MACHINE LEARNING

Unit-2

Important Q & A

1) Discuss about Linear Regression

Ans) Linear regression is a fundamental statistical method used to model the relationship between a dependent variable (also called the target or outcome variable) and one or more independent variables (also called predictors or features). The goal is to find the linear equation that best predicts the dependent variable based on the independent variables. Here's a breakdown of the key concepts:

Key Concepts

1. **Dependent Variable (Y)**: The outcome or target variable you want to predict.
2. **Independent Variables (X)**: The input features or predictors used to predict the dependent variable.
3. **Linear Equation**: The equation that represents the relationship between the dependent and independent variables. In simple linear regression, it's of the form: $$ Y = \beta_0 + \beta_1X + \epsilon $$ where:
    - Y is the dependent variable
    - β0\beta_0 is the intercept (the value of Y when X is 0)
    - β1\beta_1 is the slope coefficient (the change in Y for a one-unit change in X)
    - X is the independent variable
    - ϵ\epsilon is the error term (the difference between the actual and predicted values of Y)

Types of Linear Regression

There are two main types of linear regression:

Simple Linear Regression

This is the simplest form of linear regression, and it involves only one independent variable and one dependent variable. The equation for simple linear regression is:
y=β0+β1X
where:

- Y is the dependent variable
- X is the independent variable
- β0 is the intercept
- β1 is the slope

Multiple Linear Regression

This involves more than one independent variable and one dependent variable. The equation for multiple linear regression is:
y=β0+β1X1+β2X2+………βnXn
where:

- Y is the dependent variable
- X1, X2, …, Xn are the independent variables
- β0 is the intercept
- β1, β2, …, βn are the slopes

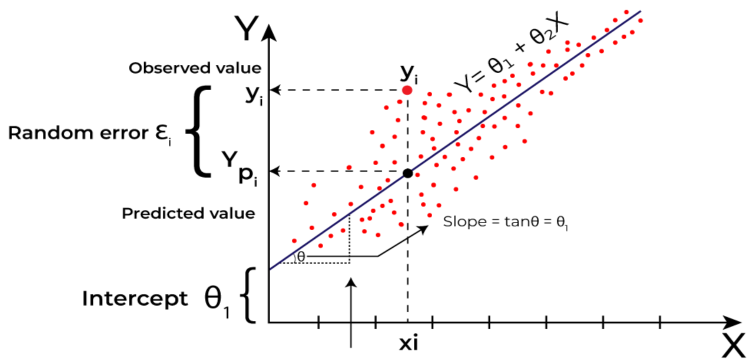
The goal of the algorithm is to find the best Fit Line equation that can predict the values based on the independent variables.

In regression set of records are present with X and Y values and these values are used to learn a function so if you want to predict Y from an unknown X this learned function can be used. In regression we have to find the value of Y, So, a function is required that predicts continuous Y in the case of regression given X as independent features.

What is the best Fit Line?

Our primary objective while using linear regression is to locate the best-fit line, which implies that the error between the predicted and actual values should be kept to a minimum. There will be the least error in the best-fit line.

The best Fit Line equation provides a straight line that represents the relationship between the dependent and independent variables. The slope of the line indicates how much the dependent variable changes for a unit change in the independent variable(s).


Y
Observed value
y_i
Random error ε_i
Y_{p_i}
Predicted value
Intercept θ_1
y_i
$Y = \theta_1 + \theta_2 X$
Slope = tanθ = θ_1
θ
xi
X

Linear Regression

Here Y is called a dependent or target variable and X is called an independent variable also known as the predictor of Y. Linear regression performs the task to predict a dependent variable value (y) based on a given independent variable (x)). Hence, the name is Linear

Regression. In the figure above, X (input) is the work experience and Y (output) is the salary of a person. The regression line is the best-fit line for our model.

We utilize the cost function to compute the best values in order to get the best fit line since different values for weights or the coefficient of lines result in different regression lines.

2) What is Multi-layer Perception? Explain its works with examples?

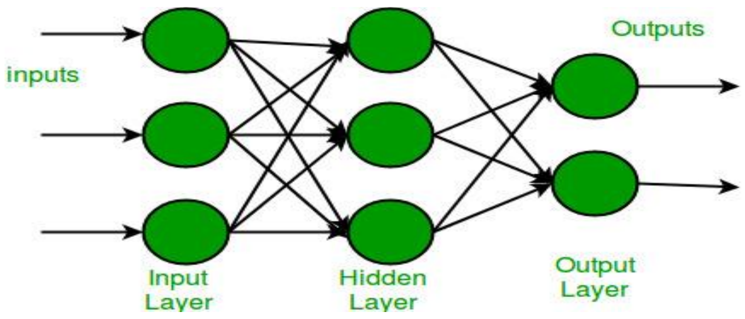
Ans) Multi-Layer Perceptron (MLP) is an artificial neural network widely used for solving classification and regression tasks. In this article, we will explore the concept of MLP in-depth and demonstrate how to implement it in Python using the TensorFlow library.

Multilayer Perceptron

A **Multi-Layer Perceptron (MLP)** consists of fully connected dense layers that transform input data from one dimension to another. It is called "multi-layer" because it contains an input layer, one or more hidden layers, and an output layer. The purpose of an MLP is to model complex relationships between inputs and outputs, making it a powerful tool for various machine learning tasks.

The key components of Multi-Layer Perceptron includes:

- **Input Layer**: Each neuron (or node) in this layer corresponds to an input feature. For instance, if you have three input features, the input layer will have three neurons.
- **Hidden Layers**: An MLP can have any number of hidden layers, with each layer containing any number of nodes. These layers process the information received from the input layer.
- **Output Layer**: The output layer generates the final prediction or result. If there are multiple outputs, the output layer will have a corresponding number of neurons.



Every connection in the diagram is a representation of the fully connected nature of an MLP. This means that every node in one layer connects to every node in the next layer. As the data moves through the network, each layer transforms it until the final output is generated in the output layer.

Working of Multi-Layer Perceptron

Let's delve in to the working of the multi-layer perceptron. The key mechanisms such as forward propagation, loss function, backpropagation, and optimization.

Step 1: Forward Propagation

In **forward propagation**, the data flows from the input layer to the output layer, passing through any hidden layers. Each neuron in the hidden layers processes the input as follows:

1. **Weighted Sum**: The neuron computes the weighted sum of the inputs:
   - $z=\sum iwixi+b$
     - Where:
       - xi is the input feature.
       - wi is the corresponding weight.
       - b is the bias term.
2. **Activation Function**: The weighted sum z is passed through an activation function to introduce non-linearity. Common activation functions include:
   - **Sigmoid**: $\sigma(z)=1/1+e-z$
   - **ReLU (Rectified Linear Unit)**: $f(z)=\max(0,z)$
   - **Tanh (Hyperbolic Tangent)**: $\tanh(z)=2/1+e-2z-1$

Step 2: Loss Function

Once the network generates an output, the next step is to calculate the **loss** using a loss function. In supervised learning, this compares the predicted output to the actual label.

For a classification problem, the commonly used **binary cross-entropy** loss function is:

$L=-1/N\sum i=1N[yilog(y\hat{}i)+(1-yi)log(1-y\hat{}i)]$

Where:

- yi is the actual label.
- $y\hat{}i$ is the predicted label.
- N is the number of samples.

For regression problems, the **mean squared error (MSE)** is often used:

$MSE=1N\sum i=1N(yi-y\hat{}i)2$

Step 3: Backpropagation

The goal of training an MLP is to minimize the loss function by adjusting the network's weights and biases. This is achieved through **backpropagation**:

1. **Gradient Calculation**: The gradients of the loss function with respect to each weight and bias are calculated using the chain rule of calculus.
2. **Error Propagation**: The error is propagated back through the network, layer by layer.

3. **Gradient Descent**: The network updates the weights and biases by moving in the opposite direction of the gradient to reduce the loss: $w=w-\eta\cdot\partial L\partial w$
    - Where:
        - w is the weight.
        - η is the learning rate.
        - $\partial L\partial w$ is the gradient of the loss function with respect to the weight.

Step 4: Optimization

MLPs rely on **optimization algorithms** to iteratively refine the weights and biases during training. Popular optimization methods include:

- **Stochastic Gradient Descent (SGD)**: Updates the weights based on a single sample or a small batch of data: $w=w-\eta\cdot\partial L\partial w$
- **Adam Optimizer**: An extension of SGD that incorporates momentum and adaptive learning rates for more efficient training:
    - $mt=\beta1mt-1+(1-\beta1)\cdot gt$
    - $vt=\beta2vt-1+(1-\beta2)\cdot gt2$
- Here, gt represents the gradient at time *t*, and $\beta1,\beta2$ are decay rates.

Now that we are done with the theory part of multi-layer perception, let's go ahead and implement some code in python using the TensorFlow library.

3) What is Back Propagation? Derive it with example

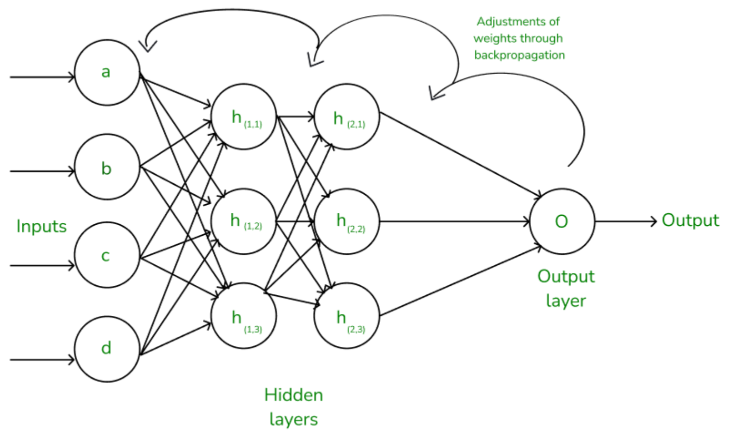
Ans) **Backpropagation (**short for "***Backward Propagation of Errors***") is a method used to train artificial neural networks. Its goal is to reduce the difference between the model's predicted output and the actual output by adjusting the weights and biases in the network.

A **neural network** is a structured system composed of computing units called neurons, which enable it to compute functions. These neurons are interconnected through edges and assigned an **activation function**, along with adjustable parameters. These parameters allow the neural network to compute specific functions. Regarding activation functions, higher activation values indicate greater neuron activation in the network.

What is Backpropagation?

Backpropagation is a powerful algorithm in deep learning, primarily used to train artificial neural networks, particularly **feed-forward networks**. It works iteratively, minimizing the cost function by adjusting weights and biases.

In each epoch, the model adapts these parameters, reducing loss by following the error gradient. Backpropagation often utilizes optimization algorithms like **gradient descent** or **stochastic gradient descent**. The algorithm computes the gradient using the chain rule from calculus, allowing it to effectively navigate complex layers in the neural network to minimize the cost function.



fig(a) A simple illustration of how the backpropagation works by adjustments of weights

Why is Backpropagation Important?

Backpropagation plays a critical role in how neural networks improve over time. Here's why:

1. **Efficient Weight Update**: It computes the gradient of the loss function with respect to each weight using the chain rule, making it possible to update weights efficiently.
2. **Scalability**: The backpropagation algorithm scales well to networks with multiple layers and complex architectures, making deep learning feasible.
3. **Automated Learning**: With backpropagation, the learning process becomes automated, and the model can adjust itself to optimize its performance.

Working of Backpropagation Algorithm

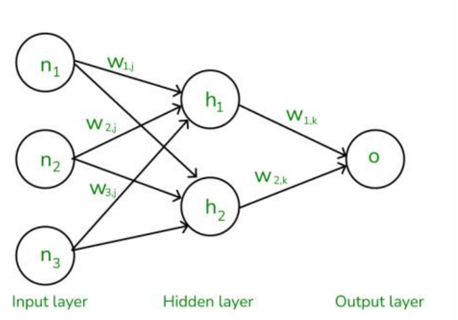
The Backpropagation algorithm involves two main steps: the **Forward Pass** and the **Backward Pass**.

How Does the Forward Pass Work?

In the **forward pass**, the input data is fed into the input layer. These inputs, combined with their respective weights, are passed to hidden layers.

For example, in a network with two hidden layers (h1 and h2 as shown in Fig. (a)), the output from h1 serves as the input to h2. Before applying an activation function, a bias is added to the weighted inputs.

Each hidden layer applies an activation function like **ReLU (Rectified Linear Unit)**, which returns the input if it's positive and zero otherwise. This adds non-linearity, allowing the model to learn complex relationships in the data. Finally, the outputs from the last hidden layer are passed to the output layer, where an activation function, such as **softmax**, converts the weighted outputs into probabilities for classification.



The forward pass using weights and biases

How Does the Backward Pass Work?

In the backward pass, the error (the difference between the predicted and actual output) is propagated back through the network to adjust the weights and biases. One common method for error calculation is the **Mean Squared Error (MSE)**, given by:

MSE=(Predicted Output−Actual Output)2

Once the error is calculated, the network adjusts weights using **gradients**, which are computed with the chain rule. These gradients indicate how much each weight and bias should be adjusted to minimize the error in the next iteration. The backward pass continues layer by layer, ensuring that the network learns and improves its performance. The activation function, through its derivative, plays a crucial role in computing these gradients during backpropagation.

4) What are radial basis functions and Networks? Explain Briefly

Ans) Radial Basis Function (RBF) Neural Networks are a specialized type of Artificial Neural Network (ANN) used primarily for function approximation tasks. Known for their distinct three-layer architecture and universal approximation capabilities, RBF Networks offer faster learning speeds and efficient performance in classification and regression problems. This article delves into the workings, architecture, and applications of RBF Neural Networks.

What are Radial Basis Functions?

Radial Basis Functions (RBFs) are a special category of feed-forward neural networks comprising three layers:

1. **Input Layer**: Receives input data and passes it to the hidden layer.
2. **Hidden Layer**: The core computational layer where RBF neurons process the data.
3. **Output Layer**: Produces the network's predictions, suitable for classification or regression tasks.

How Do RBF Networks Work?

RBF Networks are conceptually similar to K-Nearest Neighbor (k-NN) models, though their implementation is distinct. The fundamental idea is that an item's predicted target value is influenced by nearby items with similar predictor variable values. Here's how RBF Networks operate:

1. **Input Vector**: The network receives an n-dimensional input vector that needs classification or regression.
2. **RBF Neurons**: Each neuron in the hidden layer represents a prototype vector from the training set. The network computes the Euclidean distance between the input vector and each neuron's center.
3. **Activation Function**: The Euclidean distance is transformed using a Radial Basis Function (typically a Gaussian function) to compute the neuron's activation value. This value decreases exponentially as the distance increases.
4. **Output Nodes**: Each output node calculates a score based on a weighted sum of the activation values from all RBF neurons. For classification, the category with the highest score is chosen.

Key Characteristics of RBFs

- **Radial Basis Functions**: These are real-valued functions dependent solely on the distance from a central point. The Gaussian function is the most commonly used type.
- **Dimensionality**: The network's dimensions correspond to the number of predictor variables.
- **Center and Radius**: Each RBF neuron has a center and a radius (spread). The radius affects how broadly each neuron influences the input space.

Architecture of RBF Networks

The architecture of an RBF Network typically consists of three layers:

Input Layer

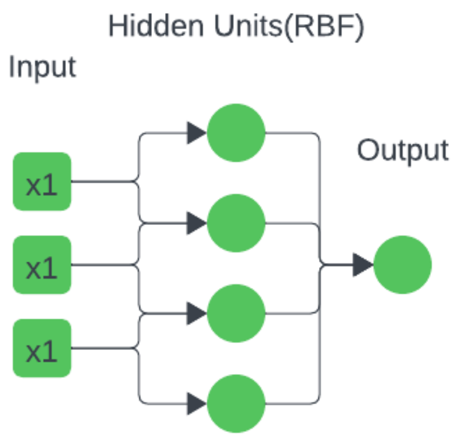
- **Function:** After receiving the input features, the input layer sends them straight to the hidden layer.
- **Components:** It is made up of the same number of neurons as the characteristics in the input data. One feature of the input vector corresponds to each neuron in the input layer.

Hidden Layer

- **Function:** This layer uses radial basis functions (RBFs) to conduct the non-linear transformation of the input data.
- **Components:** Neurons in the buried layer apply the RBF to the incoming data. The Gaussian function is the RBF that is most frequently utilized.
- **RBF Neurons:** Every neuron in the hidden layer has a spread parameter (σ) and a center, which are also referred to as prototype vectors. The spread parameter modulates the distance between the center of an RBF neuron and the input vector, which in turn determines the neuron's output.

Output Layer

- **Function:** The output layer uses weighted sums to integrate the hidden layer neurons' outputs to create the network's final output.
- **Components:** It is made up of neurons that combine the outputs of the hidden layer in a linear fashion. To reduce the error between the network's predictions and the actual target values, the weights of these combinations are changed during training.



Training Process of radial basis function neural network

An RBF neural network must be trained in three stages: choosing the center's, figuring out the spread parameters, and training the output weights.

Step 1: Selecting the Centers

- **Techniques for Centre Selection:** Centre's can be picked at random from the training set of data or by applying techniques such as k-means clustering.
- **K-Means Clustering:** The center's of these clusters are employed as the center's for the RBF neurons in this widely used center selection technique, which groups the input data into k groups.

Step 2: Determining the Spread Parameters

- **The spread parameter (σ)** governs each RBF neuron's area of effect and establishes the width of the RBF.
- **Calculation:** The spread parameter can be manually adjusted for each neuron or set as a constant for all neurons. Setting σ based on the separation between the center's is a popular method, frequently accomplished with the help of a heuristic like dividing the greatest distance between canters by the square root of twice the number of center's

Step 3: Training the Output Weights

- **Linear Regression:** The objective of linear regression techniques, which are commonly used to estimate the output layer weights, is to minimize the error between the anticipated output and the actual target values.
- **Pseudo-Inverse Method:** One popular technique for figuring out the weights is to utilize the pseudo-inverse of the hidden layer outputs matrix

5) Discuss about Splines and compare Splines with RBF Network?

Ans) Splines are piecewise polynomial functions used for interpolation and smoothing. They are widely used in computer graphics, data fitting, and numerical analysis.

Types of Splines:

- **Linear Splines**: Piecewise linear functions that connect data points with straight lines.
- **Cubic Splines**: Piecewise cubic polynomials that ensure smoothness at the data points (knots).
- **B-Splines**: Basis splines that provide a flexible way to construct splines with varying degrees of smoothness.

Key Concepts:

- **Knots**: The points where the polynomial pieces meet.
- **Continuity**: Splines ensure smooth transitions between polynomial segments. For example, cubic splines ensure continuity of the function, its first derivative, and its second derivative at the knots.
- **Applications**: Splines are used for curve fitting, interpolation, and smoothing noisy data.

Comparison

- **RBF Networks**: Focus on using radial basis functions for interpolation and function approximation. They are particularly useful in neural networks for capturing complex patterns.
- **Splines**: Focus on constructing smooth, piecewise polynomial functions for interpolation and data fitting. They ensure smooth transitions between segments and are widely used in numerical analysis.

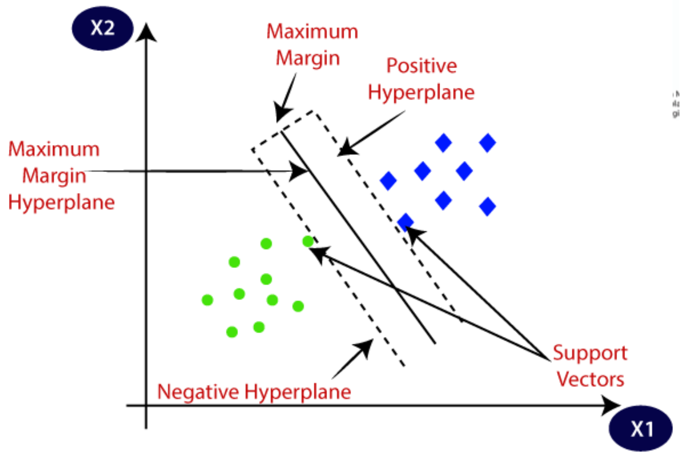
6) Explain Support Vector Machine (SVM) Algorithm with neat diagram?

Ans) Support Vector Machine Algorithm

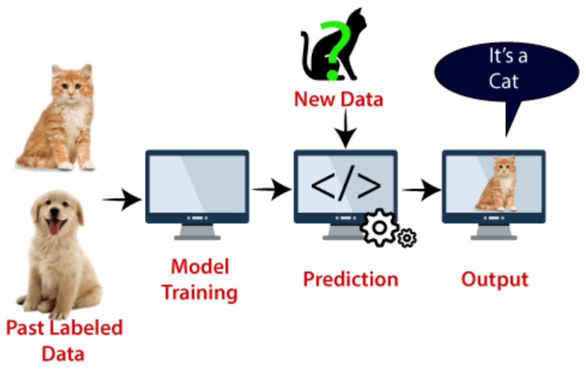
Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.

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.

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



Example: SVM can be understood with the example that we have used in the KNN classifier. 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:



SVM algorithm can be used for **Face detection, image classification, text categorization,** etc.

Types of SVM

SVM can be of two types:

- **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.
- **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.

Hyperplane and Support Vectors in the SVM algorithm:

Hyperplane: 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.

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

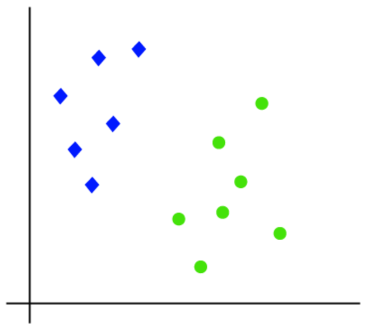
Support Vectors:

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.

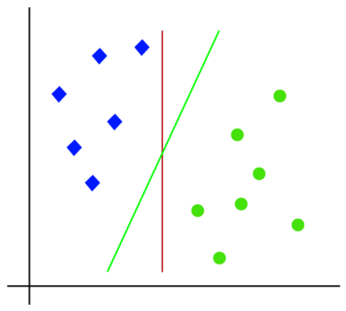
How does SVM works?

Linear SVM:

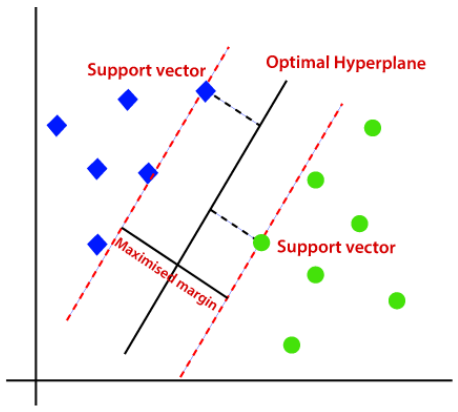
The working of the SVM algorithm can be understood by using an example. Suppose we have a dataset that has two tags (green and blue), and the dataset has two features x1 and x2. We want a classifier that can classify the pair(x1, x2) of coordinates in either green or blue. Consider the below image:



So as it is 2-d space so 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 below image:

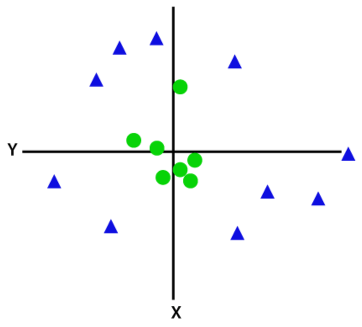


Hence, the SVM algorithm helps to find the best line or decision boundary; this best boundary or region is called as a **hyperplane**. SVM algorithm finds the closest point of the lines from both the classes. These points are called support vectors. The distance between the vectors and the hyperplane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



Non-Linear SVM:

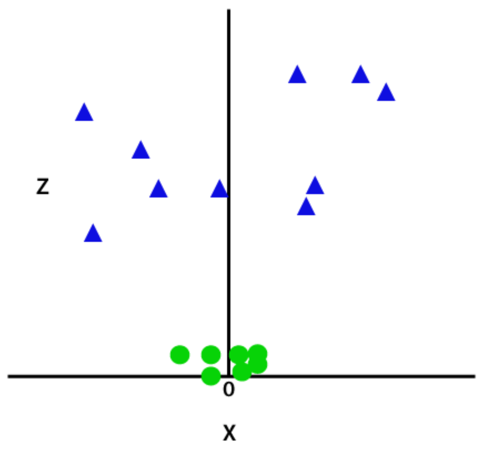
If data is linearly arranged, then we can separate it by using a straight line, but for non-linear data, we cannot draw a single straight line. Consider the below image:



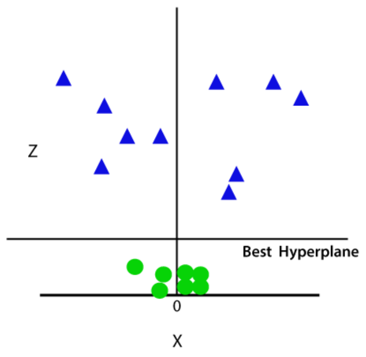
So to separate these data points, we need to add one more dimension. For linear data, we have used two dimensions x and y, so for non-linear data, we will add a third dimension z. It can be calculated as:

$z=x^2 +y^2$

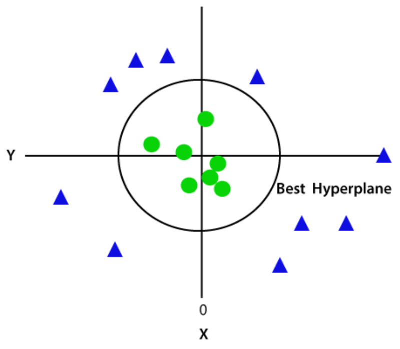
By adding the third dimension, the sample space will become as below image:



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



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:



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