(*) ARTIFICIAL NEURAL NETWORKS +

Backpropagation.

INTRODUCTION TO ANN:-

→ Human brain has millions of neurons. In Biological term, these are the smallest unit/organ of human brain which performs sensory task (i.e each of all the sense organs of our body are stimulated by the neurons)

→ In Biological term → called as Neurons
In formal term (i.e in AI) → Nodes

→ Ex: When we touch a hot bowl, then we feel immediately take out our hand. Because our skin senses that bowl is hot & pass this input to the neurons. Then neurons process the input & generates some actions (i.e, we take out our hand) All these sensing, processing & generating actions are performed within a fraction of seconds. This is how neurons work.

→ ~~In the same way~~, In AI, natural functions are done artificially.

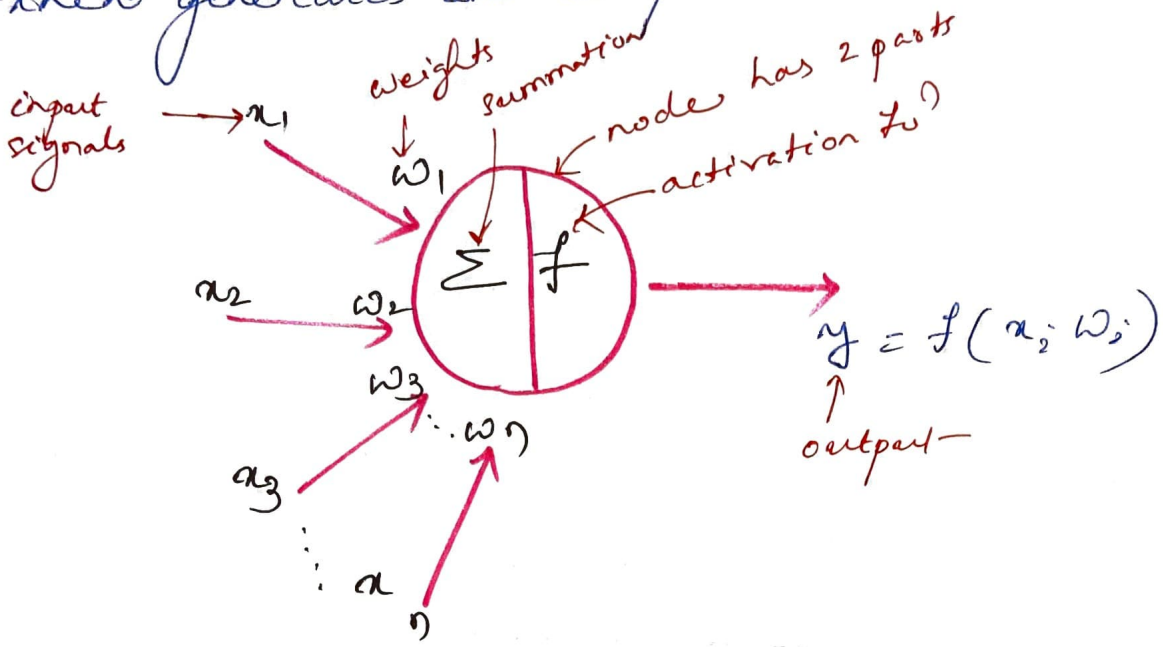
→ def?:-

An ANN is usually a computational NW, based on biological neural networks that construct the structure of the human brain.

Similar to a human brain has neurons interconnected to each other, ANNs also have neurons that are linked to each other in various layers of the networks. These neurons are called nodes.

→ In the same way nodes in AI also works.

- Nodes take the input,
- using some fn. it processes,
- then generates the output.



A node has 2 parts

1) Summation

→ Calculates the weighed sum of all the inputs.

$$\sum x_i w_i = x_1 w_1 + x_2 w_2 + x_3 w_3 + \cdots + x_n w_n$$

→ Once weighted sum is calculated

↓

It is sent to the activation function.

2) Activation function

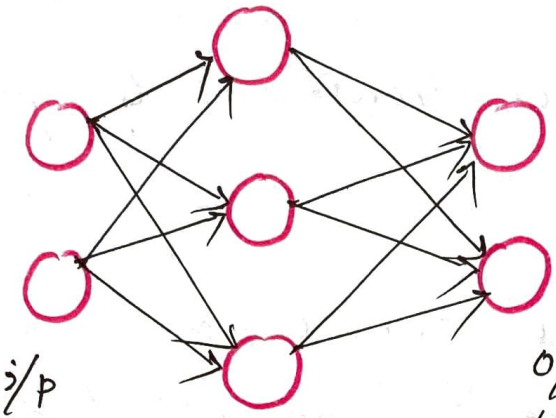
→ Generates the output based on input given

$$y = f(x_i w_i)$$

REPRESENTATION OF ANN:-

→ ANNs are divided into 3 parts

1. Input layer
2. Hidden layer — can be (1) layer (single layer ANN) / (>1) layer (multilayer ANN)
3. Output layer.



i/p layer
(receives the input signal & sent to hidden layer)

Hidden layer
(This layer consists of neurons which extract the relevant info from the input signal & processed to generate the o/p.) & sent to o/p layer.
- This layer is important as it performs
  - data processing
  - generates o/p.

o/p layer.
(responsible for producing & presenting the output to the user)

This is how ANN works.

Input layer :-

It accepts inputs in several different formats provided by the programmer.

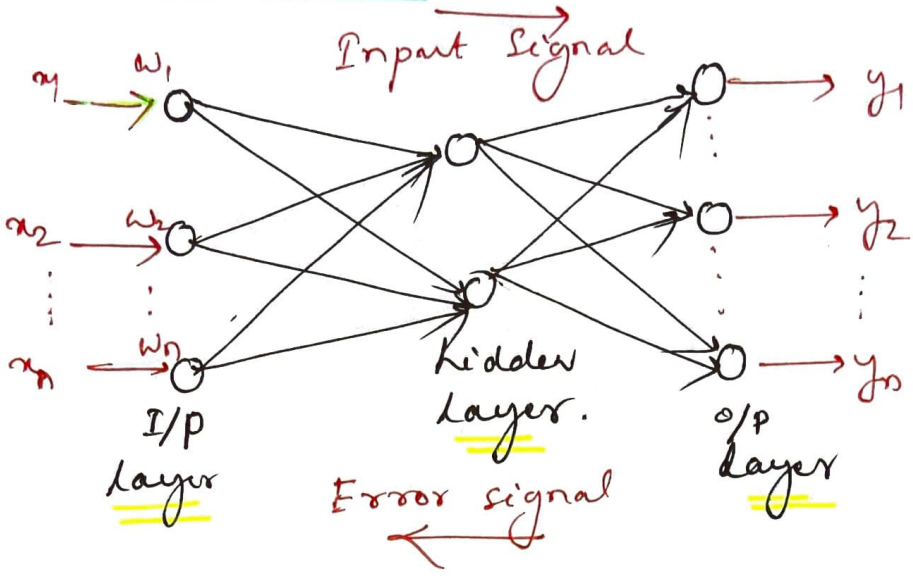
Hidden layer :-

- It is present bet.? input & output layer.
- It performs all the calculations to find hidden features & patterns.

Output layer :-

The input goes through a series of transformations using the hidden layer which finally results in output that is conveyed using this layer.

HOW ANNs WORK ?



→ ANN can be best represented as a weighted ~~graph~~ directed graph, where artificial neurons form the node.

→ The association between the neurons outputs & neuron inputs can be viewed as the directed edges with weights.

→ The ANN receives the input signal from the external source in the form of a pattern & image in the form of a vector. These inputs are then mathematically assigned by the notations (x_n) for every n no. of inputs like $x_1, x_2 \ldots x_n$.

→ Then, each of the input is multiplied by its corresponding weights.

(Here, weights → the strength of the interconnection between neurons inside the ANN)

All the weighted inputs are summarized inside the computing unit.

→ If the weighted sum is = 0, then bias is added to make the output non-zero (i.e, something is added to scale-up to the system's response.
(Bias has the same input, weight = 1)

Here the total of weighted inputs can be in the range of (0 to +∞)

Then the total of weighted input is passed through the activation fu?.

→ The activation fu? refers to the set of transfer fu?s used to achieve the desired output.

⇒ This is how ANN works.

APPROPRIATE PROBLEMS FOR ANN
i.e, Where it works ?

Situations where ANN algm. can apply;

→ Instances have many attribute value pairs.

→ Target function has discrete value, (independent-)
continuous values (dependent) or combination of both.

→ Training examples with errors or missing values.

→ long training times (∵ machine is trained with many attribute value pairs.) are acceptable.

→ Fast evaluation of the target function learent. (testing is faster)

→ Ability for humans to understand the target function learent by machine is not important. (i.e, what machine has learent difficult to understand by human in ANN)

ADVANTAGES OF ANNs

→ parallel processing capability.

→ Storing data on the entire network.

→ Capability to work with incomplete knowledge.

→ Having fault tolerance.

DISADVANTAGES OF ANNs

→ Assurance of proper network structure

→ Hardware dependence.

→ Difficulty of showing the issue to the network.

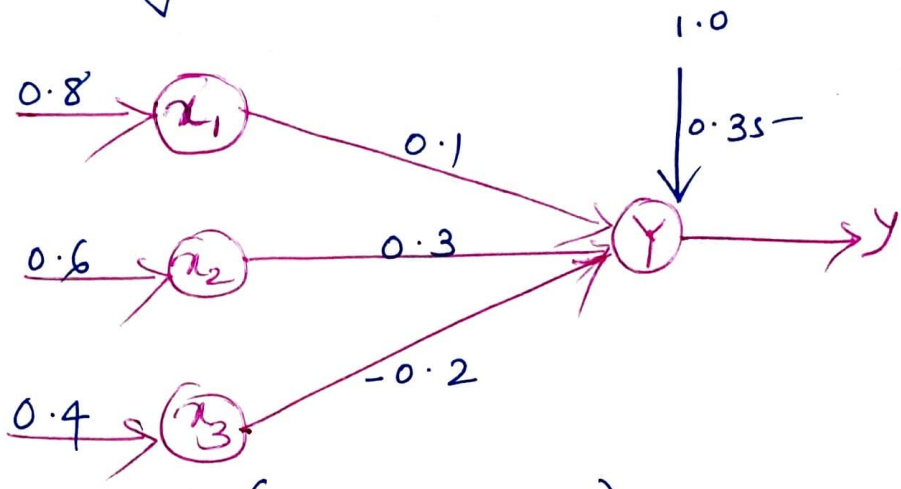
→ The duration of the network is unknown.

⊛ SIGMOID ACTIVATION FUNCTION PROBLEM ①

Question:-

Obtain the output of the neuron Y using
i) ~~bipolar~~ binary sigmoidal &
ii) bipolar sigmoidal activation function.

Solution:-



(Given ANN)

→ In this ANN,
2 layers are given :-
Input layer
↓
Output layer.

→ Input layer is having 3 neurons. (x_1, x_2, x_3).
Output layer is having 1 neuron. (Y)
Each input neuron is connected to output neuron with some weights.

$\overrightarrow{x_1 Y} = 0.1$
$\overrightarrow{x_2 Y} = 0.3$
$\overrightarrow{x_3 Y} = -0.2$

3 inputs to 3 neurons are 0.8, 0.6, 0.4.
0.35 is the bais weight

→ There are 2 types of sigmoid functions :-

1) Binary Sigmoid activation function.

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

(where, y_{in} → total input to Y)

11) Bipolar Sigmoid Activation function. (2)

$$y = f(y_{in}) = \frac{2}{1+e^{-y_{in}}} - 1$$

→ Calculation of net input to the output neuron. (y_{in})

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

(Here $n = 3$ = no. of input neurons.
$b = bais = 0.35$)

$$= b + x_1 w_1 + x_2 w_2 + x_3 w_3$$
$$= 0.35 + (0.8 \times 0.1) + (0.6 \times 0.3) + (0.4 \times 0.2)$$
$$= 0.35 + 0.08 + 0.18 - 0.08$$
$$= 0.53$$

→ Calculation of Binary Sigmoid Activation function (y)

$$y = f(y_{in})$$
$$= \frac{1}{1+e^{-y_{in}}} = \frac{1}{1+e^{-0.53}} = 0.625$$

→ Calculation of Bipolar Sigmoid Activation function(y)

$$y = f(y_{in})$$
$$= \frac{2}{1+e^{-y_{in}}} - 1$$
$$= \frac{2}{1+e^{-0.53}} - 1 = 0.259.$$

→ **Perceptron** is a single layer neural network & a multilayer perceptron is called Neural Network.

→ Perceptron is a lineare classifier (binary)

→ It is used in supervised learning.

→ It helps to classify the given input data.

→ The perceptron consists of 4 parts:-
- input values or one input layer.
- weights or Bias
- Net sum
- Activation fu.) (transformation / step fu.)

→ Perceptron works on 3 steps:-

a) All the inputs x are multiplied with their weights w.

b) Add all the multiplied values & call them as weighted sum.

c) Apply that weighted sum to the correct Activation function.

→ Perceptron are the basic unit to build ANN.

→ It takes real valued i/p, calculates linear combination of these inputs & generates the output.

$$\begin{cases} \text{output} = 1 & \text{if result} > \text{threshold} \\ = -1 & \text{otherwise.} \end{cases}$$

Here the word linear combination mean,

$$O(x_1, x_2, \dots x_n) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + w_2 x_2 \dots + w_n x_n > 0 \\ -1 & \text{otherwise} \end{cases}$$

(threshold value)

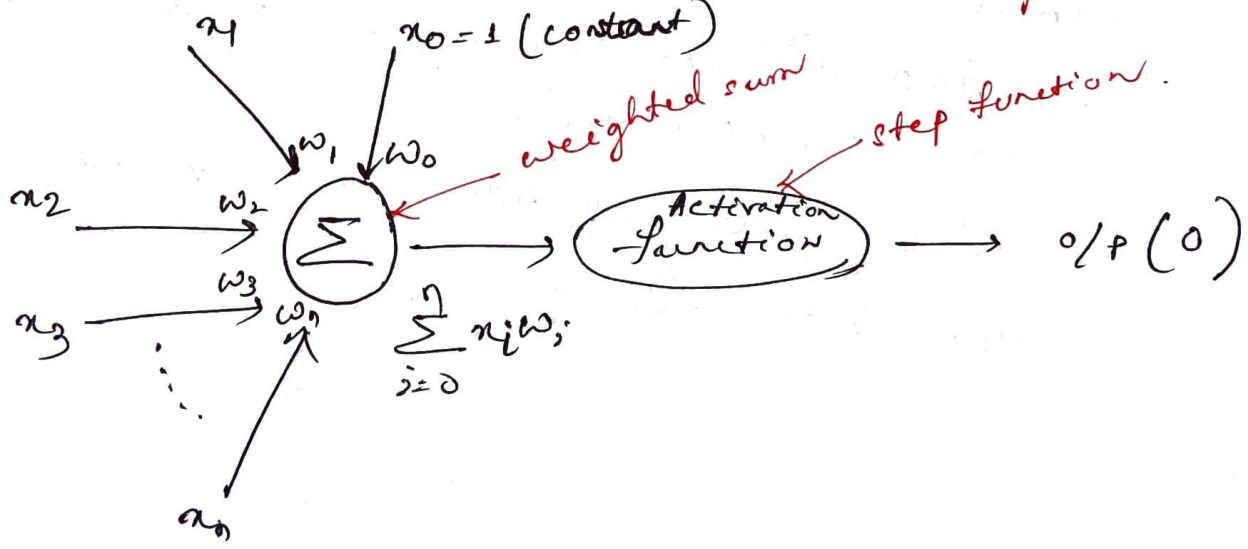
(Where, w_0 → initial weight

$w_1, w_2, \dots w_n$ → corresponding weights of inputs.

$x_0 = 1$ (initial input) → constant

$x_1, x_2, \dots x_n$ → inputs).

→ Perceptron Rule,



* This rule says that,
Each node has 2 parts.

↳ summation
↳ activation fun?.

* Here summation adds all the inputs along with their corresponding weights.

Here, x_0 (critical input) $= 1$.

$x_1, x_2, \dots x_n \rightarrow$ inputs.

$w_1, w_2, \dots w_n \rightarrow$ weights.

$$\sum_{i=0}^{n} w_i x_i = w_0 + w_1 x_1 + w_2 x_2 + \dots + w_n x_n.$$

$= (1) / (-1)$

y if $\sum_{i=0}^{n} w_i x_i > 0$ $\quad$ y if $\sum_{i=0}^{n} w_i x_i < 0$

* Then this value is sent to the activation function which ~~pass~~ is responsible for processing the input & generates the output.

(English alphabet O not zero)

* Here, $O \rightarrow$ actual output (generated by the activation fun.)

$t \rightarrow$ target output (expected output by us)

* If actual output = target output

⇒ then weights are fixed

i.e, we need to consider critical $w_0, w_1, w_2, w_n \dots$ &

for generating the output.

But, if actual output ≠ target output.

⇒ then we have to change the weights.

How to change weights?

Initially, take some random weights.

Later, keep on applying iterations

i.e, find $\sum_{i=0}^{n} w_i x_i$

and check if $(o = t)$

If $o = t$. then stop the process.

If $o \neq t$, then (apply formula to change weight)

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

add this small change in weight to make $(o = t)$

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

(where, η = learning rate (i.e, take small values like 0.1, 0.2, 0.3, ...)

t = target o/p

o = actual o/p

x_i = i/p associated with w_i

here $x_1 \longrightarrow w_1$ is associated with

$x_2 \longrightarrow w_2$

$\vdots$

$x_n \longrightarrow w_n$.

[i.e, $\Delta w_1 = \eta (t - o) x_1$

$\Delta w_2 = \eta (t - o) x_2$

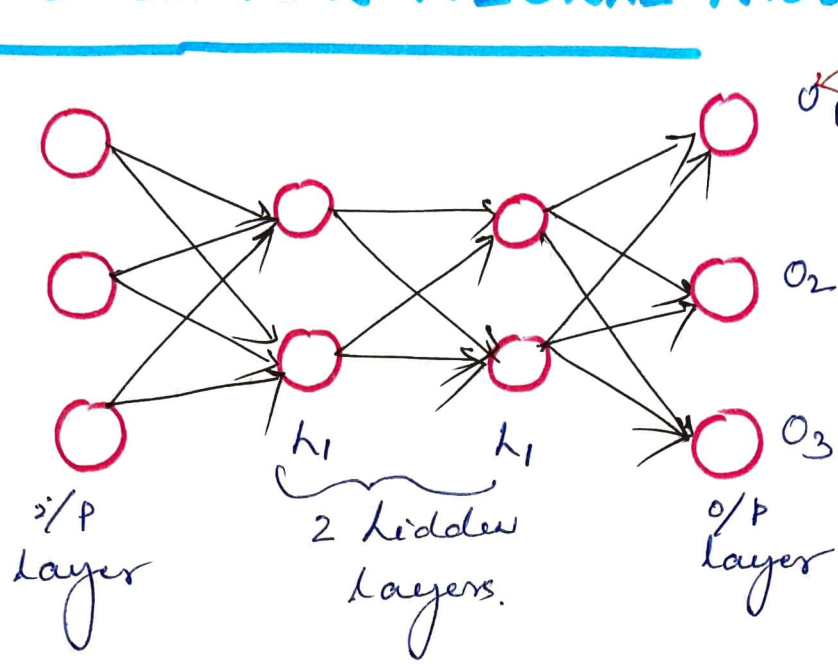
$\Delta w_n = \eta (t - o) x_n$] like this we need to find.

like this, find all the weights & find the $\sum_{i=0}^{n} w_i x_i$ for several iterations ~~& end~~ until $o = t$. i.e actual output = target output.

Once it satisfies, then stop the process and ~~it~~ ~~train the~~ finalize the weights and train the perceptron.

⇒ This is how perceptron training goes on.

✳ MULTI LAYER NEURAL NWs :→



→ The single layer neural network will work only for linearly separable data. It will not work for non linearly separable data. Multi layer neural NWs solves this problem.

→ In multi layer neural network, there are 2 hidden layers and each node in a ~~connected~~ single layer is connected to each and every node of the next layer and so on.

→ Here O_1, O_2, O_3 are the actual outputs and are compared against the target outputs t_1, t_2, t_3 which we are supposed to get. If ~~but~~ both doesn't match, then we ~~can~~ have to go back & modify the weights.

which is called as Back-propagation.

⇒ This is ~~all~~ how multi-layer neural network works.

⊛ BACK PROPAGATION :→ (Backward propagation of error)

→ Back propagation (backward propagation) is an important mathematical tool for improving the accuracy of predictions in data-mining & machine learning.

→ Artificial Neural Networks use backpropg. as a learning algorithm to compute a gradient descent w.r.t weights. Desired outputs are compared to achieved systems outputs, (here desired outputs → target outputs (t), ~~achieved~~ system outputs → actual outputs (o)) and then systems are tuned by adjusting connection weights to narrow the difference between the two as much as possible.

→ This algorithm is called as Back-propagation because the weights are updated backwards, from outputs towards inputs.

→ So, when error occurs, we go in backward direction.

i.e, output → hidden → input layer.

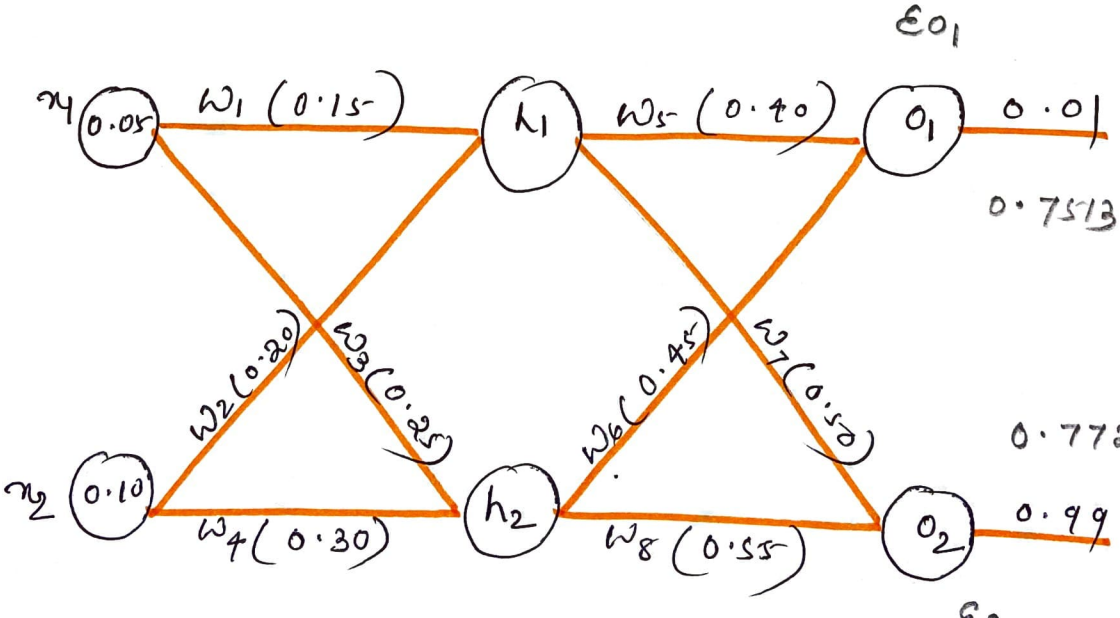
→ Here the propagation error will be calculated in 3 parts.

part1: Calculate forward propagation error.
i.e, from input → hidden → output layer.

part2: Calculate backward propagation error.
i.e. from output layer → hidden layer.

part3: Calculating backward propagation error
i.e. from hidden → input layer.

→ Ex:-



$b_1 = (0.35)$ $b_2 = (0.60)$

Let, the learning rate (η) = 0.6

Here, $x_1, x_2 \rightarrow$ are the inputs (0.05, 0.10)

$h_1, h_2 \rightarrow$ hidden layer.

$o_1, o_2 \rightarrow$ 0.01, 0.99 (target- output are given)

$w_1, w_2 \ldots w_8 \rightarrow$ corresponds weights of all edges.

$b_1 \rightarrow$ is the bias factor from input to hidden layer. (0.35)

$b_2 \rightarrow$ is the bias factor from hidden to output layer. (0.60)

Orange lines → edges.

We have to find out the actual outputs, if it matches with the target outputs, then it is said to be no error. If it does not match, then we have to change the weights in using back propagation algorithm.

part 1: Calculate forward propagation error (input → hidden → output layer)

1) Calculate h_1 (in and out)

$$h_1(in) = x_1 w_1 + x_2 w_2 + b_1$$
$$= (0.05 \times 0.15) + (0.10 \times 0.20) + 0.35$$
$$= 0.377$$

$$h_1(out) = \frac{1}{1 + e^{-h_1(in)}}$$

$$= \frac{1}{1 + e^{-(0.377)}}$$

$$= 0.5932$$

2) Calculate h_2 (in and out)

$$h_2(in) = x_1 w_3 + x_2 w_4 + b_1$$

$$= (0.05 \times 0.25) + (0.10 \times 0.30) + 0.35$$

$$= 0.0125 + 0.03 + 0.35$$

$$= 0.3925$$

$$h_2(out) = \frac{1}{1 + e^{-h_2(in)}}$$

$$= \frac{1}{1 + e^{-(0.3925)}}$$

$$= 0.5968$$

3) Calculate O_1 (in and out)

$$O_1(in) = h_1(out) \times w_5 + h_2(out) \times w_6 + b_2$$

$$= (0.5993 \times 0.40) + (0.5968 \times 0.45) + 0.60$$

$$= 1.105$$

$$O_1(out) = \frac{1}{1 + e^{-O_1(in)}}$$

$$= \frac{1}{1 + e^{-(1.105)}} = 0.7513$$

4) Calculate O_2 (in and out)

$$O_2(in) = h_1(out) \times w_7 + h_2(out) \times w_8 + b_2$$

$$= (0.5932 \times 0.50) + (0.5968 \times 0.55) + 0.60$$

$$= 1.22484$$

$$O_2(out) = \frac{1}{1 + e^{-O_2(in)}}$$

$$= \frac{1}{1 + e^{-(1.22484)}}$$

$$= 0.7729$$

5) Calculate E_{Total}

$$E_{Total} = \sum \frac{1}{2} (target - o/p)^2$$

$$= E_{O_1} + E_{O_2}$$

target output of O_1
actual " "

$$= \frac{1}{2}(0.01 - 0.75113)^2 + \frac{1}{2}(0.99 - 0.7729)^2$$

$$= \frac{1}{2}(-0.7413)^2 + \frac{1}{2}(0.2171)^2$$

$$= 0.274 + 0.0235$$

$$= 0.29837 \text{ (approximately)}$$

(Here E_{Total} → ~~Er~~ Total error
E_{O_1} → error at output O_1
E_{O_2} → " " " O_2)

part 2 : Calculate backward propagation error (output layer → hidden layer)

Now, here we have to adjust the weights which are between output layer & hidden layer.

i.e, w_5, w_6, w_7 and w_8.

first let us adjust w_5,

$$w_5^* = w_5 - \eta \frac{\partial E_{Total}}{\partial w_5}$$

(Here, η → learning rate (0.6) let
w_5 → weight is given.
∂ → partial differentiation)

$$\frac{\partial E_{Total}}{\partial w_5} = \frac{\partial E_{Total}}{\partial out_{O_1}} \times \frac{\partial out_{O_1}}{\partial net_{O_1}} \times \frac{\partial net_{O_1}}{\partial w_5}$$

(apply chain rule)

$$\frac{\partial E_{total}}{\partial out_{O_1}} = out_{O_1} - target_{O_1}$$

$$= 0.751365 - 0.01$$

$$= 0.741365$$

$$\frac{\partial out_{O_1}}{\partial net_{O_1}} = out_{O_1}(1 - out_{O_1})$$

$$= 0.751365 (1 - 0.751365)$$

$$= 0.18681502$$

$$\frac{\partial net_{O_1}}{\partial w_5} = out_{h_1}$$

$$= 0.59326992)$$

Now, $$\frac{\partial E_{Total}}{\partial w_5} = 0.74135655 \times 0.186815602 \times 0.59326992$$

$$= 0.08216704$$

Hence, $$w_5^* = w_5 - \eta \frac{\partial E_{Total}}{\partial w_5}$$

$$= 0.4 - (0.6) \times 0.08216704$$

$$= 0.35069776$$

(Here,
$w_5^* \rightarrow$ modified weight of w_5
$w_5 \rightarrow$ actual weight
$\eta \rightarrow$ learning rate (0.6) let)

Similarly we have to find the modified weights or w_6^*, w_7^* and w_8^*.

After finding the modified weight, repeat part 1 to check the output. Repeat this process several times until the actual output matches the target output (or threshold value) i.e almost nearly equal to zero.

part 3 :- Calculating backward propagation of error. (hidden → input layer)

Here we have to modify the weights w_1, w_2, w_3 and w_4.

First let us adjust w_1.

$$w_1^* = w_1 - \eta \frac{\partial E_{Total}}{\partial w_1}$$

$$\frac{\partial E_{Total}}{\partial w_1} = \frac{\partial E_{Total}}{\partial out(h_1)} \times \frac{\partial out(h_1)}{\partial net(h_1)} \times \frac{\partial net(h_1)}{\partial w_1}$$

(apply chain rule)

$$\frac{\partial E_{Total}}{\partial out(h_1)} = \frac{\partial E_{O_1}}{\partial out h_1} + \frac{\partial E_{O_2}}{\partial out h_1}$$

$$\frac{\partial E_{O_1}}{\partial net_{O_1}} \times \frac{\partial net_{O_1}}{\partial out h_1}$$

$$\frac{\partial E_{O_1}}{\partial out_{O_1}} \times \frac{\partial out_{O_1}}{\partial net_{O_1}}$$

$$\frac{\partial E_{O_2}}{\partial net_{O_2}} \times \frac{\partial net_{O_2}}{\partial out h_1}$$

$$\frac{\partial E_{O_2}}{\partial out_{O_2}} \times \frac{\partial out_{O_2}}{\partial net_{O_2}}$$

$$\frac{\partial E_{O_2}}{\partial out_{O_2}} = (out_{O_2} - target_{O_2})$$

$$= 0.772928465 - 0.99$$

$$= -0.217071535$$

$$\frac{\partial out_{O_2}}{\partial net_{O_2}} = out_{O_2}(1 - out_{O_2})$$

$$= 0.7729(1 - 0.7729)$$

$$= 0.175510052$$

Now,

$$\frac{\partial E_{O_2}}{\partial net_{O_2}} = \frac{\partial E_{O_2}}{\partial out_{O_2}} \times \frac{\partial out_{O_2}}{\partial net_{O_2}}$$

$$= -0.217071535 \times 0.175 \leftarrow 1005.2.$$

$$= \_\_\_\_$$

Similarly, find

$$\frac{\partial E_{O_1}}{\partial net_{O_1}} = \frac{\partial E_{O_1}}{\partial out_{O_1}} \times \frac{\partial out_{O_1}}{\partial net_{O_1}}$$

$\frac{\partial E_{O_1}}{\partial out_{O_1}}$ → $(out_{L} - target_{O_1})$

$\frac{\partial out_{O_1}}{\partial net_{O_1}}$ → $out_{O_1}(1 - out_{O_1})$

= find the product.

Now,

$$\frac{\partial net_{O_1}}{\partial out_{h_1}} = \text{on } O_1 \text{ from } h_1 \Rightarrow w_5$$

$$= 0.40$$

$$\frac{\partial net_{O_2}}{\partial out_{h_2}} = \text{on } O_2 \text{ from } h_1 \Rightarrow w_7$$

$$= 0.50$$

Now,

$$\frac{\partial E_{Total}}{\partial out_{h_1}} = (0.138498562 \times 0.40) + (-0.038098218 \times 0.50)$$

$$= 0.055399425 + (-0.019049119)$$

$$= 0.036350306$$

$$\frac{\partial out_{h_1}}{\partial net_{h_1}} = out_{h_1}(1 - out_{h_1})$$

$$= 0.241\,300\,709$$

$$\frac{\partial net_{h_1}}{\partial w_1} = \frac{\partial (w_1 x_1 + w_2 x_2 + b_1)}{\partial w_1}$$

$$= x_1 = 0.05$$

Now,

$$\frac{\partial E_{Total}}{\partial w_1} = 0.036350306 \times 0.241300709 \times 0.05$$

$$= 0.00438568$$

Now, $w_1^* = w_1 - \eta \frac{\partial E_{Total}}{\partial w_1}$

$$= (0.15) - ((0.6) \times 0.00438568)$$

$$= 0.149736892.$$

Similarly we have to calculate

w_2^*, w_3^*, w_4^*.

In ~~the previous~~ part2, we have calculated

w_5^*, w_6^*, w_7^* & w_8^*

Using all these updated weights, proceed part1 i.e, calculate the error again & again until the error is being is being reduced i.e almost nearly equal to zero or below the target/threshold value.

SUPPORT VECTOR MACHINES (SVM)

Presented by

Ms. Bidyutlata Sahoo

Assistant Professor

CSE (AI & ML)

OBJECTIVES

The objective of SVM algorithm is ***to find a hyperplane in an N-dimensional space that distinctly classifies the data points***.

The dimension of the hyperplane depends upon the number of features.

OUTCOMES

SVM or Support Vector Machine is ***a linear model for classification and regression problems.*** It ***can solve linear and non-linear problems*** and work well for many practical problems.

SUPPORT VECTOR MACHINES (SVM)

- Support Vector Machine or SVM is one of the most popular ***Supervised Learning algorithms***, which is ***used for Classification as well as Regression problems.***
- However, primarily, it is used for Classification problems in Machine Learning.
- The **goal of the SVM** algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. ***This best decision boundary is called a hyperplane***.

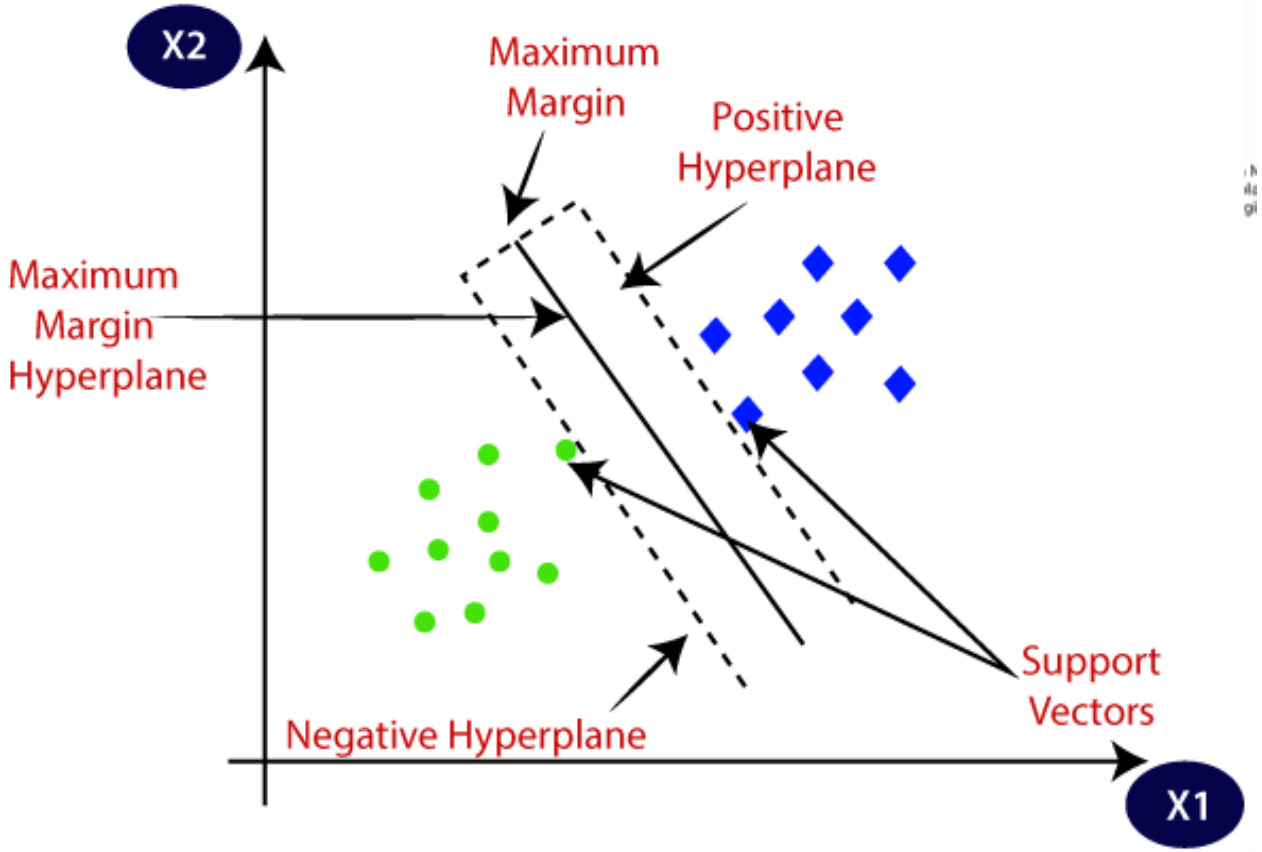
SUPPORT VECTOR MACHINES (SVM)

SVM chooses the ***extreme points/vectors*** that help in creating the hyperplane. These extreme cases are called as ***support vectors***, and hence algorithm is termed as Support Vector Machine.

SUPPORT VECTOR MACHINES (SVM)

Diagram

Consider the diagram in which there are two different categories that are classified using a decision boundary or hyperplane:



SUPPORT VECTOR MACHINES (SVM)

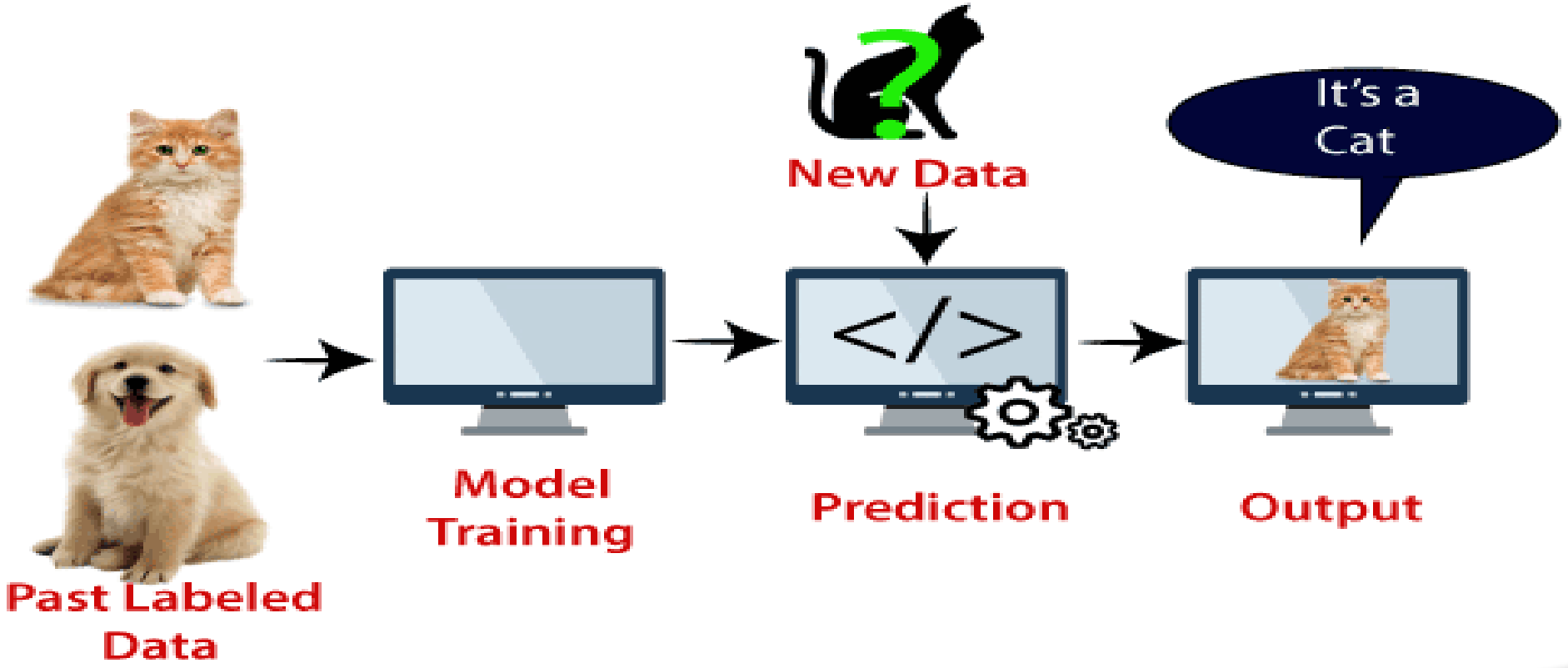
Example

Suppose we see a strange cat that also has some features of dogs, so if we want a model that can accurately identify whether it is a cat or dog, so such a model can be created by using the SVM algorithm. We will first train our model with lots of images of cats and dogs so that it can learn about different features of cats and dogs, and then we test it with this strange creature. So as support vector creates a decision boundary between these two data (cat and dog) and choose extreme cases (support vectors), it will see the extreme case of cat and dog. On the basis of the support vectors, it will classify it as a cat. Consider the below diagram:

SUPPORT VECTOR MACHINES (SVM)

Example

Consider the below diagram:

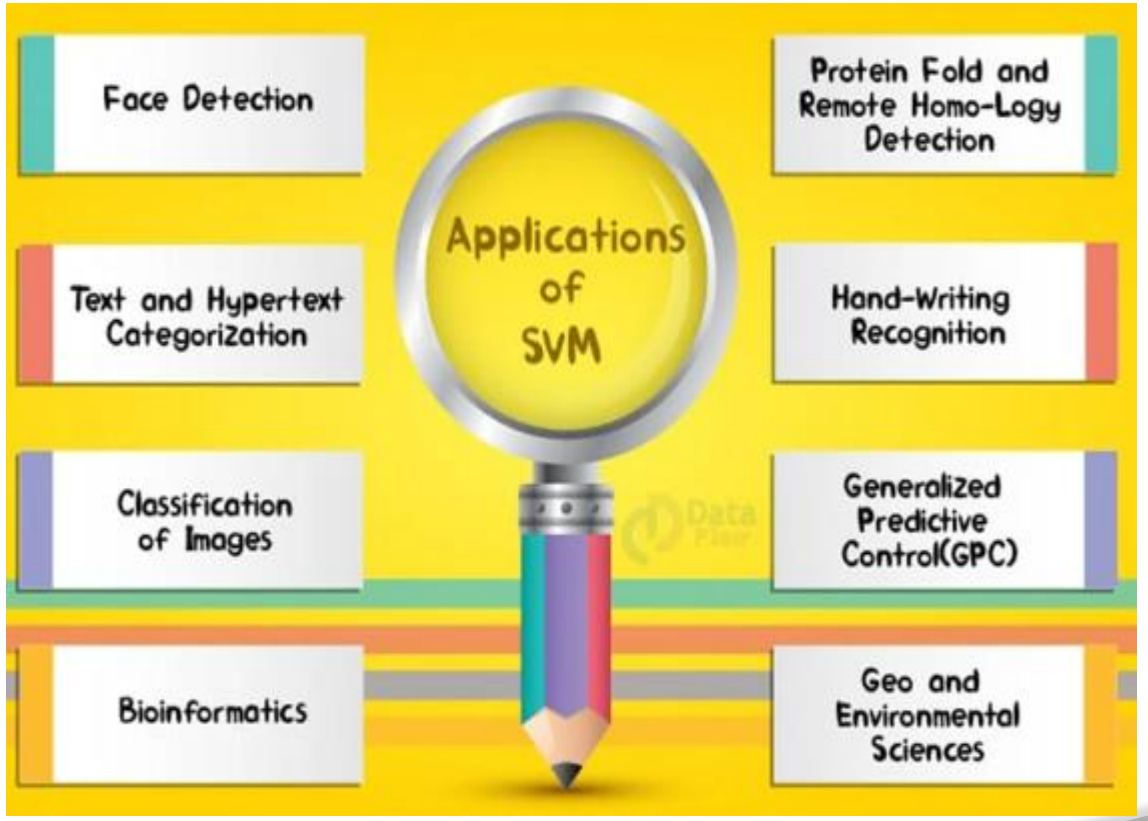


SUPPORT VECTOR MACHINES (SVM)

Applications

Some common applications of SVM are-

- *Face Detection*
- *Text and Hypertext Categorization*
- *Classification of Images*
- *Bioinformatics*
- *Protein fold and remote homology detection*
- *Handwriting recognition*
- *Generalized Predictive Control*



SUPPORT VECTOR MACHINES (SVM)

Advantages of SVM

- *SVM offers **best classification performance (accuracy) on the training data.***
- *SVM renders **more efficiency for correct classification of the future data**.*
- *It **does not make any strong assumptions on data.***
- *SVM **robust to outliers.***
- *SVM **are conceptually easy to understand.***
- *The main **strength of SVM is that they work well even when the number of SVM features is much larger than the number of instances.***

SUPPORT VECTOR MACHINES (SVM)

Disadvantages of SVM

- *SVM is **not suitable for large data sets.***
- *It **does not perform very well when the data has more noise** i.e. target classes are overlapping.*
- *In cases where the number of features for each data point exceeds the number of training data samples, the SVM will underperform.*

LINEAR AND NON-LINEAR SVM

Presented by:

Ms. Bidyutlata Sahoo

Assistant Professor

CSE (AI & ML)

OBJECTIVES

The objective of SVM algorithm is ***to find a hyperplane in an N-dimensional space that distinctly classifies the data points***.

The dimension of the hyperplane depends upon the number of features.

OUTCOMES

SVM or Support Vector Machine is ***a linear model for classification and regression problems.*** It ***can solve linear and non-linear problems*** and work well for many practical problems.

LINEAR AND NON-LINEAR SVM

Types of SVMs

SVM can be of ***two types:***

- ***Linear SVM***
- ***Non-Linear SVM***

LINEAR AND NON-LINEAR SVM

Types of SVMs

Linear SVM:

Linear SVM is used for linearly separable data, which means ***if a dataset can be classified into two classes by using a single straight line***, then such data is termed as linearly separable data, and classifier is used called as ***Linear SVM classifier***. (i.e, the training data/classifiers are separated by a hyperplane.)

LINEAR AND NON-LINEAR SVM

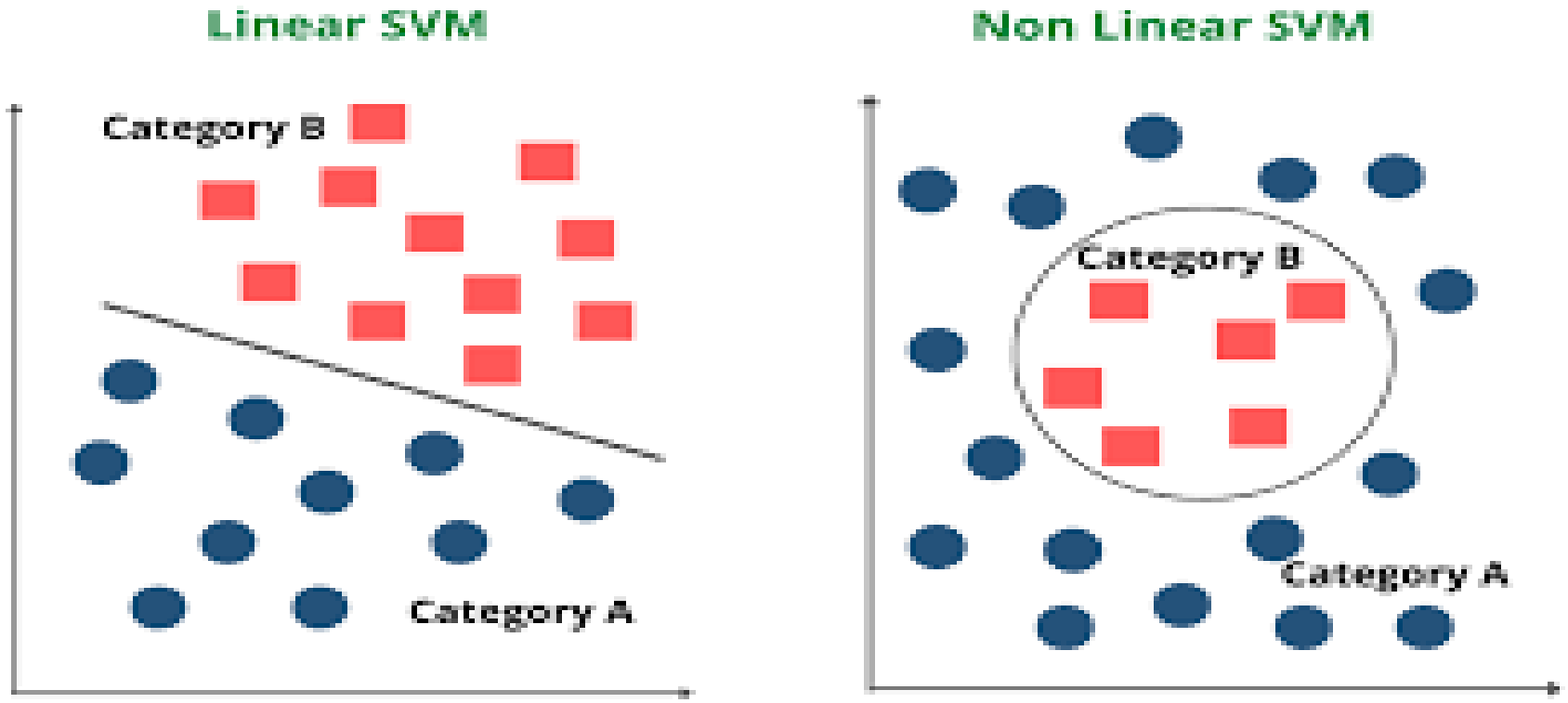
Types of SVMs

Non-linear SVM*:*

Non-Linear SVM is used for non-linearly separated data, which means ***if a dataset cannot be classified by using a straight line***, then such data is termed as non-linear data and classifier used is called as ***Non-linear SVM classifier***. (i.e, it is not possible to separate the training data using a hyperplane)

LINEAR AND NON-LINEAR SVM

Types of SVMs



LINEAR AND NON-LINEAR SVM

Hyperplane in SVM

- There can be multiple lines/decision boundaries to segregate the classes in n-dimensional space, but we need to find out the best decision boundary that helps to classify the data points. ***This best boundary is known as the hyperplane of SVM.***
- The dimensions of the ***hyperplane depend on the features present in the dataset***, which ***means if there are 2 features (as shown in image), then hyperplane will be a straight line. And if there are 3 features, then hyperplane will be a 2-dimension plane.***

LINEAR AND NON-LINEAR SVM

Hyperplane in SVM

[NOTE: We always create a hyperplane that has a maximum margin, which means the maximum distance between the data points.]

LINEAR AND NON-LINEAR SVM

Support Vectors in SVM

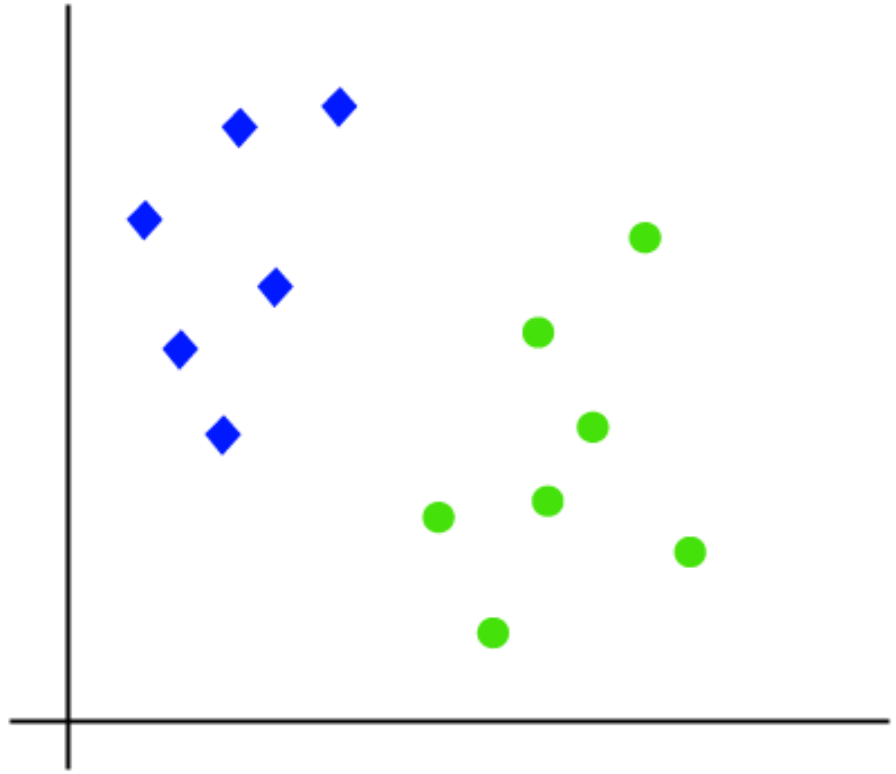
The data points or vectors that are the closest to the hyperplane and which affect the position of the hyperplane are termed as Support Vector. Since these vectors support the hyperplane, hence called a Support vector.

LINEAR AND NON-LINEAR SVM

Working of Linear SVM with Example

The working of the SVM algorithm can be understood by using an example.

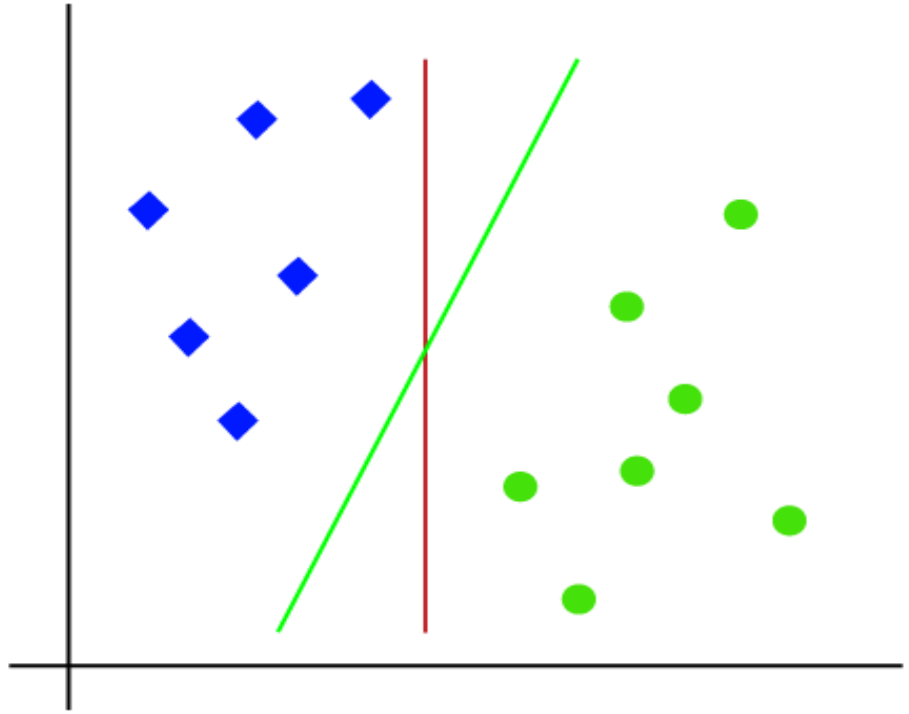
Suppose we have a dataset that has two tags (green and blue), and the dataset has two features x1 and x2. We want a classifier that can classify the pair (x1, x2) of coordinates in either green or blue. Consider the given image:



LINEAR AND NON-LINEAR SVM

Working of Linear SVM with Example

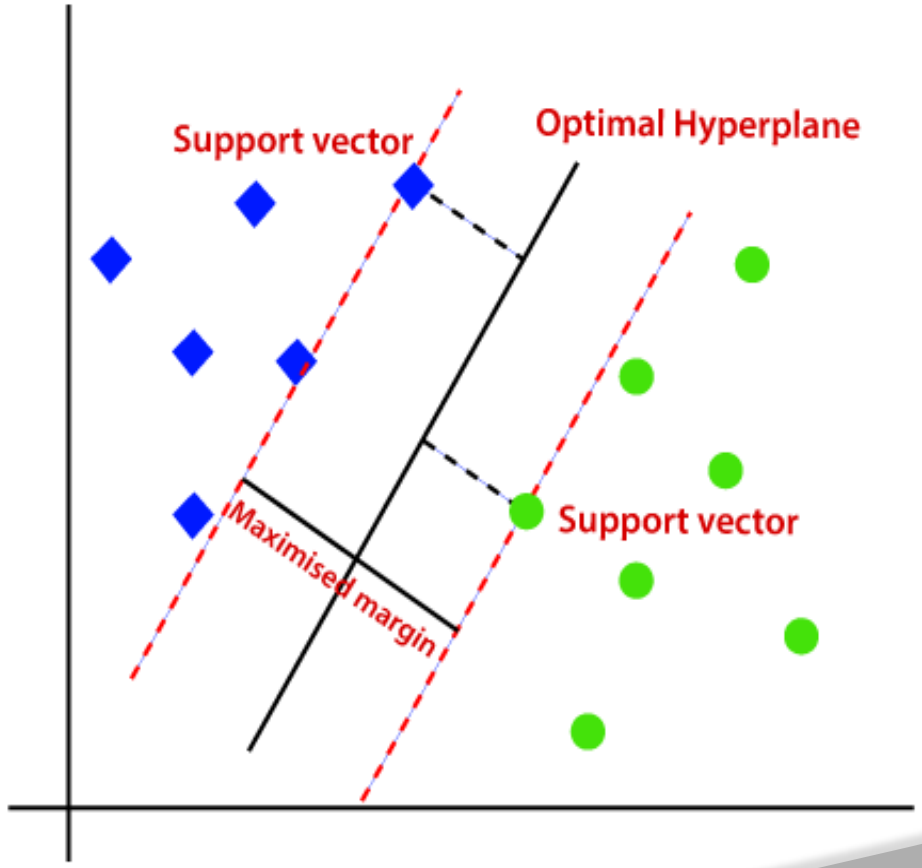
So as it is a 2-d space, by just using a straight line, we can easily separate these two classes. But there can be multiple lines that can separate these classes. Consider the given image:



LINEAR AND NON-LINEAR SVM

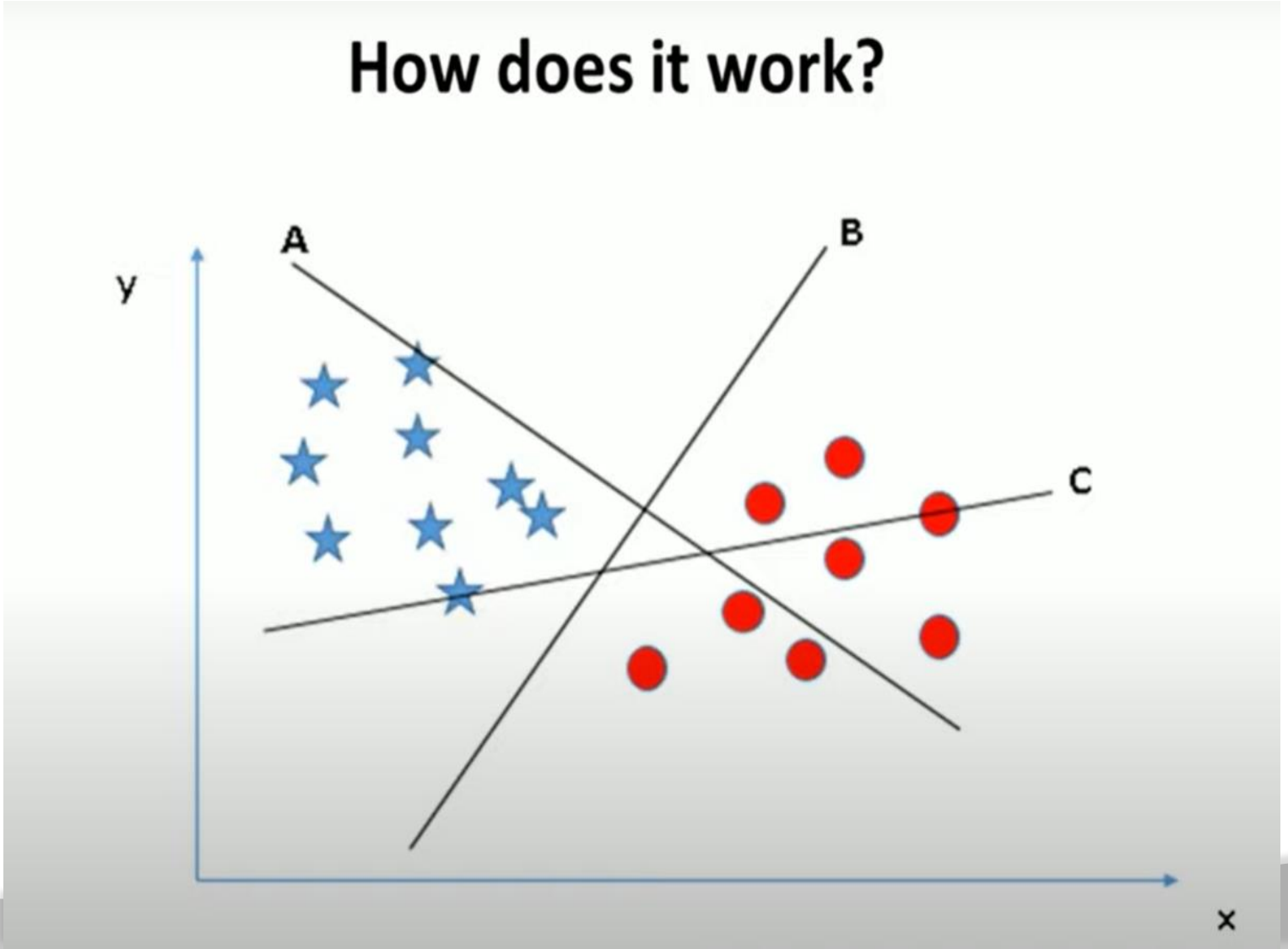
Working of Linear SVM with Example

Hence, the ***SVM algorithm helps to find the best line or decision boundary****; this best boundary or region is called as a* ***hyperplane****. SVM algorithm finds the closest point of the lines from both the classes. These points are called support vectors. The distance between the vectors and the hyperplane is called as* ***margin****. And the goal of SVM is to maximize this margin.* ***The hyperplane with maximum margin is called the optimal hyperplane.***



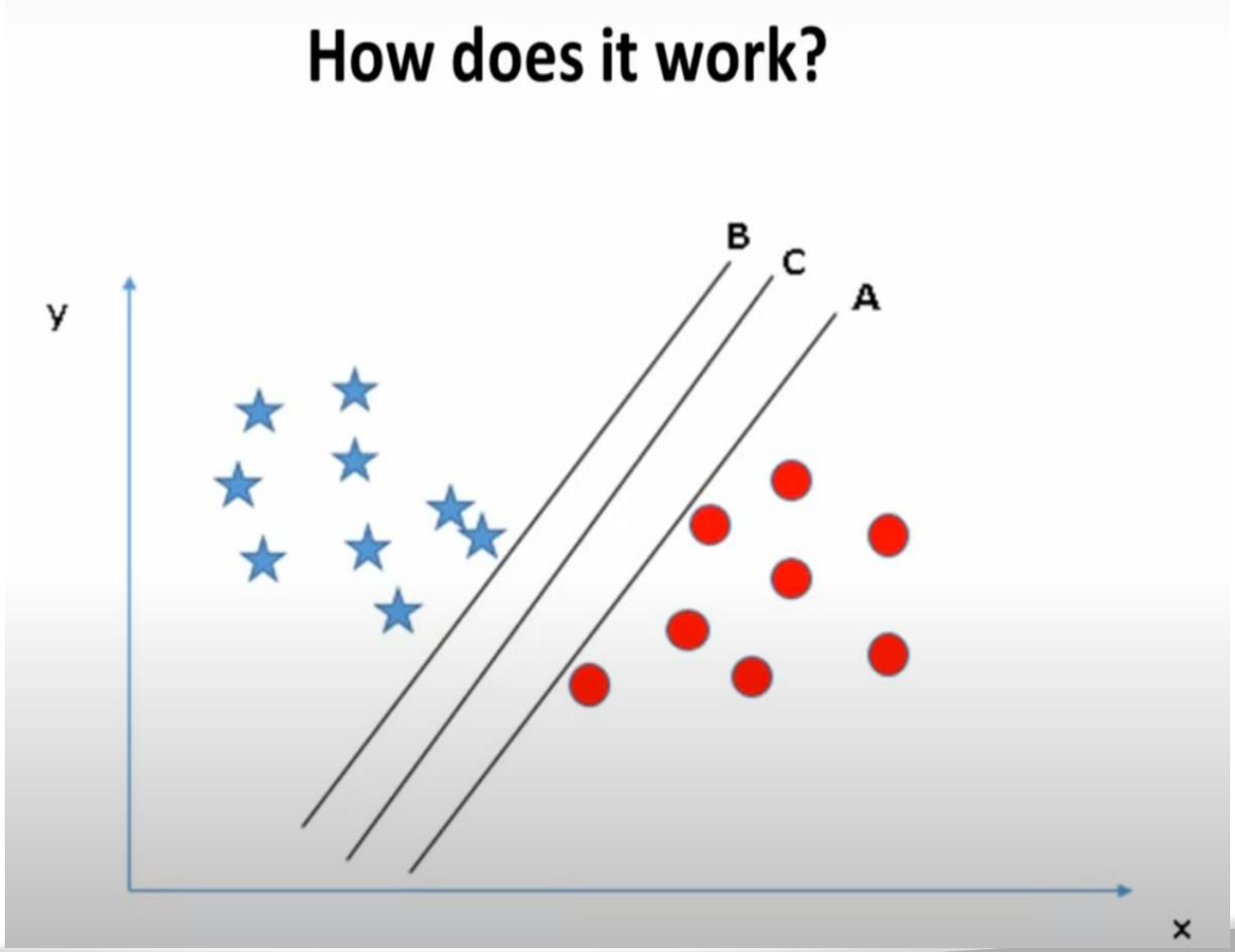
LINEAR AND NON-LINEAR SVM

Working of Linear SVM with more Examples – Scenario-1



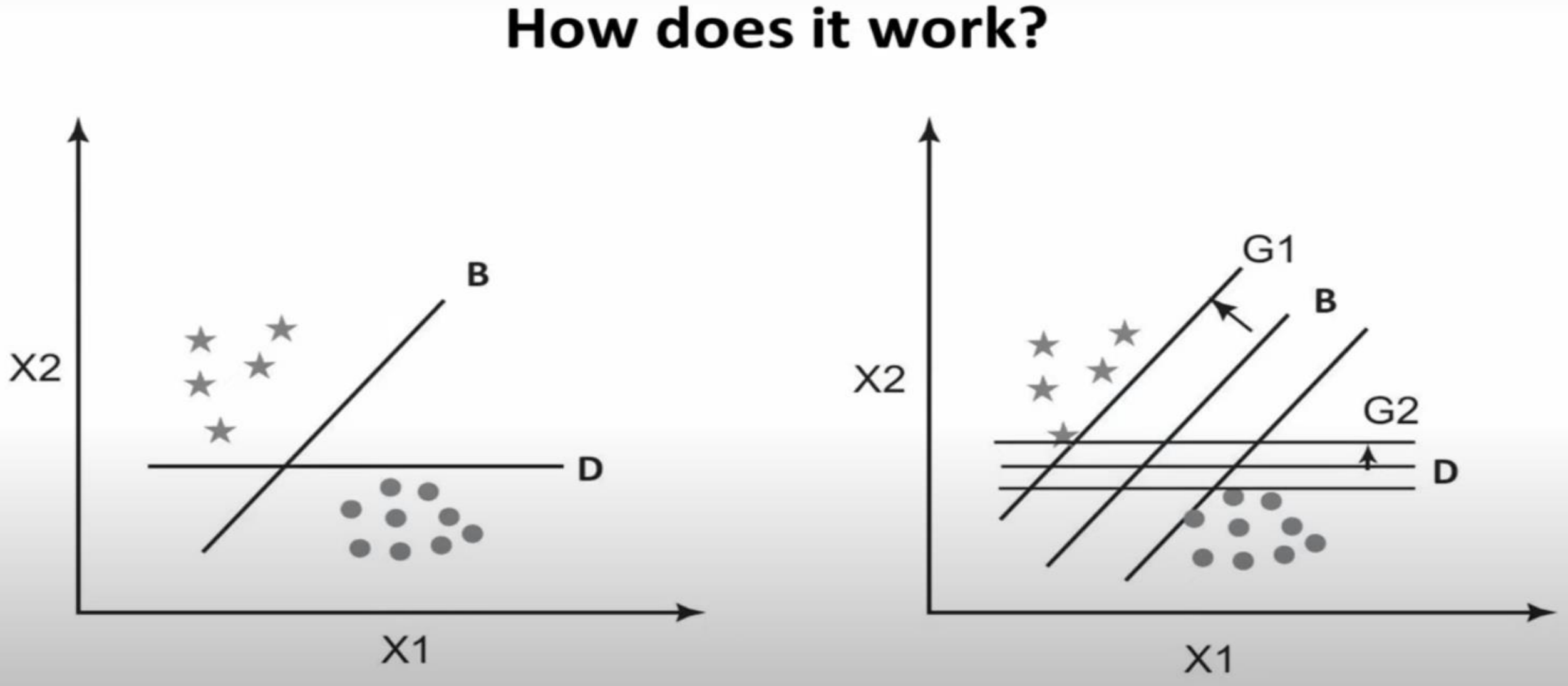
LINEAR AND NON-LINEAR SVM

Working of Linear SVM with more Examples – Scenario-2



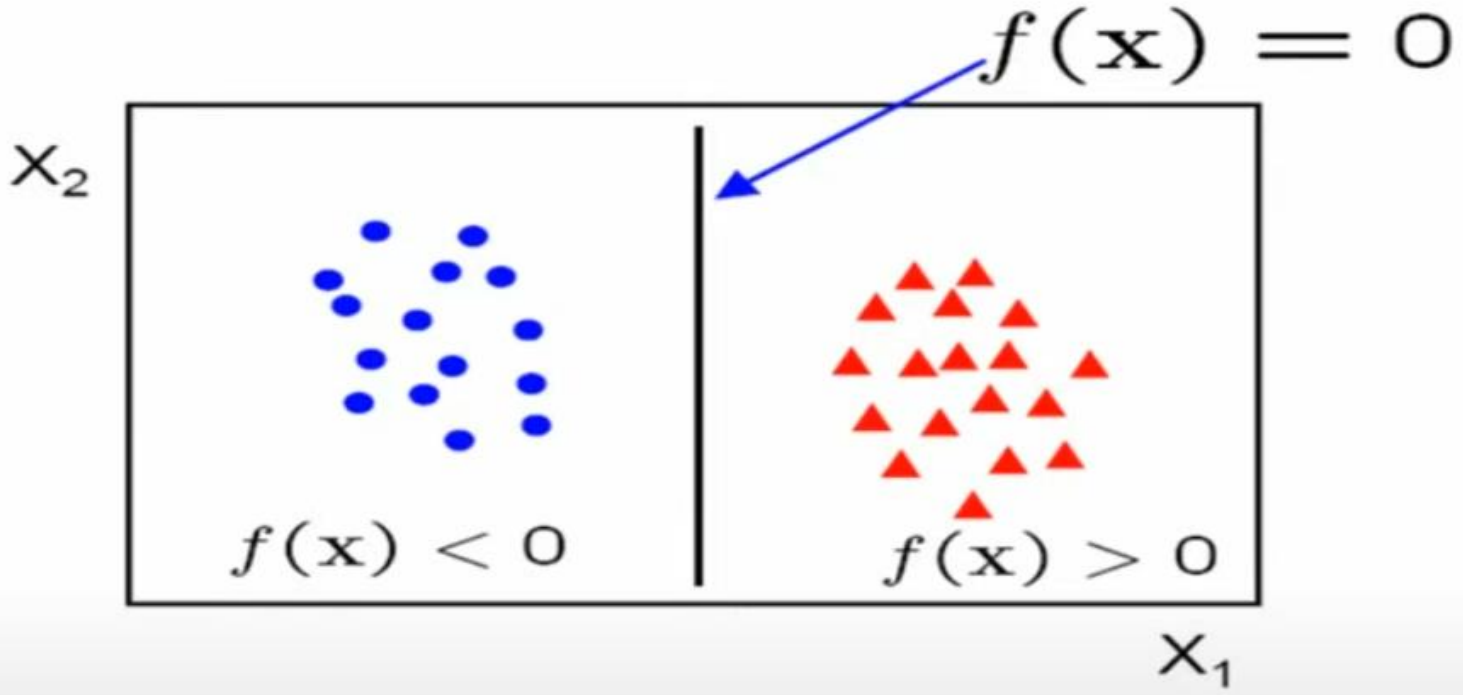
LINEAR AND NON-LINEAR SVM

Working of Linear SVM with more Examples – Scenario-3



LINEAR AND NON-LINEAR SVM

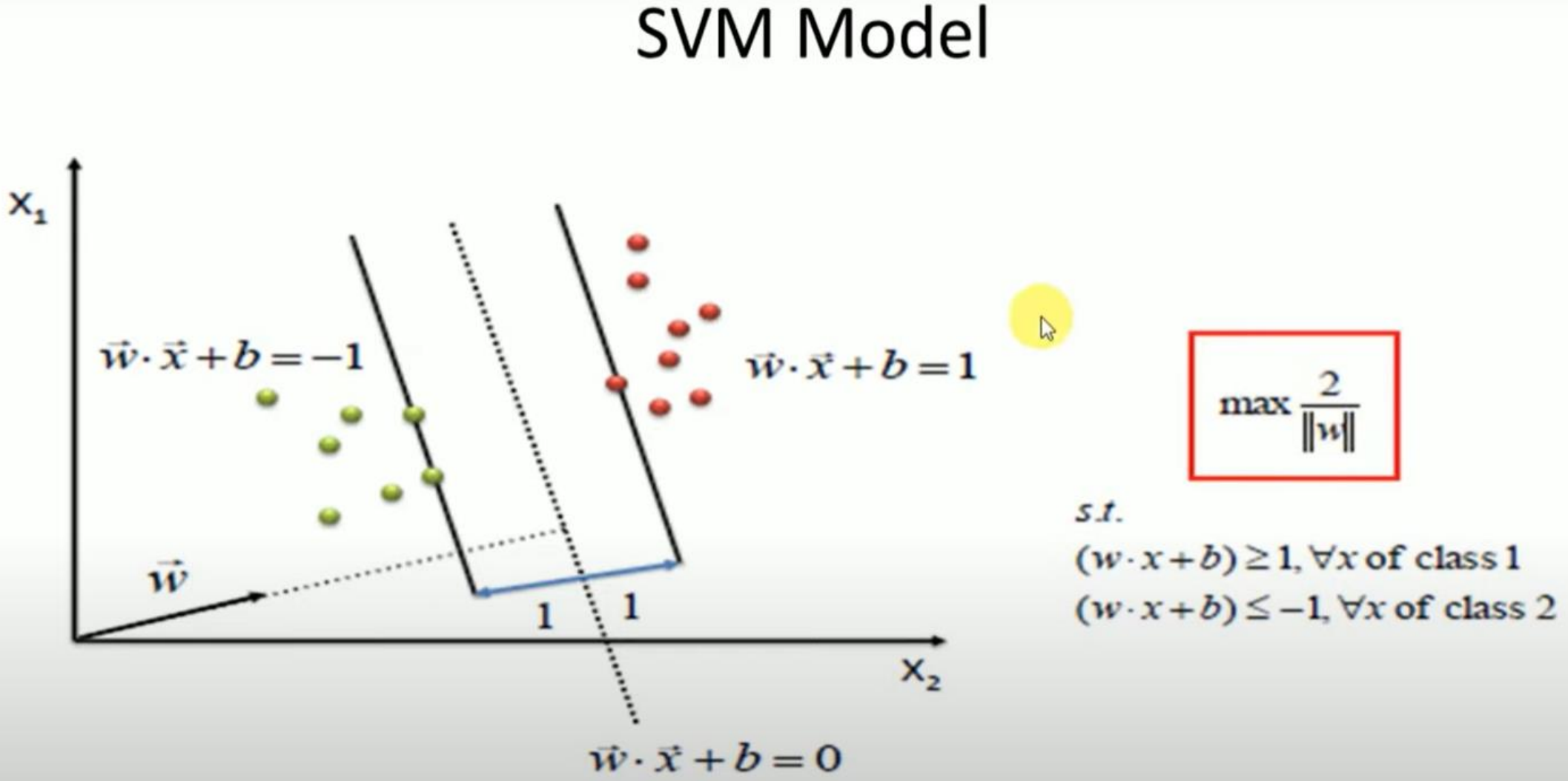
Working of Linear SVM: (Mathematical Representation)



- f(x) = W.X + b
- W is the normal to the line, X is input vector and b the bias
- W is known as the weight vector

LINEAR AND NON-LINEAR SVM

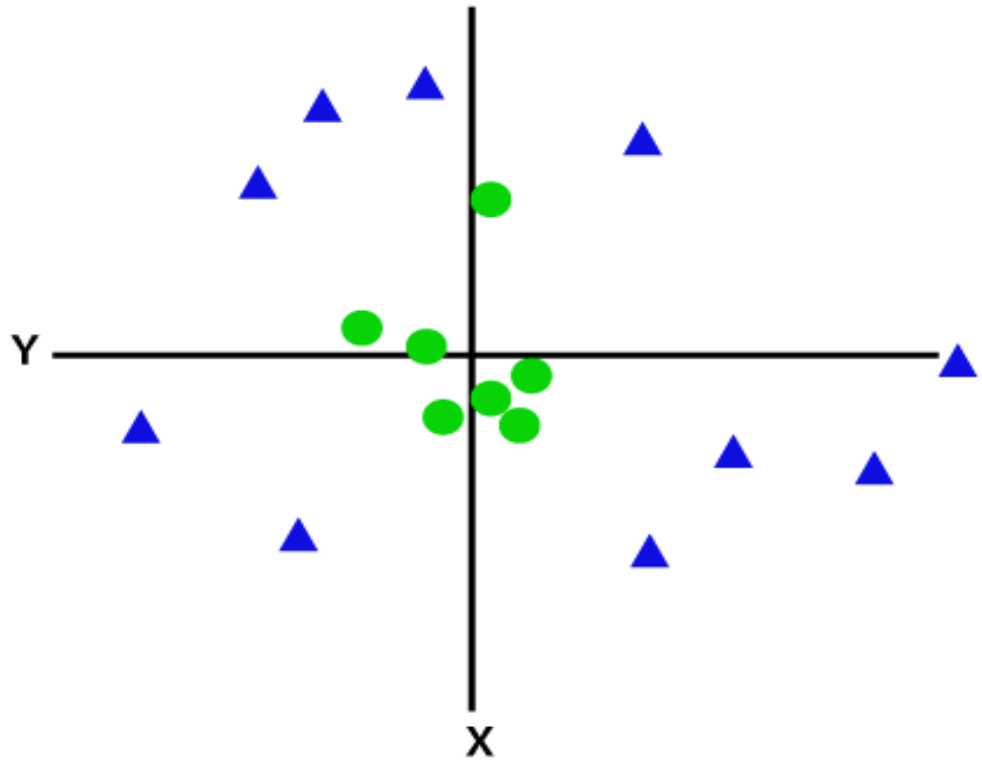
Working of Linear SVM: (mathematical representation):



LINEAR AND NON-LINEAR SVM

Working of Non-Linear SVM with Example

If data is linearly arranged, then we can separate it by using a straight line, but ***for non-linear data, we cannot draw a single straight line****. Consider the given image:*

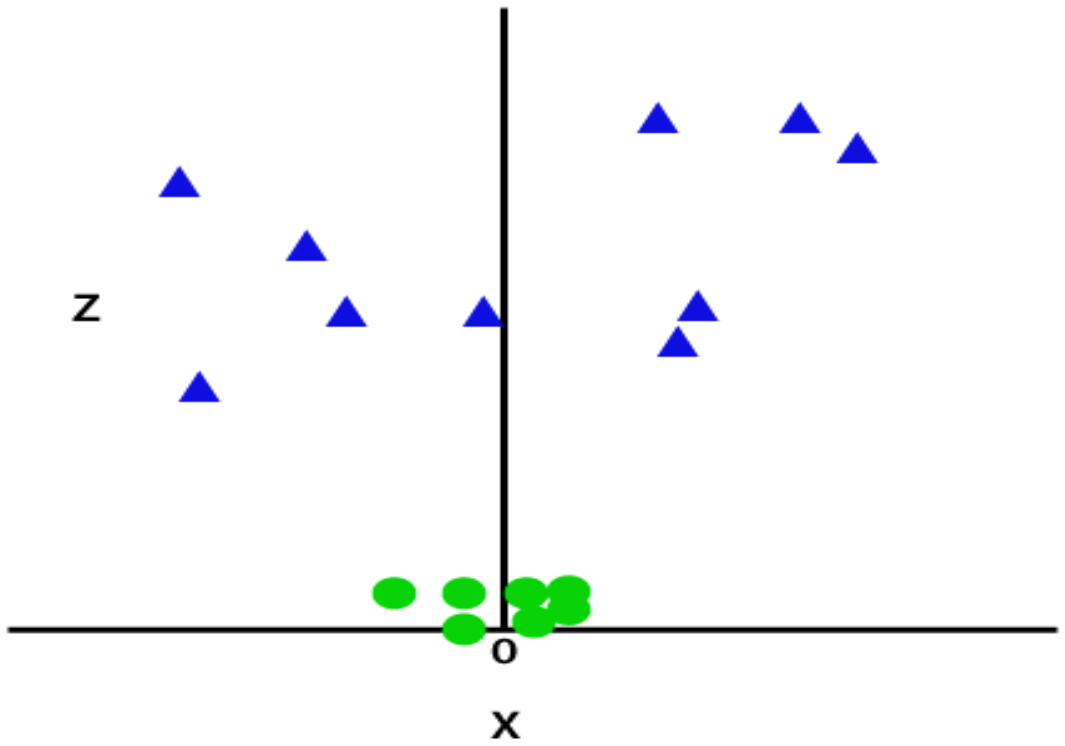


LINEAR AND NON-LINEAR SVM

Working of Non-Linear SVM with Example

So, to separate these data points, ***we need to add one more dimension****. For linear data, we have used two dimensions x and y, so* ***for non-linear data, we will add a third-dimension z.*** *It can be calculated as:*

z=x2 +y2



LINEAR AND NON-LINEAR SVM

Working of Non-Linear SVM with Example

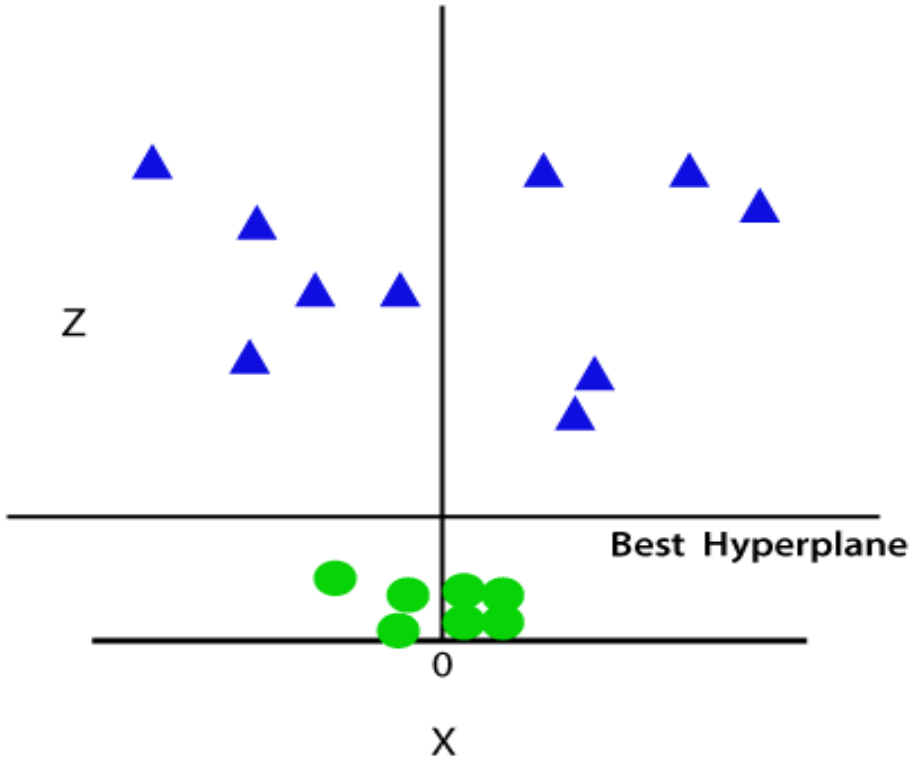
In the above plot, points to consider:

- ***All values of z would be positive always*** because z is the squared sum of both x and y.
- In the original plot, green circles appear close to the origin of x and y axes, leading to lower value of z and blue triangles relatively away from the origin result to higher value of z. In SVM, it is easy to have a linear hyperplane between these two classes. But another question arises that, should we need to add this feature manually to have a hyperplane.
- No, SVM has a technique called the ***KERNEL TRICK. (SVM KERNEL).***
- ***SVM Kernel are the functions which takes low dimensional input space and transform it to a higher dimensional space*** i.e. it converts non separable problem to separable problem, these functions are called ***kernels.*** It is mostly useful in non-linear separation problem.

LINEAR AND NON-LINEAR SVM

Working of Non-Linear SVM with Example

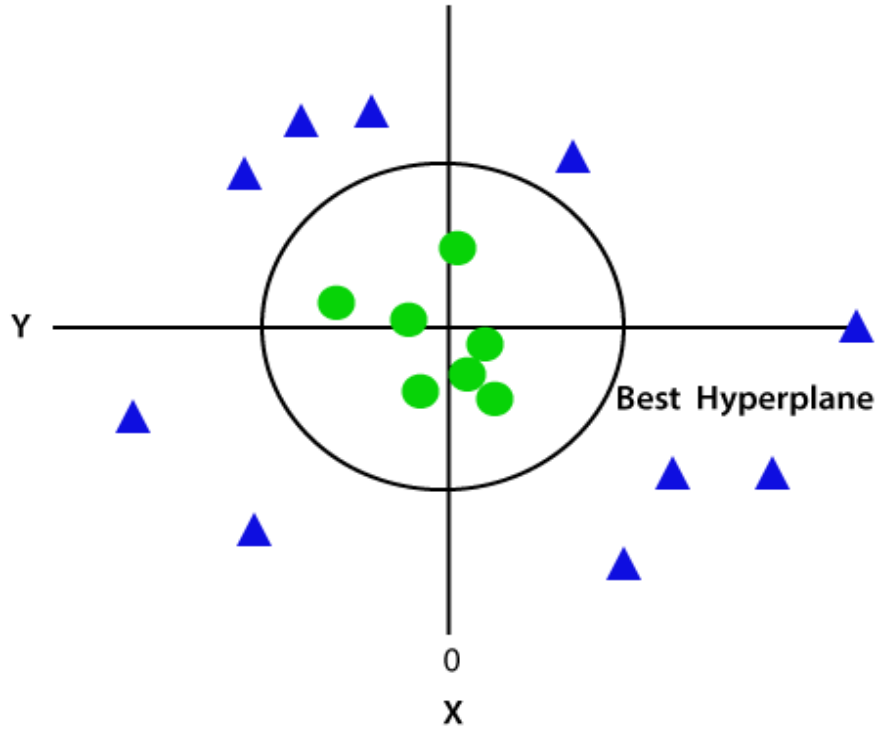
So now, SVM will divide the datasets into classes in the following way. Consider the given image:



LINEAR AND NON-LINEAR SVM

Working of Non-Linear SVM with Example

Since we are in 3-d Space, hence it is looking like a plane parallel to the x-axis. If we convert it in 2d space with z=1, then it will become as: (i.e. when we look at the hyperplane in original input space it looks like a circle)



Hence, we get a circumference of radius 1 in case of non-linear data.

LINEAR AND NON-LINEAR SVM

Data Science Libraries to implement SVM

- SciKit Learn
- PyML
- SVM^{struct} Python
- LIBSVM

K-NEAREST NEIGHBORS (KNN)

KNN is a reasonably simple classification technique that identifies the class in which a sample belongs by measuring its similarity with other nearby points.

It is a powerful ***technique for identifying the class of an unknown sample point.***

K-nearest neighbors (KNN) algorithm is a ***type of supervised ML algorithm*** which can be ***used for both classification as well as regression predictive problems. However, it is mainly used for classification predictive problems in industry***.

The following **two properties** would define KNN well −

- **Lazy learning algorithm** − KNN is a lazy learning algorithm because it does ***not have a specialized training phase and uses all the data for training while classification.***
- **Non-parametric learning algorithm** − KNN is also a non-parametric learning algorithm because it ***doesn't assume anything*** about the underlying data.

KNN ALGORITHM

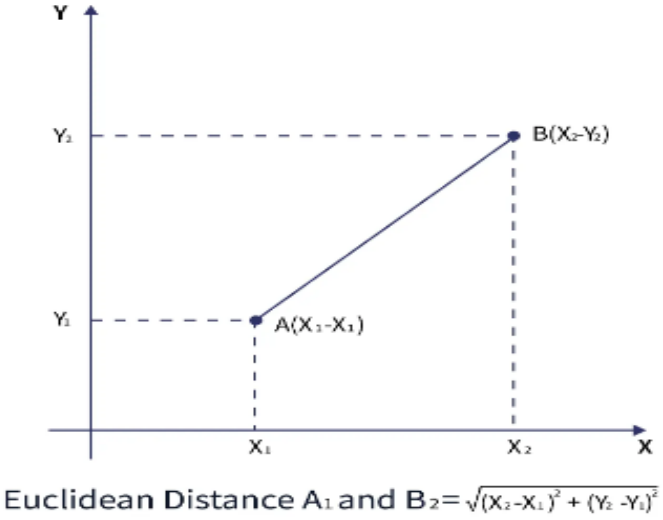
The steps for the KNN Algorithm in Machine Learning are as follows:

- **Step - 1 :**
  Select the number K of the neighbors
- **Step - 2 :**
  Calculate the Euclidean distance of each point from the target point.
- **Step - 3 :**
  Take the K nearest neighbors per the calculated Euclidean distance.
- **Step - 4 :**
  Among these k neighbors, count the number of the data points in each category.
- **Step - 5 :**
  Assign the new data points to that category for which the number of neighbors is maximum.

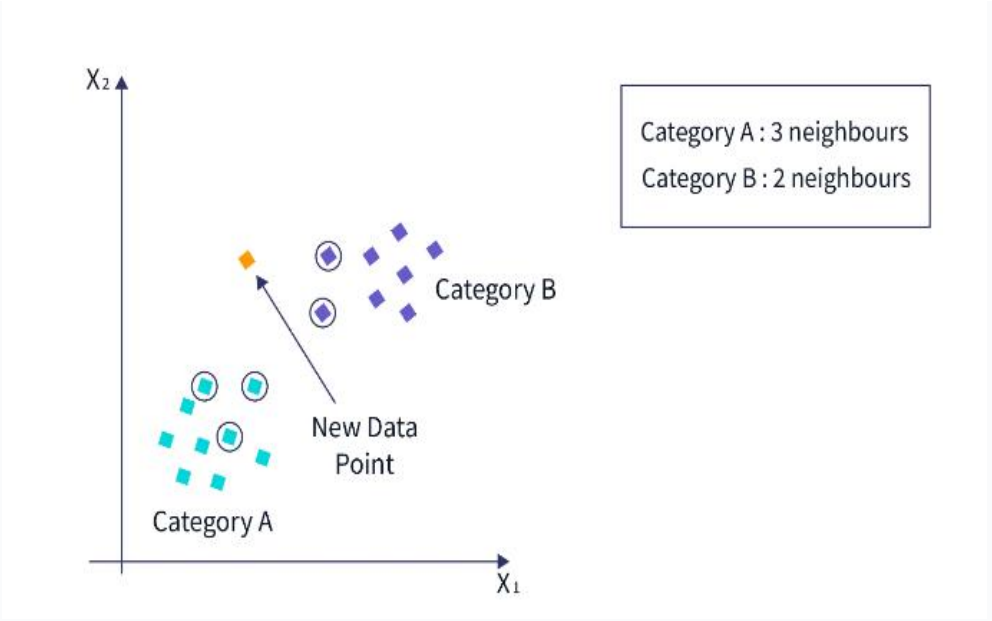
EXAMPLE

Suppose we have a new data point and need to put it in the required category. Consider the below image :

- First, we will choose the number of neighbors as k=5.
- Next, we will calculate the Euclidean distance between the data points. The Euclidean distance is the distance between two points, which we have already studied in geometry. It can be calculated as :



- By calculating the Euclidean distance, we got the nearest neighbors, as three nearest neighbors in category A and two nearest neighbors in category B. Consider the below image :



New data point will fall in category A.

HOW TO DETERMINE THE K VALUE?

- The value of k is very crucial in the KNN algorithm to define the number of neighbors in the algorithm.
- The value of k in the k-nearest neighbors (k-NN) algorithm should be chosen based on the input data. If the input data has more outliers or noise, a higher value of k would be better.
- It is ***recommended to choose an odd value for k*** to avoid ties in classification.
- ***Cross-validation*** methods can help in selecting the best k value for the given dataset.

DISTANCE METRICS USED IN KNN

Euclidean Distance $d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$

Manhattan Distance $d(x, y) = \sum_{i=1}^{n} |x_i - y_i|$

Minkowski Distance $d(x, y) = \left(\sum_{i=1}^{n} (x_i - y_i)^p\right)^{\frac{1}{p}}$

APPLICATIONS OF KNN

- KNN can be ***used in plagiarism checks*** to check if two documents are semantically identical.
- ML in ***healthcare uses KNN to diagnose diseases based on subtle symptoms***.
- Netflix uses KNN in its ***recommendation systems.***
- KNN is used in ***face recognition software***.

- ***Political scientists and psychologists use*** KNN to classify people based on their behavior and response to stimuli.
- ***Banking System***

  KNN can be used in banking system to predict weather an individual is fit for loan approval? Does that individual have the characteristics similar to the defaulters one?

- ***Calculating Credit Ratings***

  KNN algorithms can be used to find an individual's credit rating by comparing with the persons having similar traits.

- ***Politics***

  With the help of KNN algorithms, we can classify a potential voter into various classes like "Will Vote", "Will not Vote", "Will Vote to Party 'Congress', "Will Vote to Party 'BJP'.

ADVANTAGES OF KNN

- It is very ***simple algorithm to understand and interpret***.
- It is very ***useful for nonlinear data because there is no assumption*** about data in this algorithm.
- It is a ***versatile algorithm*** as we can use it for classification as well as regression.
- It has ***relatively high accuracy*** but there are much better supervised learning models than KNN.

DISADVANTAGES OF KNN

- It is ***computationally a bit expensive algorithm*** because it stores all the training data.
- ***High memory storage required*** as compared to other supervised learning algorithms.
- ***Prediction is slow*** in case of big N.
- It is very ***sensitive to the scale of data as well as irrelevant features***.

PROBLEM: (**Prediction of Sugar of Diabetic Patient given BMI and Age)**

Apply K nearest neighbor classifier to predict the diabetic patient with the given features BMI, Age. If the training examples are,

BMI	Age	Sugar
33.6	50	1
26.6	30	O
23.4	40	O
43.1	67	O
35.3	23	1
35.9	67	1
36.7	45	1
25.7	46	O
23.3	29	O
31	56	1

Assume K=3,

Test Example BMI=43.6, Age=40, Sugar=?

Solution:

The given training dataset has 10 instances with two features BMI (Body Mass Index) and Age. Sugar is the target label. The target label has two possibilities 0 and 1. 0 means the diabetic patient has no sugar and 1 means the diabetic patient has sugar.

Given the dataset and new test instance, we need to find the distance from the new test instance to every training example. Here we use the Euclidean distance formula to find the distance.

$$Distance = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

In the next table, you can see the calculated distance from text example to training instances.

BMI	Age	Sugar	Formula	Distance
33.6	50	1	√((43.6-33.6)^2+(40-50)^2)	14.14
26.6	30	O	√((43.6-26.6)^2+(40-30)^2)	19.72
23.4	40	O	√((43.6-23.4)^2+(40-40)^2)	20.20
43.1	67	O	√((43.6-43.1)^2+(40-67)^2)	27.00
35.3	23	1	√((43.6-35.3)^2+(40-23)^2)	18.92
35.9	67	1	√((43.6-35.9)^2+(40-67)^2)	28.08
36.7	45	1	√((43.6-36.7)^2+(40-45)^2)	8.52
25.7	46	O	√((43.6-25.7)^2+(40-46)^2)	18.88
23.3	29	O	√((43.6-23.3)^2+(40-29)^2)	23.09
31	56	1	√((43.6-31)^2+(40-56)^2)	20.37

Once you calculate the distance, the next step is to find the nearest neighbors based on the value of k. In this case, the value of k is 3. Hence, we need to find 3 nearest neighbors.

BMI	Age	Sugar	Distance	Rank
33.6	50	1	14.14	2
26.6	30	O	19.72	
23.4	40	O	20.20	
43.1	67	O	27.00	
35.3	23	1	18.92	
35.9	67	1	28.08	
36.7	45	1	8.52	1
25.7	46	O	18.88	3
23.3	29	O	23.09	

31	56	1	20.37	

Now, we need to **apply the majority voting technique** to decide the resulting label for the new example. Here the 1st and 2nd nearest neighbors have target label 1 and the 3rd nearest neighbor has target label 0. Target label 1 has the majority. Hence the new example is classified as 1, That is the diabetic patient has **Sugar**.

Test Example BMI=43.6, Age=40, Sugar=1