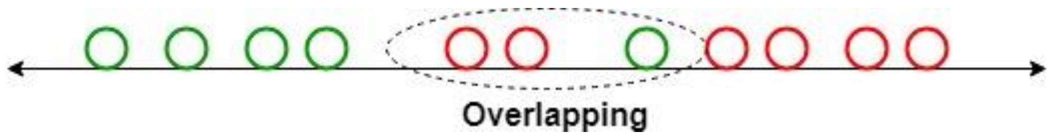
Machine Learning

4. Dimensionality Reduction and Probabilistic Learning

Linear Discriminant Analysis (LDA)

Linear Discriminant Analysis (LDA), also known as Normal Discriminant Analysis or Discriminant Function Analysis, is a dimensionality reduction technique primarily utilized in supervised classification problems. It facilitates the modeling of distinctions between groups, effectively separating two or more classes. LDA operates by projecting features from a higher-dimensional space into a lower-dimensional one. In machine learning, LDA serves as a supervised learning algorithm specifically designed for classification tasks, aiming to identify a linear combination of features that optimally segregates classes within a dataset**.**

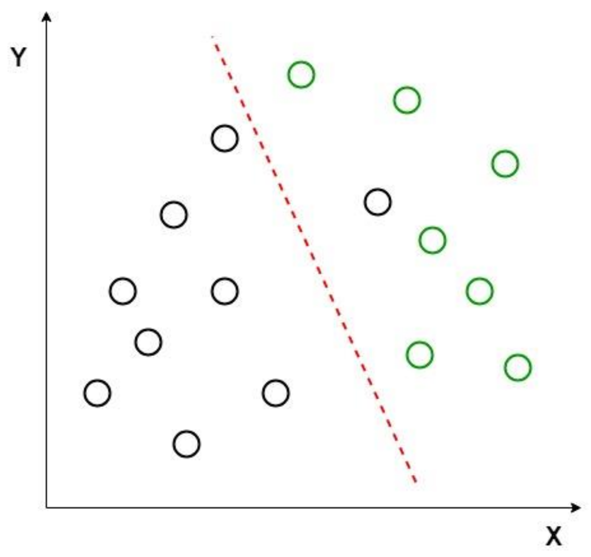
For example, we have two classes and we need to separate them efficiently. Classes can have multiple features. Using only a single feature to classify them may result in some overlapping as shown in the below figure. So, we will keep on increasing the number of features for proper classification.



Assumptions of LDA

LDA assumes that the data has a Gaussian distribution and that the covariance matrices of the different classes are equal. It also assumes that the data is linearly separable, meaning that a linear decision boundary can accurately classify the different classes.

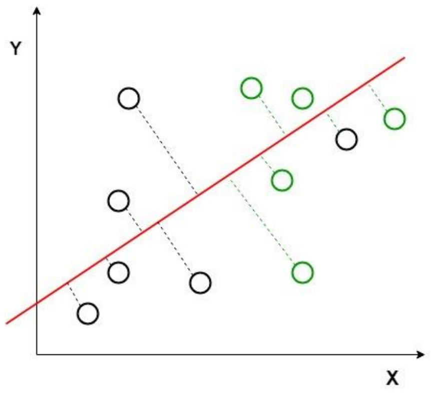
Suppose we have two sets of data points belonging to two different classes that we want to classify. As shown in the given 2D graph, when the data points are plotted on the 2D plane, there’s no straight line that can separate the two classes of data points completely. Hence, in this case, LDA (Linear Discriminant Analysis) is used which reduces the 2D graph into a 1D graph in order to maximize the separability between the two classes.



Here, Linear Discriminant Analysis uses both axes (X and Y) to create a new axis and projects data onto a new axis in a way to maximize the separation of the two categories and hence, reduces the 2D graph into a 1D graph.

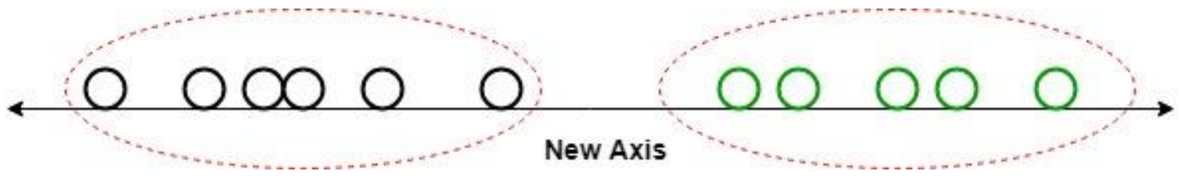
Two criteria are used by LDA to create a new axis:

1. Maximize the distance between the means of the two classes.
2. Minimize the variation within each class.



The perpendicular distance between the line and points

In the above graph, it can be seen that a new axis (in red) is generated and plotted in the 2D graph such that it maximizes the distance between the means of the two classes and minimizes the variation within each class. In simple terms, this newly generated axis increases the separation between the data points of the two classes. After generating this new axis using the above-mentioned criteria, all the data points of the classes are plotted on this new axis and are shown in the figure given below.



But Linear Discriminant Analysis fails when the mean of the distributions are shared, as it becomes impossible for LDA to find a new axis that makes both classes linearly separable. In such cases, we use non-linear discriminant analysis.

How does LDA work?

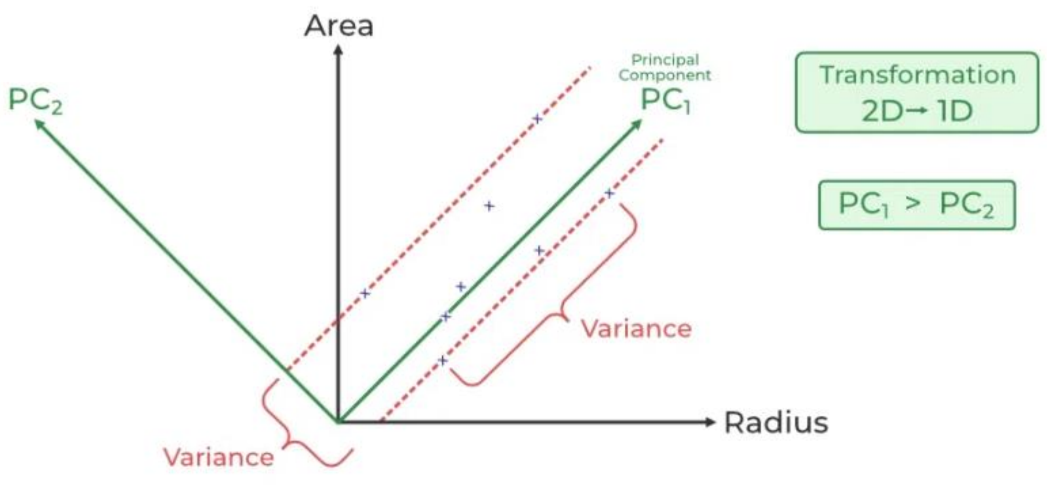
LDA works by projecting the data onto a lower-dimensional space that maximizes the separation between the classes. It does this by finding a set of linear discriminants that maximize the ratio of between-class variance to within-class variance. In other words, it finds the directions in the feature space that best separates the different classes of data.

Principal Component Analysis (PCA)

Principal Component Analysis(PCA) technique was introduced by the mathematician **Karl Pearson** in 1901**.** It works on the condition that while the data in a higher dimensional space is mapped to data in a lower dimension space, the variance of the data in the lower dimensional space should be maximum.

- **Principal Component Analysis (PCA)** is a statistical procedure that uses an orthogonal transformation that converts a set of correlated variables to a set of uncorrelated variables.PCA is the most widely used tool in exploratory data analysis and in machine learning for predictive models. Moreover,
- Principal Component Analysis (PCA) is an unsupervised learning algorithm technique used to examine the interrelations among a set of variables. It is also known as a general factor analysis where regression determines a line of best fit.
- The main goal of Principal Component Analysis (PCA) is to reduce the dimensionality of a dataset while preserving the most important patterns or relationships between the variables without any prior knowledge of the target variables.

Principal Component Analysis (PCA) is used to reduce the dimensionality of a data set by finding a new set of variables, smaller than the original set of variables, retaining most of the sample's information, and useful for the regression and classification of data.



Principal Component Analysis

1. Principal Component Analysis (PCA) is a technique for dimensionality reduction that identifies a set of orthogonal axes, called principal components, that capture the maximum variance in the data. The principal components are linear combinations of the original variables in the dataset and are ordered in decreasing order of importance. The total variance captured by all the principal components is equal to the total variance in the original dataset.
2. The first principal component captures the most variation in the data, but the second principal component captures the maximum variance that is orthogonal to the first principal component, and so on.

3. Principal Component Analysis can be used for a variety of purposes, including data visualization, feature selection, and data compression. In data visualization, PCA can be used to plot high-dimensional data in two or three dimensions, making it easier to interpret. In feature selection, PCA can be used to identify the most important variables in a dataset. In data compression, PCA can be used to reduce the size of a dataset without losing important information.
4. In Principal Component Analysis, it is assumed that the information is carried in the variance of the features, that is, the higher the variation in a feature, the more information that features carries.

Step 1: Standardization

First, we need to standardize our dataset to ensure that each variable has a mean of 0 and a standard deviation of 1.

$$Z = \frac{X-\mu}{\sigma}$$

Here,

- μ is the mean of independent features $\mu = \{\mu_1, \mu_2, \cdots, \mu_m\}$
- σ is the standard deviation of independent features $\sigma = \{\sigma_1, \sigma_2, \cdots, \sigma_m\}$

Step2: Covariance Matrix Computation

Covariance measures the strength of joint variability between two or more variables, indicating how much they change in relation to each other. To find the covariance we can use the formula:

$$cov(x1, x2) = \frac{\sum_{i=1}^{n}(x1_i - \bar{x1})(x2_i - \bar{x2})}{n-1}$$

The value of covariance can be positive, negative, or zeros.

- Positive: As the x1 increases x2 also increases.
- Negative: As the x1 increases x2 also decreases.
- Zeros: No direct relation

Step 3: Compute Eigenvalues and Eigenvectors of Covariance Matrix to Identify Principal Components

Let A be a square nXn matrix and X be a non-zero vector for which

$$AX = \lambda X$$

for some scalar values λ. then λ is known as the eigenvalue of matrix A and X is known as the eigenvector of matrix A for the corresponding eigenvalue.

It can also be written as :

$$AX - \lambda X = 0$$
$$(A - \lambda I)X = 0$$

where I am the identity matrix of the same shape as matrix A. And the above conditions will be true only if $(A – \lambda I)$ will be non-invertible (i.e. singular matrix). That means,

$$|A – \lambda I| = 0$$

From the above equation, we can find the eigenvalues \lambda, and therefore corresponding eigenvector can be found using the equation $AX = \lambda X$.

Independent Component Analysis (ICA)

Independent Component Analysis (ICA) is a statistical and computational technique used in machine learning to separate a multivariate signal into its independent non-Gaussian components. The goal of ICA is to find a linear transformation of the data such that the transformed data is as close to being statistically independent as possible.

The heart of ICA lies in the principle of statistical independence. ICA identify components within mixed signals that are statistically independent of each other.

Statistical Independence Concept:

It is a probability theory that if two random variables X and Y are statistically independent. The joint probability distribution of the pair is equal to the product of their individual probability distributions, which means that knowing the outcome of one variable does not change the probability of the other outcome.

$$P(X and Y) = P(X) * P(Y)$$

Assumptions in ICA

1. The first assumption asserts that the source signals (original signals) are statistically independent of each other.
2. The second assumption is that each source signal exhibits non-Gaussian distributions.

The observed random vector is $X = (x_1, ..., x_m)^T$, representing the observed data with m components. The hidden components are represented by the random vector $S = (s_1, ..., s_n)^T$, where n is the number of hidden sources.

Linear Static Transformation

The observed data X is transformed into hidden components S using a linear static transformation representation by the matrix W.

$S = WX$

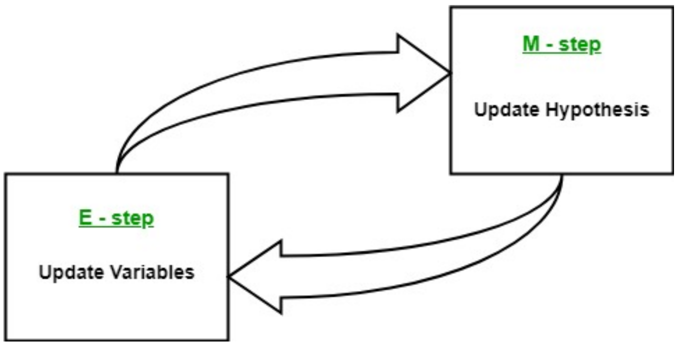
Here, W = transformation matrix.

The goal is to transform the observed data x in a way that the resulting hidden components are independent. The independence is measured by some function $F(s_1, ..., s_n)$. The task is to find the optimal transformation matrix W that maximizes the independence of the hidden components.

The Expectation Maximization Algorithm

The Expectation-Maximization (EM) algorithm is an iterative optimization method that combines different unsupervised machine learning algorithms to find maximum likelihood or maximum posterior estimates of parameters in statistical models that involve unobserved latent variables. The EM algorithm is commonly used for latent variable models and can handle missing data. It consists of an estimation step (E-step) and a maximization step (M-step), forming an iterative process to improve model fit.

- In the E step, the algorithm computes the latent variables i.e. expectation of the log-likelihood using the current parameter estimates.
- In the M step, the algorithm determines the parameters that maximize the expected log-likelihood obtained in the E step, and corresponding model parameters are updated based on the estimated latent variables.

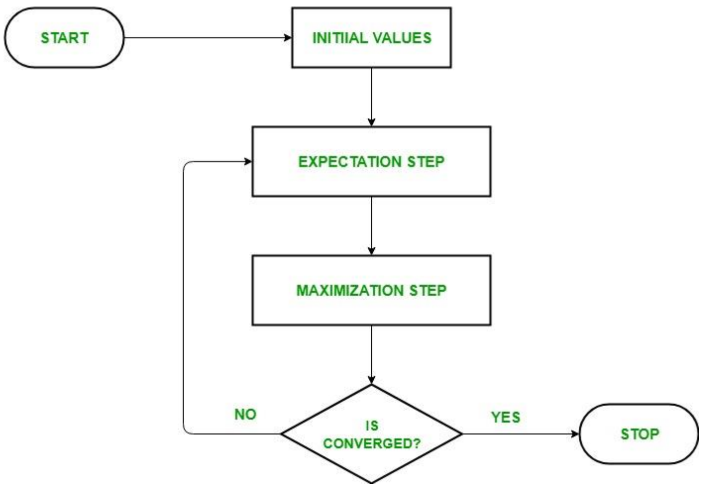


Expectation-Maximization in EM Algorithm

By iteratively repeating these steps, the EM algorithm seeks to maximize the likelihood of the observed data. It is commonly used for unsupervised learning tasks, such as clustering, where latent variables are inferred and has applications in various fields, including machine learning, computer vision, and natural language processing.

How Expectation-Maximization (EM) Algorithm Works:

The essence of the Expectation-Maximization algorithm is to use the available observed data of the dataset to estimate the missing data and then use that data to update the values of the parameters. Let us understand the EM algorithm in detail.



EM Algorithm Flowchart

1. **Initialization:**
    - Initially, a set of initial values of the parameters are considered. A set of incomplete observed data is given to the system with the assumption that the observed data comes from a specific model.
2. **E-Step (Expectation Step):** In this step, we use the observed data in order to estimate or guess the values of the missing or incomplete data. It is basically used to update the variables.
    - Compute the posterior probability or responsibility of each latent variable given the observed data and current parameter estimates.
    - Estimate the missing or incomplete data values using the current parameter estimates.
    - Compute the log-likelihood of the observed data based on the current parameter estimates and estimated missing data.
3. **M-step (Maximization Step):** In this step, we use the complete data generated in the preceding "Expectation" – step in order to update the values of the parameters. It is basically used to update the hypothesis.
    - Update the parameters of the model by maximizing the expected complete data log-likelihood obtained from the E-step.
    - This typically involves solving optimization problems to find the parameter values that maximize the log-likelihood.
    - The specific optimization technique used depends on the nature of the problem and the model being used.
4. **Convergence**: In this step, it is checked whether the values are converging or not, if yes, then stop otherwise repeat *step-2* and *step-3* i.e. "Expectation" – step and "Maximization" – step until the convergence occurs.
    - Check for convergence by comparing the change in log-likelihood or the parameter values between iterations.
    - If the change is below a predefined threshold, stop and consider the algorithm converged.
    - Otherwise, go back to the E-step and repeat the process until convergence is achieved.

Module 5. Kernel Methods and Radial Basis Function Networks

Cover's Theorem

Statement of the Theorem

In its simplest form, Cover's theorem states:

"A complex pattern-classification problem cast in a high-dimensional space nonlinearly is more likely to be linearly separable than in a low-dimensional space, provided that the transformation to the high-dimensional space is nonlinear."

The theorem expresses the number of homogeneously linearly separable sets of N points in D dimensions as an explicit *counting* function $C(N, D)$ of the number of points N and the dimensionality D.

It requires, as a necessary and sufficient condition, that the points are in general position. Simply put, this means that the points should be as linearly independent (non-aligned) as possible. This condition is satisfied "with probability 1" or almost surely for random point sets, while it may easily be violated for real data, since these are often structured along smaller-dimensionality manifolds within the data space.

The function $C(N, D)$ follows two different regimes depending on the relationship between N and D.

- For $N \leq D + 1$, the function is exponential in N. This essentially means that *any* set of labelled points in general position and in number no larger than the dimensionality + 1 is linearly separable; in jargon, it is said that a linear classifier *shatters* any point set with $N \leq D + 1$. This limiting quantity is also known as the Vapnik-Chervonenkis dimension of the linear classifier.
- For $N > D + 1$, the counting function starts growing less than exponentially. This means that, given a sample of fixed size N, for larger dimensionality D it is more probable that a random set of labelled points is linearly separable. Conversely, with fixed dimensionality, for larger sample sizes the number of linearly separable sets of random points will be smaller, or in other words the probability to find a linearly separable sample will decrease with N.

The Interpolation Problem

An interpolation problem refers to the task of finding an estimation function that accurately predicts the value of a dependent variable based on a given set of independent variables. The interpolation function is designed to pass through all the data points in the dataset, providing a smooth and continuous estimation. Unlike regression, interpolation does not require a specific underlying model for the data, making it useful when reliable data points are available.

Given a set of known data points $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, where:

- x_i are the independent variable values, and
- y_i are the corresponding dependent variable values,

the goal is to find a function $f(x)$ such that:

1. $f(x_i) = y_i$ for all $i = 1, 2, \ldots, n$ (exact fit at the given points),
2. $f(x)$ can be used to estimate y for any $x \in [x_1, x_n]$.

1. **Polynomial Interpolation**

- Constructs a polynomial $P(x)$ of degree $n-1$ that passes through all n data points.
- Common techniques:
  - **Lagrange Interpolation:**

$$P(x) = \sum_{i=1}^{n} y_i \prod_{j=1, j\neq i}^{n} \frac{x - x_j}{x_i - x_j}$$

  - **Newton's Divided Difference Interpolation:** Builds the polynomial iteratively using divided differences.

2. **Linear Interpolation**

- Connects adjacent points with straight lines. For two points (x_1, y_1) and (x_2, y_2), the interpolated value at x is:

$$y = y_1 + \frac{(x - x_1)(y_2 - y_1)}{x_2 - x_1}$$

- Simple and efficient but may not capture complex patterns.

3. Spline Interpolation

- Fits a series of piecewise polynomials (usually cubic) to the data points, ensuring smoothness at the boundaries.
- Popular for smooth and natural-looking curves.

4. Radial Basis Function (RBF) Interpolation

- Uses radial basis functions to interpolate, effective for multidimensional data.
- A common choice is the Gaussian RBF.

5. Nearest-Neighbor Interpolation

- Assigns the value of the nearest data point to the unknown point. Simple but may produce blocky results.

6. Barycentric Interpolation

- A numerically stable method for evaluating polynomial interpolation, especially for large datasets.

Radial Basis Function Networks

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.

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.

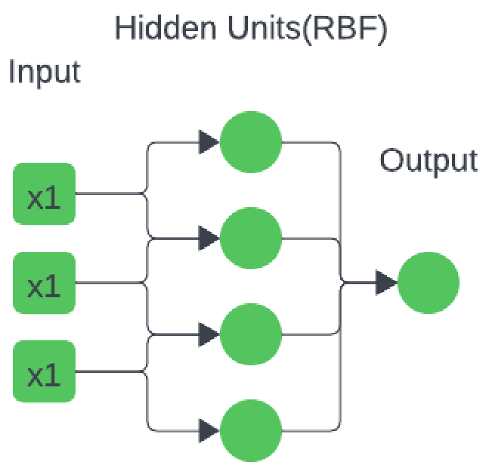
Hidden Layer

- **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.

K Means Clustering

K means clustering, assigns data points to one of the K clusters depending on their distance from the center of the clusters. It starts by randomly assigning the clusters centroid in the space. Then each data point assign to one of the cluster based on its distance from centroid of the cluster. After assigning each point to one of the cluster, new cluster centroids are assigned. This process runs iteratively until it finds good cluster. In the analysis we assume that number of cluster is given in advanced and we have to put points in one of the group.

In some cases, K is not clearly defined, and we have to think about the optimal number of K. K Means clustering performs best data is well separated. When data points overlapped this clustering is not suitable. K Means is faster as compare to other clustering technique. It provides strong coupling between the data points. K Means cluster do not provide clear information regarding the quality of clusters. Different initial assignment of cluster centroid may lead to different clusters. Also, K Means algorithm is sensitive to noise. It may have stuck in local minima.

What is the objective of k-means clustering?

The goal of clustering is to divide the population or set of data points into a number of groups so that the data points within each group are more comparable to one another and different from the data points within the other groups. It is essentially a grouping of things based on how similar and different they are to one another.

How k-means clustering works?

We are given a data set of items, with certain features, and values for these features (like a vector). The task is to categorize those items into groups. To achieve this, we will use the K-means algorithm, an unsupervised learning algorithm. 'K' in the name of the algorithm represents the number of groups/clusters we want to classify our items into.

(It will help if you think of items as points in an n-dimensional space). The algorithm will categorize the items into k groups or clusters of similarity. To calculate that similarity, we will use the Euclidean distance as a measurement.

The algorithm works as follows:

1. First, we randomly initialize k points, called means or cluster centroids.
2. We categorize each item to its closest mean, and we update the mean's coordinates, which are the averages of the items categorized in that cluster so far.
3. We repeat the process for a given number of iterations and at the end, we have our clusters.

The "points" mentioned above are called means because they are the mean values of the items categorized in them. To initialize these means, we have a lot of options. An intuitive method is to initialize the means at random items in the data set. Another method is to initialize the means at random values between the boundaries of the data set (if for a feature *x,* the items have values in [0,3], we will initialize the means with values for *x* at [0,3]).

The above algorithm in pseudocode is as follows:

```
Initialize k means with random values
--> For a given number of iterations:
```

--> Iterate through items:

--> Find the mean closest to the item by calculating
the euclidean distance of the item with each of the means

--> Assign item to mean

--> Update mean by shifting it to the average of the items in that cluster

Recursive Least Squares Estimation of weight vector

Recursive Least Squares (RLS) is an efficient algorithm used to iteratively estimate the weight vector $\mathbf{w}$ of a linear model while minimizing the sum of squared errors. Unlike ordinary least squares (OLS), RLS updates the weight vector dynamically as new data becomes available, making it highly suitable for real-time or online learning.

Formulation

The linear model is:

$$y(t) = \mathbf{w}^\top \mathbf{x}(t) + e(t)$$

where:

- $y(t)$: Output at time t.
- $\mathbf{x}(t)$: Input vector at time t.
- $\mathbf{w}$: Weight vector to be estimated.
- $e(t)$: Error or noise.

The goal is to minimize the **cumulative error**:

$$J(t) = \sum_{i=1}^{t} \lambda^{t-i} \left(y(i) - \mathbf{w}^\top \mathbf{x}(i)\right)^2$$

where:

- λ: Forgetting factor ($0 < \lambda \leq 1$), which gives less weight to older data.

Key Steps in the RLS Algorithm

1. **Initialize**:
   - $\mathbf{w}(0) = \mathbf{0}$ (or any initial guess).
   - $\mathbf{P}(0) = \frac{1}{\delta}\mathbf{I}$, where $\delta > 0$ is a small positive constant, and $\mathbf{I}$ is the identity matrix.

2. **For each new data point $(\mathbf{x}(t), y(t))$:**

   - **Compute the gain vector:**

$$\mathbf{k}(t) = \frac{\mathbf{P}(t-1)\mathbf{x}(t)}{\lambda + \mathbf{x}^\top(t)\mathbf{P}(t-1)\mathbf{x}(t)}$$

   - **Update the weight vector:**

$$\mathbf{w}(t) = \mathbf{w}(t-1) + \mathbf{k}(t)\left[y(t) - \mathbf{x}^\top(t)\mathbf{w}(t-1)\right]$$

   - **Update the inverse covariance matrix:**

$$\mathbf{P}(t) = \frac{1}{\lambda}\left[\mathbf{P}(t-1) - \mathbf{k}(t)\mathbf{x}^\top(t)\mathbf{P}(t-1)\right]$$

Hybrid Learning Procedure for RBF Networks

A **hybrid learning procedure** for **Radial Basis Function (RBF) networks** combines supervised and unsupervised learning to optimize the network parameters, including the centers, spreads (widths), and weights. This method ensures the RBF network performs well in function approximation, pattern recognition, or regression tasks.

Steps in Hybrid Learning

1. **Initialize Parameters:**
   - Select the number of RBF neurons (M).
   - Randomly initialize centers ($\mathbf{c}_j$) or use clustering methods like K-means.
2. **Determine Centers and Spreads:**
   - Perform clustering on input data to find centers.
   - Compute spreads based on the distribution of the centers.
3. **Compute RBF Outputs:**
   - Calculate the hidden layer activations ($\phi_j(\mathbf{x})$) for each input.
4. **Optimize Weights:**
   - Solve for weights using supervised learning techniques.
5. **Evaluate and Update:**
   - Compute the error and adjust parameters iteratively if necessary.

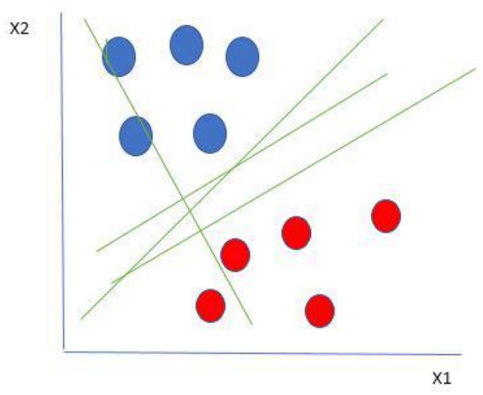
Support Vector Machines

A **Support Vector Machine (SVM)** is a supervised machine learning **algorithm** used for both **classification** and **regression** tasks. While it can be applied to regression problems, SVM is best suited for **classification** tasks. The primary objective of the **SVM algorithm** is to identify the **optimal hyperplane** in an N-dimensional space that can effectively separate data points into different classes in the feature space. The algorithm ensures that the margin between the closest points of different classes, known as **support vectors**, is maximized.

The dimension of the hyperplane depends on the number of features. For instance, if there are two input features, the hyperplane is simply a line, and if there are three input features, the hyperplane becomes a 2-D plane. As the number of features increases beyond three, the complexity of visualizing the hyperplane also increases.

Consider two independent variables, **x1** and **x2**, and one dependent variable represented as either a blue circle or a red circle.

- In this scenario, the hyperplane is a line because we are working with two features (**x1** and **x2**).
- There are multiple lines (or **hyperplanes**) that can separate the data points.
- The challenge is to determine the **best hyperplane** that maximizes the separation margin between the red and blue circles.

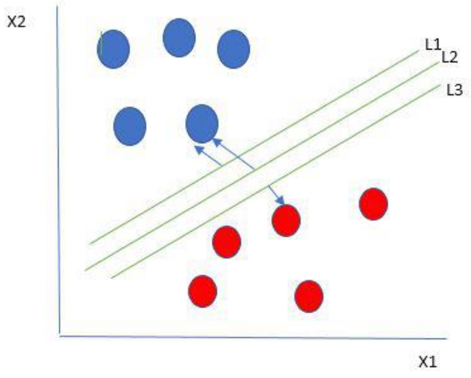


Linearly Separable Data points

From the figure above it’s very clear that there are multiple lines (our hyperplane here is a line because we are considering only two input features x1, x2) that segregate our data points or do a classification between red and blue circles. ***So how do we choose the best line or in general the best hyperplane that segregates our data points?***

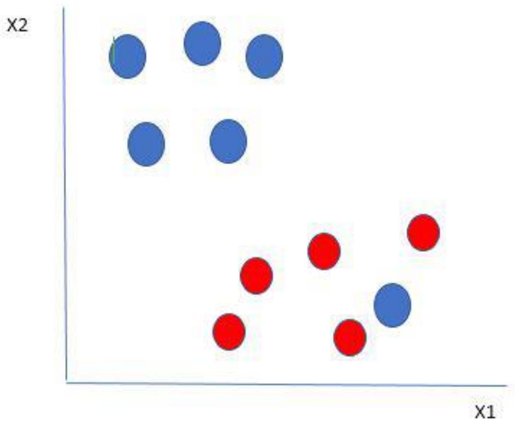
How does Support Vector Machine Algorithm Work?

One reasonable choice for the **best hyperplane** in a **Support Vector Machine (SVM)** is the one that maximizes the **separation margin** between the two classes. The **maximum-margin hyperplane**, also referred to as the **hard margin**, is selected based on maximizing the distance between the hyperplane and the nearest data point on each side.



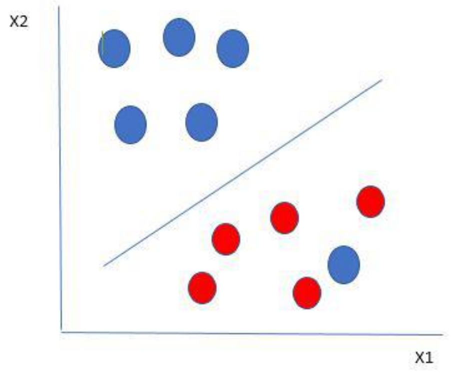
Multiple hyperplanes separate the data from two classes

So we choose the hyperplane whose distance from it to the nearest data point on each side is maximized. If such a hyperplane exists it is known as the **maximum-margin hyperplane/hard margin**. So from the above figure, we choose L2. Let's consider a scenario like shown below



Selecting hyperplane for data with outlier

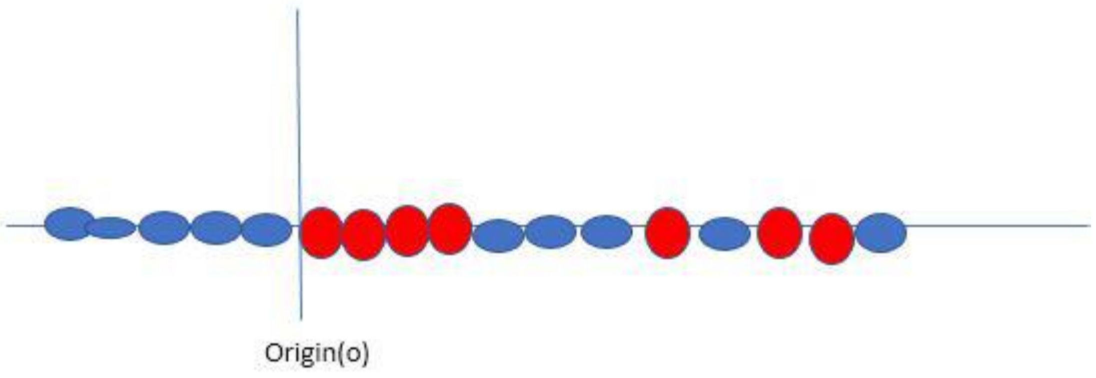
Here we have one blue ball in the boundary of the red ball. So how does SVM classify the data? It's simple! The blue ball in the boundary of red ones is an outlier of blue balls. The SVM algorithm has the characteristics to ignore the outlier and finds the best hyperplane that maximizes the margin. SVM is robust to outliers.



Hyperplane which is the most optimized one

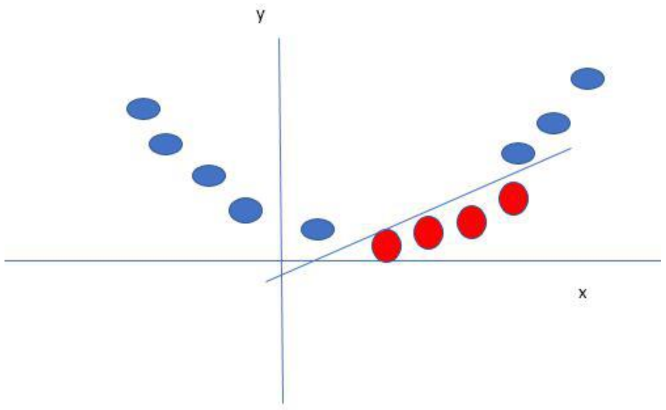
So in this type of data point what SVM does is, finds the maximum margin as done with previous data sets along with that it adds a penalty each time a point crosses the margin. So the margins in these types of cases are called **soft margins**. When there is a soft margin to the data set, the SVM tries to minimize *(1/margin+Λ(∑penalty))*. Hinge loss is a commonly used penalty. If no violations no hinge loss.If violations hinge loss proportional to the distance of 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?



Original 1D dataset for classification

Say, our data is shown in the figure above. SVM solves this by creating a new variable using a **kernel**. We call a point xi on the line and we create a new variable yi as a function of distance from origin o.so if we plot this we get something like as shown below



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. A non-linear function that creates a new variable is referred to as a kernel.