

Algorithm

1. Initialize:

- Start with an initial feature subset S (can be empty, full, or a predefined set).

2. Step 1: Forward Selection (Plus-L)

- Add L features to S from the remaining pool of features.
- The L features are chosen to **maximize the performance improvement** when added to S .

3. Step 2: Backward Elimination (Minus-R)

- From the updated subset S , remove R features that contribute the least (or degrade performance the most).
- Update S : Remove R features to minimize performance loss.

4. Step 3: Repeat

- Continue alternating **Plus-L** and **Minus-R** steps until a stopping criterion is met (e.g., subset size, no significant performance improvement, or all features have been evaluated).

5. Output:

- The resulting set S is the selected feature subset.



Branch & Bound Algorithm

- Branching is the process of generating subproblems.
- Bounding refers to ignoring partial solutions that cannot be better than the current best solution.
- It is a search procedure to find the optimal solution.
- It eliminates those parts of a search space which does not contain better solution(Pruning).
- In this method we basically extend the cheapest partial path.

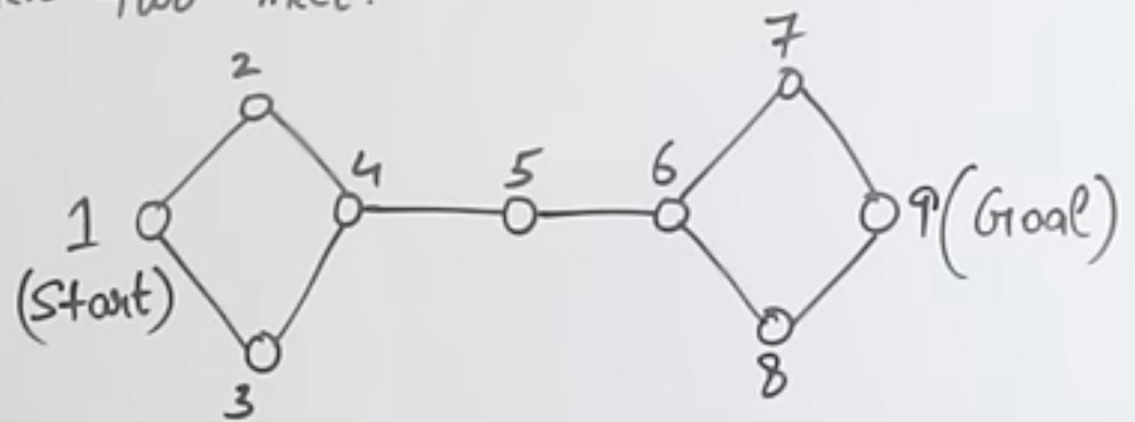
Bidirectional Search

→ Two simultaneous search from an initial node to goal and backward from goal to initial, stopping when two meet.

→ Time Complexity $\sim (b^{d/2})$

Complete in breadth first search

Not in depth first search



Step-by-Step Algorithm

1. Initialize:

- Start with the **full feature set** $S = F$, where F contains all the available features.

2. Step 1: Backward Elimination

- Identify the **worst feature** f^- (from the current set S) to remove.
- f^- is the feature that **minimizes the decrease in performance** when removed from S .
- Update S : $S = S \setminus \{f^-\}$.

3. Step 2: Forward Addition (Conditional Reintroduction)

- After removing f^- , check if **adding back any previously removed feature** improves performance.
- If such a feature f^+ exists, reintroduce it: $S = S \cup \{f^+\}$.

4. Step 3: Repeat

- Repeat **Steps 1 and 2** until a stopping criterion is met (e.g., desired subset size, no significant performance improvement, or all features have been evaluated).

5. Output:

- The resulting set S is the selected feature subset.

Step-by-Step Algorithm

1. Initialize:

- Start with an **empty feature set**: $S = \emptyset$.

2. Step 1: Forward Addition

- Identify the **best feature** f^+ (from the remaining features) to add to the current set S .
- f^+ is chosen because it **maximizes performance** (e.g., accuracy or any evaluation metric) when added to S .
- Update S : $S = S \cup \{f^+\}$.

3. Step 2: Backward Elimination (Conditional Removal)

- After adding f^+ , check if **removing any single feature** from S improves the performance.
- If such a feature f^- exists, remove it: $S = S \setminus \{f^-\}$.

4. Step 3: Repeat

- Repeat **Steps 1 and 2** until a stopping criterion is met (e.g., desired subset size, no significant performance improvement, or all features have been evaluated).

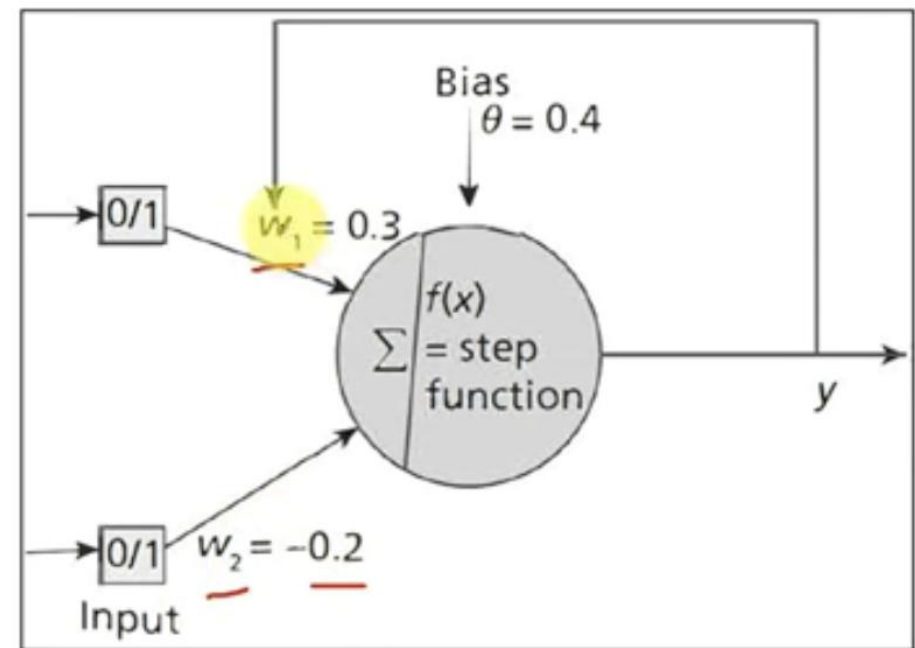
5. Output:

- The resulting set S is the selected feature subset.



Perceptron Algorithm Boolean AND Function

- Consider a perceptron to represent the Boolean function AND with the initial weights $w_1 = 0.3$, $w_2 = -0.2$, learning rate $\alpha = 0.2$ and bias $\theta = 0.4$.
- The activation function used here is the Step function $f(x)$ which gives the output value as binary, i.e., 0 or 1.
- If value of $f(x)$ is greater than or equal to 0, it outputs 1 or else it outputs 0.
- Design a perceptron that performs the Boolean function AND and update the weights until the Boolean function gives the desired output.



Perceptron Algorithm Boolean AND Function

Epoch 1

Epoch	x_1	x_2	Y_{des}	Y_{est}	Error	w_1	w_2	Status
1	0	0	0	Step $((0 \times 0.3 + 0 \times -0.2) - 0.4) = 0$	0	0.3	-0.2	No change
	0	1	0	Step $((0 \times 0.3 + 1 \times -0.2) - 0.4) = 0$	0	0.3	-0.2	No change
	1	0	0	Step $((1 \times 0.3 + 0 \times -0.2) - 0.4) = 0$	0	0.3	-0.2	No change
	1	1	1	Step $((1 \times 0.3 + 1 \times -0.2) - 0.4) = 0$	1	0.5	0	Change

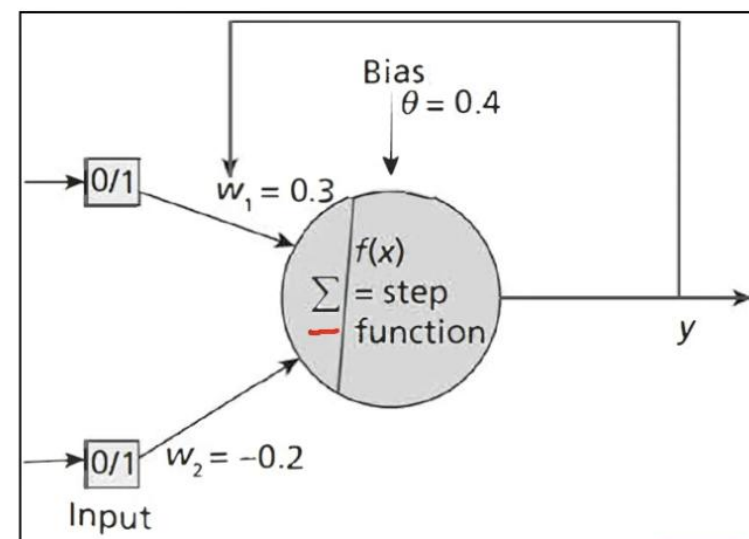
For input (1, 1) the weights are updated as follows:

$$\Delta w_1 = \alpha \times e(t) \times x_1 = 0.2 \times 1 \times 1 = 0.2$$

$$w_1 = w_1 + \Delta w_1 = 0.3 + \Delta w_1 = 0.3 + 0.2 = 0.5$$

$$\Delta w_2 = \alpha \times e(t) \times x_2 = 0.2 \times 1 \times 1 = 0.2$$

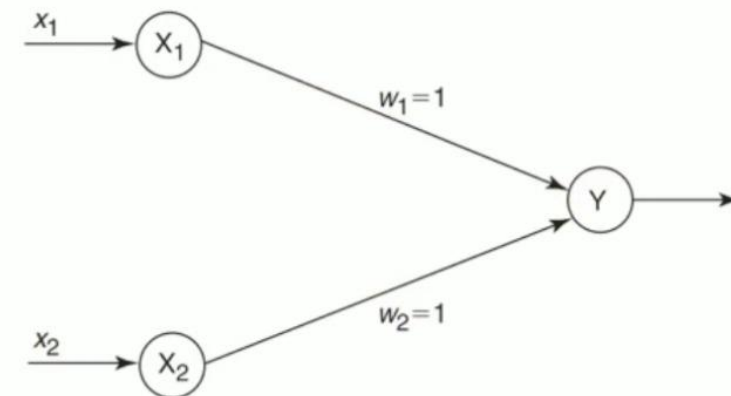
$$w_2 = w_2 + \Delta w_2 = -0.2 + \Delta w_2 = -0.2 + 0.2 = 0$$



Implement ANDNOT function using McCulloch-Pitts Neuron

- Consider the truth table for ANDNOT function
- The M-P neuron has no particular training algorithm
- In M-P neuron, only analysis is being performed.
- Hence, assume the weights be $w_1 = 1$ and $w_2 = 1$.

x_1	x_2	y
0	0	0
0	1	0
1	0	1
1	1	0



Implement XOR function using McCulloch–Pitts Neuron

Consider the truth table for XOR function

The M–P neuron has no particular training algorithm

x_1	x_2	y
0	0	0
0	1 ✓	1 ✓
1	0 ✓	1 ✓
1	1	0

In M-P neuron, only analysis is being performed.

XOR function cannot be represented by simple and single logic function; it is represented as

Implement XOR function using McCulloch–Pitts Neuron

$$y = x_1 \overline{x_2} + \overline{x_1} x_2$$

$$y = z_1 + z_2$$

where

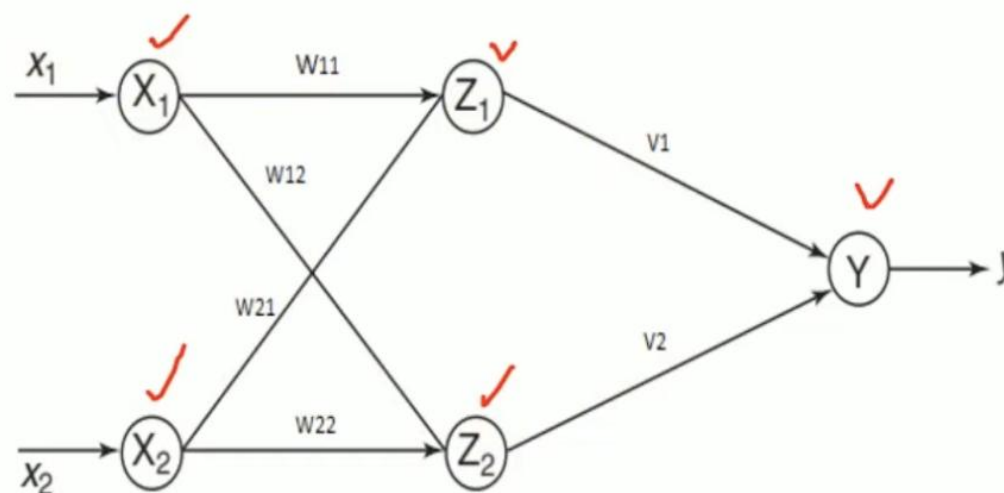
$$z_1 = x_1 \overline{x_2} \quad (\text{function 1})$$

$$z_2 = \overline{x_1} x_2 \quad (\text{function 2})$$

$$y = z_1 \text{ (OR) } z_2 \quad (\text{function 3})$$

A single-layer net is not sufficient to represent the XOR function. We need to add an intermediate layer is necessary.

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0



Perceptron Training Algorithm - Single Output Class

- Initialize the weights and the bias. Also initialize the learning rate α ($0 < \alpha \leq 1$). T
- Until the final stopping condition is false.
 - for each training pair indicated by s:t. ✓

- Set each input unit $i = 1$ to n : $x_i = s_i$
- Calculate the output of the network.

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

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

- Weight and bias adjustment:

If $y \neq t$, then

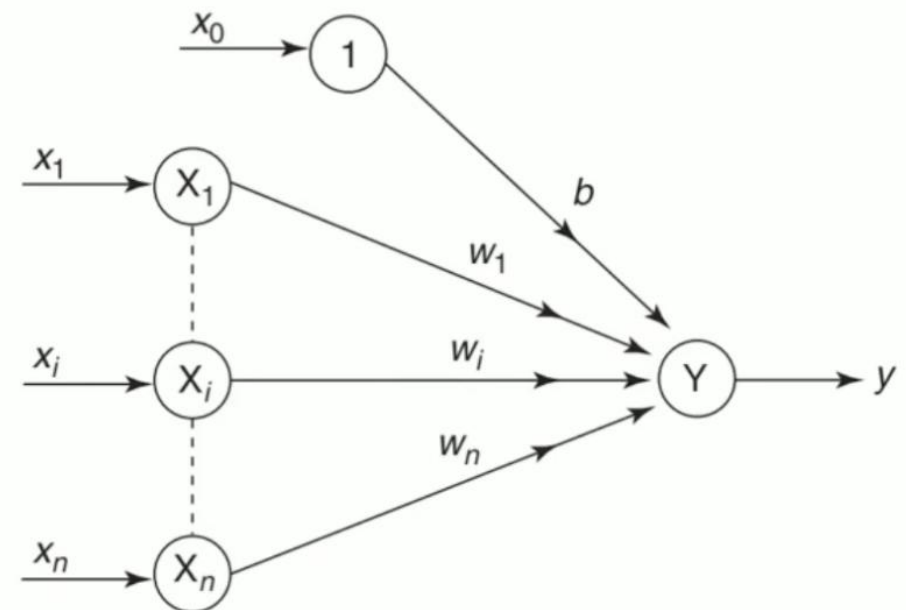
$$w_i(\text{new}) = w_i(\text{old}) + \alpha t x_i$$

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

else we have

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

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



Perceptron Training Algorithm - Multiple Output Class ⁱ

- Initialize the weights and the bias. Also initialize the learning rate α ($0 < \alpha \leq 1$).
- Until the final stopping condition is false.
 - for each training pair indicated by $s : t$.

- Set each input unit $i = 1$ to n : $x_i = s_i$
- Calculate the output of the network.

$$y_{inj} = b_j + \sum_{i=1}^n x_i w_{ij}$$

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

- Weight and bias adjustment:

If $t_j \neq y_j$, then

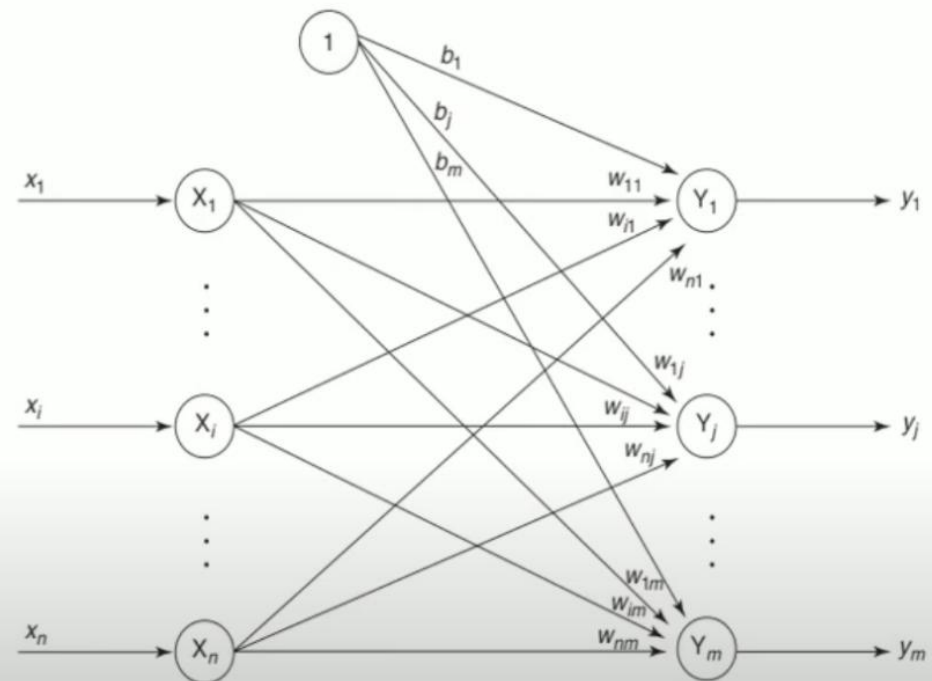
$$w_{ij}(\text{new}) = w_{ij}(\text{old}) + \alpha t_j x_i$$

$$b_j(\text{new}) = b_j(\text{old}) + \alpha t_j$$

else, we have

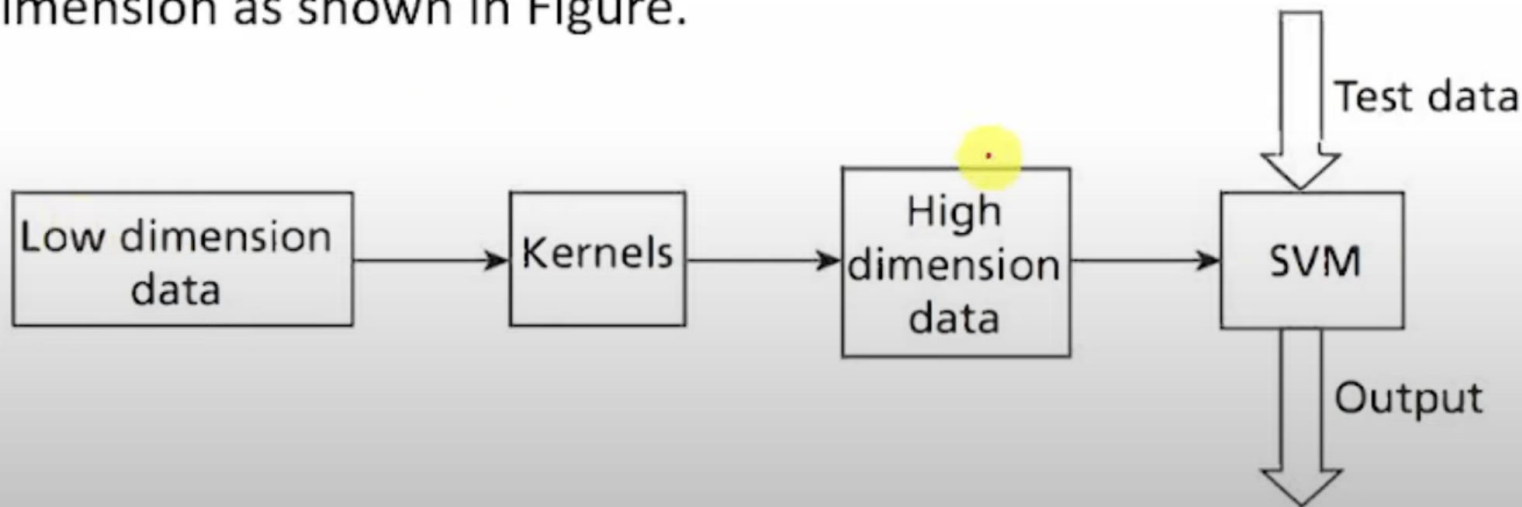
$$w_{ij}(\text{new}) = w_{ij}(\text{old})$$


$$b_j(\text{new}) = b_j(\text{old})$$



- **What is a Kernel?**

- Kernels are a set of functions used to transform data from lower dimension to higher dimension and to manipulate data using dot product at higher dimensions.
- The use of kernels is to apply transformation to data and perform classification at the higher dimension as shown in Figure.



- While mapping functions play an important role, there are many disadvantages as mapping involves more computations and learning costs. 
- Also, the disadvantages of transformations are that there is no generalized thumb rule available describing what transformations should be applied and if the data is large, mapping process takes huge amount of time.
- In real applications, there might be many features in the data and applying transformations that involve many polynomial combinations of these features will lead to extremely high and impractical computational costs.
- In this context, only kernels are useful.

Kernel Trick in Support Vector Machine

For example, one mapping function $\phi: \mathbb{R}^2 \rightarrow \mathbb{R}^3$ used to transform a 2D data to 3D data is given as follows:

$$\phi(x, y) = (x^2, \sqrt{2}xy, y^2)$$

Consider a point (2, 3) in 2D space, if you apply above mapping function we can convert it into 3D space and it looks like this,

Here $x = 2$ and $y = 3$,

Hence point in 3D space is:

$$(2^2, \sqrt{2} * 2 * 3, 3^2) = (4, 6\sqrt{2}, 9)$$

- Kernel Trick for 2nd degree Polynomial Mapping

$$\phi(x, y) = (x^2, \sqrt{2}xy, y^2)$$

$$\begin{aligned} \phi(\mathbf{a})^T \cdot \phi(\mathbf{b}) &= \begin{pmatrix} a_1^2 \\ \sqrt{2} a_1 a_2 \\ a_2^2 \end{pmatrix}^T \cdot \begin{pmatrix} b_1^2 \\ \sqrt{2} b_1 b_2 \\ b_2^2 \end{pmatrix} = a_1^2 b_1^2 + 2a_1 b_1 a_2 b_2 + a_2^2 b_2^2 \\ &= (a_1 b_1 + a_2 b_2)^2 = \left(\begin{pmatrix} a_1 \\ a_2 \end{pmatrix}^T \cdot \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \right)^2 = (\mathbf{a}^T \cdot \mathbf{b})^2 \end{aligned}$$

Kernel Trick in Support
For example, one mapping function $\phi: \mathbb{R}^2 \rightarrow \mathbb{R}^3$
is given as follows:
 $\phi(x, y) = (x^2, \sqrt{2}xy, y^2)$
Consider a point $(2, 3)$ in 2D space. If you
convert it into 3D space and it looks like this,
then $x = 2$ and $y = 3$.
Hence point in 3D space is

Types of Kernel in Support Vector Machine

Linear Kernel

- Linear kernels are of the type

$$\underline{k(x, y)} = x^T \cdot y$$

- where x and y are two vectors.
- Therefore $k(x, y) = \underline{\phi(x)} \cdot \phi(y) = x^T \cdot y$

Types of Kernel in Support Vector Machine

Polynomial Kernel

- Polynomial kernels are of the type:

$$\underline{k(x, y) = (x^T y)^q}$$

- This is called homogeneous kernel.
- Here, q is the degree of the polynomial.
- If $q = 2$ then it is called **quadratic kernel**.

Types of Kernel in Support Vector Machine

Polynomial Kernel

- For inhomogeneous kernels, this is given as:

$$k(x, y) = (c + x^T y)^q$$

- Here c is a constant and q is the degree of the polynomial.
- If c is zero and degree is one, the polynomial kernel is reduced to a linear kernel.
- The value of degree q should be optimal as more degree may lead to overfitting.

Gaussian Kernel

- Radial Basis Functions (RBFs) or Gaussian kernels are extremely useful in SVM.
- The RBF function is shown as below:

$$k(x, y) = e^{\frac{-(x-y)^2}{2\sigma^2}}$$

- Here, γ is an important parameter. If γ is small, then the RBF is similar to linear SVM and if γ is large, then the kernel is influenced by more support vectors.

- Consider two data points $x = (1, 2)$ and $y = (2, 3)$ with $\sigma = 1$.
- Apply RBF kernel and find the value of RBF kernel for these points.

$$k(x, y) = e^{\frac{-(x-y)^2}{2\sigma^2}}$$

- Substitute the value of x and y in RBF kernel.
- The squared distance between the points $(1, 2)$ and $(2, 3)$ is given as:

$$(1 - 2)^2 + (2 - 3)^2 = 2$$

- If $\sigma = 1$, then $k(x, y) = e^{\left\{-\frac{2}{2}\right\}} = e^{-1} = 0.3679$.

Kernel Trick

- Kernel trick means replacing the dot product in mapping functions with a kernel function.

$$\underline{k(x, y)} = \underline{\phi(x)} \cdot \overset{\checkmark}{\underline{\phi(y)}}$$

- Similar to mapping functions, kernels help in mapping data from input space to higher-dimensional feature space with least computations.
- Performing the kernel operation is much easier compared to mapping functions.
- This is illustrated in the following numerical example.

- Consider two data points (1, 2) and (3, 4)
- Apply a polynomial kernel $k(x, y) = (x^T y)^2$ and show that it is equivalent to mapping function $\phi = (x^2, y^2, \sqrt{2} xy)$
- **Solution:**
- The mapping function is given as $\phi = (x^2, y^2, \sqrt{2} xy)$
- Let us apply the mapping function first for the first data point (1, 2) using Eq. given as:
- $\phi = (x^2, y^2, \sqrt{2} xy)$
- *First data point* $\phi(1, 2) = (1^2, 2^2, \sqrt{2} * 1 * 2) = \underline{(1, 4, 2\sqrt{2})}$

Kernel Trick in Support Vector Machine Solved Example

- Consider two data points (1, 2) and (3, 4)
- Apply a polynomial kernel $k(x, y) = (x^T y)^2$ and show that it is equivalent to mapping function $\phi = (x^2, y^2, \sqrt{2} xy)$
- **Solution:**
- First data point $\phi(1, 2) = (1^2, 2^2, \sqrt{2} * 1 * 2) = (1, 4, 2\sqrt{2})$
- Second data point $\phi(3, 4) = (3^2, 4^2, \sqrt{2} * 3 * 4) = (9, 16, 12\sqrt{2})$
- $\phi(1, 2) \cdot \phi(3, 4) = (1, 4, 2\sqrt{2}) \cdot (9, 16, 12\sqrt{2}) = (1 \times 9 + 4 \times 16 + 24(2)) = 121$

Support Vector Machine – Non Linear Example Solved

$$\Phi_1 \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{cases} \begin{pmatrix} 4 - x_2 + |x_1 - x_2| \\ 4 - x_1 + |x_1 - x_2| \end{pmatrix} & \text{if } \sqrt{x_1^2 + x_2^2} > 2 \\ \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} & \text{otherwise} \end{cases}$$

Positive Examples

$$\left\{ \begin{pmatrix} 2 \\ 2 \end{pmatrix}, \begin{pmatrix} 2 \\ -2 \end{pmatrix}, \begin{pmatrix} -2 \\ -2 \end{pmatrix}, \begin{pmatrix} -2 \\ 2 \end{pmatrix} \right\} \rightarrow$$