Machine Learning Algorithms Guide

shbahmanyar98@gmail.com

A list of all Machine Learning algorithms:

Regression Algorithms

Linear Regression

Polynomial Regression

Ridge and Lasso Regression

Linear Classification Algorithms

Logistic Regression

Stochastic Gradient Descent Classifier

Naive Bayes Algorithms

How Naive Bayes Works

Gaussian Naive Bayes

Multinomial Naive Bayes

Bernoulli Naive Bayes

SVM and KNN Algorithms

Support Vector Machines

K Nearest Neighbours

Regression Algorithms

Linear Regression Algorithm

Linear Regression Algorithm is a statistical technique for calculating the value of a dependent variable based on the value of an independent variable. The goal of linear regression is to find the best-fit line that describes the relationship between the dependent and the independent variable.
Let's understand how the Linear Regression algorithm works by taking an example of a real-time business problem.
Let's say you are a bakery owner. You want to predict how much money you will make in a day based on the number of cupcakes you sell.
To predict this, you will gather data on the number of cupcakes you sell and how much money you earned in several days. Then, you will use a Linear Regression algorithm to find the best-fit line that describes the relationship between the number of cupcakes sold and the amount of money made.

Once you have the best-fit line, you can use it to predict how much money you will make for a different number of cupcakes sold. For example, if the line tells you that you make $50 by selling 20 cupcakes, then you can predict that selling 40 cupcakes will make $100 in a day.

That's how the Linear Regression algorithm works. It finds the best-fit line that describes the relationship between the input variable and the output variable so we can predict the output variable based on a new input variable.

Advantages & Disadvantages of Linear Regression

Here are some advantages and disadvantages of the Linear Regression algorithm that you should know:

Advantages:

- It's easy to understand and use for making predictions in real-world problems.
- It can analyze the relationship between two variables and make predictions based on their relationship.

Disadvantages:

- It assumes a linear relationship between the input and output variables, which may not always be true.
- It's sensitive to outliers and can be affected by multicollinearity.

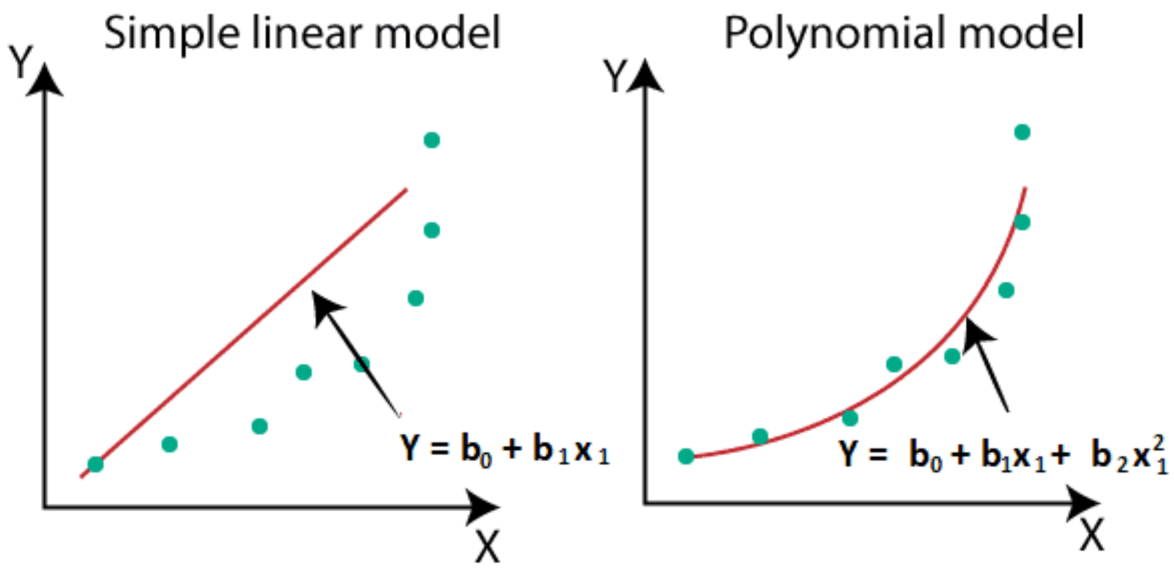
Polynomial Regression Algorithm

polynomial regression is an algorithm that allows us to model nonlinear relationships between input features and output labels. It can be used in real-time business problems, such as sales forecasting, where the relationship between variables is not linear.

Suppose you work as a Data Science professional in a company that sells a certain product. You have historical sales data from past years and want to predict next year's sales. However, the relationship between sales and time (in months) is not linear, and you cannot use a simple linear regression model to accurately predict future sales.

This is where polynomial regression comes in. Instead of using a straight line to fit the data, it fits a polynomial curve of degree 'n' to the data points. The degree 'n' determines the complexity of the curve and can be chosen according to the degree of non-linearity of the data. For example, if the data has a quadratic relationship, we can use a degree of 2, which will fit a parabolic curve to the data points.

below shape show different between Simple linear regression and polynomial regression:



Advantages and Disadvantages of Polynomial Regression Algorithm

Advantages:

- Polynomial regression can model a wide range of nonlinear relationships between input and output variables. It can capture complex patterns that are difficult to model with linear regression.
- Polynomial regression is a simple algorithm that can be easily implemented and understood. It does not require advanced mathematical knowledge or complex algorithms.

Disadvantages:

- Polynomial regression can easily overfit the data if the degree of the polynomial curve is too high. It can lead to poor generalization and inaccurate predictions on new data.

- Polynomial regression can be sensitive to outliers in the data. Outliers can significantly affect the shape of the polynomial curve and lead to inaccurate predictions.

Ridge and Lasso Regression

Ridge and Lasso Regression are regularization techniques used in linear regression to prevent overfitting and improve the model's generalization to unseen data. They work by adding a penalty term to the linear regression loss function. Ridge encourages small coefficients and can prevent multicollinearity, while Lasso can perform feature selection by setting some coefficients to zero. Let's explore both these algorithms in detail one by one.

Ridge Regression

Ridge Regression, also known as L2 regularization, adds a penalty term proportional to the square of the magnitude of the coefficients. It encourages the model to have smaller but non-zero coefficients. It helps in reducing the impact of high-value coefficients and, in turn, the risk of **overfitting**.

Ridge regression handles the problem of multicollinearity, which occurs when predictor variables are highly correlated. Multicollinearity refers to the situation where predictor variables are highly correlated with each other. It can cause issues in traditional linear regression, such as unstable and unreliable coefficient estimates. Ridge Regression helps mitigate this problem by reducing the impact of multicollinearity.

The mathematical formula for ridge regression is ***β = (X^T X + λI)^-1 X^T y***, where β is the vector of coefficients, X is the matrix of independent variables, y is the vector of values of dependent variables, λ is the penalty parameter, and I is the identity matrix.

Ridge Regression works by introducing a penalty term, controlled by the regularization parameter λ, to the ordinary least squares equation. This penalty term restricts the magnitude of the coefficient estimates, thereby reducing their sensitivity to small changes in the input data. By adding this penalty, Ridge Regression shrinks the coefficients towards zero, but they do not become exactly zero. It allows the algorithm to handle multicollinearity and provide more stable and reliable predictions.

Lasso Regression

Lasso Regression, also known as L1 regularization, adds a penalty term proportional to the absolute values of the coefficients. It is a type of regression algorithm used for feature selection and regularization. It is similar to ridge regression, but it adds an L1 penalty term to the regression equation, resulting in a sparse model where some of the coefficients are set to zero.

The mathematical formula for lasso regression is $\boldsymbol{\beta = argmin(\Sigma(yi - \beta0 - \Sigma j=1 \text{ to } p\ xi,j\beta j)\hat{}2 + \lambda\Sigma|\beta j|)}$, where β is the vector of coefficients, $\beta0$ is the intercept term, xi,j is the value of the jth independent variable for the ith observation, yi is the value of the dependent variable for the ith observation, λ is the parameter of penalty, and p is the number of independent variables.

In Lasso regression, the goal is to minimize the sum of squared residuals, just like in ordinary linear regression. However, Lasso introduces an additional term called the L1 penalty or the absolute value of the coefficients.

The L1 penalty encourages the coefficients of less important features to become exactly zero, effectively performing automatic feature selection. This characteristic sets Lasso regression apart from Ridge regression, where the coefficients are only shrunk towards zero but not exactly zero.

To achieve this, Lasso regression adjusts the coefficients during the model fitting process. As the penalty term increases, some coefficients are driven to zero, effectively excluding the corresponding features from the model.

Linear Classification Algorithms

Logistic Regression Algorithm

Logistic Regression is a statistical model used for binary classification problems. It is used to predict the probability of an outcome based on the input features. It uses a sigmoid function to map the input features to output the probability.
Imagine you are an e-commerce company deciding whether to target a customer with a marketing email based on their past purchase behavior. You want to use the customer's previous purchase history as a factor in your decision.
To predict whether to send a marketing email, you can use Logistic Regression. You'll start by looking at historical data to see how past purchase behaviors relate to email engagement. For example, you can look at the data from the past year and notice that customers who made frequent purchases were more likely to engage with marketing emails, while those who made fewer purchases were less likely to engage.

In this case, the logistic regression model would take the customer's past purchase behavior as an input feature and produce a probability of whether the customer will engage with the marketing email.
For example, suppose the model predicts a 70% chance that the customer will engage with the marketing email based on their purchase history. You can then use this prediction to decide whether or not to send the email to the customer.

This is how the logistic regression algorithm works. It helps you predict whether a binary event (such as email engagement) will happen based on input features that are relevant to the event. By using logistic regression, you can make more informed decisions that improve your marketing strategies.

Advantages & Disadvantages of Logistic Regression

Here are some advantages and disadvantages of the Logistic Regression algorithm that you should know:

Advantages:

- It works well for binary classification problems, where the output variable has only two possible values.
- It can handle both continuous and categorical input variables.

Disadvantages:

- Logistic Regression assumes that the relationship between the input variable and the output variable is linear, which may not be true in all cases.
- It can be sensitive to outliers or biased data, which affects predictions.

Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) is a simple yet very efficient approach to fitting linear classifiers and regressors under convex loss functions such as (linear) Support Vector Machines and Logistic Regression. Even though SGD has been around in the machine learning community for a long time, it has received a considerable amount of attention just recently in the context of large-scale learning.

SGD has been successfully applied to large-scale and sparse machine learning problems often encountered in text classification and natural language processing. Given that the data is sparse, the classifiers in this module easily scale to problems with more than 10^5 training examples and more than 10^5 features.

Naive Bayes Algorithms

Naive Bayes

Naive Bayes is an algorithm that uses probabilities to make predictions. It is used for classification problems, where the goal is to predict the class an input belongs to.

Suppose you are a movie streaming service like Netflix and want to recommend movies to your users based on their interests. You have a dataset of movies and their genre tags, as well as information about your users' past movie ratings.

To make recommendations, you can use the Naive Bayes algorithm. Naive Bayes is a statistical algorithm that can predict the probability of an event occurring based on the input characteristics.

For example, suppose a user has watched action and adventure movies before, and you want to recommend a new movie. In this case, the Naive Bayes algorithm will calculate the probability that the user will like a new movie based on its genre.

Advantages and Disadvantages of the Naive Bayes Algorithm

Here are some advantages and disadvantages of the Naive Bayes algorithm that you should know:

Advantages:

- It can handle both continuous and categorical input variables.
- It is less prone to overfitting than other algorithms, which means it can generalize well on new data.

Disadvantages:

- It assumes that the input features are independent, which may not be true in all cases.
- It can be sensitive to the quality of the input data, such as missing values or noisy data.

Gaussian Naive Bayes

Gaussian Naive Bayes is a machine learning classification technique based on a probablistic approach that assumes each class follows a normal distribution. It assumes each parameter has an independent capacity of predicting the output variable. It is able to predict the probability of a dependent variable to be classified in each group. The combination of the prediction for all parameters is the final prediction that returns a probability of the dependent variable to be classified in each group. The final classification is assigned to the group with the higher probability.

Advantages and Disadvantages of the Gussian Naive Bayes Algorithm

Advantages:

- Simplicity, efficiency in training,and works well with high-dimensional data.

Disadvantages:

- Assumes features are independent, which may not hold in reality.

Multinomial Naive Bayes

Multinomial Naive Bayes (MNB) is a very popular and efficient machine learning algorithm that is based on Bayes' theorem. It is commonly used for text classification tasks where we need to deal with discrete data like word counts in documents.

Multinomial Naive Bayes is a probabilistic classifier to calculate the probability distribution of text data, which makes it well-suited for data with features that represent discrete frequencies or counts of events in various natural language processing (NLP) tasks.

Multinomial Distribution

The term "multinomial" refers to the type of data distribution assumed by the model. The features in text classification are typically word counts or term frequencies. The multinomial distribution is used to estimate the likelihood of seeing a specific set of word counts in a document.

Advantages:

Multinomial Naive Bayes offers several advantages, particularly for text classification tasks

- Low computational cost: This algorithm is very efficient to train and use, making it suitable for real-time applications .
- Scalability: It can handle large datasets effectively due to its efficient nature.
- Performance with small datasets: Surprisingly, Naive Bayes can outperform more complex models with smaller datasets .

Disadvantages:

- Naive Independence Assumption: This algorithm assumes that features are independent of each other, which may not always hold true in real-world data.
- Zero Probability Problem: When a new data point has a feature value not seen in the training data, the model might assign a zero probability to that class.

Bernoulli Naive Bayes

Bernoulli Naive Bayes model learns from occurrences between features such as word counts and discrete classes. The input vector must contain positive values, such as counts or TF-IDF values.

SVM and KNN Algorithms

SVM Algorithm

Support Vector Machine (SVM) is a popular supervised learning algorithm used for classification and regression problems in Machine Learning. The basic idea behind SVM is to find a decision boundary that separates data points of different classes with the maximum margin.
The basic idea behind SVM is to find a decision boundary that separates data points of different classes with the maximum margin. It means that SVM aims to find the best line or curve that separates data points of different classes with the

greatest possible distance. The data points closest to the decision boundary are called support vectors. Let's understand how the SVM algorithm works by taking an example of a real-time business problem.

Suppose a company wants to predict whether a customer will default on a loan based on their credit score and income. The business can use SVM to find the best decision boundary that separates defaulting customers from non-defaulting customers based on their credit score and income. The SVM algorithm would analyze the historical data of customers who failed to repay their loans and those who did not, then find the best decision boundary that maximizes the margin between the two classes.

Once the decision boundary is found, it will predict whether a new customer will default on their loan based on their credit score and income. If a new customer falls on the default side of the decision boundary, the business can take appropriate action to mitigate the risk of default, such as declining the loan or raising the interest rate.

KNN Algorithm

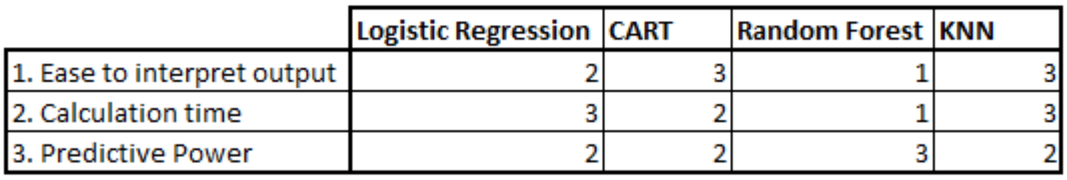
KNN Algorithm can be used for both classification and regression predictive problems. However, it is more widely used in classification problems in the industry. To evaluate any technique, we generally look at 3 important aspects:

1. Ease of interpreting output

2. Calculation time

3. Predictive Power

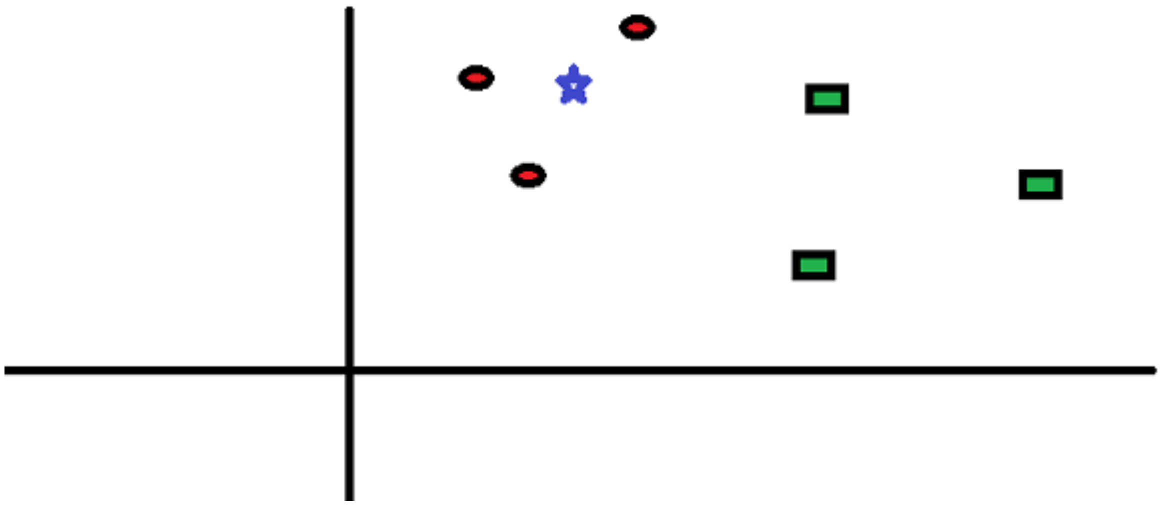
Let us take a few examples to place KNN in the scale :



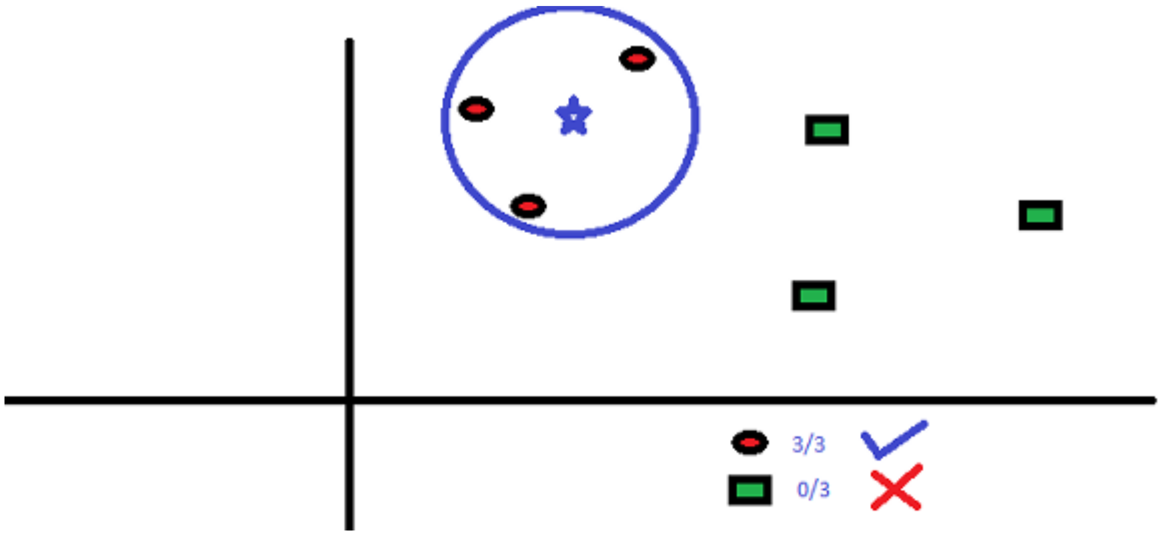
	Logistic Regression	CART	Random Forest	KNN
1. Ease to interpret output	2	3	1	3
2. Calculation time	3	2	1	3
3. Predictive Power	2	2	3	2

KNN classifier fairs across all parameters of consideration. It is commonly used for its ease of interpretation and low calculation time.

Let’s take a simple case to understand this algorithm. Following is a spread of red circles (RC) and green squares (GS):



You intend to find out the class of the blue star (BS). BS can either be RC or GS and nothing else. The “K” in KNN algorithm is the nearest neighbor we wish to take the vote from. Let’s say K = 3. Hence, we will now make a circle with BS as the center just as big as to enclose only three data points on the plane. Refer to the following diagram for more details:



The three closest points to BS are all RC. Hence, with a good confidence level, we can say that the BS should belong to the class RC. Here, the choice became obvious as all three votes from the closest neighbor went to RC. The choice of the parameter K is very crucial in this algorithm. Next, we will understand the factors to be considered to conclude the best K.

Decision Trees and Ensemble Methods

Decision Tree

Decision Tree is an algorithm that works like a flowchart or tree structure to make decisions based on input data. It starts with a question or condition and then follows different branches based on the answers to subsequent questions until a prediction is made.

Decision Tree is an algorithm used to solve problems that require making decisions based on multiple criteria or characteristics. Let’s understand how the decision tree algorithm works by taking an example of a real-time business problem.

Suppose you are a bank that wants to determine if a person is likely to be approved for a loan based on their credit score. You collected data on past loan applicants, including their credit scores and whether their loan applications were approved.
A decision tree algorithm is like a flowchart that can help the bank make decisions based on a person's credit score. In this problem, the decision tree algorithm will start with a "root" node representing the first question, such as whether a person's credit score is above or below a certain threshold.

Depending on the answer, the algorithm follows branches to subsequent questions, such as income level or employment status, until a "leaf" node is reached.

The leaf node represents a decision, such as predicting whether to approve or deny a loan based on input characteristics.

So, a decision tree algorithm is like a flowchart that makes decisions based on a series of questions and answers. It starts at a "root" node with a question, branches to subsequent questions based on the answers, and finally reaches a "leaf" node representing a decision.

Advantages and Disadvantages of Decision Tree Algorithm

Here are some advantages and disadvantages of the Decision Tree algorithm that you should know:

Advantages:

- Decision trees produce easy-to-interpret decision rules, allowing stakeholders to understand and trust the decision-making process.
- Decision trees can effectively handle missing data by making decisions based on the data available for each division.

Disadvantages:

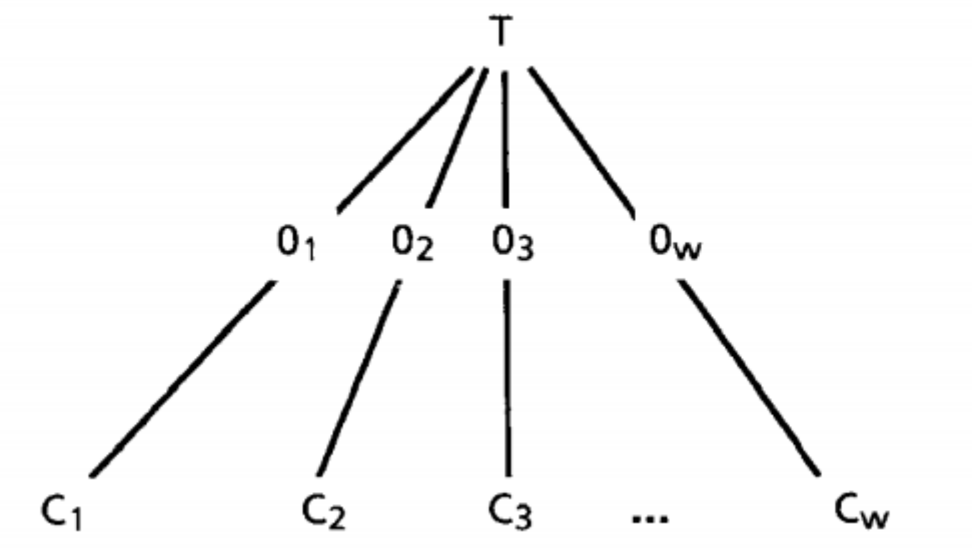
- Decision trees are prone to overfitting, especially when the tree becomes too deep or complex.
- Decision trees are sensitive to small changes in input data, which can result in different tree structures and decision rules.

ID3, C4.5, CART

Decision Trees are Machine Learning algorithms that is used for both classification and Regression. Decision Trees can be used for multi-class classification tasks also. Decision Trees use a Tree like structure for making predictions where each internal nodes represents the *test*(if attribute A takes vale <5) on an attribute and each branch represent the outcome of the test on the attribute. The leaf nodes represent the class label. Some of the popular algorithms that are used to generate a Decision tree from a Dataset are ID3, c4.5 and CART.

ID3 Algorithm

ID3 stands for Iterative Dichotomiser 3 which is a learning algorithm for Decision Tree introduced by Quinlan Ross in 1986. ID3 is an iterative algorithm where a subset(window) of the training set is chosen at random to build a decision tree. This tree will classify every objects within this window correctly. For all the other objects that are not there in the window, the tree tries to classify them and if the tree gives correct answer for all these objects then the algorithm terminates. If not, then the incorrectly classified objects are added to the window and the process continues. This process continues till a correct Decision Tree is found. This method is fast and it finds the correct Decision Tree in a few iterations. Consider an arbitrary collection of C objects. If C is empty or contains only objects of a single class, then the Decision Tree will be a simple tree with just a leaf node labelled with that class. Else consider T to be test on an object with outcomes{O_1, O_2, O_3....Ow}. Each Object in *C* will give one of these Outcomes for the test *T*. Test *T* will partition *C* into {C_1, C_2, C_3....Cw}. where C_i contains objects having outcomes O_i. We can visualize this with the following diagram :



When we replace each individual C_i in the above figure with a Decision Tree for C_i, we would get a Decision tree for all the *C*. This is a divide-and-conquer strategy which will yield single-object subsets that will satisfy the one-class requirement for a leaf. So as long as we have a test which gives a non-trivial partition of any set of objects, this procedure will always produce a Decision Tree that can correctly Classify each object in *C*. For simplicity, let us consider the test to be branching on the values of an attribute, Now in ID3, For choosing the root of a tree, ID3 uses an Information based approach that depends on two assumptions. Let *C* contain p objects of class *P* and n of class *N*. These assumptions are :

1. A correct decision tree for *C* will also classify the objects in such a way that the objects will have same proportion as in *C*. The Probability that an arbitrary object will belong to class *P* is given below as :

$$\frac{p}{p+n}$$

And the probability that it will belong to class *N* is given as :

$$\frac{n}{p+n}$$

2. A decision tree returns a class to which an object belongs to. So a decision tree can be considered as a source of a message *P* or *N* and the expected information needed to generate this message is given as :

$$I(p,n) = -\frac{p}{p+n}log_2\left(\frac{p}{p+n}\right) - \frac{n}{p+n}log_2\left(\frac{n}{p+n}\right)$$

Let us consider an attribute AA as the root with values $\{A_1,A_2.....,Av\}$. Now *A* will partition *C* into $\{C_1,C_2.....,Cv\}$, where C_i has those objects in CC that have a value of A_i of *A*. Now consider C_i having p_i objects of class *P* and *ni* objects of class *N*. The expected information required for the subtree for C_i is $I(p_i,n_i)$. The expected information required for the tree with *A* as root is obtained by :

$$E(A) = \sum_{i=1}^{v} \frac{p_i + n_i}{p+n} I(p_i, n_i)$$

Now this is a weighted average where the weight for the *i_th branch is the proportion of Objects in _C* that belong to C_i. Now the information that is gained by selecting *A* as root is given by :

$$gain(A) = I(p,n) - E(A)$$

Here *I* is called the Entropy. So here ID3 choose that attribute to branch for which there is maximum Information Gain. So ID3 examines all the attributes and selects that *A* which maximizes the *gain(A)* and then uses the same process recursively to form Decision Trees for the subsets $\{C_1,C_2.....,Cv\}$ till all the instances within a branch belong to same class.

Drawback Of Information Gain

Information gain is biased towards test with many occurances. Consider a feature that uniquely identifies each instance of a Training set and if we split on this feature, it would result in many brances with each branch containing instances of a single class alone(in other words pure) since we get maximum information gain and hence results in the Tree to overfit the Training set.

Gain Ratio

This is a modification to Information Gain to deal with the above mentioned problem. It reduces the bias towards multi-valued attributes. Consider a training dataset which contains *p* and *n* objects of class *P* and *N* respectively and the attribute *A* has values $\{A_1,A_2.....,Av\}$. Let the number of objects with value A_i of attribute *A* be p_i and n_i respectively. Now we can define the Intrinsic Value(IV) of *A* as :

$$IV(A) = -\sum_{i=1}^{v} \frac{p_i + n_i}{p + n} log_2 \left(\frac{p_i + n_i}{p + n} \right)$$

IV(A) measures the information content of the value of Attribute *A*. Now the Gain Ratio or the Information Gain Ratio is defined as the ratio between the Information Gain and the Intrinsic Value.

$$GainRatio(A) = \frac{gain(A)}{IV(A)}$$

Now here we try to pick an Attribute for which the Gain Ratio is as large as possible. This ratio may not be defined when $IV(A) = 0$. Also gain ratio may tend to favour those attributes for which the Intrinsic Value is very small. When all the attributes are Binary, the gain ratio criteria has been found to produce smaller trees.

C4.5 Algorithm

This is another algorithm that is used to create a Decision Tree. This is an extension to ID3 algorithm. Given a training dataset $S = S_1, S_2, ...$C4.5 grows the initial tree using the divide-and-conquer approach as :

- If all the instances in S belongs to the same class, or if S is small, then the tree is leaf and is given the label of the same class.
- Else, choose a test based on a single attribute which has two or more outcomes. Then make the test as the root of the tree with a branch for each outcome of the test.
- Now partition S into corresponding subsets $S_1, S_2, ...,$ based on the outcome of each case.
- Now apply the procedure recursively to each of the subset $S_1, S_2, ...$

Here the splitting criteria is Gain Ratio. Here the attributes can either be numeric or nominal and this determines the format of the test of the outcomes. If an attribute is numeric, then for an Attribute A, the test will be $\{A \leq h, A > h\}$. Here h is the threshold found by sorting Son the values of A and then choosing the split between successive values that maximizes the Gain Ratio. Here the initial tree is Pruned to avoid Overfitting by removing those branches that do not help and replacing them with leaf nodes. Unlike ID3, C4.5 handles missing values. Missing values are marked separately and are not used for calculating Information gain and Entropy.

Classification And Regression Trees(CART)

This is a decision Tree Technique that produces either a Classification Tree when the dependent variable is categorical or a Regression Tree when the dependent variable is numeric.

Classification Trees :

Consider a Dataset (*D*) with features $X = x_1, x_2 \ldots, x_n$ and let $y = y_1, y_2 \ldots y_m$ be set of all the possible classes that *X* can take. Tree based classifiers are formed by making repetitive splits on *X* and subsequently created subsets of *X*. For eg. *X* could be divided such that $\{x|x_3 \leq 53.5\}$ and $\{x|x_3 > 53.5\}$. Then the first set could be divided further into $X_1 = \{x|x_3 \leq 53.5, x_1 \leq 29.5\}$ *and* $X_2 = \{x|x_3 \leq 53.5, x_1 > 29.5\}$ *and the other set could be split into* $X_3 = \{x|x_3 > 53.5, x_1 \leq 74.5\}$ *and* $X_4 = \{x|x_3 > 53.5, x_1 > 74.5\}$. *This can be applied to problems with multiple classes also. When we divide XX into subsets, these subsets need not be divided using the same variable. ie one subset could be split based on* x_1 and other on x_2. Now we need to determine how to best split *X* into subsets and how to split the subsets also. CART uses binary partition recursively to create a binary tree. There are three issues which CART addresses :

- Identifying the Variables to create the split and determining the rule for creating the split.
- Determine if the node of a tree is terminal node or not.
- Assigning a predicted class to each terminal node.

Random Forest

the Random Forest algorithm is an ensemble learning method for classification and regression tasks. It is composed of multiple decision trees, where each tree is built on a random subset of the input features and a random subset of the training data. So, if you are new to machine learning and want to know how the Random Forest algorithm works, this article is for you. In this article, I will introduce how the Random Forest algorithm works and how to implement it using Python.

andom Forests are known for their ability to handle **high-dimensional data**, handle missing values, and perform feature selection. Let's understand how the Random Forest algorithm works by taking an example of a real-time business problem.

Suppose you own a store and want to predict which products will be popular with your customers. You have a lot of data about your customers, such as their age, gender, and previous purchases. You also have product data such as price, category and features.

Now, to predict a product, you can seek the opinion of a panel of experts. Each expert may have a different point of view depending on their knowledge and experience. For example, one expert may be good at predicting which products are popular with young people, while another expert may be good at predicting which products are popular with women.

The Random Forest algorithm works similarly. It combines the opinions of many "experts", called decision trees. Each decision tree looks at a subset of data and predicts based on predefined rules. For example, a decision tree might indicate that if a customer is female and has purchased a product in the same category before, she is likely to purchase a new product in that category.

When you have many decision trees, you can predict by taking the average of their predictions. It helps reduce the impact of an individual tree making a mistake. For example, if one tree predicts that a product will be very popular, but most of the other trees predict that it will be average, then the Random Forest algorithm will predict that it will be average.

Thus, the Random Forest algorithm is an ensemble learning method for classification and regression tasks. It is composed of multiple decision trees, where each tree is built on a random subset of the input features and a random subset of the training data. During the learning phase, each decision tree is built by recursively partitioning the data into subsets based on the values of the input features. Split points are chosen to maximize information gain or decrease impurity in each node. The final result of the algorithm is obtained by aggregating the predictions of all the trees, typically taking a majority vote for classification problems or the average for regression problems.

Advantages and Disadvantages of Random Forest Algorithm

Advantages:

- Random Forest can handle a large number of input features, including both categorical and numerical features.
- It is less prone to overfitting compared to other machine learning algorithms, such as decision trees.

Disadvantages:

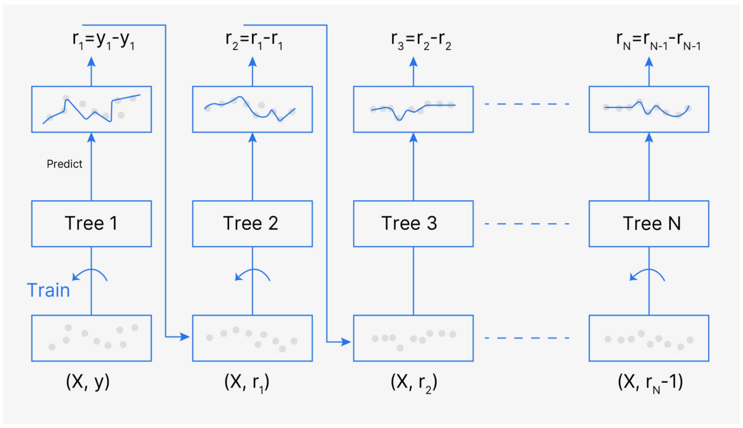
- Random Forest is a black box model, which means that it is difficult to understand how it makes its predictions.
- It may take longer to train compared to other machine learning algorithms, especially on large datasets.

Boosting Algorithms

Gradient boosting

Prediction models are one of the most commonly used machine learning models. Gradient boosting is a method standing out for its prediction speed and accuracy, particularly with large and complex datasets. From Kaggle competitions to machine learning solutions for business, this algorithm has produced the best results.It is more popularly known as Gradient Boosting Machine or GBM.

Gradient boosting is a machine learning ensemble technique that combines the predictions of multiple weak learners, typically decision trees, sequentially. It aims to improve overall predictive performance by optimizing the model's weights based on the errors of previous iterations, gradually reducing prediction errors and enhancing the model's accuracy. This is most commonly used for linear regression.



Adaboost

Before getting into the details of the gradient boosting algorithm, we must have some knowledge about the AdaBoost Algorithm which is again a boosting method. This algorithm starts by building a decision stump and then assigning equal weights to all the data points. Then it increases the weights for all the points that are misclassified and lowers the weight for those that are easy to classify or are correctly classified. A new decision stump is made for these weighted data points. The idea behind this is to improve the predictions made by the first stump. The main difference between these two algorithms is that gradient boosting has a fixed base estimator i.e., decision trees whereas in AdaBoost we can change the base estimator according to our needs.

Clustering Algorithms

K-Means Clustering

The K-Means Clustering is a clustering algorithm capable of clustering an unlabeled dataset quickly and efficiently in just a very few iterations.

Clustering means identifying similar instances and assigning them to clusters or groups of similar instances. It is used in a wide variety of applications such as:

- Customer Segmentation
- Data Analysis
- Dimensionality Reduction
- Anomaly Detection
- Semi-supervised learning
- Searching Images
- Image Segmentation

K-Means is a clustering algorithm in machine learning that can group an unlabeled dataset very quickly and efficiently in just a few iterations. It works by labelling all instances on the cluster with the closest centroid. When the instances are centred around a particular point, that point is called a centroid.

If you receive the instance labels, you can easily locate all items by averaging all instances for each cluster. But here we are not given a label or centroids, so we have to start by placing the centroids randomly by selecting k random instances and using their locations as the centroids.

Then we label the instances, update the centroids, re-label the instances, update the centroids again and so on. The K-Means clustering algorithm is guaranteed to converge in a few iterations, it will not continue to iterate forever.

DBSCAN Clustering

DBSCAN stands for Density-Based Spatial Clustering for Applications with Noise. This is an unsupervised clustering algorithm which is used to find high-density base samples to extend the clusters.

The DBSCAN Clustering algorithm is based on the concept of core samples, non-core samples, and outliers:

- Core Samples: The samples present in the high-density area have minimum sample points with the eps radius.
- Non-core samples: The samples close to core samples but are not core samples but are very near to the core samples. The no-core samples lie within the eps radius of the core samples but they don't have minimum samples points.
- Outliers: The samples that are not part of the core samples and the non-core samples and are far away from all the samples.

The DBSCAN clustering algorithm works well if all the clusters are dense enough and are well represented by the low-density regions.

Agglomerative Clustering

Agglomerative clustering is based on hierarchical clustering which is used to form a hierarchy of clusters. It is one of the types of clustering algorithms in machine learning. Unlike the **K-Means** and **DBSCAN** clustering algorithms, it is not very common but it is very efficient to form a hierarchy of clusters.

Agglomerative clustering is one of the clustering algorithms where the process of grouping similar instances starts by creating multiple groups where each group contains one entity at the initial stage, then it finds the two most similar groups, merges them, repeats the process until it obtains a single group of the most similar

instances. For example, think of bubbles floating on the water and getting attached, at the end, you will see a large group of bubbles.
Some of the advantages of using this algorithm for clustering are:

- It adapts very well to a large number of instances
- It can capture the clusters of different shapes
- It forms flexible and informative clusters
- It can also be used with any pairwise distance

BIRCH Clustering

The BIRCH is a Clustering algorithm in machine learning. It stands for Balanced Reducing and Clustering using Hierarchies.
BIRCH is a clustering algorithm in machine learning that has been specially designed for clustering on a very large data set. It is often faster than other clustering algorithms like batch K-Means. It provides a very similar result to the batch K-Means algorithm if the number of features in the dataset is not more than 20.
When training the model using the BIRCH algorithm, it creates a tree structure with enough data to quickly assign each data point to a cluster. By storing all the data points in the tree, this algorithm allows the use of limited memory while working on a very large data set.

Mean Shift Clustering

The mean shift algorithm is a nonparametric clustering algorithm that does not require prior knowledge of the number of clusters.

Mean Shift clustering is a nonparametric clustering algorithm that does not require any prior knowledge of the number of clusters. Below is the complete process of the Mean Shift algorithm:

- It starts by placing a circle centered on each sample
- Then for each circle, it calculates the mean of all the samples located in the circle
- Then it moves the circle so that it is centered on the mean
- Then it iterates the mean shift step until all of the circles stop moving
- Then it shifts the circles in the direction of the highest density until each circle reaches a maximum of local density
- Then all the instances whose circles have settled in the same place are assigned to the same cluster

Some of the features of this algorithm are like the DBSCAN clustering algorithm, like how it finds any number of clusters of any shape. But unlike the DBSCAN clustering algorithm, this algorithm tends to cut clusters into chunks when they have internal density variations.
It is not a popular clustering algorithm because it is not suitable while working with large datasets. I hope you now have understood what mean-shift clustering is in machine learning.

Neural Network Architectures

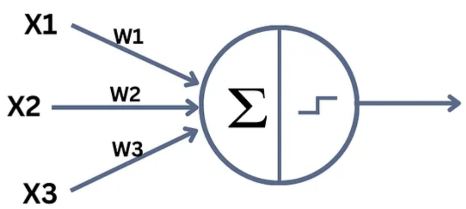
Perceptron

Perceptron is one of the simplest architecture of Artificial Neural Networks in Machine Learning. It was invented by Frank Rosenblatt in 1957.

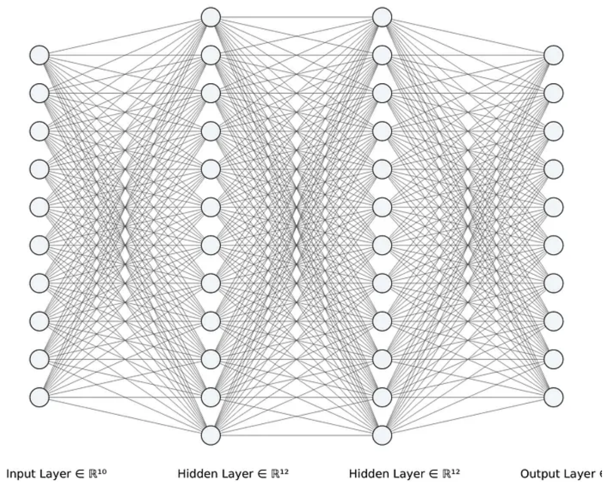
Perceptron is a type of neural network architecture that falls under the category of the simplest form of artificial neural networks. The Perceptrons are generally based on different types of artificial neurons known as Threshold Logic Unit (TLU) or sometimes Linear Threshold Unit (LTU). The inputs and outputs of a perceptron are numbers, unlike the values we see using a classification algorithm like logistic regression (True or False values).
Perceptrons are made up of a single layer of Threshold Logic Unit where each TLU is connected to all inputs. A single TLU can be used to solve the binary classification problem and if all neurons in one layer are connected to each neuron in the previous layer, it is called a fully connected layer or dense layer. Such types of architectures can be used in the problems of multiclass classification.

Multilayer Perceptron

A Multilayer Perceptron or MLP is one of the simplest feed-forward neural networks. Multilayer Perceptrons are the types of neural networks which are bidirectional as they foreword propagation of the inputs and backward propagation of the weights.
Some machine learning practitioners often confuse Perceptron and a Multilayer Perceptron with each other. Perceptron is the most basic architecture of the neural network, it is also known as a single-layered neural network. Perceptron is specially designed for the problems of binary classification, but MLPs has nothing to do with perceptron.
A Multilayer Perceptron has an input layer and an output layer with one or more hidden layers. In MLPs, all neurons in one layer are connected to all neurons in the next layer. Here, the input layer receives the input signals and the desired task is performed by the output layer. And the hidden layers are responsible for all the calculations.



Single-layer perceptron



Multi-layer perceptron

Convolutional Neural Network

A Convolutional Neural Network (CNN), also known as ConvNet, is a specialized type of deep learning algorithm mainly designed for tasks that necessitate object recognition, including image classification, detection, and segmentation. CNNs are employed in a variety of practical scenarios, such as autonomous vehicles, security camera systems, and others.

The importance of CNNs

There are several reasons why CNNs are important in the modern world, as highlighted below:

- CNNs are distinguished from classic machine learning algorithms such as SVMs and decision trees by their ability to autonomously extract features at a large scale, bypassing the need for manual feature engineering and thereby enhancing efficiency.
- The convolutional layers grant CNNs their translation-invariant characteristics, empowering them to identify and extract patterns and

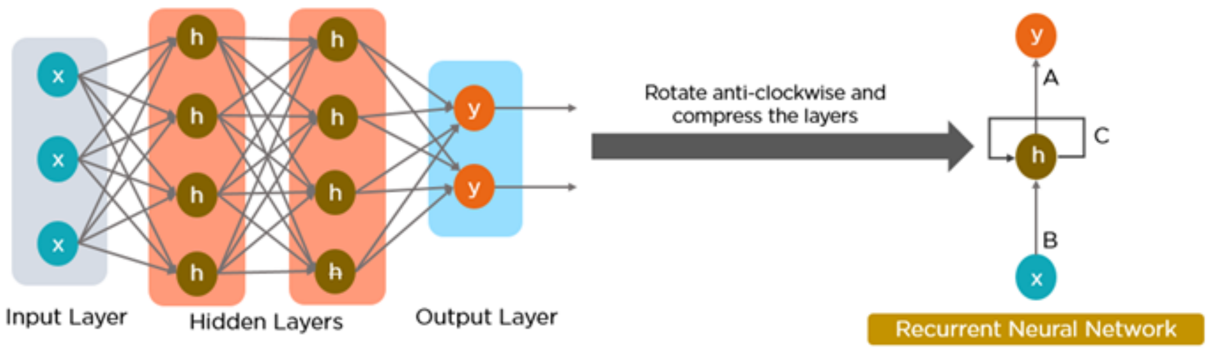
features from data irrespective of variations in position, orientation, scale, or translation.

- A variety of pre-trained CNN architectures, including VGG-16, ResNet50, Inceptionv3, and EfficientNet, have demonstrated top-tier performance. These models can be adapted to new tasks with relatively little data through a process known as fine-tuning.
- Beyond image classification tasks, CNNs are versatile and can be applied to a range of other domains, such as natural language processing, time series analysis, and speech recognition.

recurrent neural network

A recurrent neural network (RNN) is a deep learning model that is trained to process and convert a sequential data input into a specific sequential data output. Sequential data is data such as words, sentences, or time-series data where sequential components interrelate based on complex semantics and syntax rules. An RNN is a software system that consists of many interconnected components mimicking how humans perform sequential data conversions, such as translating text from one language to another. RNNs are largely being replaced by transformer-based artificial intelligence (AI) and large language models (LLM), which are much more efficient in sequential data processing.

RNNs are a type of neural network that can be used to model sequence data. RNNs, which are formed from feedforward networks, are similar to human brains in their behaviour. Simply said, recurrent neural networks can anticipate sequential data in a way that other algorithms can't.

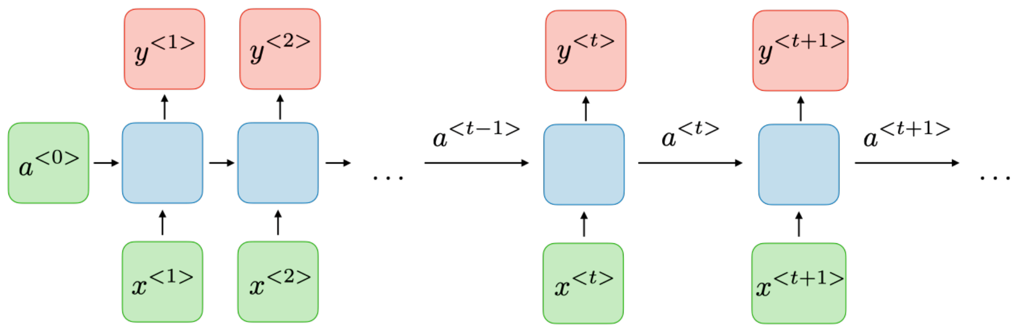


All of the inputs and outputs in standard neural networks are independent of one another, however in some circumstances, such as when predicting the next word of a phrase, the prior words are necessary, and so the previous words must be remembered. As a result, RNN was created, which used a Hidden Layer to overcome the problem. The most important component of RNN is the Hidden state, which remembers specific information about a sequence.

RNNs have a Memory that stores all information about the calculations. It employs the same settings for each input since it produces the same outcome by performing the same task on all inputs or hidden layers.

The Architecture of a Traditional RNN

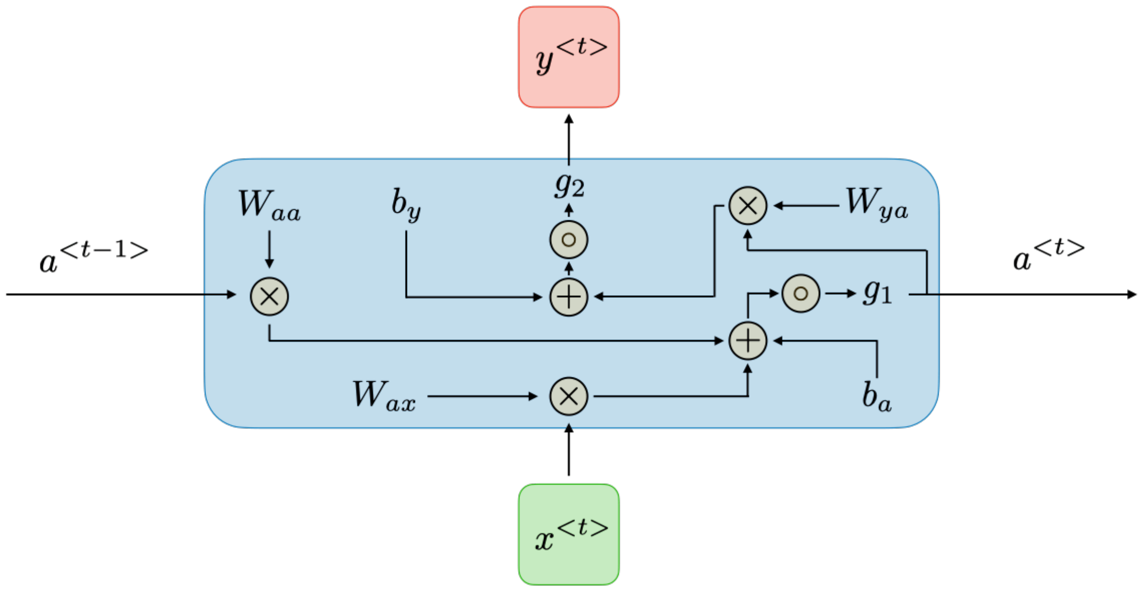
RNNs are a type of neural network that has hidden states and allows past outputs to be used as inputs. They usually go like this:



For each timestep t, the activation $a^{<t>}$ and the output $y^{<t>}$ are expressed as follows:

$$\boxed{a^{<t>} = g_1(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a)} \quad \text{and} \quad \boxed{y^{<t>} = g_2(W_{ya}a^{<t>} + b_y)}$$

where $W_{ax}, W_{aa}, W_{ya}, b_a, b_y$ are coefficients that are shared temporally and g_1, g_2 activation functions.



RNN architecture can vary depending on the problem you're trying to solve. From those with a single input and output to those with many (with variations between).

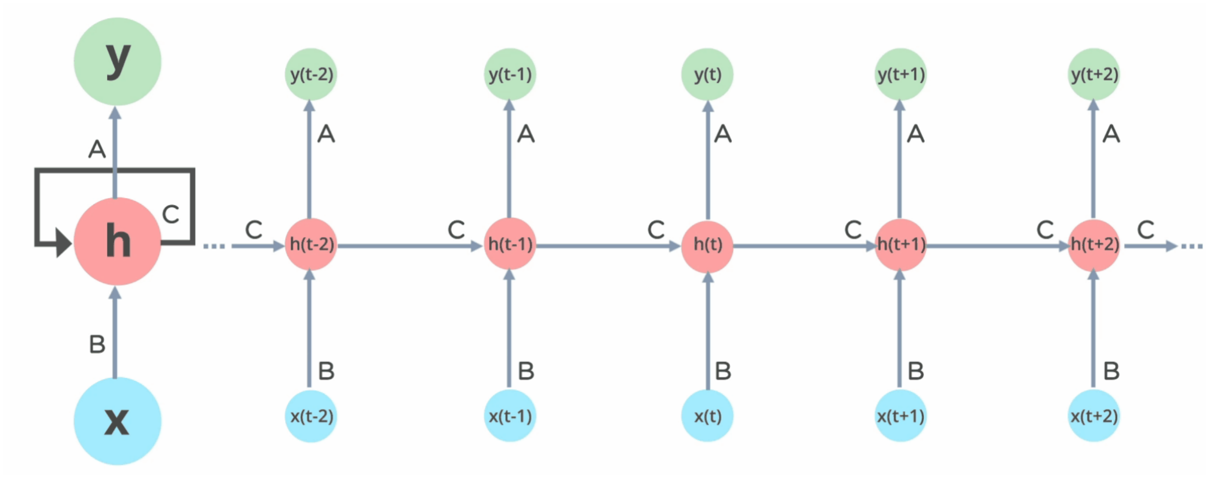
Below are some examples of RNN architectures that can help you better understand this.

- One To One: There is only one pair here. A one-to-one architecture is used in traditional neural networks.
- One To Many: A single input in a one-to-many network might result in numerous outputs. One too many networks are used in the production of music, for example.
- Many To One: In this scenario, a single output is produced by combining many inputs from distinct time steps. Sentiment analysis and emotion

identification use such networks, in which the class label is determined by a sequence of words.

- Many To Many: For many to many, there are numerous options. Two inputs yield three outputs. Machine translation systems, such as English to French or vice versa translation systems, use many to many networks.

The information in recurrent neural networks cycles through a loop to the middle hidden layer.



The input layer x receives and processes the neural network's input before passing it on to the middle layer.

Multiple hidden layers can be found in the middle layer h, each with its own activation functions, weights, and biases. You can utilize a recurrent neural network if the various parameters of different hidden layers are not impacted by the preceding layer, i.e. There is no memory in the neural network.

The different activation functions, weights, and biases will be standardized by the Recurrent Neural Network, ensuring that each hidden layer has the same characteristics. Rather than constructing numerous hidden layers, it will create only one and loop over it as many times as necessary.

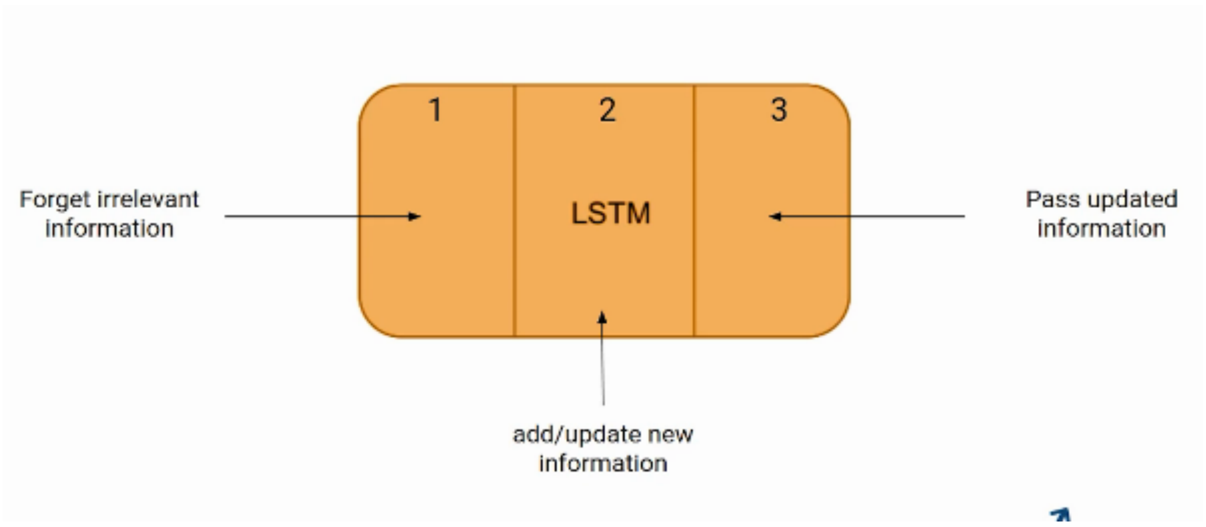
Long Short-Term Memory

LSTM (Long Short-Term Memory) is a recurrent neural network (RNN) architecture widely used in Deep Learning. It excels at capturing long-term dependencies, making it ideal for sequence prediction tasks.

Unlike traditional neural networks, LSTM incorporates feedback connections, allowing it to process entire sequences of data, not just individual data points. This makes it highly effective in understanding and predicting patterns in sequential data like time series, text, and speech.

LSTM has become a powerful tool in artificial intelligence and deep learning, enabling breakthroughs in various fields by uncovering valuable insights from sequential data.

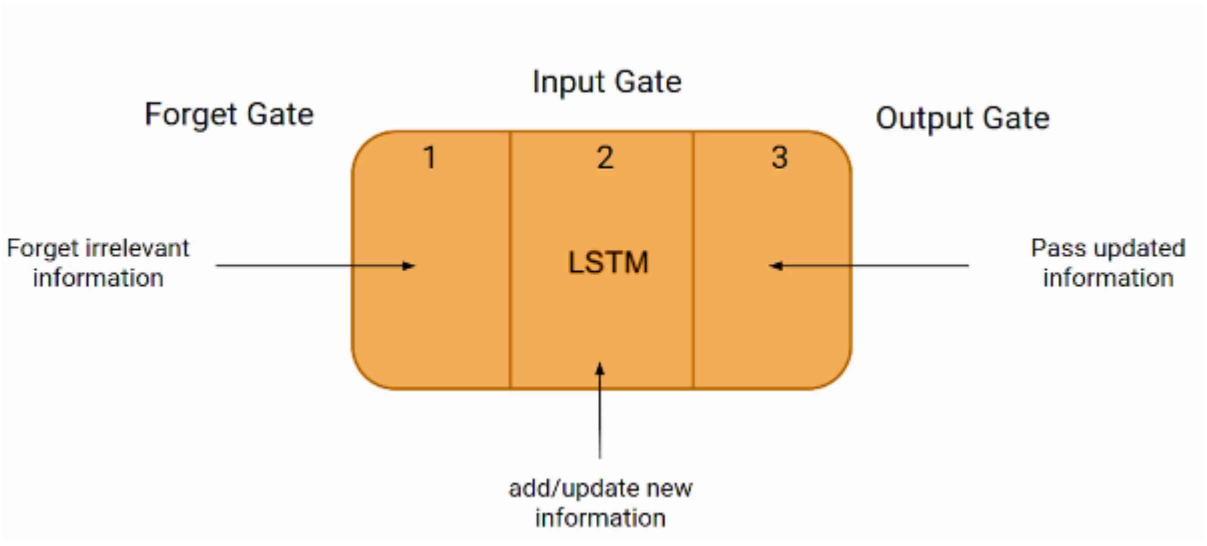
At a high level, LSTM works very much like an RNN cell. Here is the internal functioning of the LSTM network. The LSTM network architecture consists of three parts, as shown in the image below, and each part performs an individual function.



The Logic Behind LSTM

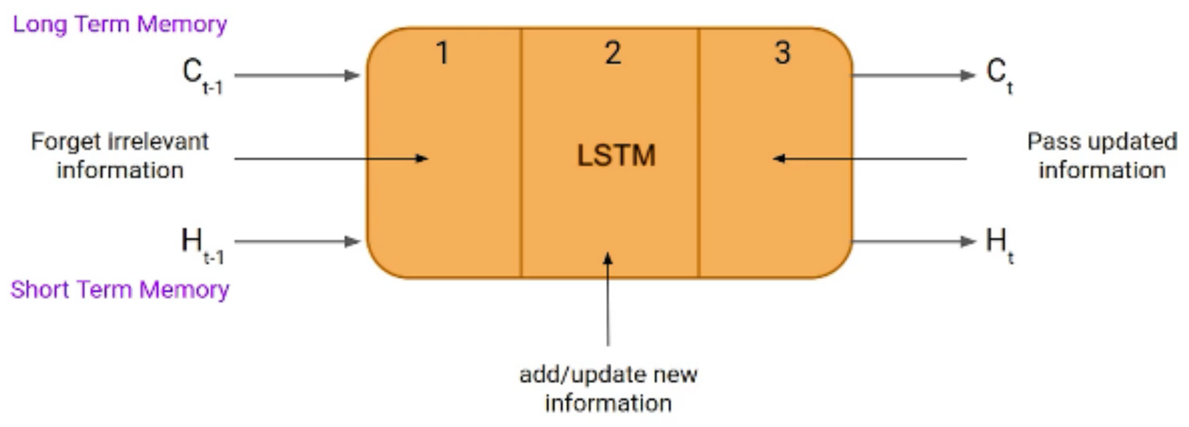
The first part chooses whether the information coming from the previous timestamp is to be remembered or is irrelevant and can be forgotten. In the second part, the cell tries to learn new information from the input to this cell. At last, in the third part, the cell passes the updated information from the current timestamp to the next timestamp. This one cycle of LSTM is considered a single-time step.

These three parts of an LSTM unit are known as gates. They control the flow of information in and out of the memory cell or lstm cell. The first gate is called Forget gate, the second gate is known as the Input gate, and the last one is the Output gate. An LSTM unit that consists of these three gates and a memory cell or lstm cell can be considered as a layer of neurons in traditional feedforward neural network, with each neuron having a hidden layer and a current state.

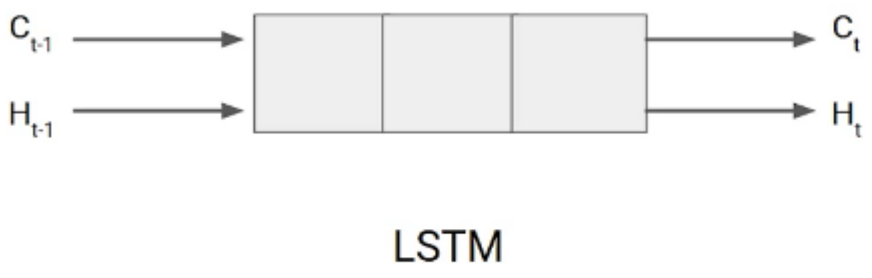


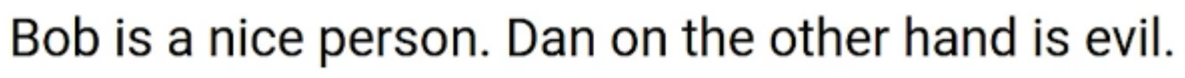
Just like a simple RNN, an LSTM also has a hidden state where H(t-1) represents the hidden state of the previous timestamp and Ht is the hidden state of the current timestamp. In addition to that, LSTM also has a cell state represented by C(t-1) and C(t) for the previous and current timestamps, respectively.

Here the hidden state is known as Short term memory, and the cell state is known as Long term memory. Refer to the following image.



It is interesting to note that the cell state carries the information along with all the timestamps.




Bob is a nice person. Dan on the other hand is evil.

Example of LTSM Working

Let’s take an example to understand how LSTM works. Here we have two sentences separated by a full stop. The first sentence is “Bob is a nice person,” and the second sentence is “Dan, on the Other hand, is evil”. It is very clear, in the first sentence, we are talking about Bob, and as soon as we encounter the full stop(.), we started talking about Dan.

As we move from the first sentence to the second sentence, our network should realize that we are no more talking about Bob. Now our subject is Dan. Here, the Forget gate of the network allows it to forget about it. Let’s understand the roles played by these gates in LSTM architecture.

Generative Adversarial Network

Generative Adversarial Networks (GANs) were developed in 2014 by Ian Goodfellow and his teammates. GAN is basically an approach to generative modeling that generates a new set of data based on training data that look like training data. GANs have two main blocks(two neural networks) which compete with each other and are able to capture, copy, and analyze the variations in a dataset. The two models are usually called Generator and Discriminator which we will cover in Components on GANs.

GAN let’s break it into separate three parts

- Generative – To learn a generative model, which describes how data is generated in terms of a probabilistic model. In simple words, it explains how data is generated visually.
- Adversarial – The training of the model is done in an adversarial setting.
- Networks – use deep neural networks for training purposes.

The generator network takes random input (typically noise) and generates samples, such as images, text, or audio, that resemble the training data it was trained on. The goal of the generator is to produce samples that are indistinguishable from real data.

The discriminator network, on the other hand, tries to distinguish between real and generated samples. It is trained with real samples from the training data and generated samples from the generator. The discriminator’s objective is to correctly classify real data as real and generated data as fake.

The training process involves an adversarial game between the generator and the discriminator. The generator aims to produce samples that fool the discriminator, while the discriminator tries to improve its ability to distinguish between real and generated data. This adversarial training pushes both networks to improve over time.

As training progresses, the generator becomes more adept at producing realistic samples, while the discriminator becomes more skilled at differentiating between real and generated data. Ideally, this process converges to a point where the generator is capable of generating high-quality samples that are difficult for the discriminator to distinguish from real data.

GANs have demonstrated impressive results in various domains, such as image synthesis, text generation, and even video generation. They have been used for tasks like generating realistic images, creating deepfakes, enhancing low-resolution images, and more. GANs have greatly advanced the field of generative modeling and have opened up new possibilities for creative applications in artificial intelligence.

Transformer Networks

Transformer networks are a type of deep learning architecture that have revolutionized the field of natural language processing (NLP). Introduced by Google in 2017, they rely on a mechanism called "self-attention" to understand the relationships between words in a sequence, like a sentence. This allows them to effectively handle long-range dependencies, something that previous architectures like recurrent neural networks (RNNs).

Here's a breakdown of how transformers work

1. Text to Numbers: Text is first converted into numerical representations called tokens.
2. Embedding: Each token is then mapped to a vector using a word embedding. This captures the meaning and relationships between words.
3. Self-Attention: The core of the transformer is the self-attention mechanism. Here, the model attends to different parts of the input sequence to understand how each word relates to the other. Imagine focusing on a specific word in a sentence and considering how other words influence its meaning.
4. Encoding and Decoding: Transformers can be used for various tasks, often involving an encoder-decoder structure. The encoder process takes the input sequence and generates a contextual representation, while the decoder uses this representation to generate the output sequence.

The success of transformers has led to their widespread adoption in various NLP tasks, including

- Machine translation Transformers can translate languages more accurately by considering the entire sentence.
- Text summarization They can capture the key points of a document and generate concise summaries.
- Question answering Transformers can answer your questions by understanding the context of your question and the relevant passage.