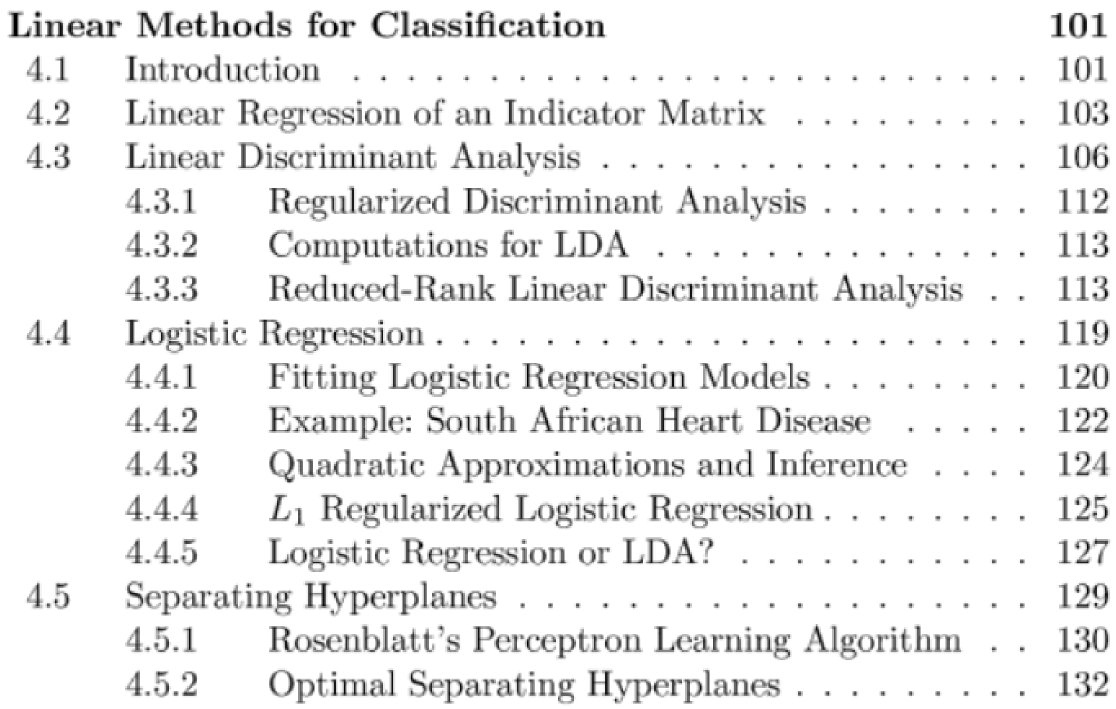
Linear Models for Classification

Discriminant Analysis

Logistic Regression

Separating Hyperplanes.



Logistic Regression

The objective of logistic regression is to model the **posterior probabilities of K classes via linear functions in x, while at the same time ensuring that they sum to one and remain in [0,1].**

$$log\frac{Pr(G=1|X=x)}{Pr(G=K|X=x)} = \beta_{0_1} + \beta_1^T x$$

$$log\frac{Pr(G=2|X=x)}{Pr(G=K|X=x)} = \beta_{0_2} + \beta_2^T x$$

$$\cdots$$

$$log\frac{Pr(G=k-1|X=x)}{Pr(G=K|X=x)} = \beta_{0_(k-1)} + \beta_{k-1}^T x$$

Logistic regression, self-generated.

Here K-1 is the number of logit transformations(log-odds). The last class is used as the denominator in the log-odds ratio. We can obtain ***Pr(G=k|X=x)*** using exponents

$$Pr(G=K|X=x) = \frac{exp(\beta_{k0} + \beta_k^T x)}{1 + \sum_{l=1}^{K-1} exp(\beta_{0_l} + \beta_l^T x)}, K = 1, \ldots, K-1$$

$$Pr(G=K|X=x) = \frac{1}{1 + \sum_{l=1}^{K-1} exp(\beta_{0_l} + \beta_l^T x)}$$

Obtaining ***Pr(G=k|X=x),*** *self-generated.*

To simplify the notation we will use Θ as the entire parameter set Θ = ***{β01,β1^T,…,β0(k-1),β(k-1)T}*** and thus denote the probabilities ***Pr(G=k|X=x)=Pk(x:Θ).***

Fitting logistic regression models

Logistic models are usually fitted by maximum likelihood, using the conditional likelihood of G given X. Since P(G|X) specifies the conditional distribution, the multinomial distribution is appropriate. The log-likelihood for N observations is:

$$l(\theta) = \sum_{i=1}^{N} log P_{g_i}(x_i; \theta)$$

Log-likelihood for N observations, self-generated.

where ***Pk(xi:Θ)=Pr(G=k|X=xi,Θ).***

In the two-class case, the algorithm is simple, it is convenient to code the two classes ***gi = {1,2}***via a ***0,1*** response ***yi*** where:

- ***yi =1*** if ***gi =1***
- ***yi = 0*** if ***gi = 2***

Let ***p1(x:Θ)=p(x:Θ)*** and ***p2(x:Θ)=1-p(x:Θ).*** The log-likelihood can be written as

$$l(\beta) = \sum_{i=1}^{N} \{y_i log\, p(x_i : \beta) + (1-y_i) log\, (1-p(x_i : \beta))\} = \sum_{i=1}^{N} \{y_i \beta^T x_i - log\, (1+e^{\beta^T x_i})\}$$

Log-likelihood for N observations, self-generated.

Here ***β = {β_01,β_1},*** and we assume that the vector of inputs ***xi*** includes the constant term **1** to accommodate the intercept. To maximize the log-likelihood we set the derivates to **0**.

$$\frac{dl(\beta)}{d\beta} = -\sum_{i=1}^{N} x_i(y_i - p(x_i : \beta)) = 0$$

The first derivative, self-generated.

These are p+1 equations non-linear in ***β.*** To solve their equations, we can use the Newton-Raphson algorithm, which needs the second derivative or the hessian matrix.

$$\frac{d^2 l(\beta)}{d\beta d\beta^T} = -\sum_{i=1}^{N} x_i x_i^T p(x_i : \beta)(1 - p(x_i : \beta)))$$

The second derivative, self-generated.

starting with ***βold***, a single Newton update is:

$$\beta^{new} = \beta^{old} - (\frac{d^2 l(\beta)}{d\beta d\beta^T})^{-1} \frac{dl(\beta)}{d\beta}$$

βold *Newton update, self-generated****.***

where the derivatives are evaluated at ***βold.*** In matrix notation, we can write it as:

$$\frac{dl(\beta)}{d\beta} = X^T(Y-P) \vdots \frac{d^2 l(\beta)}{d\beta d\beta^T} = -X^T W X$$

Derivatives in matrix notation, self-generated

Then, the Newton step looks like this:

$$\beta^{new} = \beta^{old} + (X^T W X)^{-1} X^T (y-p) = (X^T W X)^{-1} X^T (X\beta^{old} + W^{-1}(y-p)) = (X^T W X)^{-1} X^T W_z$$

$$where\ z = X\beta^{old} + W^{-1}(y-p)$$

Newton steps in matrix notation, self-generated.

z is known as the adjusted response. This model is an iterative reweighted least-squares(IRLS) since at each iteration it solves the weighted least squares problem:

$$\beta^{new} \leftarrow argmin_x (z - X\beta)^T W (Z - X\beta)$$

IRLS maximization problem, self-generated.

Normally ***β=0*** is a good starting point for the iterative process.

Quadratic approximation and inference

The maximum-likelihood parameter estimates ***β^*** satisfying a self-consistency relationship, they are coefficients of a weighted least-squares fit, where the response is:

$$z_i = X_i^T \hat{\beta} + \frac{y_i - \hat{p}_i}{\hat{p}_i(1 - \hat{p}_i)}$$

The weighted least-squares response, self-generated.

the weights are ***wi = pi^(1-pi^)***, both depending on ***β^*** itself. Tho connection with least-squares offers a lot to us:

- The weighted residual sum of squares is the familiar Pearson Chi-Squared statistic.
- Asymptotic likelihood theory says that if the model is correct, ***β*** is consistent, so it converges to the true ***β.***
- The Central Limit Theorem shows that the distribution of ***β^*** converges to ***N(β,(X^TWX)^(-1)).***

As model building for logistic regression models can be costly due to the required iteration, there exist some popular shortcuts to avoid the iteration, some of them are Rao score test to Wald test, they are based on the maximum-likelihood fit of the current model.

L1 regularized logistic regression

The L1 penalty can be used in any linear regression model. For logistic regression, we can add it as:

$$max_{\beta_0,\beta}\{\sum_{i=1}^{N}[y_i(\beta_0 + \beta^T x_i) - log(1 + e^{\beta_0 + \beta^T x_i})] - \lambda \sum_{j=1}^{p} |\beta_j|\}$$

L1-regularization for logistic regression, self-generated.

We can solve it with IRLS using the quadratic approximation. The score equation for non-zero coefficient variables have the form:

$$x_j^T(y - p) = \lambda \; sign(\beta_j)$$

Non-zero coefficient variables, self-generated.

which generalizes the LAR. Path algorithms as LAR for lasso are more difficult because the coefficients are pice-wise smooth rather than linear. Nevertheless, progress can be estimated with quadratic approximations.

Logistic regression vs linear discriminant analysis

Both models seem to be the same, they have exactly the same form, but the **logistic regression is more general in that It makes fewer assumptions.**

The logistic regression takes the density function *Pr(x)* and fits the parameters of *P(G|X)* by maximizing the conditional likelihood. Meanwhile, LDA first the parameters by maximizing the full log-likelihood, based on the prior density.

By relying on more assumptions, logistic regression has more information about the parameters and can obtain lower variances on estimation.

Observations far from the decision boundaries are important for the covariance matrix estimation, which makes LDA not robust to gross outliers.

If the data in a plane can be separated by a line, the logistic regression will not converge to the best solution, otherwise, LDA will do it.

Conclusion

Logistic regression and linear discriminant analysis are both good approaches to perform a simple classification. Logistic regression is a more robust model because of the lower amount of assumptions, but in practice, both models perform similarly.

Separating Hyperplanes

- **Rosenblatts Perception Learning Algorithm**
- **Optimal Separating Hyperplanes**

Rosenblatts Perception Learning Algorithm

Perceptron in Machine Learning

In Machine Learning and Artificial Intelligence, Perceptron is the most commonly used term for all folks. It is the primary step to learn Machine Learning and Deep Learning technologies, which consists of a set of weights, input values or scores, and a threshold. ***Perceptron is a building block of an Artificial Neural Network***. Initially, in the mid of 19th century, **Mr. Frank Rosenblatt** invented the Perceptron for performing certain calculations to detect input data capabilities or business intelligence. Perceptron is a linear Machine

Learning algorithm used for supervised learning for various binary classifiers. This algorithm enables neurons to learn elements and processes them one by one during preparation. In this tutorial, "Perceptron in Machine Learning," we will discuss in-depth knowledge of Perceptron and its basic functions in brief. Let's start with the basic introduction of Perceptron.

What is the Perceptron model in Machine Learning?

Perceptron is Machine Learning algorithm for supervised learning of various binary classification tasks. Further, ***Perceptron is also understood as an Artificial Neuron or neural network unit that helps to detect certain input data computations in business intelligence***.

Perceptron model is also treated as one of the best and simplest types of Artificial Neural networks. However, it is a supervised learning algorithm of binary classifiers. Hence, we can consider it as a single-layer neural network with four main parameters, i.e., **input values, weights and Bias, net sum, and an activation function.**

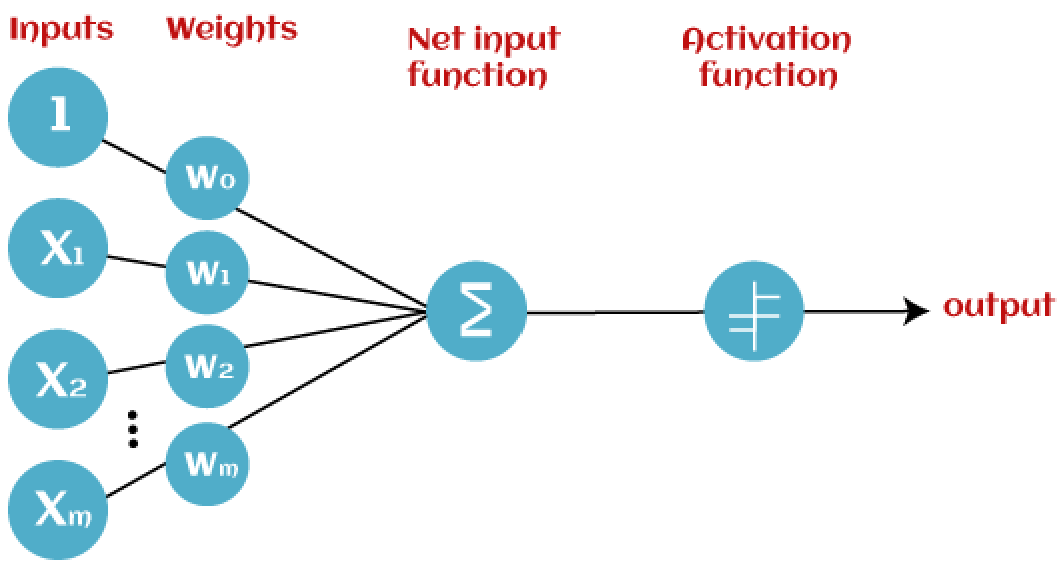
What is Binary classifier in Machine Learning?

In Machine Learning, binary classifiers are defined as the function that helps in deciding whether input data can be represented as vectors of numbers and belongs to some specific class.

Binary classifiers can be considered as linear classifiers. In simple words, we can understand it as a ***classification algorithm that can predict linear predictor function in terms of weight and feature vectors.***

Basic Components of Perceptron

Mr. Frank Rosenblatt invented the perceptron model as a binary classifier which contains three main components. These are as follows:



- **Input Nodes or Input Layer:**

This is the primary component of Perceptron which accepts the initial data into the system for further processing. Each input node contains a real numerical value.

- **Wight and Bias:**

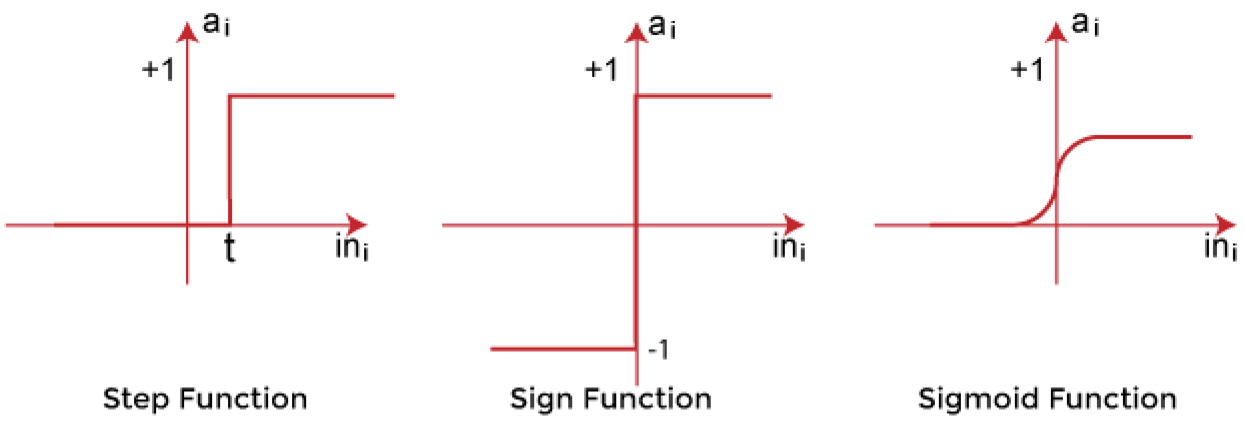
Weight parameter represents the strength of the connection between units. This is another most important parameter of Perceptron components. Weight is directly proportional to the strength of the associated input neuron in deciding the output. Further, Bias can be considered as the line of intercept in a linear equation.

- **Activation Function:**

These are the final and important components that help to determine whether the neuron will fire or not. Activation Function can be considered primarily as a step function.

Types of Activation functions:

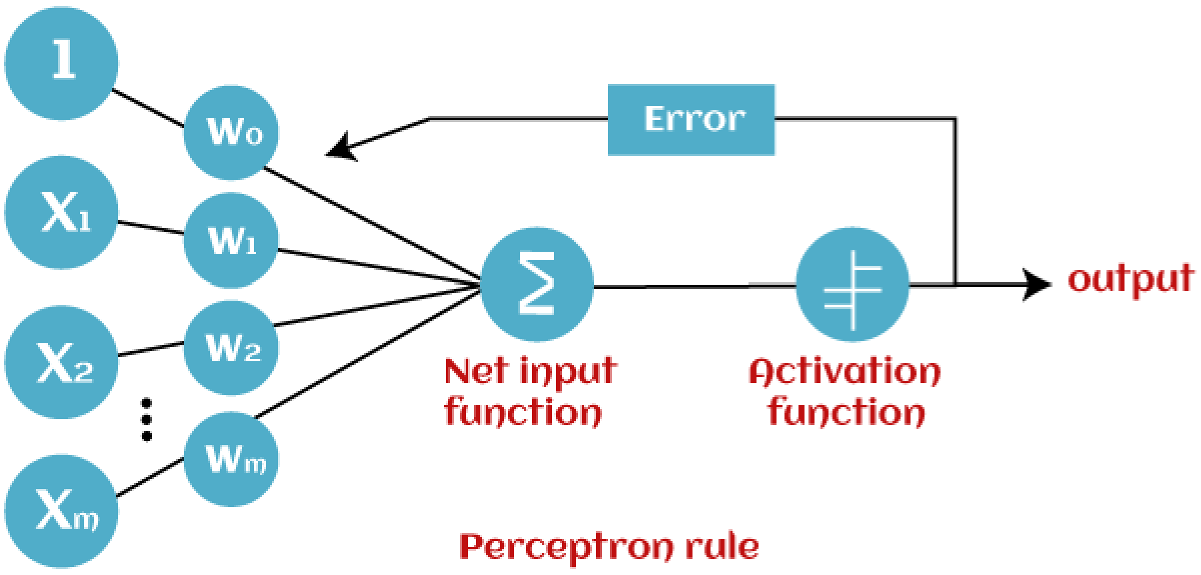
- Sign function
- Step function, and
- Sigmoid function



The data scientist uses the activation function to take a subjective decision based on various problem statements and forms the desired outputs. Activation function may differ (e.g., Sign, Step, and Sigmoid) in perceptron models by checking whether the learning process is slow or has vanishing or exploding gradients.

How does Perceptron work?

In Machine Learning, Perceptron is considered as a single-layer neural network that consists of four main parameters named input values (Input nodes), weights and Bias, net sum, and an activation function. The perceptron model begins with the multiplication of all input values and their weights, then adds these values together to create the weighted sum. Then this weighted sum is applied to the activation function 'f' to obtain the desired output. This activation function is also known as the **step function** and is represented by **'f'**.



This step function or Activation function plays a vital role in ensuring that output is mapped between required values (0,1) or (-1,1). It is important to note that the weight of input is indicative of the strength of a node. Similarly, an input's bias value gives the ability to shift the activation function curve up or down.

Perceptron model works in two important steps as follows:

Step-1

In the first step first, multiply all input values with corresponding weight values and then add them to determine the weighted sum. Mathematically, we can calculate the weighted sum as follows:

∑wi*xi = x1*w1 + x2*w2 +…wn*xn

Add a special term called **bias 'b'** to this weighted sum to improve the model's performance.

∑wi*xi + b

Step-2

In the second step, an activation function is applied with the above-mentioned weighted sum, which gives us output either in binary form or a continuous value as follows:

Y = f(∑wi*xi + b)

Types of Perceptron Models

Based on the layers, Perceptron models are divided into two types. These are as follows:

1. Single-layer Perceptron Model
2. Multi-layer Perceptron model

Single Layer Perceptron Model:

This is one of the easiest Artificial neural networks (ANN) types. A single-layered perceptron model consists feed-forward network and also includes a threshold transfer function inside the model. The main objective of the single-layer perceptron model is to analyze the linearly separable objects with binary outcomes.

In a single layer perceptron model, its algorithms do not contain recorded data, so it begins with inconstantly allocated input for weight parameters. Further, it sums up all inputs (weight). After adding all inputs, if the total sum of all inputs is more than a pre-determined value, the model gets activated and shows the output value as +1.

If the outcome is same as pre-determined or threshold value, then the performance of this model is stated as satisfied, and weight demand does not change. However, this model consists of a few discrepancies triggered when multiple weight inputs values are fed into the model. Hence, to find desired output and minimize errors, some changes should be necessary for the weights input.

"Single-layer perceptron can learn only linearly separable patterns."

Multi-Layered Perceptron Model:

Like a single-layer perceptron model, a multi-layer perceptron model also has the same model structure but has a greater number of hidden layers.

The multi-layer perceptron model is also known as the Backpropagation algorithm, which executes in two stages as follows:

- **Forward Stage:** Activation functions start from the input layer in the forward stage and terminate on the output layer.
- **Backward Stage:** In the backward stage, weight and bias values are modified as per the model's requirement. In this stage, the error between actual output and demanded originated backward on the output layer and ended on the input layer.

Hence, a multi-layered perceptron model has considered as multiple artificial neural networks having various layers in which activation function does not remain linear, similar to a single layer perceptron model. Instead of linear, activation function can be executed as sigmoid, TanH, ReLU, etc., for deployment.

A multi-layer perceptron model has greater processing power and can process linear and non-linear patterns. Further, it can also implement logic gates such as AND, OR, XOR, NAND, NOT, XNOR, NOR.

Advantages of Multi-Layer Perceptron:

- A multi-layered perceptron model can be used to solve complex non-linear problems.
- It works well with both small and large input data.
- It helps us to obtain quick predictions after the training.
- It helps to obtain the same accuracy ratio with large as well as small data.

Disadvantages of Multi-Layer Perceptron:

- In Multi-layer perceptron, computations are difficult and time-consuming.
- In multi-layer Perceptron, it is difficult to predict how much the dependent variable affects each independent variable.
- The model functioning depends on the quality of the training.

Perceptron Function

Perceptron function "f(x)" can be achieved as output by multiplying the input 'x' with the learned weight coefficient 'w'.

Mathematically, we can express it as follows:

$f(x)=1$; if $w.x+b>0$

otherwise, $f(x)=0$

- 'w' represents real-valued weights vector
- 'b' represents the bias
- 'x' represents a vector of input x values.

Characteristics of Perceptron

The perceptron model has the following characteristics.

1. Perceptron is a machine learning algorithm for supervised learning of binary classifiers.
2. In Perceptron, the weight coefficient is automatically learned.
3. Initially, weights are multiplied with input features, and the decision is made whether the neuron is fired or not.
4. The activation function applies a step rule to check whether the weight function is greater than zero.
5. The linear decision boundary is drawn, enabling the distinction between the two linearly separable classes +1 and -1.
6. If the added sum of all input values is more than the threshold value, it must have an output signal; otherwise, no output will be shown.

Limitations of Perceptron Model

A perceptron model has limitations as follows:

- The output of a perceptron can only be a binary number (0 or 1) due to the hard limit transfer function.
- Perceptron can only be used to classify the linearly separable sets of input vectors. If input vectors are non-linear, it is not easy to classify them properly.

Future of Perceptron

The future of the Perceptron model is much bright and significant as it helps to interpret data by building intuitive patterns and applying them in the future. Machine learning is a rapidly growing technology of Artificial Intelligence that is continuously evolving and in the developing phase; hence the future of perceptron technology will continue to support and facilitate analytical behavior in machines that will, in turn, add to the efficiency of computers.

The perceptron model is continuously becoming more advanced and working efficiently on complex problems with the help of artificial neurons.

Conclusion:

In this article, you have learned how Perceptron models are the simplest type of artificial neural network which carries input and their weights, the sum of all weighted input, and an activation function. Perceptron models are continuously contributing to Artificial Intelligence and Machine Learning, and these models are becoming more advanced. Perceptron enables the computer to work more efficiently on complex problems using various Machine Learning technologies. The Perceptrons are the fundamentals of artificial neural networks, and everyone should have in-depth knowledge of perceptron models to study deep neural networks.

optimal separating hyperplane:

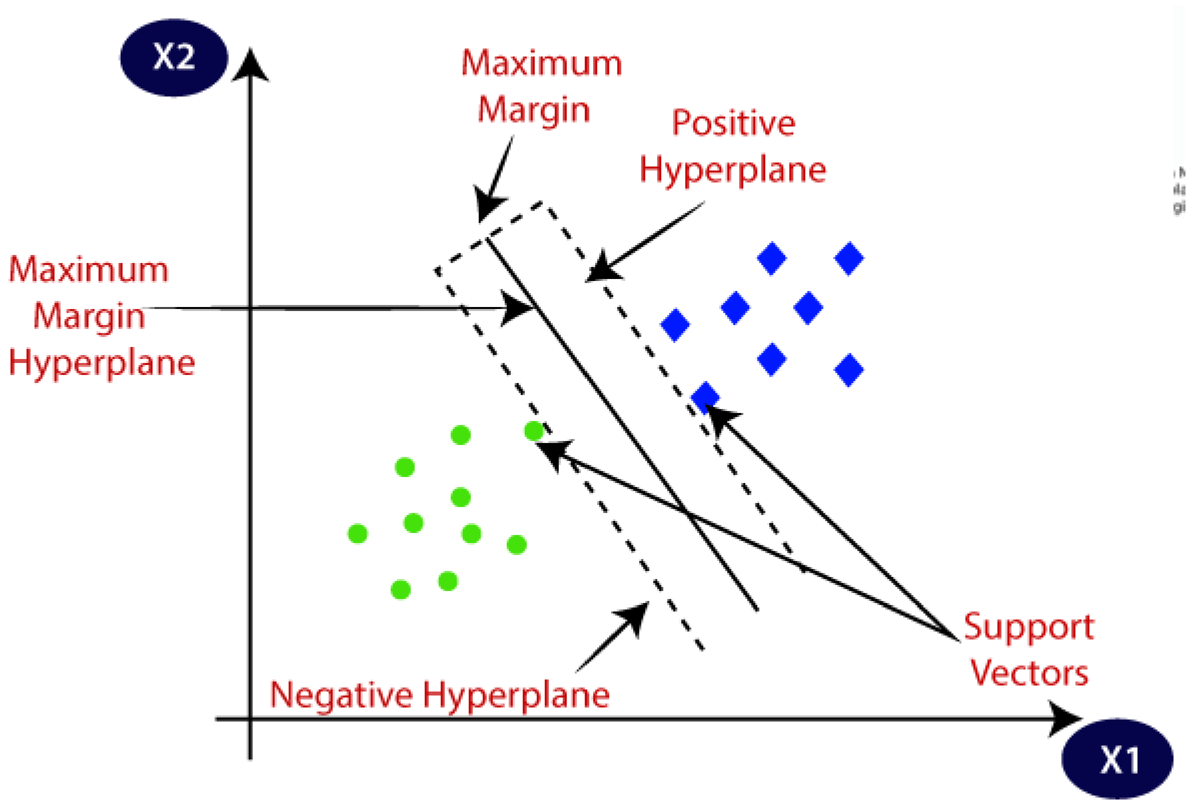
The optimal separating hyperplane is one of the core ideas behind the support vector machines

Support Vector Machine Algorithm

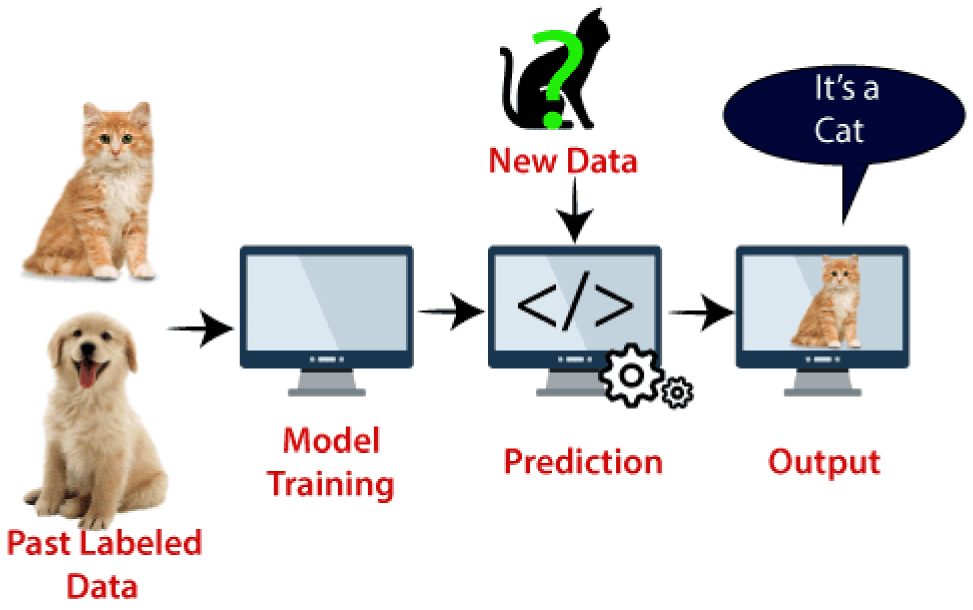
Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.

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.

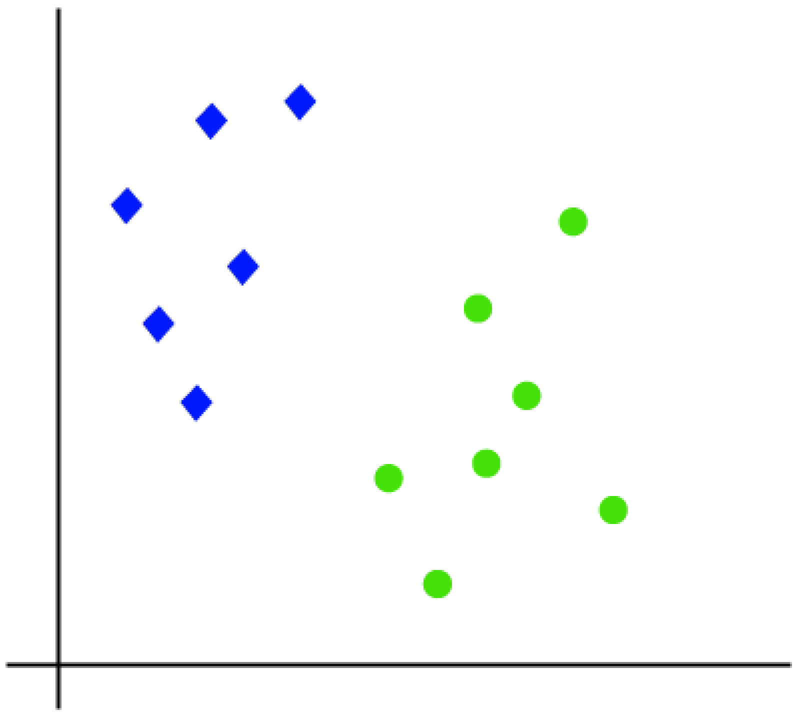
Support Vectors:

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.

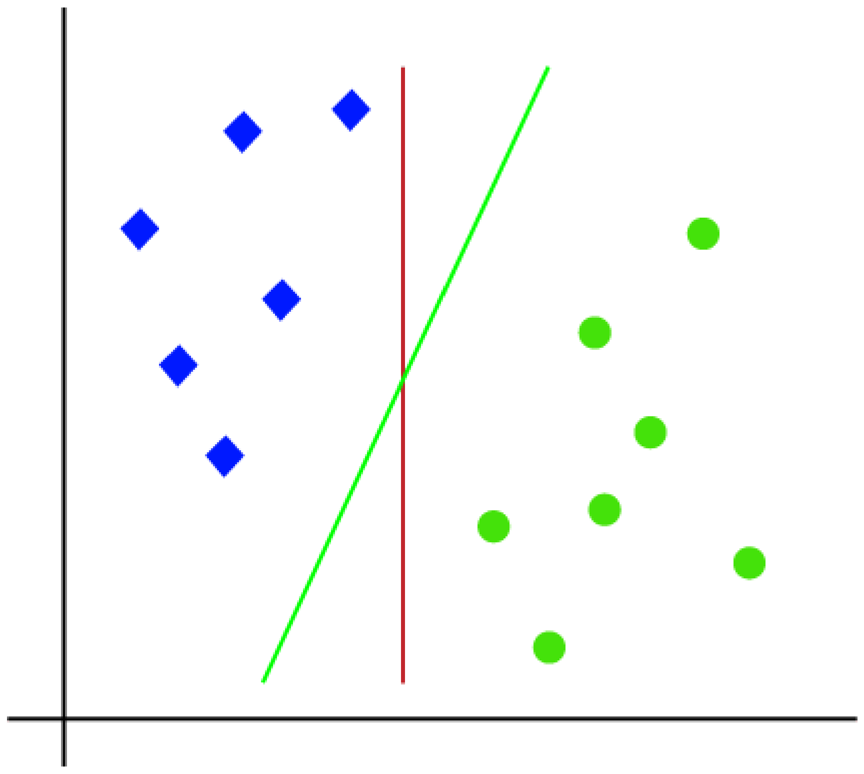
How does SVM works?

Linear SVM:

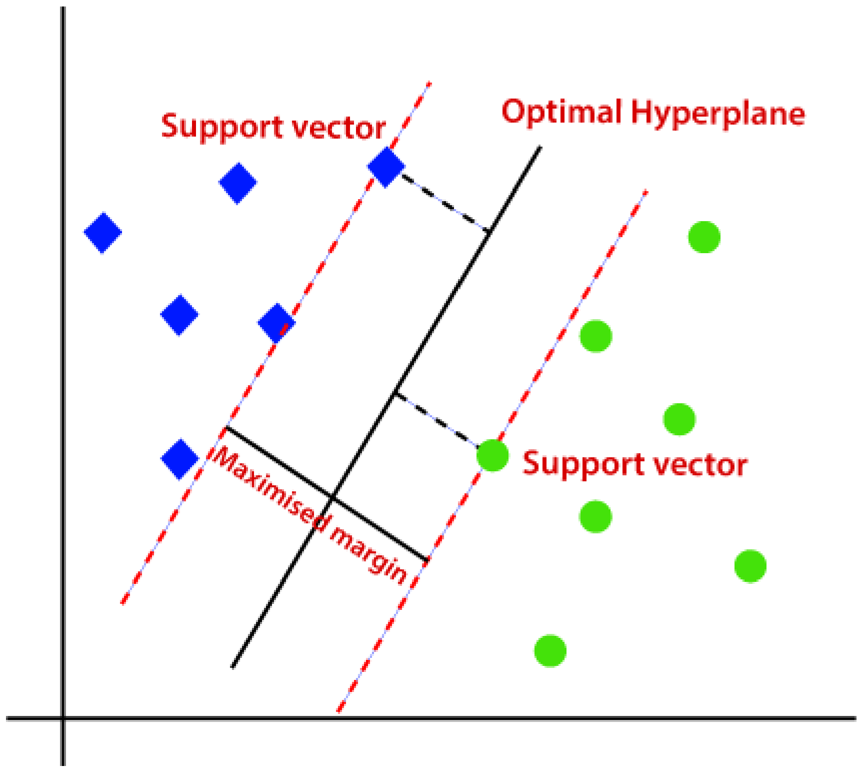
The working of the SVM algorithm can be understood by using an example. Suppose we have a dataset that has two tags (green and blue), and the dataset has two features x1 and x2. We want a classifier that can classify the pair(x1, x2) of coordinates in either green or blue. Consider the below image:



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:

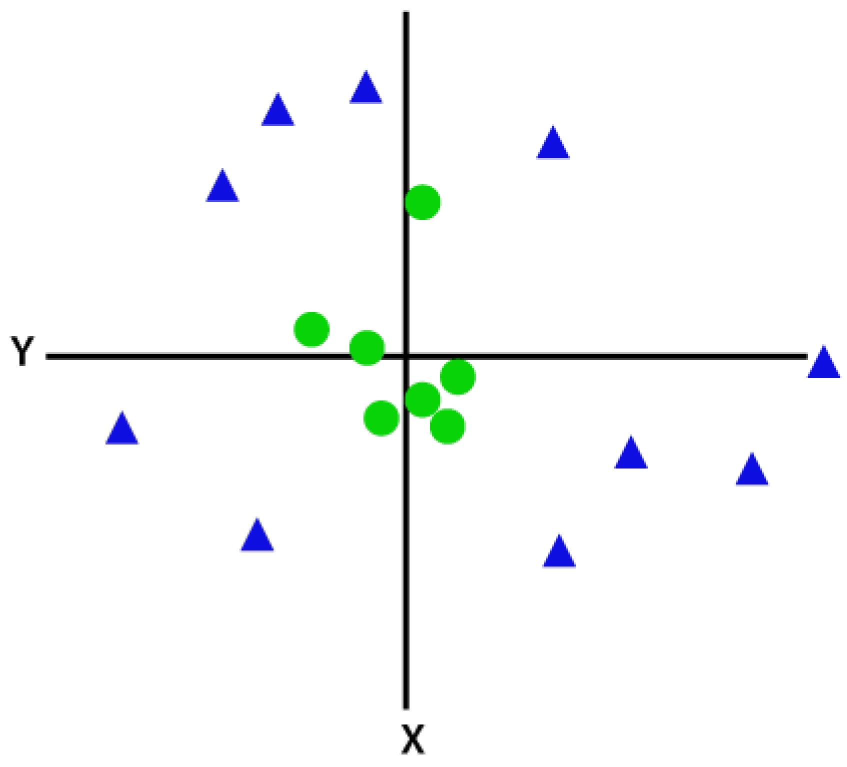


Hence, the SVM algorithm helps to find the best line or decision boundary; this best boundary or region is called as a **hyperplane**. SVM algorithm finds the closest point of the lines from both the classes. These points are called support vectors. The distance between the vectors and the hyperplane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

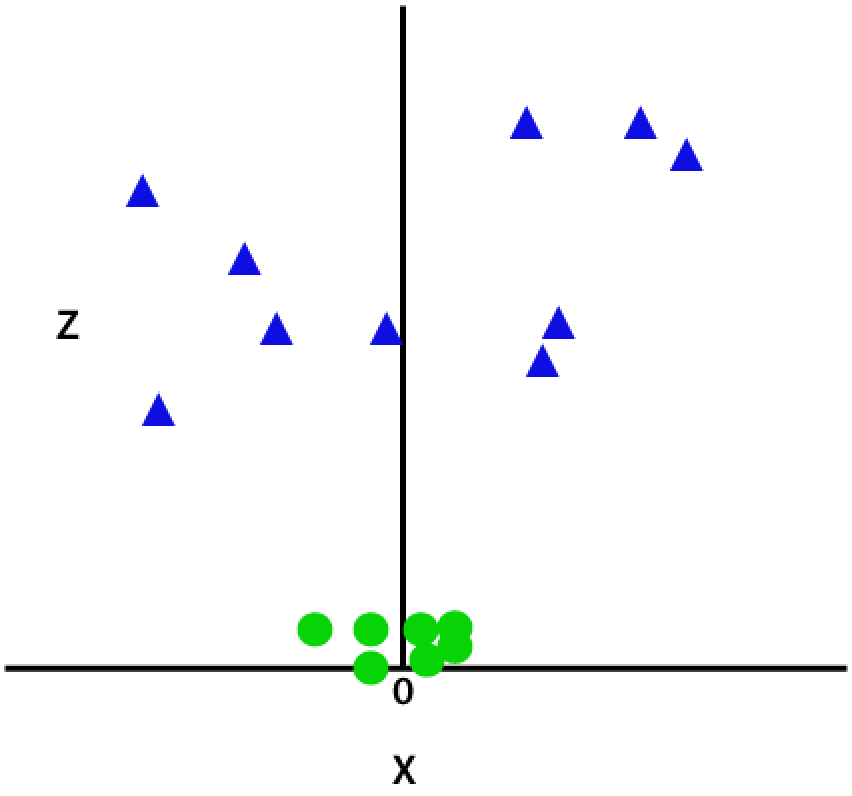
If data is linearly arranged, then we can separate it by using a straight line, but for non-linear data, we cannot draw a single straight line. Consider the below image:



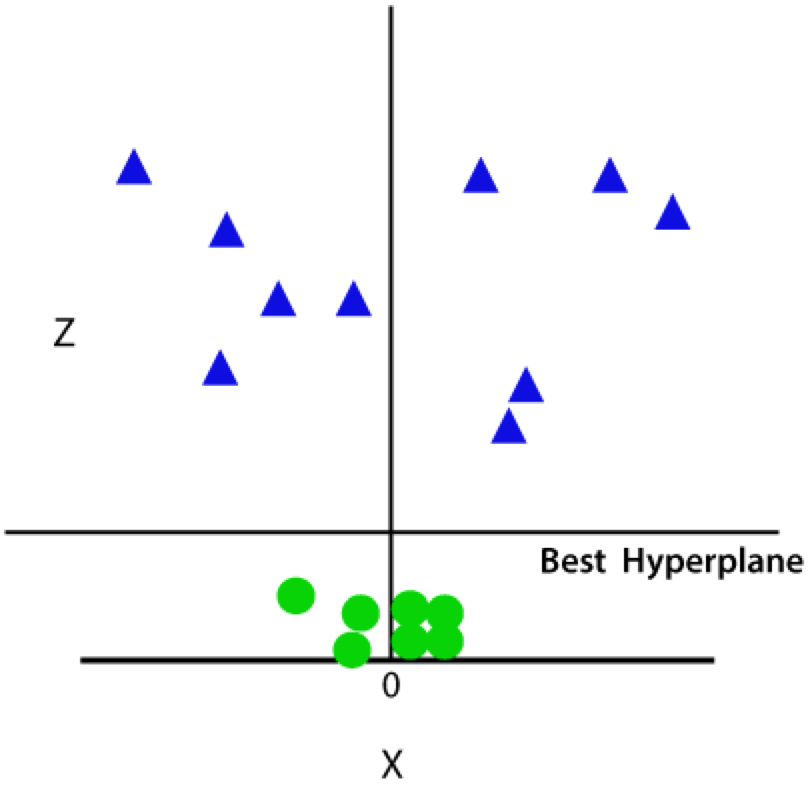
So to separate these data points, we need to add one more dimension. For linear data, we have used two dimensions x and y, so for non-linear data, we will add a third dimension z. It can be calculated as:

$z=x^2 +y^2$

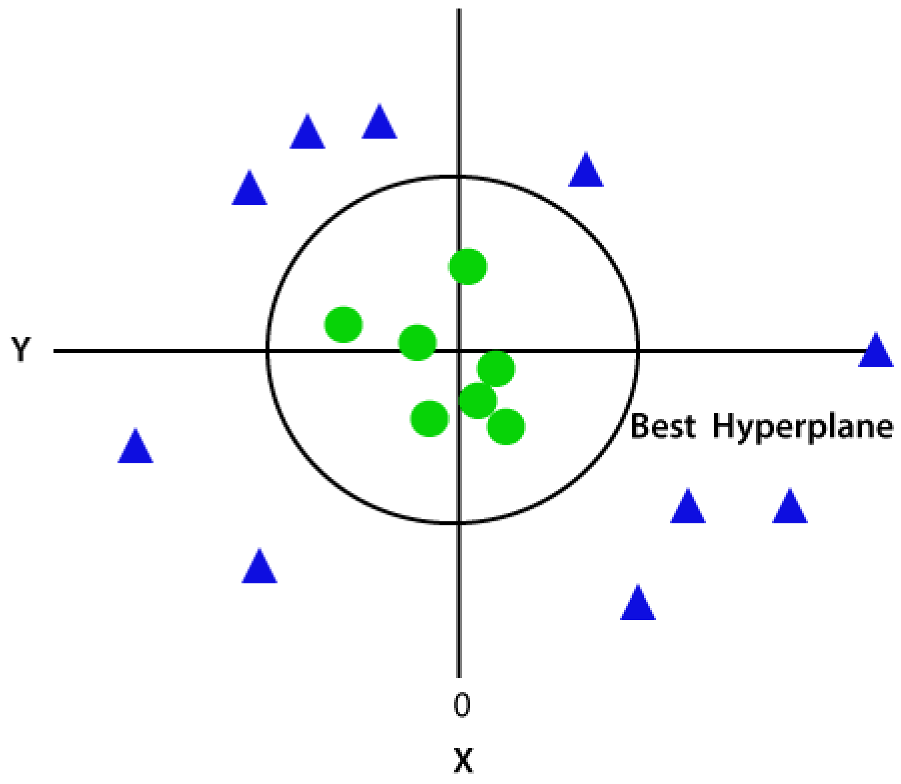
By adding the third dimension, the sample space will become as below image:



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.

Reference Links:

- https://medium.datadriveninvestor.com/the-basic-methods-for-classification-9c10a961b0ee
- https://www.javatpoint.com/perceptron-in-machine-learning
- https://www.javatpoint.com/machine-learning-support-vector-machine-algorithm