Machine Learning (19CSE305)

Backpropagation

Dr. Peeta Basa Pati
Ms. Priyanka V
Department of Computer Science & Engineering,
Amrita School of Engineering, Bengaluru

Topics

- Backpropagation

Neural Networks

- Artificial neural network (ANN) is a machine learning approach that models human brain and consists of a number of artificial neurons.
- Each neuron in ANN receives a number of inputs.
- An activation function is applied to these inputs which results in activation level of neuron (output value of the neuron).
- Knowledge about the learning task is given in the form of examples called training examples.

Contd..

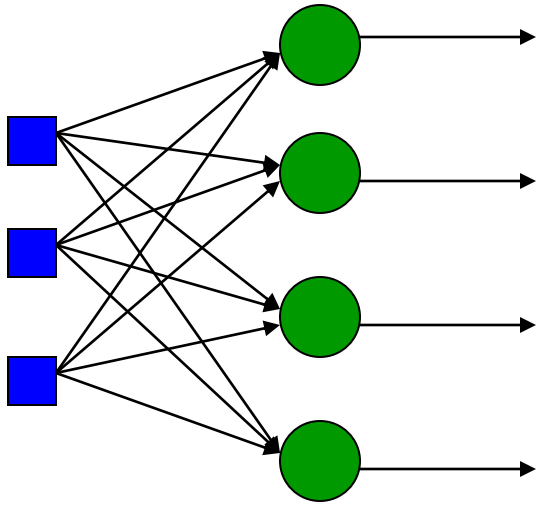
- An Artificial Neural Network is specified by:
  - **neuron model**: the information processing unit of the NN,
  - **an architecture**: a set of neurons and links connecting neurons. Each link has a weight,
  - **a learning algorithm**: used for training the NN by modifying the weights in order to model a particular learning task correctly on the training examples.
- The aim is to obtain a NN that is trained and generalizes well.
- It should behaves correctly on new instances of the learning task.

Network Architectures

- Three different classes of network architectures
  - single-layer feed-forward
  - multi-layer feed-forward
  - recurrent
- The architecture of a neural network is linked with the learning algorithm used to train

Single Layer Feed-forward

Input layer of source nodes

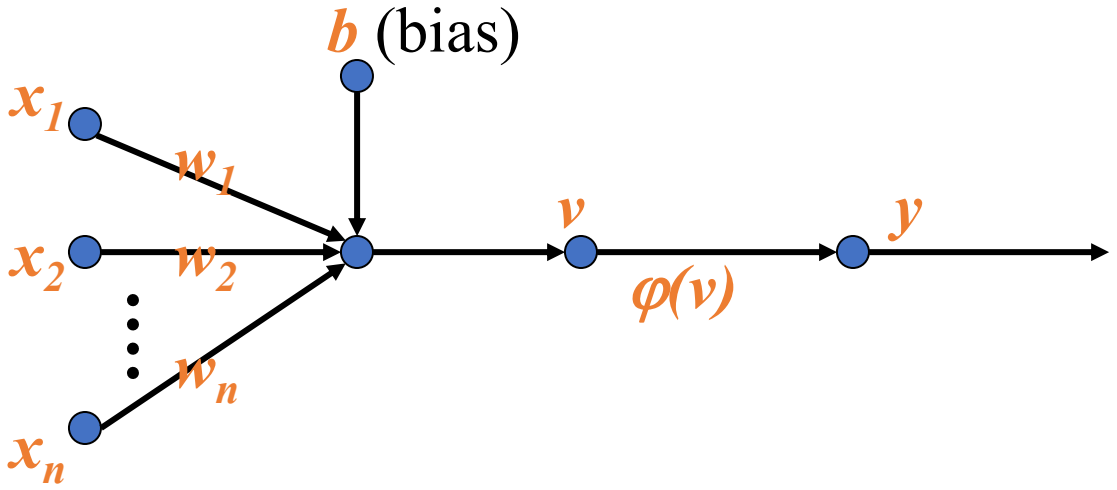


Output layer of neurons

Perceptron: Neuron Model
(Special form of single layer feed forward)

- The perceptron was first proposed by Rosenblatt (1958) is a simple neuron that is used to classify its input into one of two categories.
- A perceptron uses a **step function** that returns +1 if weighted sum of its input ≥ 0 and -1 otherwise

$$\varphi(v) = \begin{cases} +1 \text{ if } v \geq 0 \\ -1 \text{ if } v < 0 \end{cases}$$

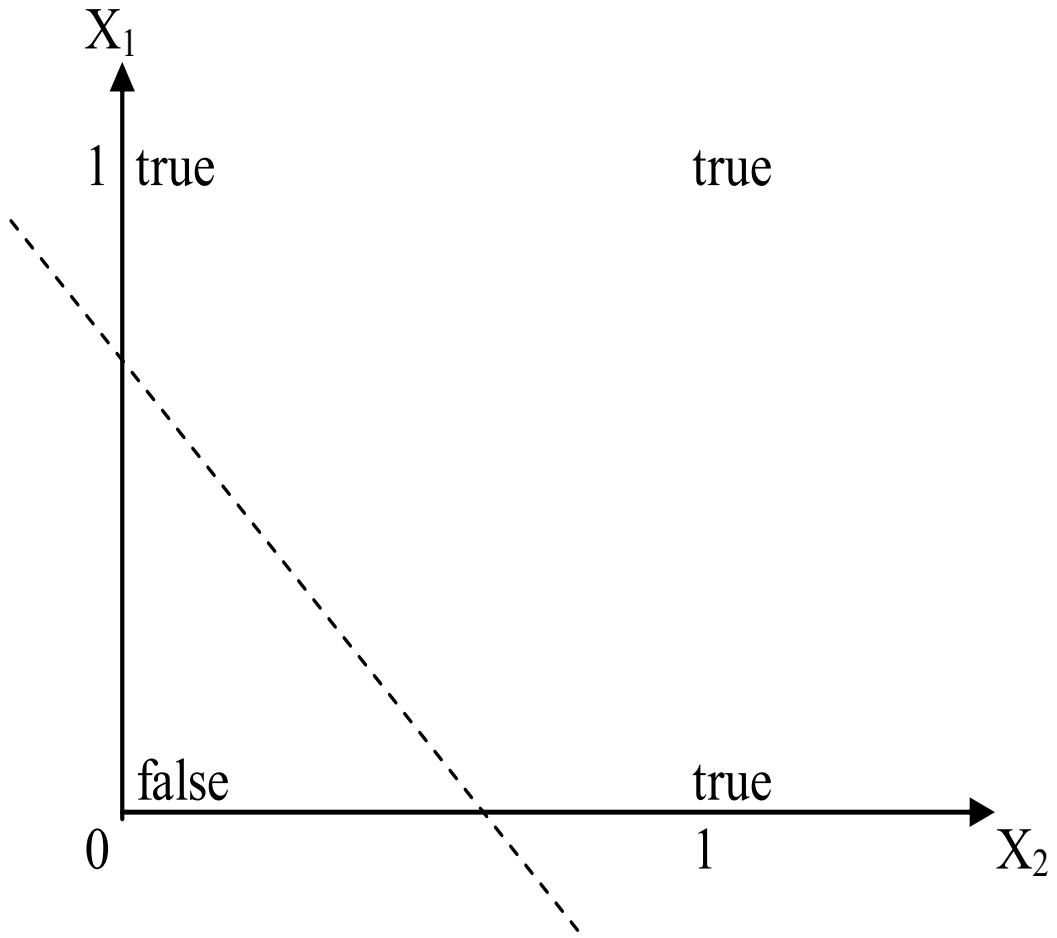


Perceptron for Classification

- used for binary classification.
- First train a perceptron for a classification task.
  - Find suitable weights in such a way that the training examples are correctly classified.
  - Geometrically try to find a hyper-plane that separates the examples of the two classes.
- can only model linearly separable classes.
- When the two classes are not linearly separable, it may be desirable to obtain a linear separator that minimizes the mean squared error.

- Given training examples of classes C_1, C_2 train the perceptron in such a way that :
  - *If the output of the perceptron is +1 then the input is assigned to class* C_1
  - *If the output is -1 then the input is assigned to* C_2

Boolean function OR – Linearly separable



Perceptron: Limitations

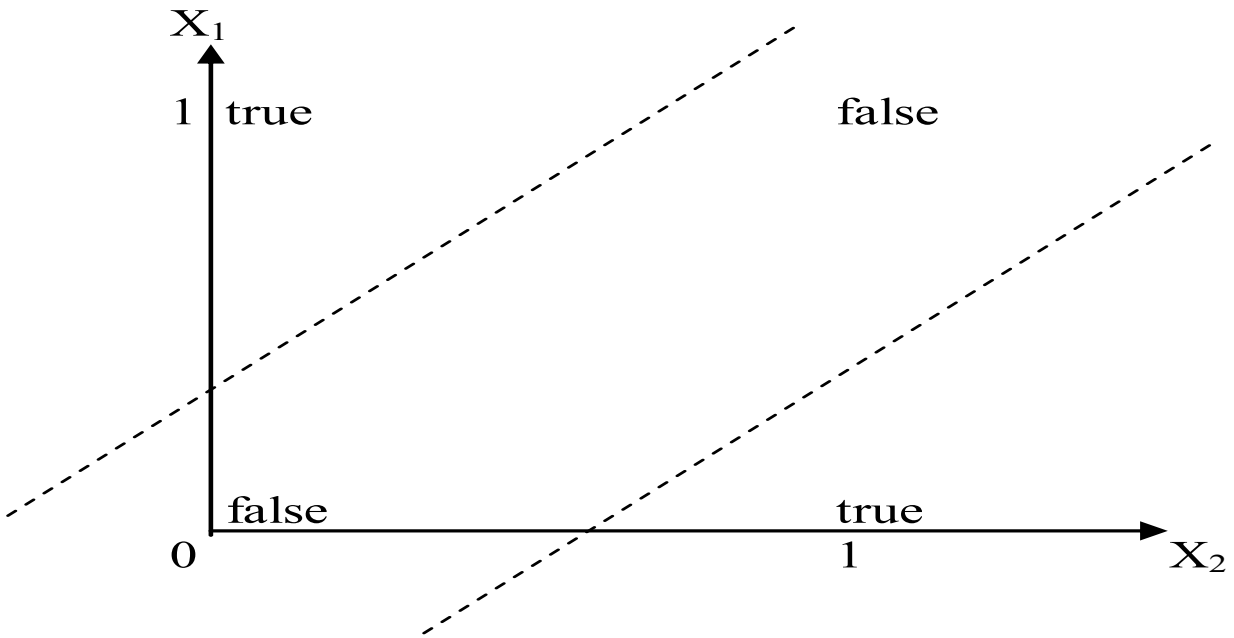
- The perceptron can only model linearly separable functions,
  - those functions which can be drawn in 2-dim graph and single straight line separates values in two part.
- Boolean functions given below are linearly separable:
  - AND
  - OR
  - COMPLEMENT
- It cannot model XOR function as it is non linearly separable.
  - When the two classes are not linearly separable, it may be desirable to obtain a linear separator that minimizes the mean squared error.

XOR – Non linearly separable function

- A typical example of non-linearly separable function is the XOR that computes the logical **exclusive or.**.
- This function takes two input arguments with values in {0,1} and returns one output in {0,1},
- Here 0 and 1 are encoding of the truth values **false** and **true**,
- The output is **true** if and only if the two inputs have different truth values.
- XOR is non linearly separable function which can not be modeled by perceptron.
- For such functions we have to use multi layer feed-forward network.

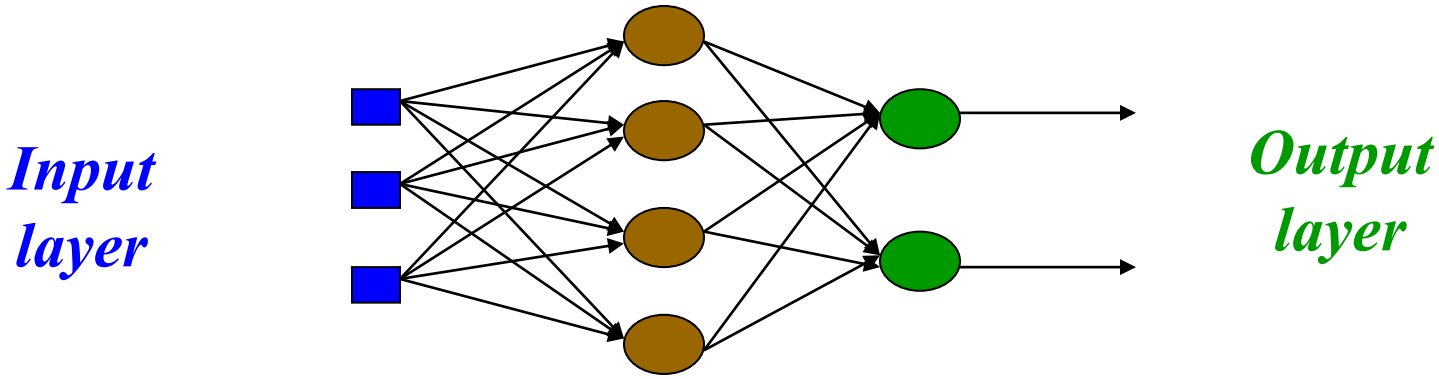
Input		Output
X_1	X_2	X_1 XOR X_2
0	0	0
0	1	1
1	0	1
1	1	0

These two classes (true and false) cannot be separated using a line. Hence XOR is non linearly separable.



Multi layer feed-forward NN (FFNN)

- FFNN is a more general network architecture, where there are hidden layers between input and output layers.
- Hidden nodes do not directly receive inputs nor send outputs to the external environment.
- FFNNs overcome the limitation of single-layer NN.
- They can handle non-linearly separable learning tasks.



3-4-2 Network

FFNN NEURON MODEL

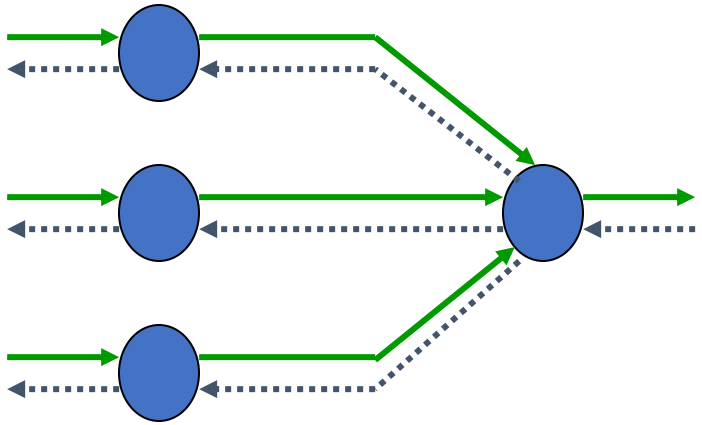
- The classical learning algorithm of FFNN is based on the gradient descent method.
- For this reason the activation function used in FFNN are continuous functions of the weights, differentiable everywhere.
- Eg. **sigmoid function**

Training Algorithm: Backpropagation

- The Backpropagation algorithm learns in the same way as single perceptron.
- It searches for weight values that minimize the total error of the network over the set of training examples (training set).
- Backpropagation consists of the repeated application of the following two passes:
  - **Forward pass**: In this step, the network is activated on one example and the error of (each neuron of) the output layer is computed.
  - **Backward pass**: in this step the network error is used for updating the weights. The error is propagated backwards from the output layer through the network layer by layer. This is done by recursively computing the local gradient of each neuron.

Backpropagation

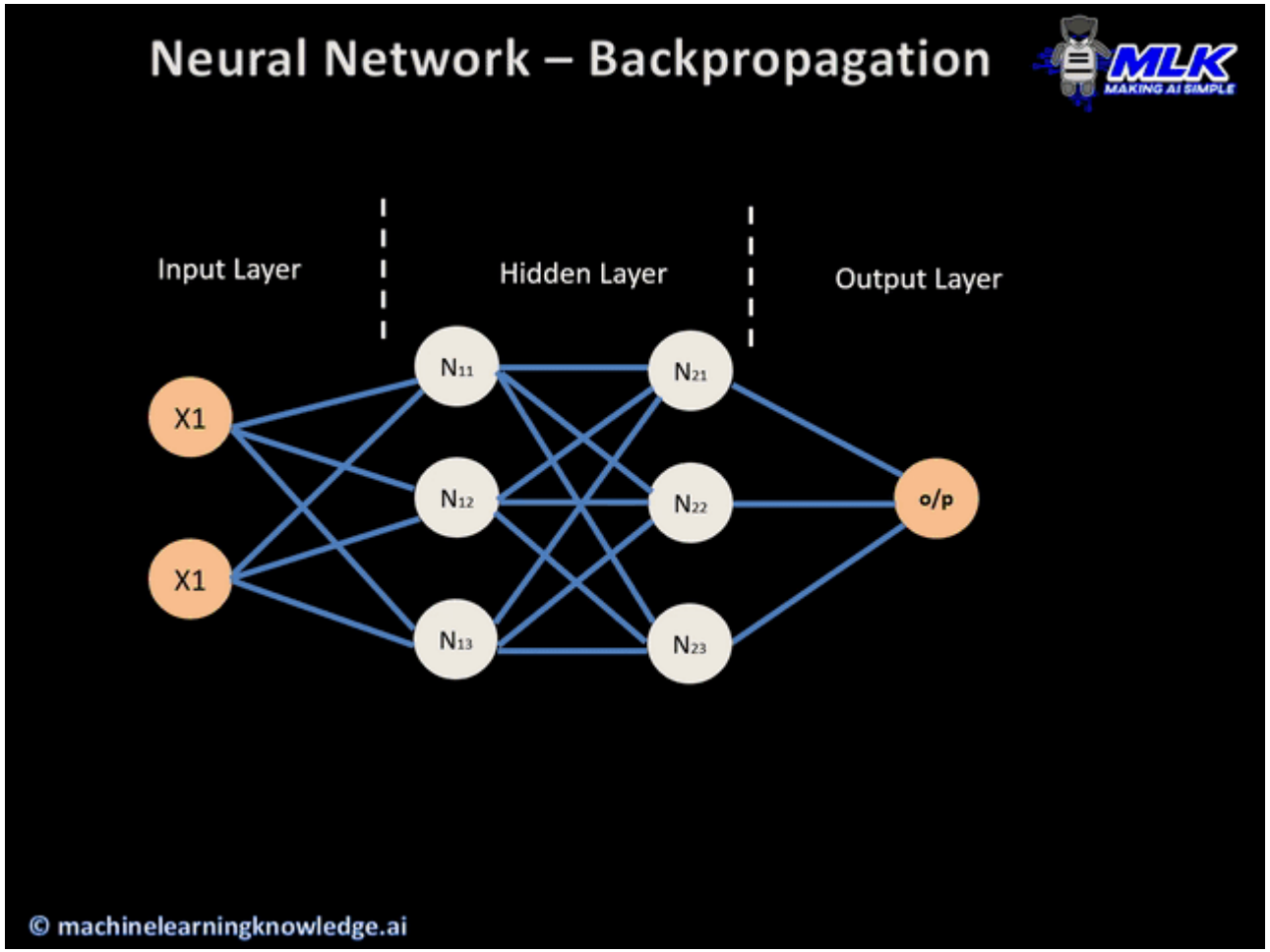
- Back-propagation training algorithm



Network activation
Forward Step

Error propagation
Backward Step

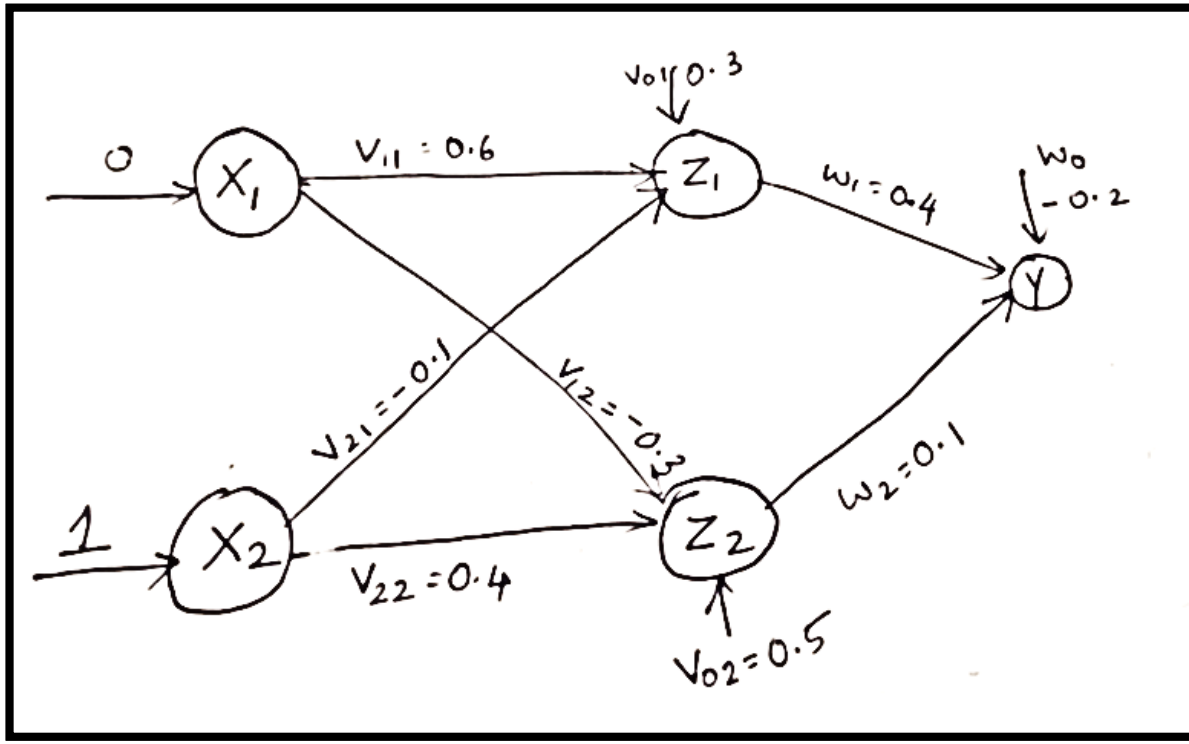
- Backpropagation adjusts the weights of the NN in order to minimize the network total mean squared error.


Neural Network – Backpropagation
MLK
MAKING AI SIMPLE
Input Layer
Hidden Layer
Output Layer
X1
X1
N11
N12
N13
N21
N22
N23
o/p
© machinelearningknowledge.ai

- Using back-propagation network, find the weights for the net. It is presented with the input pattern [0, I] and the target output is I. Use a learning rate α = 0.25 and sigmoidal activation function. initial weights are $[v_{11}\ \ v_{21}\ \ v_{01}]$ = [0.6 -0.1 0.3]

$[v_{12}\ v_{22}\ v_{02}] = [-0.3\ 0.4\ 0.5]$ and $[w_1\ w_2\ w_0] = [0.4$
$0.1\ -0.2]$, and the learning rate is $\alpha = 0.25$. Acti-
vation function used is binary sigmoidal activation
function and is given by

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



Calculate the net input: For z_1 layer

$$z_{in1} = v_{01} + x_1 v_{11} + x_2 v_{21}$$
$$= 0.3 + 0 \times 0.6 + 1 \times -0.1 = 0.2$$

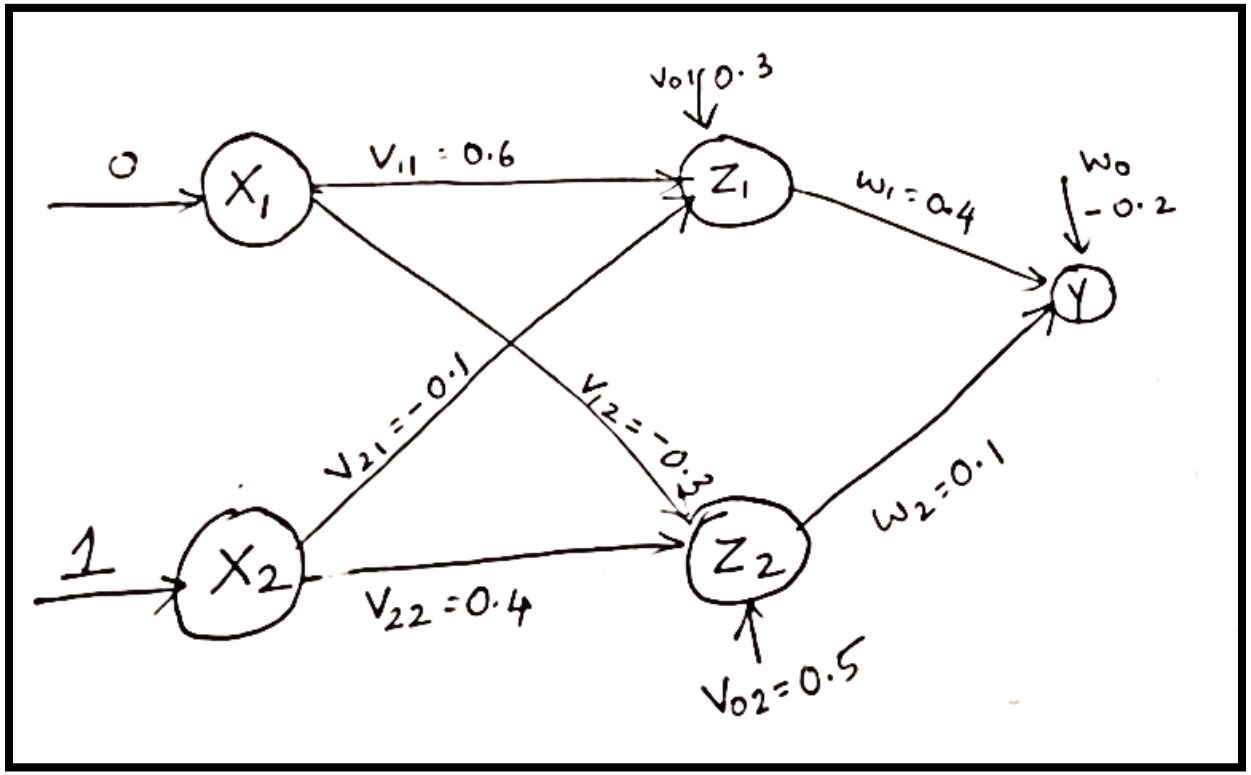
For z_2 layer

$$z_{in2} = v_{02} + x_1 v_{12} + x_2 v_{22}$$
$$= 0.5 + 0 \times -0.3 + 1 \times 0.4 = 0.9$$

Applying activation to calculate the output, we obtain

$$z_1 = f(z_{in1}) = \frac{1}{1 + e^{-z_{in1}}} = \frac{1}{1 + e^{-0.2}} = 0.5498$$

$$z_2 = f(z_{in2}) = \frac{1}{1 + e^{-z_{in2}}} = \frac{1}{1 + e^{-0.9}} = 0.7109$$

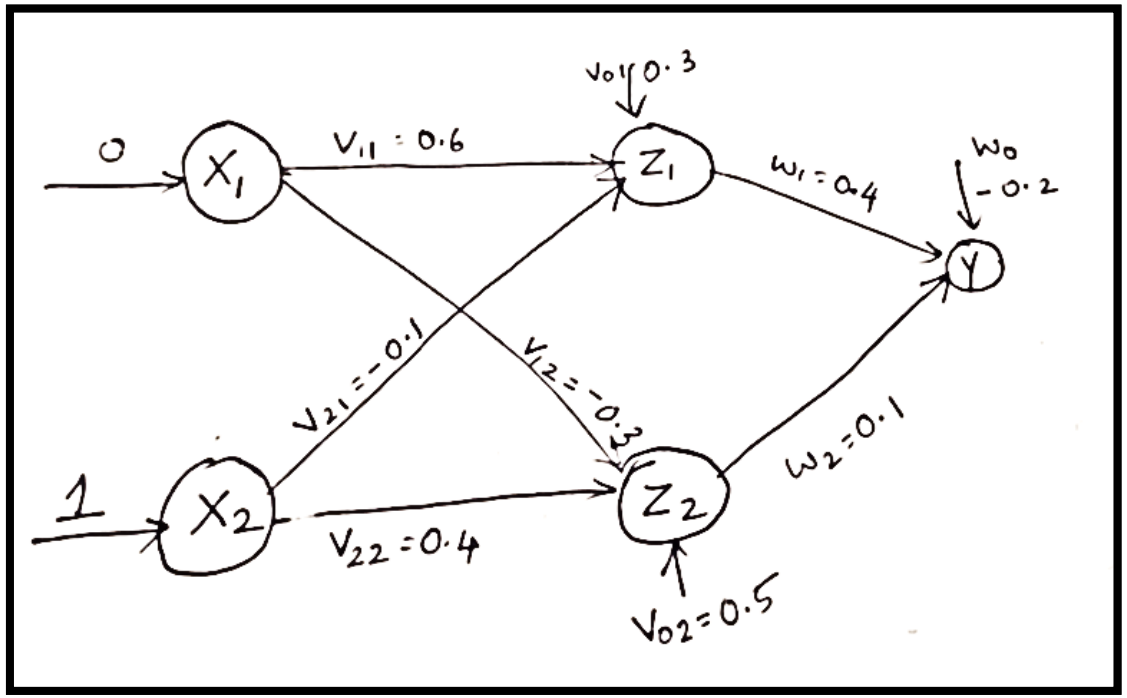


Calculate the net input entering the output layer.
For y layer

$$y_{in} = w_0 + z_1 w_1 + z_2 w_2$$
$$= -0.2 + 0.5498 \times 0.4 + 0.7109 \times 0.1$$
$$= 0.09101$$

Applying activations to calculate the output, we obtain

$$y = f(y_{in}) = \frac{1}{1 + e^{-y_{in}}} = \frac{1}{1 + e^{-0.09101}} = 0.5227$$



- Compute the error portion δ_k:

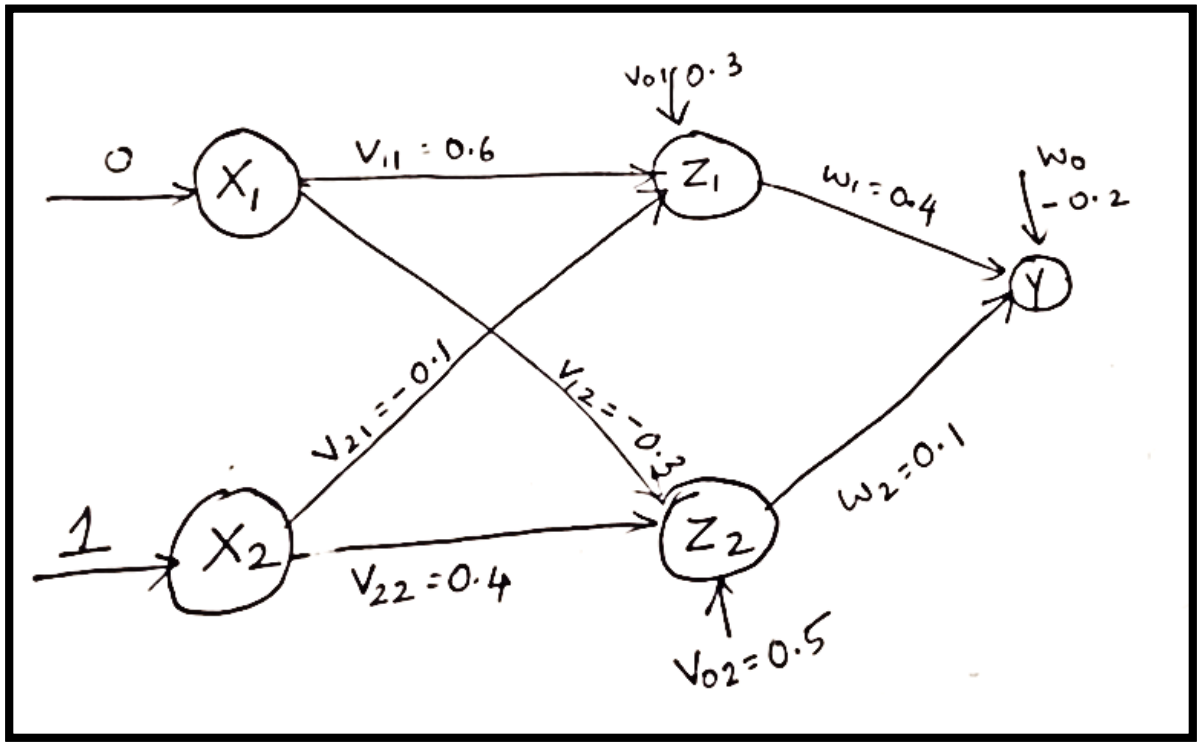
$$\delta_k = (t_k - y_k) f'(y_{ink})$$

Now

$$f'(y_{in}) = f(y_{in})[1 - f(y_{in})] = 0.5227[1 - 0.5227]$$

$$f'(y_{in}) = 0.2495$$

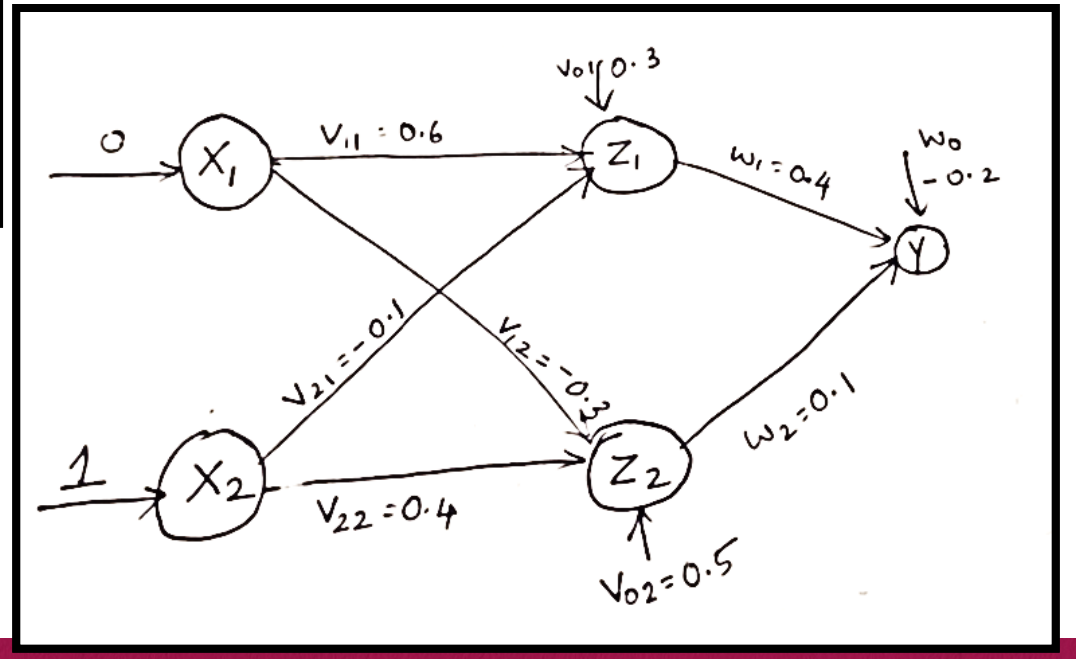
$$\delta_1 = (1 - 0.5227)\,(0.2495) = 0.1191$$



$$\Delta w_1 = \alpha \delta_1 z_1 = 0.25 \times 0.1191 \times 0.5498$$
$$= 0.0164$$

$$\Delta w_2 = \alpha \delta_1 z_2 = 0.25 \times 0.1191 \times 0.7109$$
$$= 0.02117$$

$$\Delta w_0 = \alpha \delta_1 = 0.25 \times 0.1191 = 0.02978$$



- Compute the error portion δ_j between input and hidden layer ($j = 1$ to 2):

$$\delta_j = \delta_{inj} f'(z_{inj})$$

$$\delta_{inj} = \sum_{k=1}^{m} \delta_k w_{jk}$$

$$\delta_{inj} = \delta_1 w_{j1} \quad [\because \text{only one output neuron}]$$

$$\Rightarrow \delta_{in1} = \delta_1 w_{11} = 0.1191 \times 0.4 = 0.04764$$

$$\Rightarrow \delta_{in2} = \delta_1 w_{21} = 0.1191 \times 0.1 = 0.01191$$

Error, $\delta_1 = \delta_{in1} f'(z_{in1})$

$$f'(z_{in1}) = f(z_{in1})[1 - f(z_{in1})]$$
$$= 0.5498[1 - 0.5498] = 0.2475$$
$$\delta_1 = \delta_{in1} f'(z_{in1})$$
$$= 0.04764 \times 0.2475 = 0.0118$$

Error, $\delta_2 = \delta_{in2} f'(z_{in2})$

$$f'(z_{in2}) = f(z_{in2})[1 - f(z_{in2})]$$
$$= 0.7109[1 - 0.7109] = 0.2055$$
$$\delta_2 = \delta_{in2} f'(z_{in2})$$
$$= 0.01191 \times 0.2055 = 0.00245$$

Now find the changes in weights between input and hidden layer:

$$\Delta v_{11} = \alpha \delta_1 x_1 = 0.25 \times 0.0118 \times 0 = 0$$

$$\Delta v_{21} = \alpha \delta_1 x_2 = 0.25 \times 0.0118 \times 1 = 0.00295$$

$$\Delta v_{01} = \alpha \delta_1 = 0.25 \times 0.0118 = 0.00295$$

$$\Delta v_{12} = \alpha \delta_2 x_1 = 0.25 \times 0.00245 \times 0 = 0$$

$$\Delta v_{22} = \alpha \delta_2 x_2 = 0.25 \times 0.00245 \times 1 = 0.0006125$$

$$\Delta v_{02} = \alpha \delta_2 = 0.25 \times 0.00245 = 0.0006125$$

- Compute the final weights of the network:

$$v_{11}(\text{new}) = v_{11}(\text{old}) + \Delta v_{11} = 0.6 + 0 = 0.6$$

$$v_{12}(\text{new}) = v_{12}(\text{old}) + \Delta v_{12} = -0.3 + 0 = -0.3$$

$$v_{21}(\text{new}) = v_{21}(\text{old}) + \Delta v_{21}$$
$$= -0.1 + 0.00295 = -0.09705$$

$$v_{22}(\text{new}) = v_{22}(\text{old}) + \Delta v_{22}$$
$$= 0.4 + 0.0006125 = 0.4006125$$

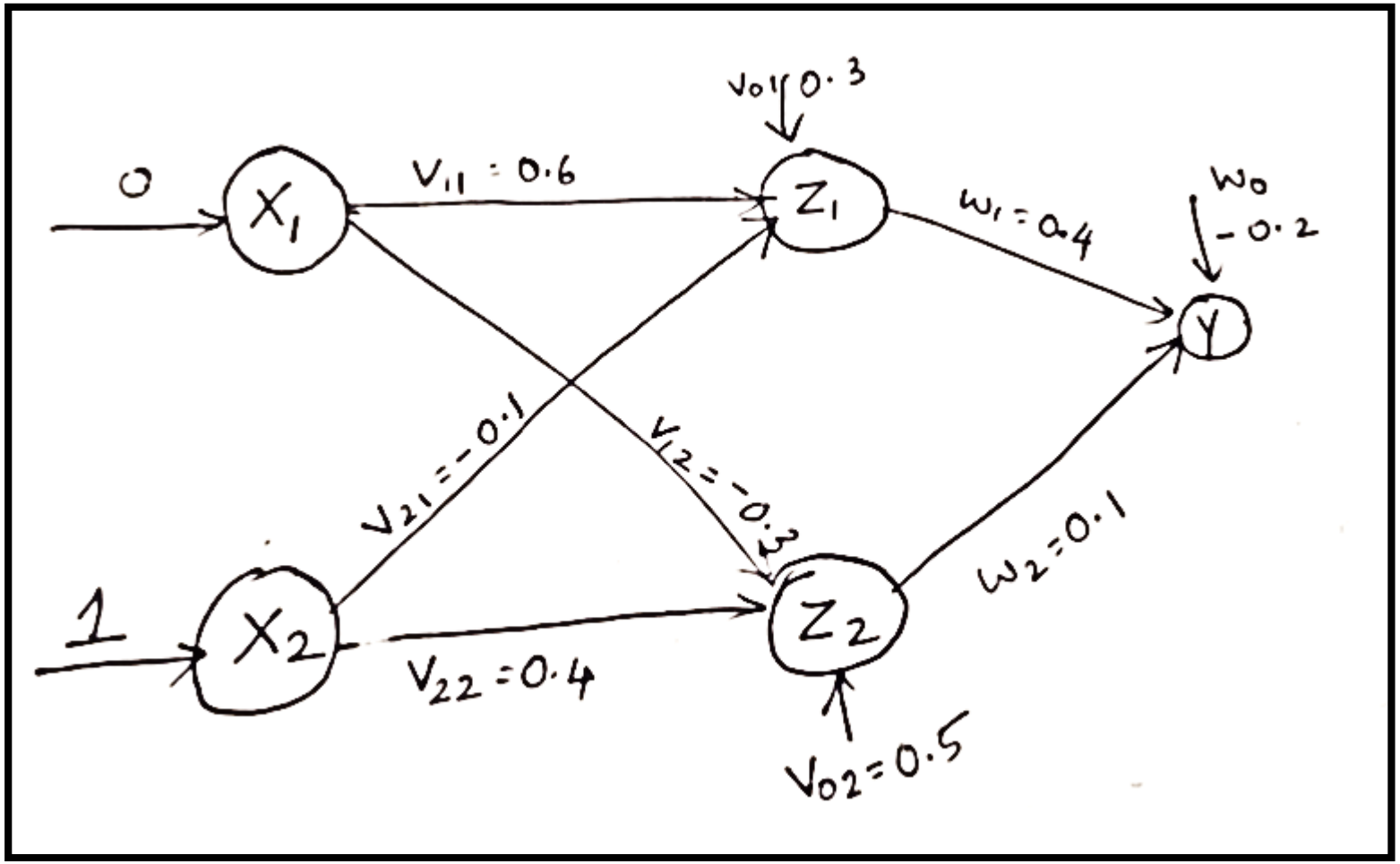
$$w_1(\text{new}) = w_1(\text{old}) + \Delta w_1 = 0.4 + 0.0164$$
$$= 0.4164$$

$$w_2(\text{new}) = w_2(\text{old}) + \Delta w_2 = 0.1 + 0.02117$$
$$= 0.12117$$

$$v_{01}(\text{new}) = v_{01}(\text{old}) + \Delta v_{01} = 0.3 + 0.00295$$
$$= 0.30295$$

$$v_{02}(\text{new}) = v_{02}(\text{old}) + \Delta v_{02}$$
$$= 0.5 + 0.0006125 = 0.5006125$$

$$w_0(\text{new}) = w_0(\text{old}) + \Delta w_0 = -0.2 + 0.02978$$
$$= -0.17022$$



Thank you !!!!!

Machine Learning (19CSE305)

Perceptron -Training

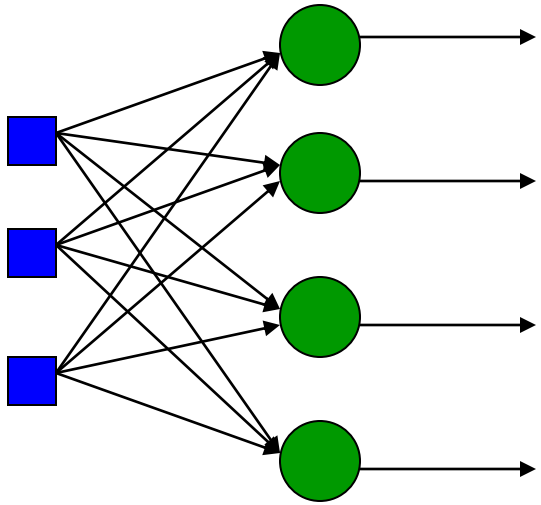
Dr. Peeta Basa Pati
Ms. Priyanka V
Department of Computer Science & Engineering,
Amrita School of Engineering, Bengaluru

Topics

- Percpetron

Single Layer Feed-forward

Input layer of source nodes

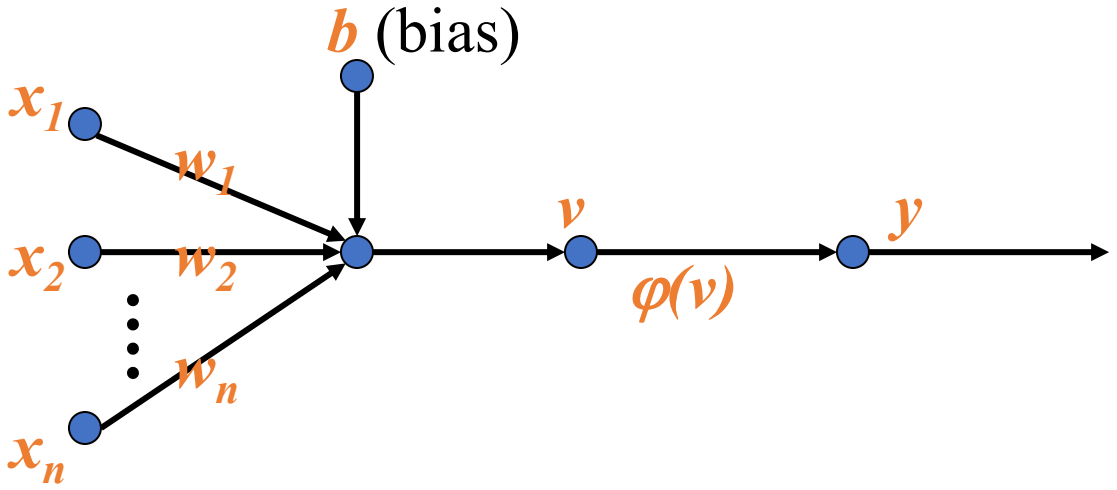


Output layer of neurons

Perceptron: Neuron Model
(Special form of single layer feed forward)

- The perceptron was first proposed by Rosenblatt (1958) is a simple neuron that is used to classify its input into one of two categories.
- A perceptron uses a **step function** that returns +1 if weighted sum of its input ≥ 0 and -1 otherwise

$$\varphi(v) = \begin{cases} +1 \text{ if } v \geq 0 \\ -1 \text{ if } v < 0 \end{cases}$$



Perceptron for Classification

- used for binary classification.
- First train a perceptron for a classification task.
  - Find suitable weights in such a way that the training examples are correctly classified.
  - Geometrically try to find a hyper-plane that separates the examples of the two classes.
- can only model linearly separable classes.
- When the two classes are not linearly separable, it may be desirable to obtain a linear separator that minimizes the mean squared error.

- Given training examples of classes C_1, C_2 train the perceptron in such a way that :
  - *If the output of the perceptron is +1 then the input is assigned to class* C_1
  - *If the output is -1 then the input is assigned to* C_2

Perceptron training

Step 0: Initialize the weights and the bias (for easy calculation they can be set to zero). Also initialize the learning rate $\alpha (0 < \alpha \leq 1)$. For simplicity α is set to 1.

Step 1: Perform Steps 2–6 until the final stopping condition is false.

Step 2: Perform Steps 3–5 for each training pair indicated by *s:t*.

Step 3: The input layer containing input units is applied with identity activation functions:

$$x_i = s_i$$

Step 4: Calculate the output of the network. To do so, first obtain the net input:

$$y_{in} = b + \sum_{i=1}^{n} x_i w_i$$

where "n" is the number of input neurons in the input layer. Then apply activations over the net input calculated to obtain the output:

$$y = f(y_{in}) = \begin{cases} 1 & \text{if } y_{in} > \theta \\ 0 & \text{if } -\theta \leq y_{in} \leq \theta \\ -1 & \text{if } y_{in} < -\theta \end{cases}$$

Step 5: *Weight and bias adjustment:* Compare the value of the actual (calculated) output and desired (target) output.

If $y \neq t$, then

$$w_i(\text{new}) = w_i(\text{old}) + \alpha t x_i$$
$$b(\text{new}) = b(\text{old}) + \alpha t$$

else, we have

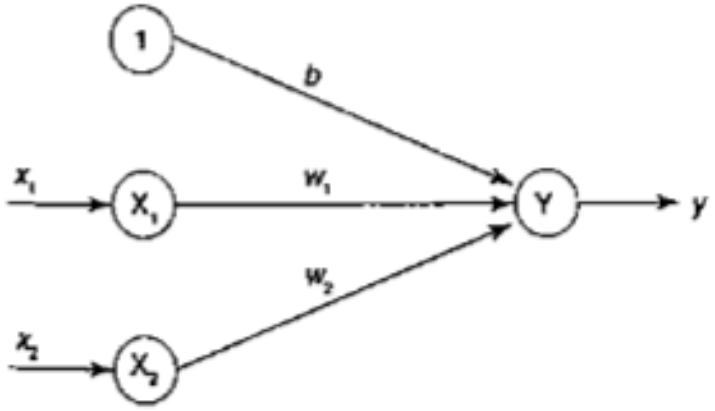
$$w_i(\text{new}) = w_i(\text{old})$$
$$b(\text{new}) = b(\text{old})$$

Step 6: Train the network until there is no weight change. This is the stopping condition for the network. If this condition is not met, then start again from Step 2.

- Implement AND function using perceptron networks for `bipolar inputs &` targets.
- The initial weights and threshold are set to zero, $w_1 = w_2 = b = 0$ and $\theta = 0$.
- The learning rate is set equal to 1.

Table 1

x_1	x_2	t
1	1	1
1	−1	−1
−1	1	−1
−1	−1	−1



$$y = f(y_{in}) = \begin{cases} 1 & \text{if } y_{in} > 0 \\ 0 & \text{if } y_{in} = 0 \\ -1 & \text{if } y_{in} < 0 \end{cases}$$

Epoch 1

Input 1

$x_1 = 1,\ x_2 = 1,\ t = 1$ $\qquad w_1 = 0,\ w_2 = 0,\ b = 0.$

$\alpha = 1$

$y_{in} = b + x_1 w_1 + x_2 w_2$

$= 0 + 1 \times 0 + 1 \times 0 = 0$

$y = f(y_{in}) = 0$

$t = 1,\ y = 0 \quad \therefore t \neq y$

So update weights

$b_{new} = b_{old} + \alpha t$

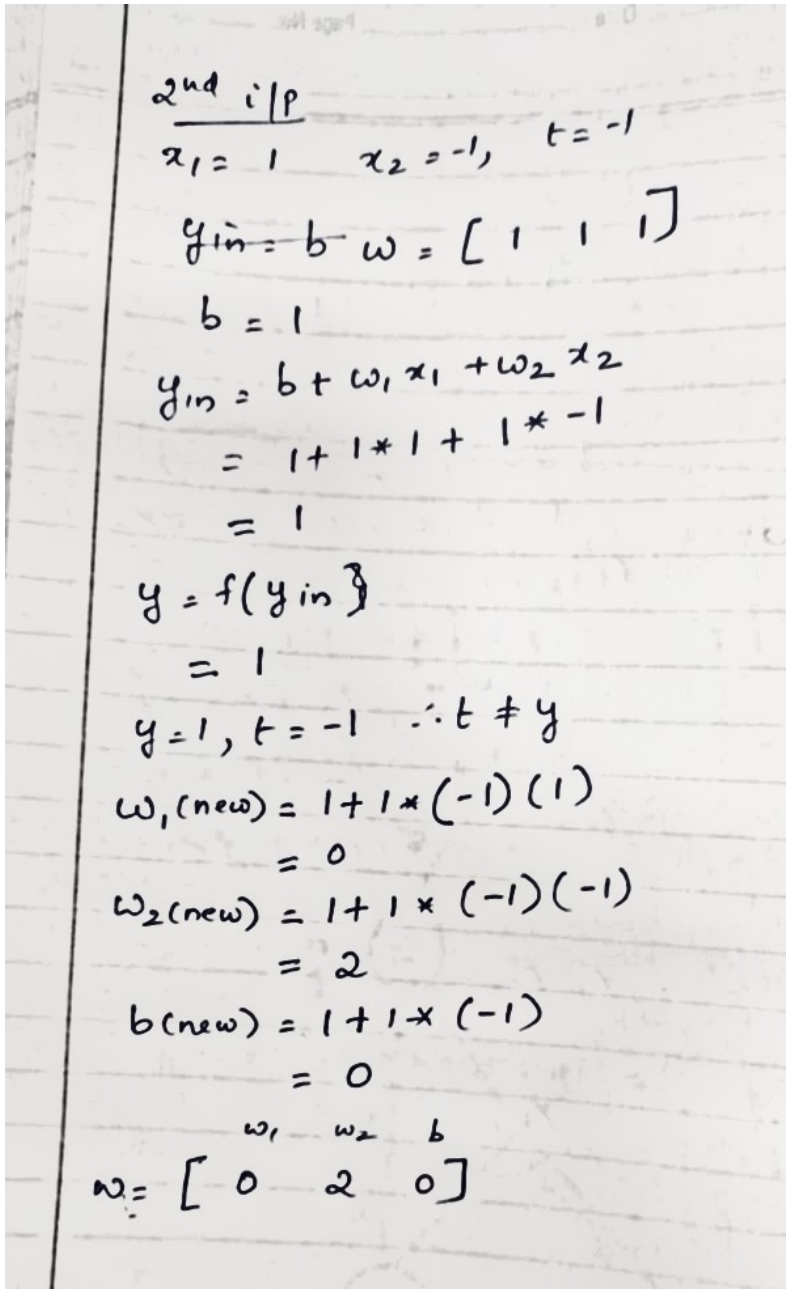
$= 0 + 1 * 1 = 1$

$w_{1new} = w_{1old} + \alpha t\, x_1$

$= 0 + 1 * 1 * 1 = 1$

$w_{2new} = w_{2old} + \alpha t\, x_2$

$= 0 + 1 * 1 * 1 = 1$



3rd i/p

$x_1 = -1, \; x_2 = 1 \quad t = -1$

$w_1 = 0 \quad w_2 = 2, \; b = 0$

$y_{in} = 0 + 0 * (-1) + 2(1)$

$= 2$

$y = f(y_{in}) = 1$

$y = 1, \; t = -1 \quad \therefore t \neq y$

update weights

$w_{1(new)} = 0 + 1 * (-1)(-1)$

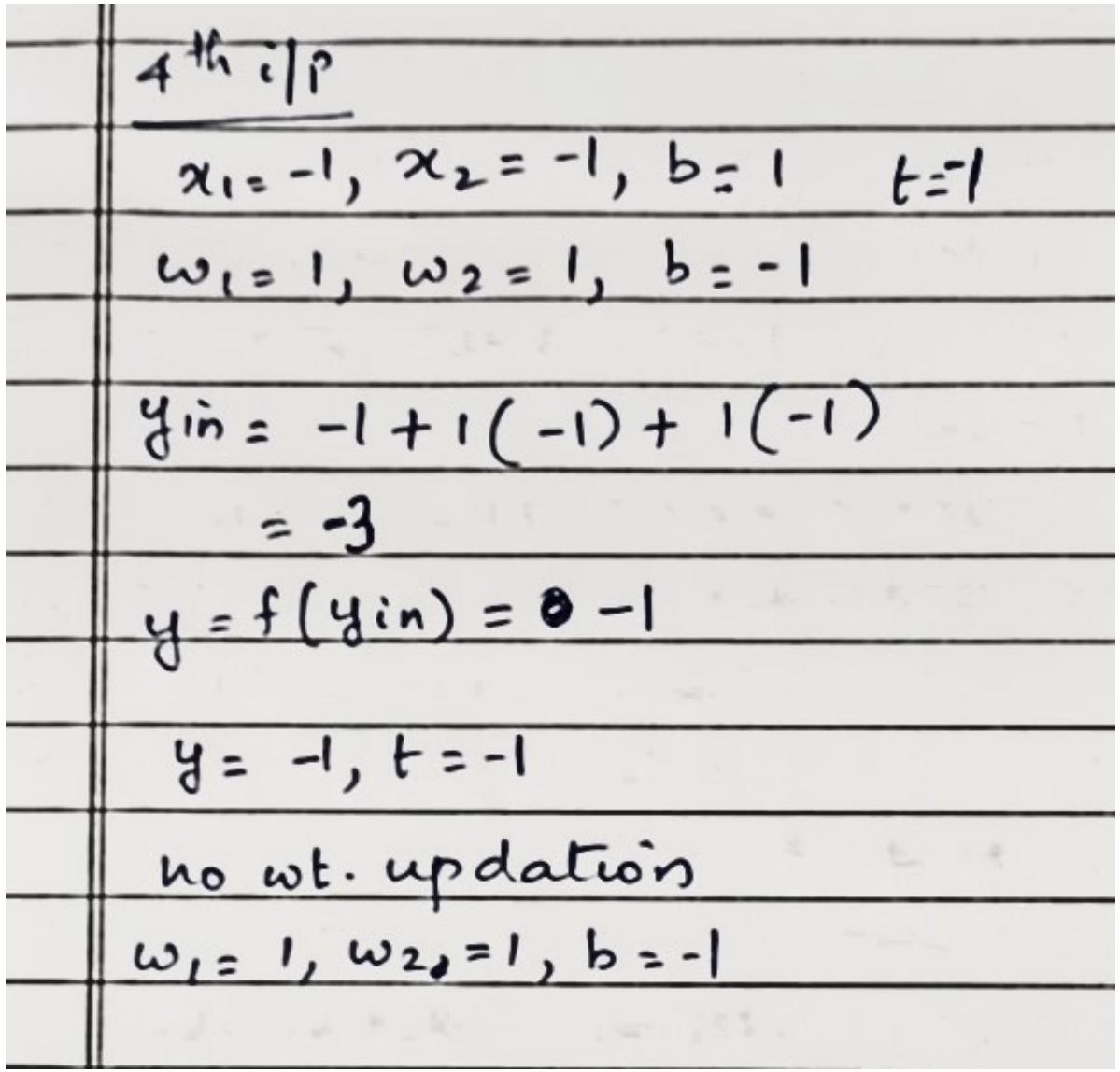
$= 1$

$w_{2(new)} = 2 + 1 * (-1)(1)$

$= 1$

$b_{new} = 0 + 1 * (-1)$

$= -1$



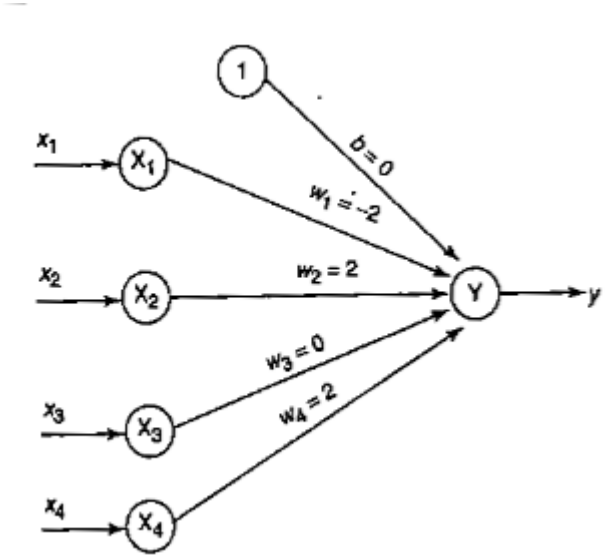
Input			Target	Net input	Calculated output	Weight changes			Weights		
x_1	x_2	1	(t)	(y_{in})	(y)	Δw_1	Δw_2	Δb	w_1 (0	w_2 0	b 0)
EPOCH-1											
1	1	1	1	0	0	1	1	1	1	1	1
1	−1	1	−1	1	1	−1	1	−1	0	2	0
−1	1	1	−1	2	1	+1	−1	−1	1	1	−1
−1	−1	1	−1	−3	−1	0	0	0	1	1	−1
EPOCH-2											
1	1	1	1	1	1	0	0	0	1	1	−1
1	−1	1	−1	−1	−1	0	0	0	1	1	−1
−1	1	1	−1	−1	−1	0	0	0	1	1	−1
−1	−1	1	−1	−3	−1	0	0	0	1	1	−1

Find the weights required to perform the following classification using perceptron network. The vectors (1, 1, 1, 1) and (−1, 1 −1, −1) are belonging to the class (so have target value 1), vectors (1, 1, 1, −1) and (1, −1, −1, 1) are not belonging to the class (so have target value −1). Assume learning rate as 1 and initial weights as 0.

Input					
x_1	x_2	x_3	x_4	b	Target (t)
1	1	1	1	1	1
−1	1	−1	−1	1	1
1	1	1	−1	1	−1
1	−1	−1	1	1	−1

$$y = \begin{cases} 1 & \text{if } y_{in} > 0.2 \\ 0 & \text{if } -0.2 \le y_{in} \le 0.2 \\ -1 & \text{if } y_{in} < -0.2 \end{cases}$$

Inputs					Target	Net input	Output	Weight changes					Weights				
$(x_1$	x_2	x_3	x_4	$b)$	(t)	(y_{in})	(y)	$(\Delta w_1$	Δw_2	Δw_3	Δw_4	$\Delta b)$	$(w_1$	w_2	w_3	w_4	$b)$
													(0	0	0	0	0)
EPOCH-1																	
(1	1	1	1	1)	1	0	0	1	1	1	1	1	1	1	1	1	1
(−1	1	−1	−1	1)	1	−1	−1	−1	1	−1	−1	1	0	2	0	0	2
(1	1	1	−1	1)	−1	4	1	−1	−1	−1	1	−1	−1	1	−1	1	1
(1	−1	−1	1	1)	−1	1	1	−1	1	1	−1	−1	−2	2	0	0	0
EPOCH-2																	
(1	1	1	1	1)	1	0	0	1	1	1	1	1	−1	3	1	1	1
(−1	1	−1	−1	1)	1	3	1	0	0	0	0	0	−1	3	1	1	1
(1	1	1	−1	1)	−1	4	1	−1	−1	−1	1	−1	−2	2	0	2	0
(1	−1	−1	1	1)	−1	−2	−1	0	0	0	0	0	−2	2	0	2	0
EPOCH-3																	
(1	1	1	1	1)	1	2	1	0	0	0	0	0	−2	2	0	2	0
(−1	1	−1	−1	1)	1	2	1	0	0	0	0	0	−2	2	0	2	0
(1	1	1	−1	1)	−1	−2	−1	0	0	0	0	0	−2	2	0	2	0
(1	−1	−1	1	1)	−1	−2	−1	0	0	0	0	0	−2	2	0	2	0



Thank you !!!!!

Machine Learning (19CSE305)

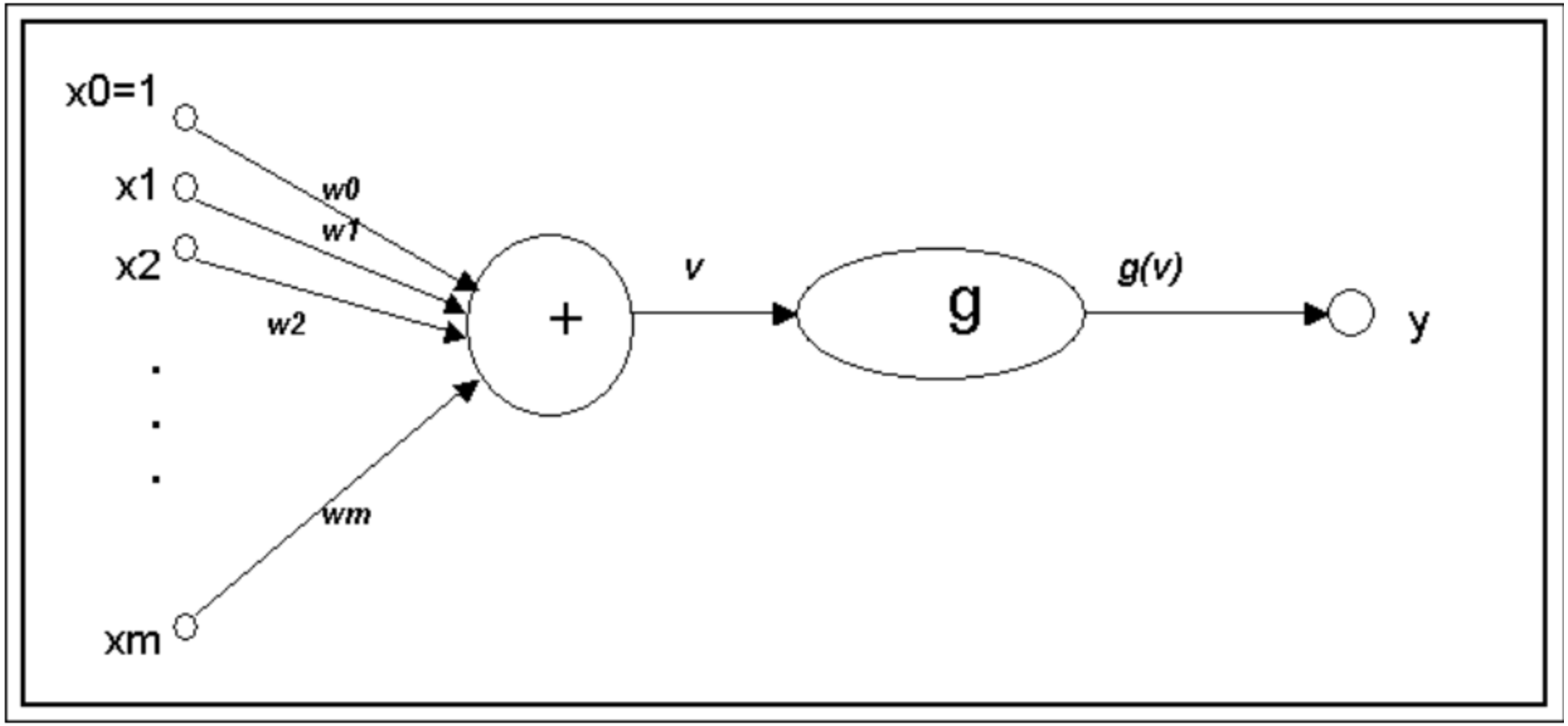
Neural Network

Dr. Peeta Basa Pati
Ms. Priyanka V
Department of Computer Science & Engineering,
Amrita School of Engineering, Bengaluru

ARTIFICIAL NEURON

1. The three basic components of the (artificial) neuron are:
    1. The synapses or connecting links that provide weights, w_j, to the input values, x_j for $j = 1, ...m$;
    2. An adder that sums the weighted input values to compute the input to the activation function $v = w_0 + \sum_{j=1}^{m} w_j x_j$,where w_0 is called the bias (not to be confused with statistical bias in prediction or estimation) is a numerical value associated with the neuron. It is convenient to think of the bias as the weight for an input x_0 whose value is always equal to one, so that $v = \sum_{j=0}^{m} w_j x_j$;
    3. An activation function g (also called a squashing function) that maps v to $g(v)$ the output value of the neuron. This function is a monotone function.

Perceptron

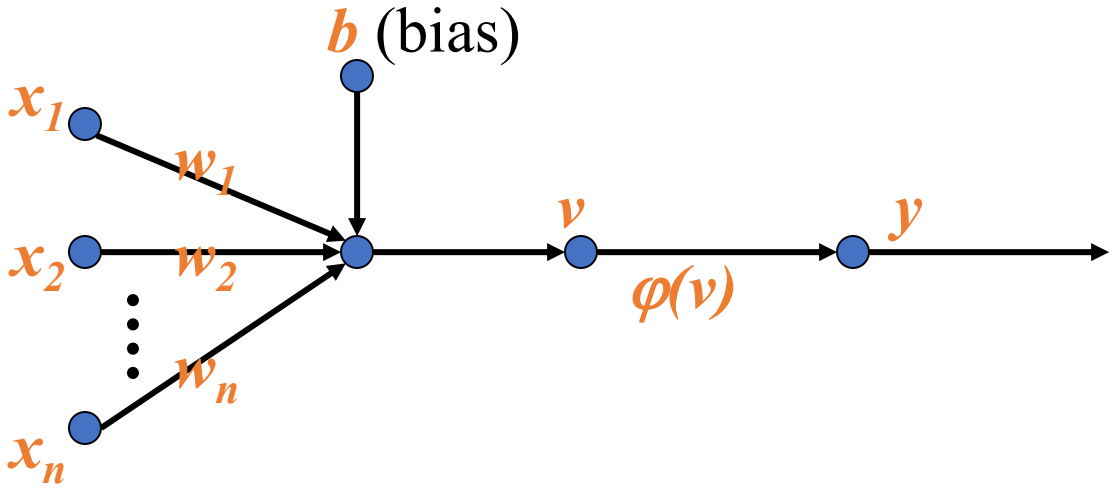


- The network is presented with cases from the training data one at a time
- The weights are revised after each case in an attempt to minimize the mean square error.
- This process of incremental adjustment of weights is based on the error made on training cases and is known as ’training’ the neural net
- if the network is trained in this manner by repeatedly presenting test data observations one-at-a-time then for suitably small (absolute) values of η the network will learn (converge to) the optimal values of w.
- One of the difficulties with the perceptron learning rule is that, if the data set happens not to be linearly separable, then the learning algorithm will never terminate.

Perceptron: Neuron Model
(Special form of single layer feed forward)

- The perceptron was first proposed by Rosenblatt (1958) is a simple neuron that is used to classify its input into one of two categories.
- A perceptron uses a **step function** that returns +1 if weighted sum of its input ≥ 0 and -1 otherwise

$$\varphi(v) = \begin{cases} +1 \text{ if } v \geq 0 \\ -1 \text{ if } v < 0 \end{cases}$$



Perceptron training

Step 0: Initialize the weights and the bias (for easy calculation they can be set to zero). Also initialize the learning rate $\alpha (0 < \alpha \leq 1)$. For simplicity α is set to 1.

Step 1: Perform Steps 2–6 until the final stopping condition is false.

Step 2: Perform Steps 3–5 for each training pair indicated by *s:t*.

Step 3: The input layer containing input units is applied with identity activation functions:

$$x_i = s_i$$

Step 4: Calculate the output of the network. To do so, first obtain the net input:

$$y_{in} = b + \sum_{i=1}^{n} x_i w_i$$

where "n" is the number of input neurons in the input layer. Then apply activations over the net input calculated to obtain the output:

$$y = f(y_{in}) = \begin{cases} 1 & \text{if } y_{in} > \theta \\ 0 & \text{if } -\theta \leq y_{in} \leq \theta \\ -1 & \text{if } y_{in} < -\theta \end{cases}$$

Step 5: *Weight and bias adjustment:* Compare the value of the actual (calculated) output and desired (target) output.

If $y \neq t$, then

$$w_i(\text{new}) = w_i(\text{old}) + \alpha t x_i$$
$$b(\text{new}) = b(\text{old}) + \alpha t$$

else, we have

$$w_i(\text{new}) = w_i(\text{old})$$
$$b(\text{new}) = b(\text{old})$$

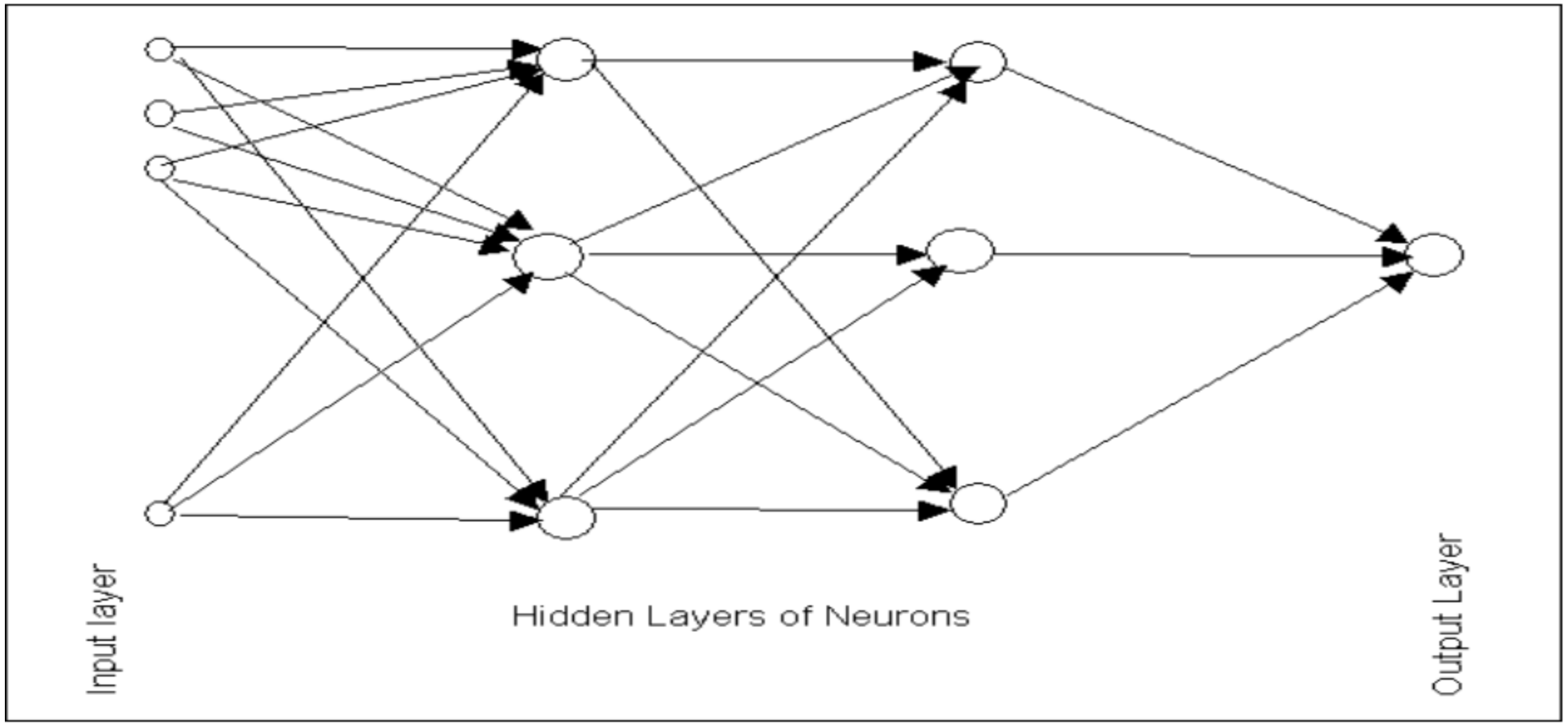
Step 6: Train the network until there is no weight change. This is the stopping condition for the network. If this condition is not met, then start again from Step 2.

- Weight updation can be also as :
- (delta rule)

$$w_i \leftarrow w_i + \Delta w_i$$

$$\Delta w_i = \eta(t - o)x_i$$

Multilayer feedforward networks.



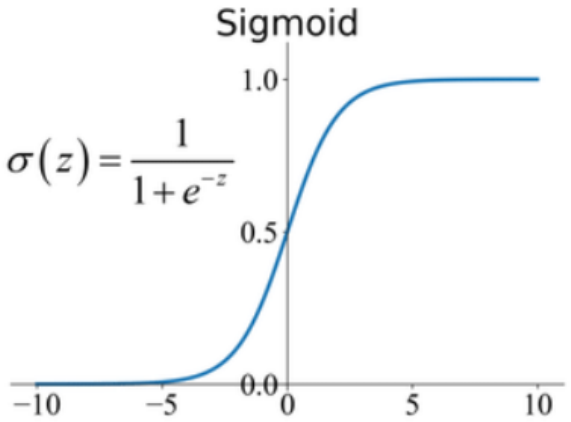
- In a supervised setting where a neural net is used to predict a numerical quantity there is one neuron in the output layer and its output is the prediction.
- When the network is used for classification, the output layer typically has as many nodes as the number of classes and the output layer node with the largest output value gives the network's estimate of the class for a given input.
- In the special case of two classes it is common to have just one node in the output layer, the classification between the two classes being made by applying a cut-off to the output value at the node

Backward Propagation Algorithm

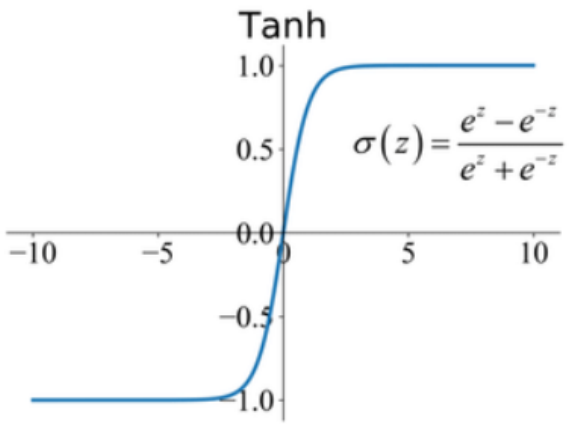
- Forward Pass: Computation of outputs of all the neurons in the network
- values of wij are initialized to small (generally random) numbers in the range 0.00±0.05.
- Backward pass: Propagation of error and adjustment of weights
- Learning Rate η in the range 0.1 to 0.9

Activation function 👁 👁

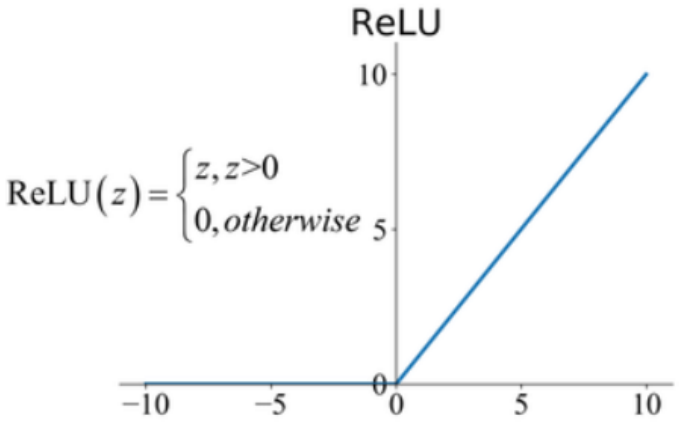
An activation function is a function used in artificial neural networks which outputs a small value for small inputs, and a larger value if its inputs exceed a threshold. Activation functions are useful because they add non-linearities into neural networks, allowing the neural networks to learn powerful operations.



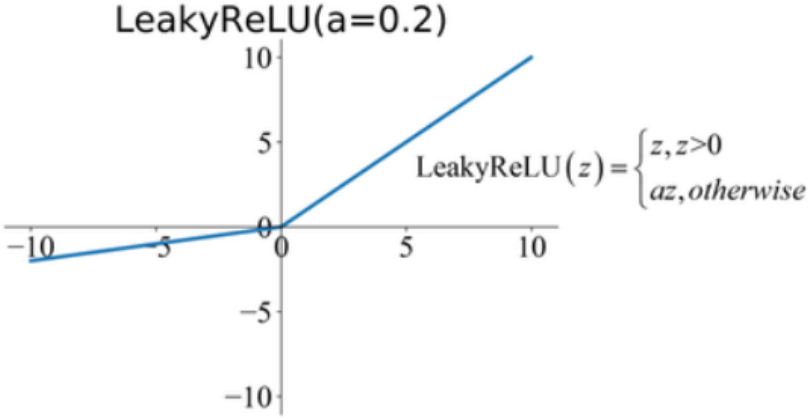
(a)



(b)



(c)



(d)

The need for Bias

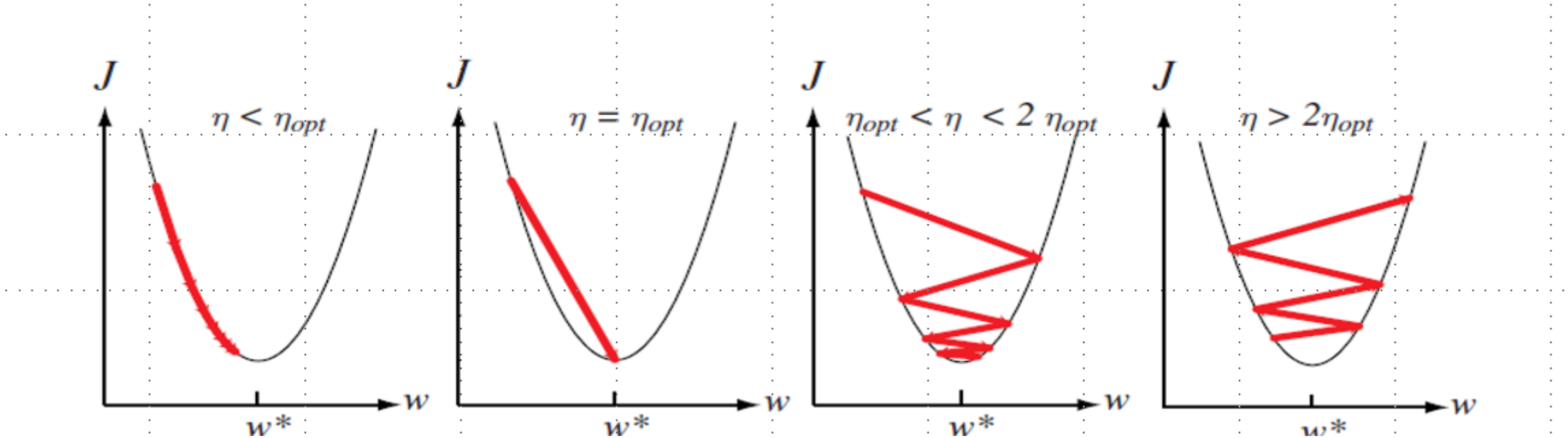
Bias allows you to shift the activation function by adding a constant

$$y = (m*x) + c$$

$$\text{Output} = (W*x) + b(\text{bias})$$

helps the model in a way that it can fit best for the given data

Learning Rate



- initial weights
  - random., near to 0
- Number of neurons in hidden layer
  - Validation
  - Domain knowledge
  - In practice this issue is solved by trial and error.
  - Two types of adaptive algorithms can be used:
    - start from a large network and successively remove some neurons and links until network performance degrades.
    - begin with a small network and introduce new neurons until performance is satisfactory

Recurrent Neural Network

- output from previous step are fed as input to the current step.
- cases like when it is required to predict the next word of a sentence, the previous words are required and hence there is a need to remember the previous words

Thank you !!!!!

Machine Learning (19CSE305)

Error Surface & Parameter Optimization

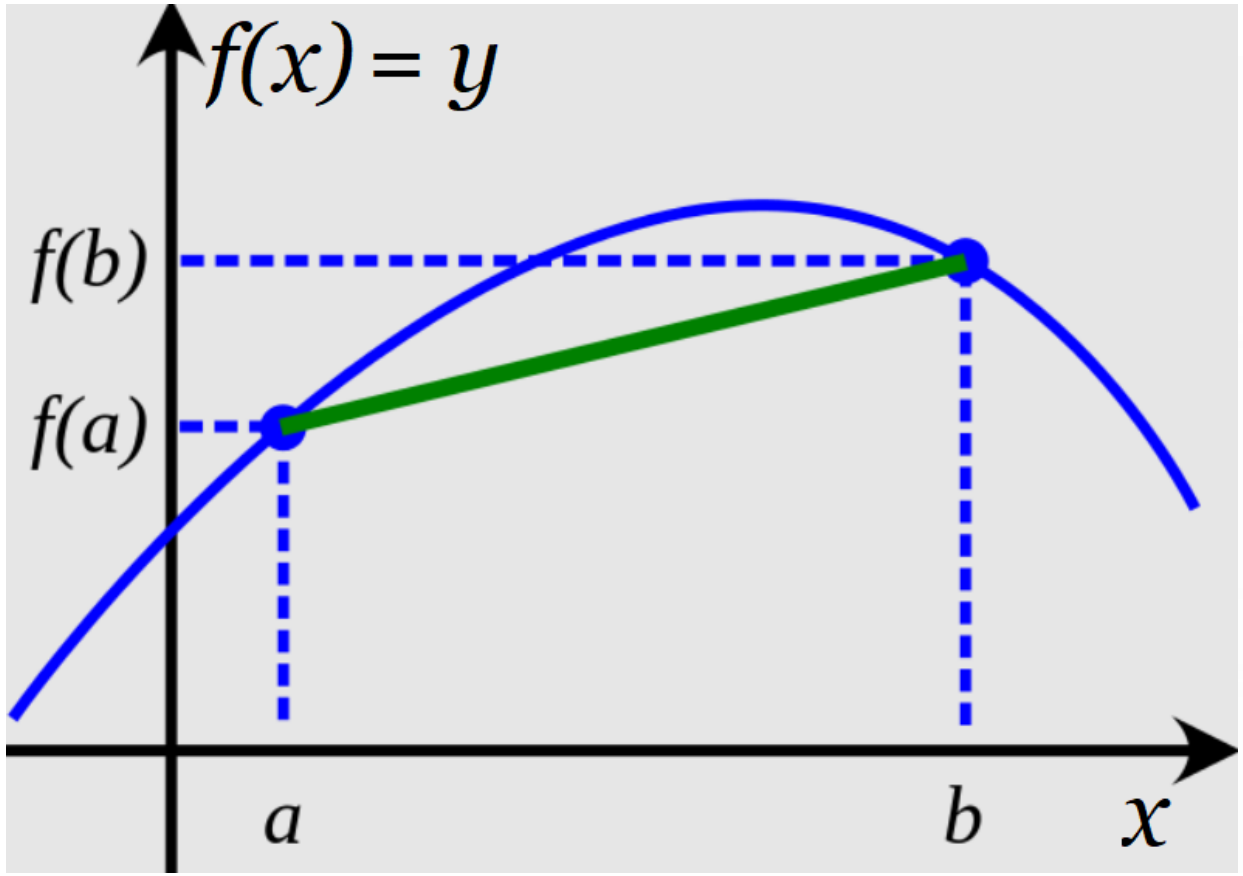
AMRITA VISHWA VIDYAPEETHAM
DEEMED TO BE UNIVERSITY

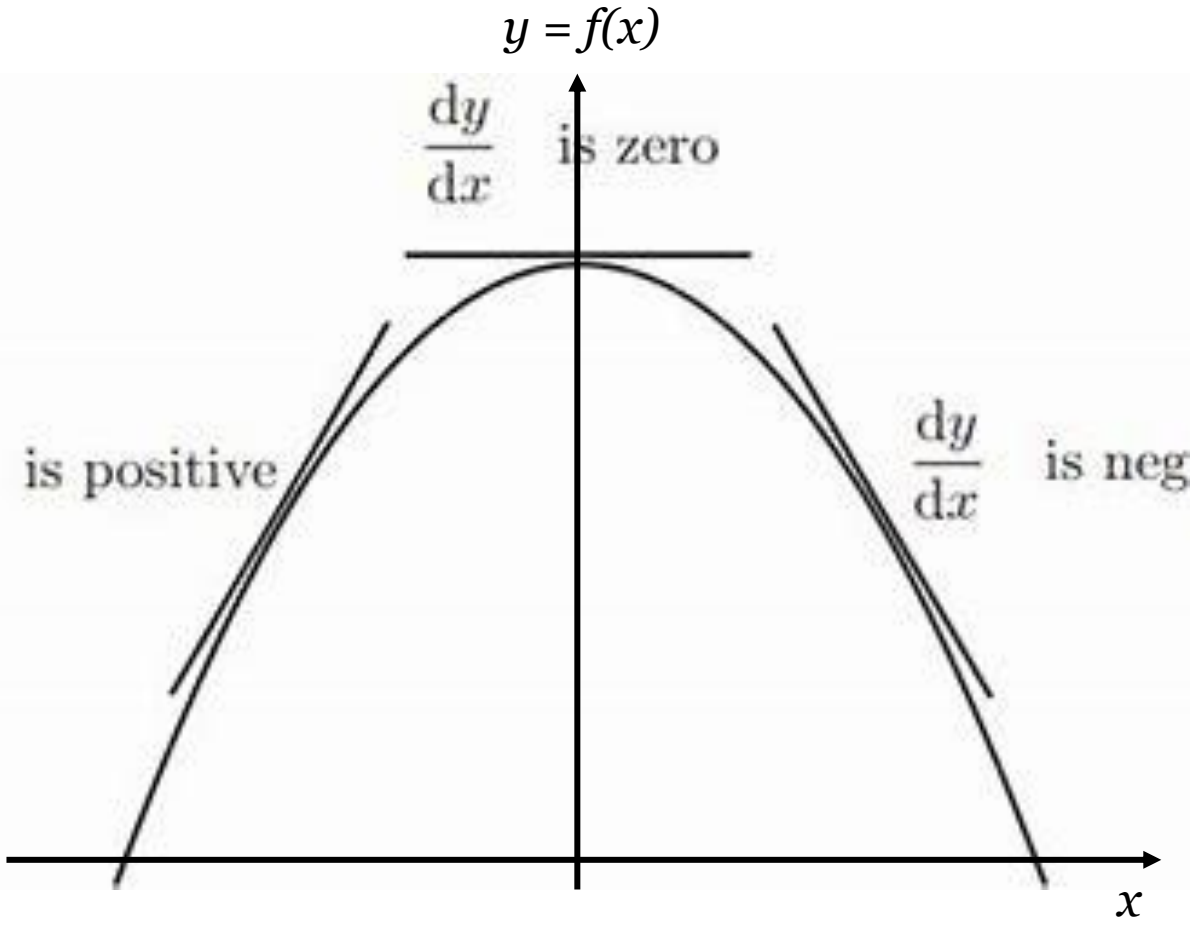
Dr. Peeta Basa Pati
Ms. Priyanka V
Department of Computer Science & Engineering,
Amrita School of Engineering, Bengaluru

Topics

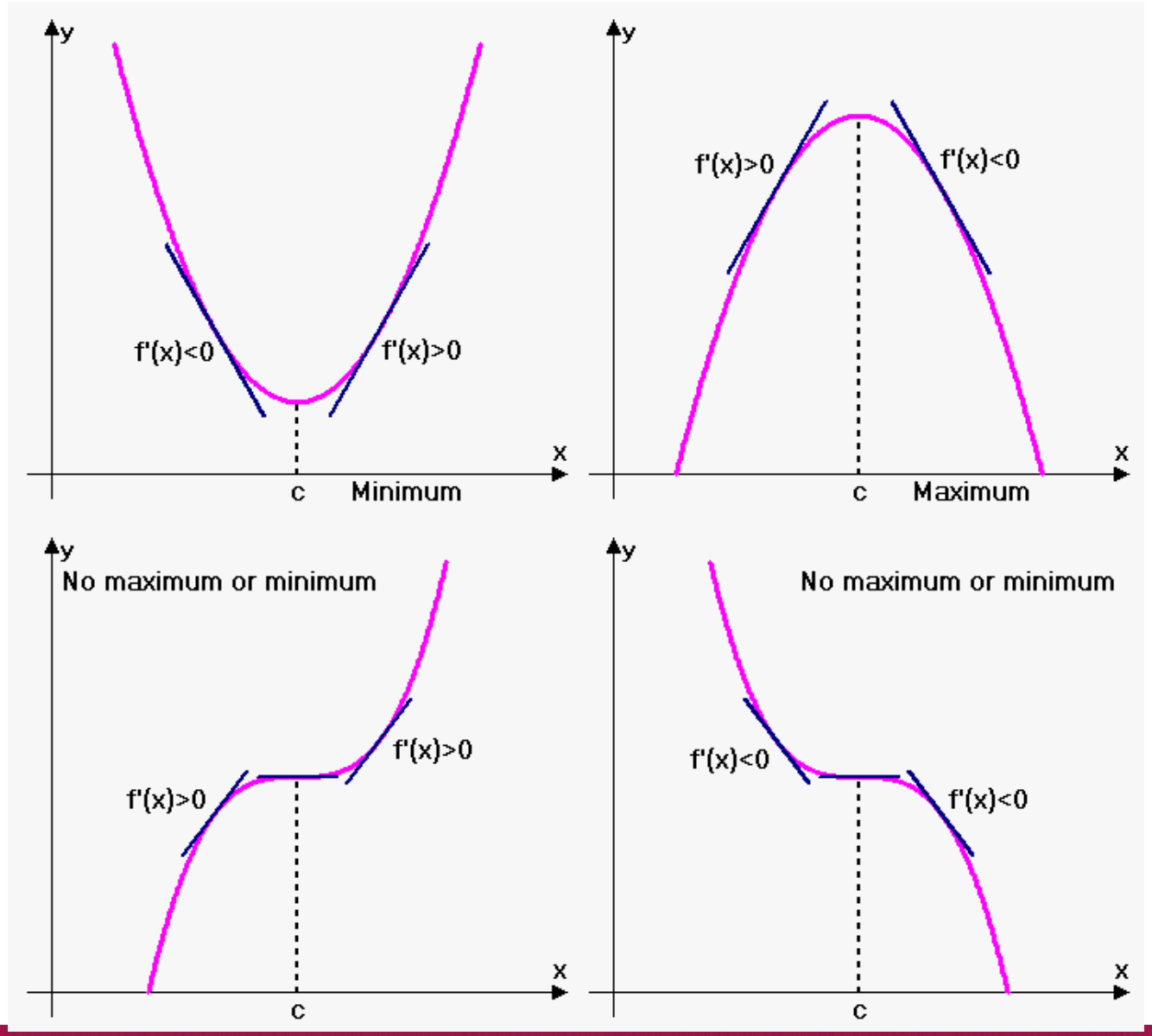
- Parameter space and error surface
- Optimal points
- Algorithms
  - Brute force
  - Gradient Descent
- Quiz

Functions and Derivatives





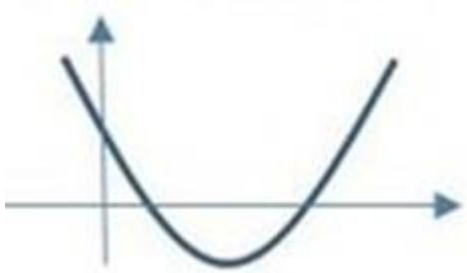
Functions and Derivatives



Source: Internet

Functions and Derivatives

Function

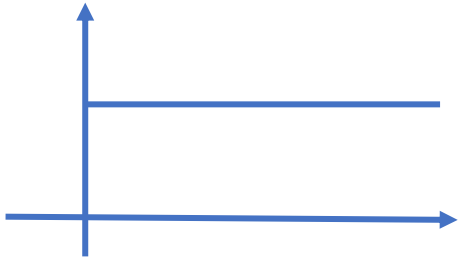


$$y = (x-1) * (x-3)$$
$$= x^2 - 4x + 3$$

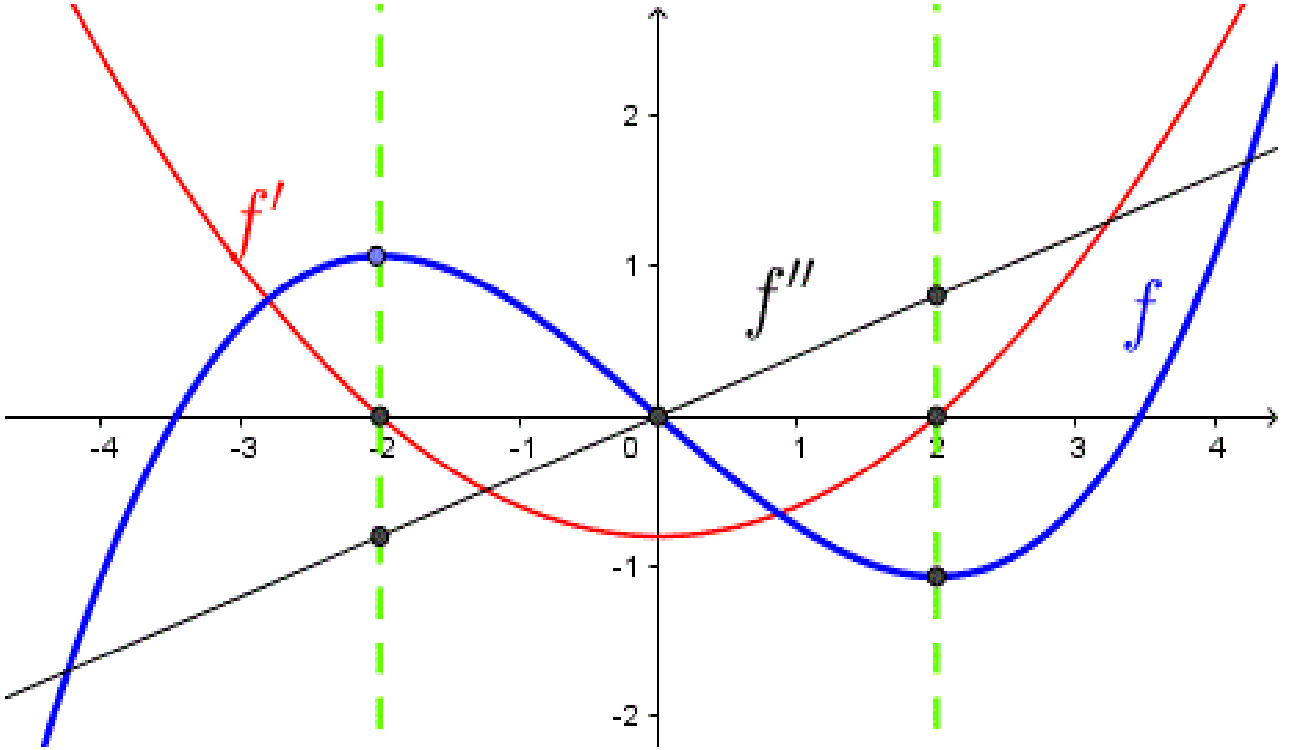
1st Derivative

$$\frac{dy}{dx} = y' = 2x - 4$$

2nd Derivative

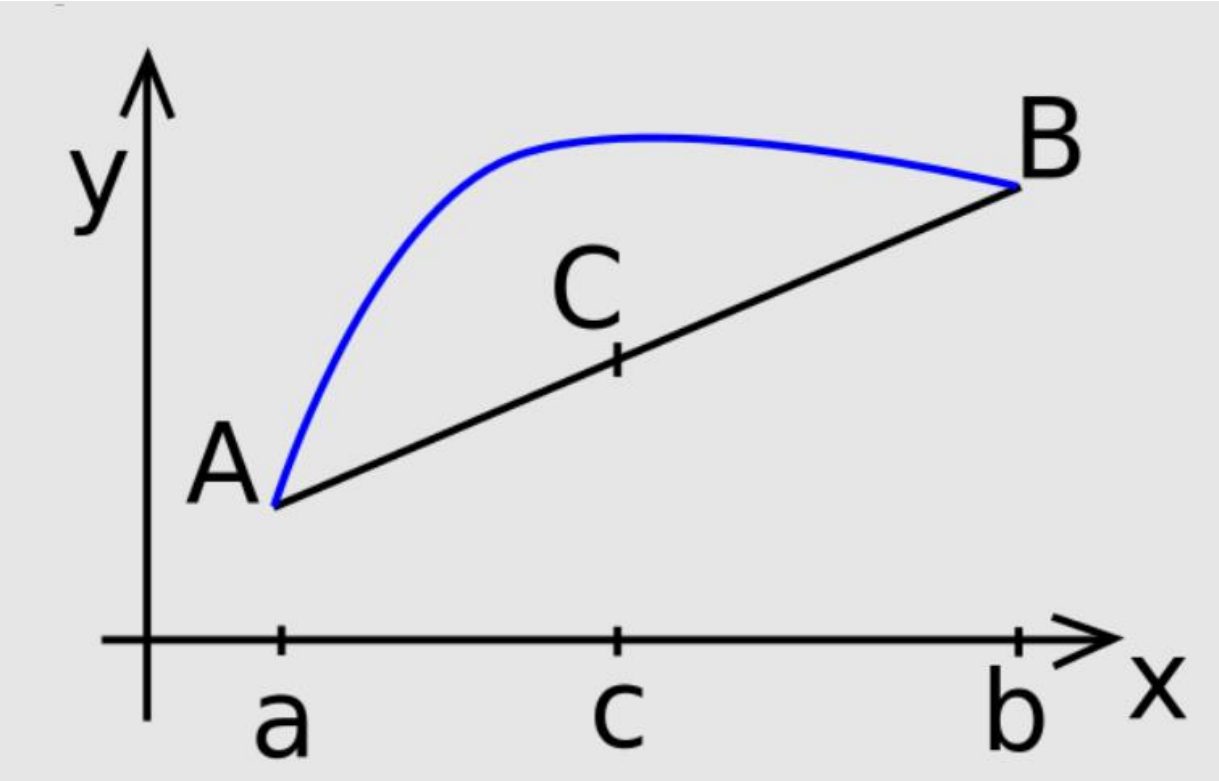


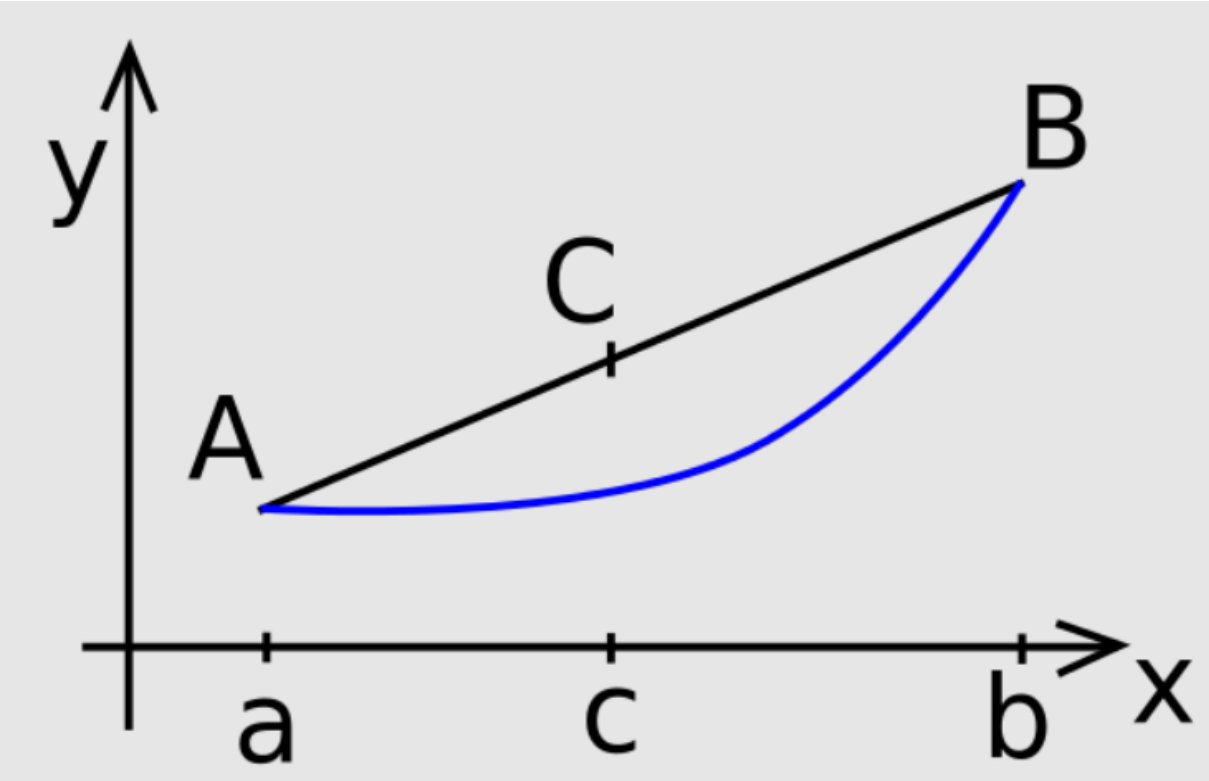
$$\frac{dy'}{dx} = \frac{d^2y}{dx^2} = 2$$



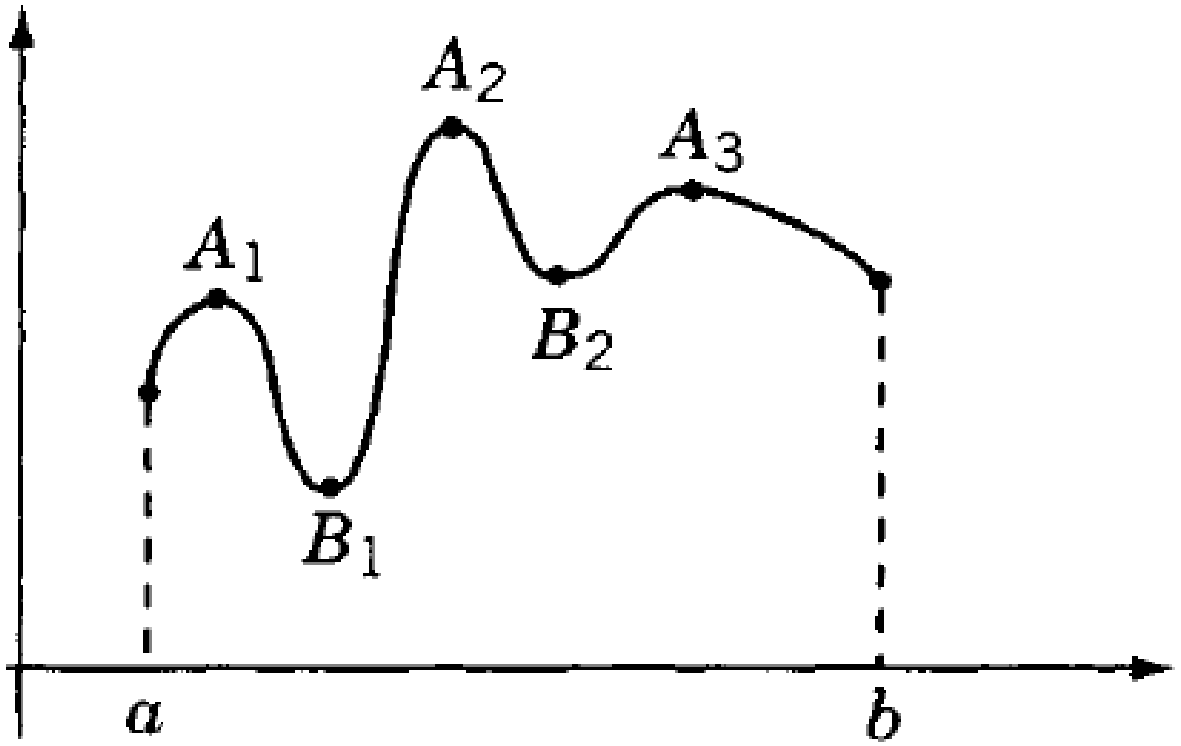
Source: www.analyzemath.com/

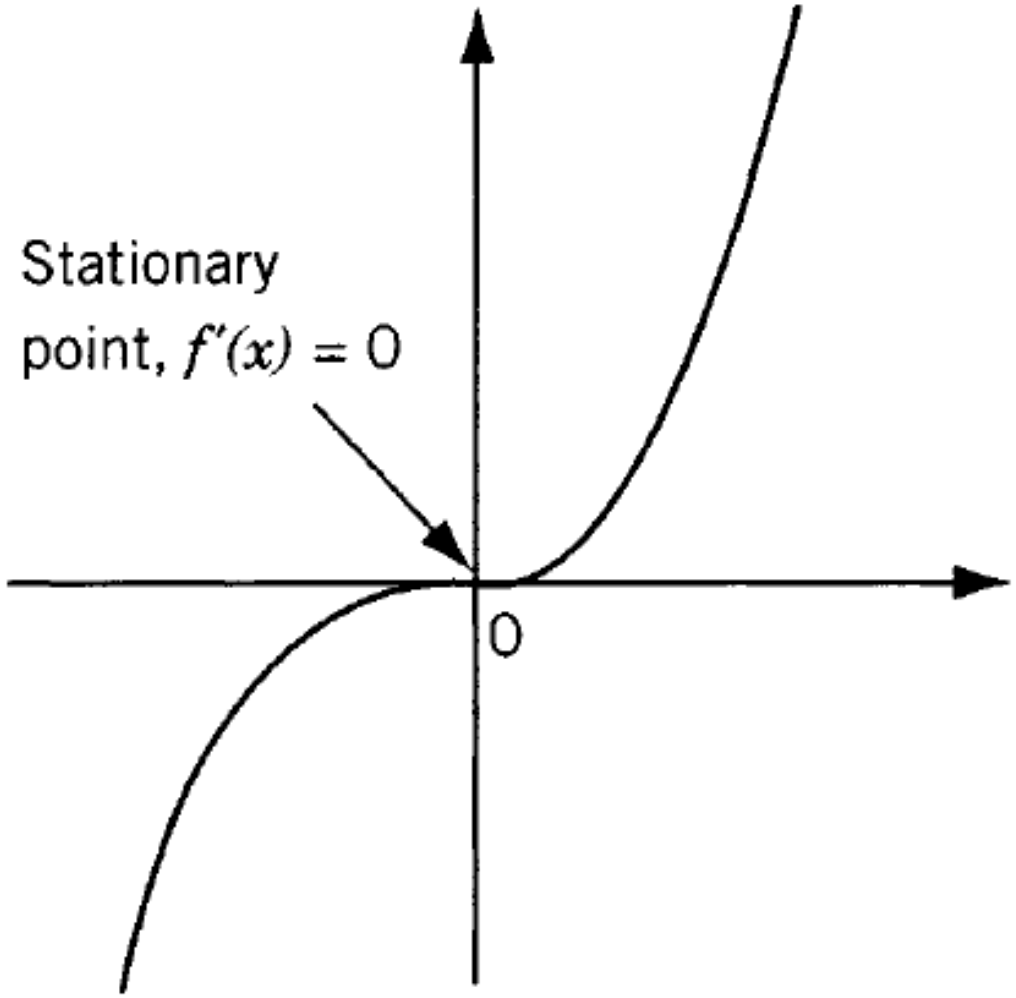
Concave and Convex Functions





Local & Global – Minimum, Maximum; Saddle point

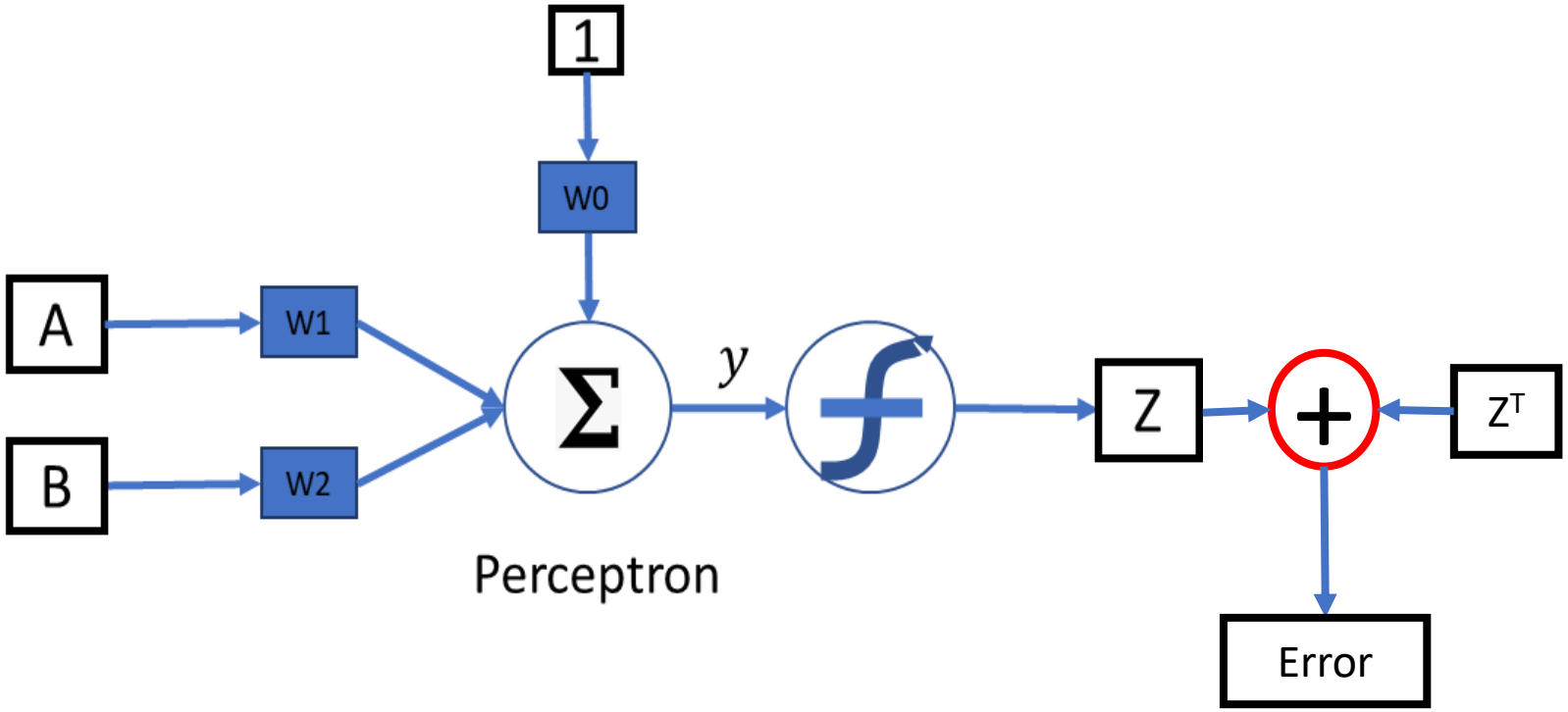




Source: Engineering Optimization, S S Rao

Estimation Error

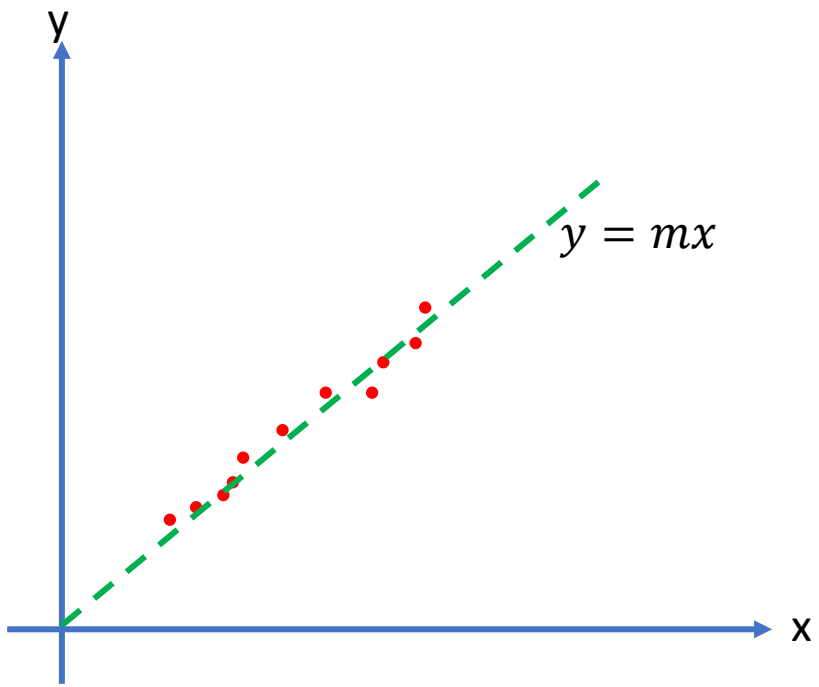
- X ➔ Input vector
- Z ➔ Estimation Output
- Z^T ➔ Target Output
- E ➔ Error

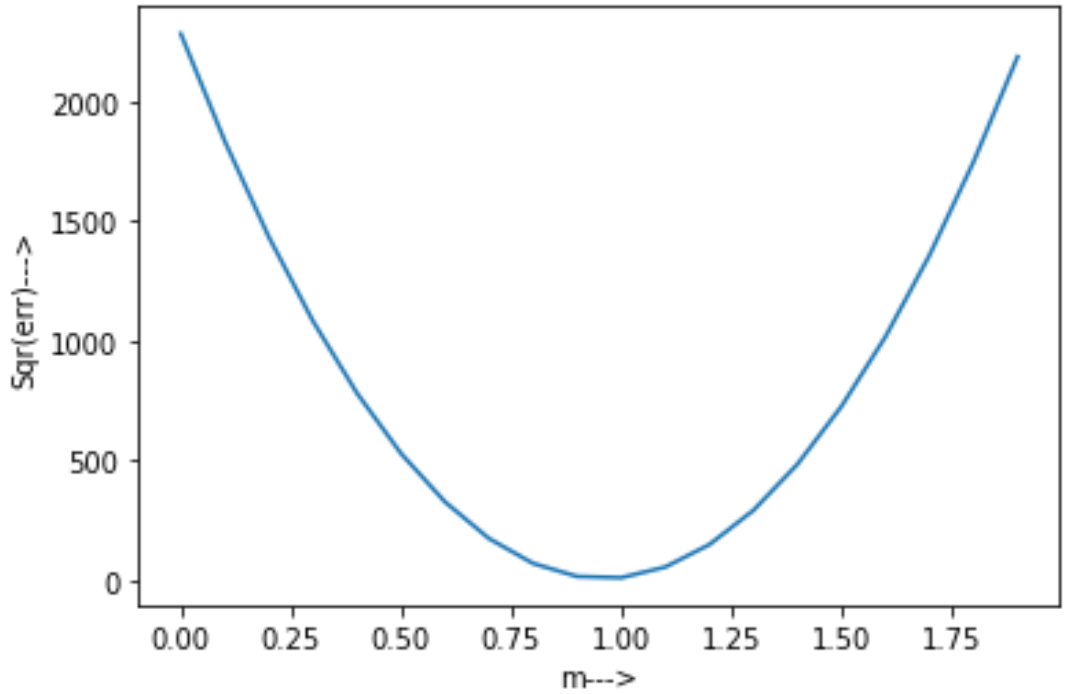


Aim of learning is to find the right set of parameters (weights in this case) such that error is minimal.

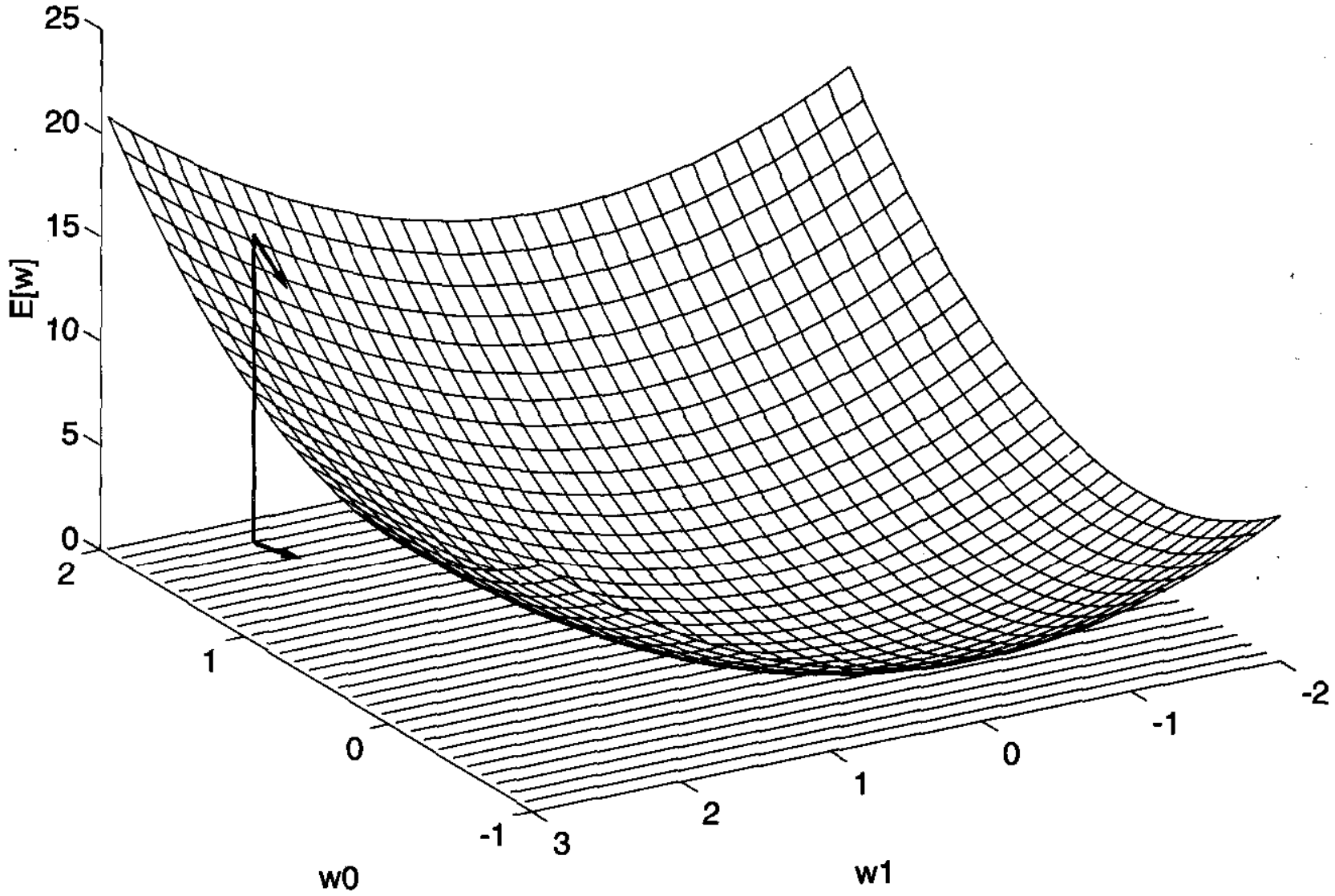
This is an optimization problem.

Single Parameter





Error space for parameters in R^2



Brute Force or Exhaustive Search

- Estimate error for all variations of all parameters (weights & bias)
- Find the global minimum value
- Search is performed over the entire range of set limits
- Huge computational cost for the exercise; however, guaranteed outcome

Derivation of Gradient Descent Algorithm

$$o(\vec{x}) = \vec{w} \cdot \vec{x}$$

$$E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

$$\vec{w} \leftarrow \vec{w} + \Delta \vec{w}$$

$$\Delta \vec{w} = -\eta \nabla E(\vec{w})$$

$$\nabla E(\vec{w}) \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \cdots, \frac{\partial E}{\partial w_n} \right]$$

$$w_i \leftarrow w_i + \Delta w_i$$

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

$$= \frac{1}{2} \sum_{d \in D} \frac{\partial}{\partial w_i} (t_d - o_d)^2$$

$$= \frac{1}{2} \sum_{d \in D} 2(t_d - o_d) \frac{\partial}{\partial w_i} (t_d - o_d)$$

$$= \sum_{d \in D} (t_d - o_d) \frac{\partial}{\partial w_i} (t_d - \vec{w} \cdot \vec{x}_d)$$

$$\frac{\partial E}{\partial w_i} = \sum_{d \in D} (t_d - o_d)(-x_{id})$$

$$\Delta w_i = \eta \sum_{d \in D} (t_d - o_d)\ x_{id}$$

Notes on Gradient Descent Algorithm

- Error is summed over all inputs and then the weights are updated
- Linear (pass through) activation function is used → f(x) = x
- Assumes a convex error space
- If there are local minima, the search may get stuck and never come-out
- Since error is summed over all inputs, the convergence may be slow
- Incremental or stochastic gradient descent is a variation of the algorithm
  - Weight update done with each input
  - Sometimes helps in overcoming the local minima
- The same principle can be applied with other activation functions as well. However, mathematical proof of convergence may be difficult.

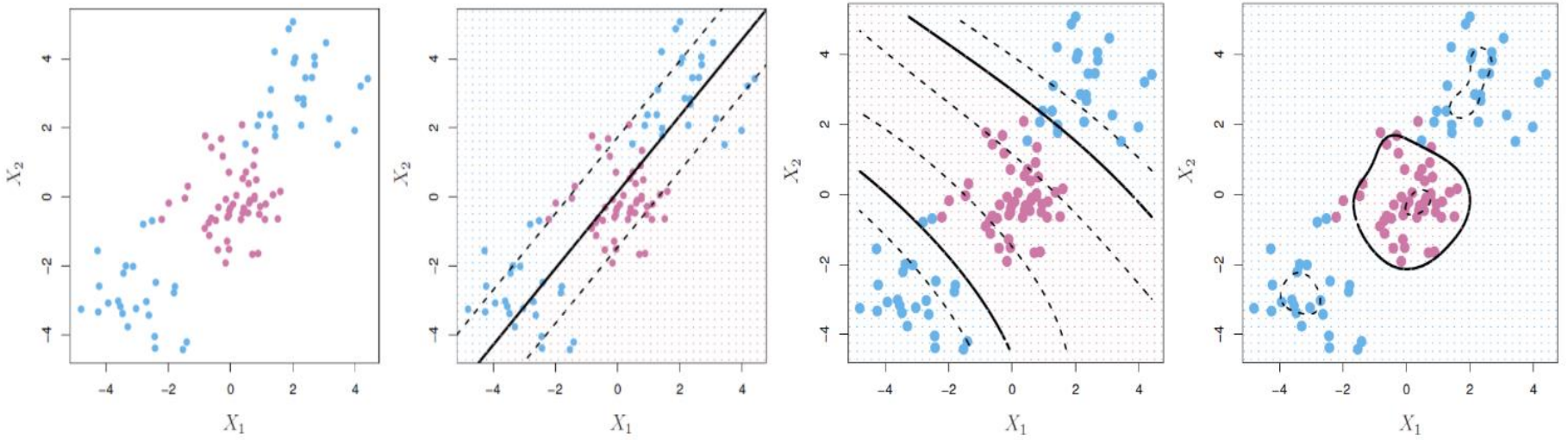
Other optimization techniques

- Genetic algorithm based approaches
- Evolutionary computation techniques
- Tabu Search
- Simulated Annealing
- Hill Climbing techniques
- Ant colony optimization
- Particle swarm optimization
- Random forest optimization

Thank you !!!!!

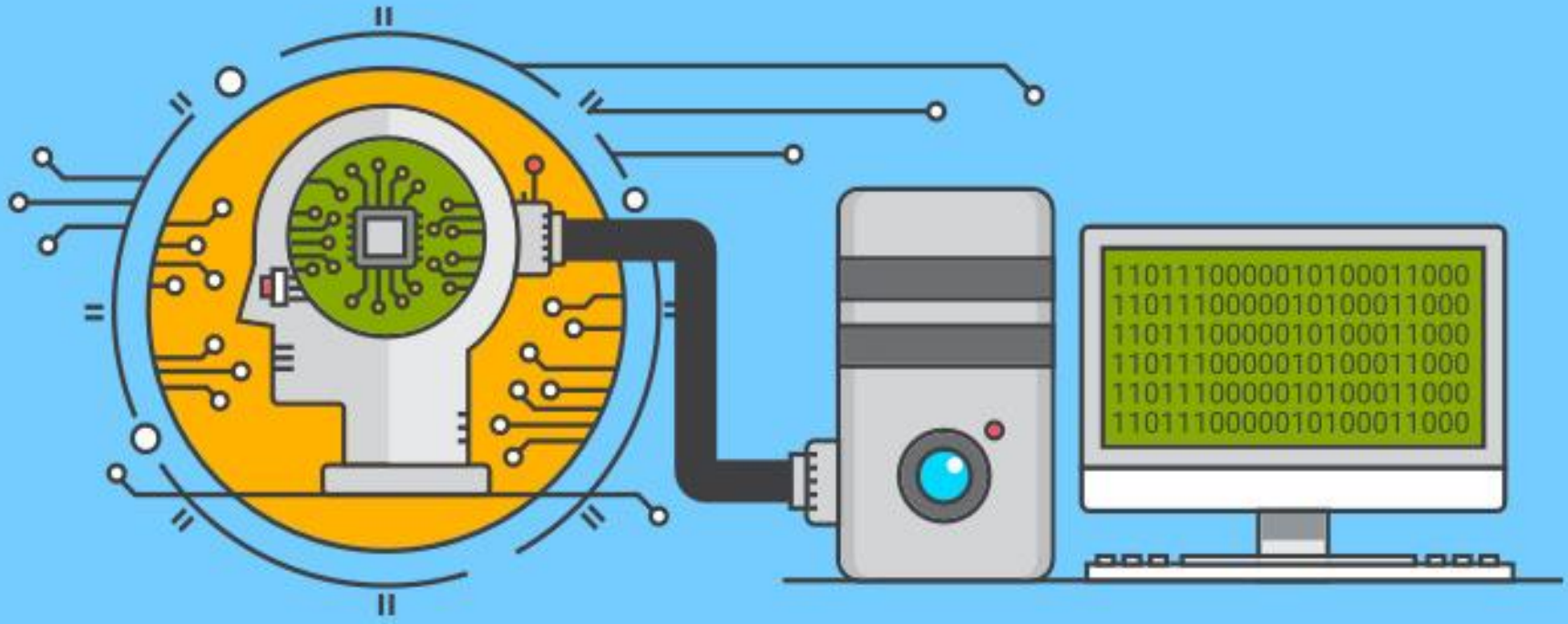
SUPPORT VECTOR MACHINES

A brief introduction



What is machine learning?

Machine learning is the concept that a computer program can learn and adapt to new data with minimal human intervention.

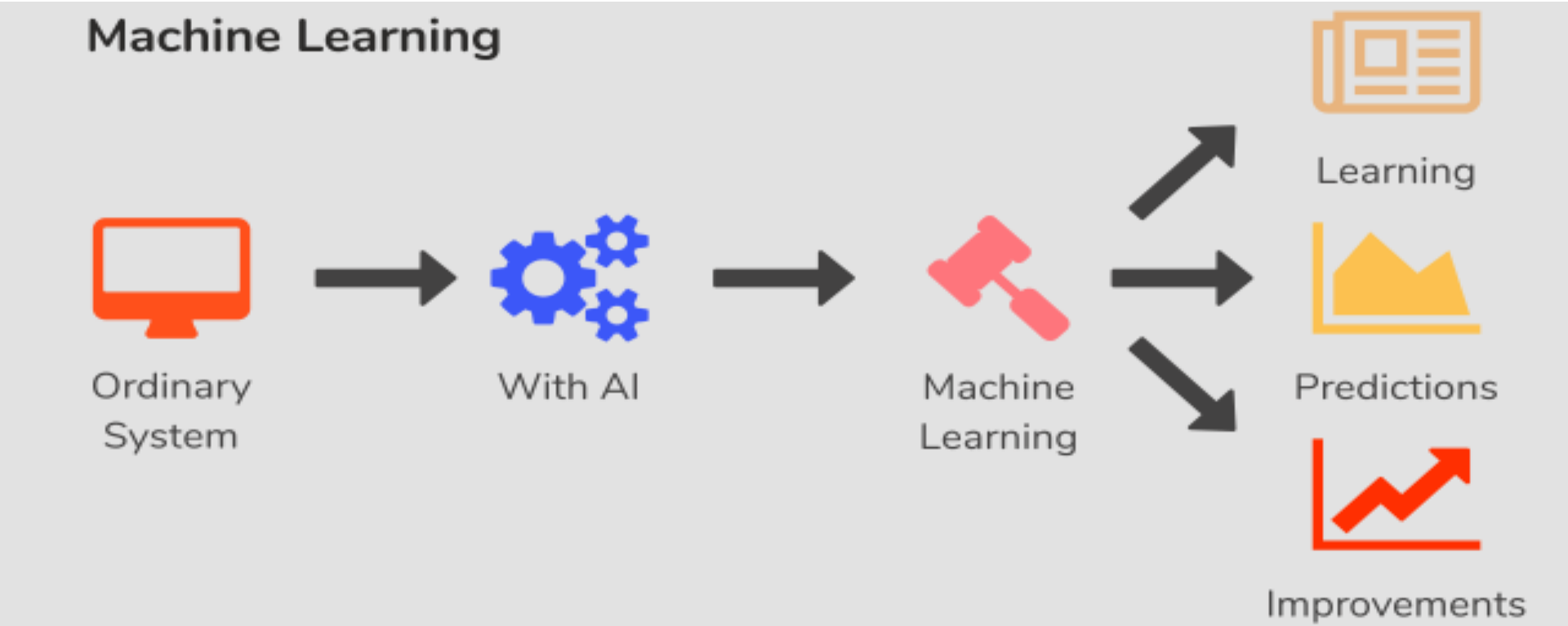


Why machine learning is important?

Data is the lifeblood of all business. Data-driven decisions increasingly make the difference between keeping up with competition or falling further behind.

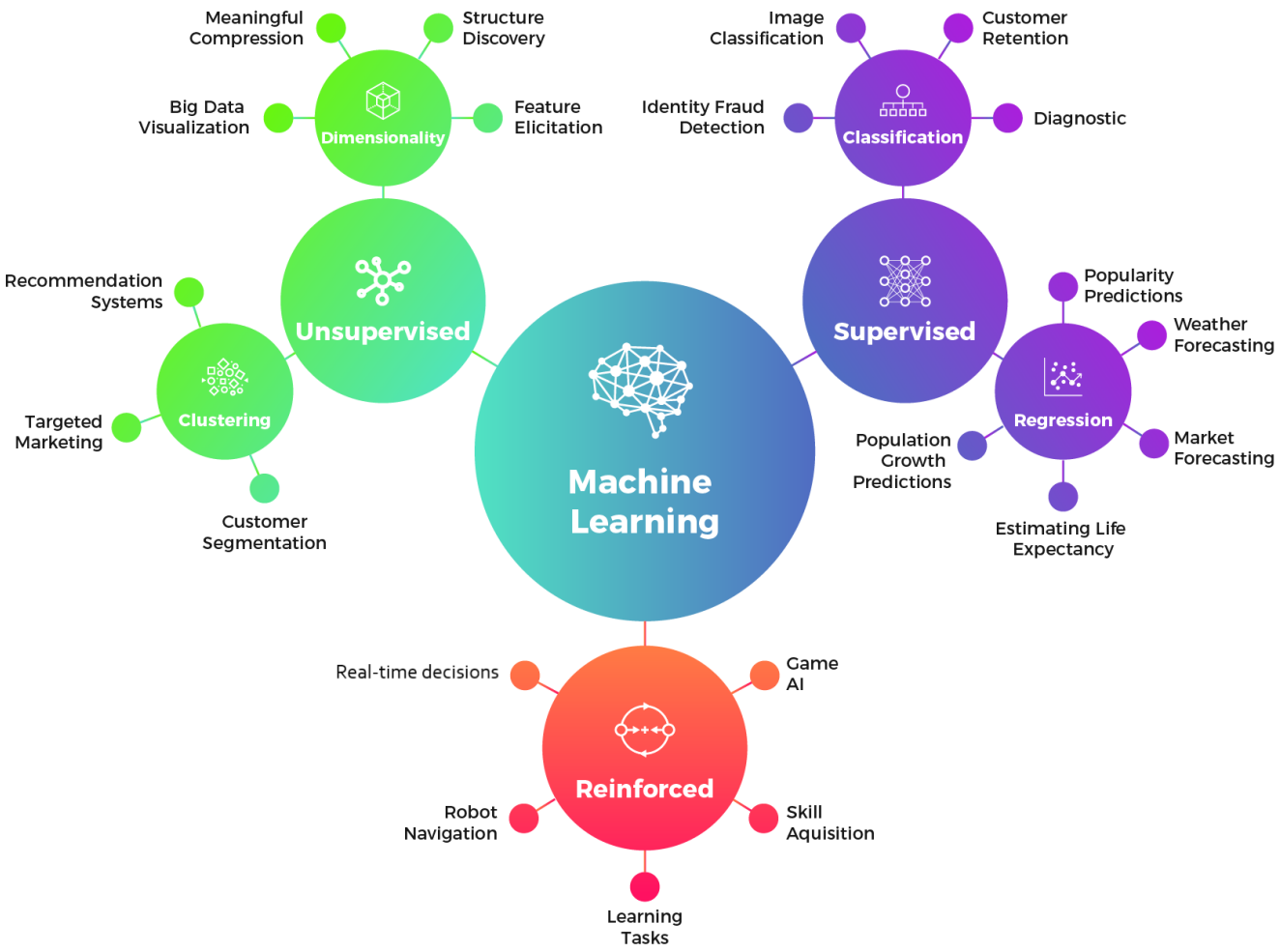
Use cases:

- Manufacturing.
- Retail.
- Healthcare and life sciences.
- Travel and hospitality.
- Financial services.
- Energy.



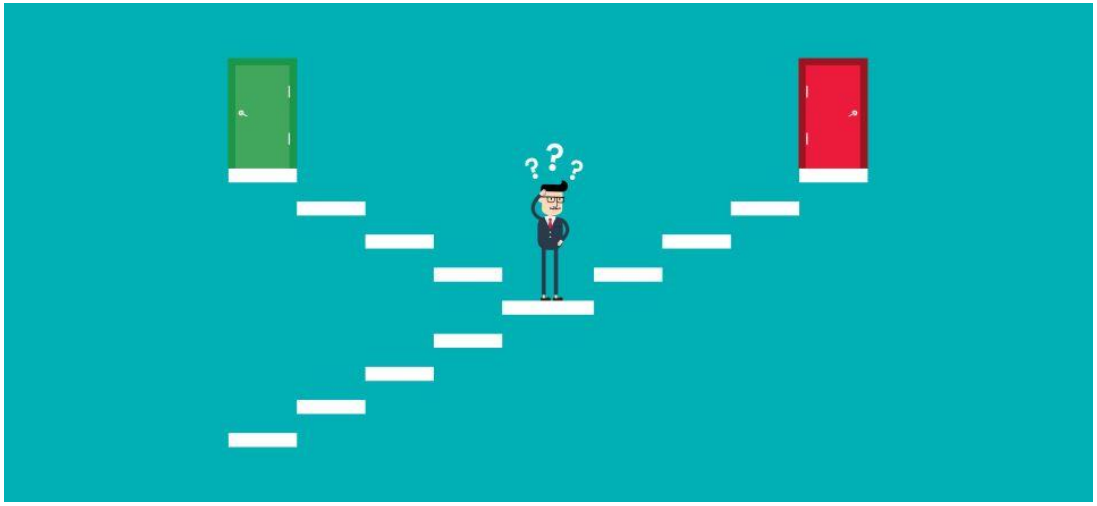
What is a machine learning model?

A machine learning model is a file that has been trained to recognize certain types of patterns. You train a model over a set of data, providing it an algorithm that it can use to reason over and learn from those data.

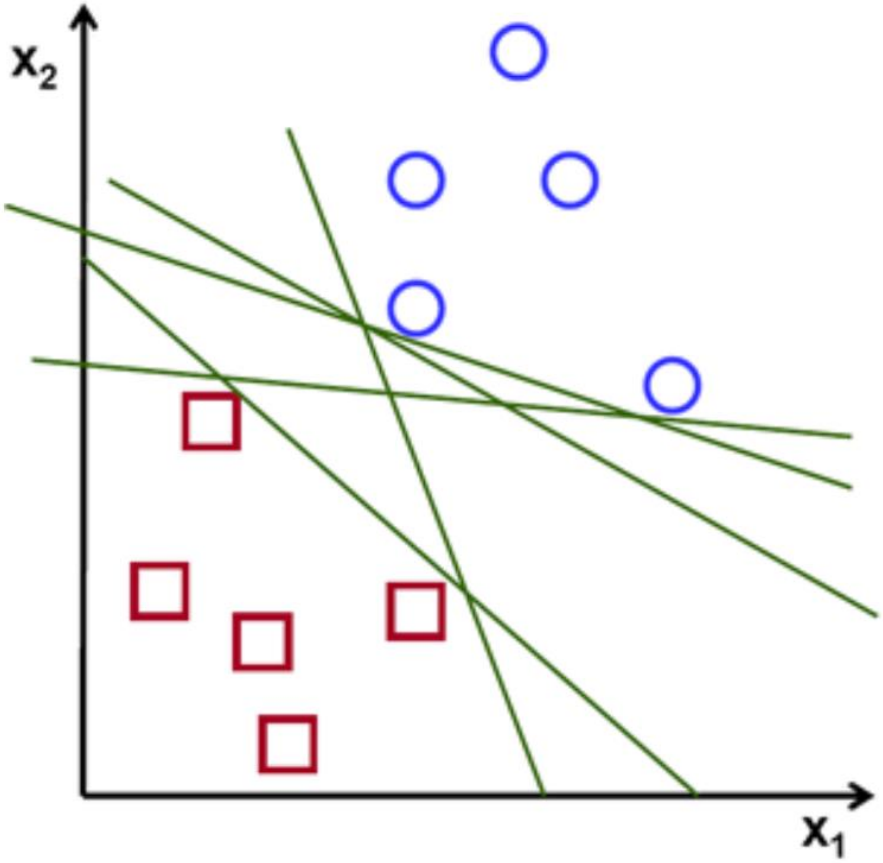


What do we do after choosing a model?

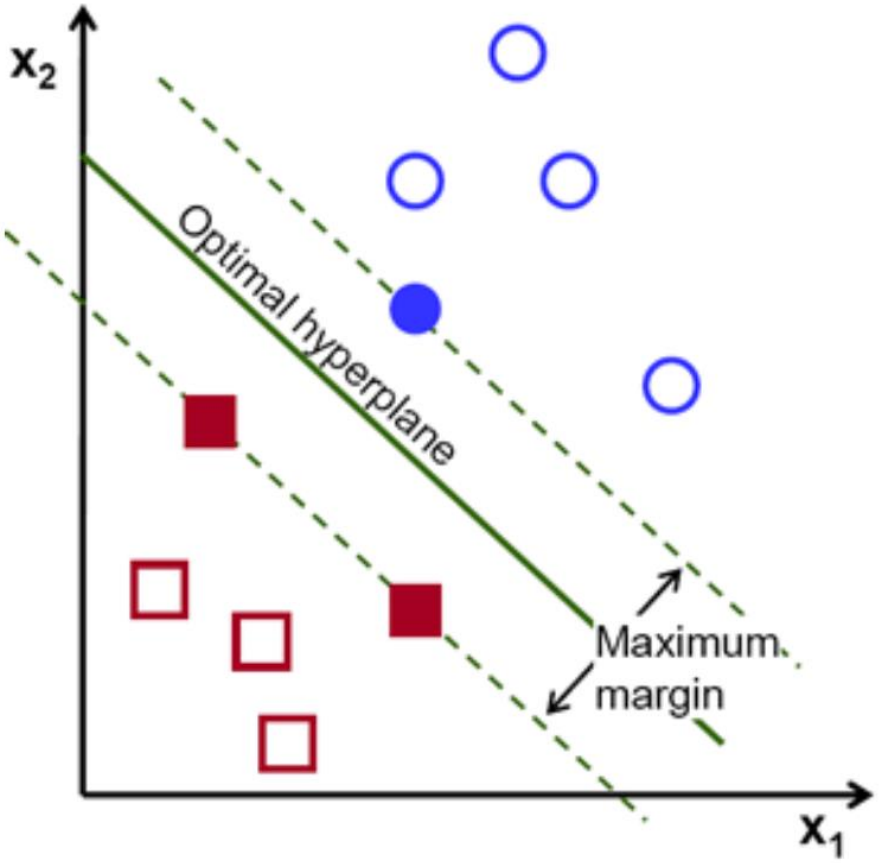
How do we chose which model works best for the data we have?



Support Vector Machine (SVM)

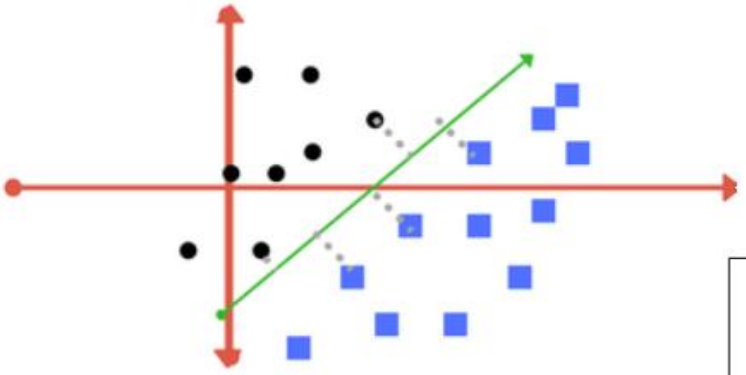


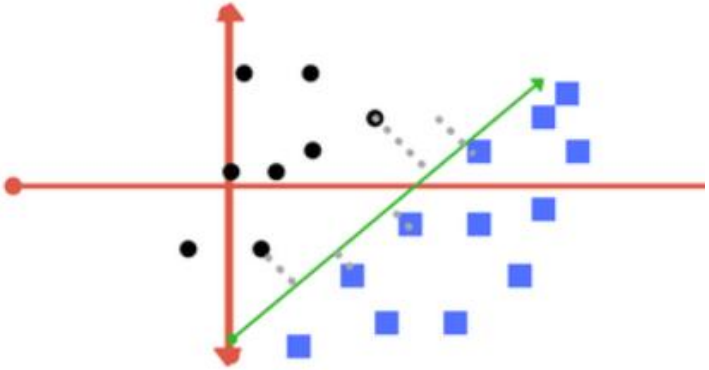
Few possible hyperplanes

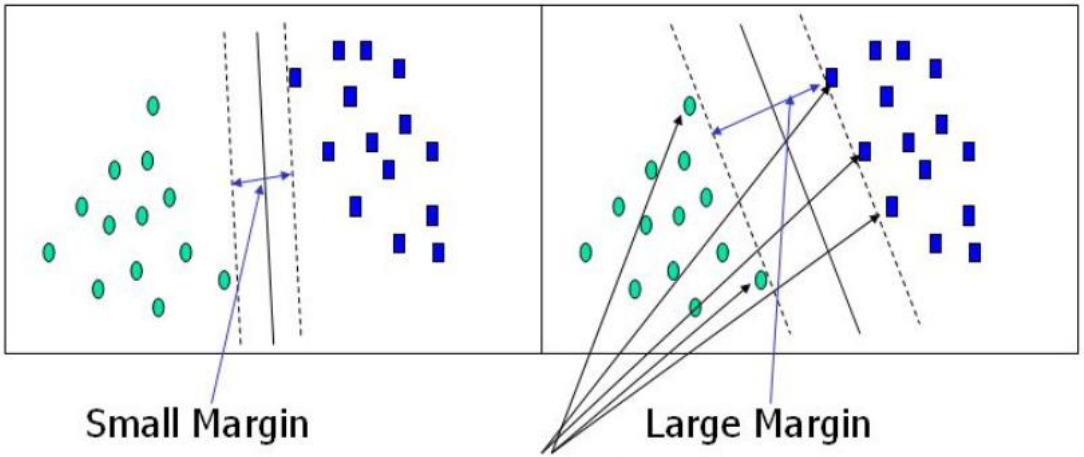


Optimal hyperplane

Margins

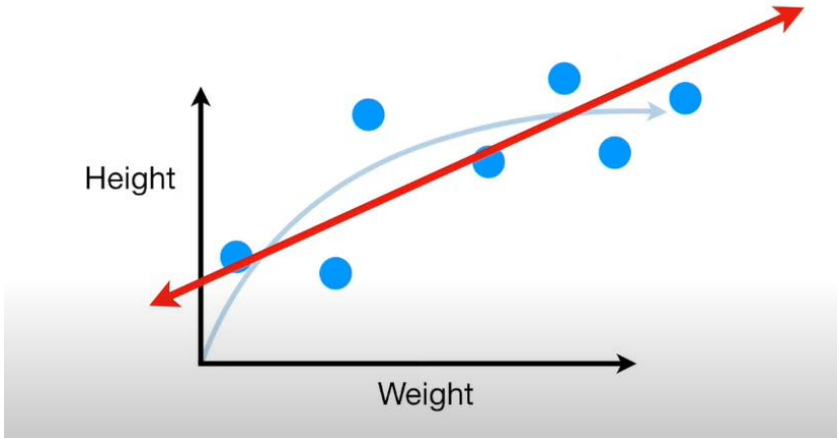


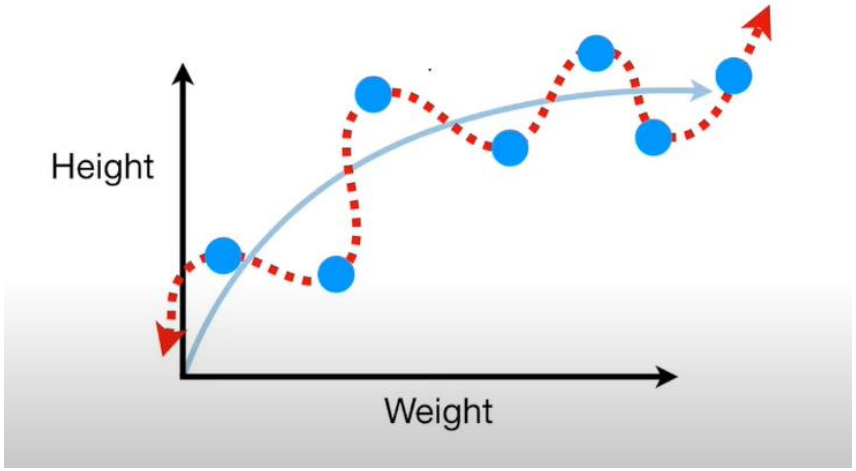




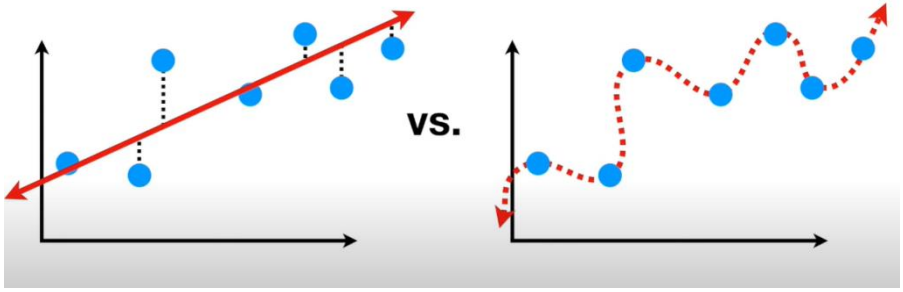
What are support vectors?

Regularization

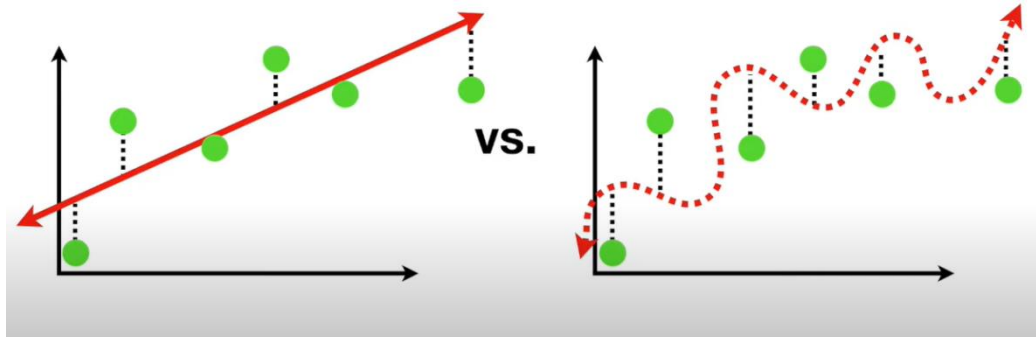




Training



Testing

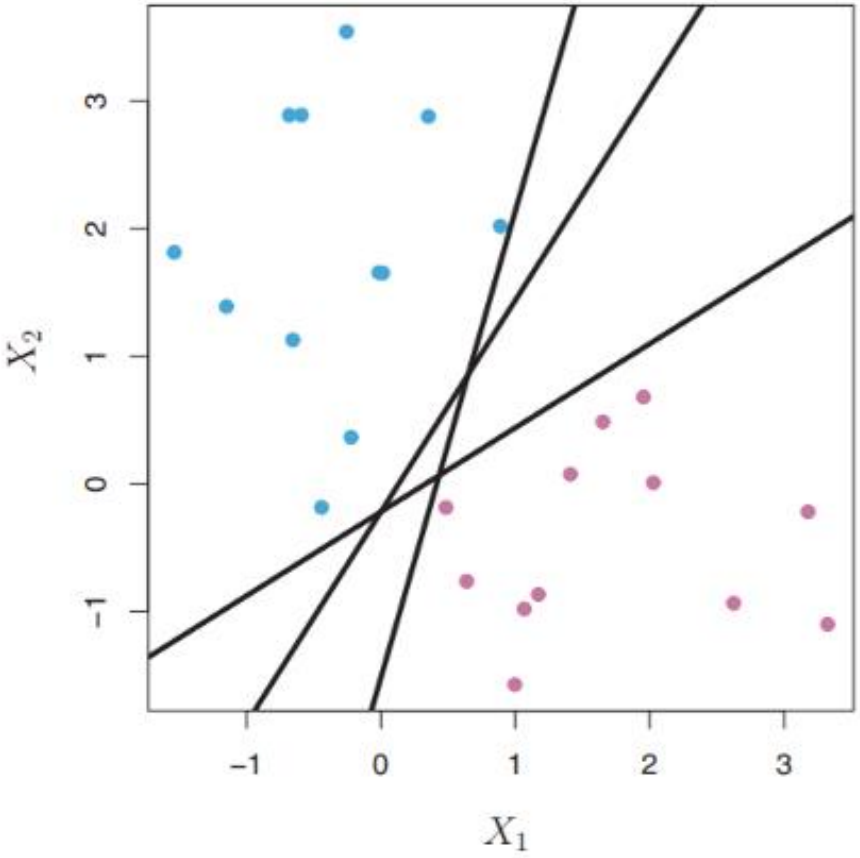


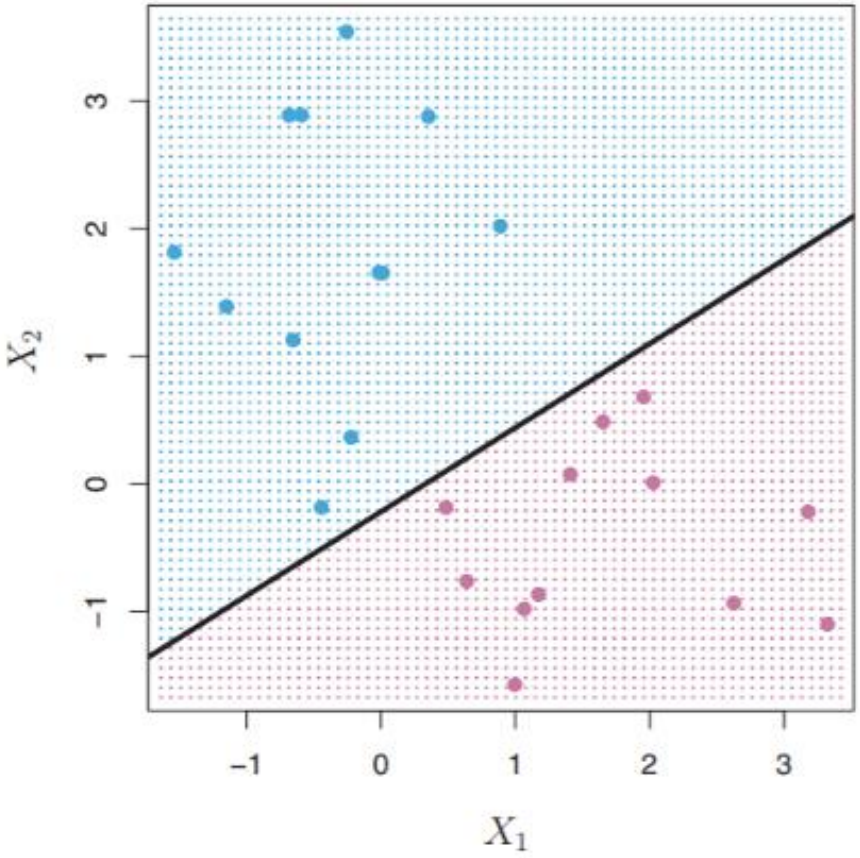
source: https://www.youtube.com/watch?v=EuBBz3bI-aA

Maximal margin classifier

What is a hyperplane?

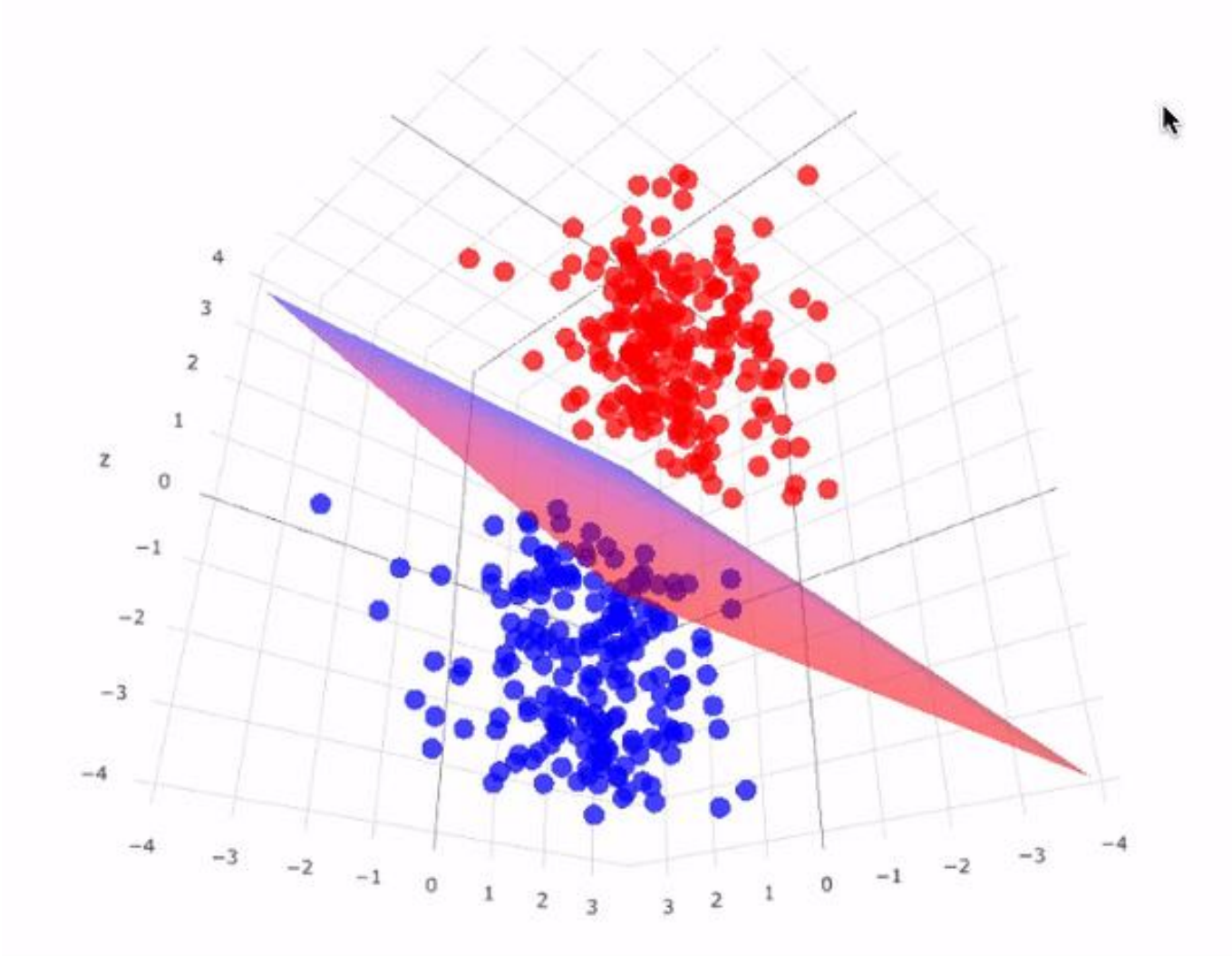
2-D



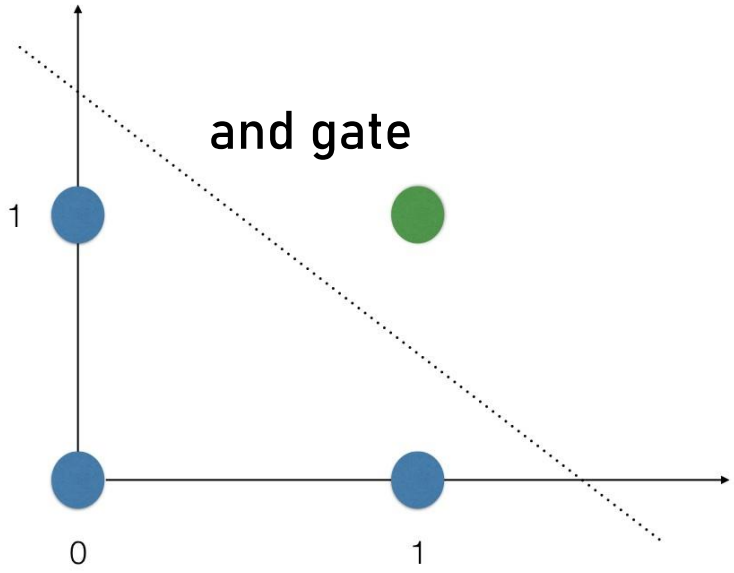


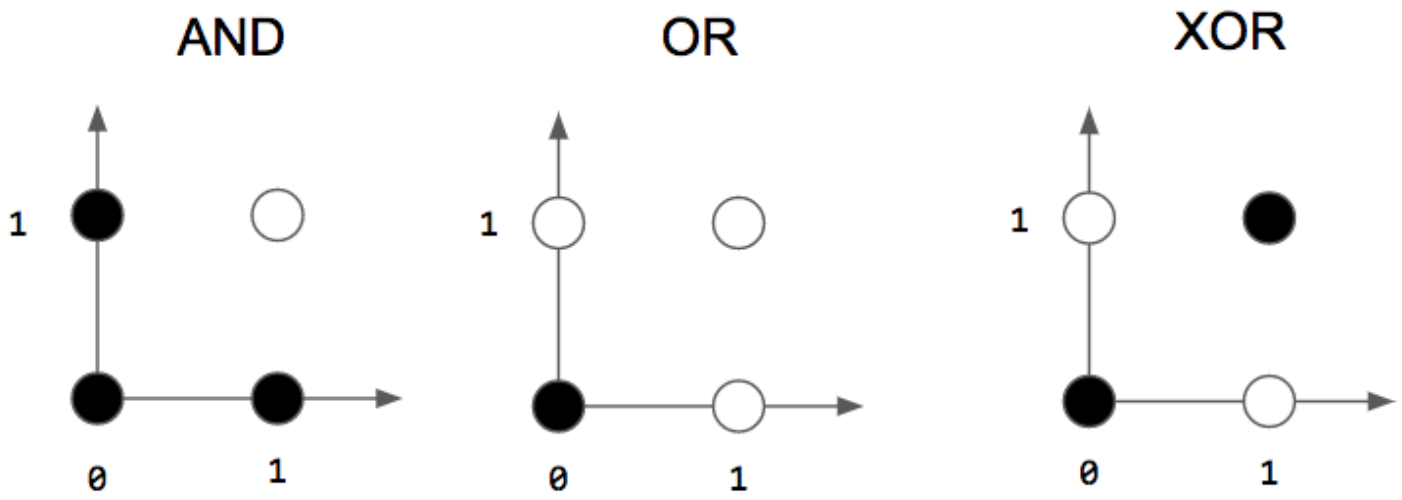
Why is it called a hyperplane?

Hyperplane in 3-D



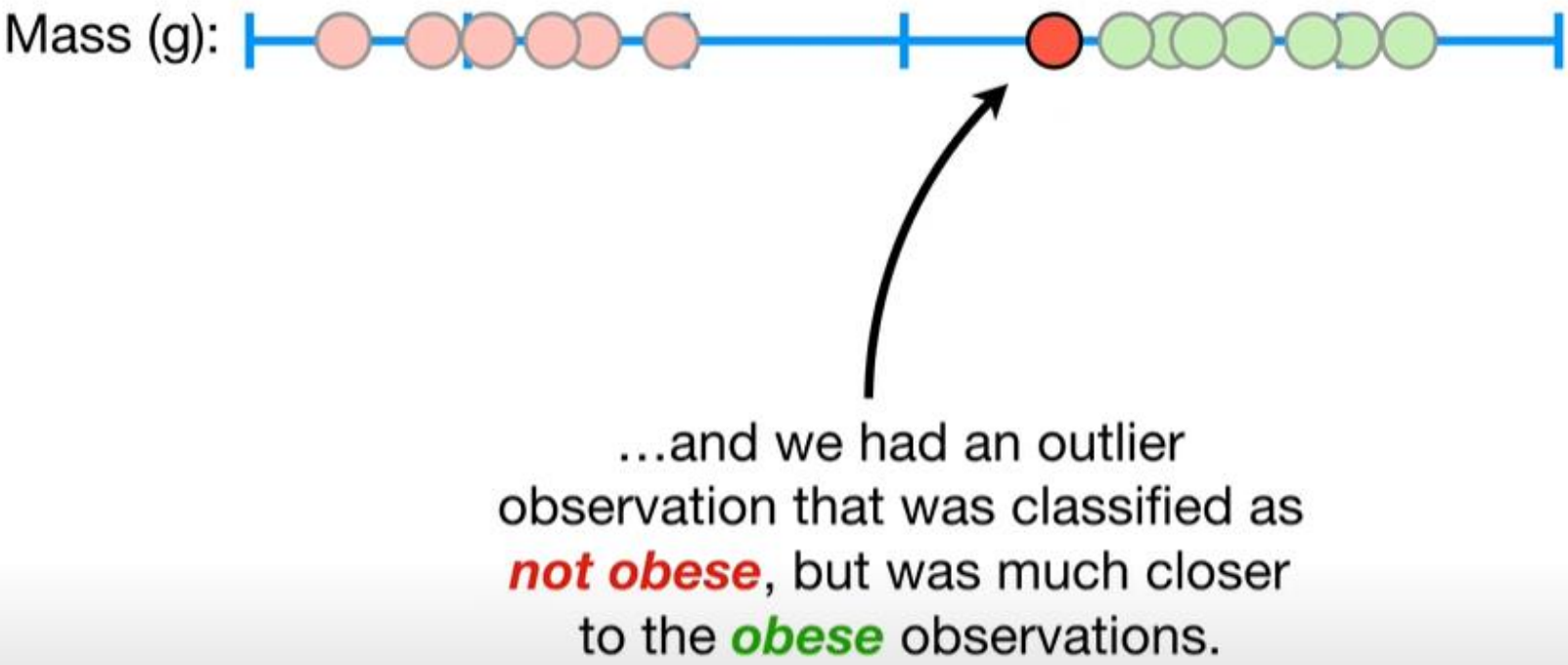
Logic gates





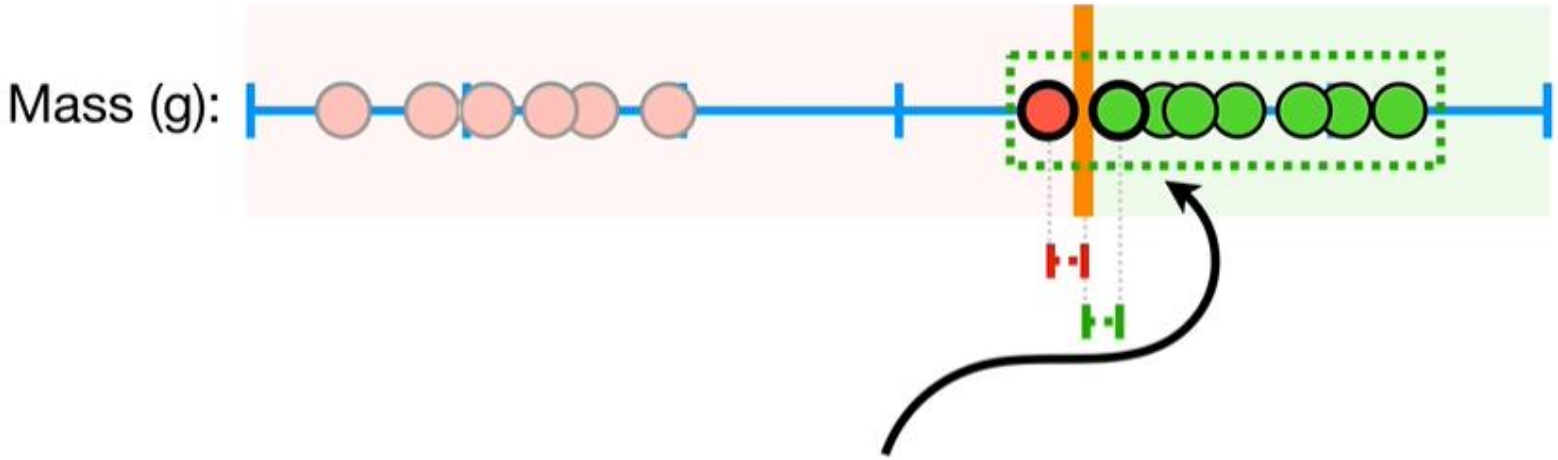
Soft Margin

Problems with Maximal margin classifier



Soft Margin

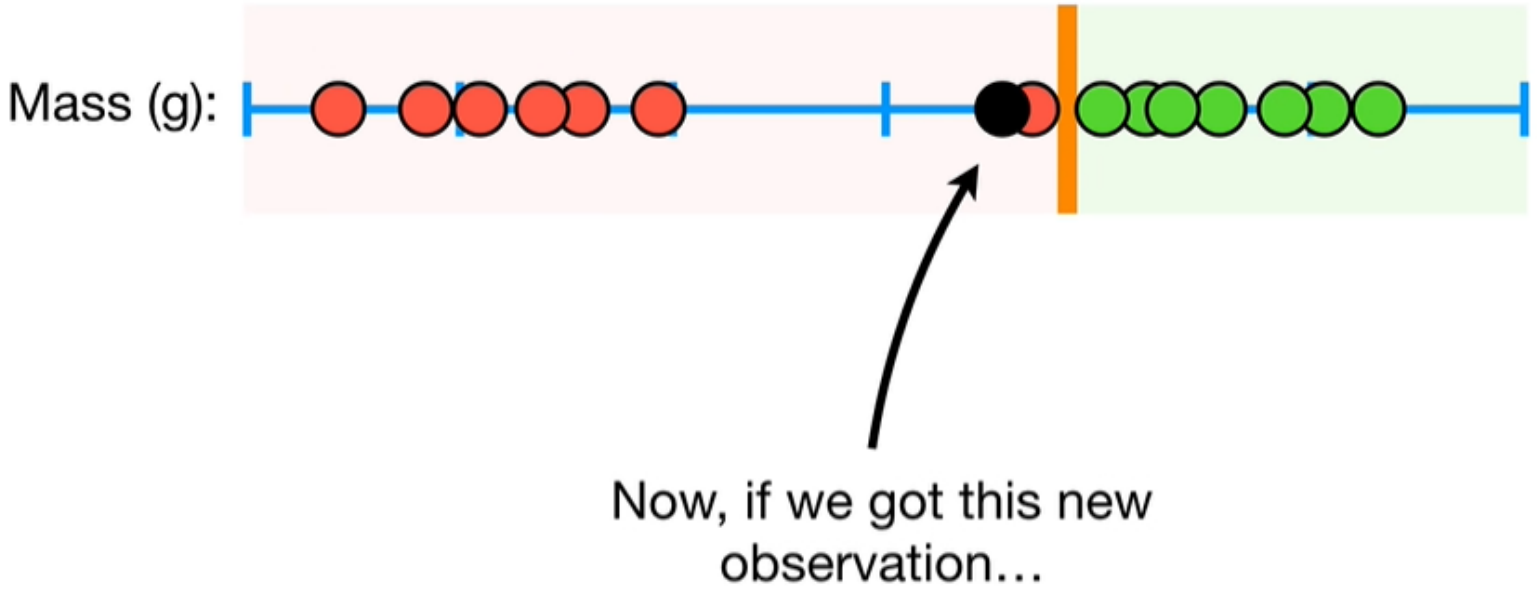
Problems with Maximal margin classifier



In this case, the **Maximum Margin Classifier** would be super close to the ***obese*** observations…

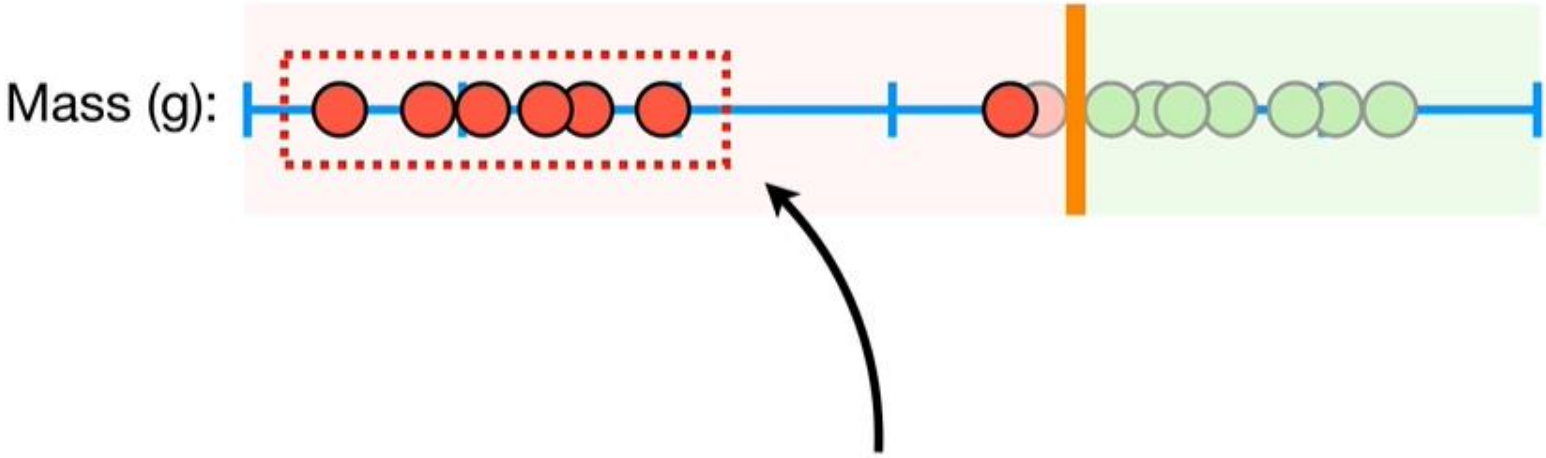
Soft Margin

Problems with Maximal margin classifier



Soft Margin

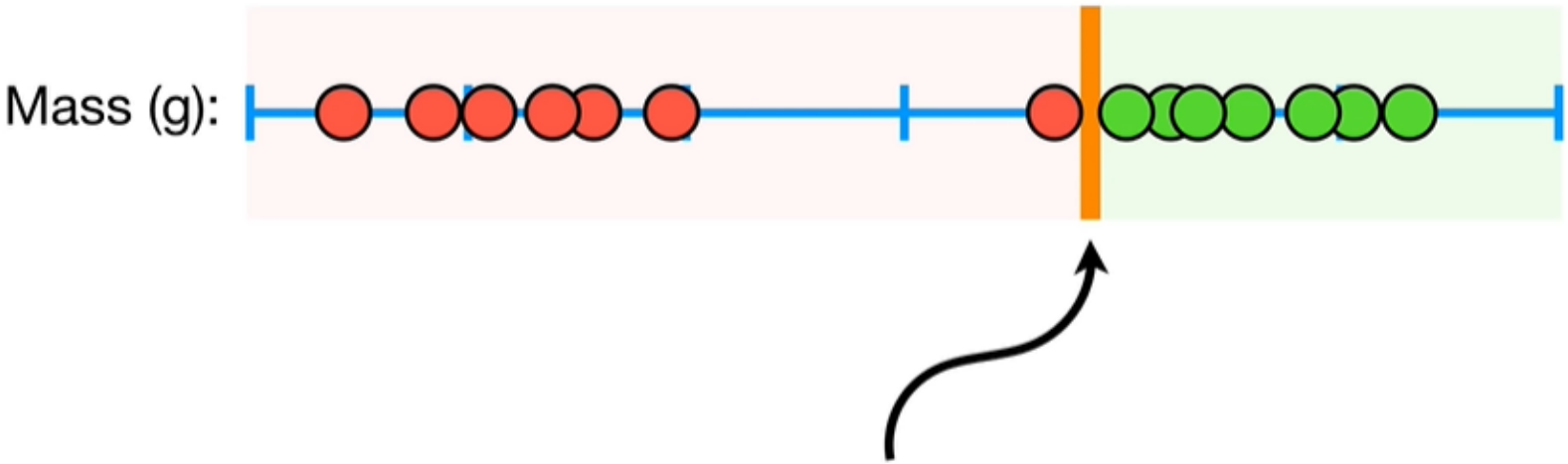
Problems with Maximal margin classifier



…we would classify it as ***not obese***, even though most of the ***not obese*** observations are much further away than the ***obese*** observations.

Soft Margin

Problems with Maximal margin classifier

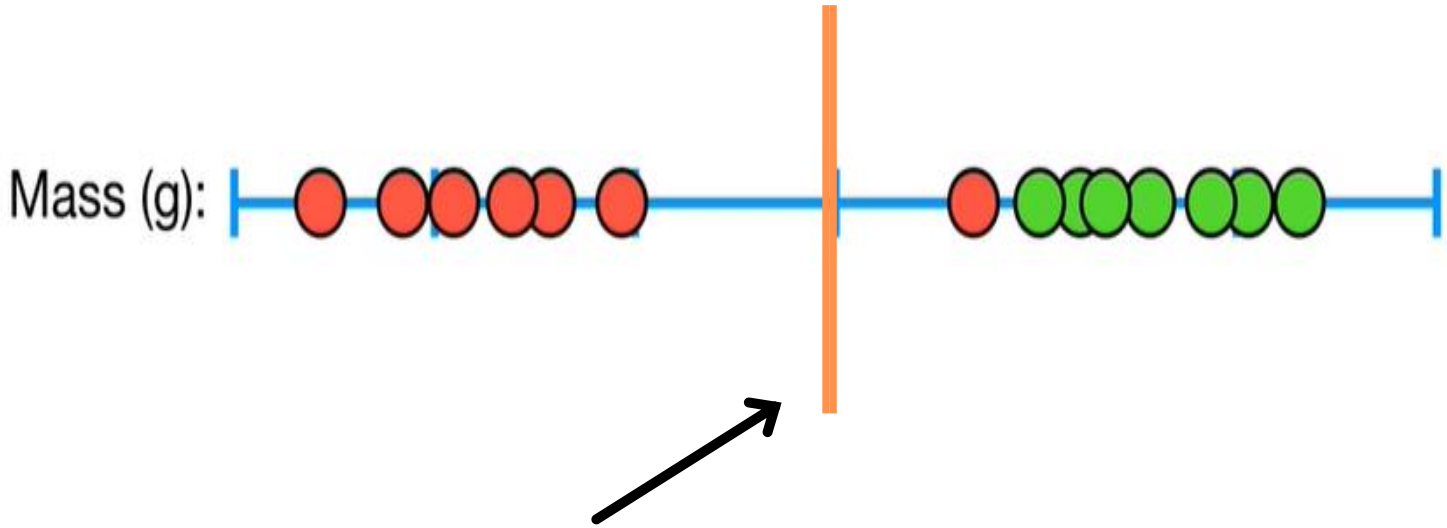


So **Maximal Margin Classifiers** are *super sensitive to outliers* in the training data and that makes them pretty lame.

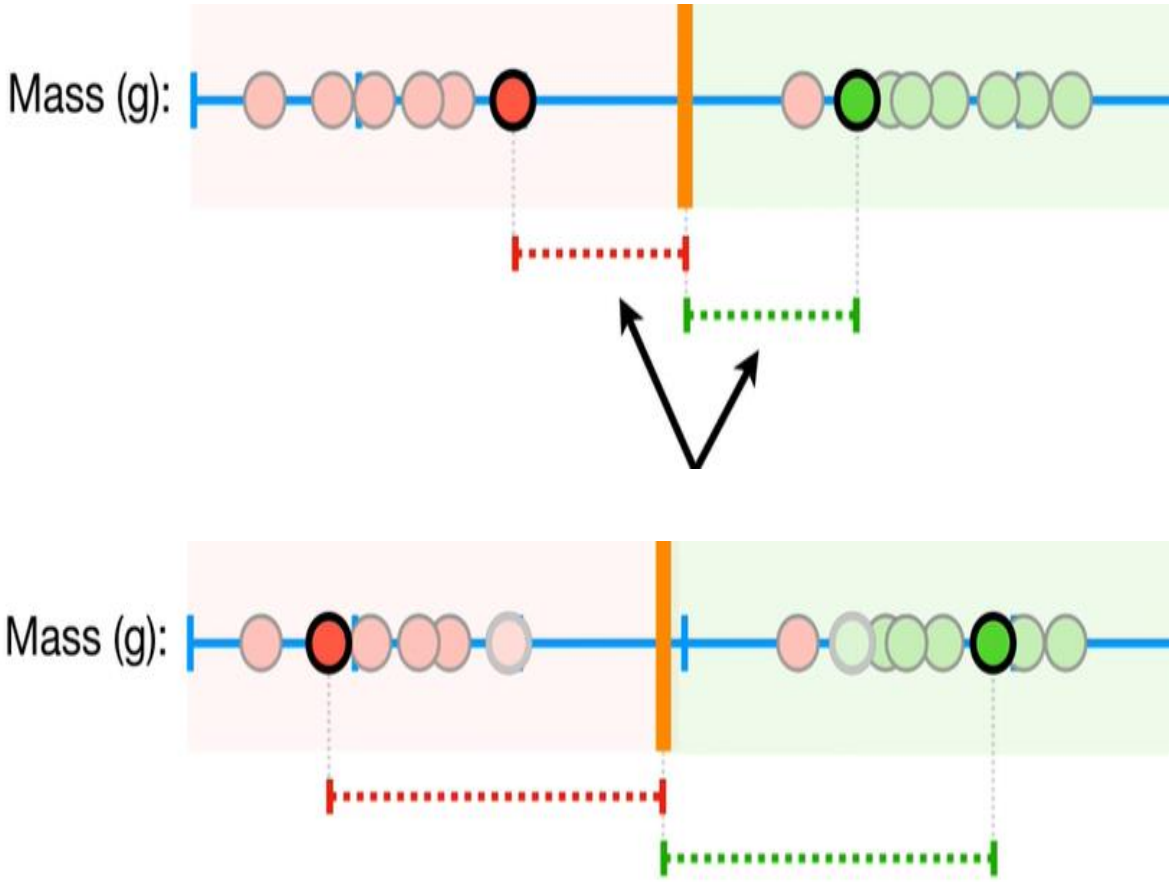
Soft Margin

Why Soft Margins?

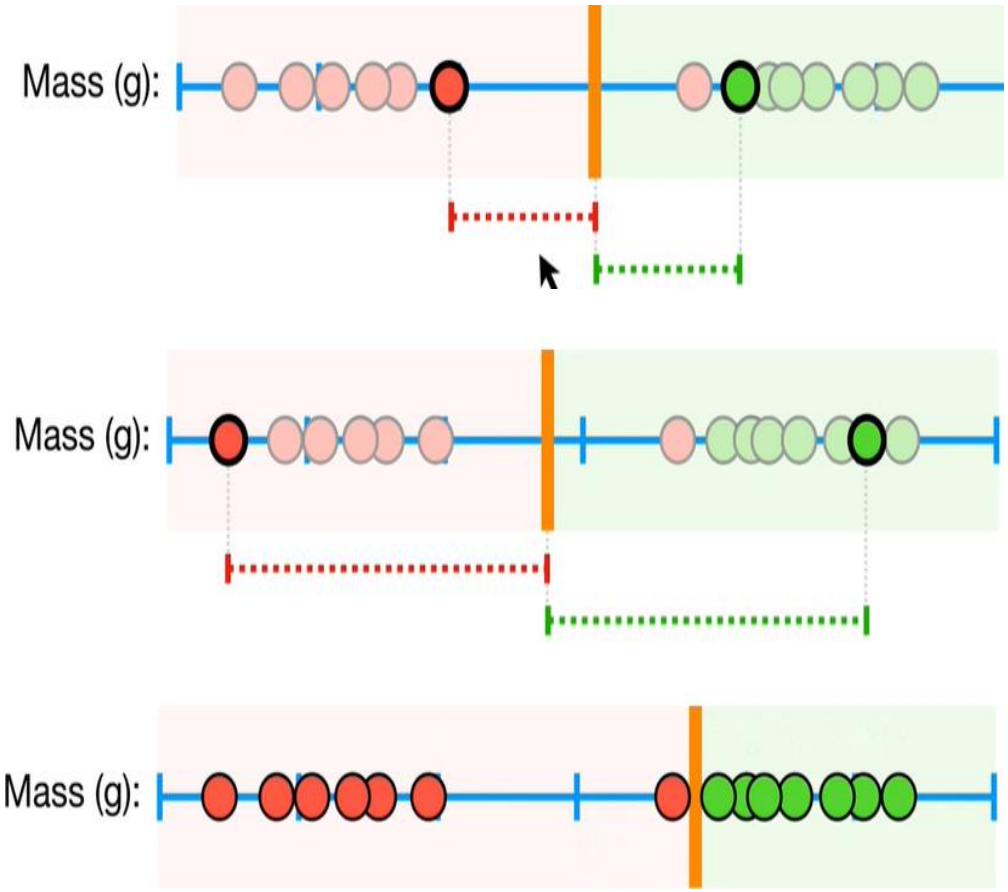
We've to draw a margin which allows misclassification



Why only here, and how is it determined?

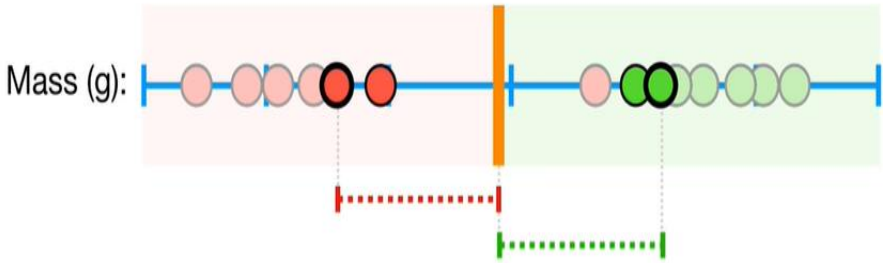


Which among them is better?

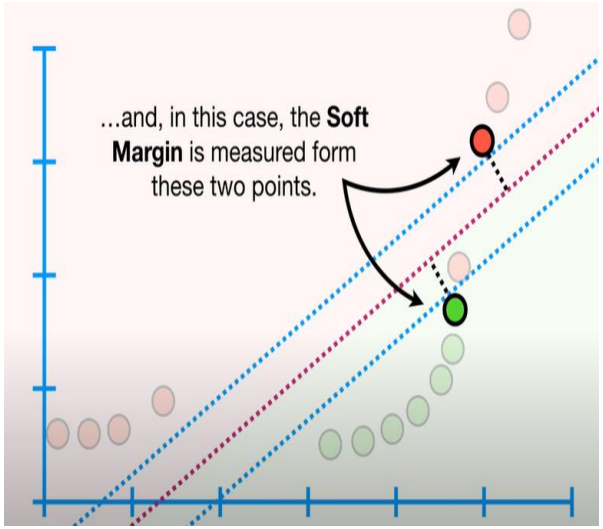


Cross-Validation is done with each pair of points to find the optimum

Support vector Classifiers



Soft Margin is a Point

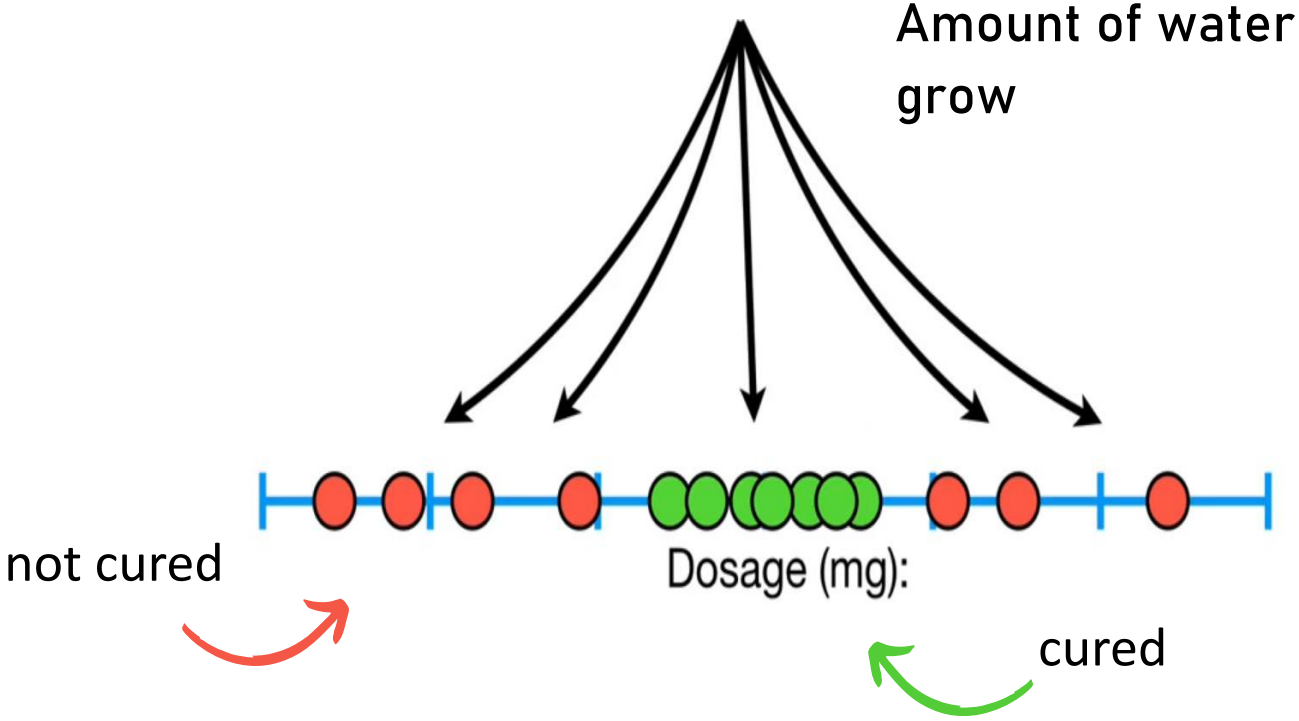


Soft Margin is a line

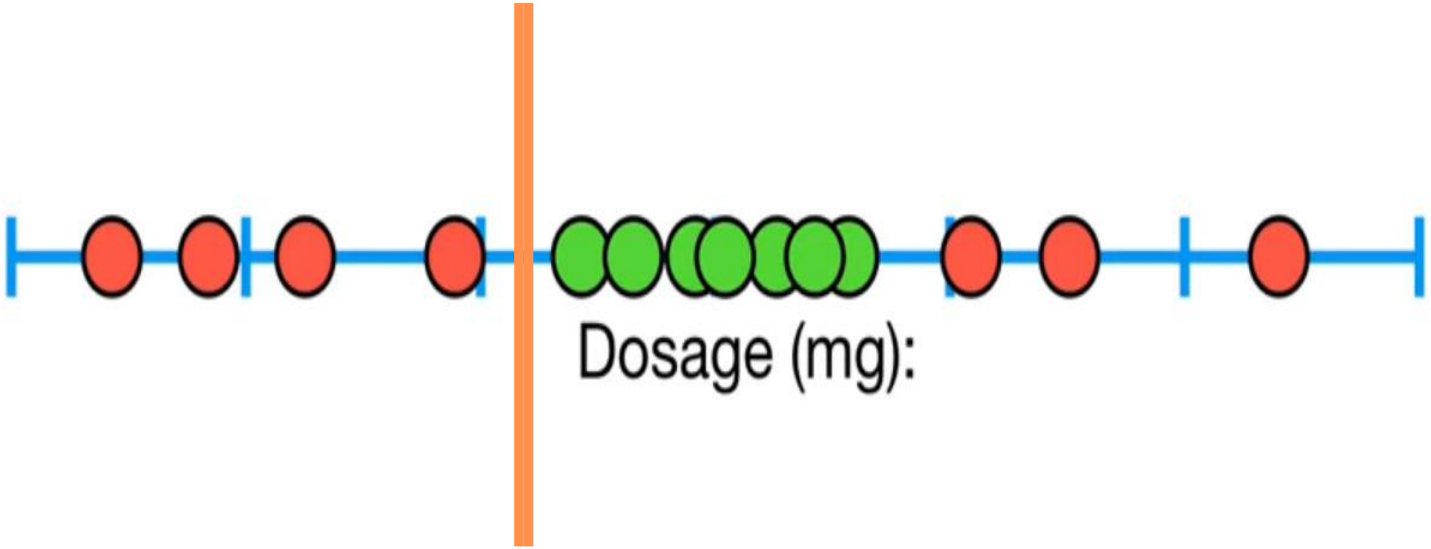
A plane in case of 3D
and rest are called as hyperplanes

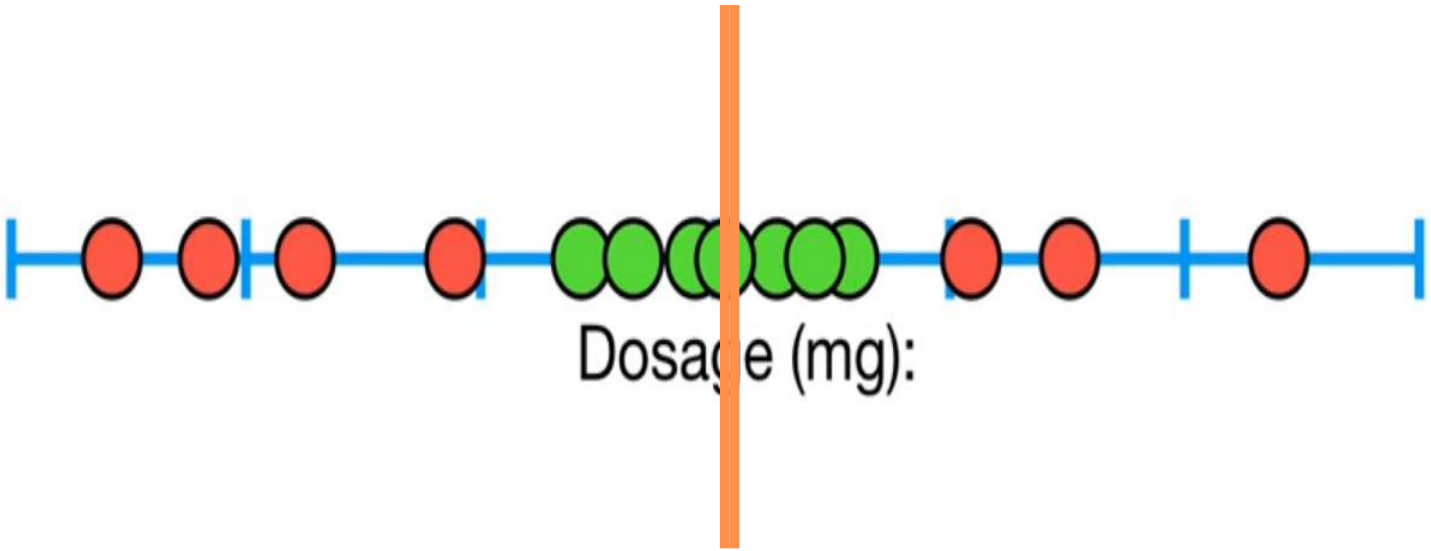
SVM intuition

Example of such cases
Dosage of medicine for recovery
Amount of water for a plant to grow

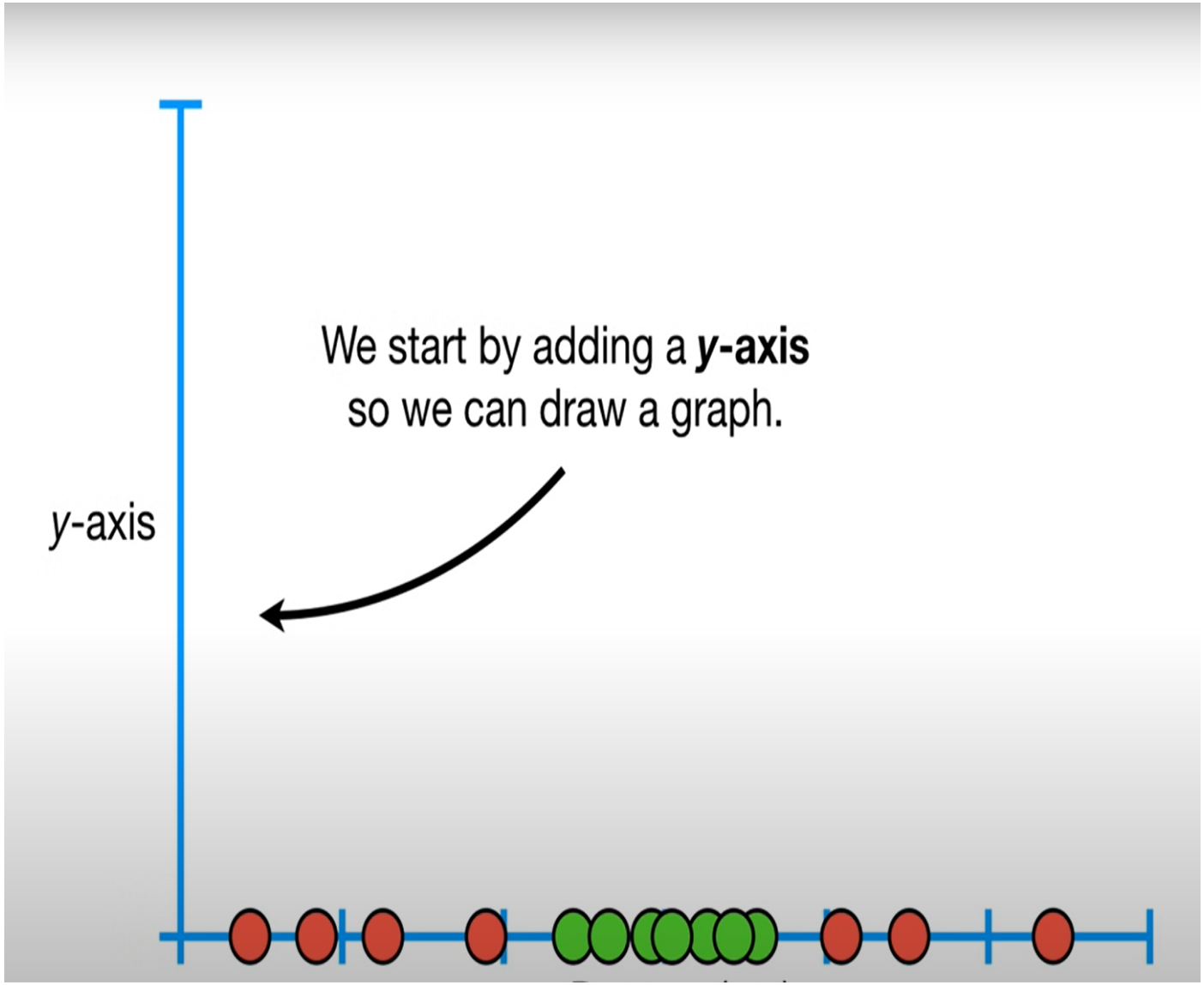


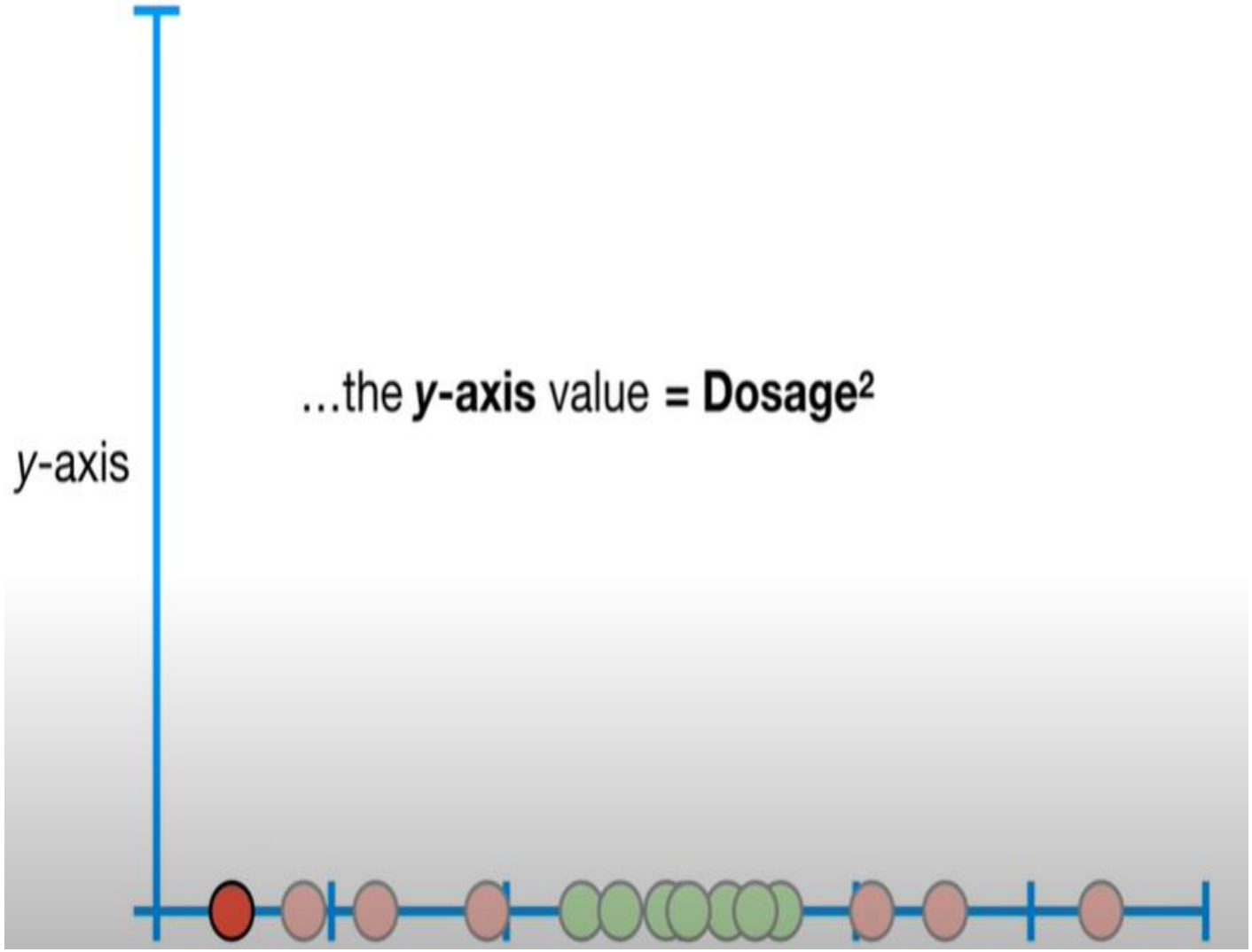
Does it work for all kinds of data?

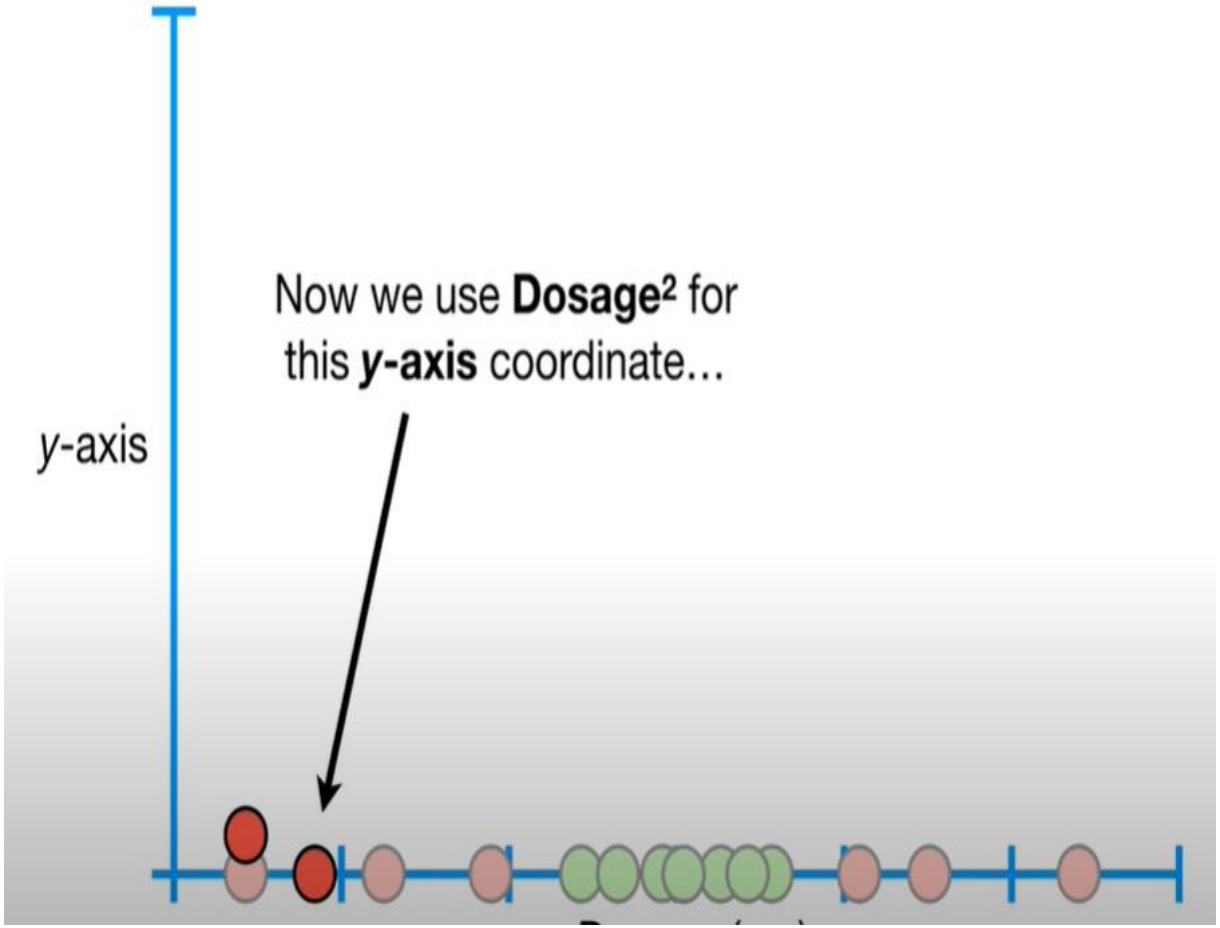


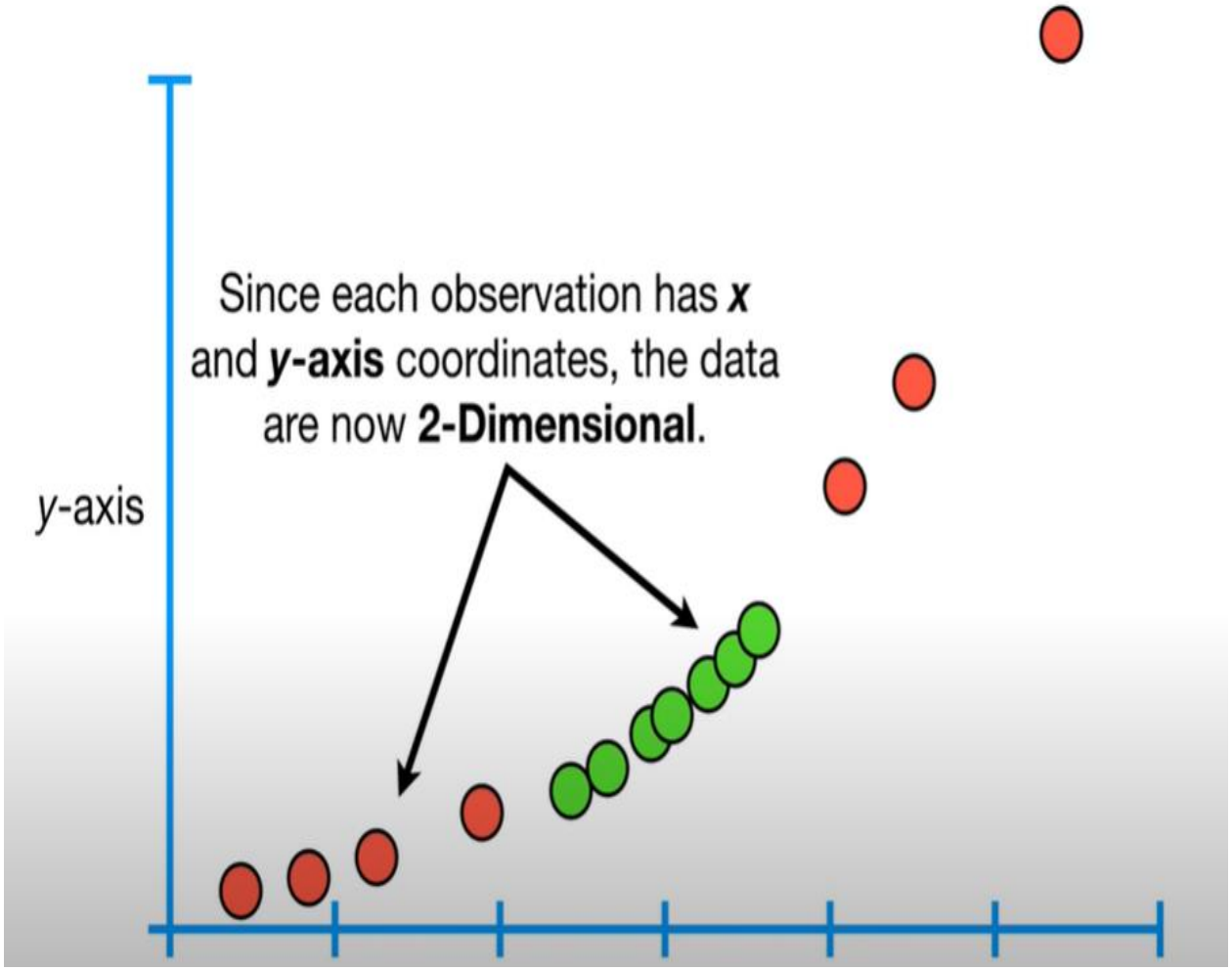


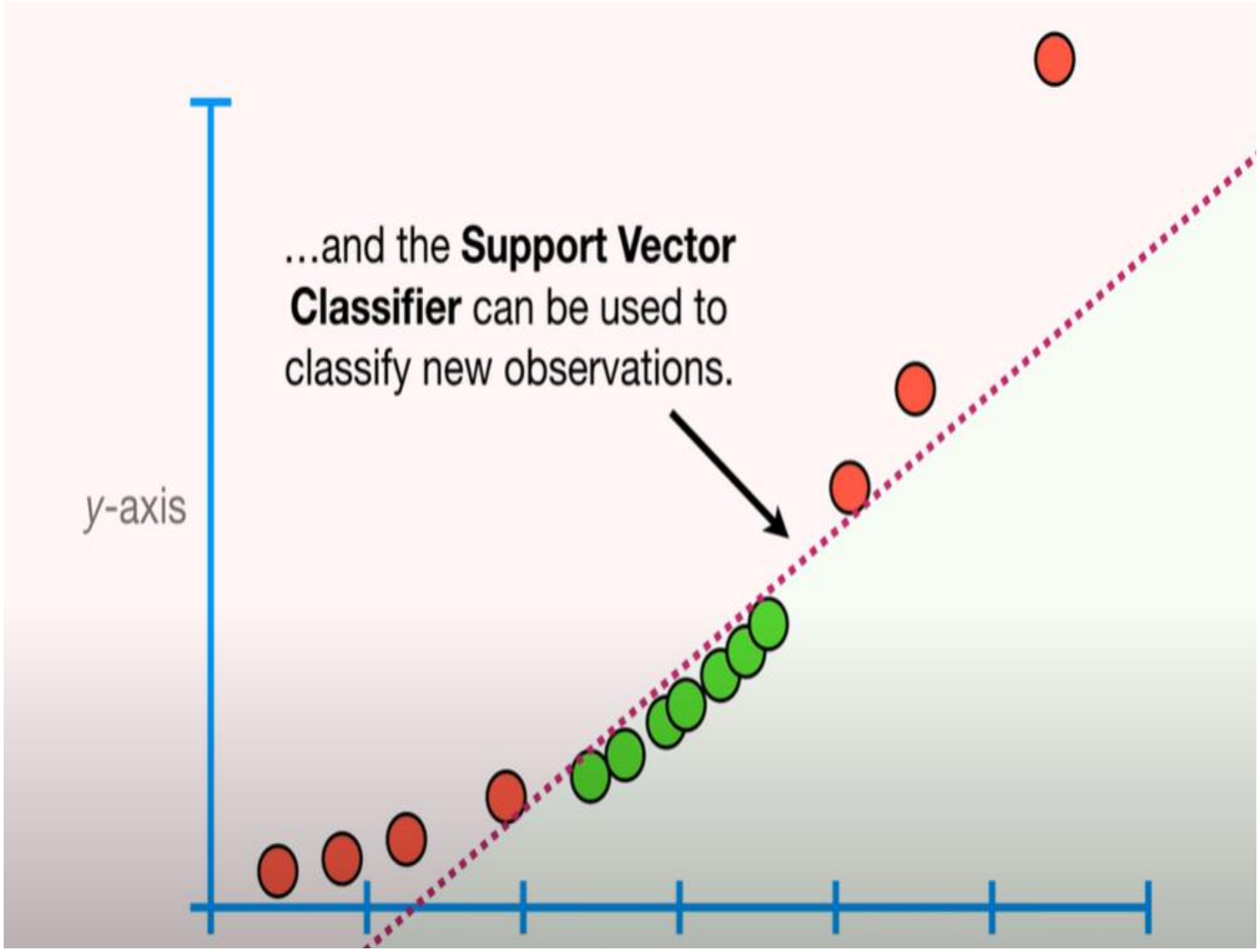
Can we do better?

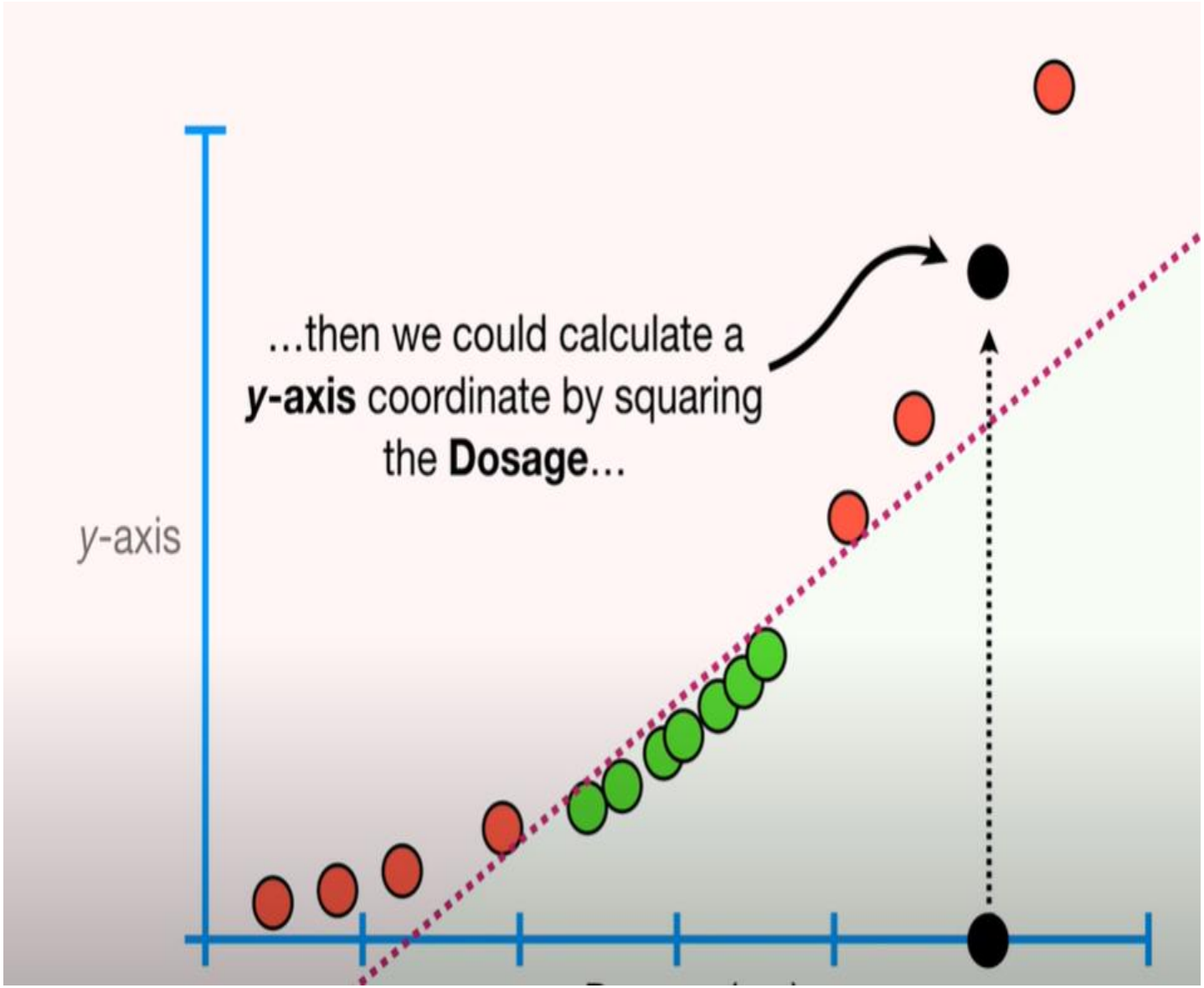

We start by adding a **y-axis**
so we can draw a graph.
y-axis

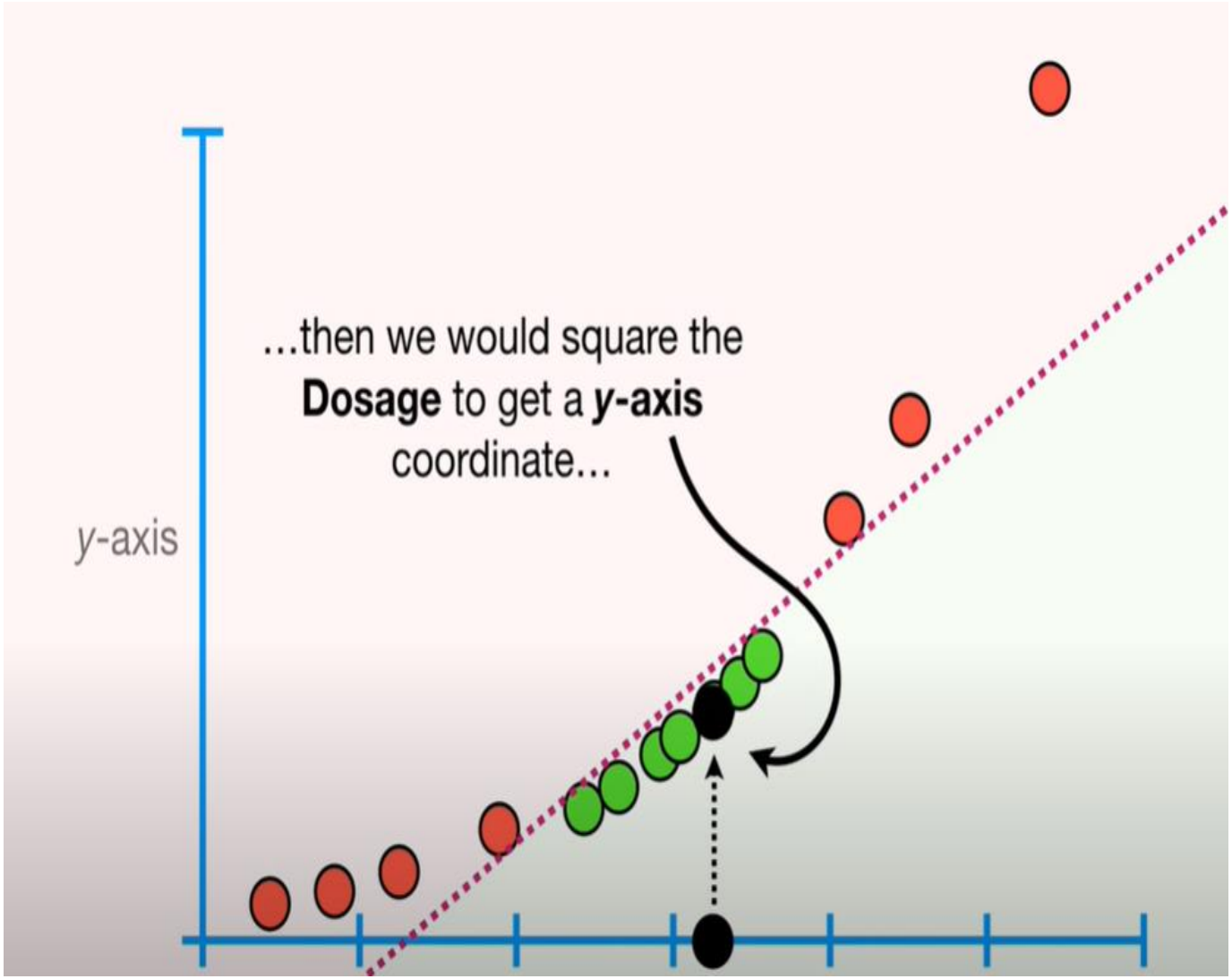

...the **y-axis** value = **Dosage²**
y-axis

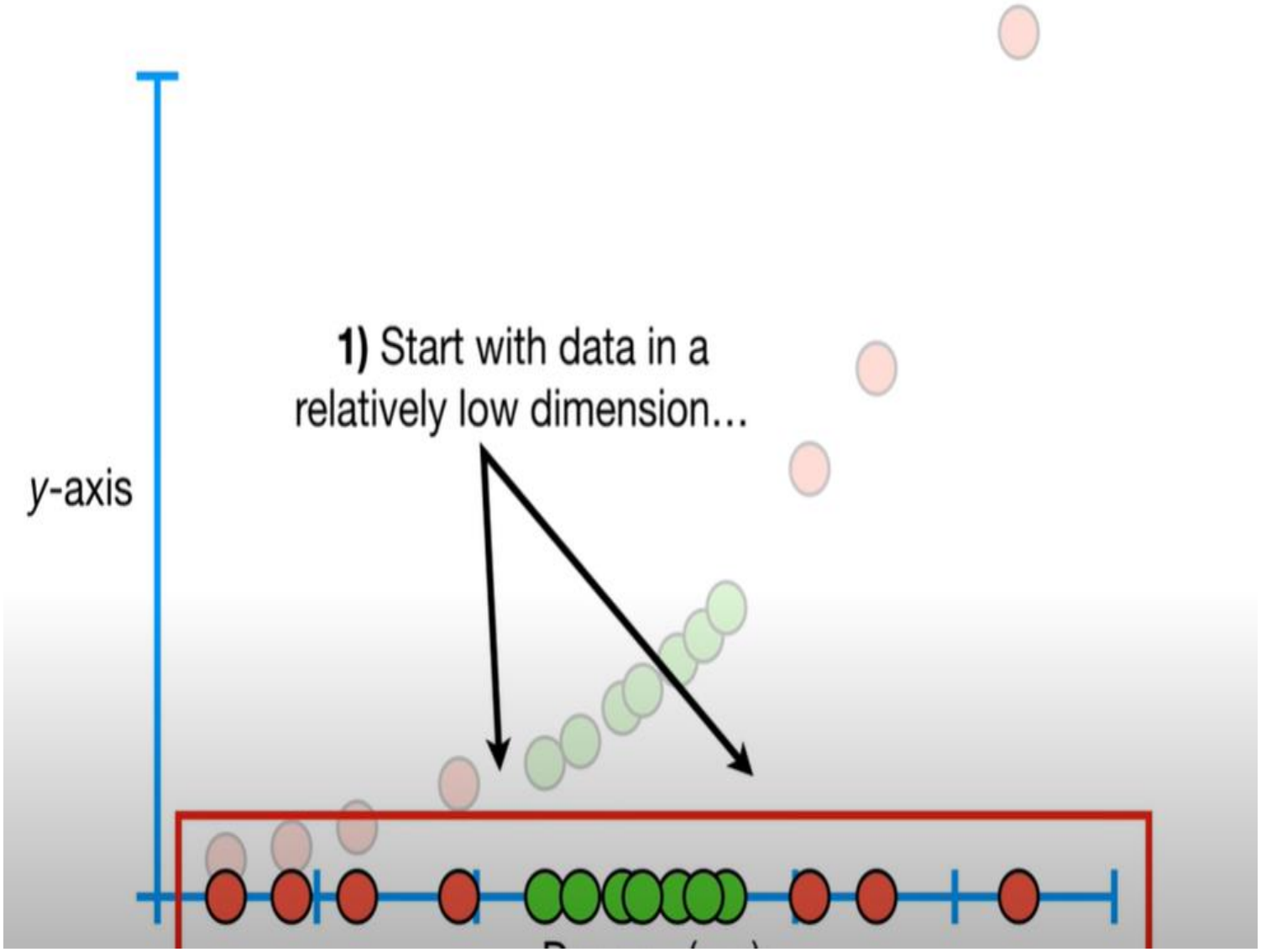

Now we use **Dosage²** for this ***y*-axis** coordinate…
y-axis

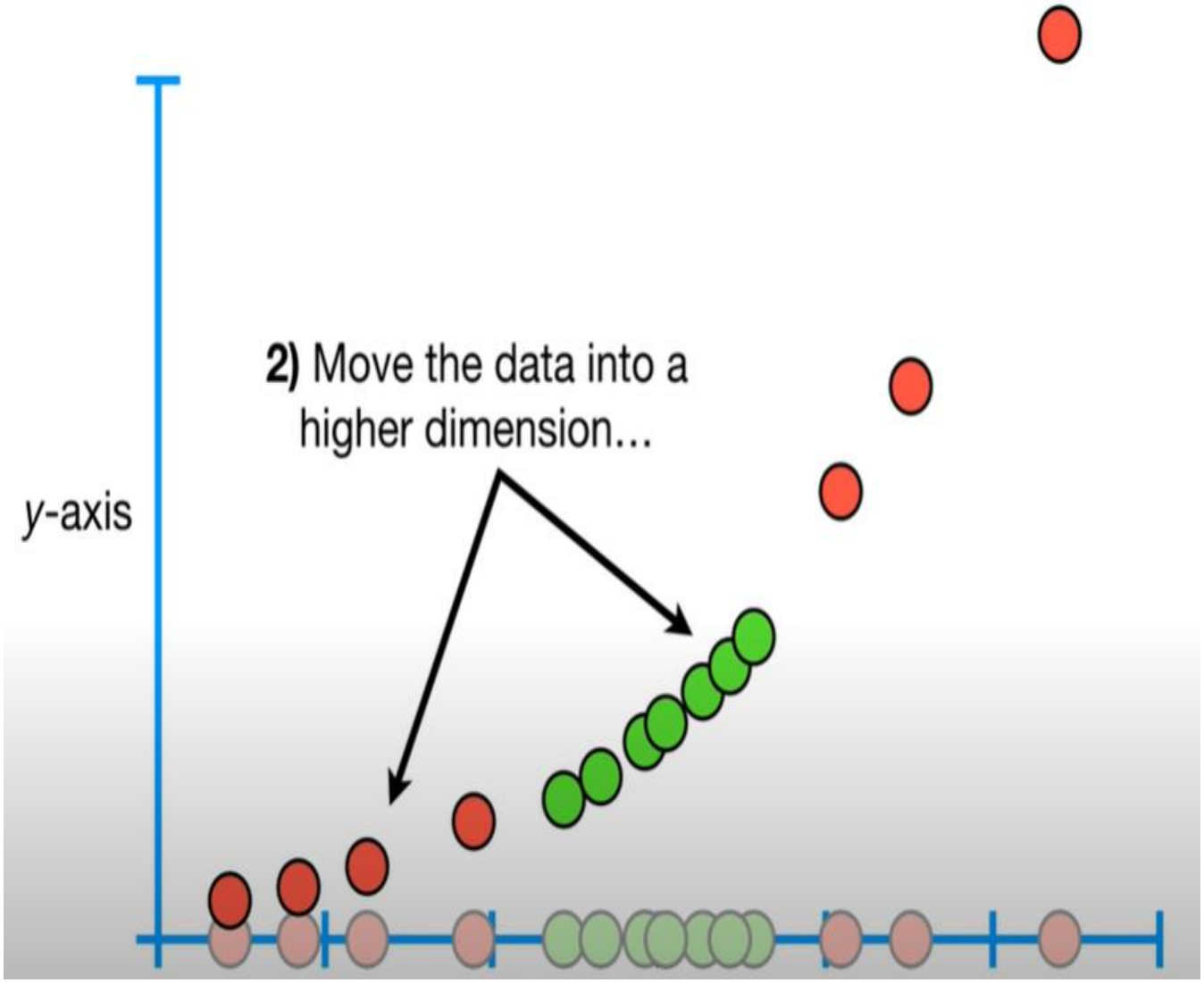

Since each observation has *x* and ***y*-axis** coordinates, the data are now **2-Dimensional**.
y-axis

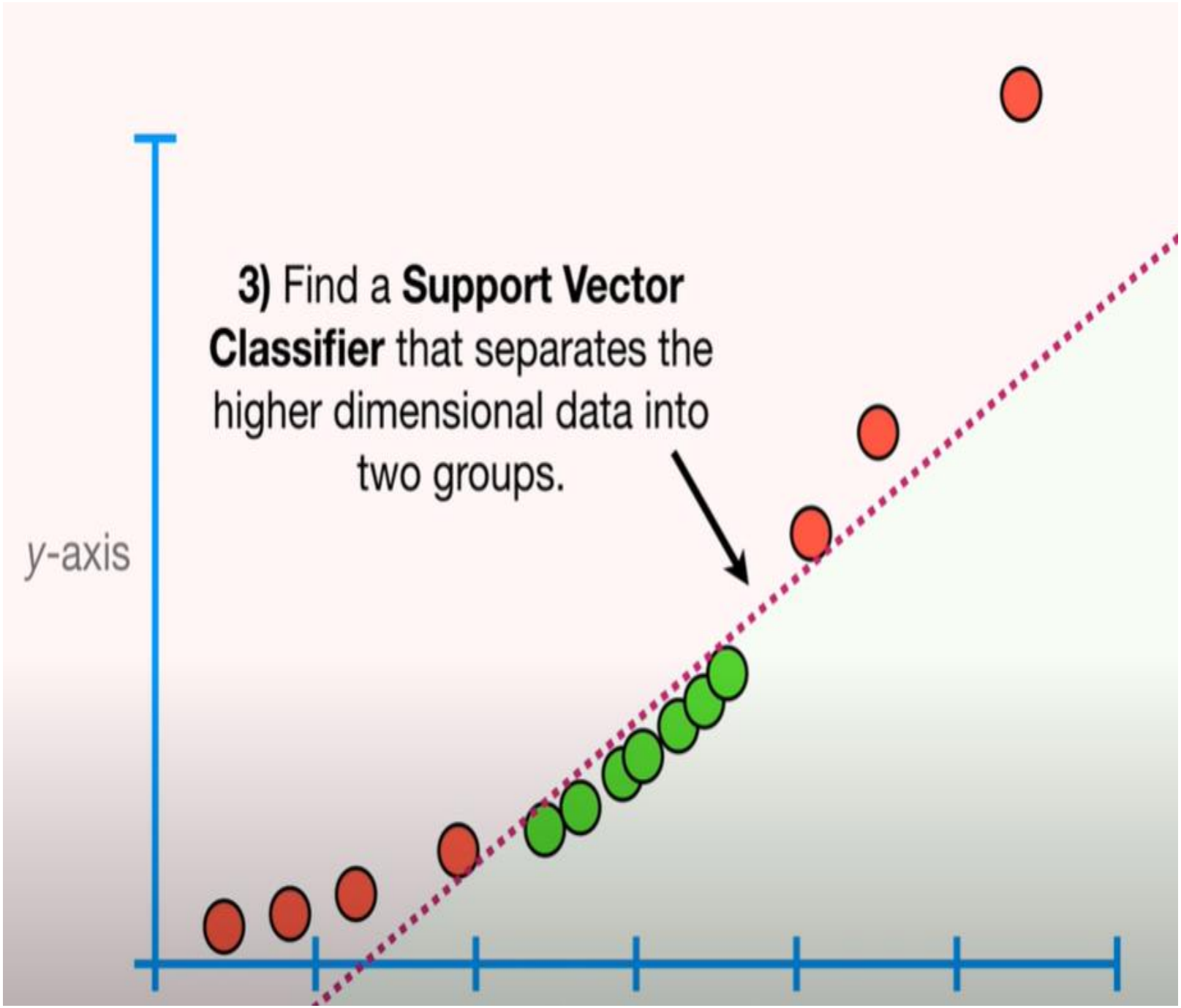

...and the **Support Vector Classifier** can be used to classify new observations.
y-axis


...then we could calculate a
y-axis coordinate by squaring
the Dosage...
y-axis


...then we would square the
Dosage to get a ***y*-axis**
coordinate...
y-axis


1) Start with data in a relatively low dimension…
y-axis


2) Move the data into a higher dimension…
y-axis


3) Find a **Support Vector Classifier** that separates the higher dimensional data into two groups.
y-axis

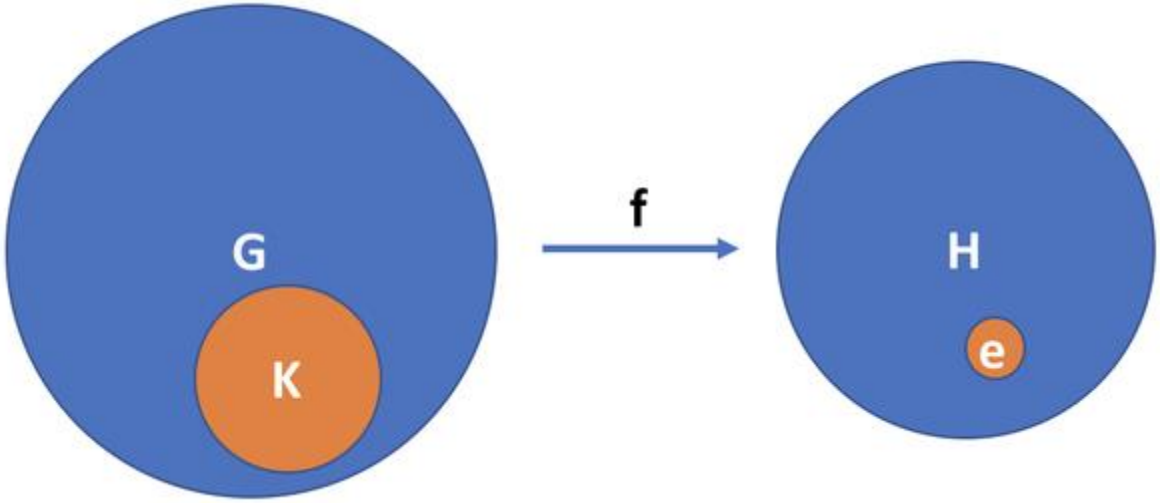
How to choose a function?

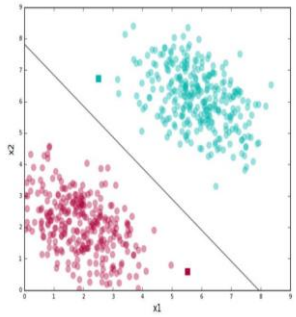
“Kernel” is a set of mathematical functions used in Support Vector Machine that provides the window to manipulate the data.

In our case we have used y = x^2, this is a polynomial of x and thus called a polynomial kernel.

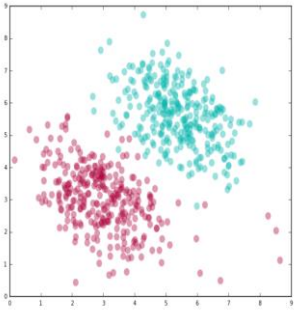
There are several other kernels that are used by SVM, namely Radial Basis Function (RBF), Laplace RBF Kernel, Sigmoid Kernel, etc.

Kernels

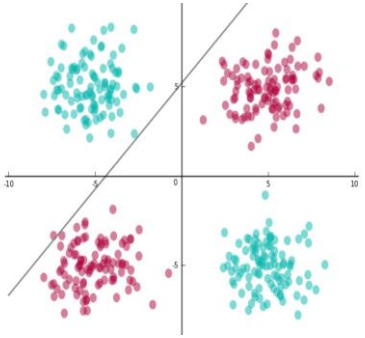




Linearly Separable



Almost Linearly Separable

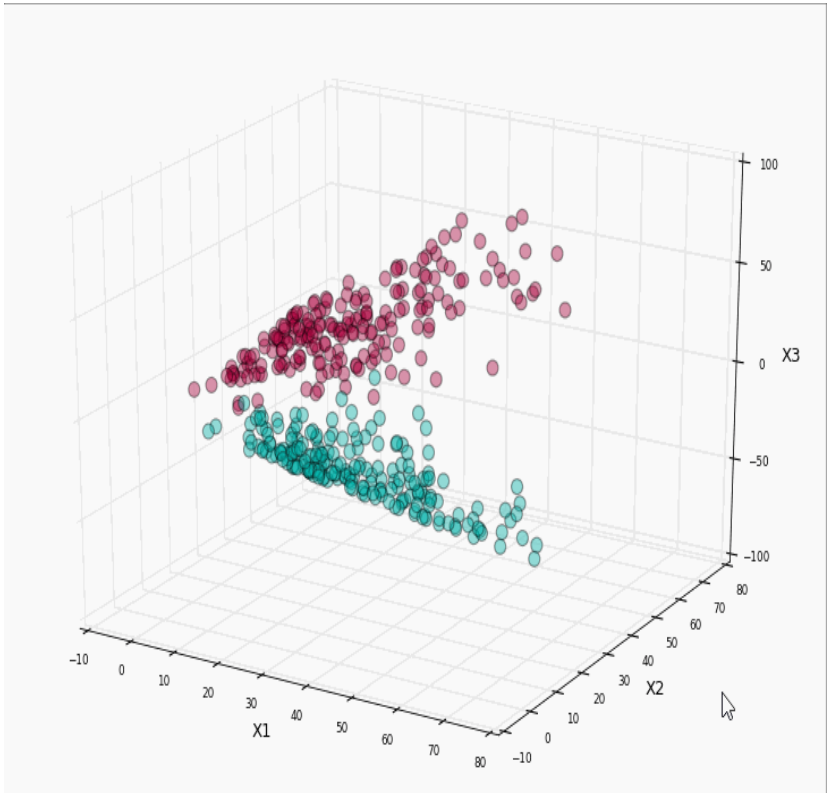


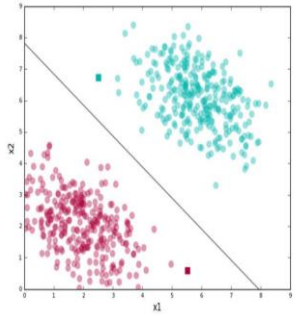
Non Linearly Separable

$$X_1 = x_1^2$$

$$X_2 = x_2^2$$

$$X_3 = \sqrt{2}x_1x_2$$



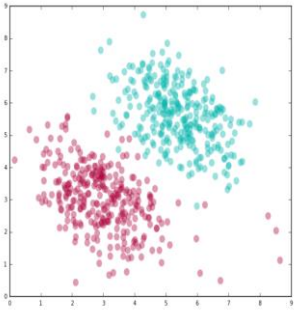


Linearly Separable

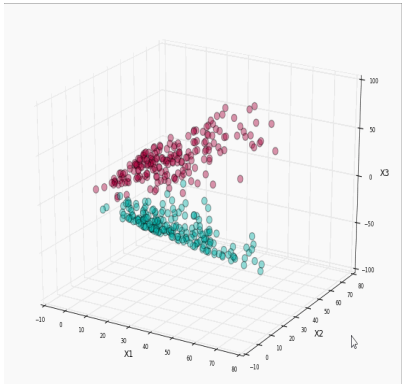
$$X_1 = x_1^2$$

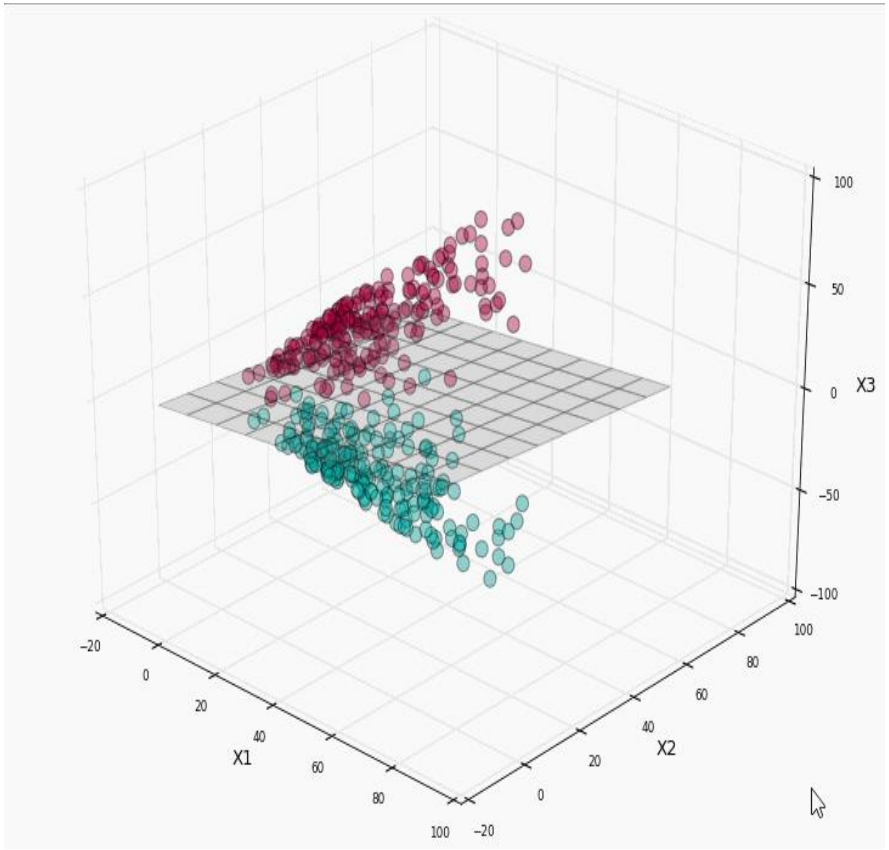
$$X_2 = x_2^2$$

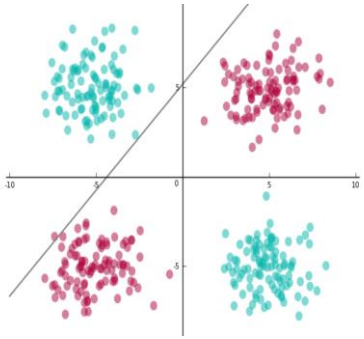
$$X_3 = \sqrt{2}x_1x_2$$



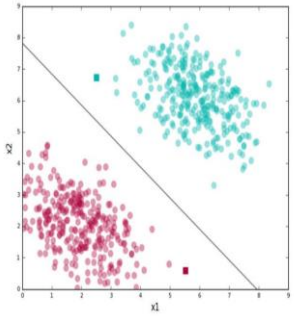
Almost Linearly Separable







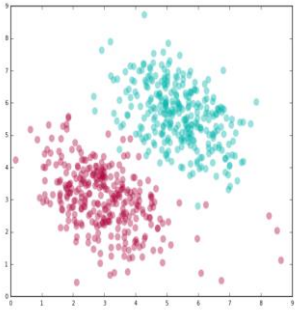
Non Linearly Separable



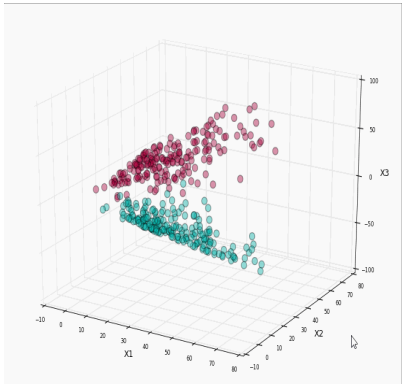
Linearly Separable

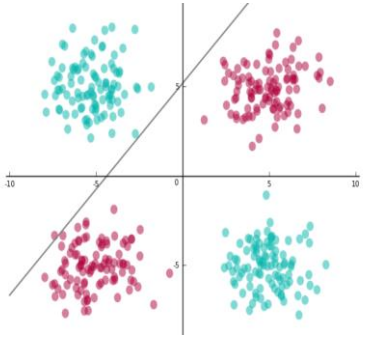
$$X_1 = x_1^2$$
$$X_2 = x_2^2$$
$$X_3 = \sqrt{2}x_1x_2$$

Kernel : set of mathematical functions to manipulate the data

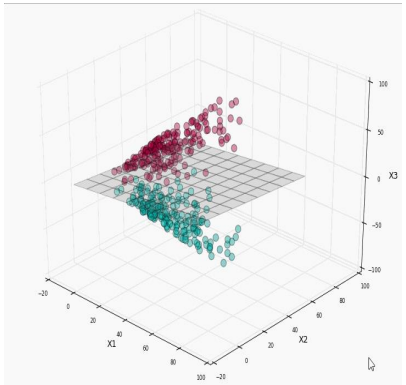


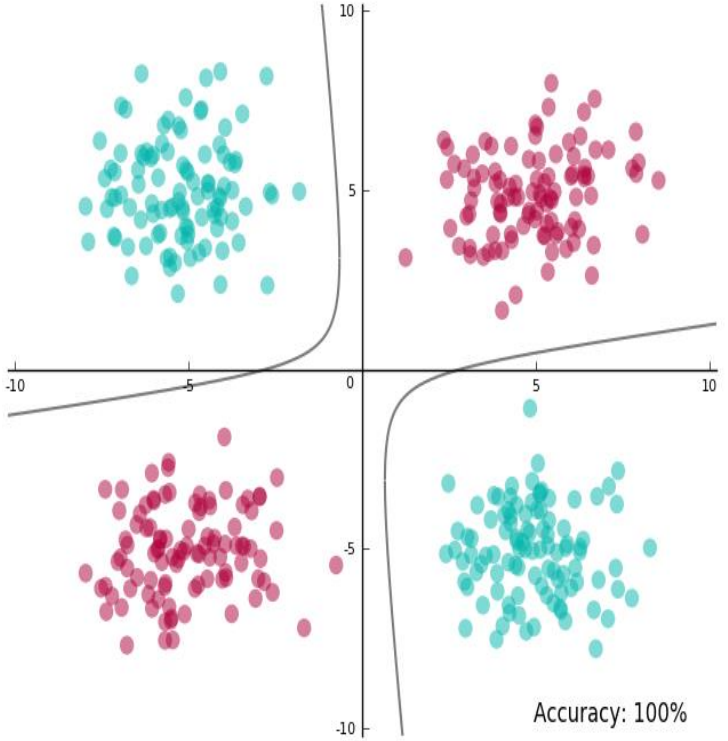
Almost Linearly Separable





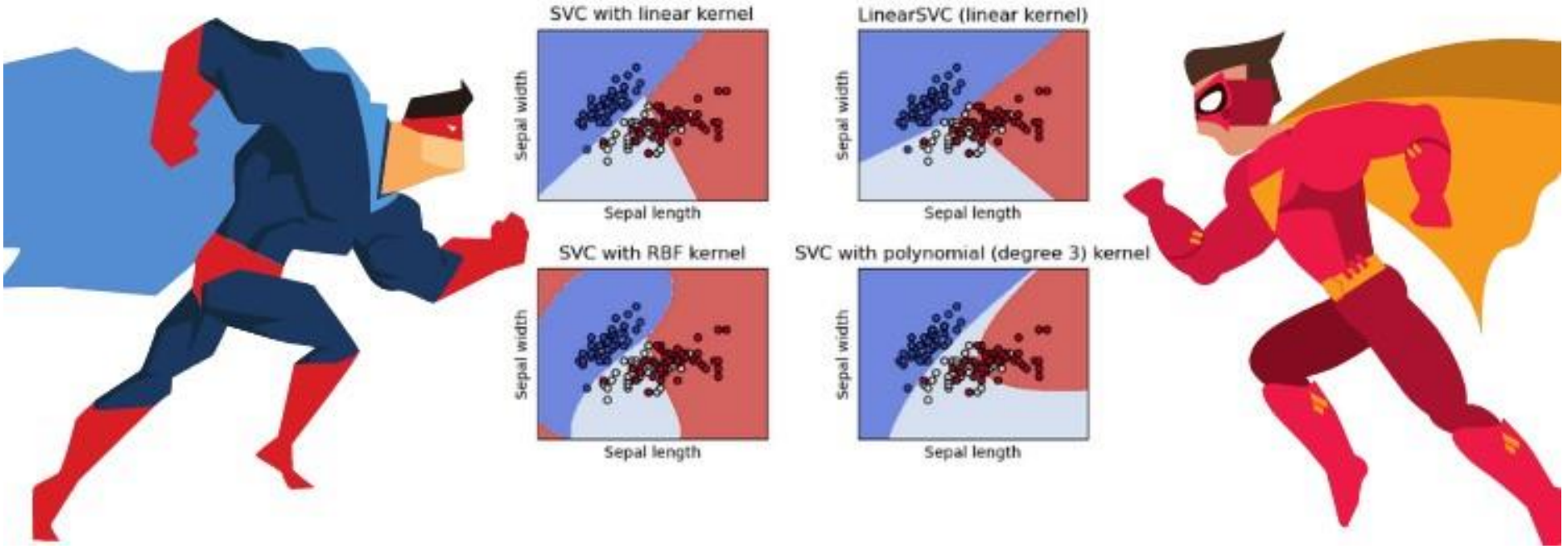
Non Linearly Separable



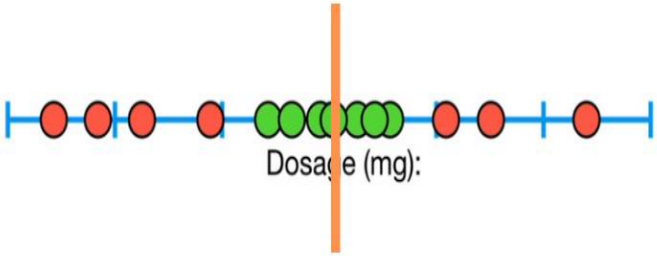


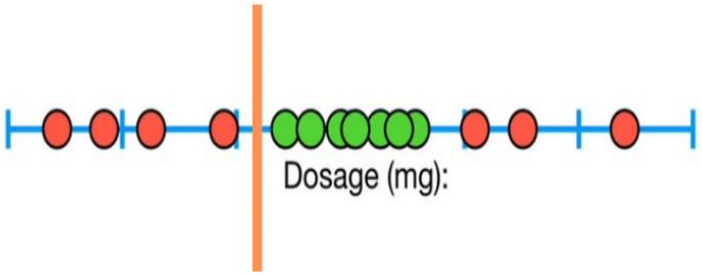
Types of Kernels

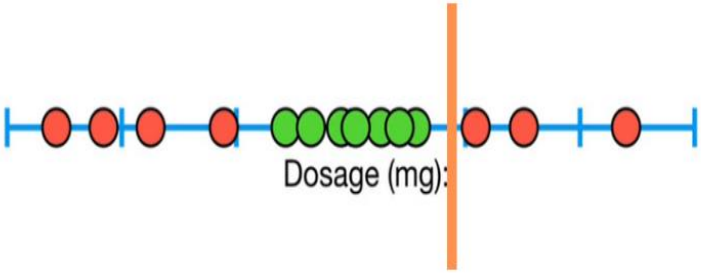
- Popular Kernels
  - Polynomial Kernel
  - Radial Kernel
  - Sigmoid



The Polynomial Kernel



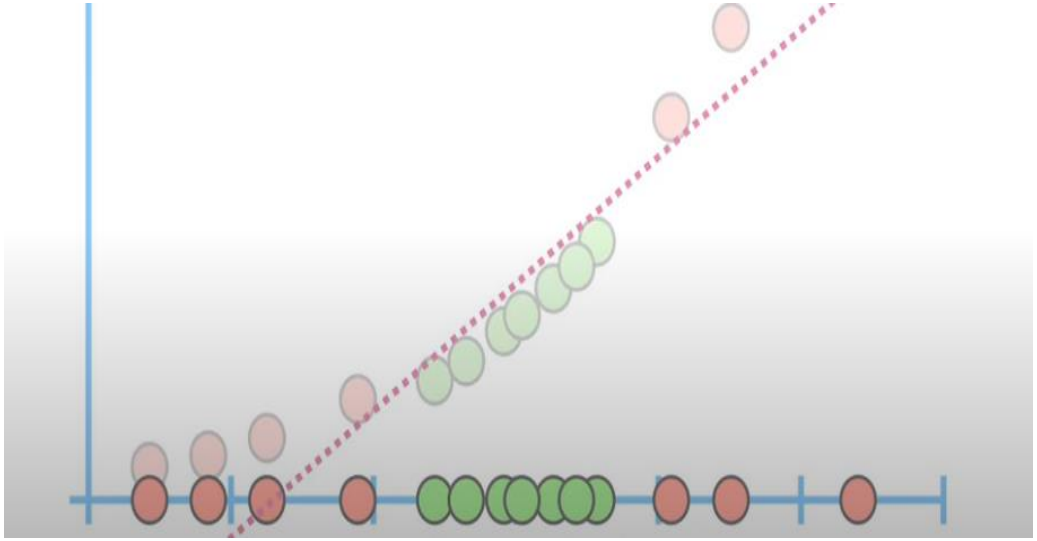




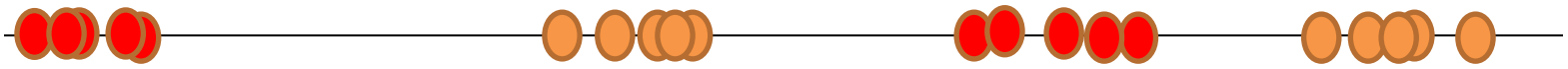
The Polynomial Kernel

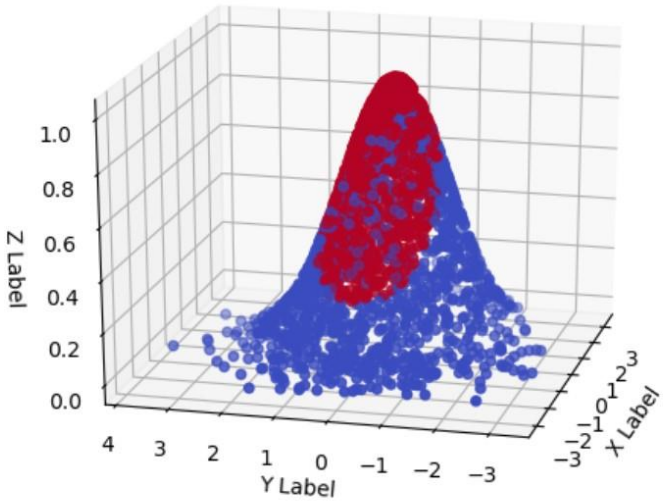
$$(a \times b + r)^d$$

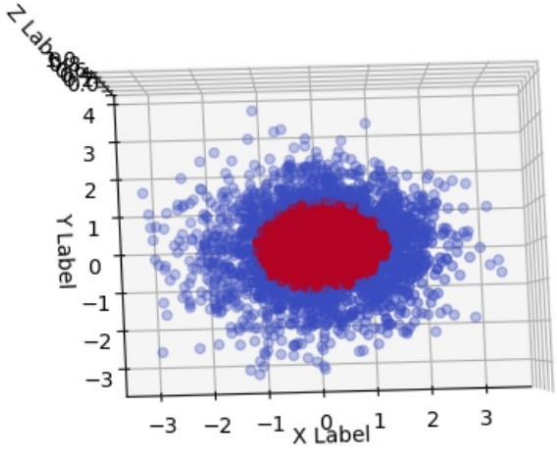
- **a** and **b** – two different observations on the dataset
- **r** – coefficient of the polynomial
- **d** – degree of the polynomial



What if we have data like this?





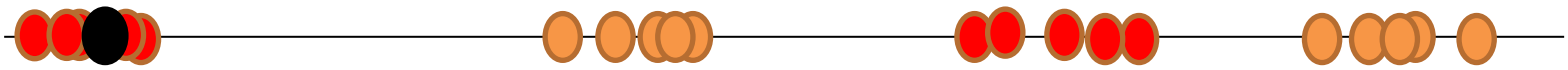


The Radial Kernel

$$e^{-\gamma(a-b)^2}$$

- Finds Support Vector Classifiers in infinite dimensions
- For samples like the below data, Radial Kernel behaves like a **Weighted Nearest Model**

- a and b – observations
- Gamma – scales the influence




THANK YOU

Machine Learning (19CSE305)

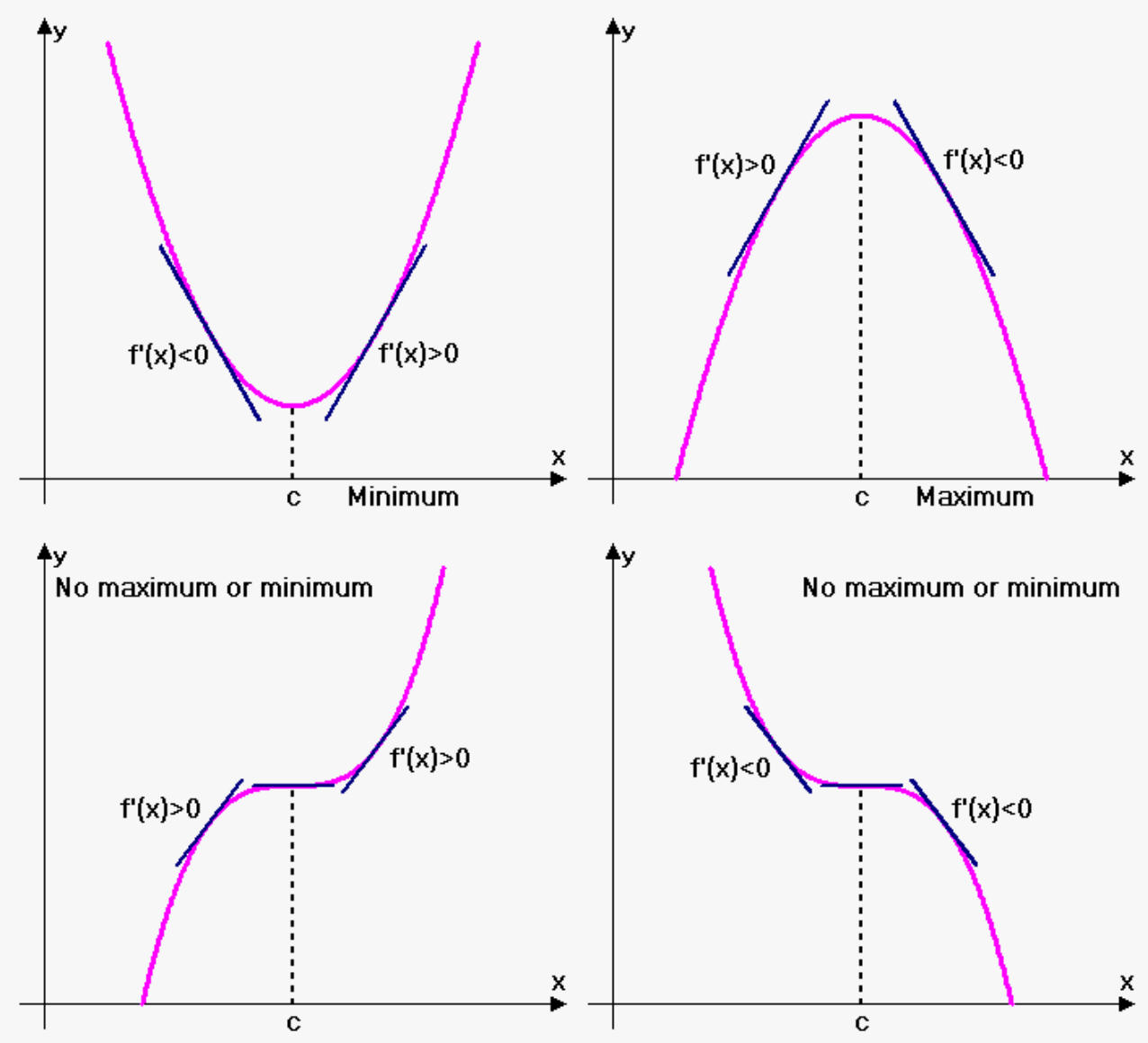
Error Surface, Parameter Optimization & SVM

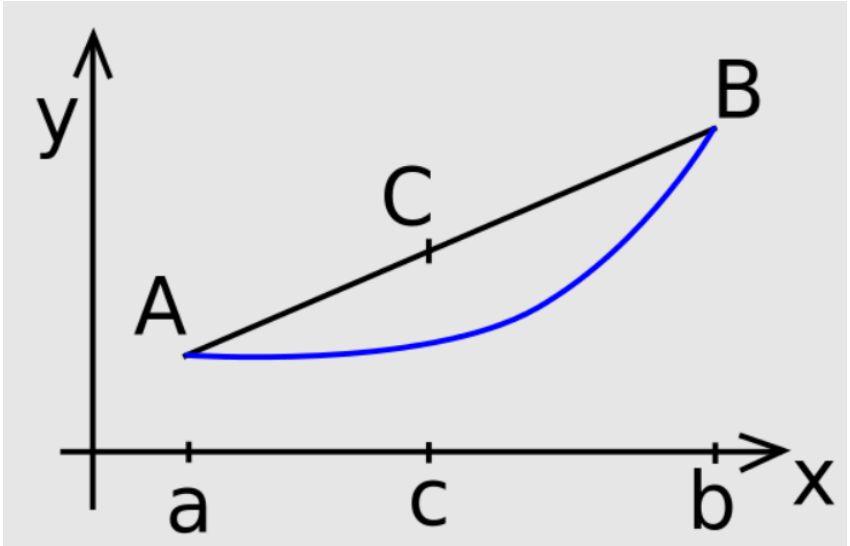
Dr. Peeta Basa Pati
Ms. Priyanka V
Department of Computer Science & Engineering,
Amrita School of Engineering, Bengaluru

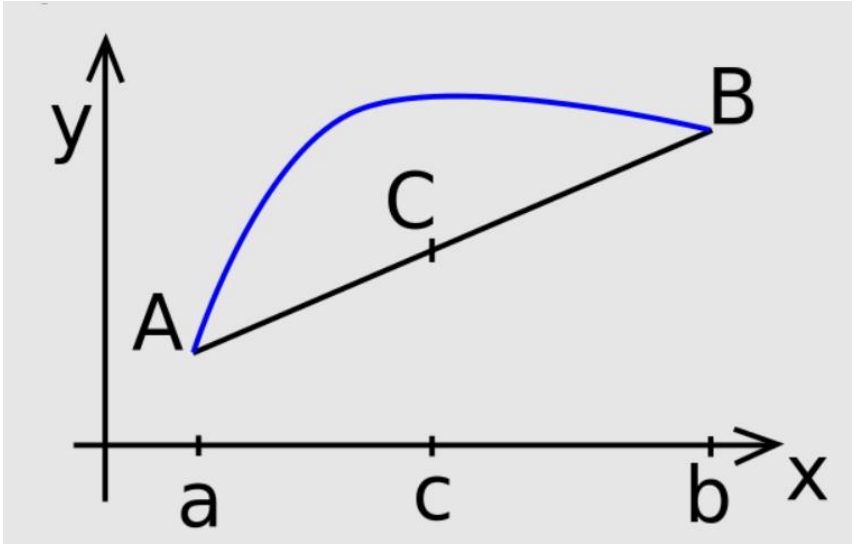
Topics

- Recap of Optimization
- Support Vectors

Functions, Derivatives & Convexity

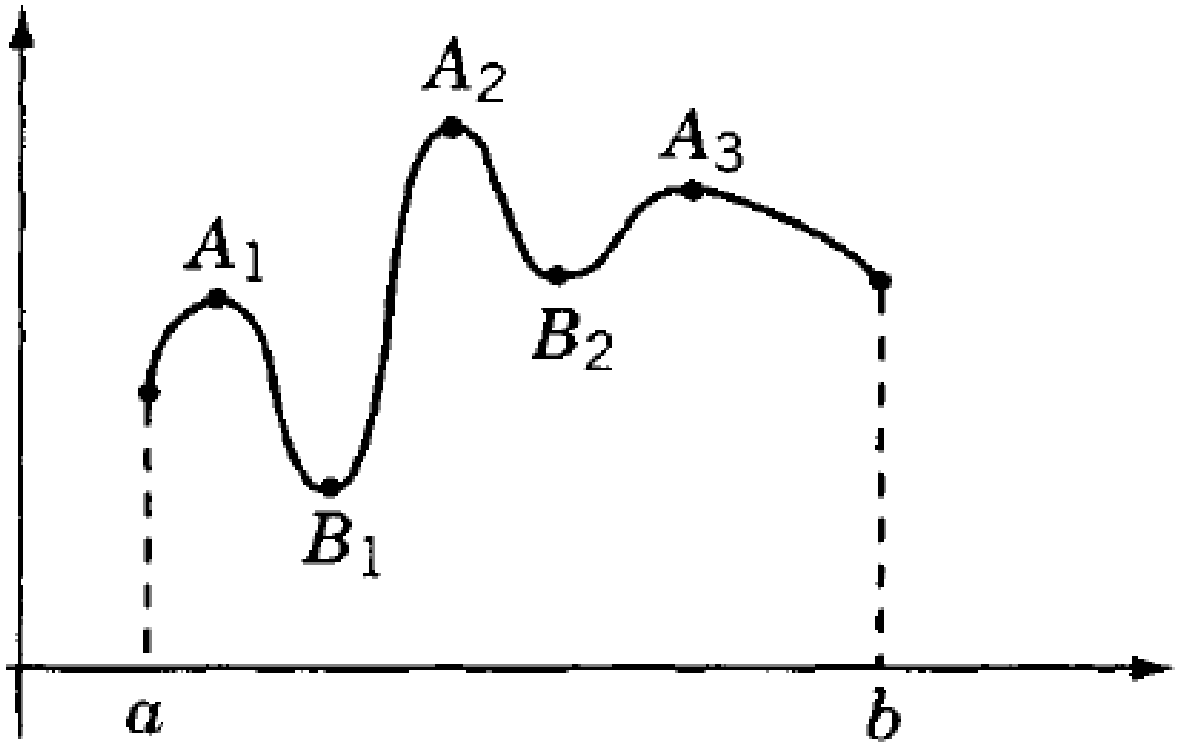






Source: Internet

Local & Global – Minimum, Maximum; Saddle point



- Brute force search
- Gradient Descent Search
- Genetic algorithm based approaches
- Evolutionary computation techniques
- Tabu Search
- Simulated Annealing
- Hill Climbing techniques
- Ant colony optimization
- Particle swarm optimization
- Random forest optimization

Source: Engineering Optimization, S S Rao

Derivation of Gradient Descent Algorithm

$$o(\vec{x}) = \vec{w} \cdot \vec{x}$$

$$E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

$$\vec{w} \leftarrow \vec{w} + \Delta \vec{w}$$

$$\Delta \vec{w} = -\eta \nabla E(\vec{w})$$

$$\nabla E(\vec{w}) \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \cdots, \frac{\partial E}{\partial w_n} \right]$$

$$w_i \leftarrow w_i + \Delta w_i$$

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

$$= \frac{1}{2} \sum_{d \in D} \frac{\partial}{\partial w_i} (t_d - o_d)^2$$

$$= \frac{1}{2} \sum_{d \in D} 2(t_d - o_d) \frac{\partial}{\partial w_i} (t_d - o_d)$$

$$= \sum_{d \in D} (t_d - o_d) \frac{\partial}{\partial w_i} (t_d - \vec{w} \cdot \vec{x}_d)$$

$$\frac{\partial E}{\partial w_i} = \sum_{d \in D} (t_d - o_d)(-x_{id})$$

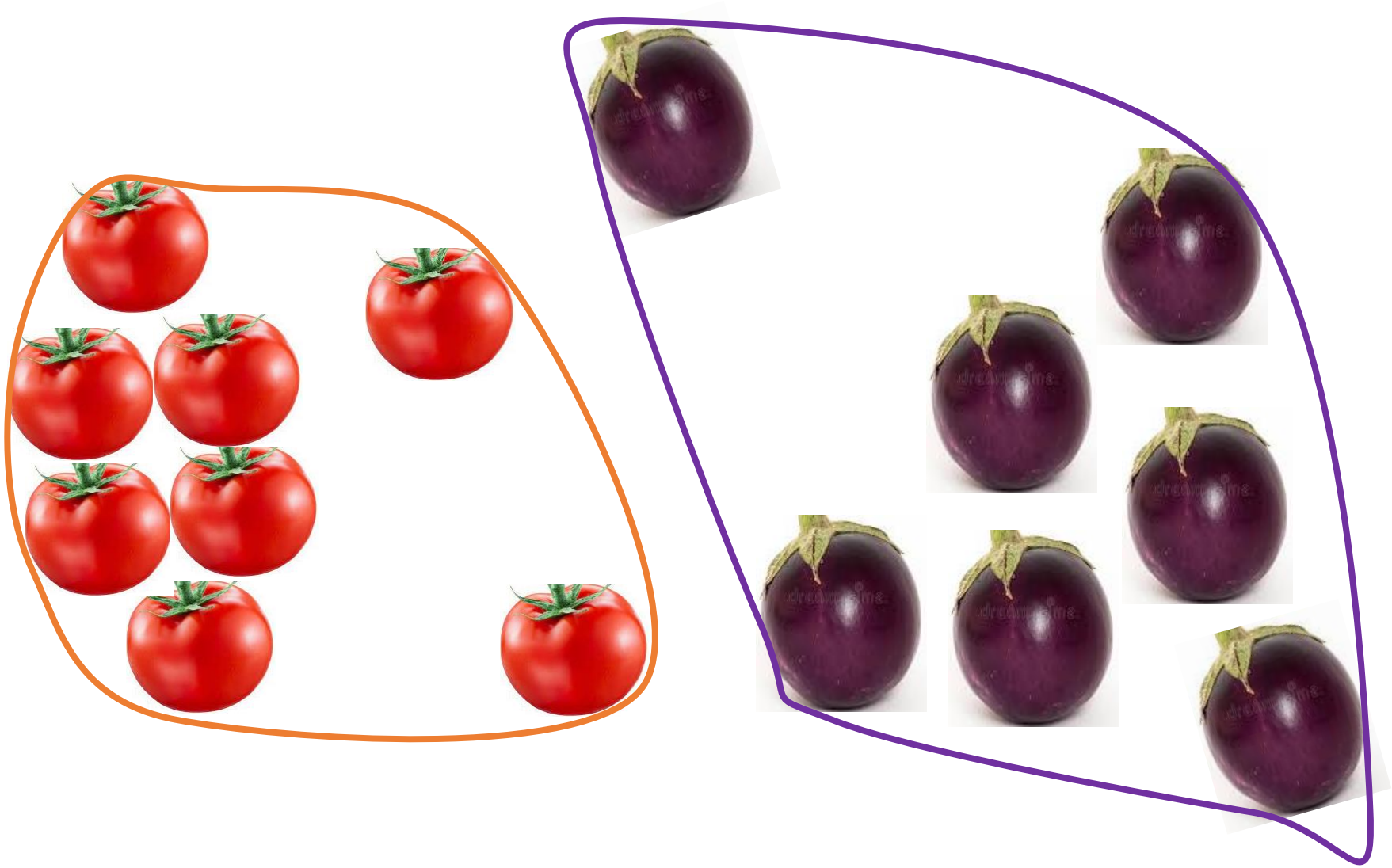
$$\Delta w_i = \eta \sum_{d \in D} (t_d - o_d)\ x_{id}$$

Notes on Gradient Descent Algorithm

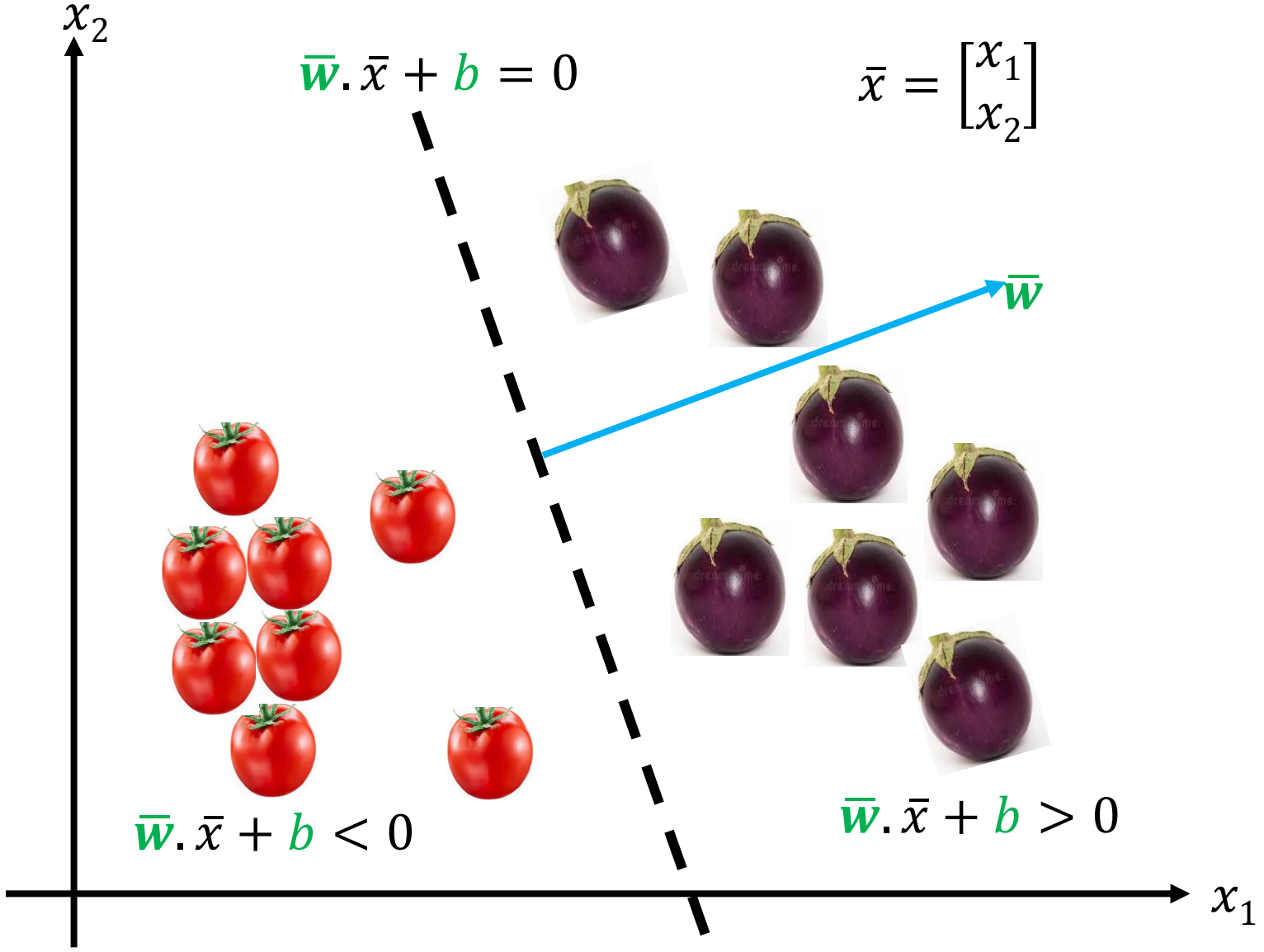
- Error is summed over all inputs and then the weights are updated
- Linear (pass through) activation function is used → f(x) = x
- Assumes a convex error space
- If there are local minima, the search may get stuck and never come-out
- Since error is summed over all inputs, the convergence may be slow
- Incremental or stochastic gradient descent is a variation of the algorithm
  - Weight update done with each input
  - Sometimes helps in overcoming the local minima
- The same principle can be applied with other activation functions as well. However, mathematical proof of convergence may be difficult.

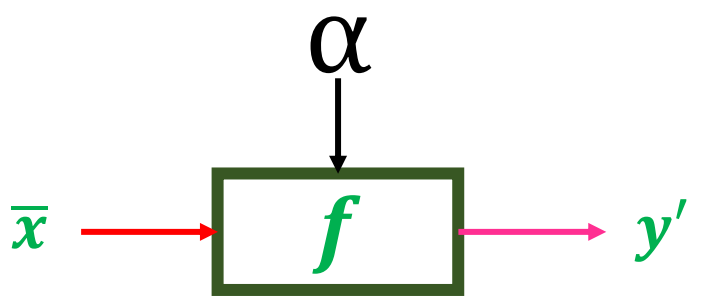
Support Vector Machines

Classes & Boundaries



Linear binary classifier



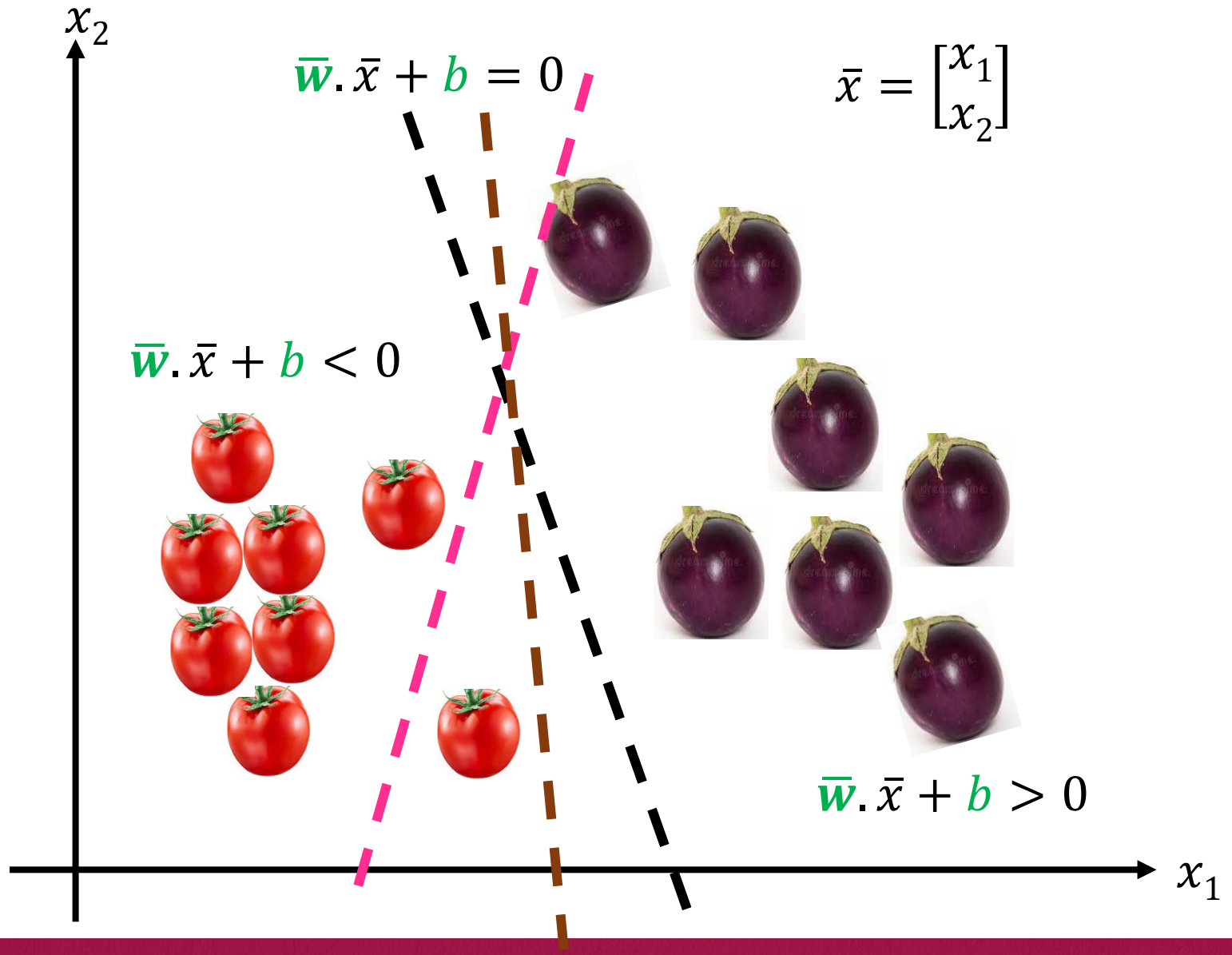


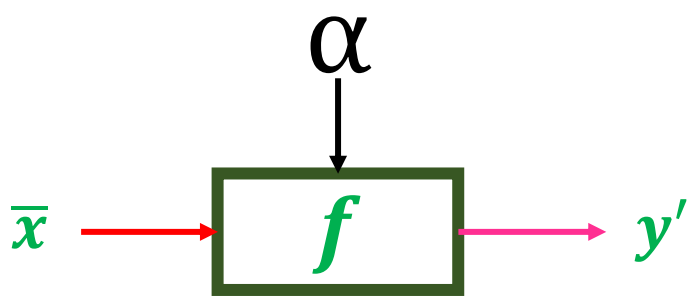
$$y' = f(\bar{x}, \bar{w}, b)$$
$$= sign\ (\bar{w}.\bar{x} + b)$$

Denotes +ve class

Denotes -ve class

Linear binary classifier



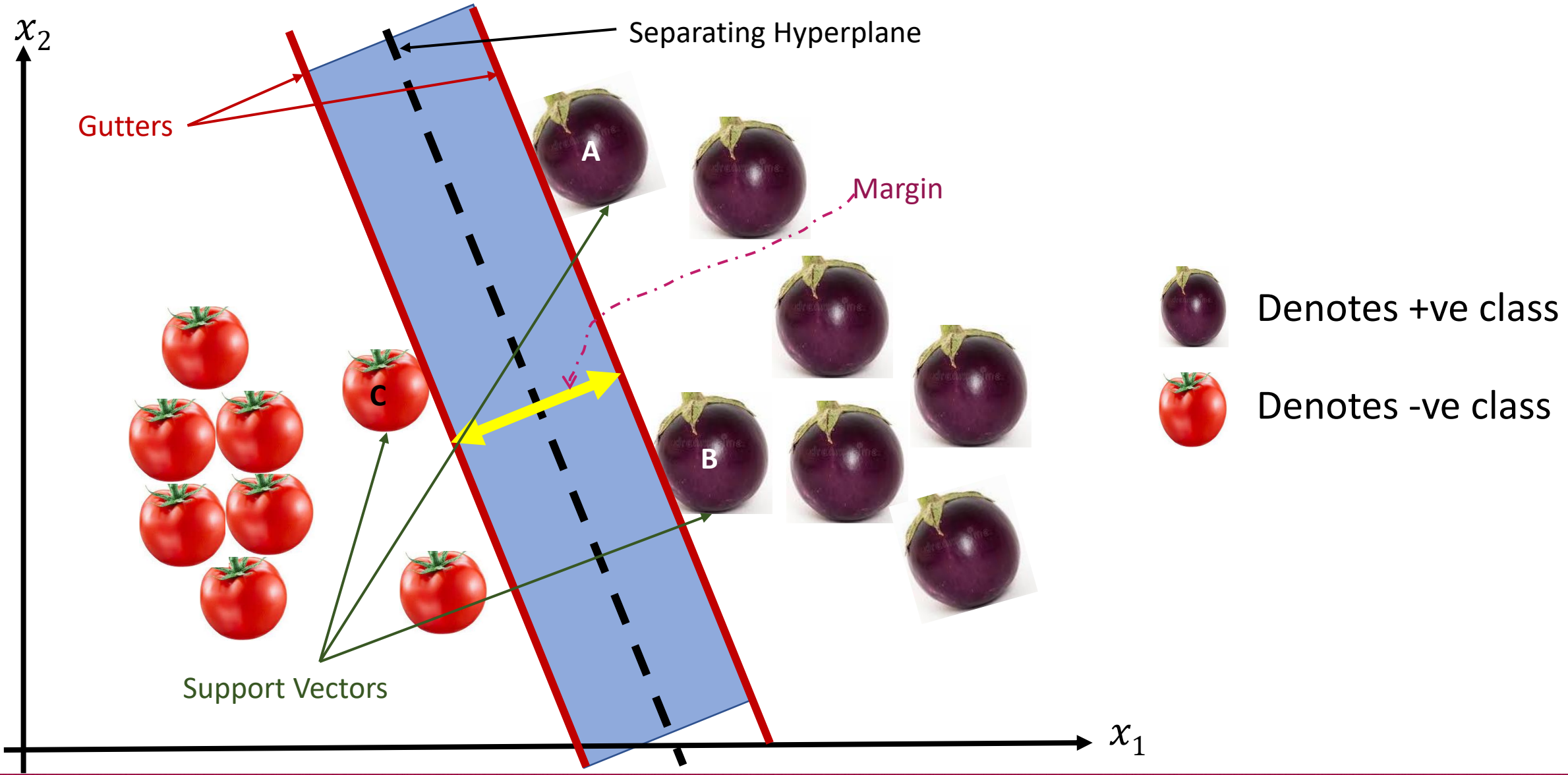


$$y' = f(\bar{x}, \bar{w}, b)$$
$$= sign\ (\bar{w}.\bar{x} + b)$$

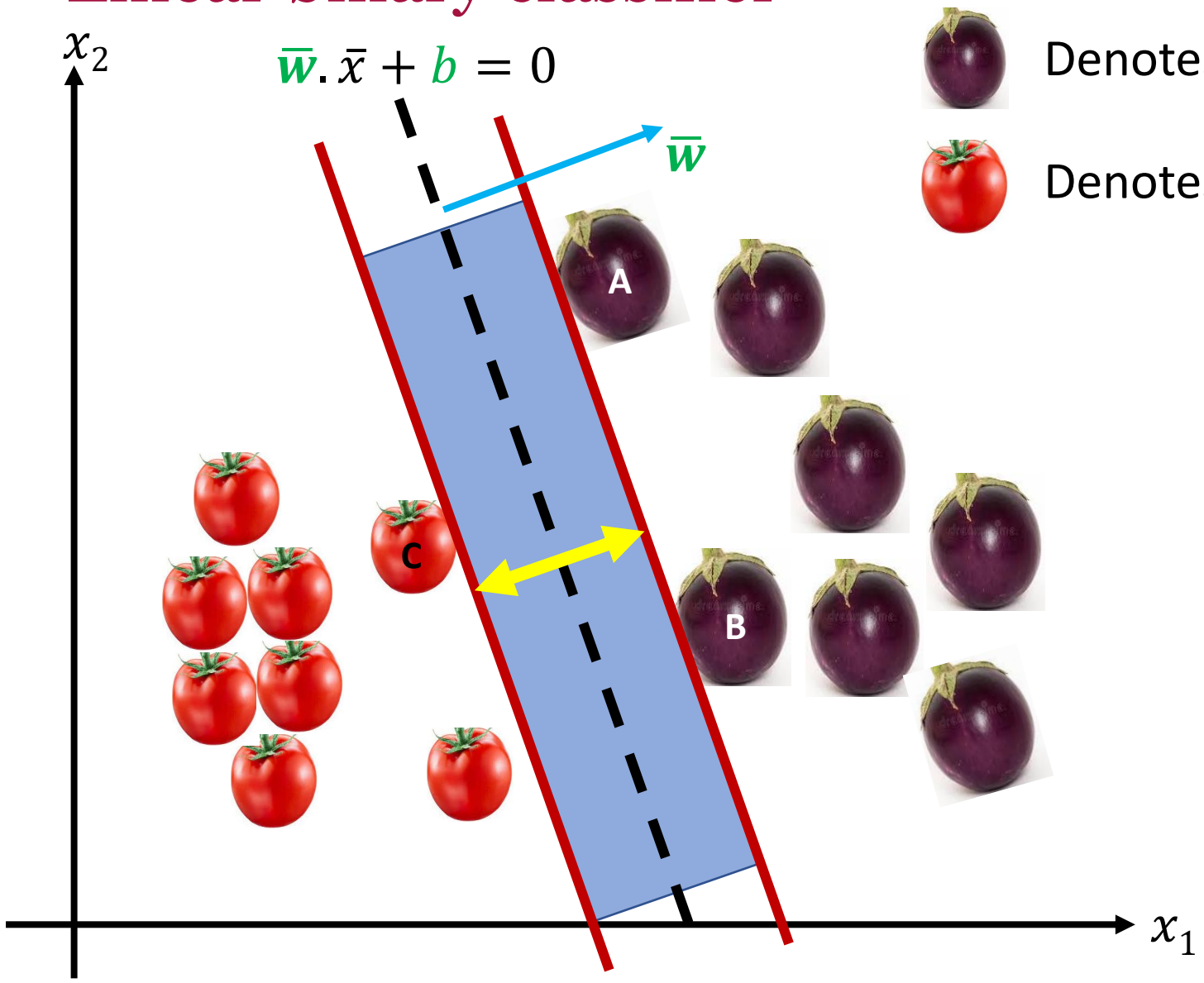
Denotes +ve class

Denotes -ve class

Support Vectors, Gutters, Separating Hyperplane & Margin

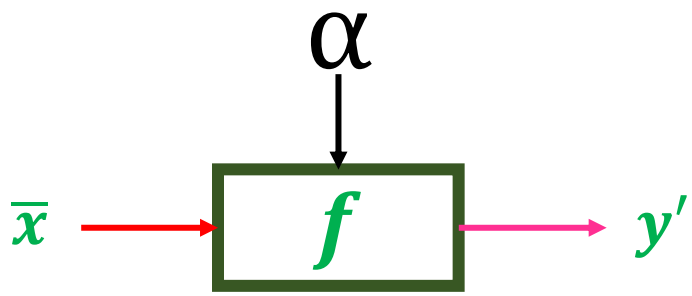


Linear binary classifier



Denotes +ve class

Denotes -ve class



$$y' = f(\bar{x}, \bar{w}, b)$$
$$= sign\ (\bar{w}.\bar{x} + b)$$

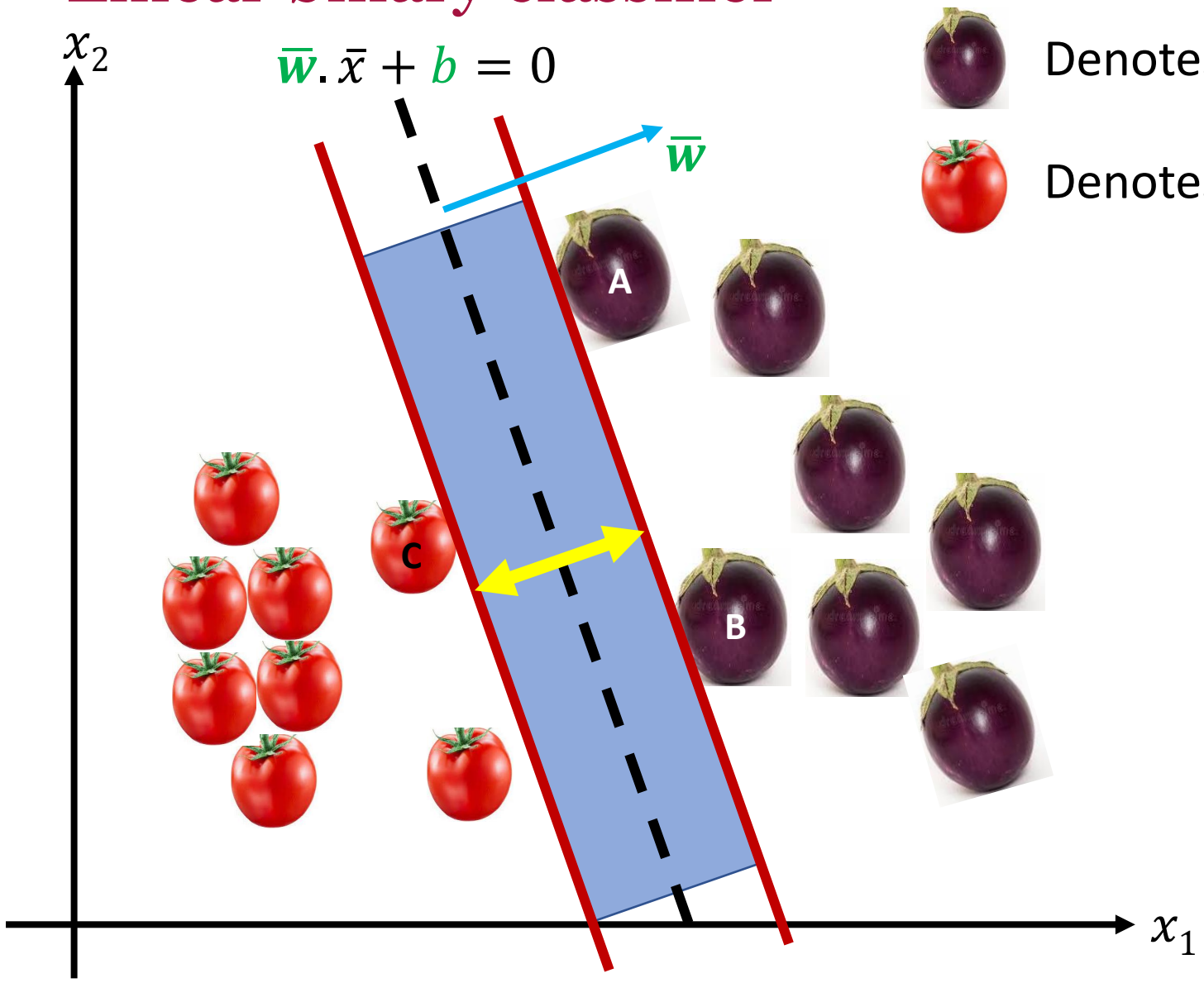
$$\bar{w}.\bar{x}_A + b = +1$$
$$\bar{w}.\bar{x}_B + b = +1$$
$$\bar{w}.\bar{x}_C + b = -1$$

Generalization

$$\bar{w}.\bar{x}_{+ve} + b \geq +1$$
$$\bar{w}.\bar{x}_{-ve} + b \leq -1$$

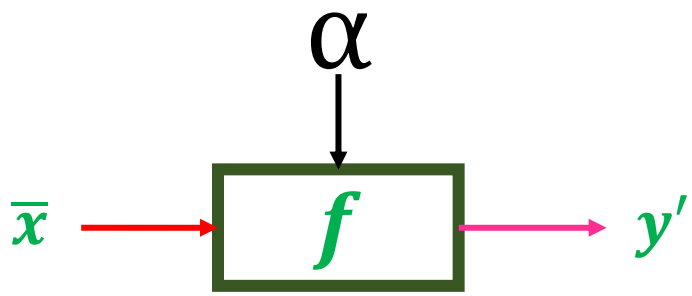
1

Linear binary classifier



Denotes +ve class

Denotes -ve class



$$y' = f(\bar{x}, \bar{w}, b)$$
$$= sign\ (\bar{w}.\bar{x} + b)$$

$$\bar{w}.\bar{x}_A + b = +1$$

$$\bar{w}.\bar{x}_B + b = +1$$

$$\bar{w}.\bar{x}_C + b = -1$$

$$M.||\bar{w}|| = (\bar{w}.\bar{x}A + b) - (\bar{w}.\bar{x}_C + b)$$

$$M = \frac{2}{||\bar{w}||} \quad (2)$$

Optimization for SVM

1

$$\bar{w}.\bar{x}_{+ve} + b \geq +1$$

$$\bar{w}.\bar{x}_{-ve} + b \leq -1$$

Multiply each equation with Y_i

Y_i = +1 for all +ve class vectors

Y_i = -1 for all –ve class vectors

$$(\bar{w}.\bar{x}_i + b)y_i \geq +1$$

2

$$M.||\bar{w}|| = (\bar{w}.\bar{x}A + b) - (\bar{w}.\bar{x}_C + b)$$

$$M = \frac{2}{||\bar{w}||}$$

We want to maximize the margin M.

➔ Minimize $||\bar{w}||$

➔ Minimize $(\bar{w}^T.\bar{w})$

Formulate the optimization problem & constraints

$$\phi(\bar{w}) = ½ \ (\bar{w}^T.\bar{w})$$

subject to $(\bar{w}.\bar{x}_i + b)y_i \geq +1$ for all i

$$y' = f(\bar{x}) = \sum_i \propto_i y_i \bar{x}_i^T.\bar{x} + b)$$

Optimization for SVM

Formulate the optimization problem & constraints

$\phi(\bar{w}) = ½ \ (\bar{w}^T . \bar{w})$

subject to $(\bar{w}.\bar{x}_i + b)y_i \geq +1$

for all i

$$\sum_i \propto i \ yi = 0$$

y_i are all scalars; hence, $\propto_i$ are also scalars.

$$y' = f(\bar{x}) = \sum_i \propto_i y_i \bar{x}_i^T . \bar{x} + b)$$

SVM

$$y' = f(\bar{x}) = \sum_i \propto_i y_i \bar{x}_i^T . \bar{x} + b)$$

Kernel function is some function which maintains the sanctity of dot product in some expanded space

Kernel function:

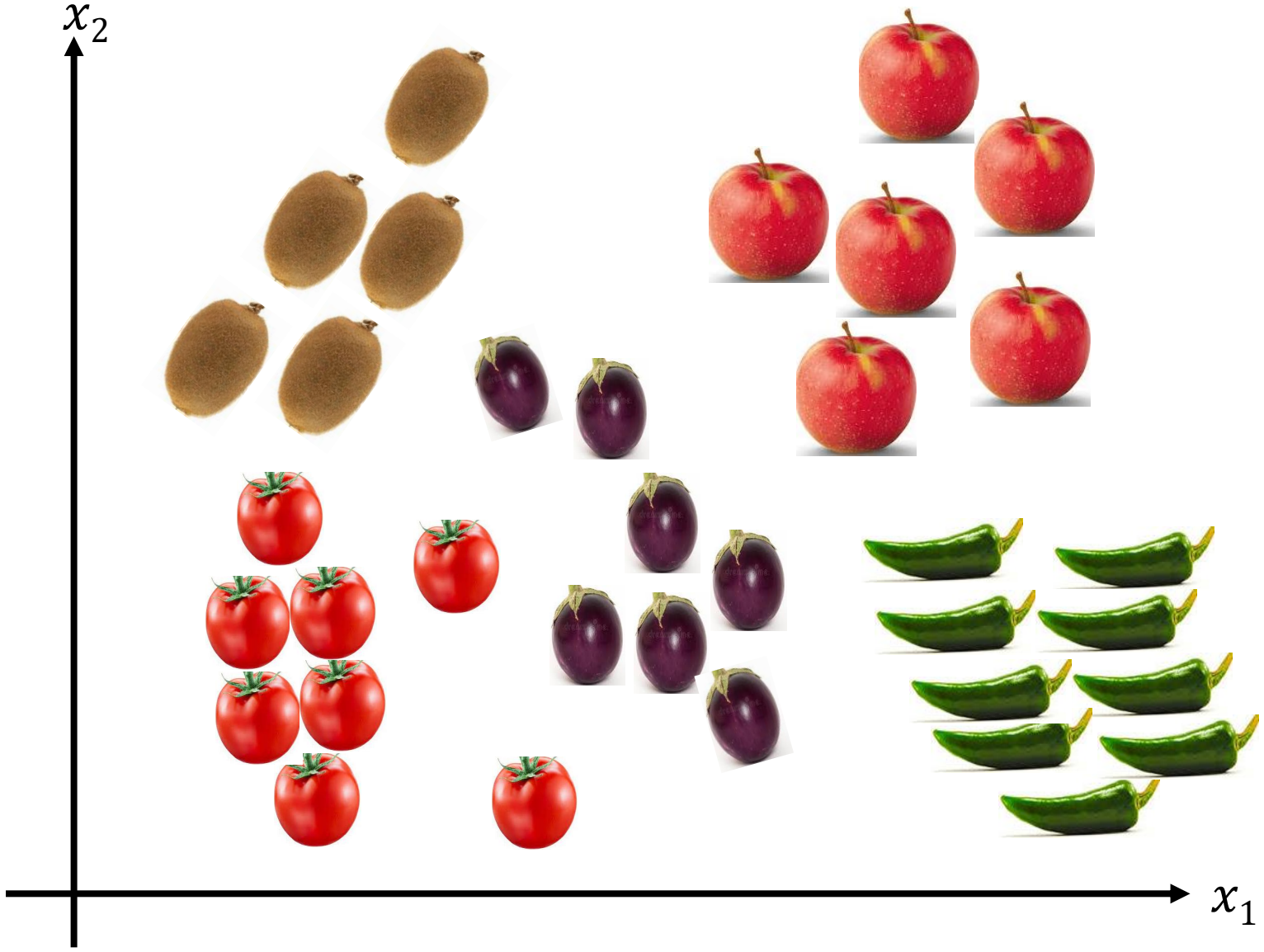
$$K(\bar{x}_i, \bar{x}_j) = \bar{x}_i^T . \bar{x}_j$$

Kernel function generalized as:

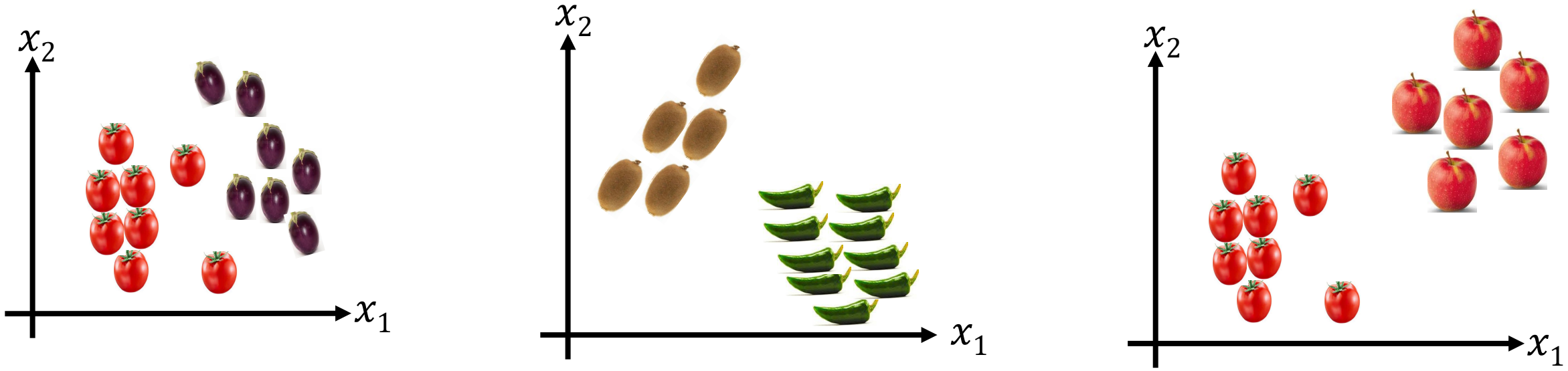
$$K(\bar{x}_i, \bar{x}_j) = \varphi(\bar{x}_i)^T . \varphi(\bar{x}_j)$$

Kernel function by mapping the vectors to some expanded space, introduces linearity which is absent in the original space.

Multiclass Scenario

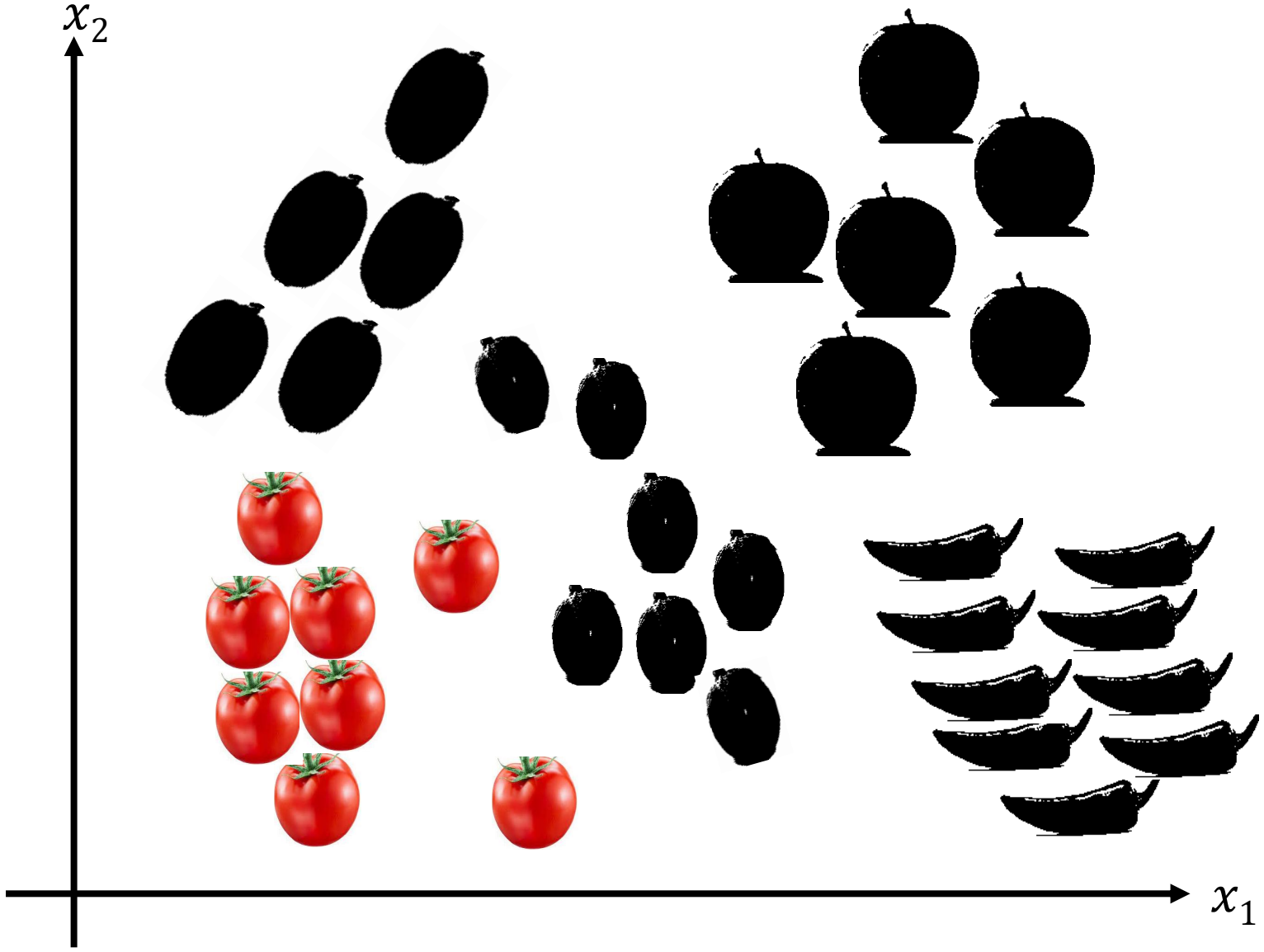


One against another approach



- Consider each class separately against all other classes combined
  - ✓ Tomato against brinjal
  - ✓ Kiwi against green chilies
  - ✓ Apples against tomatoes
- N * (N-1) / 2 SVM models generated (10 models in this example)
- Test pattern is run through each models
- Class value assigned based on absolute maximum score & sign (+/-ve) from all models

One vs all others approach



- Consider each class separately against all other classes combined
  - ✓ Tomato against all others
  - ✓ Kiwi against all others
  - ✓ Apples against all others
- SVM models generated as many classes are available (5 models in this example)
- Test pattern is run through each models
  - ✓ Score of being an apple
  - ✓ Score of being a tomato
- Class value assigned based on maximum membership score

Python Code

```
>>> from sklearn import svm
>>> X = [[0, 0], [1, 1]]
>>> y = [0, 1]
>>> clf = svm.SVC()
>>> clf.fit(X, y)
```

```
>>> clf.predict([[2., 2.]])
array([1])
```

Thank you !!!!!

Machine Learning (19CSE305)

K-Nearest neighbor

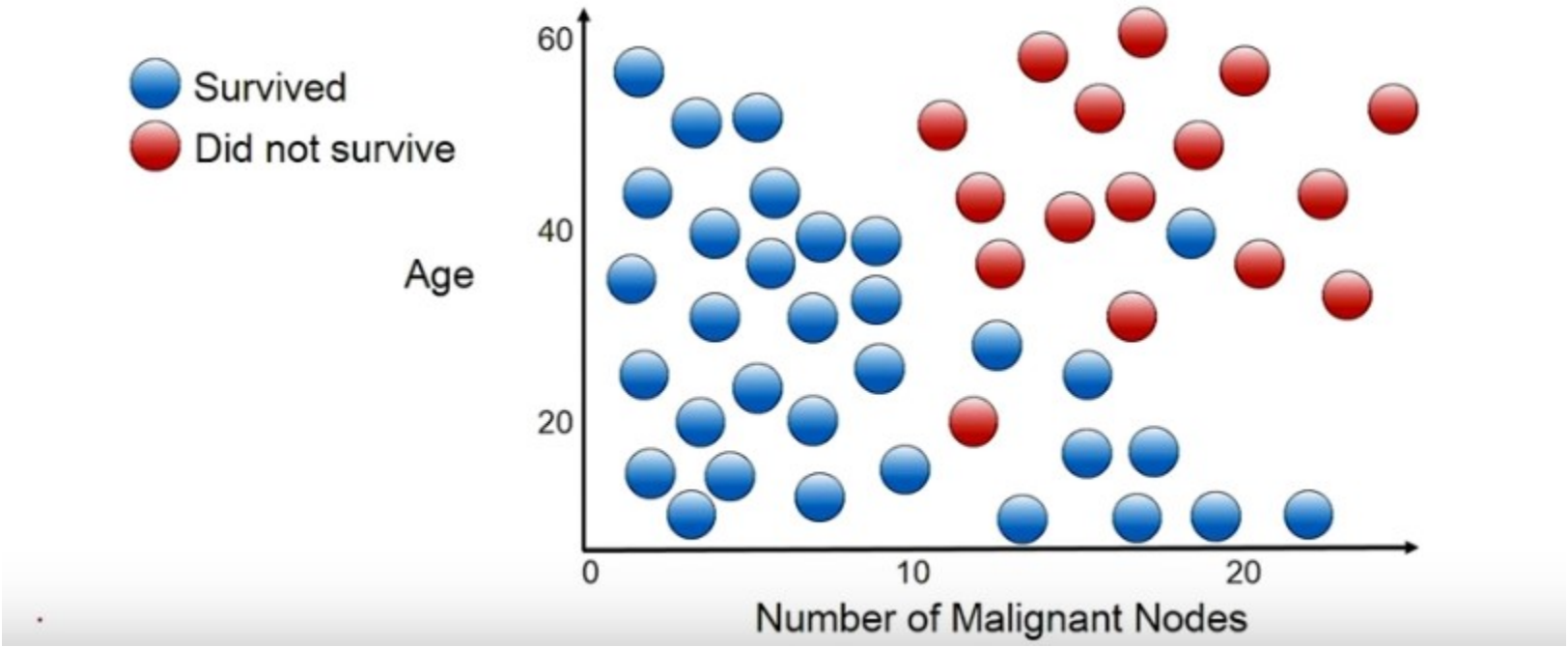
Dr. Peeta Basa Pati
Ms. Priyanka V
Department of Computer Science & Engineering,
Amrita School of Engineering, Bengaluru

Topics

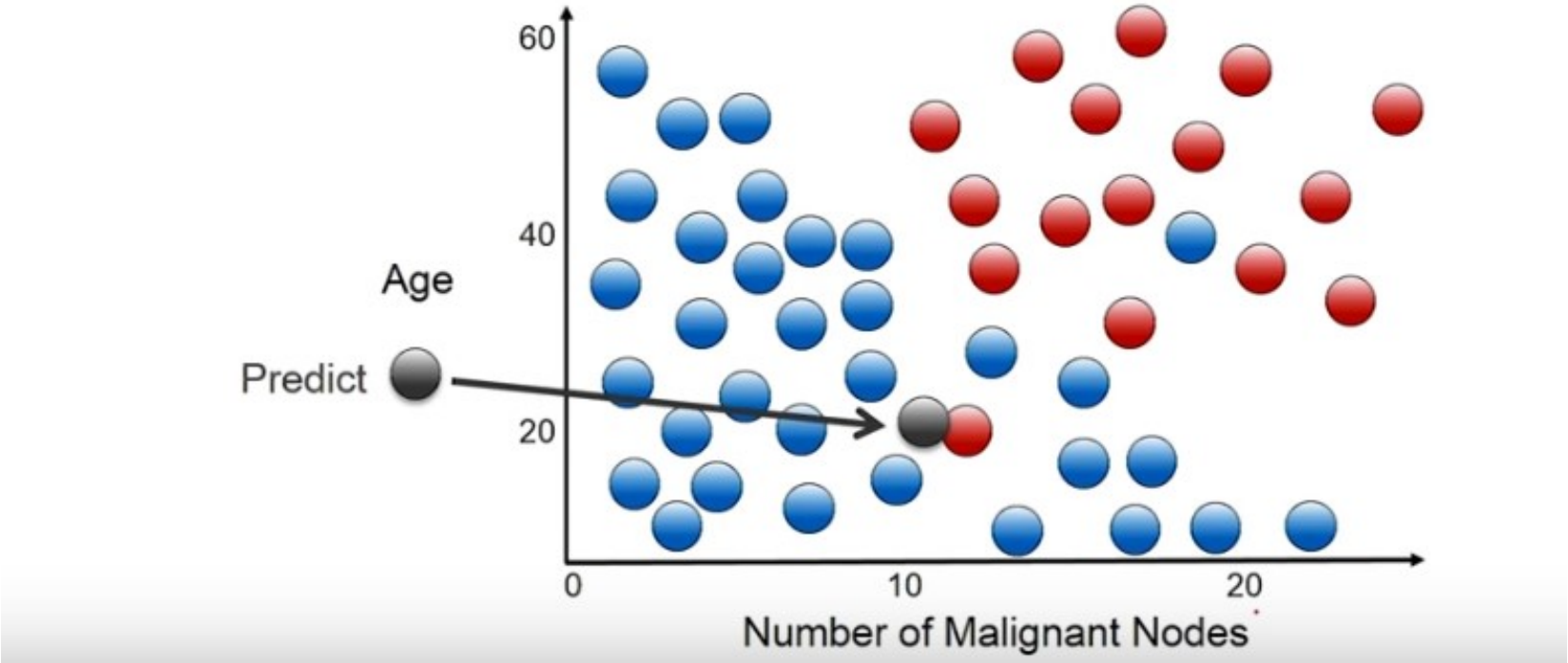
- K-nearest neighbour classifier

- Distance measure
- Voronoi diagram

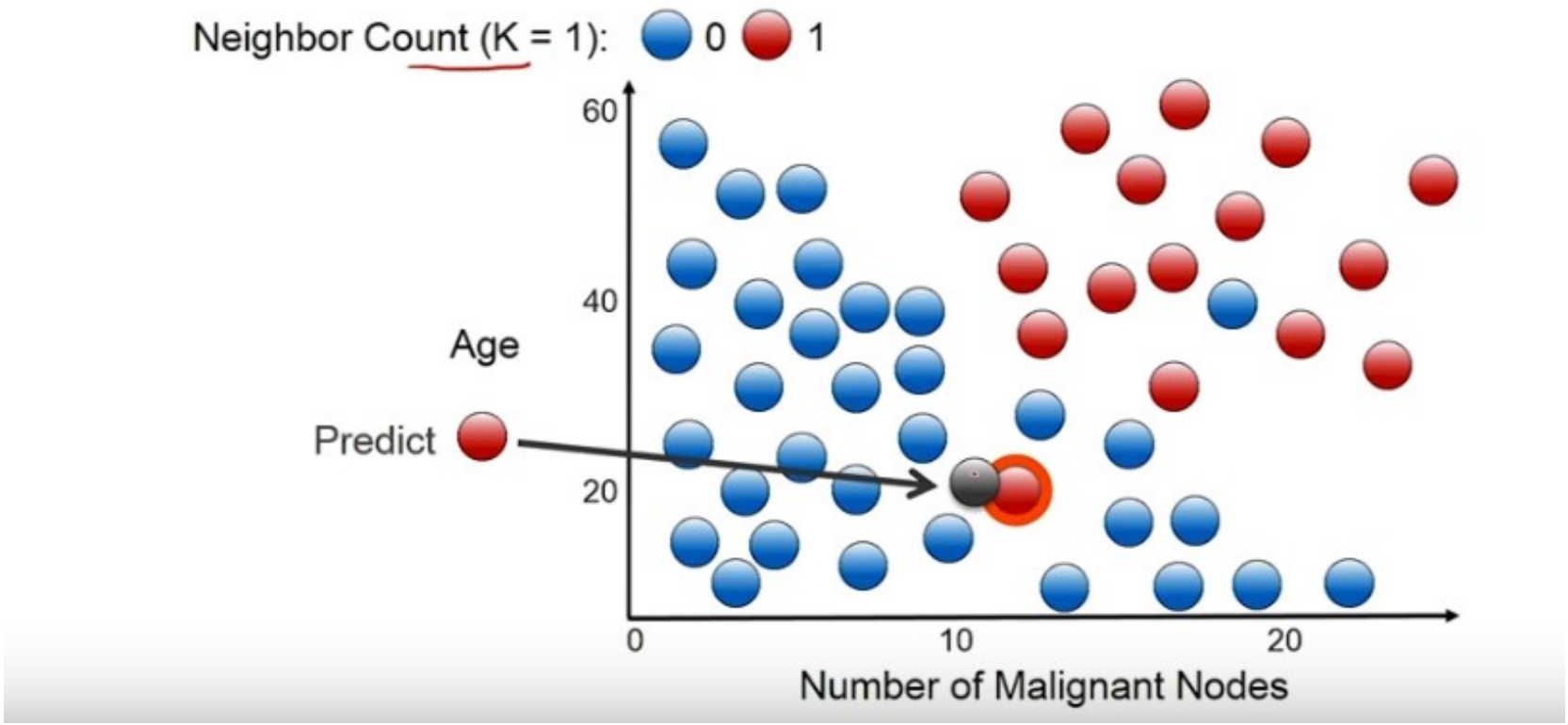
K-Nearest Neighbour- Classification



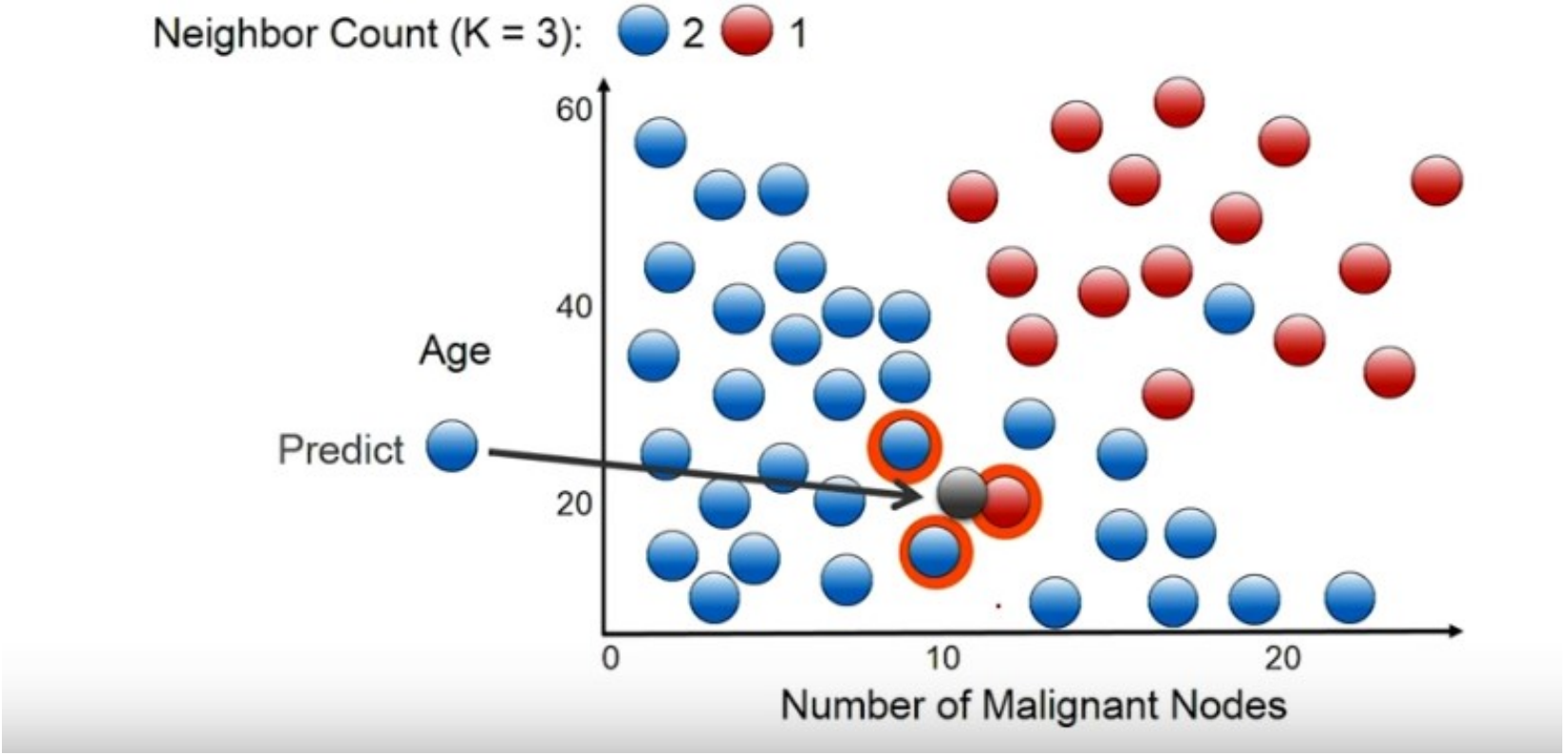
K-Nearest Neighbour- Classification



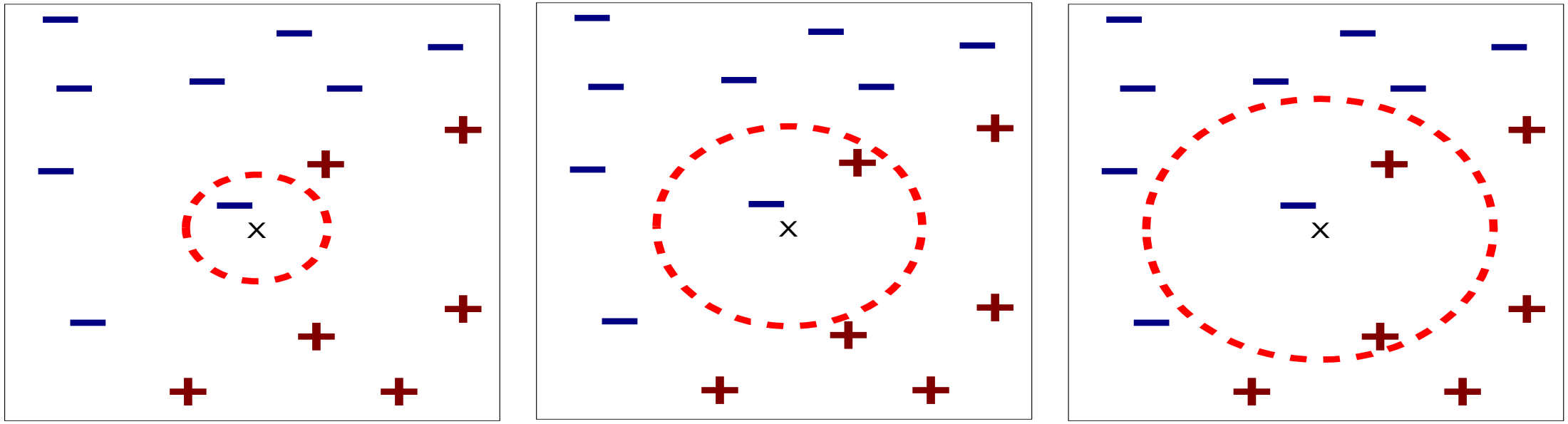
.. Classification



...Classification



Definition of Nearest Neighbor



(a) 1-nearest neighbor (b) 2-nearest neighbor (c) 3-nearest neighbor

K-nearest neighbors of a record x are data points that have the k smallest distance to x

K-nearest neighbour- algorithm

- Training phase : Save the examples
- Prediction phase: Get the test instance

Find the k- training examples

$\{(x_1,y_1), (x_2,y_2), (x_3,y_3)..... (x_k,y_k)\}$ that is closest to x_t

Classification : predict the majority class from $\{y_1, y_2, y_3.... y_k\}$

Regression : predict the average of $\{y_1, y_2, y_3.... y_k\}$

Calculating Distance

- Euclidean
- Let A and B are represented by feature vectors $A = (x_1, x_2, ..., x_n)$ and $B = (y_1, y_2, ..., y_n)$, where n is the dimensionality of the feature space.

$$dist\,(A, B) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

Example

We have data from the questionnaires survey (to ask people opinion) and objective testing with two attributes (acid durability and strength) to classify whether a special paper tissue is good or not. Here is four training samples

X1 = Acid Durability (seconds)	X2 = Strength (kg/square meter)	Y = Classification
7	7	Bad
7	4	Bad
3	4	Good
1	4	Good

Now the factory produces a new paper tissue that pass laboratory test with X1 = 3 and X2 = 7. Without another expensive survey, can we guess what the classification of this new tissue is?

Assume k = 3

X1 = Acid Durability (seconds)	X2 = Strength (kg/square meter)	**Square Distance to query instance (3, 7)**
7	7	$(7-3)^2+(7-7)^2=16$
7	4	$(7-3)^2+(4-7)^2=25$
3	4	$(3-3)^2+(4-7)^2=9$
1	4	$(1-3)^2+(4-7)^2=13$

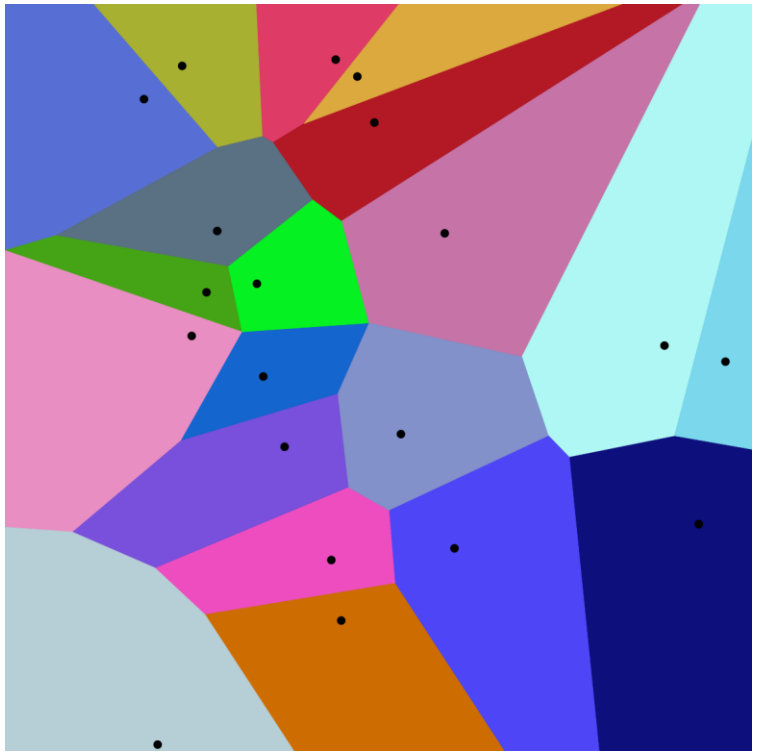
Assume k = 3

X1 = Acid Durability (seconds)	X2 = Strength (kg/square meter)	Square Distance to query instance (3, 7)	**Rank minimum distance**	**Is it included in 3-Nearest neighbors?**
7	7	$(7-3)^2+(7-7)^2=16$	**3**	**Yes**
7	4	$(7-3)^2+(4-7)^2=25$	**4**	**No**
3	4	$(3-3)^2+(4-7)^2=9$	**1**	**Yes**
1	4	$(1-3)^2+(4-7)^2=13$	**2**	**Yes**

X1 = Acid Durability (seconds)	X2 = Strength (kg/square meter)	Square Distance to query instance (3, 7)	Rank minimum distance	Is it included in 3-Nearest neighbors?	**Y = Category of nearest Neighbor**
7	7	$(7-3)^2+(7-7)^2=16$	3	Yes	**Bad**
7	4	$(7-3)^2+(4-7)^2=25$	4	No	-
3	4	$(3-3)^2+(4-7)^2=9$	1	Yes	**Good**
1	4	$(1-3)^2+(4-7)^2=13$	2	Yes	**Good**

We have 2 good and 1 bad, since 2>1 then we conclude that a new paper tissue that pass laboratory test with X1 = 3 and X2 = 7 is included in **Good** category.

Voronoi diagram k=1



Lazy vs Eager Learner

- Eager learners
  - when given a set of training tuples, will construct a generalization (i.e., classification) model before receiving new (e.g., test) tuples to classify

- Lazy learners-
  - waits until the last minute before doing any model construction to classify a given test tuple
  - simply stores it (or does only a little minor processing) and waits until it is given a test tuple.
  - Also referred as instance based learner

kNN - advantage

it is relatively straightforward to update the model when new labeled instances become available—we simply add them to the training dataset.

Nearest Neighbor Classification…

- Scaling issues
  - Attributes may have to be scaled to prevent distance measures from being dominated by one of the attributes
  - Example:
    - height of a person may vary from 1.5m to 1.8m
    - weight of a person may vary from 50kg to 110kg
    - income of a person may vary from ₹10K to ₹1crore
  - normalisation

If attributes are non numeric?

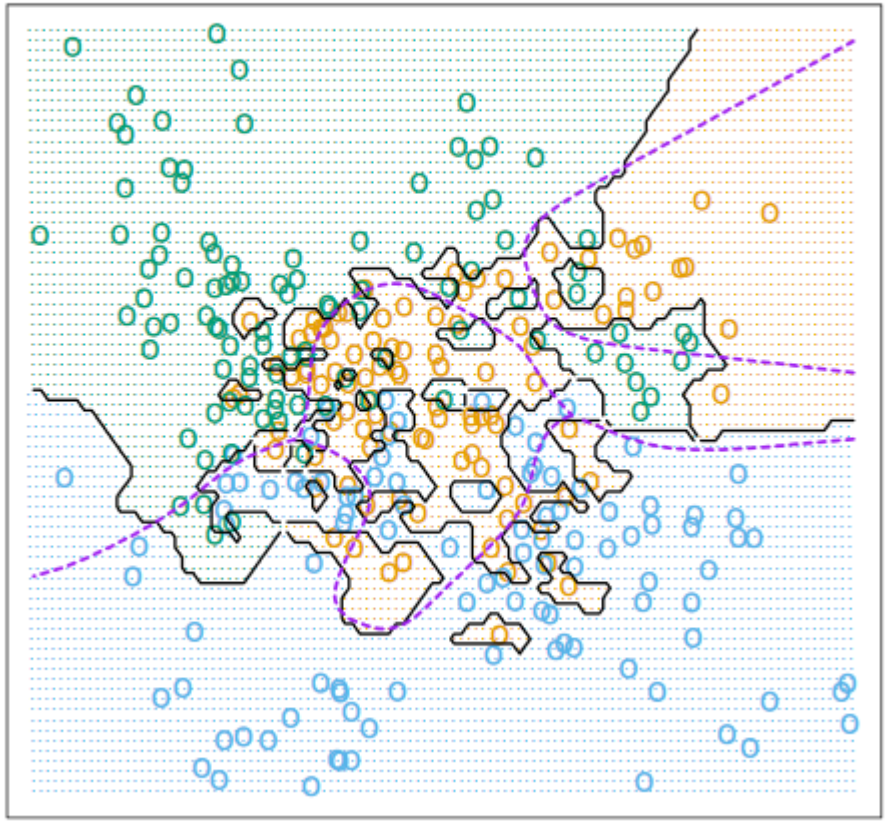
" color?"

- compare the corresponding value of the attribute in tuple X1 with that in tuple X2 .
- If the two are identical (e.g., tuples X1 and X2 both have the color blue), then the difference between the two is taken as 0.
- If the two are different (e.g., tuple X 1 is blue but tuple X 2
- is red), then the difference is considered to be 1.
- Other methods may incorporate more sophisticated schemes for differential grading (e.g., where a larger difference score is
- assigned, say, for blue and white than for blue and black).

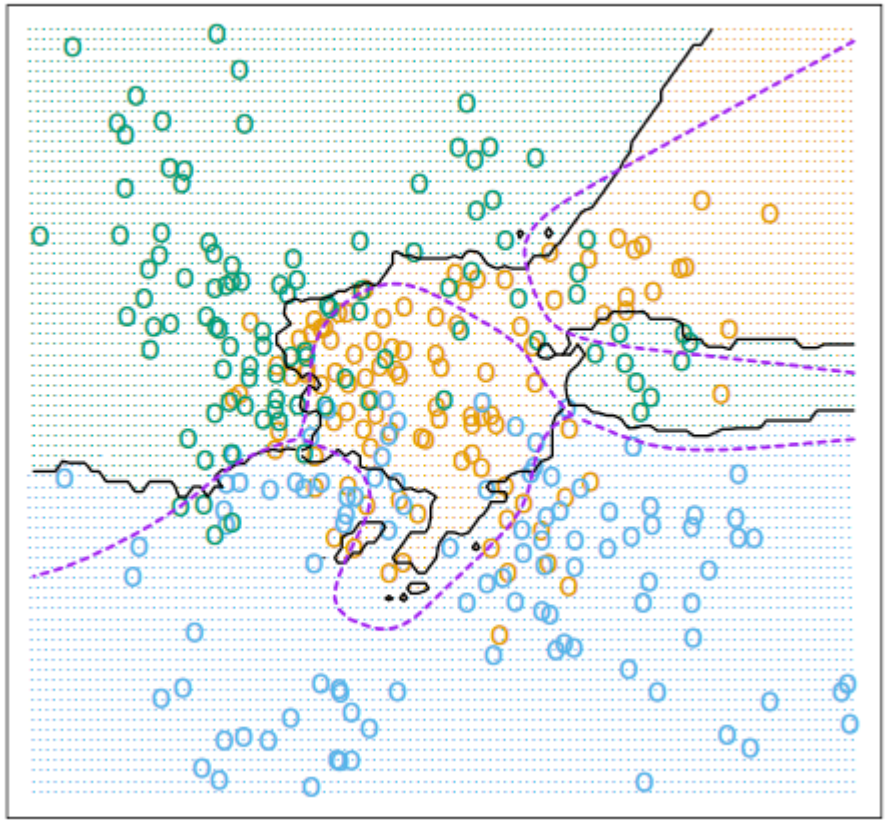
How to determine a good value for k?

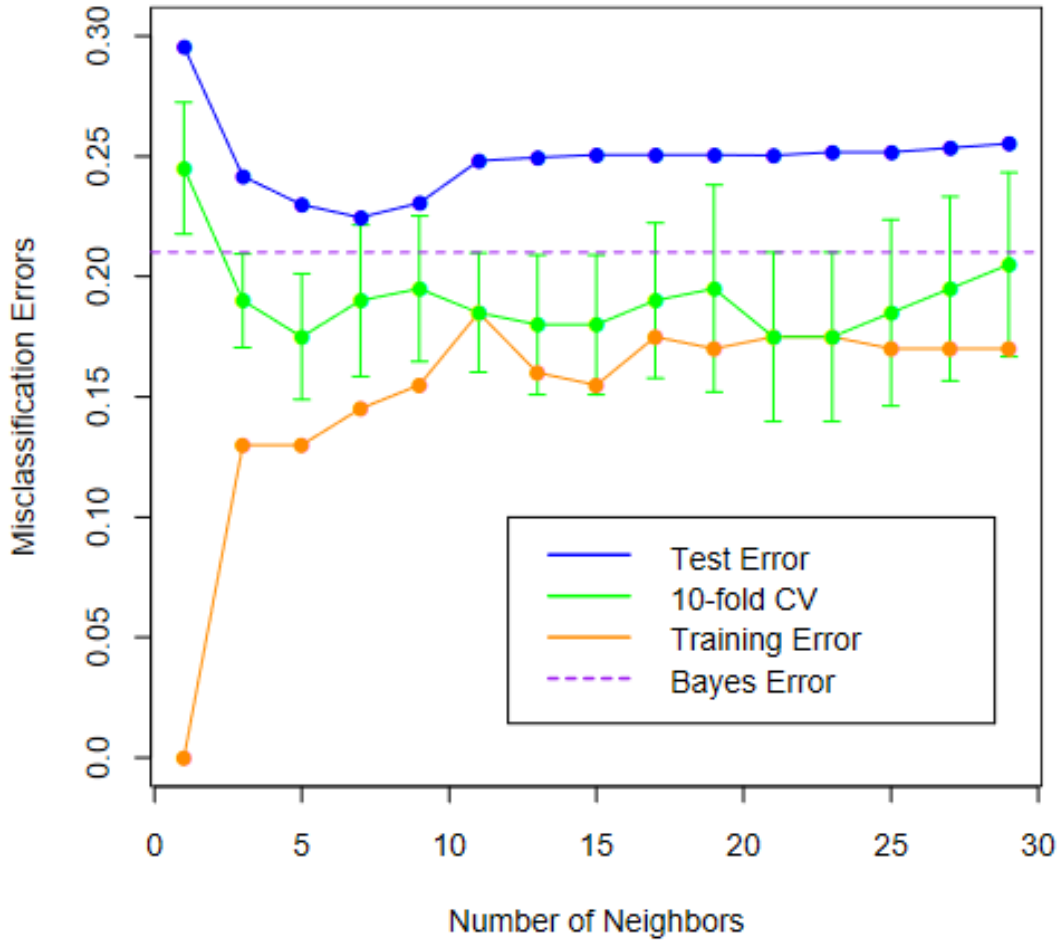
- Starting with k = 1, we use a test set to estimate the error rate of the classifier.
- This process can be repeated each time by incrementing k to allow for one more neighbor.
- The k value that gives the minimum error rate may be selected. the larger the number of training tuples, the larger the value of k will be (so that classification and numeric prediction decisions can be based on a larger portion of the stored tuples).

1-Nearest Neighbor

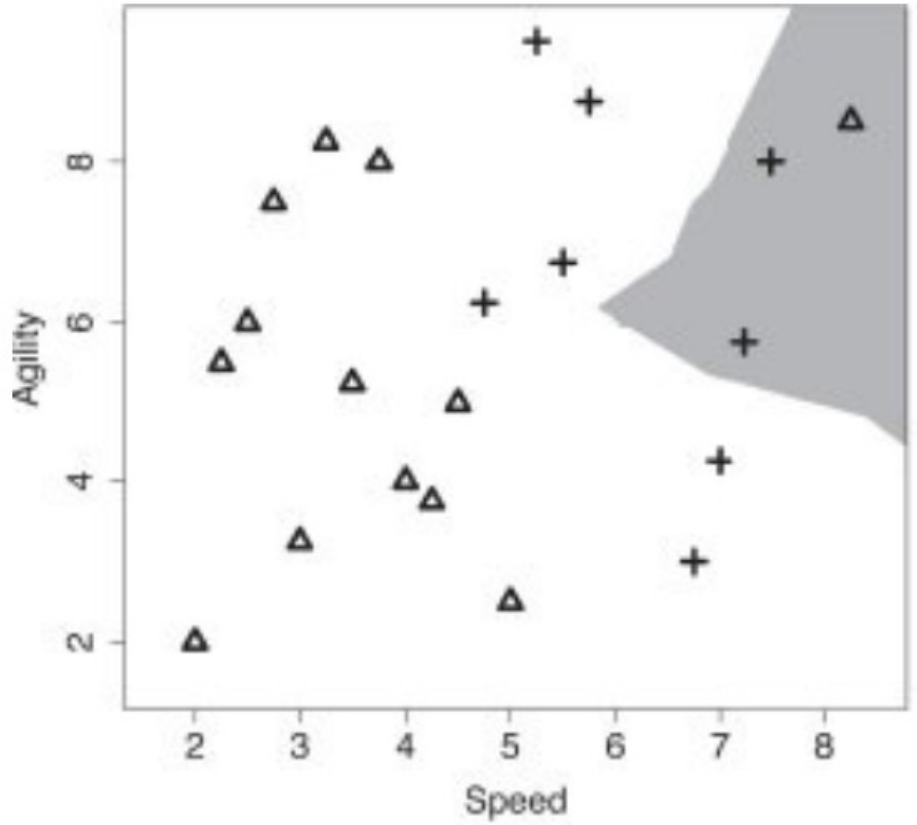


15-Nearest Neighbors





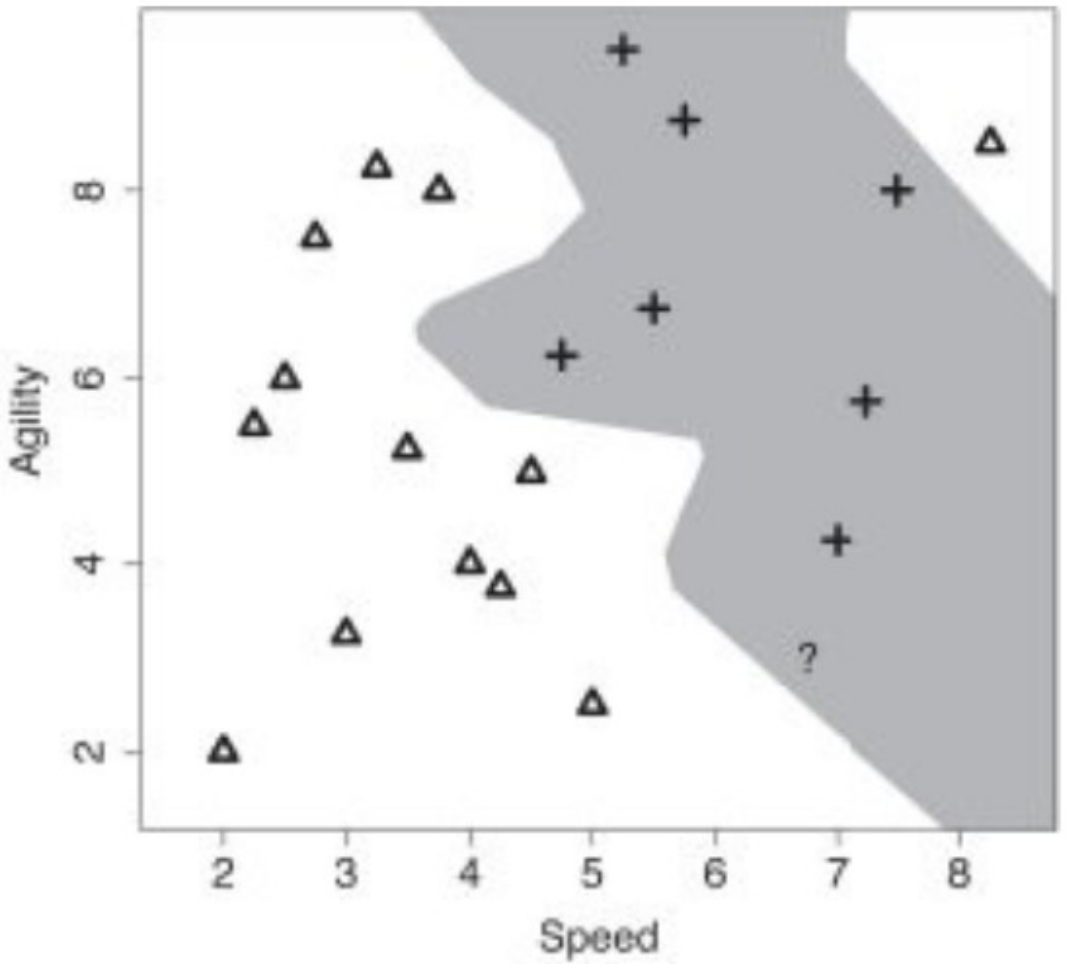
Imbalanced data

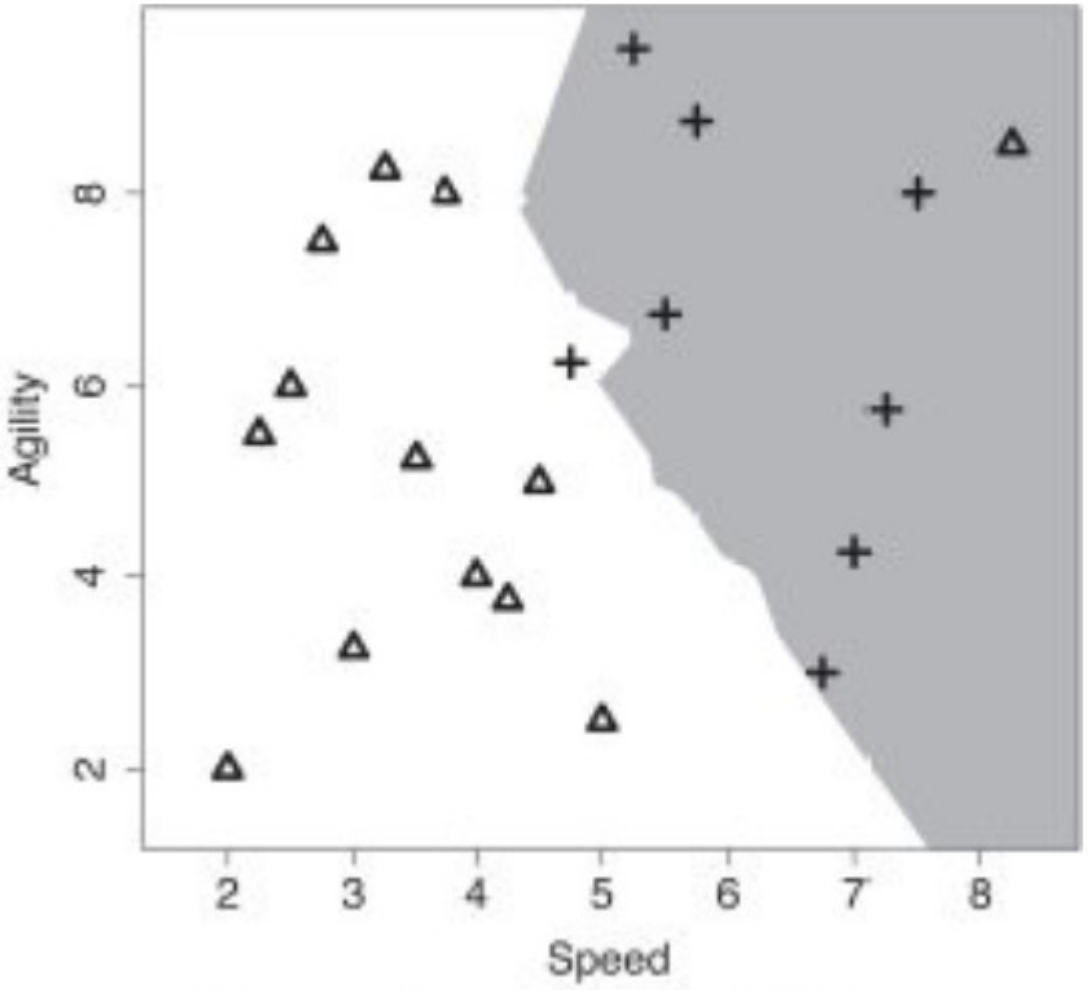


(a) Decision boundary ($k = 15$)

- The risks associated with setting k to a high value are particularly acute when we are dealing with an imbalanced dataset.
- as k increases, the majority target level begins to dominate the feature space

Noise in data





(b) Decision boundary ($k = 5$)

Distance-weighted nearest neighbor algorithm

- When a distance weighted k nearest neighbor approach is used, the contribution of each neighbor to the prediction is a function of the inverse distance between the neighbor and the query x_q
  - Give greater weight to closer neighbor

$$w \equiv \frac{1}{d(x_q, x_i)^2}$$

.

Issues with Distance-weighted knn

- if the dataset is very imbalanced, then even with a weighting applied to the contribution of the training instances, the majority target level may dominate.
- when the dataset is very large, which means that computing the reciprocal of squared distance between the query and all the training instances can become too computationally expensive to be feasible

Improvements

- Weighted Euclidean distance

$$D(c1,c2) = \sqrt{\sum_{i=1}^{N} w_i \cdot \left(attr_i(c1) - attr_i(c2)\right)^2}$$

- large weights => attribute is more important
- small weights => attribute is less important
- zero weights => attribute doesn't matter

Efficient Memory Search

- if we are working with a large dataset, the time cost in computing the distances between a query and all the training instances and retrieving the k nearest neighbors may be prohibitive.
- use k-d tree,(k-dimensional tree),.
- A k-d tree is a balanced binary tree in which each of the nodes in the tree (both interior and leaf nodes) index one of the instances in a training dataset.
- The tree is constructed so that nodes that are nearby in the tree index training instances that are nearby in the feature space

Refer for more details

- **FUNDAMENTALS OF MACHINE LEARNING FOR PREDICTIVE DATA ANALYTICS-** Algorithms, Worked Examples, and Case Studies ,John D. Kelleher, Brian Mac Namee, Aoife D'Arcy

Another approach

- speed up classification time include the use of partial distance calculations and editing the stored tuples.
- Compute the distance based on a subset of the n attributes.
- If this distance exceeds a threshold, then further computation for the given stored tuple is halted, and the process moves on to the next stored tuple.
- The editing method removes training tuples that prove useless.
- This method is also referred to as pruning or condensing because it reduces the total number of tuples stored.

Thank you !!!!!

Machine Learning (19CSE305)

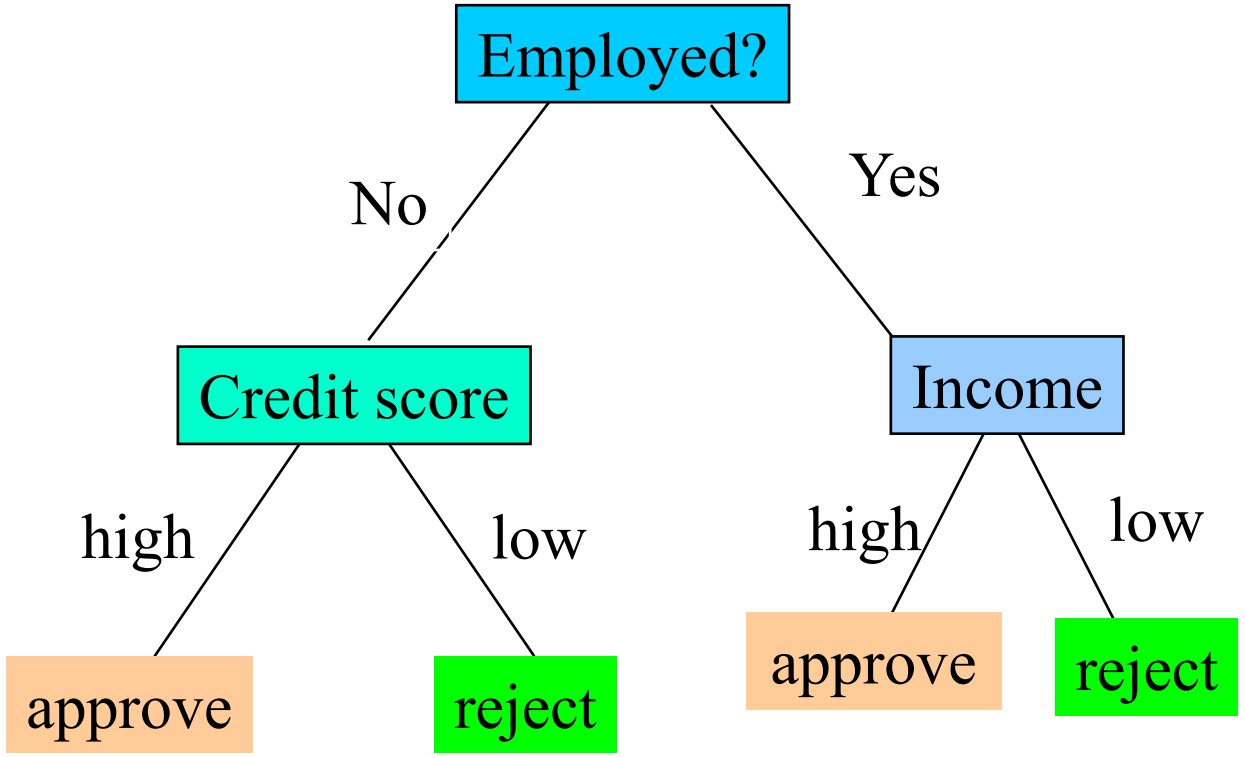
Decision Tree

Dr. Peeta Basa Pati
Ms. Priyanka V
Department of Computer Science & Engineering,
Amrita School of Engineering, Bengaluru

What are Decision trees?

- A decision tree is a tree in which each branch node represents a choice between a number of alternatives, and each leaf node represents a decision.
- A type of supervised learning algorithm.

Decision Tree An Example



Whether to approve/reject a loan application?

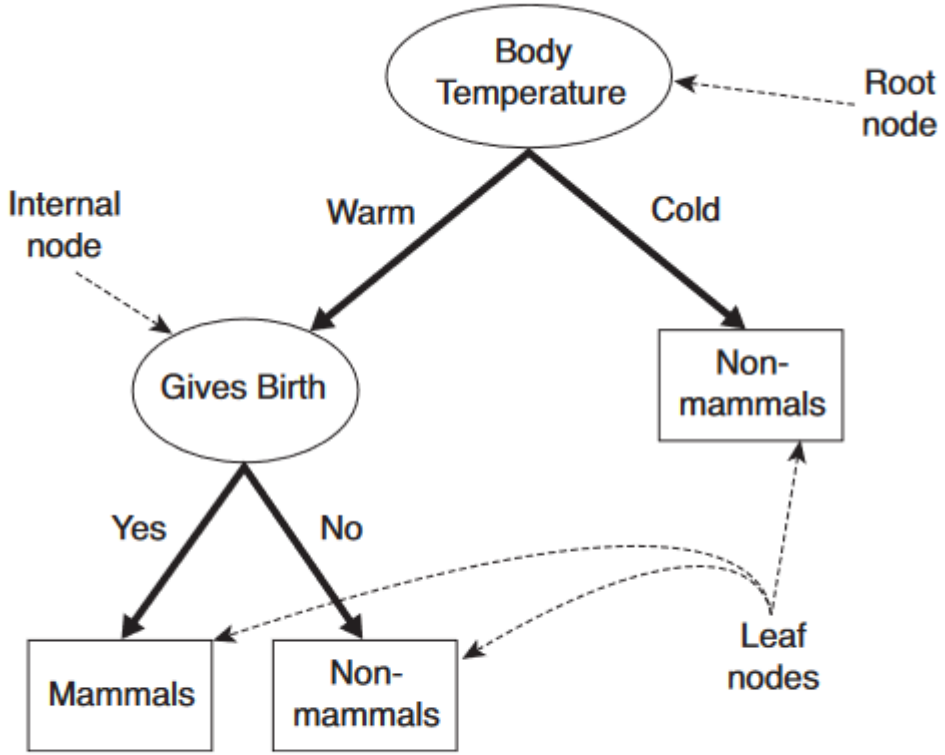
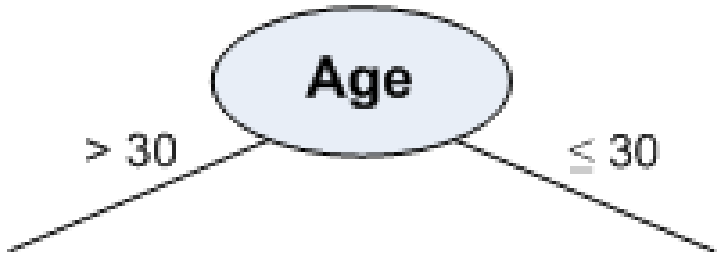


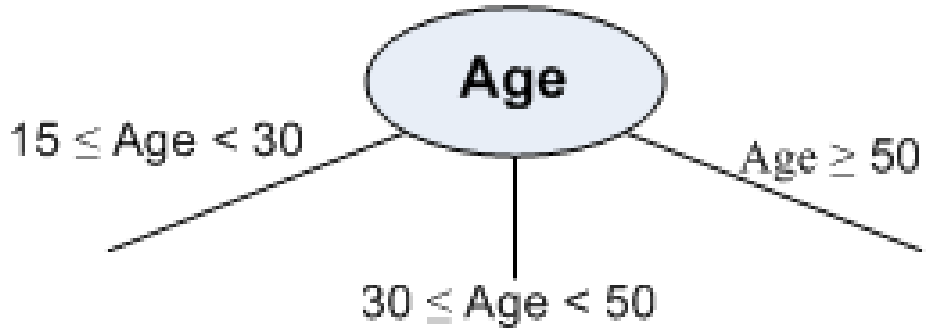
Figure 4.4. A decision tree for the mammal classification problem.

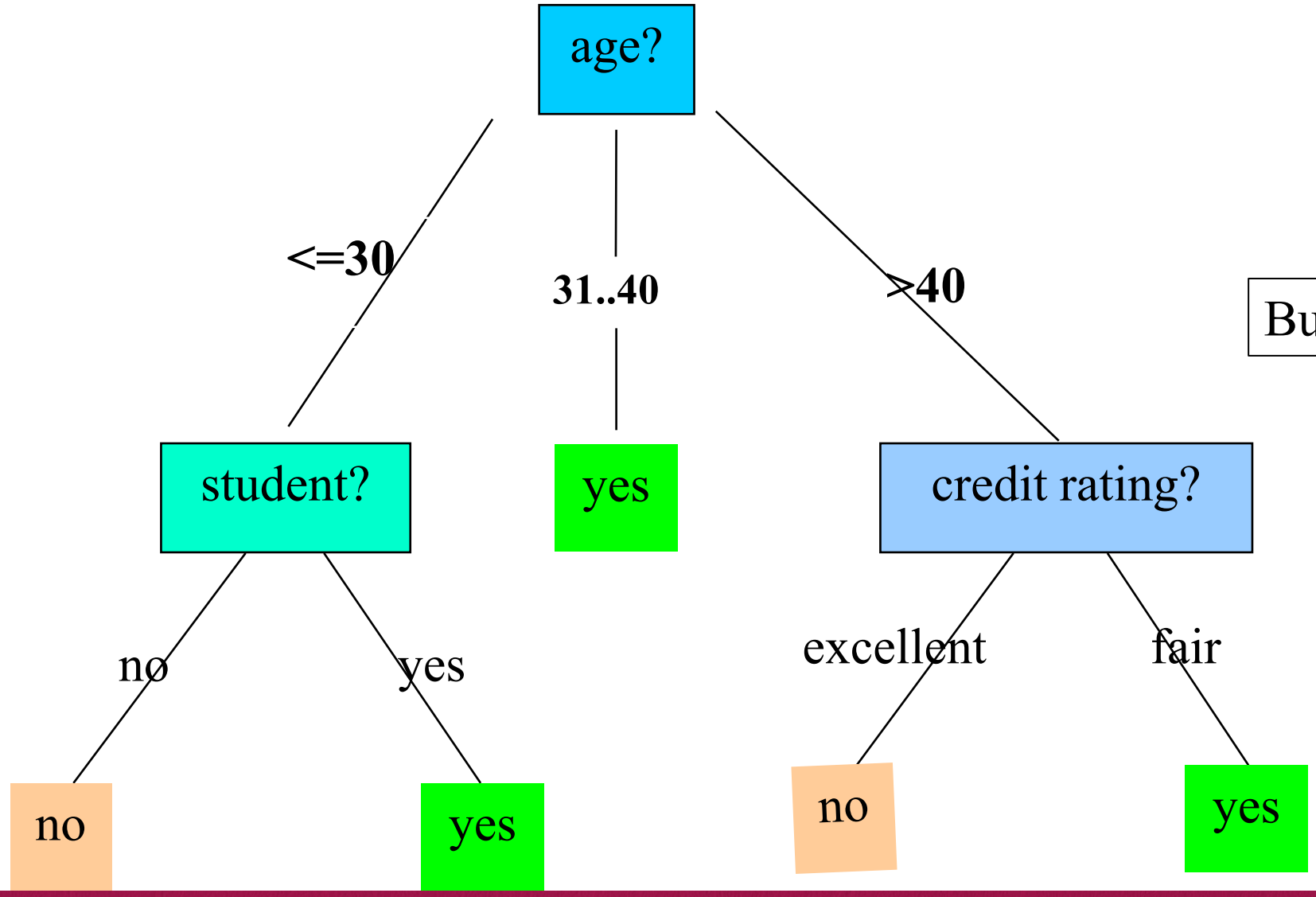
Unlabeled data	Name	Body temperature	Gives Birth	...	Class
	Flamingo	Warm	No	...	?

Class = Non-Mammals

Numerical attribute







Buys computer/not?

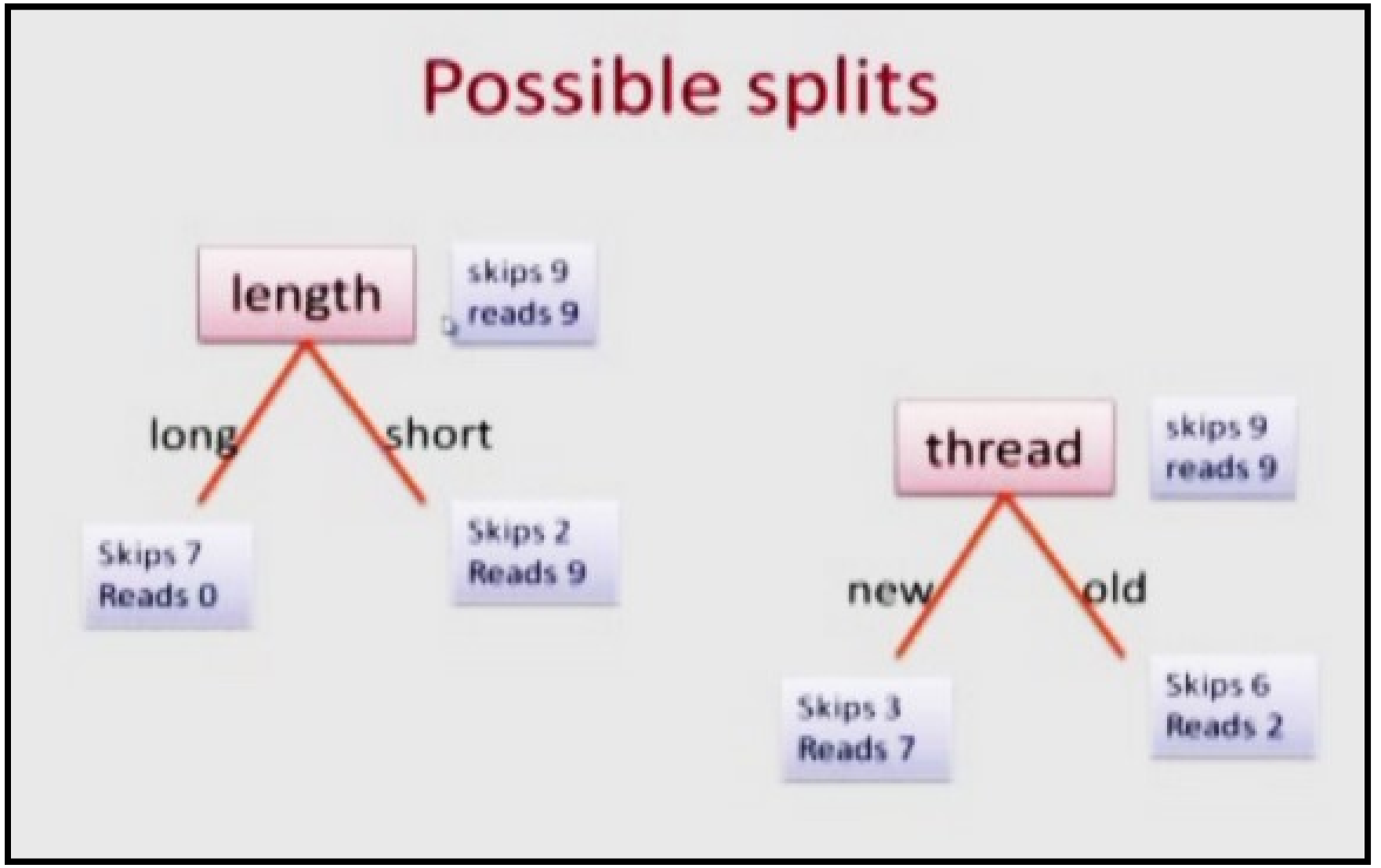
Example data

Training Examples:

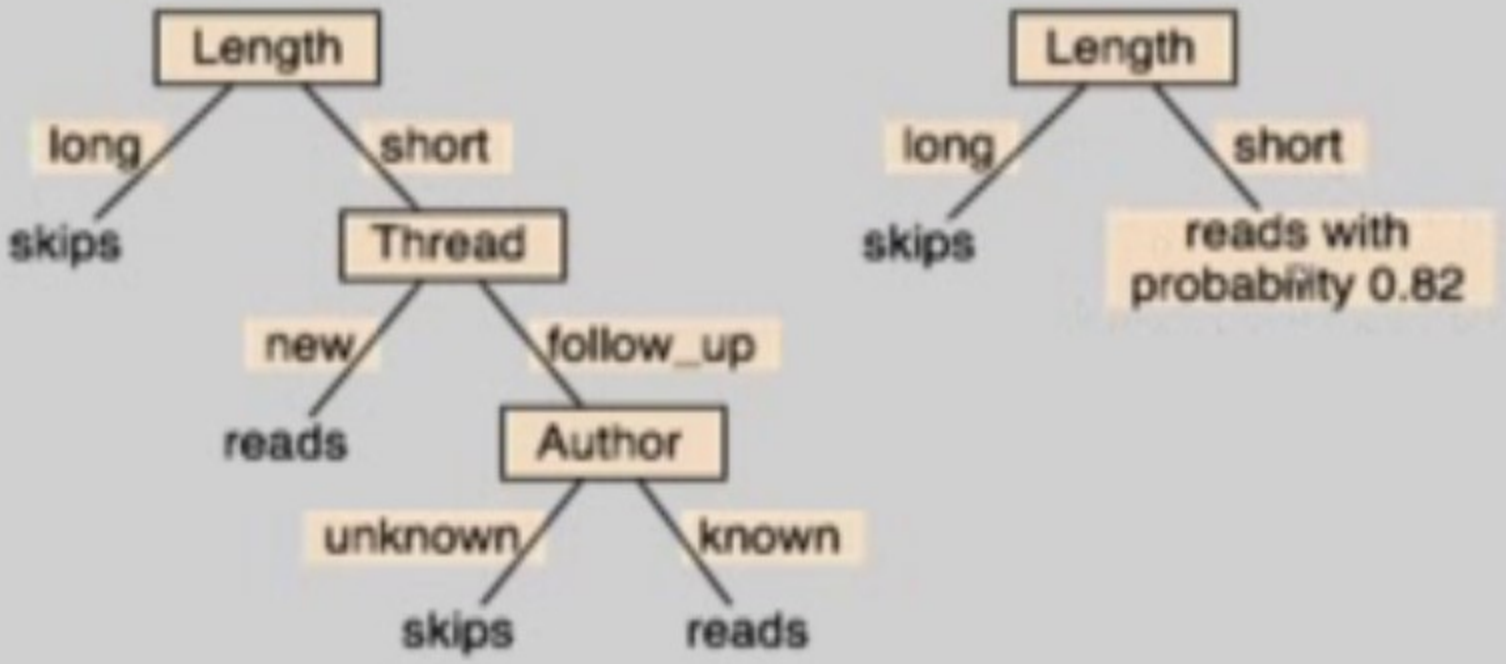
	Action	Author	Thread	Length	Where
e1	skips	known	new	long	Home
e2	reads	unknown	new	short	Work
e3	skips	unknown	old	long	Work
e4	skips	known	old	long	home
e5	reads	known	new	short	home
e6	skips	known	old	long	work

New Examples:

e7	???	known	new	short	work
e8	???	unknown	new	short	work



Two Example DTs



Basic Algorithm for Top-Down Induction of Decision Trees

[ID3, C4.5 by Quinlan]

node = root of decision tree

Main loop:

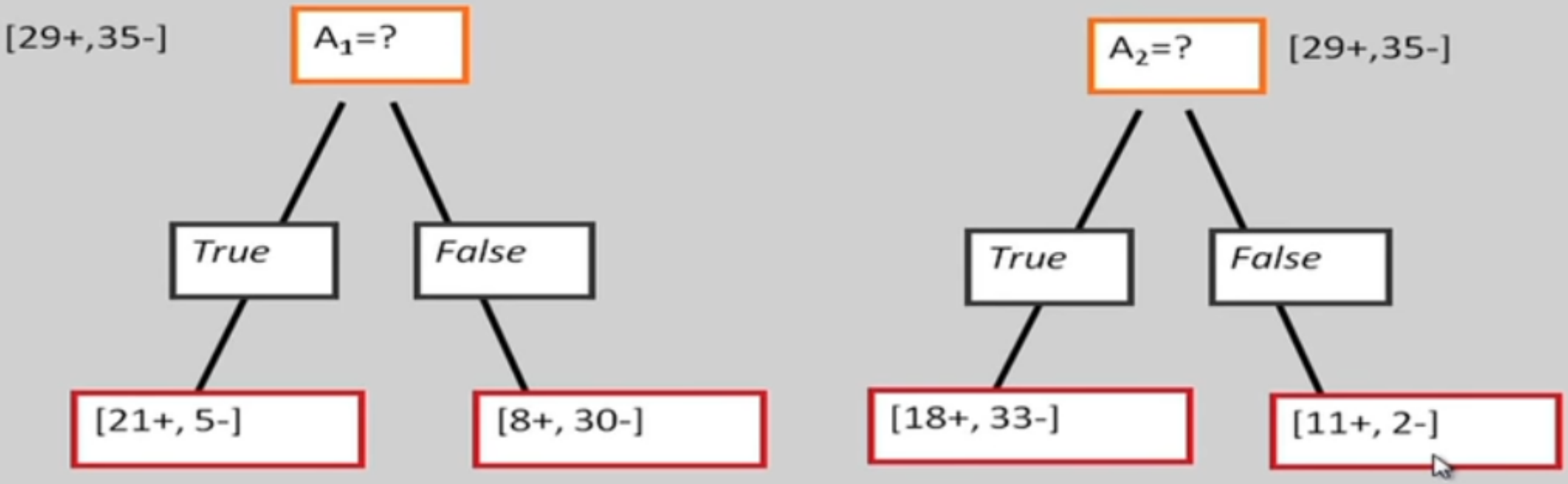
1. *A* ← the "best" decision attribute for the next node.
2. Assign *A* as decision attribute for *node*.
3. For each value of *A*, create a new descendant of *node*.
4. Sort training examples to leaf nodes.
5. If training examples are perfectly classified, stop.
   Else, recurse over new leaf nodes.

How do we choose which attribute is best?

Choices

- When to stop
  - no more input features
  - all examples are classified the same
  - too few examples to make an informative split
- Which test to split on
  - split gives smallest error.
  - With multi-valued features
    - split on all values or
    - split values into half.

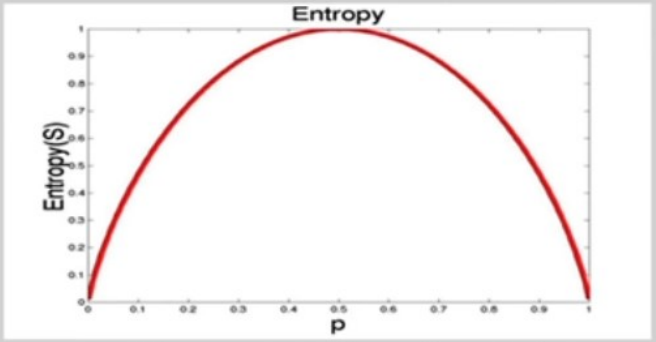
Which Attribute is "best"?



Principled Criterion

- Selection of an attribute to test at each node - choosing the most useful attribute for classifying examples.
- information gain
  - measures how well a given attribute separates the training examples according to their target classification
  - This measure is used to select among the candidate attributes at each step while growing the tree
  - Gain is measure of how much we can reduce uncertainty (Value lies between 0,1)

Entropy



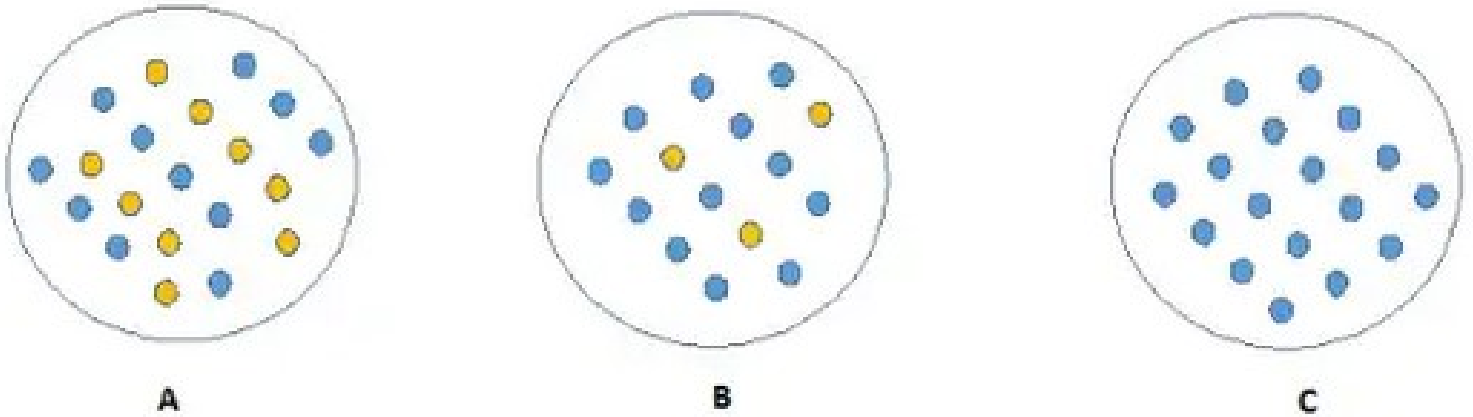
- The entropy is 0 if the outcome is ``certain''.
- The entropy is maximum if we have no knowledge of the system (or any outcome is equally possible).

- S is a sample of training examples
- p_+ is the proportion of positive examples
- p_- is the proportion of negative examples
- Entropy measures the impurity of S

$$\text{Entropy}(S) = -p_+ \log_2 p_+ - p_- \log_2 p_-$$

How to choose best decision node

Which node can be described easily?



- Less impure node requires less information to describe it.
- More impure node requires more information.

===> Information theory is a measure to define this degree of disorganization in a system known as **Entropy**.

- **Entropy** is **0** if all the members of S belong to the same class.
- **Entropy** is **1** when the collection contains an equal no. of +ve and -ve examples.
- **Entropy** is **between 0 and 1** if the collection contains unequal no. of +ve and -ve examples.

Information Gain

Gain(S,A): expected reduction in entropy due to partitioning S on attribute A

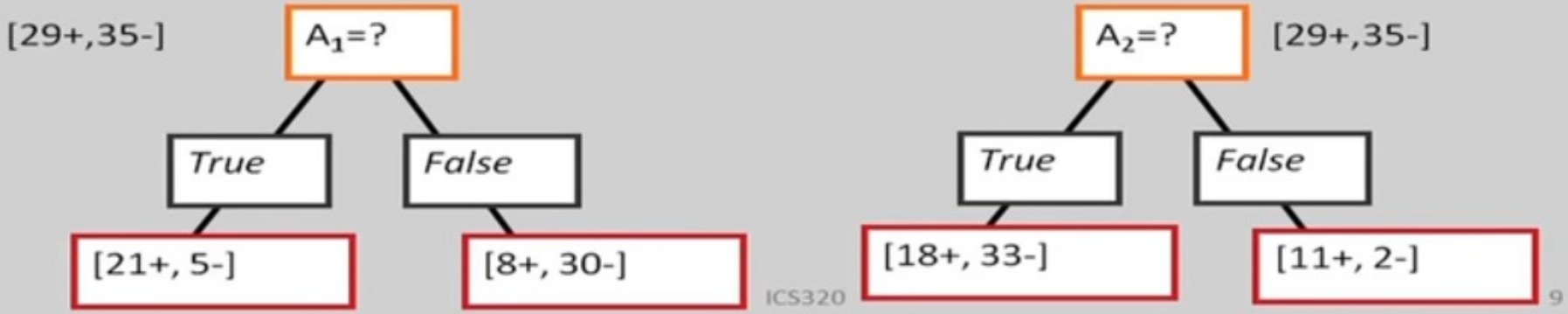
$$Gain(S,A)=Entropy(S) - \sum_{v \in values(A)} |S_v|/|S| \; Entropy(S_v)$$

$$Entropy([29+,35-]) = -29/64 \log_2 29/64 - 35/64 \log_2 35/64 = 0.99$$

Information Gain

Entropy([21+,5-]) = 0.71
Entropy([8+,30-]) = 0.74
Gain(S,A_1)=Entropy(S)
-26/64*Entropy([21+,5-])
-38/64*Entropy([8+,30-])
=0.27

Entropy([18+,33-]) = 0.94
Entropy([8+,30-]) = 0.62
Gain(S,A_2)=Entropy(S)
-51/64*Entropy([18+,33-])
-13/64*Entropy([11+,2-])
=0.12



Example

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

$$\text{Entropy}(S) = -\frac{9}{14}\log_2\frac{9}{14} - \frac{5}{14}\log_2\frac{5}{14}$$

$$= 0.940$$

- Class P: buys_computer = “yes”
- Class N: buys_computer = “no”

Age

Gain(S, Age) $Gain(S, A) \equiv Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$

$= Entropy(S) - \frac{5}{14} Entropy([2+, 3-])$

$- \frac{4}{14} Entropy[4+, 0-]$ $- \frac{5}{14} Entropy[(3^+, 2^-)]$

$= 0.94 - \frac{5}{14}\left[\frac{-2}{5}\log_2\frac{2}{5} - \frac{3}{5}\log_2\frac{3}{5}\right]$

$- \frac{4}{14}\left[\frac{-4}{4}\log_2\frac{4}{4}\right] - \frac{5}{14}\left[\frac{-3}{5}\log_2\frac{2}{5} - \frac{2}{5}\log_2\frac{2}{5}\right]$

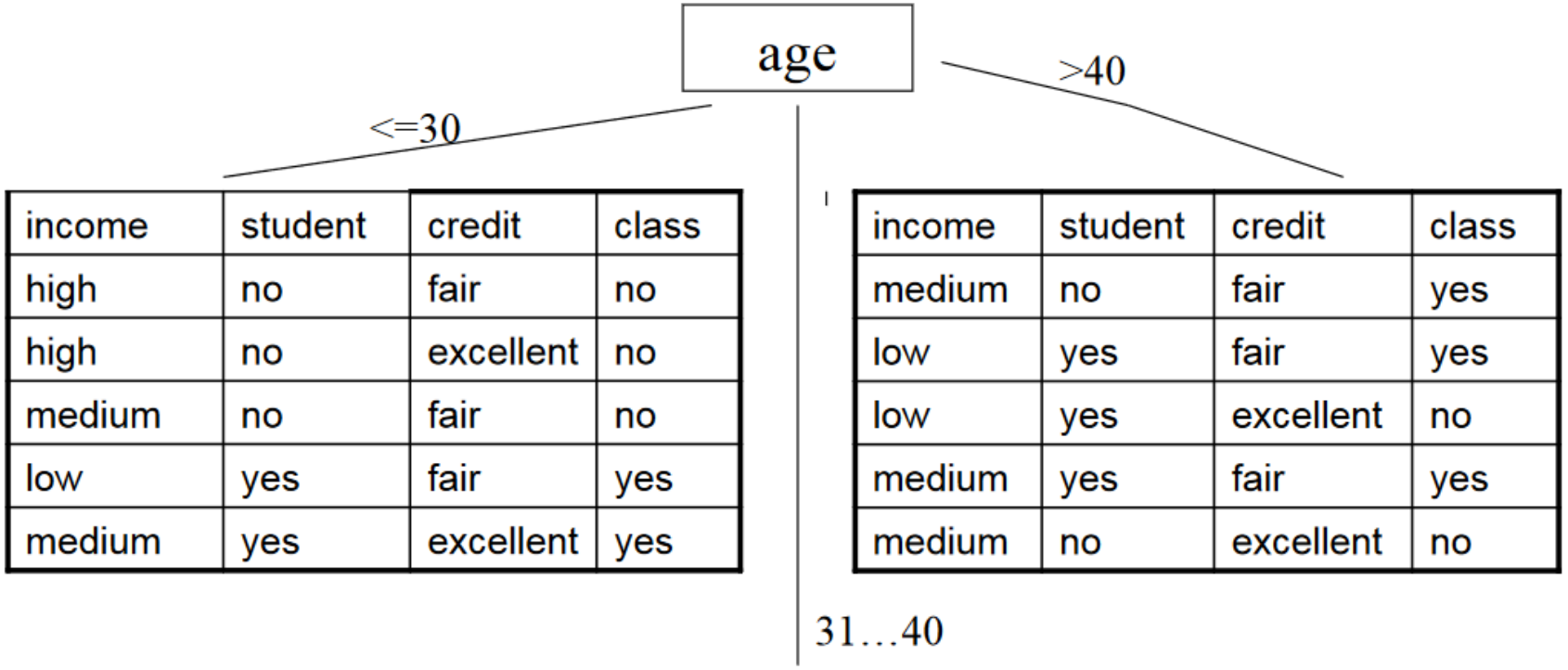
$= 0.94 - \frac{5}{14}[0.968] - \frac{4}{14}[0] - \frac{5}{14}[0.968$

$= 0.94 - 0.36 \; \cancel{= 0.968} \; - 0.36 = \underline{0.25}$

$$Gain(age) = 0.25$$
$$Gain(income) = 0.029$$
$$Gain(student) = 0.151$$
$$Gain(credit\_rating) = 0.048$$

Age has maximum information gain. So age is selected as the best node to split . So age is selected as root node

Building The Tree: we choose "age" as a root

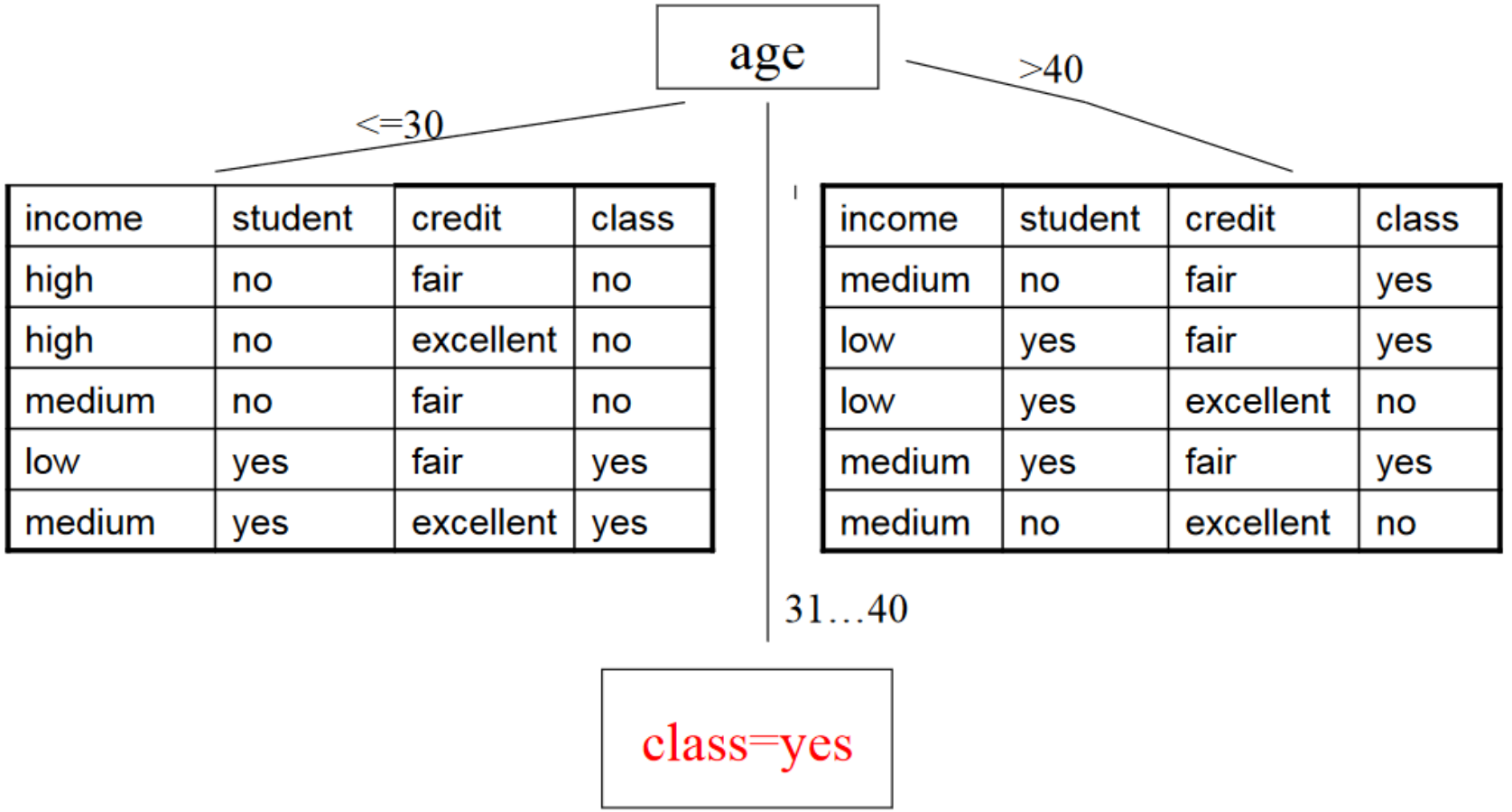


income	student	credit	class
high	no	fair	no
high	no	excellent	no
medium	no	fair	no
low	yes	fair	yes
medium	yes	excellent	yes

income	student	credit	class
medium	no	fair	yes
low	yes	fair	yes
low	yes	excellent	no
medium	yes	fair	yes
medium	no	excellent	no

income	student	credit	class
high	no	fair	yes
low	yes	excellent	yes
medium	no	excellent	yes
high	yes	fair	yes

Building The Tree: “age” as the root



income	student	credit	class
high	no	fair	no
high	no	excellent	no
medium	no	fair	no
low	yes	fair	yes
medium	yes	excellent	yes

income	student	credit	class
medium	no	fair	yes
low	yes	fair	yes
low	yes	excellent	no
medium	yes	fair	yes
medium	no	excellent	no

age

<=30

income	student	credit	class
high	no	fair	no
high	no	excellent	no
medium	no	fair	no
low	yes	fair	yes
medium	yes	excellent	yes

$S \rightarrow age <= 30 \quad [3^+, 2^-]$

$E(S_{age}) = \frac{-3}{5} \log \frac{3}{5} - \frac{2}{5} \log \frac{2}{5}$

$= 0.968$

$Gain(S_{age}, Income) = E(S_{age}) - \frac{2}{5} E([0^+, 2^-]) - \frac{2}{5} E([1^+, 1^-]) - \frac{1}{5} E([1^+, 0^-])$

$= 0.968 - 0 - \frac{2}{5}\left[\frac{1}{2} \log \frac{1}{2} - \frac{1}{2} \log \frac{1}{2}\right] - 0$

$= 0.568$

age

<=30

income	student	credit	class
high	no	fair	no
high	no	excellent	no
medium	no	fair	no
low	yes	fair	yes
medium	yes	excellent	yes

$Gain(S_{<=30}, student)$

$= 0.968 - \frac{3}{5}\left[E([0^+, 3^-])\right] - \frac{2}{5}\left[E([2^+, 0^-])\right]$

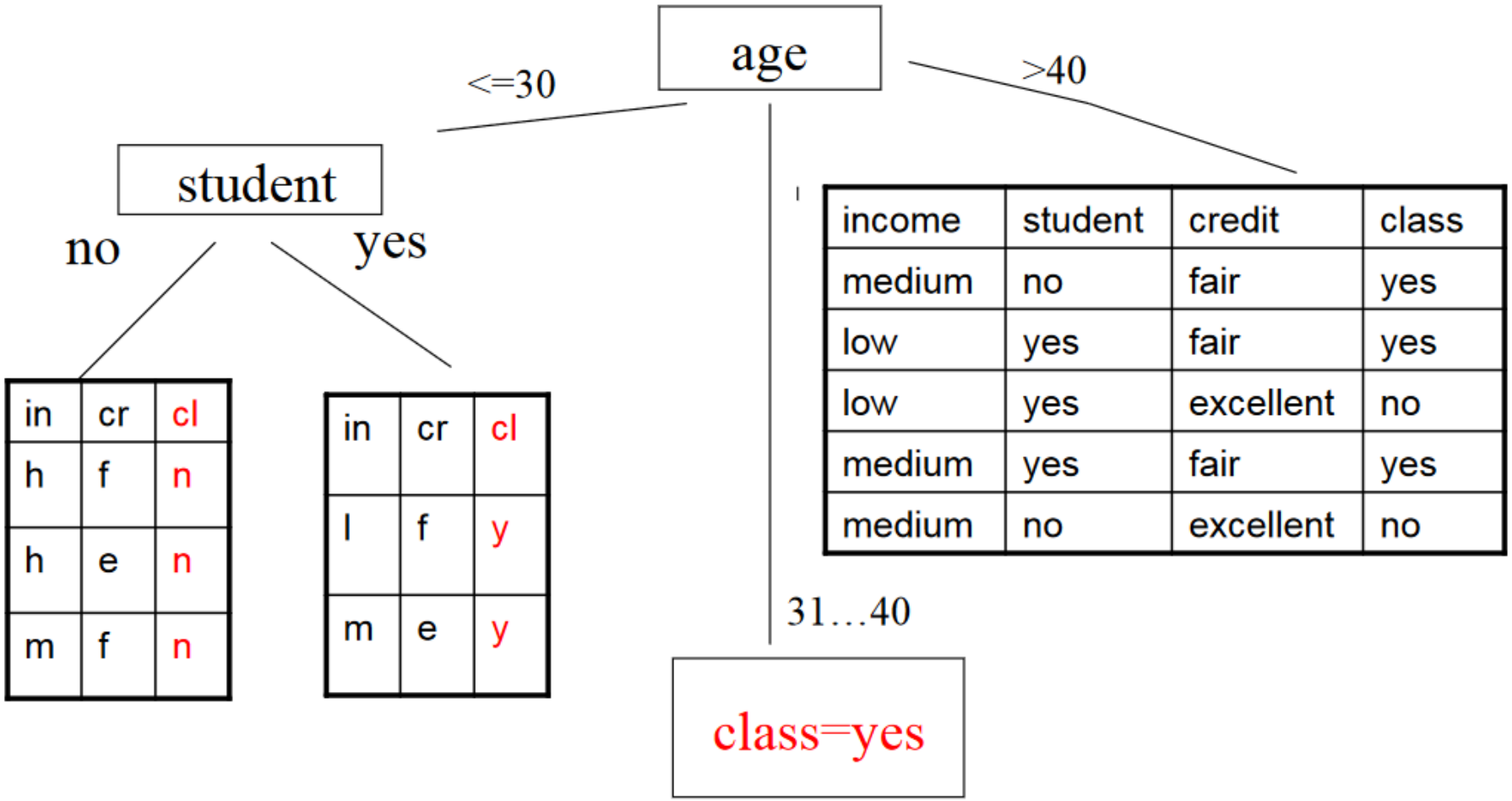
$= 0.968$

$Gain(S_{<=30}, credit)$

$= 0.968 - \frac{3}{5}E([1^+, 2^-]) - \frac{2}{5}E([1^+, 1^-])$

' max for student attribute

Building The Tree: we chose "student" on <=30 branch

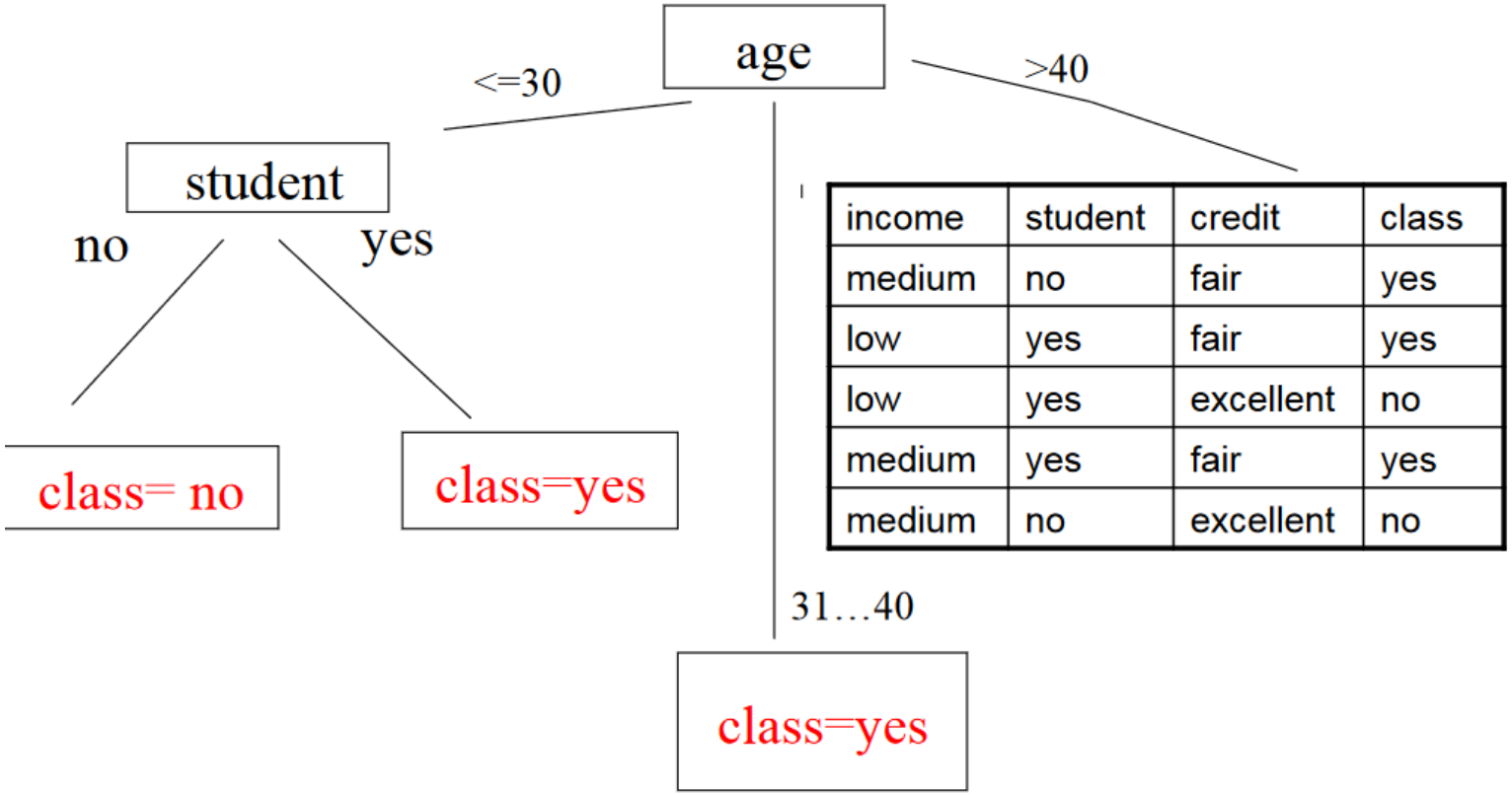


income	student	credit	class
medium	no	fair	yes
low	yes	fair	yes
low	yes	excellent	no
medium	yes	fair	yes
medium	no	excellent	no

in	cr	cl
h	f	n
h	e	n
m	f	n

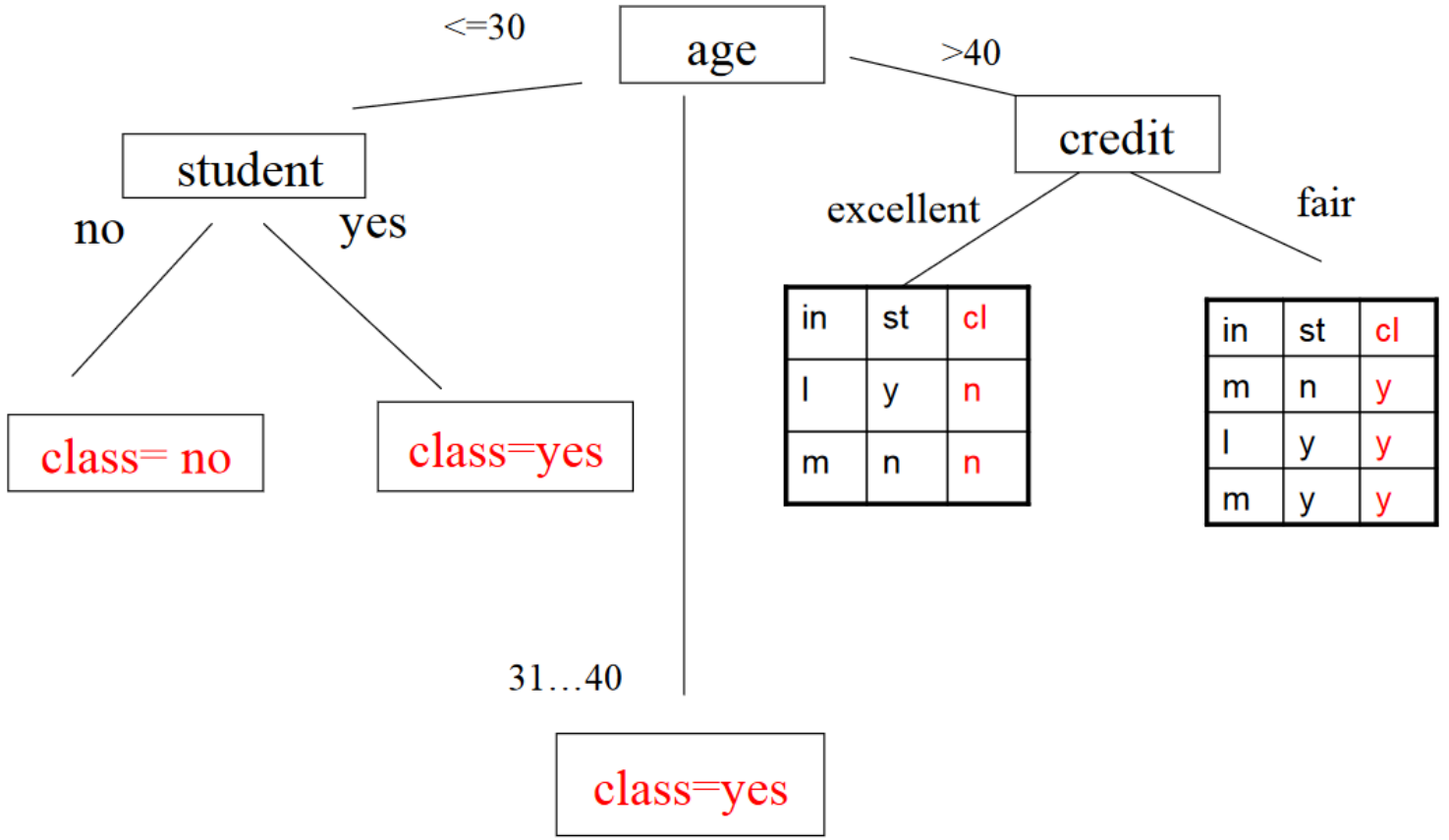
in	cr	cl
l	f	y
m	e	y

Building The Tree: we chose "student" on <=30 branch



income	student	credit	class
medium	no	fair	yes
low	yes	fair	yes
low	yes	excellent	no
medium	yes	fair	yes
medium	no	excellent	no

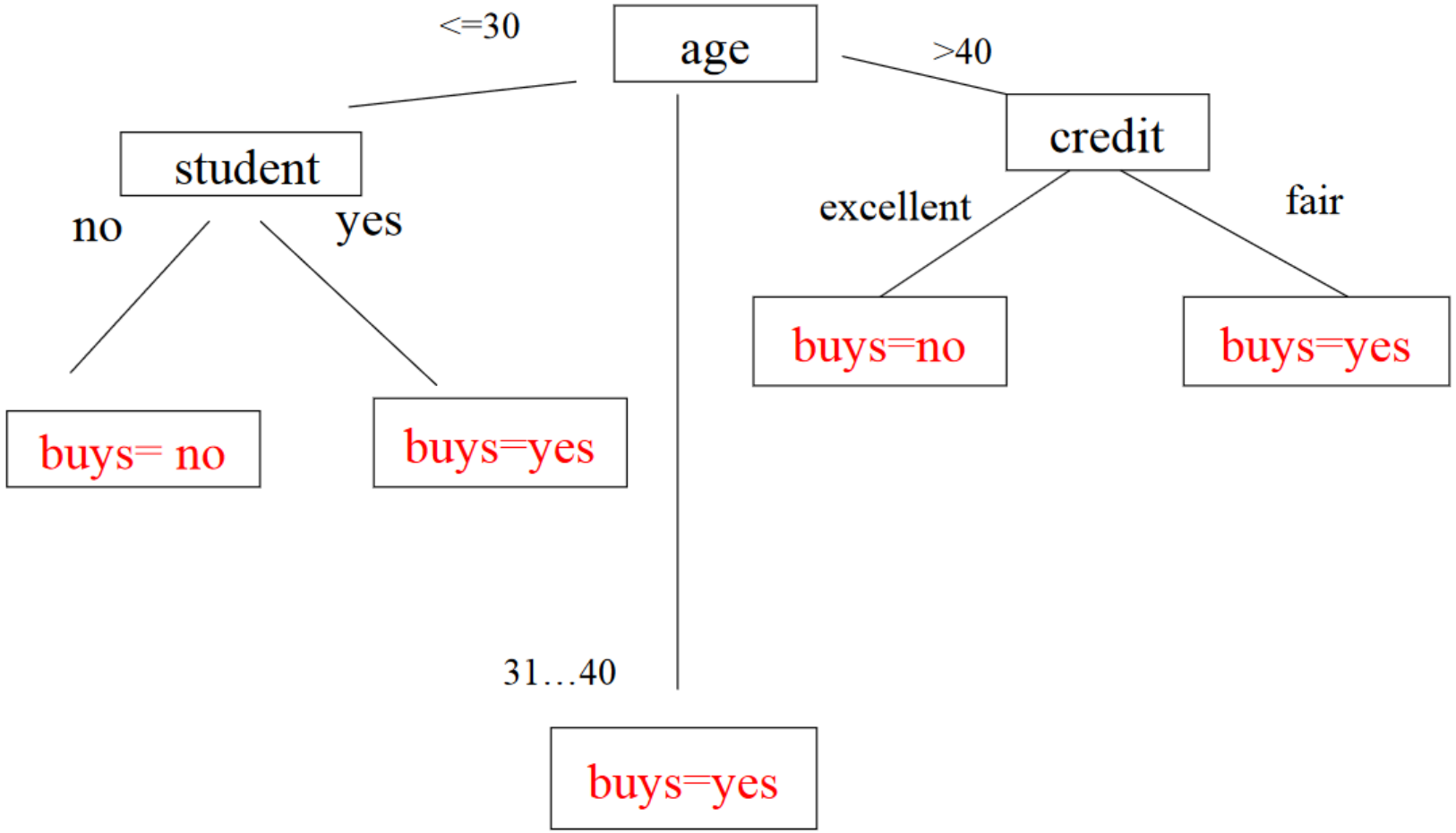
Building The Tree: we chose “credit” on >40 branch



in	st	cl
l	y	n
m	n	n

in	st	cl
m	n	y
l	y	y
m	y	y

Final tree



Rules extracted from the tree

- The rules are:

IF *age* = "<=30" AND *student* = "*no*" THEN
buys_computer = "*no*"

IF *age* = "<=30" AND *student* = "*yes*" THEN
buys_computer = "*yes*"

IF *age* = "31…40" THEN
buys_computer = "*yes*"

IF *age* = ">40" AND *credit_rating* = "*excellent*" THEN
buys_computer = "no"

IF *age* = ">40" AND *credit_rating* = "*fair*" THEN
buys_computer = "yes"

Inductive Bias

- Shorter trees are preferred over larger trees

Overfitting and Tree Pruning

- Overfitting: An induced tree may overfit the training data
  - Too many branches, some may reflect anomalies due to noise or outliers
  - Poor accuracy for unseen samples
- Two approaches to avoid overfitting
  - Prepruning: *Halt tree construction early*-do not split a node if this would result in the goodness measure falling below a threshold
    - Difficult to choose an appropriate threshold
  - Postpruning: *Remove branches* from a "fully grown" tree—get a sequence of progressively pruned trees
    - Use a set of data different from the training data to decide which is the "best pruned tree"

Attribute Selection Measures

- **Information gain**:
  - biased towards multivalued attributes
- **Gain ratio**:

$$GainRatio(S, A) \equiv \frac{Gain(S, A)}{SplitInformation(S, A)}$$

$$SplitInformation(S, A) \equiv -\sum_{i=1}^{c} \frac{|S_i|}{|S|} \log_2 \frac{|S_i|}{|S|}$$

  - tends to prefer unbalanced splits in which one partition is much smaller than the others

- **Gini index**:

$$gini(D) = 1 - \sum_{j=1}^{n} p_j^2$$

  - where pj is the relative frequency of class j in D

  - Choose attribute with low gini index
  - has difficulty when # of classes is large
  - tends to favor tests that result in equal-sized partitions and purity in both partitions

Decision tree suited when -

- Instances are represented by attribute-value pairs
- The target function has discrete output values
- The training data may contain errors.
- The training data may contain missing attribute values

Thank you !!!!!