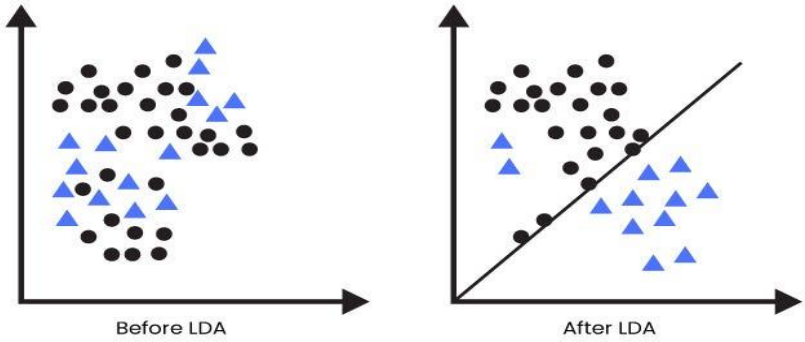
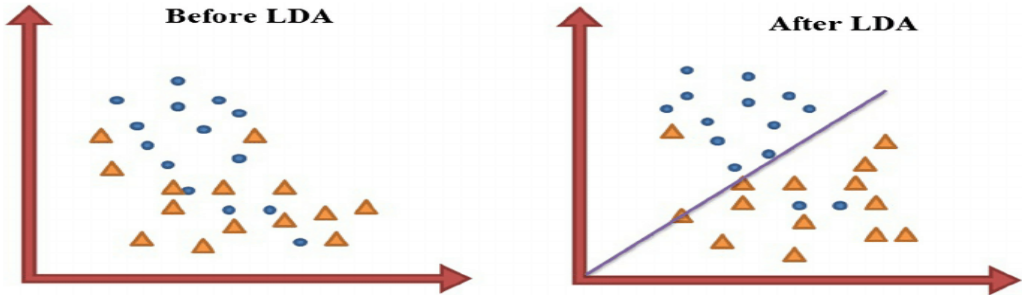
UNIT-IV

Linear Discriminants for Machine Learning: Introduction to Linear Discriminants, Linear Discriminants for Classification, Perceptron Classifier, Perceptron Learning Algorithm, Support Vector Machines, Linearly Non-Separable Case, Non-linear SVM, Kernel Trick, Logistic Regression, Linear Regression, Multi-Layer Perceptrons (MLPs), Back propagation for Training an MLP

Introduction to Linear Discriminants:



(OR)



Linear Discriminant Analysis (LDA) is a supervised machine learning technique used for both classification and dimensionality reduction

Key Concepts of LDA

- **Supervised Learning:** LDA requires labeled data, meaning the classes are already known.
- **Classification:** LDA identifies a decision boundary (a line or hyper plane) that separates different classes.

- **Dimensionality Reduction:** LDA can reduce the number of features while preserving the information necessary for classification.
- **Linear Combination of Features:** LDA finds a weighted sum of the original features to create a new set of features that best discriminate between classes.
- **Maximize Between-Class Separation:** LDA aims to maximize the distance between the means of different classes.
- **Minimize Within-Class Variance:** LDA also aims to minimize the variability of data points within each class.
- **Assumptions**: LDA assumes that data within each class follows a normal distribution with equal covariance matrices.

How LDA Works

1. **Calculate Statistics**: LDA first calculates the means and variances of each class.
2. **Compute Scatter Matrices:** It then computes within-class and between-class scatter matrices.
3. **Find Discriminant Vectors:** LDA finds eigenvectors of the matrix S (which is the inverse of the within-class scatter matrix multiplied by the between-class scatter matrix).
4. **Project Data:** The eigenvectors are used to project the data onto a lower-dimensional space, creating a new set of features.
5. **Classify:** The projected data can then be used to classify new data points based on their distance to the class means in the lower-dimensional space.

Applications of LDA

- **Pattern Recognition:** LDA is used to identify patterns and classify images or other data based on features.
- **Image Retrieval:** LDA can be used to find similar images based on their visual features.
- **Text Classification:** LDA can classify documents into different categories based on the words they contain.
- **Face Recognition:** LDA can be used to identify individuals based on their facial features.
- **Finance:** LDA can be used to classify financial events, such as predicting stock prices or identifying solvent/insolvent companies.

Linear Discriminants for Classification:

Linear discriminant analysis (LDA) is a supervised machine learning method for classification that projects data into a lower-dimensional space to maximize class reparability. It is closely related to **Fisher's Linear Discriminant (FLD)** and works well when classes are well-separated and normally distributed.

Key Concepts

Objective

- Find a linear combination of features that **separates different classes**.
- Project data onto a direction where the **between-class variance is maximized** relative to the **within-class variance**.

- **Assumptions**

- Normally distributed classes.
- Homoscedasticity (equal covariance matrices across classes).
- Independent predictors (if violated, consider regularization or QDA).

Mathematical Formulation

Mathematical Insight (Simplified)

LDA builds a function like:

$$y = w^T x + w_0$$

Where:

- x is the input (like feature values)
- w is the direction vector (how the features are combined)
- w_0 is a bias/threshold term
- y is the result — if it's closer to Class A's mean, classify as A; else, B.

So LDA turns your **multi-dimensional data** into a **1D number**, and based on that number, it makes a decision.

Perceptron Classifier:

The Perceptron **is one of the** simplest types of artificial neural networks**, introduced by** Frank Rosenblatt in 1958**. It's a** binary linear classifier

A perceptron is a linear classifier that:

- Takes multiple input features
- Applies weights to these inputs
- Produces a binary output based on a threshold function

Given:

- Input vector x=[x1,x2,...,xn]
- Weight vector w=[w1,w2,...,wn]
- Bias term b

The perceptron computes:

$z=w \cdot x+b$

Then applies the **activation function** (usually the **sign** function or step function):

$y\hat{}=\{1 \text{ if } z>0$
$0 \quad \text{otherwise}\}$

Sum and Activation

All weighted inputs are added up:

$$z = w_1x_1 + w_2x_2 + ... + w_nx_n + b$$

Then, we apply an **activation function** — in the original perceptron, it's just:

$$\text{output} = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{otherwise} \end{cases}$$

3. Geometric View

Imagine your data on a 2D graph — red and blue dots.

The perceptron tries to find a **straight line** that divides the two:

- One side = Class 0

- Other side = Class 1

That line is based on the weights and bias:

Learning Rule (Perceptron Algorithm)

1. Initialize weights w and bias b to small random values (or zeros).
2. For each training sample (x_i, y_i):
    - Predict: $\hat{y}_i = f(w \cdot x_i + b)$
    - Update if prediction is wrong:

      $w \leftarrow w + \eta(y_i - \hat{y}_i)x_i$

      $b \leftarrow b + \eta(y_i - \hat{y}_i)$

    Where η is the **learning rate**.

    Repeat this process over multiple epochs until convergence or a max number of iterations is reached.

Limitations

- Fails on **non-linearly separable** data (e.g., XOR problem)
- Only handles **binary classification** (extensions exist for multiclass)

Perceptron Learning Algorithm:

The Perceptron Learning Algorithm is an **online learning** method for training the simplest type of artificial neuron - the **perceptron**. It's a **binary linear classifier** that learns weights to separate two classes

(OR)

The Perceptron is a type of artificial neuron and the foundation of neural networks. It's a supervised learning algorithm used for binary classification

Structure of a Perceptron

A perceptron consists of:

- **Inputs**: Features or attributes of the data (e.g., weather conditions, email content).
- **Weights**: Each input has a corresponding weight that signifies its importance.

- **Bias**: A constant value added to the weighted sum, helping shift the decision boundary.
- **Activation Function**: Determines the output — usually a **sign function** that outputs +1 or -1.

Mathematical Expression:

$$y = \text{sign}(w \cdot x + b)$$

Where:

- x = input vector (features)
- w= weight vector
- b= bias
- y= output (either +1 or -1)
- sign = activation function that outputs 1 if the result is positive, else -1

Steps of the Perceptron Learning Algorithm

1. **Initialize** weights and bias (usually to zero or small random values).
2. For each input sample:
    - Calculate output using the current weights and bias.
    - If the output is correct, do nothing.
    - If incorrect, **update the weights and bias** using the learning rule.
3. Repeat the process for all training samples.
4. Continue iterating until:

    All samples are correctly classified, or
    A maximum number of iterations is reached

Advantages

- Simple and fast
- Easy to implement
- Good for linearly separable problems
- Forms the foundation of more complex neural networks

Limitations

- Works **only for binary classification**

- Can’t solve **non-linearly separable problems** (e.g., XOR problem)
- Doesn’t work well on **noisy data**
- Not suitable for **multi-class** classification unless extended

Support Vector Machines:

Support Vector Machine (SVM) is a supervised machine learning algorithm used for classification and regression tasks. While it can handle regression problems, SVM is particularly well-suited for classification tasks.
SVM aims to find the optimal hyper plane in an N-dimensional space to separate data points into different classes. The algorithm maximizes the margin between the closest points of different classes.

What is a hyper plane?

A **hyper plane** is just a fancy term for a decision boundary.

- In 2D → hyper plane = **line**
- In 3D → hyper plane = **plane**
- In more dimensions → still called a **hyper plane**

A hyper plane separates the space into two halves. SVM tries to find the best one that cleanly separates the classes.

Support Vector Machine (SVM) Terminology

- **Hyper plane**: A decision boundary separating different classes in feature space, represented by the equation **wx + b = 0** in linear classification.
- **Support Vectors**: The closest data points to the hyper plane, crucial for determining the hyper plane and margin in SVM
- **Margin**: The distance between the hyper plane and the support vectors. SVM aims to maximize this margin for better classification performance.
- **Kernel**: A function that maps data to a higher-dimensional space, enabling SVM to handle non-linearly separable data.

- **Hard Margin**: A maximum-margin hyper plane that perfectly separates the data without misclassifications.

- **Soft Margin**: Allows some misclassifications by introducing slack variables, balancing margin maximization and misclassification penalties when data is not perfectly separable.
- **C**: A regularization term balancing margin maximization and misclassification penalties. A higher C value enforces a stricter penalty for misclassifications.

- **Hinge Loss**: A loss function penalizing misclassified points or margin violations, combined with regularization in SVM.

x1	x2	Output
0	0	0
0	1	1
1	0	1
1	1	0

No straight line can separate this perfectly.

Solution: Use the Kernel Trick!

- **Kernel functions** allow SVM to project data into a **higher-dimensional space**.
- In that higher space, the data can **become linearly separable**.

Think of it like:

- In 2D, circles can't be separated by a line.
- But if you lift the circles into 3D, you can separate them with a plane!

---

7. Kernel Functions

Kernel = mathematical trick to compute in higher dimensions without explicitly transforming the data.

Popular kernels:

- **Linear Kernel**: Good for simple problems.
- **Polynomial Kernel**: Good for curved decision boundaries.
- **Radial Basis Function (RBF) or Gaussian Kernel**: Good for very complex boundaries.

RBF kernel formula:

$$K(x_i, x_j) = \exp\left(-\gamma \|x_i - x_j\|^2\right)$$

γ = controls how far the influence of a single training example reaches.

Advantages of SVM

Advantage	Why it Matters
High-dimensional effectiveness	Works well even with a lot of features
Robust margin	Good generalization to new, unseen data
Versatile	Works with different kernels for non-linear problems
Few support vectors	Model complexity depends on support vectors, not dataset size

Disadvantages of SVM

Limitation	Problem
Computationally expensive	Slow for very large datasets
Hard to choose kernel and parameters	Needs experience or cross-validation
Not good for overlapping classes	When classes mix a lot, SVM struggles

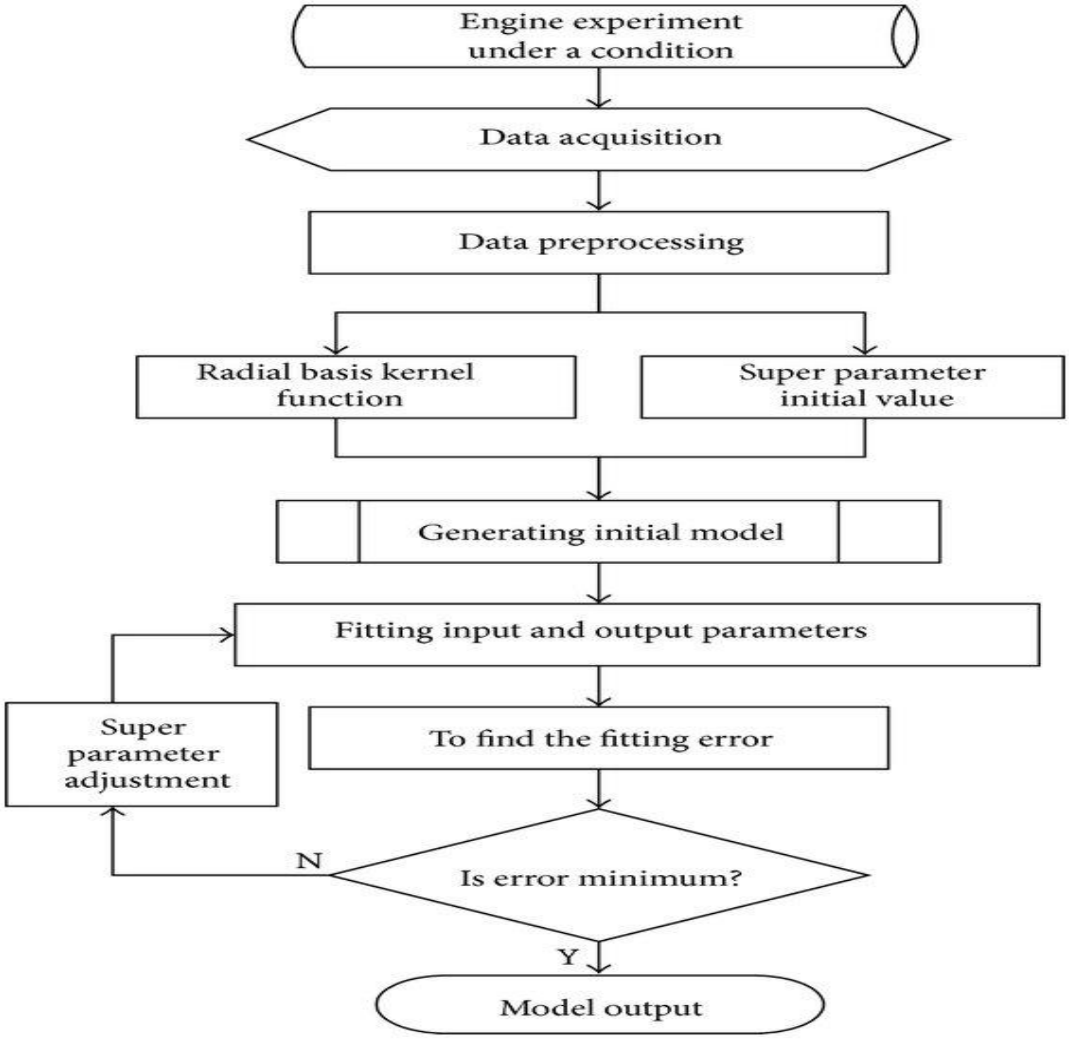
Real-Life Applications of SVM

- **Face Detection** (classify part of an image as face or not)
- **Spam Detection** (classify emails as spam or not)
- **Image Classification** (cat vs dog, etc.)

Limitation **Problem**

- **Bioinformatics** (classifying genes, proteins)
- **Handwriting Recognition** (digit classification

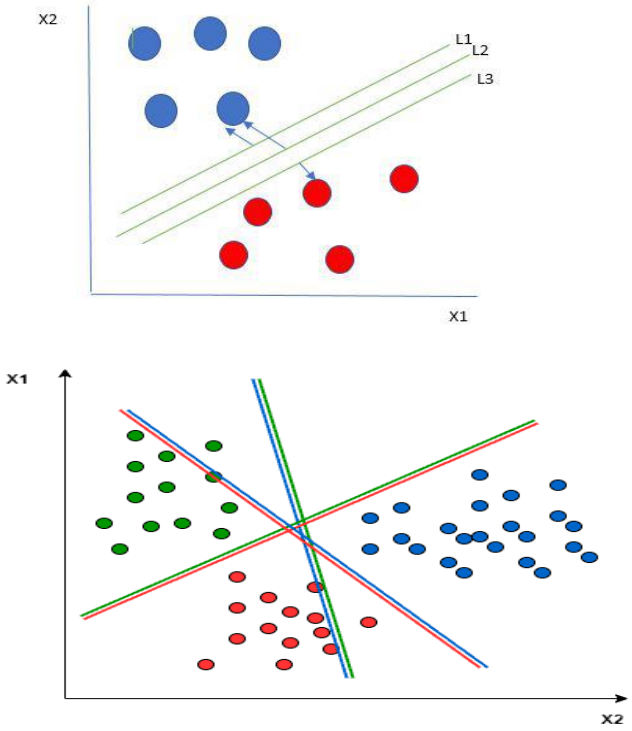
SVM FLOW CHART



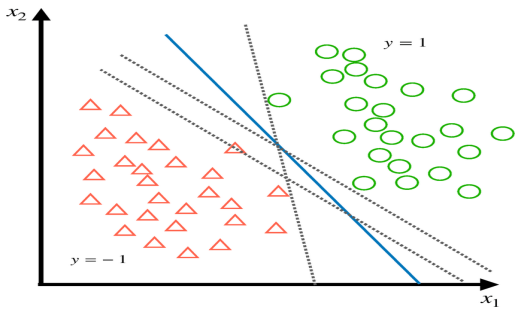
- **Dual Problem**: Involves solving for Lagrange multipliers associated with support vectors, facilitating the kernel trick and efficient computation.

How does Support Vector Machine Algorithm Work?

The key idea behind the SVM algorithm is to find the hyper plane that best separates two classes by maximizing the margin between them. This margin is the distance from the hyper plane to the nearest data points (**support vectors**) on each side.

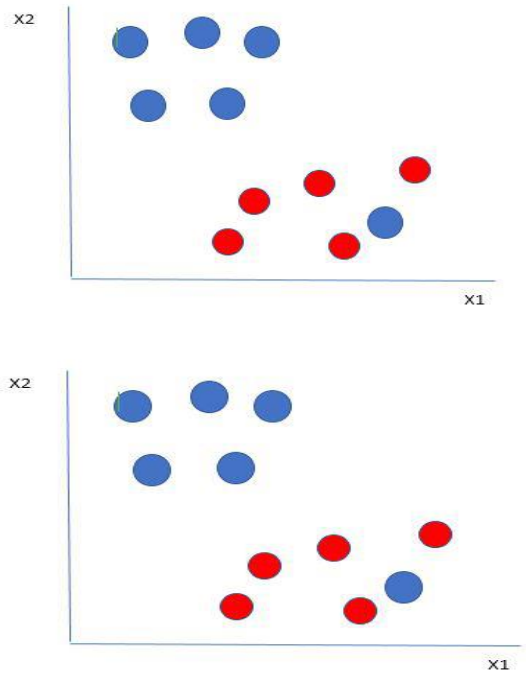


Multiple hyper planes separate the data from two classes



The best hyper plane, also known as the **"hard margin,"** is the one that maximizes the distance between the hyper plane and the nearest data points from both classes. This ensures a clear separation between the classes. So, from the above figure, we choose L2 as hard margin.

Let's consider a scenario like shown below:

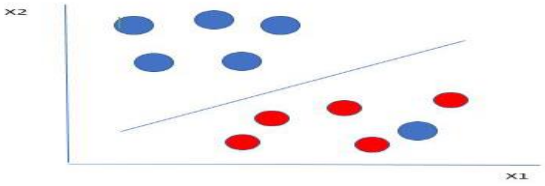


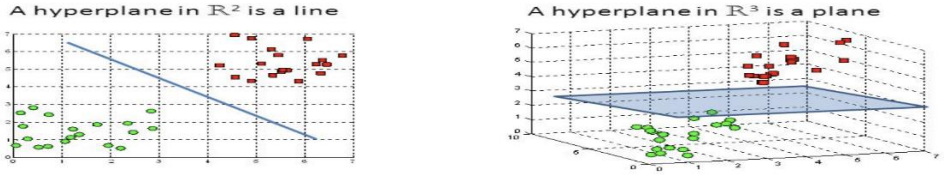
Selecting hyper plane for data with outlier

Here, we have one blue ball in the boundary of the red ball.

How does SVM classify the data

It's simple! The blue ball in the boundary of red ones is an outlier of blue balls. The SVM algorithm has the characteristics to ignore the outlier and finds the best hyper plane that maximizes the margin. SVM is robust to outliers.





Hyper plane which is the most optimized one

A soft margin allows for some misclassifications or violations of the margin to improve generalization. The SVM optimizes the following equation to balance margin maximization and penalty minimization:

$$\text{Objective Function} = (1/\text{margin}) + \lambda \sum \text{penalty}$$

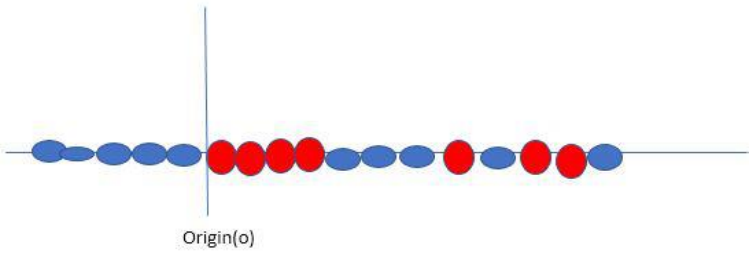
The penalty used for violations is often **hinge loss**, which has the following behavior:

- If a data point is correctly classified and within the margin, there is no penalty (loss = 0).
- If a point is incorrectly classified or violates the margin, the hinge loss increases proportionally to the distance of the violation.

Till now, we were talking about linearly separable data (the group of blue balls and red balls are separable by a straight line/linear line).

What to do if data are not linearly separable?

When data is not linearly separable (i.e., it can't be divided by a straight line), SVM uses a technique called **kernels** to map the data into a higher-dimensional space where it becomes separable. This transformation helps SVM find a decision boundary even for non-linear data.

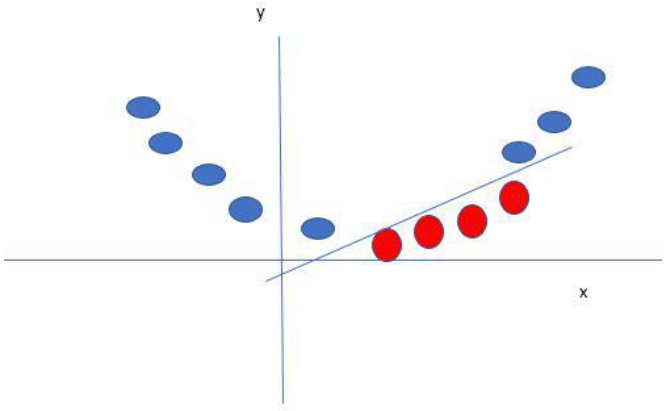


Original 1D dataset for classification

A **kernel** is a function that maps data points into a higher-dimensional space without explicitly computing the coordinates in that space. This allows SVM to work efficiently with non-linear data by implicitly performing the mapping.

For example, consider data points that are not linearly separable. By applying a kernel function, SVM transforms the data points into a higher-dimensional space where they become linearly separable.

- **Linear Kernel**: For linear reparability.
- **Polynomial Kernel**: Maps data into a polynomial space.
- **Radial Basis Function (RBF) Kernel**: Transforms data into a space based on distances between data points.



Mapping 1D data to 2D to become able to separate the two classes

In this case, the new variable y is created as a function of distance from the origin.
Mathematical Computation: SVM
Consider a binary classification problem with two classes, labeled as +1 and -1. We have a training dataset consisting of input feature vectors X and their corresponding class labels Y.

The equation for the linear hyper plane can be written as:

$$w^Tx+b=0 w^Tx+b=0$$

Where:

- w is the normal vector to the hyper plane (the direction perpendicular to it).
- b is the offset or bias term, representing the distance of the hyper plane from the origin along the normal vector w.

Distance from a Data Point to the Hyper plane
The distance between a data point x_i and the decision boundary can be calculated as:
$d_i=w^Tx_i+b/||w||$
where ||w|| represents the Euclidean norm of the weight vector w. Euclidean norm of the normal vector W

Linear SVM Classifier
Distance from a Data Point to the Hyper plane:
$\hat{y}=\{1: w^Tx+b\geq 0$
$0: w^Tx+b<0$
Where $\hat{y}$ is the predicted label of a data point.
Optimization Problem for SVM

For a linearly separable dataset, the goal is to find the hyperplane that maximizes the margin between the two classes while ensuring that all data points are correctly classified. This leads to the following optimization problem

$$\underset{w,b}{\text{minimize}} \frac{1}{2}\|w\|^2$$

Subject to the constraint:

$$y_i(w^T x_i + b) \geq 1 \text{ for } i = 1, 2, 3, \cdots, m$$

Where:

- y_i is the class label (+1 or -1) for each training instance.
- x_i is the feature vector for the i-th training instance.
- m is the total number of training instances.

The condition $y_i(w^T x_i + b) \geq 1$ ensures that each data point is correctly classified and lies outside the margin.

Advantages of Support Vector Machine (SVM)

1. **High-Dimensional Performance**: SVM excels in high-dimensional spaces, making it suitable for **image classification** and **gene expression analysis**.
2. **Nonlinear Capability**: Utilizing **kernel functions** like **RBF** and **polynomial**, SVM effectively handles **nonlinear relationships**.
3. **Outlier Resilience**: The **soft margin** feature allows SVM to ignore outliers, enhancing robustness in **spam detection** and **anomaly detection**.
4. **Binary and Multiclass Support**: SVM is effective for both **binary classification** and **multiclass classification**, suitable for applications in **text classification**.
5. **Memory Efficiency**: SVM focuses on **support vectors**, making it memory efficient compared to other algorithms.

Disadvantages of Support Vector Machine (SVM)

1. **Slow Training**: SVM can be slow for large datasets, affecting performance in **SVM in data mining** tasks.
2. **Parameter Tuning Difficulty**: Selecting the right **kernel** and a-djusting parameters like **C** requires careful tuning, impacting **SVM algorithms**.
3. **Noise Sensitivity**: SVM struggles with noisy datasets and overlapping classes, limiting effectiveness in real-world scenarios.
4. **Limited Interpretability**: The complexity of the **hyper plane** in higher dimensions makes SVM less interpretable than other models.
5. **Feature Scaling Sensitivity**: Proper **feature scaling** is essential; otherwise, SVM models may perform poorly.

Applications

- **Image Recognition:**

  SVMs can be used for tasks like object detection, handwritten digit recognition, and optical character recognition.

- **Text Classification:**

  SVMs are effective for tasks like spam detection, sentiment analysis, and topic categorization.

- **Bioinformatics:**

  SVMs can be used for protein structure prediction, gene expression analysis, and DNA classification.

- **Financial Forecasting:**

  SVMs can be used for stock market prediction, credit scoring, and fraud detection.

- **Medical Diagnosis:**
  SVMs can be used to assist in diagnosing diseases and predicting patient outcomes.

LINEAR NON-SEPRABLE CASE NON LINEAR SVM

Support Vector Machines (SVMs) are powerful supervised learning models, but their standard **linear SVM** form fails when data is **not linearly separable**. Here's how SVMs handle such cases using **soft margins** and **kernel tricks**

Why Linear SVM Fails for Non-Separable Data?

- **Linear SVM** assumes a **hyper plane** can perfectly separate classes.
- **Real-world data** often has:
  - Overlapping class distributions (noise).
  - Complex (non-linear) decision boundaries (e.g., spirals, concentric circles).

Example: XOR Problem

No straight line can separate these points:

x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0

Solution 1: Soft Margin SVM (Allowing Misclassifications)

When data is **slightly non-separable** (e.g., due to noise), we use **soft margin SVM**:

- Introduces **slack variables (ξ)** to permit some misclassifications.
- Controlled by hyper parameter **C** (penalty for misclassification):
  - **Small C** → wider margin, more misclassifications allowed (under fitting).
  - **Large C** → narrower margin, fewer misclassifications (over fitting).

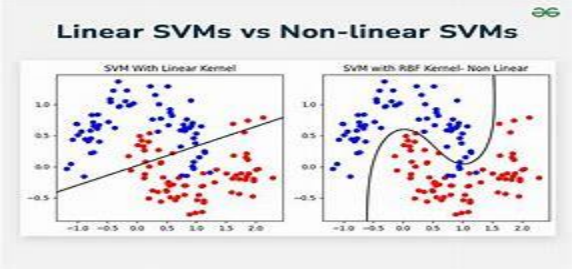
- **Optimization Problem:**
- $\min 1/2\|w\|^2+C\sum_{i=1}^{n}\xi_i$
- Subject to:
- $y_i(w^T x_i+b)\geq 1-\xi_i$ and $\xi_i\geq 0$

Solution 2: Kernel Trick (Handling Non-Linearity)

For **highly non-linear data**, we map features to a higher-dimensional space where they become separable.
Kernel functions compute dot products in this space **without explicit transformation**.

Common Kernels

Kernel	Formula	Use Case
Linear	$K(x_i,x_j)=x_i^T x_j$	Already separable.
Polynomial	$(\gamma x_i^T x_j+r)^d$	Moderate non-linearity.
RBF (Gaussian)	$\exp(-\gamma\|x_i-x_j\|^2)$	Highly non-linear (default).
Sigmoid	$\tanh(\gamma x_i^T x_j+r)$	Rarely used.



Kernel Trick:

The kernel trick is a fundamental concept in Support Vector Machines (SVM) and other machine learning algorithms. It allows us to solve complex problems by transforming the input data into a higher-dimensional space without actually computing the transformation explicitly. This enables SVMs to classify data that is not linearly separable in its original space.

Instead of computing $\phi(x)\cdot\phi(x')$ (dot product in high-dimensional space), we compute a kernel function $K(x,x')$, which gives us the same result — but way faster and more efficient.

Why Do We Need the Kernel Trick?

SVMs are **linear classifiers** by default. They find a hyperplane that best separates the classes. But what if the data isn't linearly separable?

Example:

Imagine trying to separate two classes shaped like concentric circles. A straight line won't work. But if we could somehow **transform** the data into a higher dimension, where the classes become linearly separable, we could apply a linear SVM there

What Is the Kernel Trick?

The kernel trick is a **mathematical shortcut**. It avoids the **explicit mapping** of data into higher dimensions. Instead, it uses a **kernel function** that computes the **dot product** of the data in the transformed space **directly from the original input space**.

Mathematically:

- Suppose $\phi(x)$ maps input xxx into a higher-dimensional space.
- Instead of computing $\phi(x)$, we use a kernel function:

$$K(x, y) = \phi(x)^T \phi(y)$$

So, we compute the inner product in the transformed space **without ever computing $\phi(x)$** This is the **kernel trick**.

Common Kernel Functions

1. **Linear Kernel**:

$$K(x, y) = x^T y$$

   - (No transformation; same as original space)
2. **Polynomial Kernel**:

$$K(x, y) = (x^T y + c)^d$$

   - (Transforms into polynomial space of degree d)
3. **Radial Basis Function (RBF) or Gaussian Kernel**:

$$K(x, y) = \exp\left(-\frac{\|x - y\|^2}{2\sigma^2}\right)$$

- (Maps to infinite-dimensional space; great for non-linear problems)

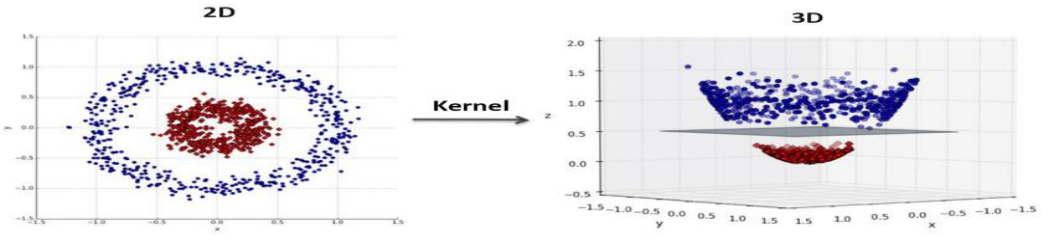
4. **Sigmoid Kernel**:

$$K(x, y) = \tanh(\alpha x^T y + c)$$

- (Related to neural networks)

Benefits of the Kernel Trick

- Handles **non-linearly separable data** effectively.
- Avoids the **computational burden** of working in high-dimensional spaces.
- Enables SVMs to model **complex decision boundaries**.



Common Kernel Functions

Name	Formula	Behavior
Linear	$K(x,x')=x^Tx'$	No trick — just a regular dot product
Polynomial	$K(x,x')=(x^Tx'+c)^d$	Adds interaction terms, curves, etc.
RBF (Gaussian)	$K(x,x')=\exp(-\gamma\|x-x'\|^2)$	Maps to **infinite-dimensional** space
Sigmoid	$K(x,x')=\tanh(\alpha x^Tx'+c)$	Like a neural network activation

Logistic Regression:

Logistic Regression is a **supervised learning algorithm** used for **binary classification** (though it can be extended to multiclass). Unlike linear regression, which predicts continuous values, logistic regression predicts **probabilities** (between 0 and 1) using a **logistic (sigmoid) function**

Logistic Regression models the probability that an input belongs to a particular class using the logistic (sigmoid) function:

$$\sigma(z) = 1/1+e^{-z}$$

Where:

- $z=w^Tx+b$www are the weights
- x is the input feature vector
- b is the bias

This squashes the output to a range between **0 and 1**, which is interpreted as a **probability**.

Decision Rule

If the probability $P(y=1|x)$ is:

- $\geq 0.5 \rightarrow$ predict **class 1**
- $< 0.5 \rightarrow$ predict **class 0**

You can change the threshold depending on the use case (e.g., 0.7 for stricter classification).

We minimize this loss over the dataset to train the model.

Use Cases

- Email spam detection
- Customer churn prediction
- Medical diagnosis (e.g., disease present or not)
- Credit scoring

Linear Regression:

Linear Regression is a **supervised learning algorithm** used for predicting **continuous numerical values** based on input features. It assumes a **linear relationship** between the independent variables (features) and the dependent variable (target).

Why Use Linear Regression?

- Predicts numerical outcomes (e.g., house prices, stock values, sales forecasts).
- Simple, interpretable, and computationally efficient.
- Foundation for many advanced ML techniques.

Assumptions

1. **Linearity**: Relationship between features and target is linear.
2. **Homoscedasticity**: Residuals (errors) have constant variance.
3. **No Multicollinearity**: Features are not highly correlated.
4. **Normality of Residuals**: Errors are normally distributed (for inference).

Simple Linear Regression (1 Feature)

$y=w_1x+b$

y = predicted output

- x = input feature
- w_1 = slope (weight)
- b= y-intercept (bias)

Multiple Linear Regression (Many Features)

$y=w_1x_1+w_2x_2+\cdots+w_nx_n+b$

$x_1,x_2,\ldots,x_n$ = input features

- $w_1,w_2,\ldots,w_n$ = weights
- b = bias term

Matrix Form

$y=X_w+b$

where:

- X = feature matrix
- w = weight vector

Cost Function (Mean Squared Error)

Linear regression minimizes the **Mean Squared Error (MSE)**:

$$J(w,b)=1/N\sum_{i=1}^{N}(yi-y\hat{}_i)^2$$

where:

- N = number of samples
- y_i = true value
- $y\hat{}_i$= predicted value

Applications of Logistic Regression:

Logistic regression is widely used in various fields for binary classification tasks, including:

- **Healthcare:** Predicting the likelihood of a disease (e.g., diabetes, ❤ heart disease) based on patient characteristics.
- **Finance:** 💰 Credit scoring (predicting loan default 🏦), fraud detection 🕵.
- **Marketing:** 📣 Customer churn prediction 🚪➡, predicting the likelihood of a customer purchasing a product 🛍.
- **Natural Language Processing (NLP):** 🗣 Sentiment analysis (☺/☹), spam detection 📧➡🗑.
- **Ecology:** 🌳 Predicting the presence or absence of a species in a given habitat 🐾.
- **Social Sciences:** □□□ Predicting voting behavior 🗳, participation in programs.

Types of Logistic Regression:

- **Binary Logistic Regression:** The dependent variable has two categories. This is the most common type. ✌
- **Multinomial Logistic Regression:** The dependent variable has three or more unordered categories (e.g., predicting the type of movie a person will watch 🎬🍿).
- **Ordinal Logistic Regression:** The dependent variable has three or more ordered categories (e.g., customer satisfaction levels: low ☹, medium 😐, high 😊).

Multi-Layer Perceptron's (MLPs)

Multi-Layer Perceptron's (MLPs) are a class of **feed forward artificial neural networks** (ANNs) that consist of multiple layers of interconnected neurons. They are capable of learning **non-linear decision boundaries** and are widely used for **classification and regression tasks**.

Key Concepts

What is an MLP?

- A **fully connected** neural network with:
  - **Input layer** (receives features).
  - **Hidden layers** (process data non-linearly).
  - **Output layer** (produces predictions).
- Uses **non-linear activation functions** (e.g., ReLU, Sigmoid, Tang).

Why Use MLPs

Can model **complex non-linear relationships**.
Works well for **structured data** (tabular, numerical, and categorical).
More powerful than linear models (Logistic Regression, Linear Regression).

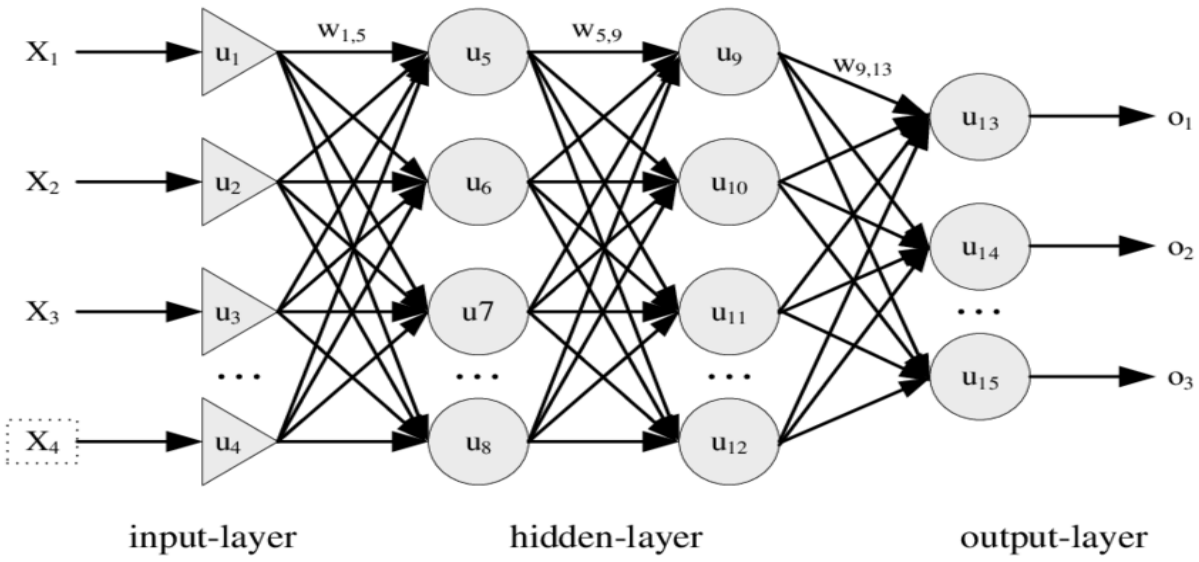
Limitations

Requires **large datasets** (prone to over fitting on small data).

Computationally expensive compared to simpler models.

Harder to interpret (black-box nature).

Architecture of an MLP



Layers in an MLP

1. **Input Layer** (size = number of features).
2. **Hidden Layers** (1 or more, each with multiple neurons).
3. **Output Layer** (size depends on the task):

- **Binary Classification**: 1 neuron (Sigmoid).
- **Multiclass Classification**: N neurons (Softmax).
- **Regression**: 1 neuron (Linear).

Neuron Computation

Each neuron computes:

$z=w^Tx+b$

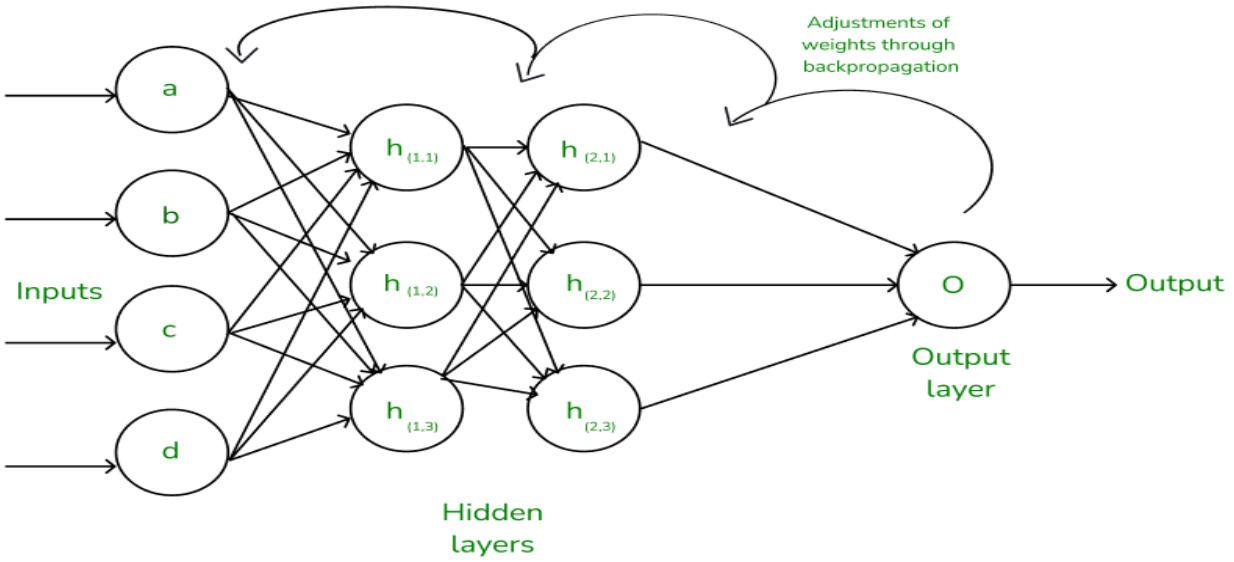
Where:

- w = weights,
- b*b* = bias.
- *f* = activation function (e.g., ReLU, Sigmoid).

Activation Functions

Activation	Formula	Pros & Cons
ReLU	$f(z)=\max(0,z)$	Fast, avoids vanishing gradient (but "dying ReLU" problem).
Sigmoid	$f(z)=1/1+e^{-z}$	Outputs probabilities (0 to 1), but suffers from vanishing gradients.
Tanh	$f(z)=e^{z}-e^{-z}/e^{z}+e^{-z}$	Zero-centered (-1 to 1), but slower than ReLU.
Softmax	$f(z_i)=e^{zi}\sum_j e^{zj}$	Used in **multiclass output layers**.

Back propagation for Training an MLP:



Back propagation (Backward Propagation of Errors) is a supervised learning algorithm used to train **Multilayer Perceptron's (MLPs)** by efficiently computing gradients of the loss function with respect to each weight in the network. Here's a step-by-step breakdown

Propagate errors backward to compute gradients using the chain rule.

Output Layer Gradients ($l=L$):

$\delta^{(L)}=\partial L/\partial z^{(L)}=\partial L/\partial a^{(L)}\odot\sigma_{out}'(z^{(L)})$

For MSE loss + linear output: $\delta^{(L)}=a^{(L)}-y$

For cross-entropy + softmax: $\delta^{(L)}=a^{(L)}-y$ (derivative simplifies).

Weight/Bias Gradients:

$\partial L/\partial W^{(l)}=\delta^{(l)}$

Hidden Layer Gradients ($l=L-1,\ldots,1$):

$\delta^{(l)}=\delta_{(l)}=$

- σ' is the derivative of the activation function (e.g., ReLU: $I(z>0)$, sigmoid: $\sigma(z)(1-\sigma(z))$.

Weight/Bias Gradients:

$\partial L/\partial W^{(l)}=\delta^{(l)}$

Key Concepts in Back propagation:

- **Chain Rule:** The fundamental calculus rule that allows the calculation of gradients through composite functions (like the layers of a neural network).

- **Gradients:** Indicate the direction of the steepest increase in the loss function. By moving in the opposite direction of the gradient, we aim to minimize the loss.
-
- **Learning Rate:** A crucial hyper parameter that determines the size of the steps taken during weight and bias updates. A too-high learning rate can lead to instability, while a too-low learning rate can result in slow convergence.

  Optimization Algorithms: Algorithms like Stochastic Gradient Descent (SGD), Adam, RMSprop, etc., use the gradients calculated by back propagation to efficiently update the network's parameters