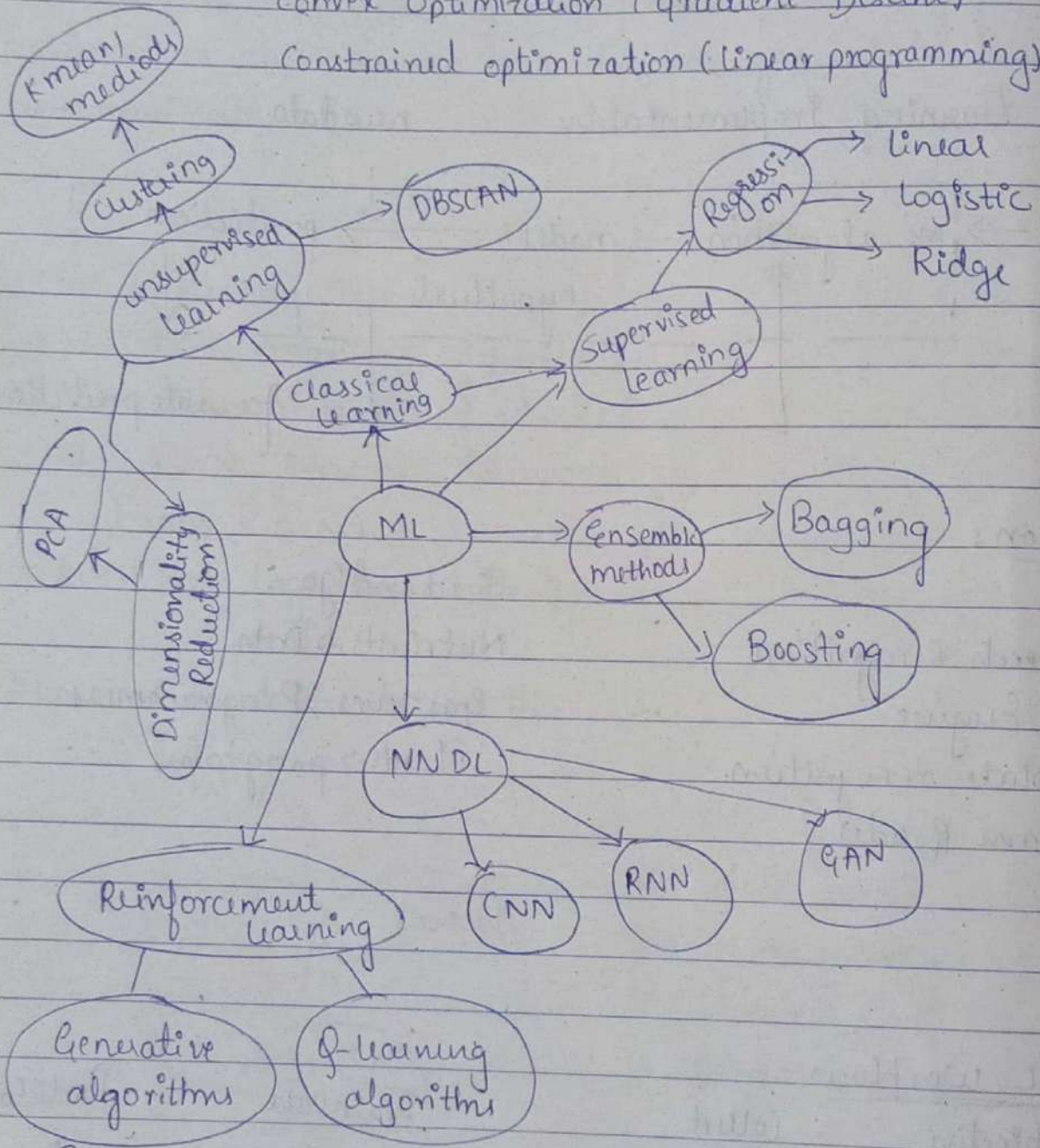


ML Components

Representation → Decision tree, Rules, instances, graphical method, models

Evaluation → Accuracy, Precision & Recall, Likelihood, & Entropy

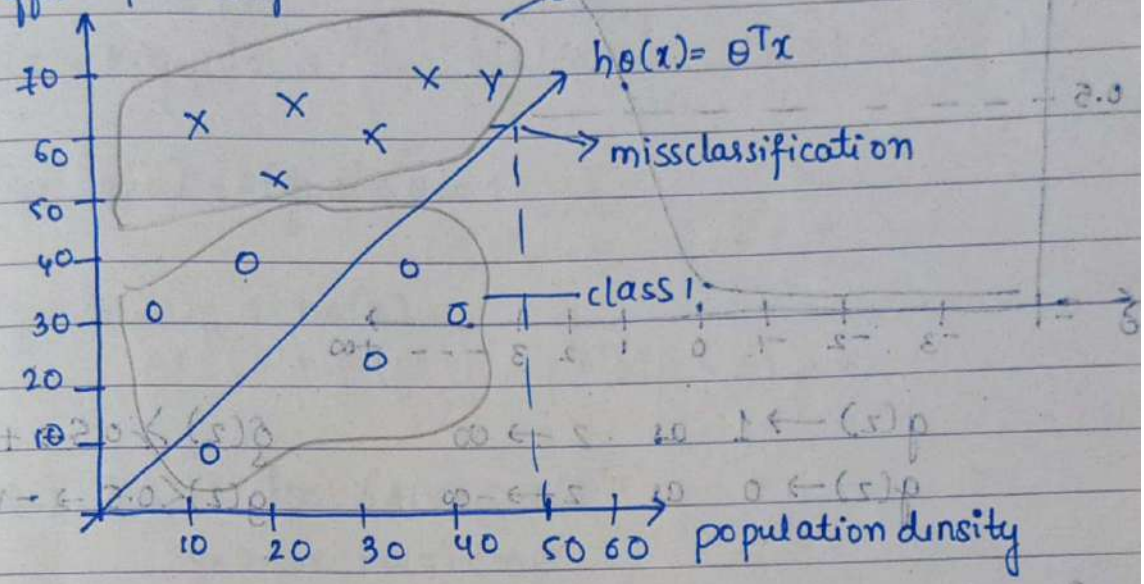
Optimization → Combinatorial optimization (Greedy Search)
Convex Optimization (Gradient Descent)
Constrained optimization (linear programming)



PCA - principal component analysis
CNN → Convolutional neural network
RNN → Recurrent neural network
GAN - Generative Adversial network

Logistic Regression

Why linear regression can't be used?



logistic regression is used for classification type of problems where the y/predicted values belongs to finite set of discrete values

$x^{(i)}$ → Inputs for training

$y^{(i)}$ → discrete values

For binary classification,

$x^{(i)} \rightarrow y$ (0) → -ve class

(1) → +ve class

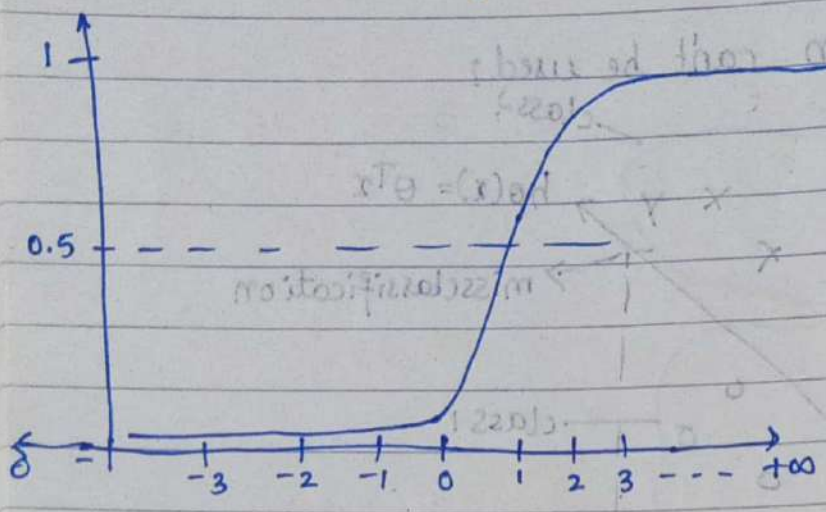
$y \in \{0, 1\}$

The hypothesis $h_\theta(x) = g(\theta^T x)$ is non-linear

$$g(\theta^T x) = \frac{1}{1 + e^{-(\theta^T x)}}$$

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

Sigmoid function (g) / Logistic function.



$g(z) \to 1$ as $z \to \infty$

$g(z) \to 0$ as $z \to -\infty$

$g(z) \geq 0.5 \Rightarrow +ve$

$g(z) < 0.5 \Rightarrow -ve$

Derivative of $g(z)$

$$\frac{d}{dz}(g(z)) = \frac{d}{dz}\left(\frac{1}{1+e^{-z}}\right)$$

$$= \left(\frac{1}{(1+e^{-z})^2}\right)(e^{-z})(-1)$$

$$= \frac{e^{-z}}{(1+e^{-z})^2}$$

$$= \left(\frac{1+e^{-z}-1}{(1+e^{-z})^2}\right) \quad \text{// add \& sub 1}$$

$$= \frac{1}{(1+e^{-z})}\left[\frac{1+e^{-z}}{1+e^{-z}} - \frac{1}{1+e^{-z}}\right]$$

$$= \frac{1}{(1+e^{-z})}\left[1 - \frac{1}{1+e^{-z}}\right]$$

$$\boxed{g'(z) = g(z)\left(1 - g(z)\right)}$$

Linear regression outputs continous no values and logistic regression outputs a probability value mapped to two / more class

$$h_\theta(x) = P(y=1 \mid x, \theta)$$

$$P(y=1 \mid x, \theta) + P(y=0 \mid x, \theta) = 1$$

$$P(y=0 \mid x, \theta) = 1 - h_\theta(x)$$

$h_\theta(x) \to y=1$ for $h_\theta(x) \gg 0.5$

$y=0$ for $h_\theta(x) < 0.5$

Stochastic Gradient Rule

General Rule:

$$P(y \mid x, \theta) = h_\theta(x)^y \cdot (1 - h_\theta(x))^{1-y}$$

Maximum likelihood functions: gives probability $y=0/1$ given by

$$L(\theta) = \prod_{i=1}^{m} h_\theta(x^{(i)})^{y^{(i)}} \cdot (1 - h_\theta x^{(i)})^{(1-y^{(i)})} \to (1)$$

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

arrive at max point more quickly.

$$\log(x^n) = n \log x \quad \Big\| \quad l(\theta) = \sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1-y^{(i)}) \log(1 - h_\theta x^{(i)}) \to (2)$$

$$l(\theta) = y \log h_\theta(x) + (1-y) \log(1 - h_\theta(x))$$

$$\theta : \theta + \alpha \frac{\partial}{\partial \theta_j} l(\theta)$$

$$\frac{\partial}{\partial \theta_j}(l(\theta)) = \left(\frac{y}{h_\theta(x)} - \frac{(1-y)}{(1-h_\theta(x))} \right) \frac{\partial}{\partial \theta} h_\theta(x)$$

$$= \left(\frac{y}{g(\theta^T x)} - \frac{(1-y)}{(1-g(\theta^T x))} \right) g(\theta^T x)(1-g(\theta^T x)) x_j$$

$$= y(1 - g(\theta^T x) - (1-y) g(\theta^T x)) x_j$$

$$= (y - y g(\theta^T x) - g(\theta^T x) + y g(\theta^T x)) x_j$$

$$= (y - g(\theta^T x)) x_j$$

$$\therefore \theta_j = \theta_j + \alpha \left(\frac{\partial}{\partial \theta_j} (l(\theta)) \right)$$

// Stochastic Gradient Rule for θ updation

$$\theta_j = \theta_j + \alpha (y^i - g(\theta^T x^{(i)})) x_j^i$$

In logistic regression, the SGD algorithm minimizes the loss function by computing its gradient

m - no. of training samples

(x, y) -> x's (input variable / feature) y's -> target variable

$(x^{(i)}, y^{(i)})$ - i^{th} training example.

Univariate Linear Regression

$h_\theta(x) = \theta_0 + \theta_1 x$.

θ_0, θ_1 → parameters

Basic Model:

Training Set

↓

Learning Algorithm

↓

Size Of House → h → Estimated Price

hypothesis

h maps from x's to y's

Cost Function:

Notion: Choose the parameters so that $h_\theta(x)$ is close to y for our training examples (x, y)

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

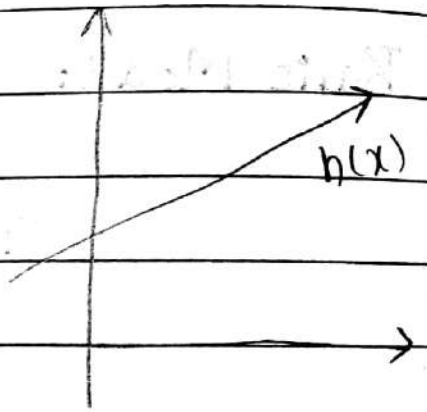
$$h_\theta(x^{(i)}) = \theta_0 + \theta_1 x^{(i)}$$

$J(\theta_0, \theta_1)$ also called Squared-Error-Cost-function

Cost function - Intuition I

Hypothesis:

$$h_\theta(x) = \theta_0 + \theta_1 x.$$



Parameters:

θ_0, θ_1

Cost function:

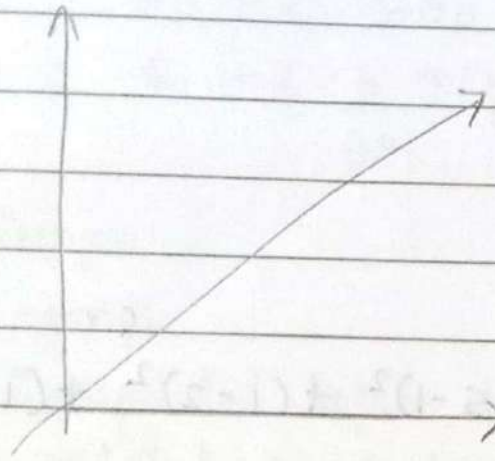
$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$

Goal: $\underset{\theta_0, \theta_1}{\text{minimize}}\ J(\theta_0, \theta_1)$

Scenario I:

$\theta_0 = 0$ (So all hypothesis pass through origin)

$h_\theta(x) = \theta_1 x$



Cost function:

$$J(\theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

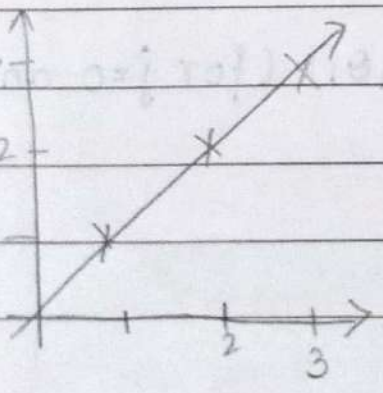
$$\text{minimize}_{\theta_1} \; J(\theta_1)$$

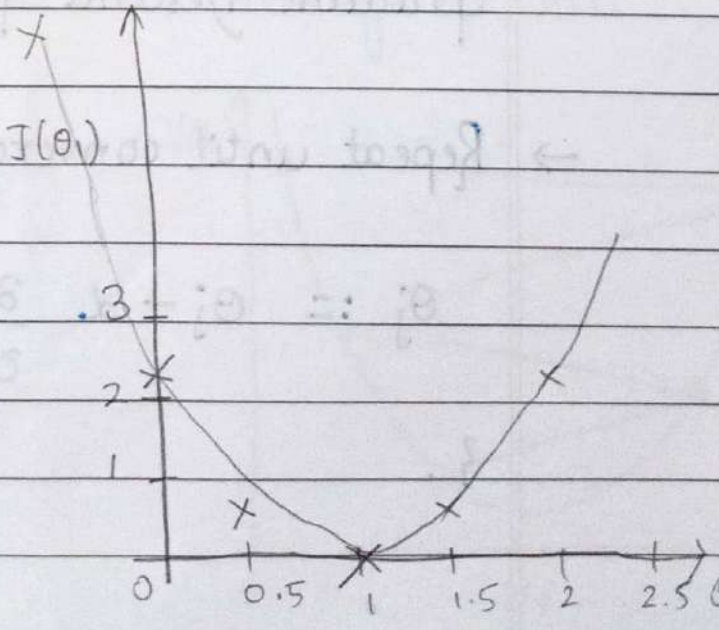
$h_\theta(x)$
(for fixed value of θ_1, function of x)

$J(\theta_1)$
function of θ_1

Consider $\theta_1 = 1$

So $h_\theta(x) = x$





When $\theta_1 = 1$

$h_\theta(x) = x$

So, $J(\theta_1) = 0$

When $\theta_1 = 0.5$

$h_\theta(x) = 0.5x$

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

When $\theta_1 = 0$

$h_\theta(x) = 0.$

$$J(\theta) = \frac{1}{2m}\left[1^2 + 2^2 + 3^2\right] = 2.3$$

For linear regression we get bow shaped surface for cost function.

Gradient Descent for minimizing cost function.

→ Repeat until convergence {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \quad (\text{for } j=0 \text{ and } j=1)$$

}.

Correct way of updations:-

1 tempo := $\theta_0 - \alpha \frac{\partial}{\partial \theta_0} (J(\theta_0, \theta_1))$

2 temp1 := $\theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$

3 θ_0 := tempo

4. θ_1 := temp1

→ If θ_0 is updated before step2 then temp1 is calculated with updated θ_0 (which is wrong)

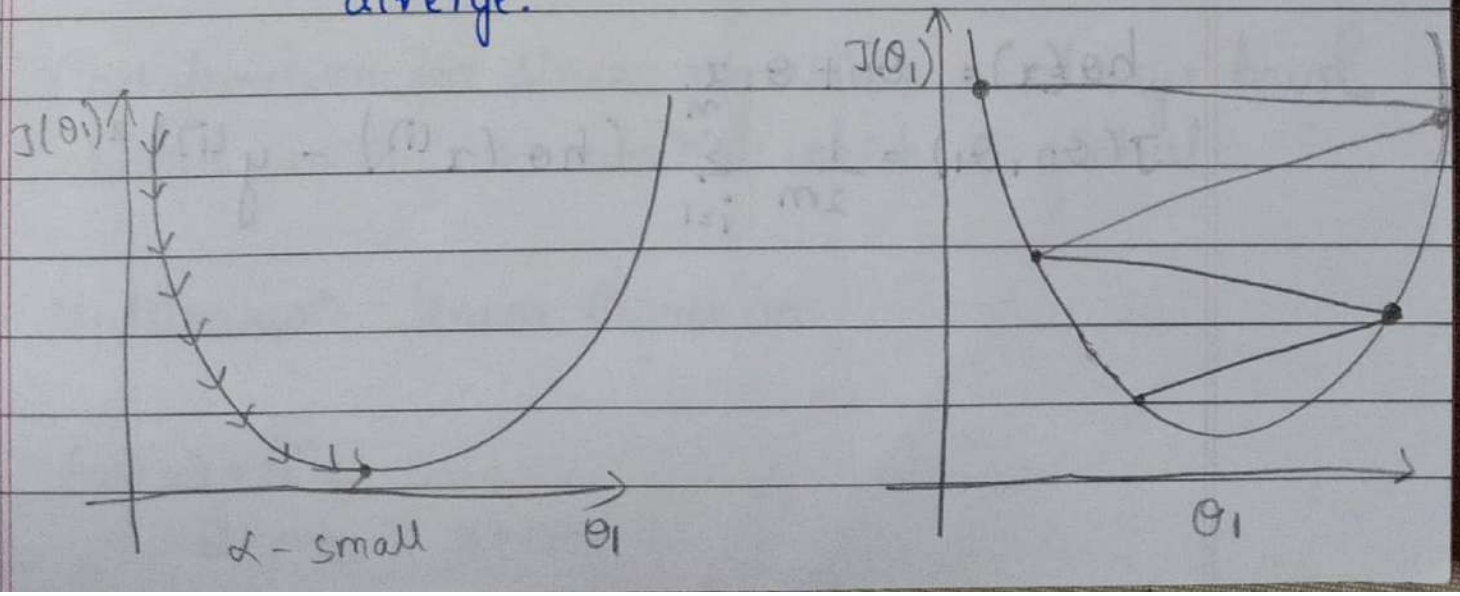
α - learning rate

→ How bigger the step is that we take downhill with creating descent.

→ If α is too large, then that corresponds to very aggressive gradient descent procedure and we might miss the actual global minima.

→ If α is too small then it takes longer time to converge.

→ If α is too large, gradient descent can overshoot the minimum. It may fail to converge or even diverge.



Gradient descent can converge to a local minimum even with the learning rate α fixed

→ As we approach a local minimum, gradient descent will automatically take smaller steps. So, no need to decrease α over time. ∵ Derivative decreases.

→ If we are at local minima then there is no update

Gradient descent on linear regression

$$\{ \theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$

$$\text{for}(j=1, j=0)$$

$$\}$$

$$h_\theta(x) = \theta_0 + \theta_1 x$$

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

$$\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) = \frac{\partial}{\partial \theta_j} \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

$$= \frac{\partial}{\partial \theta_j} \frac{1}{2m} \sum_{i=1}^{m} (\theta_0 + \theta_1 x^{(i)} - y^{(i)})^2$$

$$\theta_0 \Rightarrow j=0 : \frac{\partial}{\partial \theta_0} (J(\theta_0, \theta_1)) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})$$

$$\theta_1 \Rightarrow j=1 : \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) . x^{(i)}$$

Repeat until convergence

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) . x^{(i)}$$

Cost function for linear regression is always bowl shaped with one minimd which is global

Multivariate linear Regression

$h_\theta(x) = \theta^T x$

$\theta = n+1 \quad x = n+1$

Update rule (when $n \geq 1$)

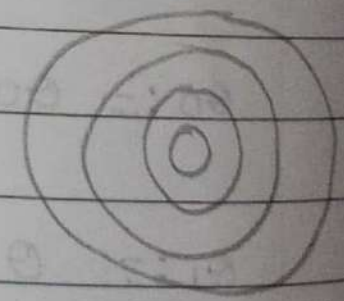
$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

How to make gradient descent faster?

1) → Feature Scaling
Make sure all features are on same scale

→ skinnier, more time to converge

$$f_1 = \frac{f_1}{(f_1 max)}$$



Get every feature $-1 \leq x_i \leq 1$ range.
$-3 \leq x_i \leq 3$

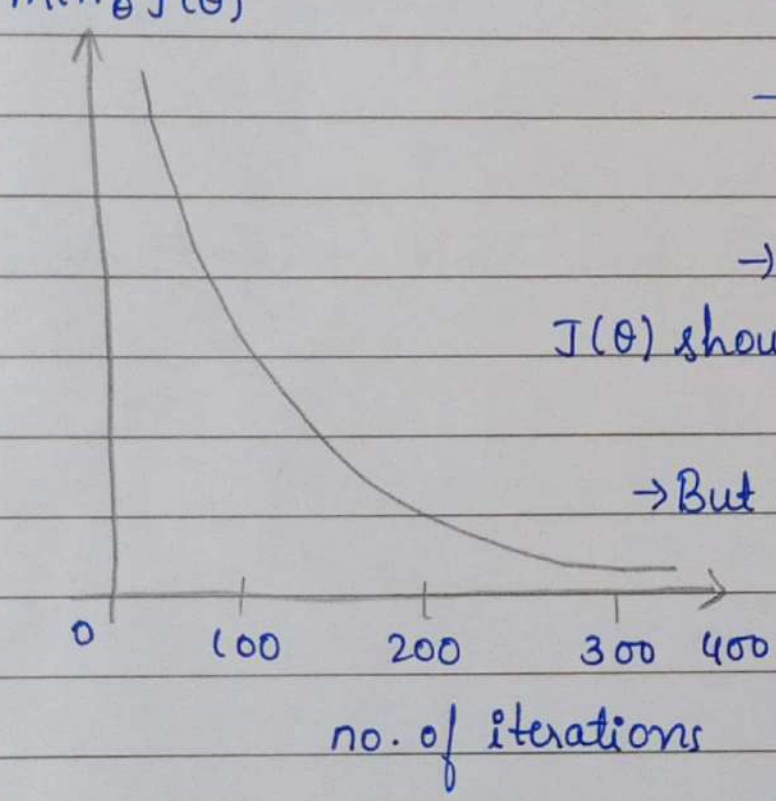
Mean normalization: $x_0 = 1$

$$x_1 = \frac{\text{size/value} - \text{mean}}{\text{max}}$$

$$x_1 \leftarrow \frac{x_1 - \mu_1}{s_1 \text{ (range)}}$$

2) Learning Rate

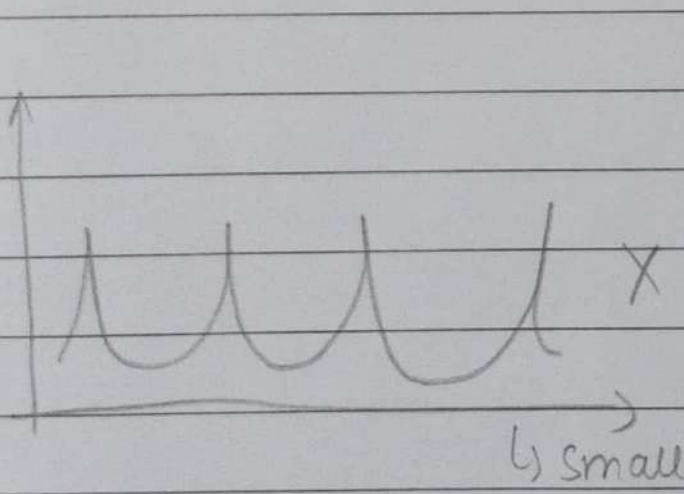
Plot cost v/s iterations.

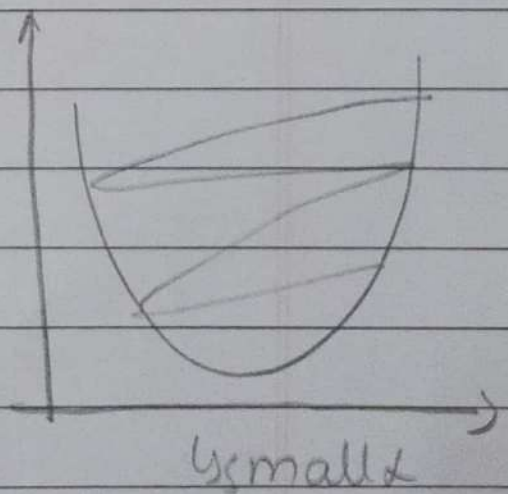


→ $J(\theta)$ should decrease after every time.

→ For sufficiently small α, $J(\theta)$ should decrease on every iteration

→ But if α is too small, gd can be slow to converge





Linear Regression

$h_\theta(x) = \theta^T x.$

$h_\theta(x) = \theta_0 + \theta_1 x.$

$$\text{Cost} = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

Logistic Regression

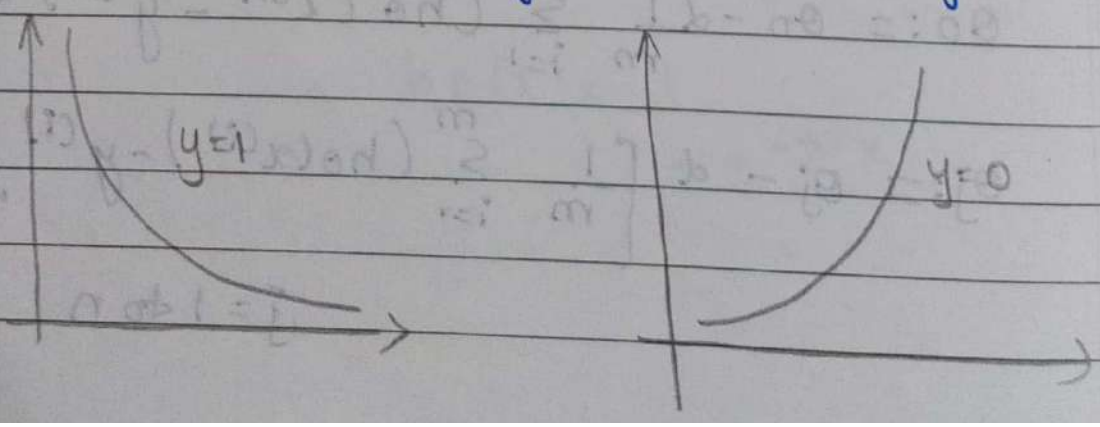
$$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}}.$$

Cost function:

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \text{Cost}\left(h_\theta(x^{(i)}), y^{(i)}\right)$$

$\text{Cost}(h_\theta(x), y) = -\log(h_\theta(x))$ if $y = 1$

$\text{Cost}(h_\theta(x), y) = -\log(1 - h_\theta(x))$ if $y = 0$



$$cost(h_\theta(x), y) = -y \log(h_\theta(x)) - (1-y) \log(1-h_\theta(x))$$

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

Gradient descent

$$\theta_j : \theta_j - \frac{\alpha}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

Regularization

$$J(\theta) = \frac{1}{2m} \left[\sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^{n} \theta_j^2 \right]$$

Gradient descent for linear regression with regularization

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)} - y^{(i)}) x_0^{(i)}$$

$$\theta_j := \theta_j - \alpha \left[\frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} + \frac{\lambda}{m} \theta_j \right]$$

$j = 1$ to n

$$\theta_j := \theta_j\left(1 - \alpha\frac{\lambda}{m}\right) - \alpha\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)x_j^{(i)}$$

$$1 - \alpha\frac{\lambda}{m} < 1$$

Cost function of logistic regression with regularization.

$$J(\theta) = \left[\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log\left(1 - h_\theta(x^{(i)})\right)\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

GD of logistic with regularization

$$\theta_0 : \theta_0 - \alpha\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)x_0^{(i)}$$

$$\theta_j : \theta_j - \alpha\left[\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)\cdot x_j^{(i)} + \frac{\lambda\,\theta_j}{m}\right]$$

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

* LINEAR REGRESSION :

- Hypothesis : $h_\theta(x) = \theta_0 + \theta_1 x$
- Cost function : choose θ_0 & θ_1 so that $h_\theta(x)$ is close to y for our traing examples (x, y). also known as error

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

- Gradient Descent : Algorithm to minimize θ_0, θ_1.
  - It minimizes the cost function

repeat {

$$\text{temp0} := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$$

$$\text{temp1} := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$

$\theta_0 := \text{temp0}$

$\theta_1 := \text{temp1}$

}

for i in range (m):

$$\theta_j = \theta_j - \alpha (\hat{y}^i - y^i) x_j^i$$

Stochastic Gradient Descent :-

- It is a variation of gradient descent algorithm that calculates error & updates the model for each example in the training dataset.

- It processes 1 example per iteration. Hence, the parameters are being updated even after one iteration in which only a single example has been processed. Thus, this is quite faster than batch gradient descent.

- But again, when the no. of training examples is large, even then it processes only one example which can be an additional overhead for the system as the no. of iterations will be large.

Algorithm:

Let $x^{(i)}, y^{(i)}$ be the training example

$$\text{cost}(\theta, (x^{(i)}, y^{(i)})) = \frac{1}{2} \sum (h_\theta(x^{(i)}) - y^{(i)})^2$$

$$J_{train}(\theta) = \frac{1}{m} \sum \text{cost}(\theta, (x^{(i)}, y^{(i)}))$$

For i = 1 to m, repeat {

$$\theta_j = \theta_j - (\text{learning rate}) * \sum (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

for j = 0 to n

}

Batch Gradient Descent:

- It is a variation of gradient descent algorithm that calculates the error for each example in the training dataset, but only updates the model after all training examples have been evaluated.

- It processes all the training examples for each iteration. But if the no. of training examples is large, then batch gradient descent is computationally expensive.

- Thus, if the no. of train examples is large, we do not prefer batch gradient descent.

Algorithm:-

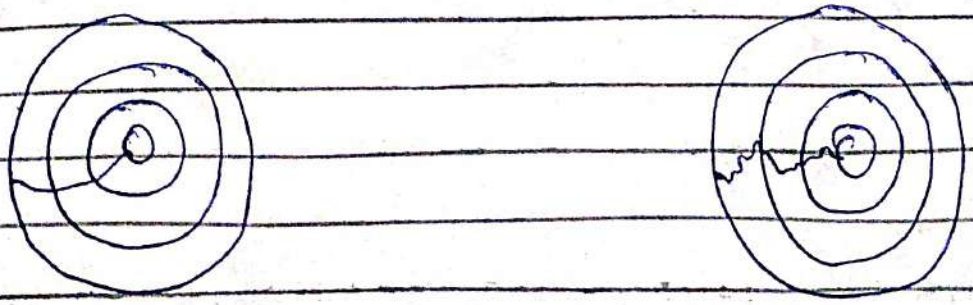
$$J_{train}(\theta) = \frac{1}{2m} \sum (h_\theta(x^{(i)}) - y^{(i)})^2$$

Repeat {

$$\theta_j = \theta_j - (\text{learningrate}/m) * \sum (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

for every $j = 0$ to n

}



BGD SGD

* Early Stopping :-

- Early stopping mechanism is a form of regularization in ML to avoid overfitting of the model.

- During training, the model is evaluated on a validation dataset after each epoch.
  If the performance of the model on the validation dataset starts to degrade i.e. loss begins to increase or accuracy begins to decrease then the training process is stopped.

- The model at the time training is stopped, is then used & is known to have good generalization process.

- This procedure is called "early stopping".

- Early stopping rules provide guidance as to how many iterations can be run before the learner begins to over-fit.

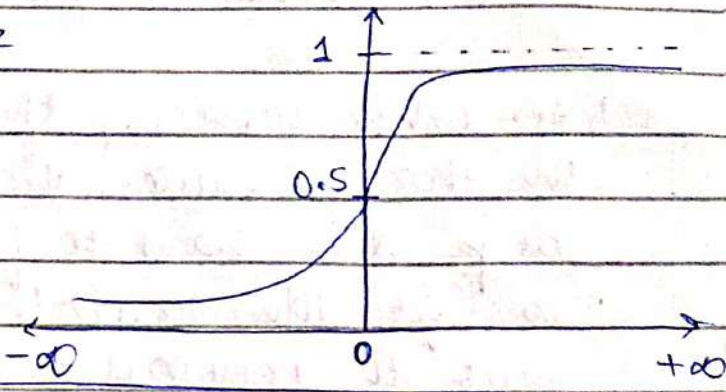
★ Logistic Regression :

- **hypothesis :-** $h_\theta(x) = g(\theta^T x) = \frac{1}{1+e^{-\theta^T x}}$

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

$g(z) \to 1$ as $z \to \infty$

$g(z) \to 0$ as $z \to -\infty$



- **cost function :-**

$$\text{Cost}(h_\theta(x), y) = -y \log(h_\theta(x)) + (1-y) \log(1-h_\theta(x))$$

- **Gradient Descent :-**

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

repeat :

{

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

}

★ Regularization :-

$$J(\theta) = \frac{1}{2m} \left[\sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2 + \lambda \sum_{j=1}^{m} \theta_j^2 \right]$$

λ → reg. parameter

* Overfitting :-

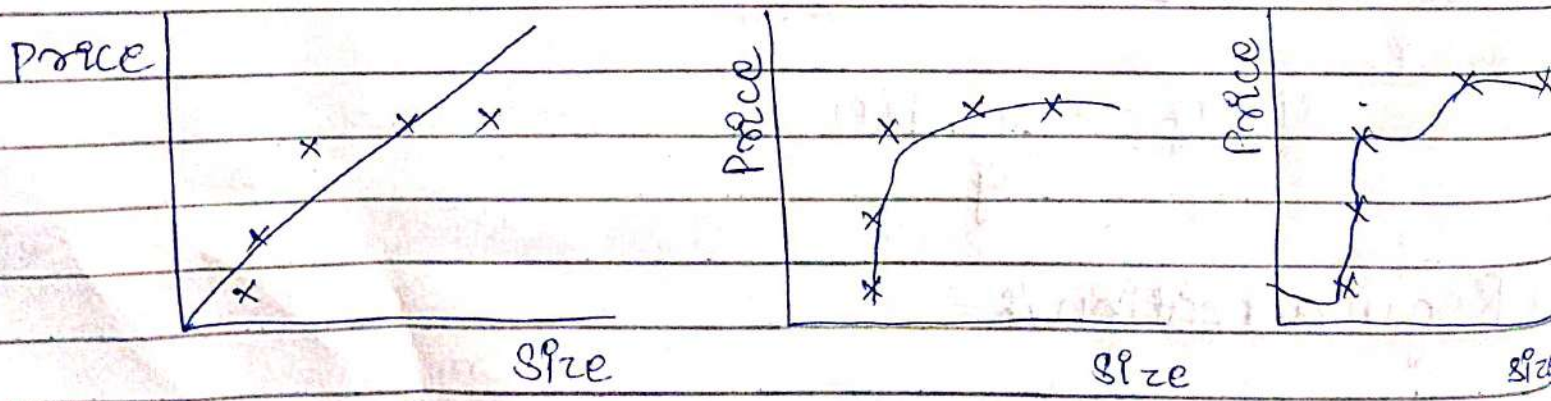
- Overfitting is a condition where a statistical model begins to describe the random error in the data, rather than the relationships b/w variabl

- In other words, if we have too many feature in the formula, the learned hypothesis will try very hard to find the decision boundary to fit the training data set well but it fails to generalize to make the accurate predictions on new unseen examples.

- It fails to generalize, how well a hypothesis applies to a new example.

- Thus, the test ~~data~~ error is very high.

→ Linear Regression :-

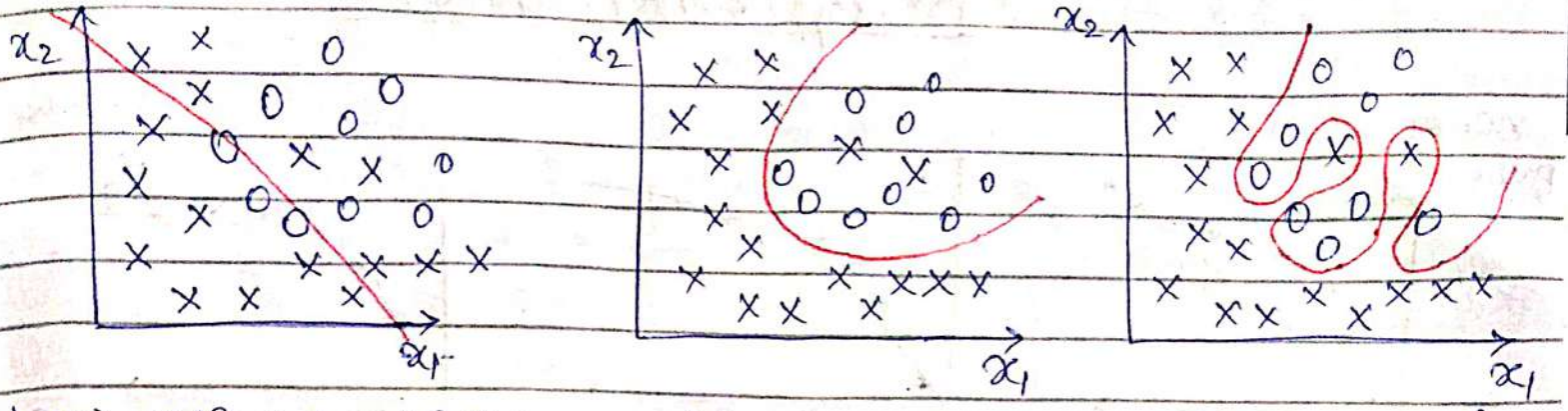


- Underfit
- (High bias)
- $\theta_0 + \theta_1 x$

- Best fit
- $\theta_0 + \theta_1 x + \theta_2 x^2$

Overfit
(High va
$\theta_0 + \theta_1 x + \theta_2 x^2$
$+ \theta_3 x^3 + \theta_4 x^4$

→ Logistic Regression :-



$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$
↳ sigmoid fun.
Underfit

$g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 + \theta_5 x_1 x_2)$

$g(\theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^2 x_2 + \theta_4 x_1^2 x_2^2 + \theta_5 x_1^2 x_2^3 + \ldots)$
Overfit

- **Cause:-** Overfitting occurs due to
  - large feature set.
  - large no. of parameters.
- **How to fix Overfitting?**

1). **Reduce the no. of features**
- manually select which features to keep.
- model select algorithm (automatically select which features to keep, which features to throw out).

2). **Regularization**
- keep all the features but reduce the magnitude or value of parameters (θ_j) to make the value smaller.
- It works well when we have a lot of features, each of which contributes a bit to predicting y.