

Module 3

Dr. Srabana Pramanik & Ms Mary Divya Shamili

Assistant Professor

IC of Applied Data Science

Dept. SoCSE& IS

Classification Model

- **Classification** :Introduction, Different types of Classification,
- Logistic Regression,
- Support Vector Machines,
- K-Nearest Neighbors,
- Naïve Bayes Classifier,
- Decision Tree Classification,
- Random Forest Classification,
- Evaluation.
-

Classification model

- A classification model is a type of machine learning model that is used to categorize or label data into different classes or categories based on input features.
- Classification models are widely used in a variety of applications, including:
 - 1.Sentiment Analysis: Classifying text as positive, negative, or neutral.
 - 2.Email Spam Detection: Identifying whether an email is spam or not.
 - 3.Image Classification: Categorizing images into various classes, such as recognizing different types of animals in photos.
 - 4.Medical Diagnosis: Predicting whether a patient has a particular disease based on symptoms and test results.
 - 5.Customer Churn Prediction: Determining if a customer is likely to leave a subscription or service.
 - 6.Fraud Detection: Identifying fraudulent transactions in financial data.
 - 7.Handwriting Recognition: Recognizing handwritten characters or digits.
 - 8.Object Detection: Locating and classifying objects within images or videos.

There are different types of classification models, Important ones are :

- Logistic Regression,
 - Support Vector Machines,
 - K-Nearest Neighbors,
 - Naïve Bayes Classifier,
 - Decision Tree Classification,
 - Random Forest Classification,
-
- The choice of classification model depends on the nature of the data, the problem at hand, and the performance requirements.
 - After training, the model can then be used to predict the class of new, unseen data based on the patterns it has learned during training.
 - The model's performance is often evaluated using metrics like accuracy, precision, recall, F1 score, and area under the receiver operating characteristic curve (AUC-ROC), among others.

Logistic Regression

- Logistic regression is a popular method for binary classification, where the goal is to predict the probability of an input belonging to a particular class.
- The logistic regression model uses a logistic function to model the probability of an input belonging to the positive class.
- The logistic function maps any input to a value between 0 and 1, which can be interpreted as a probability.

$$p(y = 1|x) = \frac{1}{1 + e^{-z}} \quad (1)$$

where $z = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n$ is the linear combination of the input features x_1, x_2, \dots, x_n and their corresponding coefficients $\theta_1, \theta_2, \dots, \theta_n$.

Softmax Regression

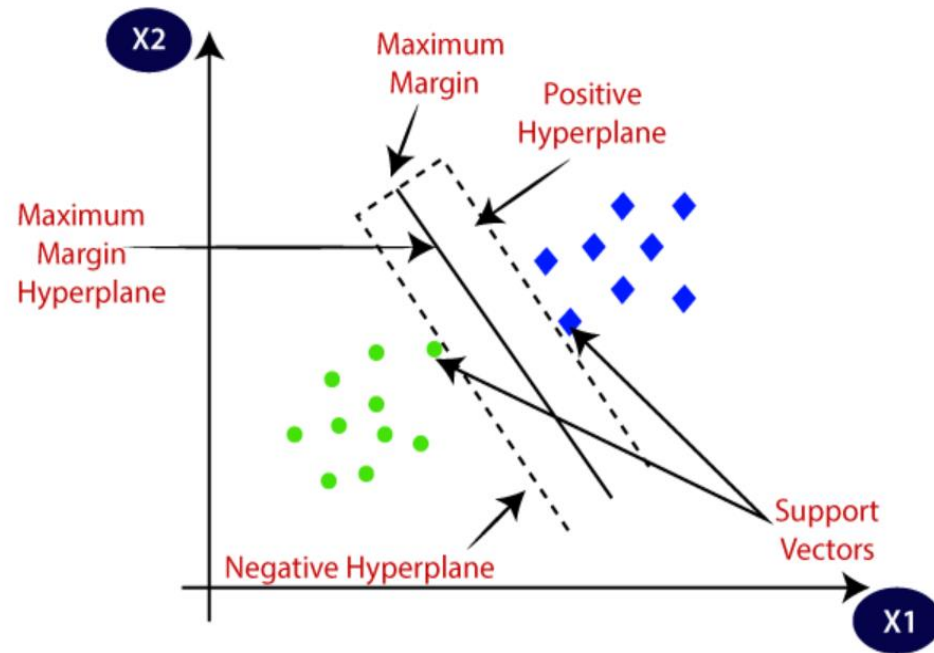
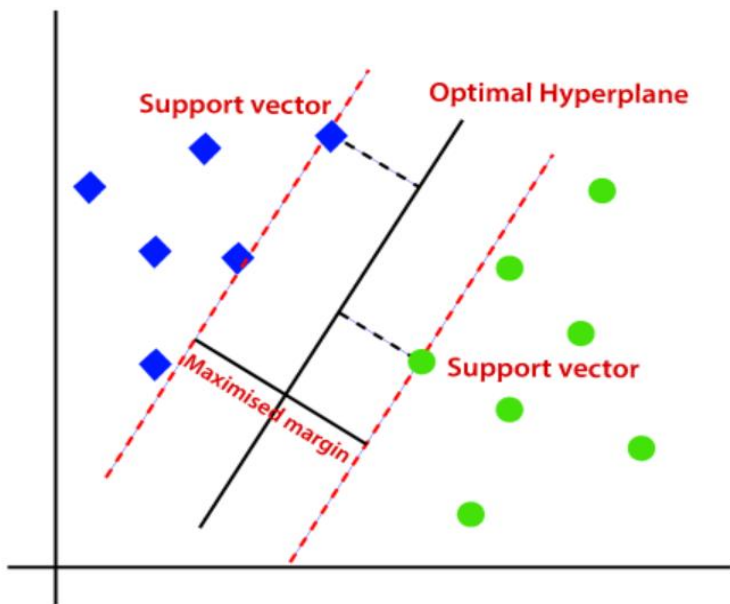
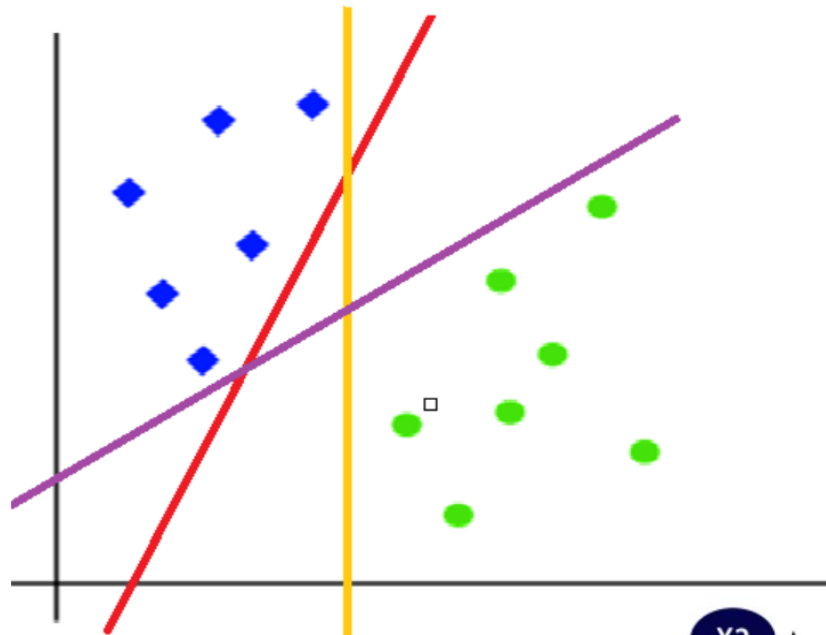
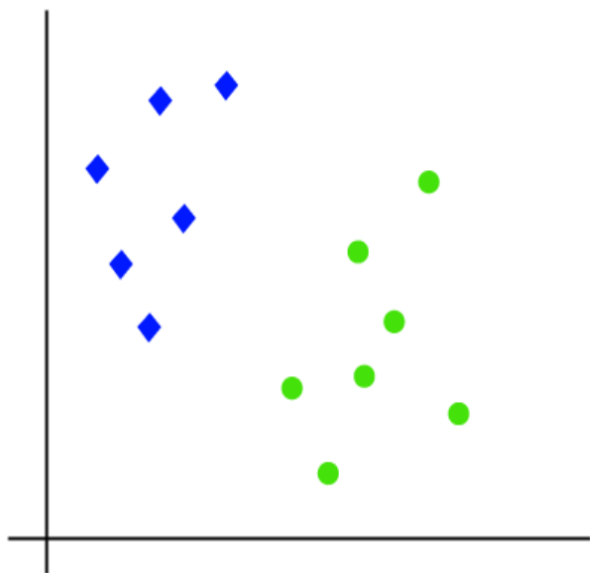
- Softmax regression is a generalization of logistic regression that can be used for multi-class classification problems.
- The softmax function is used to compute the probability of each class, and the class with the highest probability is chosen as the predicted class.
- The softmax function outputs a probability distribution over the classes, and its outputs sum to 1.

$$p(y = i|x) = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}} \quad (2)$$

where K is the number of classes, z_i is the linear combination of input features for class i , and the denominator sums over all classes.

Support Vector Machine

- SVM is a powerful supervised algorithm that works best on smaller datasets but on complex ones.
- Support Vector Machine, abbreviated as SVM can be used for both regression and classification tasks, but generally, they work best in classification problems.
- It is a supervised machine learning problem where we try to find a hyperplane that best separates the two classes.
- SVM and logistic regression both the algorithms try to find the best hyperplane, but the main difference is logistic regression is a probabilistic approach whereas support vector machine is based on statistical approaches.
- There can be an infinite number of hyperplanes passing through a point and classifying the two classes perfectly. So, which one is the best?
- To classify these points, we can have many decision boundaries, but the question is which is the best and how do we find it?
- Since we are plotting the data points in a 2-dimensional graph we call this decision boundary a **straight line** but if we have more dimensions, we call this decision boundary a **“hyperplane”**
- The best hyperplane is that plane that has the maximum distance from both the classes, and this is the main aim of SVM.
- This is done by finding different hyperplanes which classify the labels in the best way then it will choose the one which is farthest from the data points or the one which has a maximum margin.

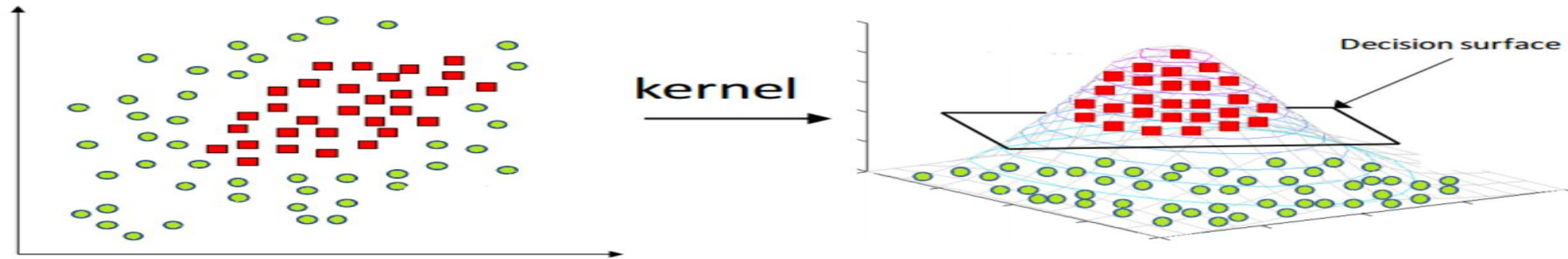


Types of Support Vector Machine Algorithms

- 1. **Linear SVM**
- When the data is perfectly linearly separable only then we can use Linear SVM. Perfectly linearly separable means that the data points can be classified into 2 classes by using a single straight line(if 2D).
- 2. **Non-Linear SVM**
- When the data is not linearly separable then we can use Non-Linear SVM, which means when the data points cannot be separated into 2 classes by using a straight line (if 2D) then we use some advanced techniques like kernel tricks to classify them. In most real-world applications we do not find linearly separable datapoints hence we use kernel trick to solve them.
- **Important Terms**
- **Support Vectors:** These are the points that are closest to the hyperplane. A separating line will be defined with the help of these data points.
- **Margin:** it is the distance between the hyperplane and the observations closest to the hyperplane (support vectors). In SVM large margin is considered a good margin. There are two types of margins **hard margin** and **soft margin**.

Kernel Trick of SVM

- The most interesting feature of SVM is that it can even work with a non-linear dataset and for this, we use “Kernel Trick” which makes it easier to classifies the points. Suppose we have a dataset like this:



- Here we see we cannot draw a single line or say hyperplane which can classify the points correctly. So what we do is try converting this lower dimension space to a higher dimension space using some quadratic functions which will allow us to find a decision boundary that clearly divides the data points. These functions which help us do this are called Kernels and which kernel to use is purely determined by hyperparameter tuning.
- Support Vector Machines are a powerful machine learning algorithm that can handle both linear and non-linearly separable data.
- The kernel trick allows us to extend SVMs to handle complex data, and the soft margin SVM allows us to handle data that is not perfectly separable.
- With these techniques, we can build powerful models for classification and regression tasks.

What is Naive Bayes?

- Thomas Bayes introduced the the Bayes theorem.
- The Naive Bayes classifier works on the principle of [conditional probability](#), as given by the Bayes theorem.
- Consider the following example of tossing two coins. If we toss two coins and look at all the different possibilities, we have the sample space as:
- {HH, HT, TH, TT}
- While calculating the math on probability, we usually denote probability as P. Some of the probabilities in this event would be as follows:
- The probability of getting two heads = $1/4$
- The probability of at least one tail = $3/4$
- The probability of the second coin being head given the first coin is tail = $1/2$
- The probability of getting two heads given the first coin is a head = $1/2$
- The Bayes theorem gives us the conditional probability of event A, given that event B has occurred. In this case, the first coin toss will be B and the second coin toss A.

According to Bayes theorem:

- Let us apply Bayes theorem to our coin example. Here, we have two coins, and the first two probabilities of getting two heads and at least one tail are computed directly from the sample space.
- Now in this sample space, let A be the event that the second coin is head, and B be the event that the first coin is tails.
- Again, we reversed it because we want to know what the second event is going to be.
- We're going to focus on A, and we write that out as a probability of A given B:
- Probability = $P(A|B)$
- $= [P(B|A) * P(A)] / P(B)$
- $= [P(\text{First coin being tail given the second coin is the head}) * P(\text{Second coin being head})] / P(\text{First coin being tail})$
- $= [(1/2) * (1/2)] / (1/2)$
- $= 1/2 = 0.5$
- Bayes theorem calculates the conditional probability of the occurrence of an event based on prior knowledge of conditions that might be related to the event.

$$P(A | B) = \frac{P(B | A)P(A)}{P(B)}$$

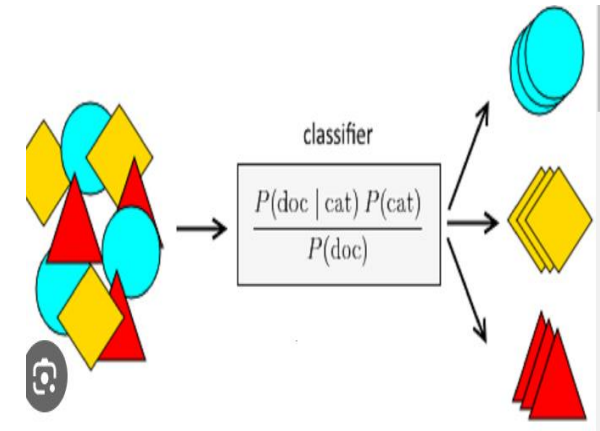
where:

$P(A|B)$ = Conditional Probability of A given B

$P(B|A)$ = Conditional Probability of A given B

$P(A)$ = Probability of event A

$P(B)$ = Probability of event A



1.Convert the data set into a frequency table: In this first step data set is converted into a frequency table

2.Create Likelihood table by finding the probabilities: Create Likelihood table by finding the probabilities like Overcast probability = 0.29 and probability of playing is 0.64.

3.Use Naive Bayesian equation to calculate the posterior probability: Now, use Naive Bayesian equation to calculate the posterior probability for each class. The class with the highest posterior probability is the outcome of the prediction.

Weather	Play
Sunny	No
Overcast	Yes
Rainy	Yes
Sunny	Yes
Sunny	Yes
Overcast	Yes
Rainy	No
Rainy	No
Sunny	Yes
Rainy	Yes
Sunny	No
Overcast	Yes
Overcast	Yes
Rainy	No

Frequency Table		
Weather	No	Yes
Overcast		4
Rainy	3	2
Sunny	2	3
Grand Total	5	9

Likelihood table				
Weather	No	Yes		
Overcast		4	=4/14	0.29
Rainy	3	2	=5/14	0.36
Sunny	2	3	=5/14	0.36
All	5	9		
	=5/14	=9/14		
	0.36	0.64		

Problem

- **Problem:** Players will play if the weather is sunny. Is this statement correct?
- We can solve it using the above-discussed method of posterior probability.
- $P(\text{Yes} \mid \text{Sunny}) = P(\text{Sunny} \mid \text{Yes}) * P(\text{Yes}) / P(\text{Sunny})$
- Here $P(\text{Sunny} \mid \text{Yes}) * P(\text{Yes})$ is in the numerator, and $P(\text{Sunny})$ is in the denominator.
- Here we have $P(\text{Sunny} \mid \text{Yes}) = 3/9 = 0.33$, $P(\text{Sunny}) = 5/14 = 0.36$, $P(\text{Yes}) = 9/14 = 0.64$
- Now, $P(\text{Yes} \mid \text{Sunny}) = 0.33 * 0.64 / 0.36 = 0.60$, which has higher probability.

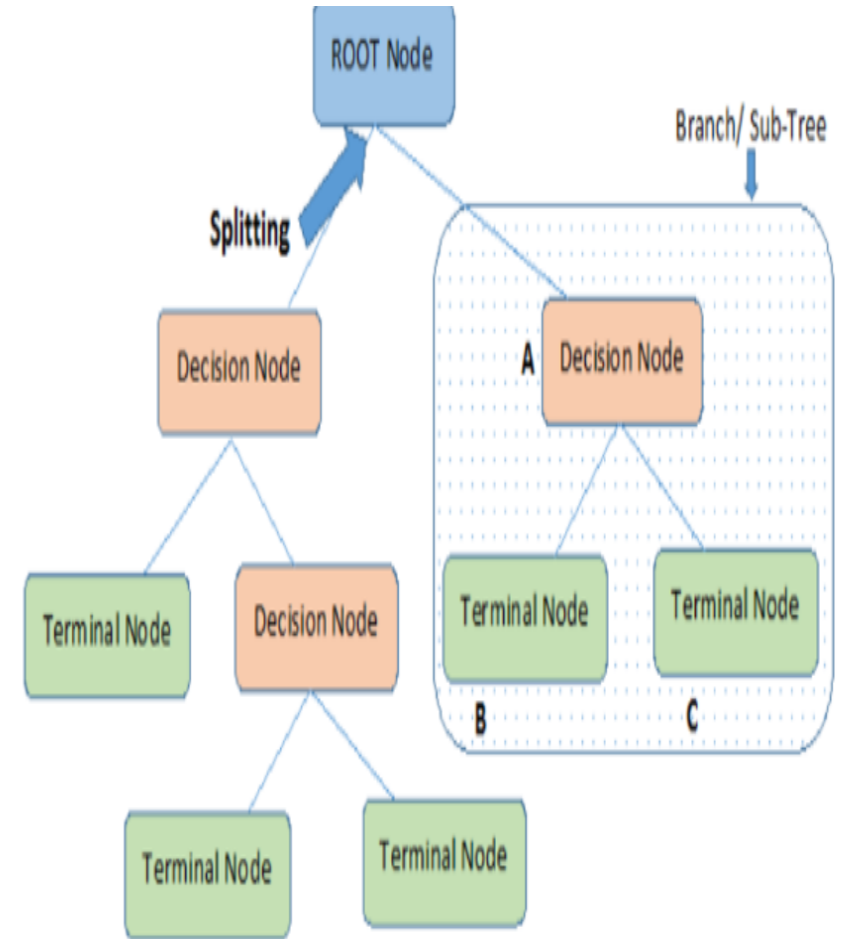
Decision tree:

- Decision tree , can be used for classification and regression analysis. It is a supervised learning model.
- The purpose of a decision tree is to make decisions or predictions by learning from past data.
- It helps to understand the relationships between input variables and their outcomes and identify the most significant features that contribute to the final decision.
- It is graphical representation of all possible solution.
- The decision tree classifier (Pang-Ning et al., 2006) creates the classification model by building a decision tree.
- Each node in the tree specifies a test on an attribute, each branch descending from that node corresponds to one of the possible values for that attribute.
- Decision tree learning employs a **divide and conquer strategy** by conducting a greedy search to identify the optimal split points within a tree.
- This process of splitting is then repeated in a **top-down**, recursive manner until all, or the majority of records have been classified under specific class labels.
- A decision tree is a tree-like structure that represents a series of decisions and their possible consequences. It is used in machine learning for classification and regression tasks.
- An example of a decision tree is a flowchart that helps a person decide what to wear based on the weather conditions.
- The best algorithm for decision trees depends on the specific problem and dataset.
- Popular decision tree algorithms include ID3, C4.5, CART, and Random Forest.
- Random Forest is considered one of the best algorithms as it combines multiple decision trees to improve accuracy and reduce overfitting.

•

Decision Tree Terminologies

- **Root Node:** The initial node at the beginning of a decision tree, where the entire population or dataset starts dividing based on various features or conditions.
- **Decision Nodes:** Nodes resulting from the splitting of root nodes are known as decision nodes. These nodes represent intermediate decisions or conditions within the tree.
- **Leaf Nodes:** Nodes where further splitting is not possible, often indicating the final classification or outcome. Leaf nodes are also referred to as terminal nodes.
- **Sub-Tree:** Similar to a **subsection** of a graph being called a sub-graph, a sub-section of a decision tree is referred to as a sub-tree. It represents a specific portion of the decision tree.
- **Pruning:** The process of removing or cutting down specific nodes in a decision tree to prevent overfitting and simplify the model.
- **Branch / Sub-Tree:** A **subsection** of the entire decision tree is referred to as a branch or sub-tree. It represents a specific path of decisions and outcomes within the tree.
- **Parent and Child Node:** In a decision tree, a node that is divided into sub-nodes is known as a parent node, and the sub-nodes emerging from it are referred to as child nodes. The parent node represents a decision or condition, while the child nodes represent the potential outcomes or further decisions based on that condition.



Decision Tree Assumptions: some common assumption and consideration when creating decision trees:

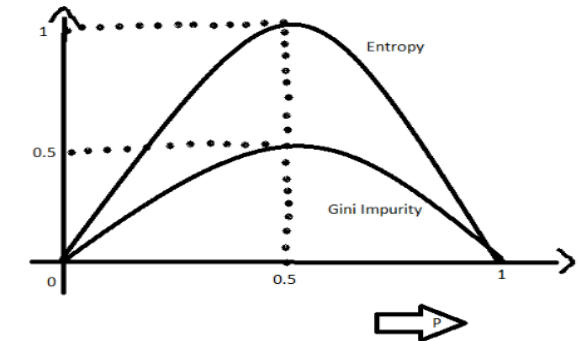
- **Entropy:** Entropy is nothing but the uncertainty in our dataset or measure of disorder.

The formula for Entropy is shown below:

$$E(S) = -p_{(+)} \log p_{(+)} - p_{(-)} \log p_{(-)}$$

Here,

- p_{+} is the probability of positive class
- p_{-} is the probability of negative class
- S is the subset of the training example



$$E(S) = -\sum_{i=1}^c p_i \log_2 p_i$$

$$Gini(E) = 1 - \sum_{j=1}^c p_j^2$$

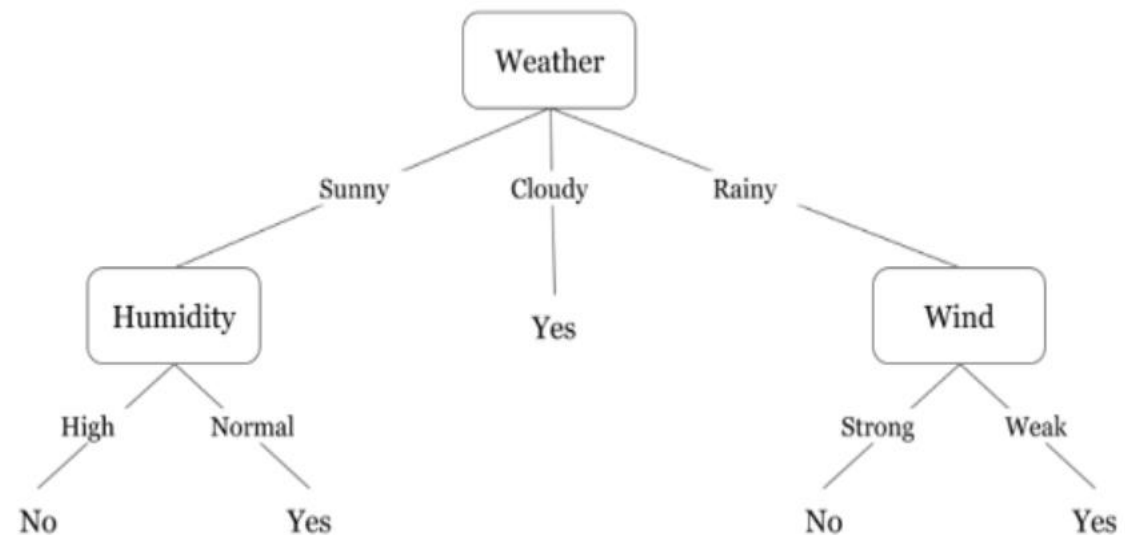
- **The Gini Impurity formula** is: $1 - (p_1)^2 - (p_2)^2$, where p_1 and p_2 represent the probabilities of the two classes in a binary classification problem.
- If the dataset is big then you should use Gini impurity otherwise entropy (because entropy contains logarithm part)
- **The information gain** $G(S,A)$ where A is an attribute
- $G(S,A) \equiv E(S) - \sum_{v \text{ in Values}(A)} \left(\frac{|S_v|}{|S|} \right) * E(S_v)$

Decision Algorithm

- **Building a Decision Tree**

1. First test all attributes and select the one that would function as the best root by following Information gain;
2. Break-up the training set into subsets based on the branches of the root node by following entropy or Gini impurity;
3. Test the remaining attributes to see which ones fit best underneath the branches of the root node by following Information gain again;
4. Continue this process for all other branches until
 - a. all examples of a subset are of one type or reach an leaf node.
 - b. there are no examples left (return majority classification of the parent)
 - c. there are no more attributes left (default value should be majority classification)

Day	Weather	Temperature	Humidity	Wind	Play?
1	Sunny	Hot	High	Weak	No
2	Cloudy	Hot	High	Weak	Yes
3	Sunny	Mild	Normal	Strong	Yes
4	Cloudy	Mild	High	Strong	Yes
5	Rainy	Mild	High	Strong	No
6	Rainy	Cool	Normal	Strong	No
7	Rainy	Mild	High	Weak	Yes
8	Sunny	Hot	High	Strong	No
9	Cloudy	Hot	Normal	Weak	Yes
10	Rainy	Mild	High	Strong	No



Pruning

- Usually, real-world datasets have a **large number of features**, which will result in a **large number of splits**, which in turn gives a huge tree.
- Such trees take time to build and can lead to overfitting.
- **That means the tree will give very good accuracy on the training dataset but will give bad accuracy in test data.**
- **There are many ways to tackle this problem through hyperparameter tuning.**
- Pruning is another method that can help us avoid overfitting.
- It helps in improving the performance of the tree by **cutting the nodes or sub-nodes** which are not significant. Additionally, it **removes the branches** which have **very low importance**.
- There are mainly 2 ways for pruning:
- **Pre-pruning** – we can stop growing the tree earlier, which means we can prune/remove/cut a node if it has low importance **while growing** the tree.
- **Post-pruning** – Once our **tree is built to its depth**, we can start pruning the nodes based on their significance.
- **max_depth** : We can set the maximum depth of our decision tree using the **max_depth** parameter.
- The more the value of **max_depth**, the more complex your tree will be.
- The training error will off-course decrease if we increase the **max_depth** value but when our test data comes into the picture, we will get a very bad accuracy.
- Hence you need a value that will not overfit as well as underfit our data and for this, you can use GridSearchCV.
- The GridSearchCV is used to search the best combination of these hyperparameters which are best for the model.
- **min_samples_leaf** – **represents the minimum number of samples required to be in the leaf node.** The more you increase the number, the more is the possibility of overfitting.
- **max_features** – **it helps us decide what number of features to consider when looking for the best split.**
- **min_samples_split**: **The minimum number of samples required to split an internal node.** If the number of samples at a node is less than 'min_samples_split', the node won't be split, and it will become a leaf node.

Random Forest technique

- Random forest is an ensemble model using bagging as the ensemble method and decision tree as the individual model.
- Each tree in the forest outputs a prediction and the most voted becomes the output of the model.
- This is helpful to make the model with more accuracy and stable, preventing overfitting.

Steps Involved in Random Forest Algorithm

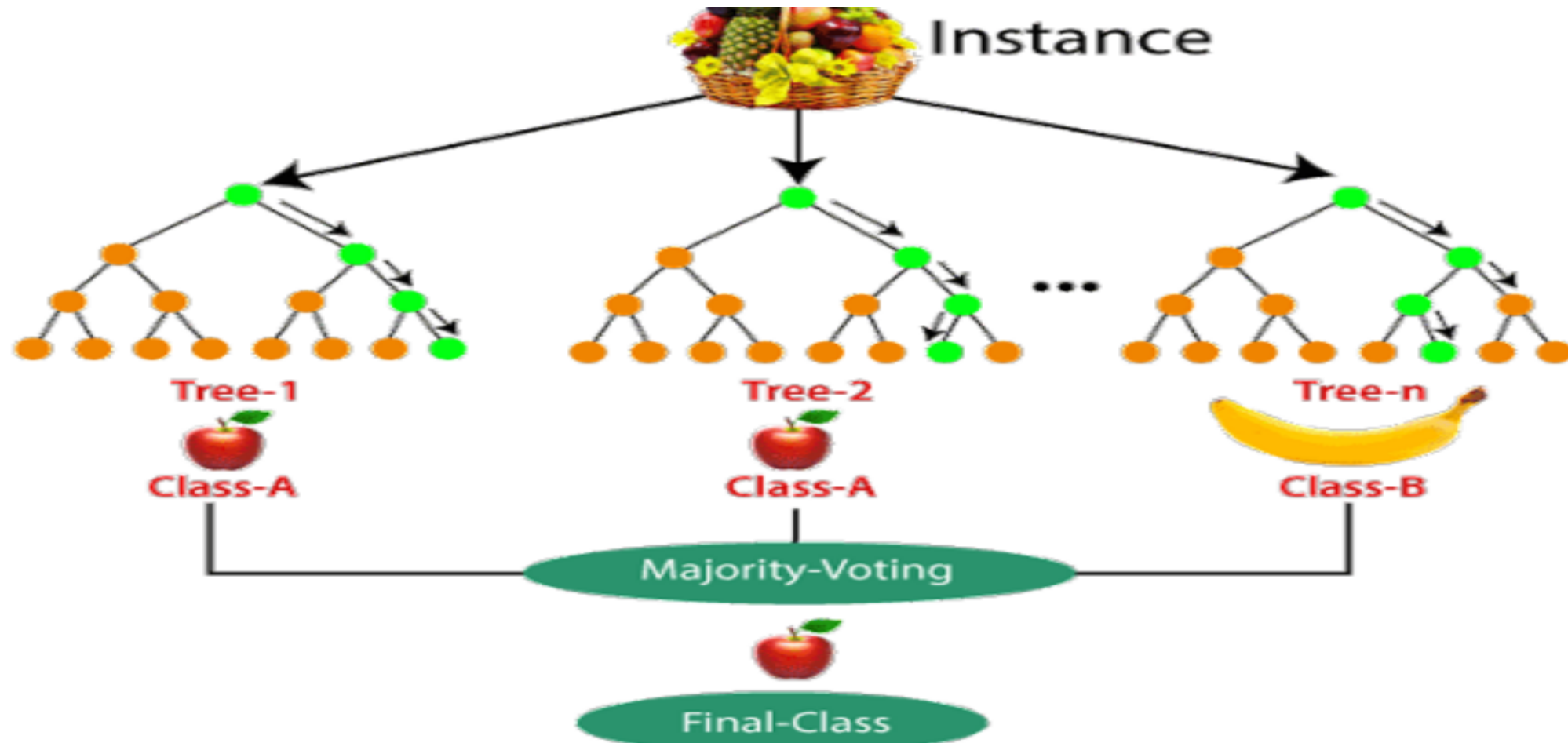
Step 1: In the Random forest model, a subset of data points and a subset of features are selected for constructing each sample or subset of training data. Simply put, n random records and m features are taken from the data set having k number of records.

Step 2: Individual decision trees are constructed for each sample.

Step 3: Each decision tree will generate an output.

Step 4: Final output is considered based on ***Majority Voting or Averaging***.

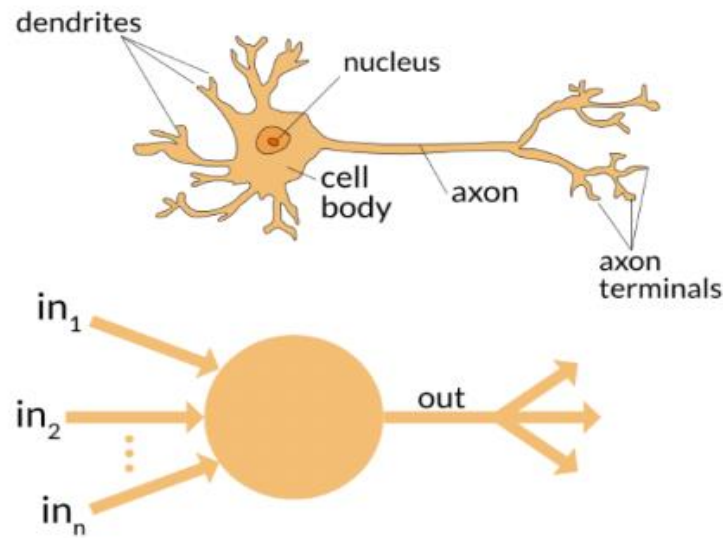
Random Forest technique



The Concept of Artificial Neurons (Perceptrons) in Neural Networks

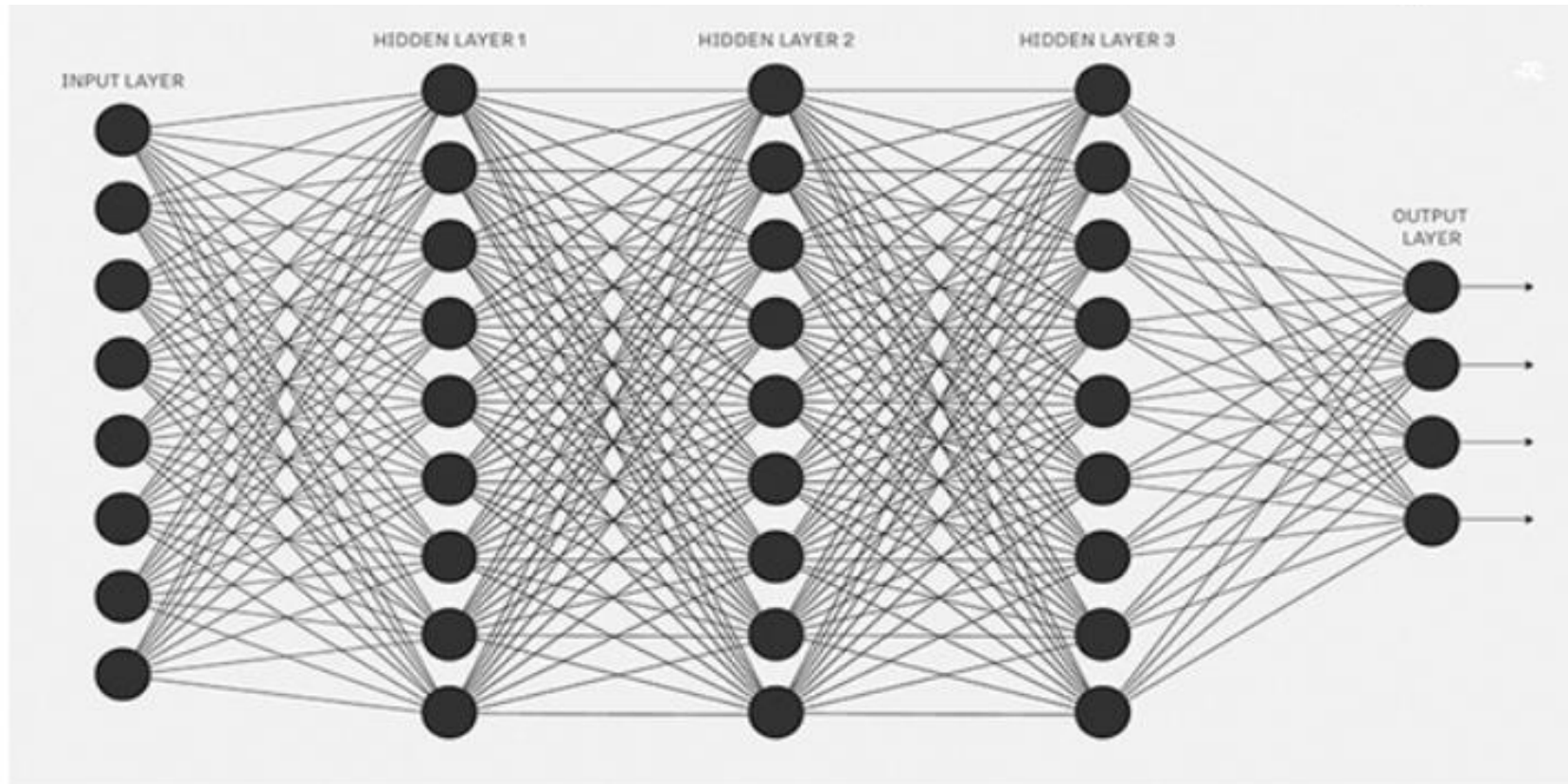
- **Artificial neurons** (also called **Perceptrons**, **Units** or **Nodes**) are the simplest elements or building blocks in a neural network. They are inspired by biological neurons that are found in the human brain.

The brain receives the stimulus from the outside world, does the processing on the input, and then generates the output. As the task gets complicated, multiple neurons form a complex network, passing information among themselves.



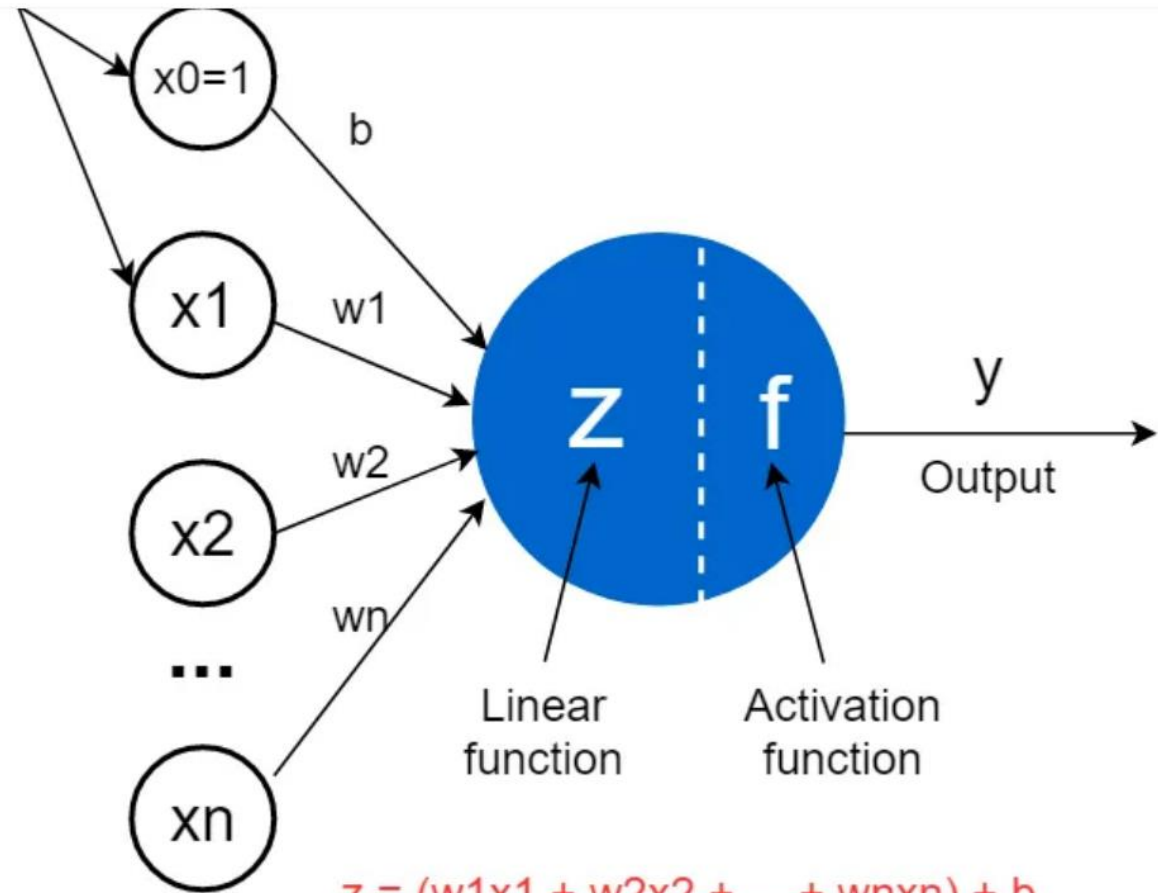
NN network

An Artificial Neural Network tries to mimic a similar behavior. The network you see below is a neural network made of interconnected neurons. Each neuron is characterized by its weight, bias and activation function.



The structure of a Perceptron

A perceptron takes the inputs, x_1, x_2, \dots, x_n , multiplies them by weights, w_1, w_2, \dots, w_n and adds the bias term, b , then computes the linear function, z on which an activation function, f is applied to get the output, y .



$$z = (w_1x_1 + w_2x_2 + \dots + w_nx_n) + b$$

$$y = f(z)$$

The input is fed to the input layer, the neurons perform a linear transformation on this input using the weights and biases.

$$x = (\text{weight} * \text{input}) + \text{bias}$$

Post that, an activation function is applied on the above result.

$$Y = \text{Activation}(\Sigma(\text{weight} * \text{input}) + \text{bias})$$

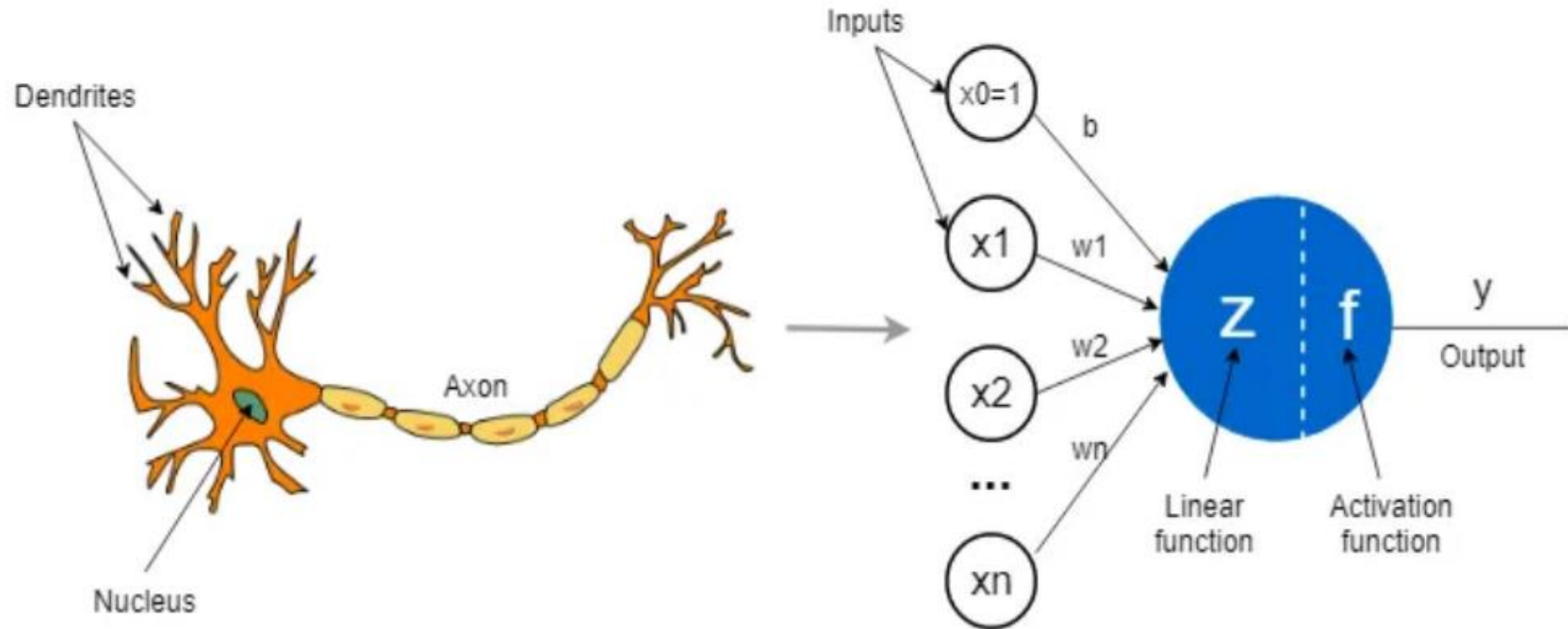
Finally, the output from the activation function moves to the next hidden layer and the same process is repeated. This forward movement of information is known as the ***forward propagation***.

What if the output generated is far away from the actual value? Using the output from the forward propagation, error is calculated. Based on this error value, the weights and biases of the neurons are updated. This process is known as ***back-propagation***.

How Perceptrons are inspired by biological neurons?

- An artificial neuron as a mathematical model was inspired by a biological neuron.
- A biological neuron receives its input signals from other neurons through **dendrites** (small fibers). Likewise, a perceptron receives its data from other perceptrons through **input neurons** that take numbers.
- The connection points between dendrites and biological neurons are called **synapses**. Likewise, the connections between inputs and perceptrons are called **weights**. They measure the importance level of each input.
- In a biological neuron, the **nucleus** produces an output signal based on the signals provided by dendrites. Likewise, the **nucleus** (colored in blue) in a perceptron performs some calculations based on the input values and produces an output.
- In a biological neuron, the output signal is carried away by the **axon**. Likewise, the axon in a perceptron is the **output value** which will be the input for the next perceptrons.

Biological neuron to Perceptron



Inside a perceptron

- A perceptron usually consists of two mathematical functions.
- 1) Perceptron's linear function
- 2) Perceptron's non-linear (activation) function

Perceptron's linear function- is the weighted sum of the inputs plus bias unit and can be calculated as follows.

$$z = (w_1.x_1 + w_2.x_2 + \dots + w_n.x_n) + b$$

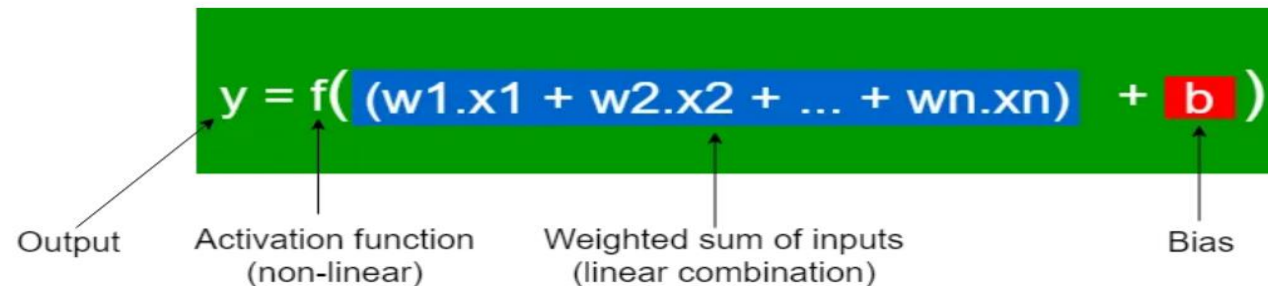
The **x1**, **x2**, ..., **xn** are inputs that take numerical values. There can be several (finite) inputs for a single neuron. They can be raw input data or outputs of the other perceptrons.

Inside a perceptron.....

- The **w1, w2, ..., wn** are **weights** that take numerical values and control the level of importance of each input. The higher the value, the more important the input.
- **w1.x1 + w2.x2 + ... + wn.xn** is called the weighted sum of inputs.
- The **b** is called the **bias term** or **bias unit** that also takes a numerical value. It is added to the weighted sum of inputs. The purpose of including a bias term is to shift the activation function of each perceptron to not get a zero value. In other words, if all **x1, x2, ..., xn** inputs are 0, the **z** is equal to the value of bias.
- The weights and biases are called the **parameters** in a neural network model. The optimal values for those parameters are found during the learning (training) process of the neural network.

Perceptron's non-linear (activation) function

The purpose of an activation function is to introduce non-linearity to the network. Without an activation function, a neural network can only model linear relationships and can't model non-linear relationships present in the data. Most of the relationships are non-linear in real-world data. Therefore, neural networks would be useless without activation functions.



The diagram shows the equation $y = f(w_1.x_1 + w_2.x_2 + \dots + w_n.x_n) + b$ on a green background. The expression $(w_1.x_1 + w_2.x_2 + \dots + w_n.x_n)$ is highlighted in blue, and the bias b is highlighted in red. Arrows point from labels below to the corresponding parts of the equation: 'Output' points to y , 'Activation function (non-linear)' points to f , 'Weighted sum of inputs (linear combination)' points to the blue-highlighted sum, and 'Bias' points to b .

$$y = f(w_1.x_1 + w_2.x_2 + \dots + w_n.x_n) + b$$

Output

Activation function (non-linear)

Weighted sum of inputs (linear combination)

Bias

This is also called the non-linear component of the perceptron. It is denoted by f . It is applied on z to get the output y based on the type of activation function we use.

What does it mean by “firing a neuron”?

- For this, consider the following **binary step** activation function which is also known as the **threshold activation function**. We can set any value to the threshold and here we specify the value 0.
- We say that a neuron or perceptron fires (or activates) only when the value of **z** exceeds the threshold value, 0
- Therefore, the type of activation function determines how the neuron activates or fires and the bias term **b** controls the ease of firing. Now consider the linear function, **z**.
- **$z = (w1.x1 + w2.x2 + \dots + wn.xn) + b$**
 $z = (\text{weighted sum of inputs}) + \text{bias}$
- Let's assume bias is -2. Here also, we consider the binary step activation function. So, the neuron fires (activates) only when the weighted sum of inputs exceeds +2.

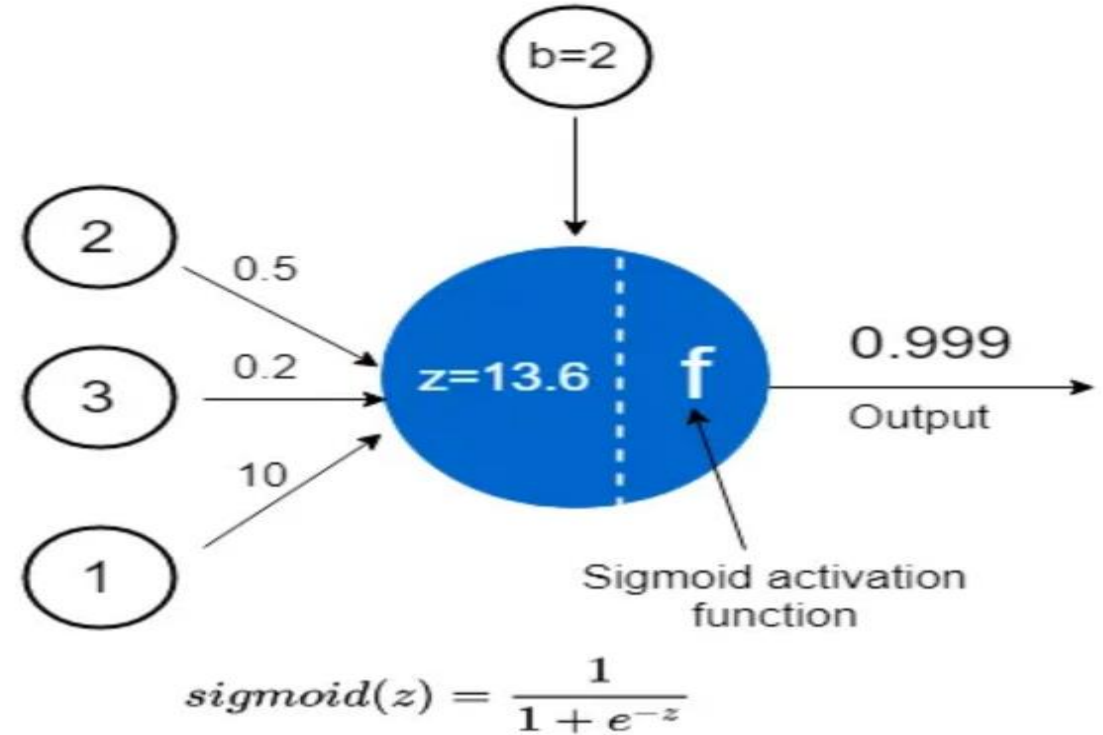
$$\text{binary step}(z) = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{if } z \leq 0 \end{cases}$$

Use of Sigmoid Function as Activation Function

$y = \text{sigmoid}(13.6)$

$y = 0.999$

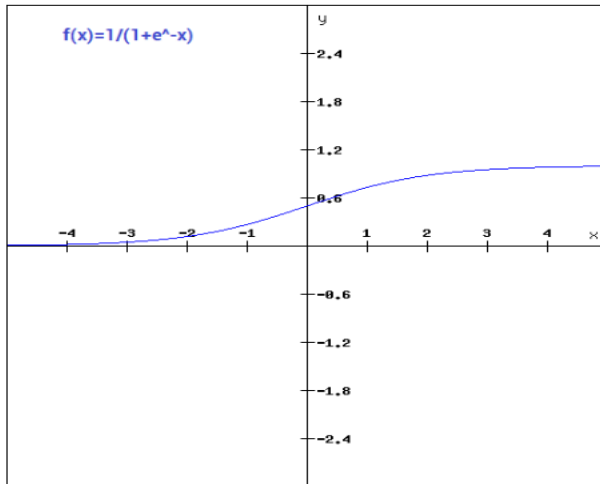
$y \sim 1$



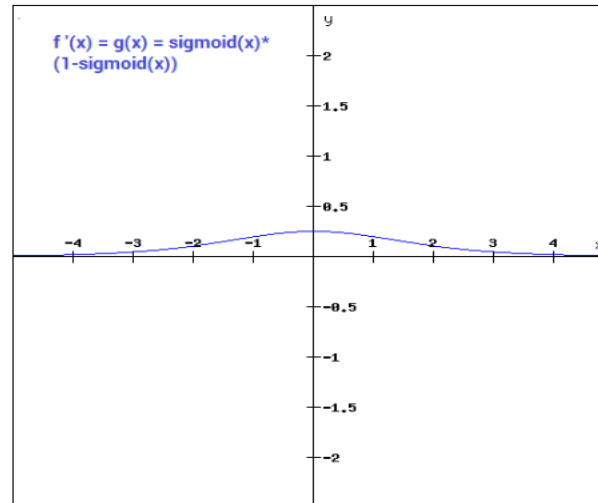
Sigmoid function-non linear function

It is one of the most widely used non-linear activation function. Sigmoid transforms the values between the range 0 and 1. Here is the mathematical expression for sigmoid-

$$f(x) = 1/(1+e^{-x})$$



$$f'(x) = \text{sigmoid}(x) * (1 - \text{sigmoid}(x))$$



```
import numpy as np
def sigmoid_function(x):
    z = (1/(1 + np.exp(-x)))
    return z
```

```
sigmoid_function(7), sigmoid_function(-22)
```

Output:

```
(0.9990889488055994, 2.7894680920908113e-10)
```

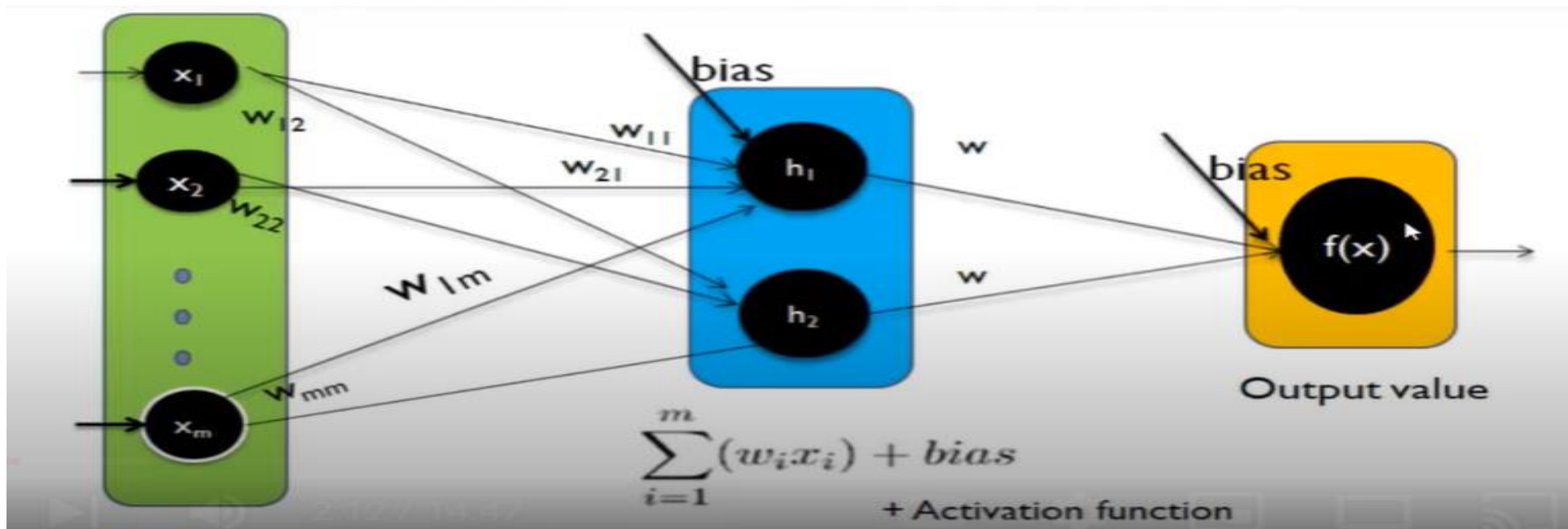
The gradient values are significant for range -3 and 3 but the graph gets much flatter in other regions. This implies that for values greater than 3 or less than -3, will have very small gradients. As the gradient value approaches zero, the network is not really learning.

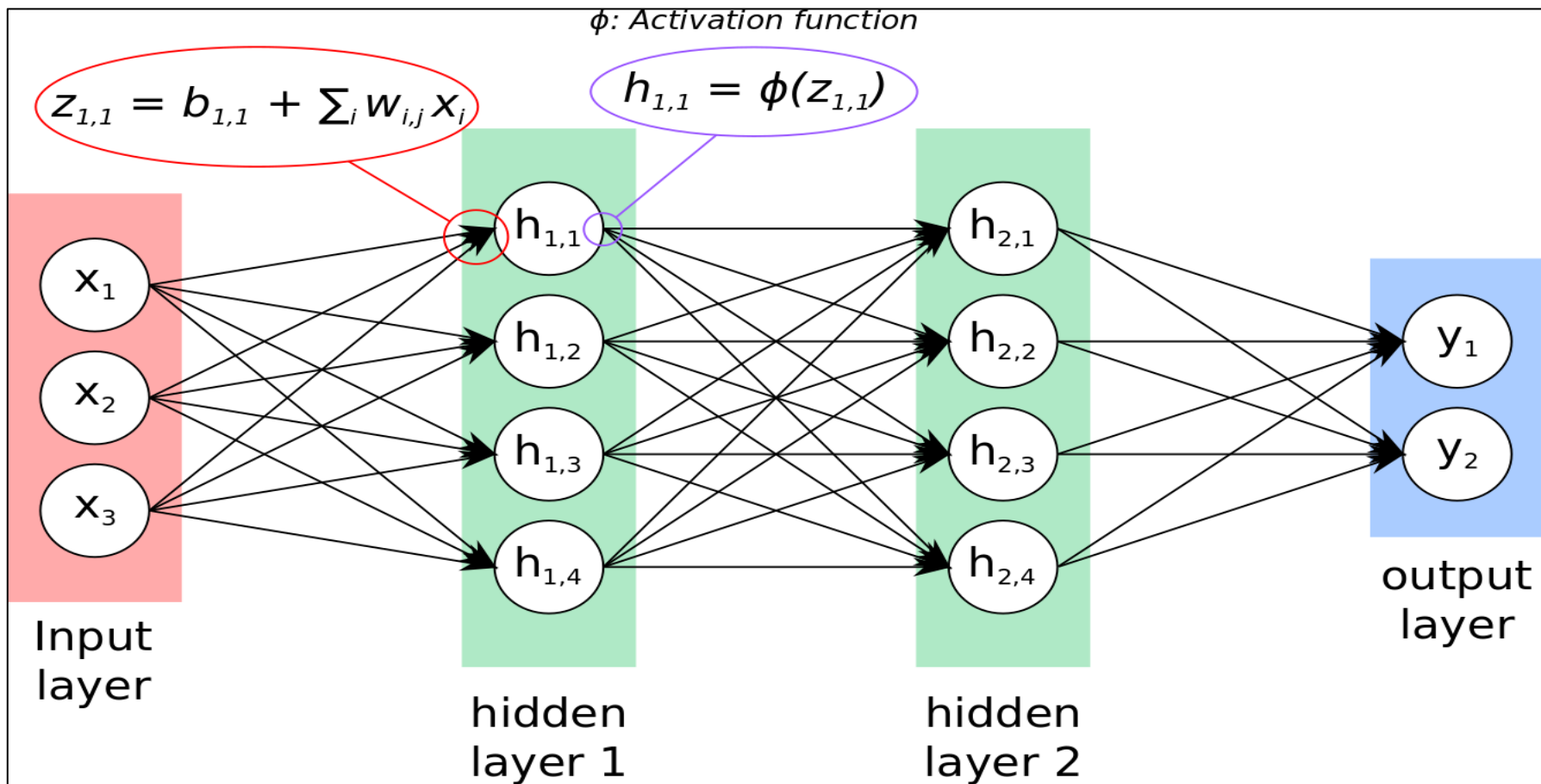
Multi-layer Perceptron

- MLPs are used in a wide range of applications, including image classification, speech recognition, and natural language processing.
- However, they can be computationally expensive and require a large amount of data to train effectively.
- To improve the performance of MLPs, various techniques such as regularization, dropout, and batch normalization can be used.
- These techniques help to prevent overfitting, reduce the impact of noisy data, and improve the convergence rate of the algorithm.

Multi-layer Perceptron

- Components: Input layer, Hidden layers, Output layer
- All are fully connected. Inputs of hidden layer nodes are associated with random weights.
- For hidden layer node, Relu or Tanh function is used as activation function.
- The output of hidden layer again associated with some weight which is assigned randomly.
- For every node bias is also assigned.
- The final out is go through the activation function which is selected depends on the classification or regression problem.
- If we are doing some binary classification problem then we can select sigmoid function, step function etc. In case of multiple classification problem, we can choose SoftMax function, relu, etc.





Steps to train the model and modify the weights of all node

- For every epoch,
 1. Training data is propagated to the MLP through input layers. It passes through the hidden layers, if any forwarding outputs of activation functions to the next layer. Finally the output is generated at the output layer by applying activation functions.
 2. The predicted output will be compared with actual output and hence error will be calculated.
 3. If $\text{error} > 0$, apply backpropagation methodology to modify weights starting from output layer moving towards input layer.
 4. Check accuracy score. If satisfied, stop. Else, go to step 1.

Model evaluation: Confusion Matrix

- A confusion matrix is a table that is often used to evaluate the performance of a classification algorithm. It summarizes the performance of a classification algorithm on a set of data for which the true values are known. The matrix has four entries:

- 1.True Positives (TP):** The number of instances that were correctly predicted as positive.
 - 2.True Negatives (TN):** The number of instances that were correctly predicted as negative.
 - 3.False Positives (FP):** The number of instances that were incorrectly predicted as positive (i.e., the model predicted positive, but the true class is negative).
 - 4.False Negatives (FN):** The number of instances that were incorrectly predicted as negative (i.e., the model predicted negative, but the true class is positive).
- From these four values, various performance metrics can be calculated, including:

		Actual Values	
		Positive (1)	Negative (0)
Predicted Values	Positive (1)	TP	FP
	Negative (0)	FN	TN

- **Accuracy:** This metric measures the overall correctness of the model's predictions. It is the ratio of correctly predicted instances (**both true positives and true negatives**) to **the total number of instances**.
- **Accuracy:** $(TP + TN) / (TP + TN + FP + FN)$
- **Precision (Positive Predictive Value):** Precision is the ratio of correctly **predicted positive observations** to the **total predicted positives**. It is a measure of the accuracy of the positive predictions.
- **Precision (Positive Predictive Value):** $TP / (TP + FP)$
- **Recall (Sensitivity or True Positive Rate):** Recall is the ratio of correctly **predicted positive observations** to the **total actual positives**. It is a measure of the model's ability to capture all the relevant cases.
- **Recall (Sensitivity or True Positive Rate):** $TP / (TP + FN)$
- **Specificity (True Negative Rate):** Specificity is the ratio of correctly **predicted negative observations** to the **total actual negatives**. It is a measure of the model's ability to correctly identify the negative cases.
- **Specificity (True Negative Rate):** $TN / (TN + FP)$
- **F1 Score:** The F1 score is the **harmonic mean of precision and recall**. It provides a balance between precision and recall and is particularly useful **when the class distribution is imbalanced**.
- **F1 Score:** $2 * (Precision * Recall) / (Precision + Recall)$
- These metrics help in assessing how well a classification algorithm is performing, especially in binary classification problems.
- The confusion matrix is also applicable to multi-class classification problems, where the matrix will have more rows and columns corresponding to different classes.

Model Evaluation: AUC-ROC: Area Under the Receiver Operating Characteristic curve

- Area Under the Receiver Operating Characteristic curve (AUC-ROC) is a commonly used metric for evaluating the performance of a classification model.
- The ROC curve is a graphical representation of the trade-off between the true positive rate (sensitivity) and false positive rate (1-specificity) for different threshold values.
- Here's a brief explanation of AUC-ROC and how it is used for model evaluation:

1. Receiver Operating Characteristic (ROC) Curve:

1. The ROC curve is a graphical representation of a model's ability to discriminate between positive and negative classes across different threshold values.
2. The x-axis represents the false positive rate (1-specificity), and the y-axis represents the true positive rate (sensitivity).
3. The curve is generated by varying the threshold for classification, and each point on the curve corresponds to a different threshold.

2. Area Under the Curve (AUC):

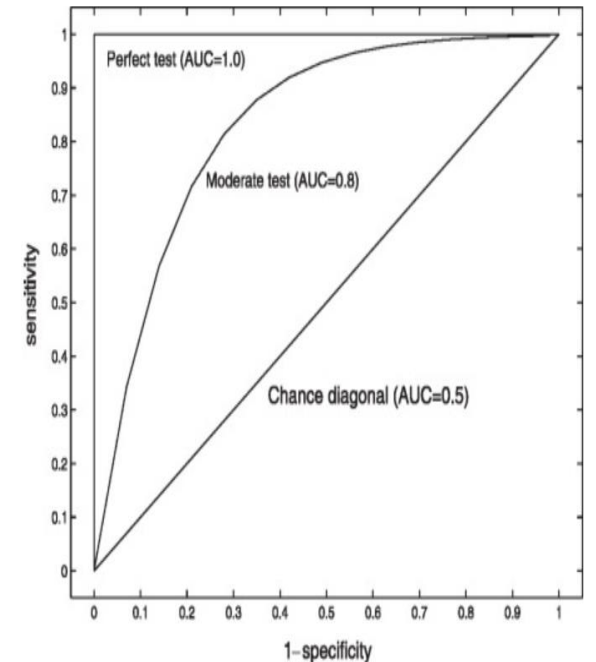
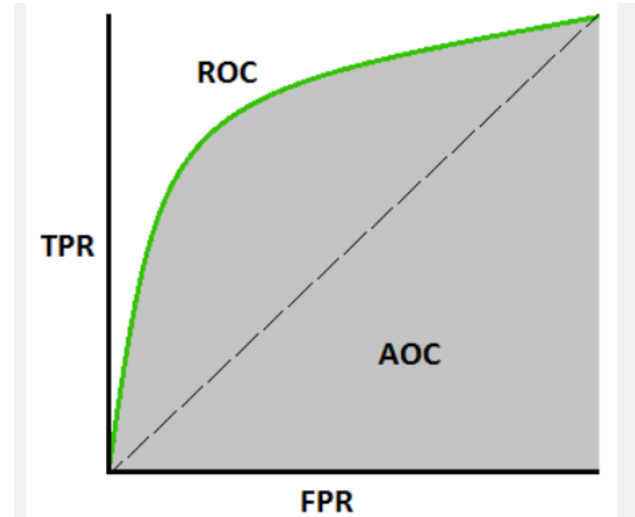
1. AUC-ROC quantifies the overall performance of a classification model.
2. The AUC is the area under the ROC curve. A perfect model has an AUC of 1.0, while a random model has an AUC of 0.5.
3. Higher AUC values indicate better discrimination ability of the model.

3. Interpretation of AUC:

1. AUC is often interpreted as the probability that the model will rank a randomly chosen positive instance higher than a randomly chosen negative instance.
2. AUC is scale-invariant and is not affected by class imbalance.

4. Model Evaluation:

1. Generally, a model with an AUC between 0.5 and 0.7 is considered poor, between 0.7 and 0.8 is acceptable, between 0.8 and 0.9 is good, and above 0.9 is excellent.
2. It's important to consider the specific context of the problem and the requirements of the application when interpreting AUC.



- Thank You