AIMLT Active Learning
T.Y. CSE – CSF (Batch A1)

Optimization in Machine Learning

Name Of Participants:
Gaurav Choudhary (PA11)
1032210933
Sahaj Mishra (PA16)
1032211066
Sayyam Saboo (PA17)
1032211097

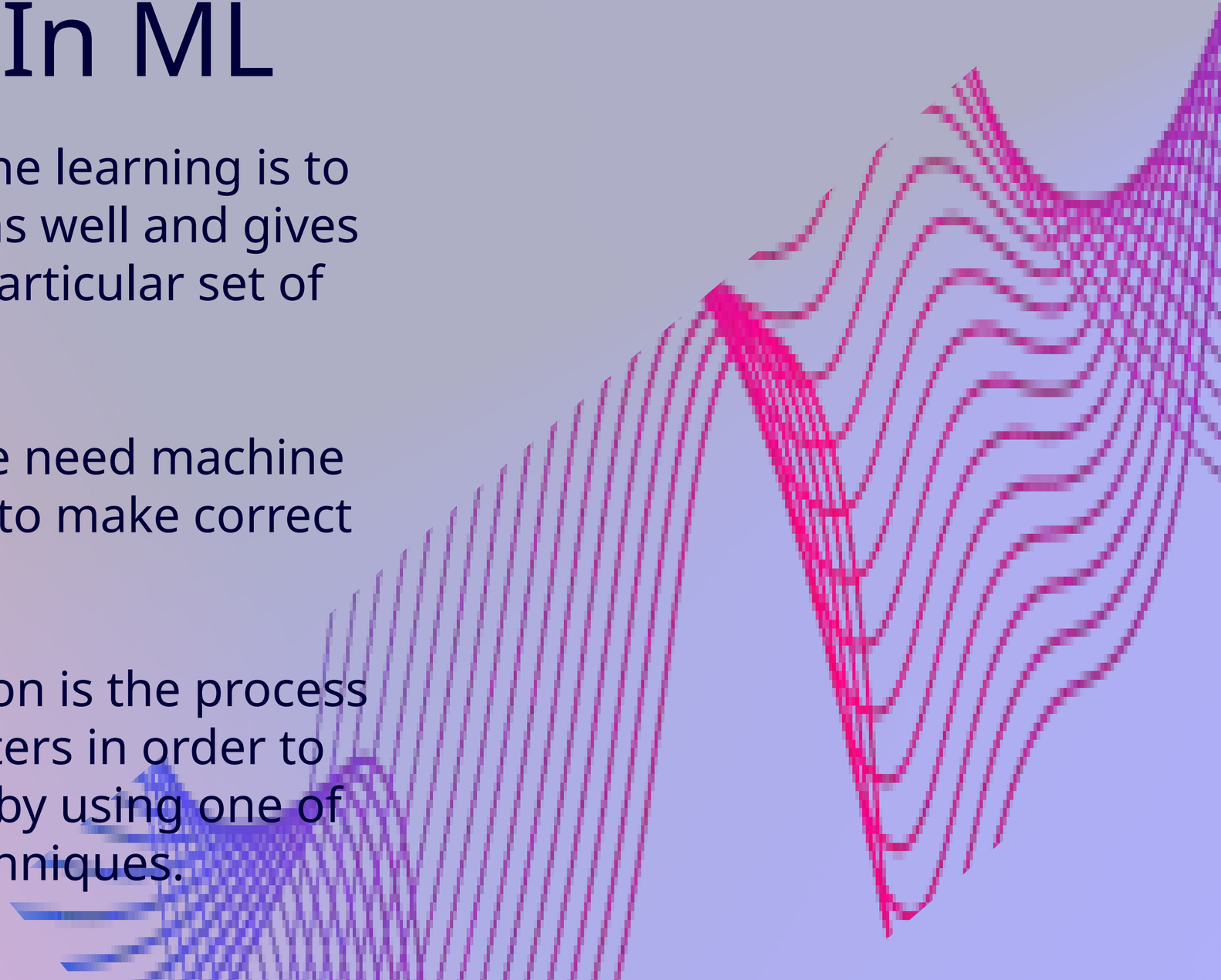
Contents

Optimization In ML

The principal goal of machine learning is to create a model that performs well and gives accurate predictions in a particular set of cases.

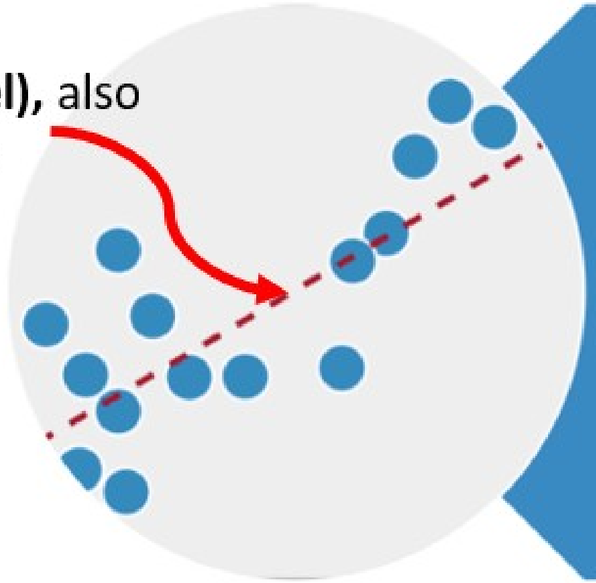
In order to achieve that, we need machine learning optimization so as to make correct predictions.

Machine learning optimization is the process of adjusting hyperparameters in order to minimize the cost function by using one of the optimization techniques.



Where is Optimization used in ML?

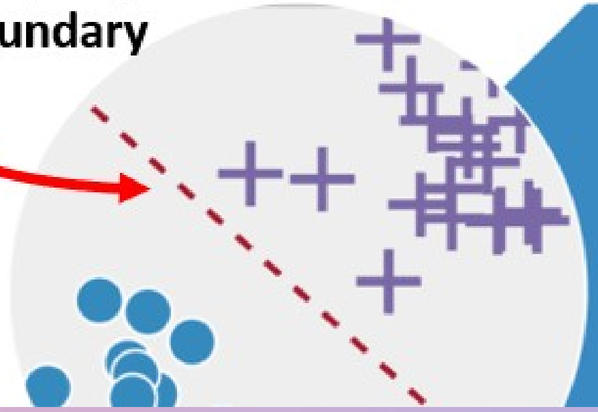
Regression Line (Model), also called the **Best-Fit Line**



Regression

- Linear Regression
- Random Forest
- Multi-layer Perceptron
- AdaBoost
- Gradient Boosting
- Convolutional Neural Networks

Separating Hyperplane (SHP), also called **Decision Boundary**



Classification

- Logistic Regression
- Decision Tree
- KNN
- Support vector machines

Name of Regression Algorithm	Loss (Cost) Function minimized
Linear Regression	Mean Square Error (MSE)
Polynomial Regression	Mean Square Error (MSE)
Regularized Regression (Ridge/LASSO)	SSE + Penalty Term
Decision Tree Regressor	Sum of Squares of Errors (SSE)
Support Vector Regressor (SVR)	L1-loss (same as Sum of absolute values of Errors) or L2-Loss (similar to SSE)

Name of Classifier Algorithm	Loss (Cost) Function minimized
Logistic Regression	Binary Cross Entropy (BCE), commonly called as negative-log-loss
Support Vector Classifier (SVC)	Hinge Loss
Decision Tree Classifier	Gini or Entropy or "log_loss" (you can choose)

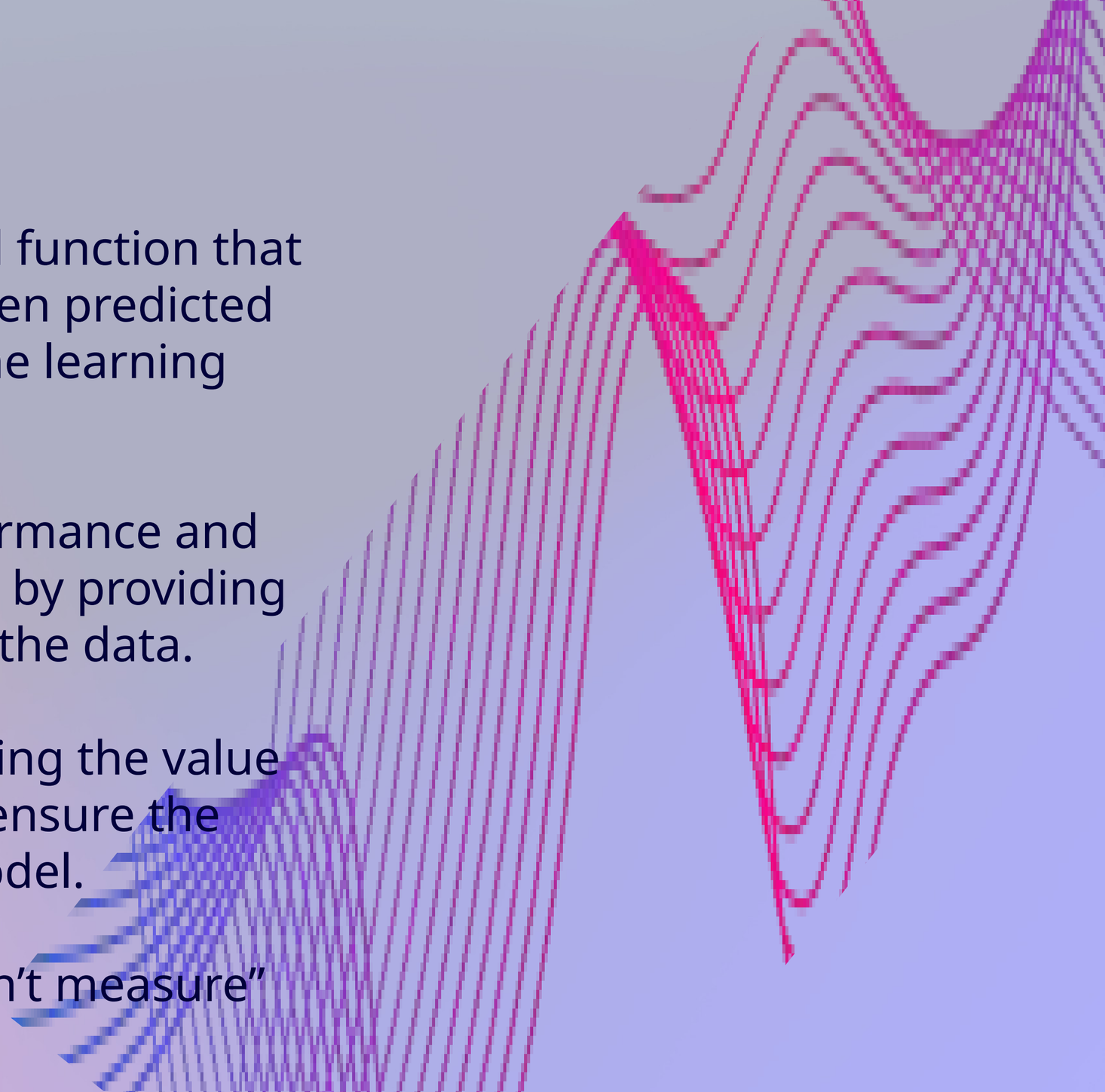
Loss Function

A loss function is a mathematical function that quantifies the difference between predicted and actual values in a machine learning model.

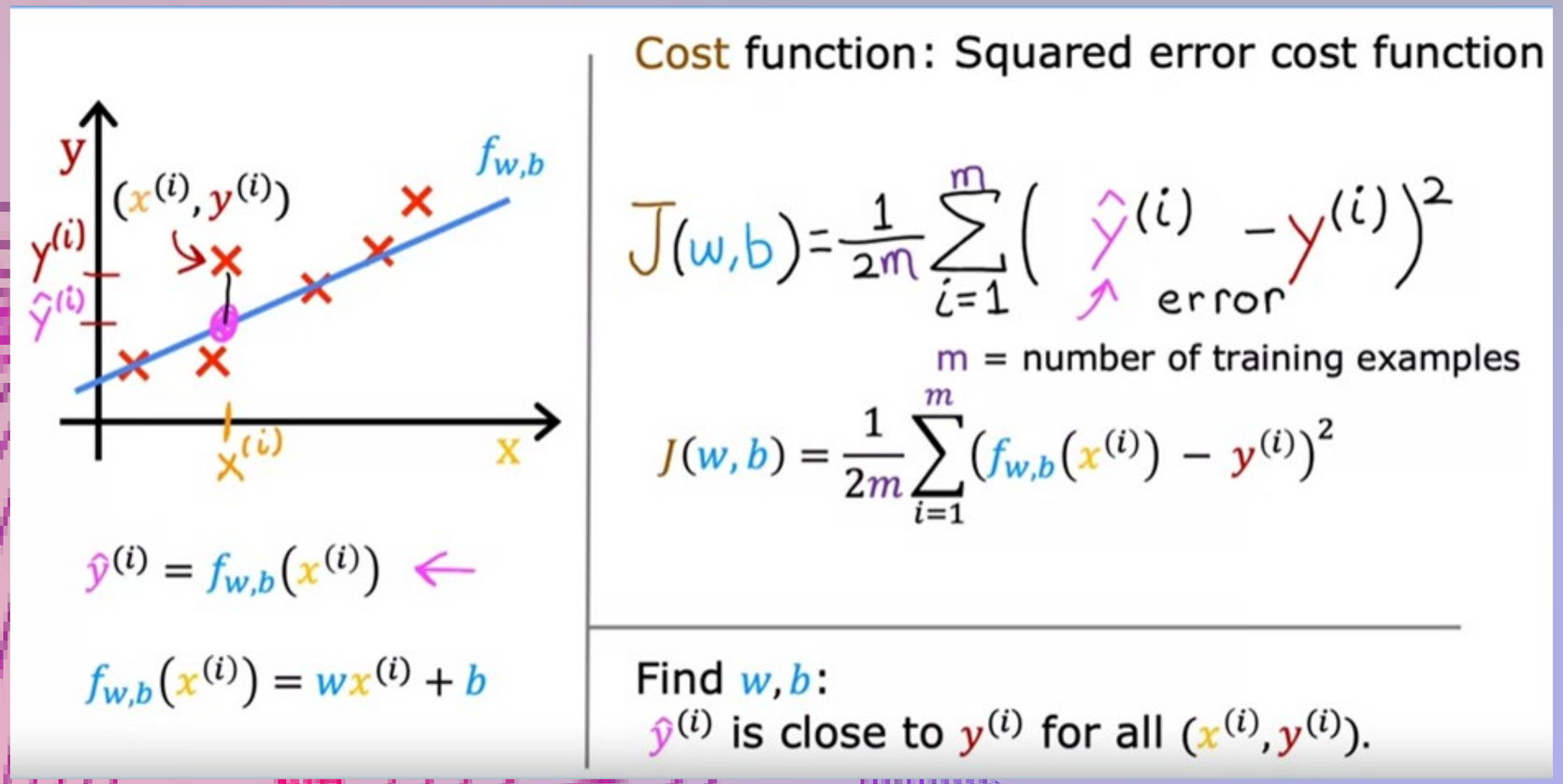
It measures the model’s performance and guides the optimization process by providing feedback on how well it fits the data.

Therefore, by repeatedly analyzing the value of this loss function, we can ensure the optimization of our model.

“You can’t improve what you can’t measure”
- Peter Druker



Cost Function for Linear Regression



Pros:

Intuitive and easy

Only one local minima

Cons:

Not robust to outlier

Cost function is not fixed

Mean Absolute Error

$$MAE = \frac{1}{N}\sum_{i=1}^{N} |Y_i - \hat{Y}_i|$$

Pros:

It is robust to outlier

Cons:

In this cost function, we cannot use gradient descent minimizing algorithm for directly

Binary Cross Entropy

$$\text{Log Loss} = -\frac{1}{N}\sum_{i=1}^{N} y_i \log \hat{y}_i + (1-y_i)\log(1-\hat{y}_i)$$

Pros:

cost function is a differential, therefore much hard to deal with

Cons:

Multiple local minima is observed, therefore making it difficult to use the gradient descent algorithm, It is also non intuitive

Gradient Descent Algorithm

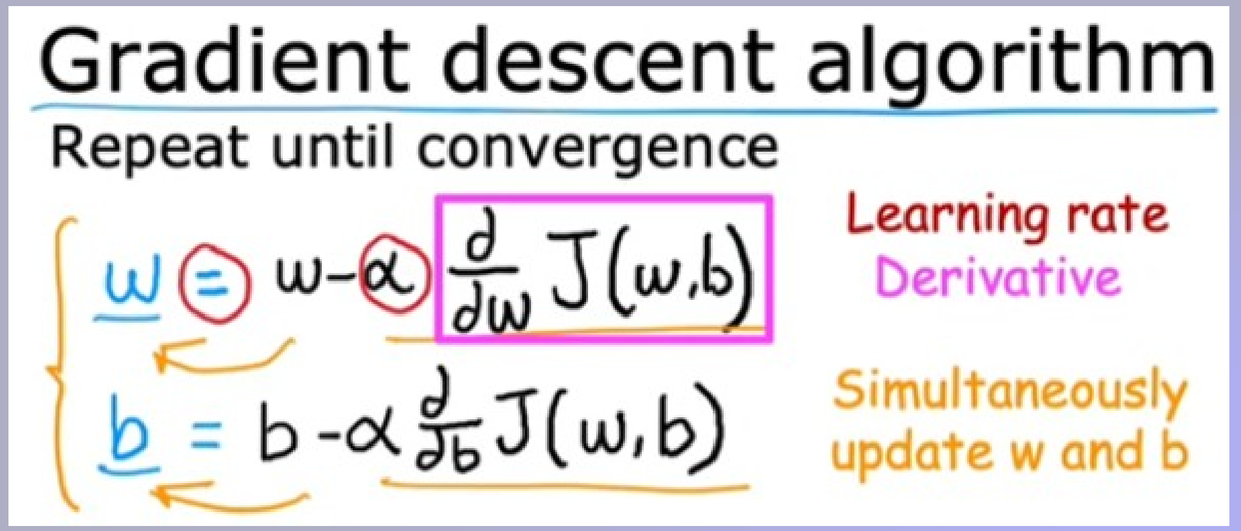
Gradient descent is an iterative optimization algorithm that aims to minimize the loss function by adjusting parameters based on the negative gradient.

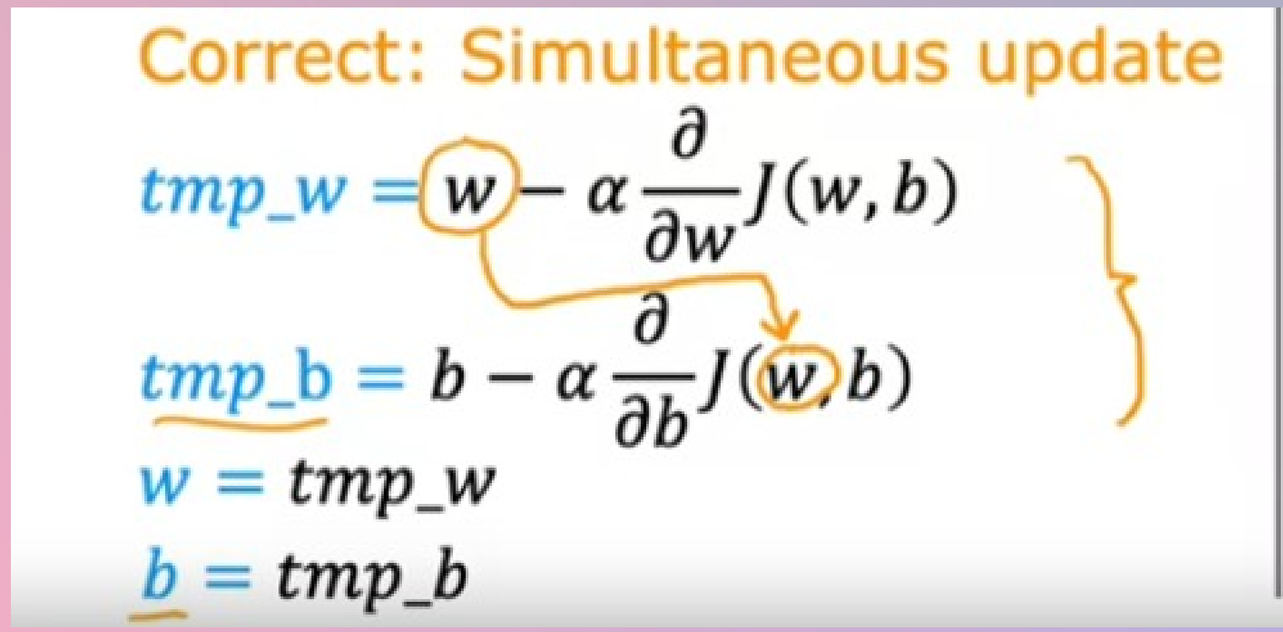
It provides a general framework for optimizing models by iteratively refining their parameters based on the cost function.

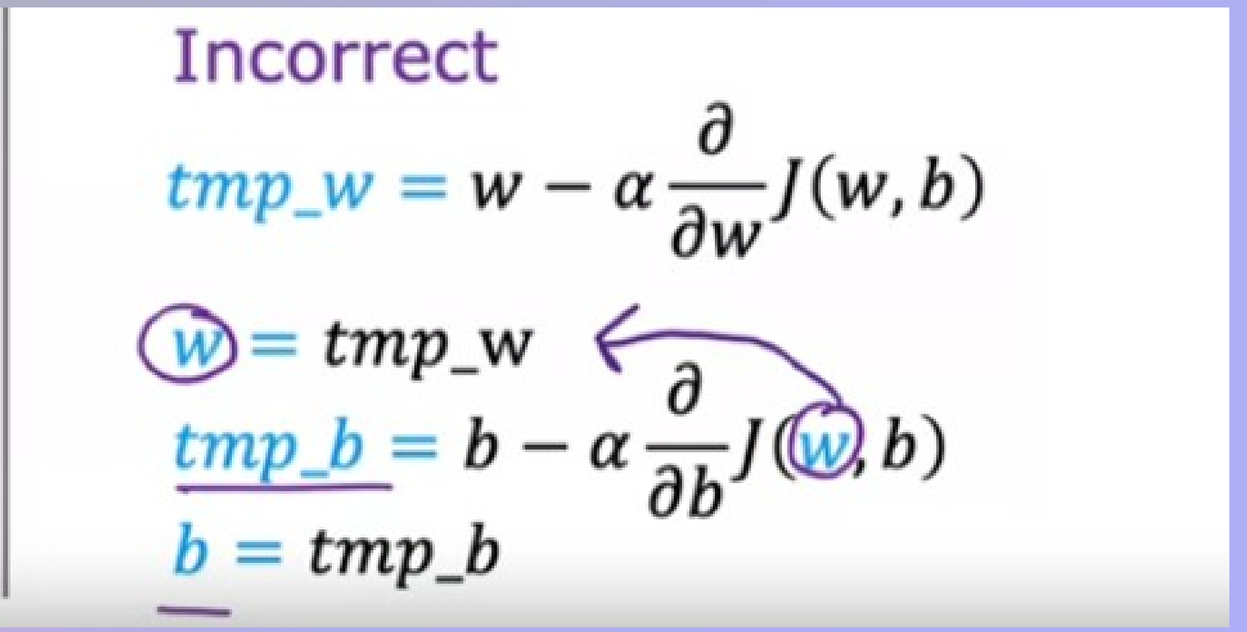
Model	$f_{w,b}(x) = wx + b$
Parameters	w, b ~~before: b=0~~
Cost Function	$J(w,b) = \frac{1}{2m}\sum_{i=1}^{m}(f_{w,b}(x^{(i)}) - y^{(i)})^2$
Objective	$\underset{w,b}{\text{minimize}}\, J(w,b)$

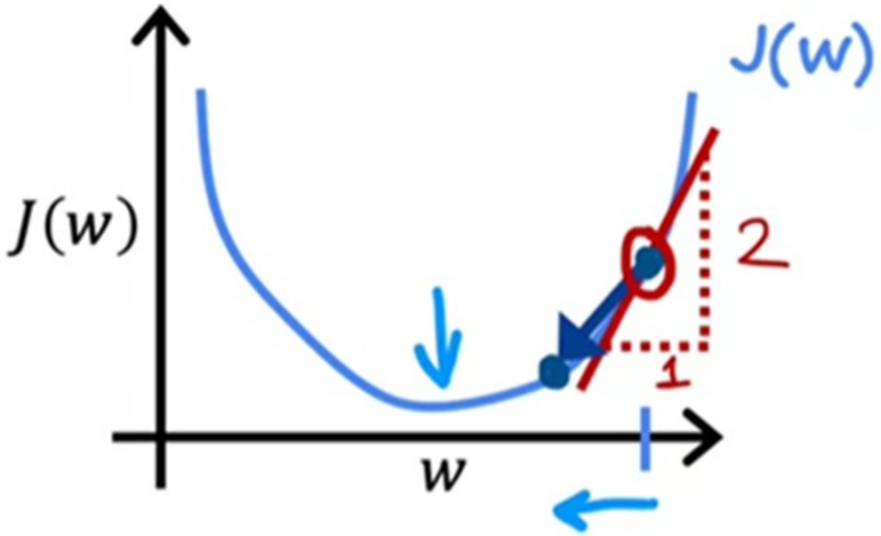
To achieve the goal of minimizing the loss function, it performs two steps iteratively:

- Compute the gradient (slope), at that point
- Take a step in the direction opposite to the gradient, opposite direction of increment of the slope from the current point by alpha times the gradient at that point



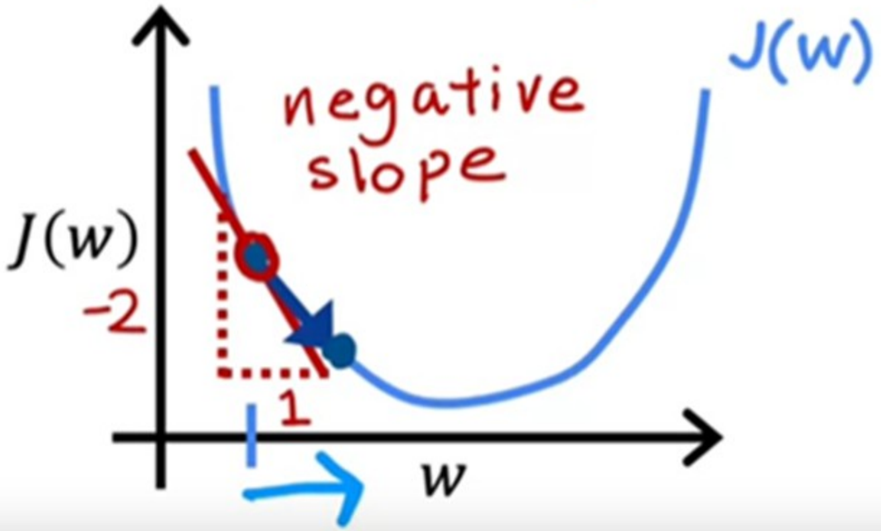






$$w = w - \alpha \underbrace{\frac{d}{dw} J(w)}_{>0}$$

$$w = w - \alpha \cdot (\text{positive number})$$



$$\underbrace{\frac{d}{dw} J(w)}_{<0}$$

$$w = w - \alpha \cdot (\text{negative number})$$

Types of Gradient Descent

Batch Gradient Descent:

Batch gradient descent updates the model’s parameters using the gradient of the entire training set.

Guarantees convergence to the global minimum, but can be computationally expensive and slow for large datasets.

Stochastic Gradient Descent:

Stochastic gradient descent updates the model’s parameters using the gradient of one training example at a time.

Stochastic gradient descent is computationally efficient and can converge faster than batch gradient descent

Mini- Batch Gradient Descent:

Updates the model’s parameters using the gradient of a small subset of the training set, known as a mini-batch.

It is computationally efficient and less noisy than stochastic gradient descent, while still being able to converge to a good solution.

Feature Scaling

Sometimes, in a model of multiple features, all the features may have vastly different scales due to which the convergence may takes a lot of time or it may not occur at all. Also, we may get biased results as the feature with larger scale may dominate the learning process. To prevent this, we use Feature Scaling.

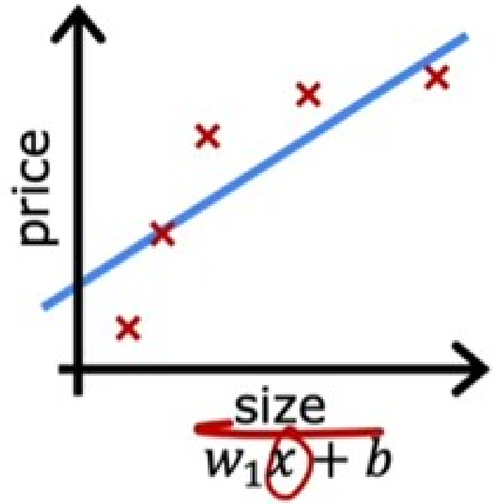
It is normalizing the range of all the features so that they lie in similar range.

- Max Normalization: Here, Range [a, b] becomes [a/b, 1]
- Mean Normalization: Range [a, b] becomes [(a - mu)/(b -a), (b - mu)/(b - a)]
- Z-Score Normalization: Range [a, b] --> [(a - mu)/sigma, (b - mu)/sigma]

(mu and sigma are the mean and standard deviation of a particular input

Need for Regularization?

Regression example

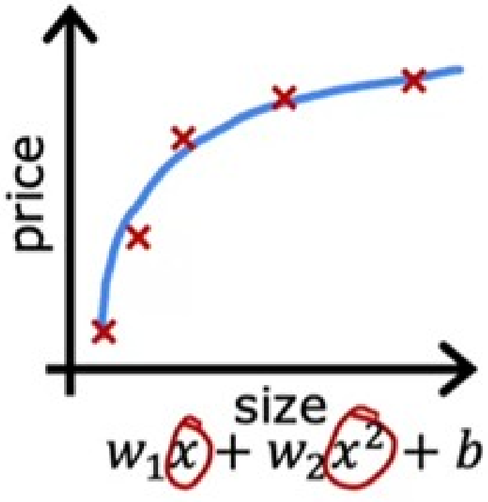


$w_1x + b$

underfit

- Does not fit the training set well

high bias

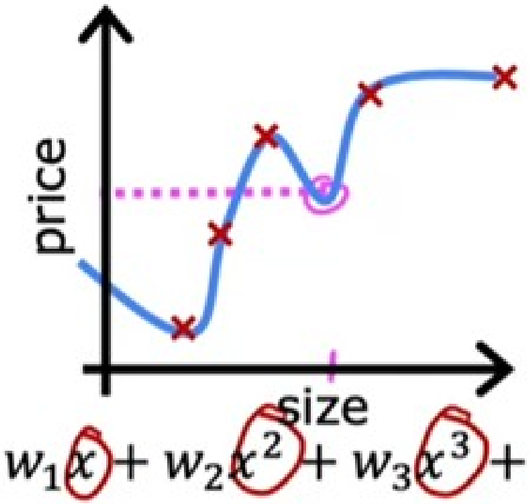


$w_1x + w_2x^2 + b$

just right

- Fits training set pretty well

generalization



$w_1x + w_2x^2 + w_3x^3 + w_4x^4 + b$

overfit

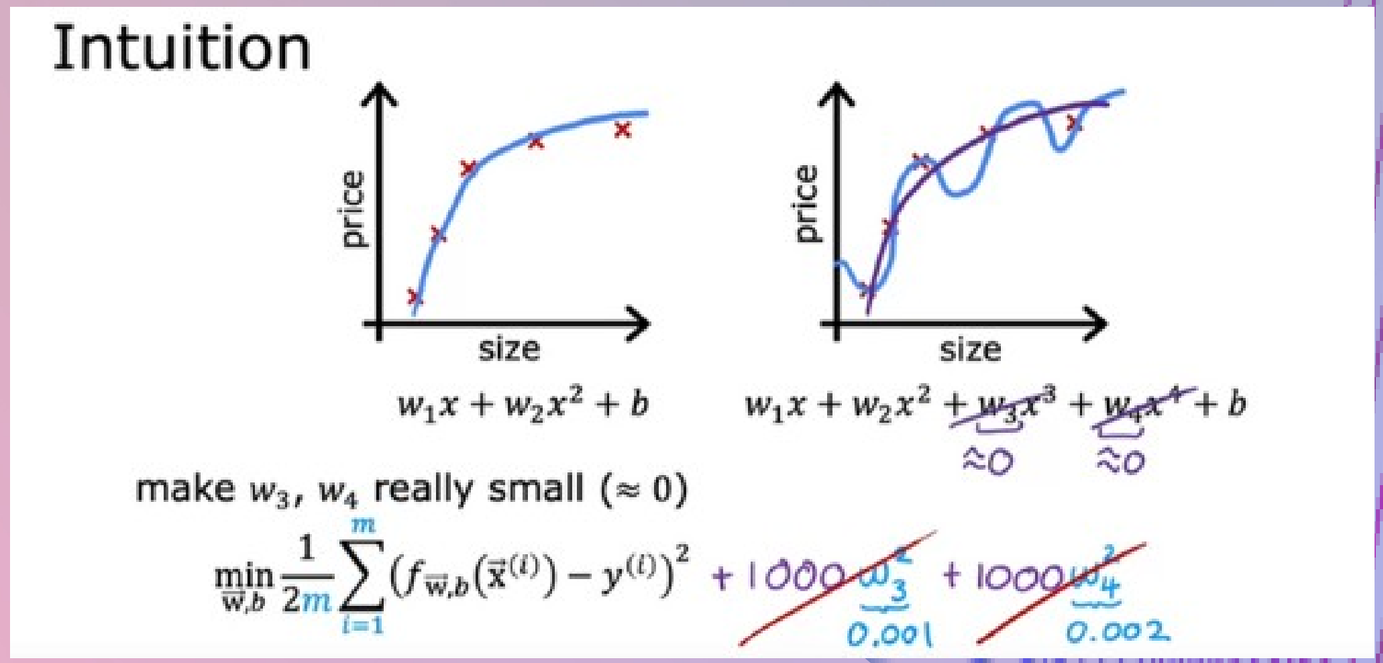
- Fits the training set extremely well

high variance

What is Regularization?

When training a machine learning model, the model can be easily overfitted or under fitted. To avoid this, we use regularization in machine learning to properly fit the model to our test set.

Regularization techniques help reduce the possibility of overfitting and help us obtain an optimal model.



How to perform Regularization?

Regularized linear regression

$$\min_{\vec{w},b} J(\vec{w},b) = \min_{\vec{w},b}\left[\frac{1}{2m}\sum_{i=1}^{m}\left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)^2 + \frac{\lambda}{2m}\sum_{j=1}^{n} w_j^2\right]$$

Gradient descent

repeat {

$$w_j = w_j - \alpha \frac{\partial}{\partial w_j} J(\vec{w},b) \qquad j = 1,\ldots,n$$

$$= \frac{1}{m}\sum_{i=1}^{m}\left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right) x_j^{(i)} + \frac{\lambda}{m} w_j$$

$$b = b - \alpha \frac{\partial}{\partial b} J(\vec{w},b)$$

$$= \frac{1}{m}\sum_{i=1}^{m}\left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)$$

don't have to regularize b

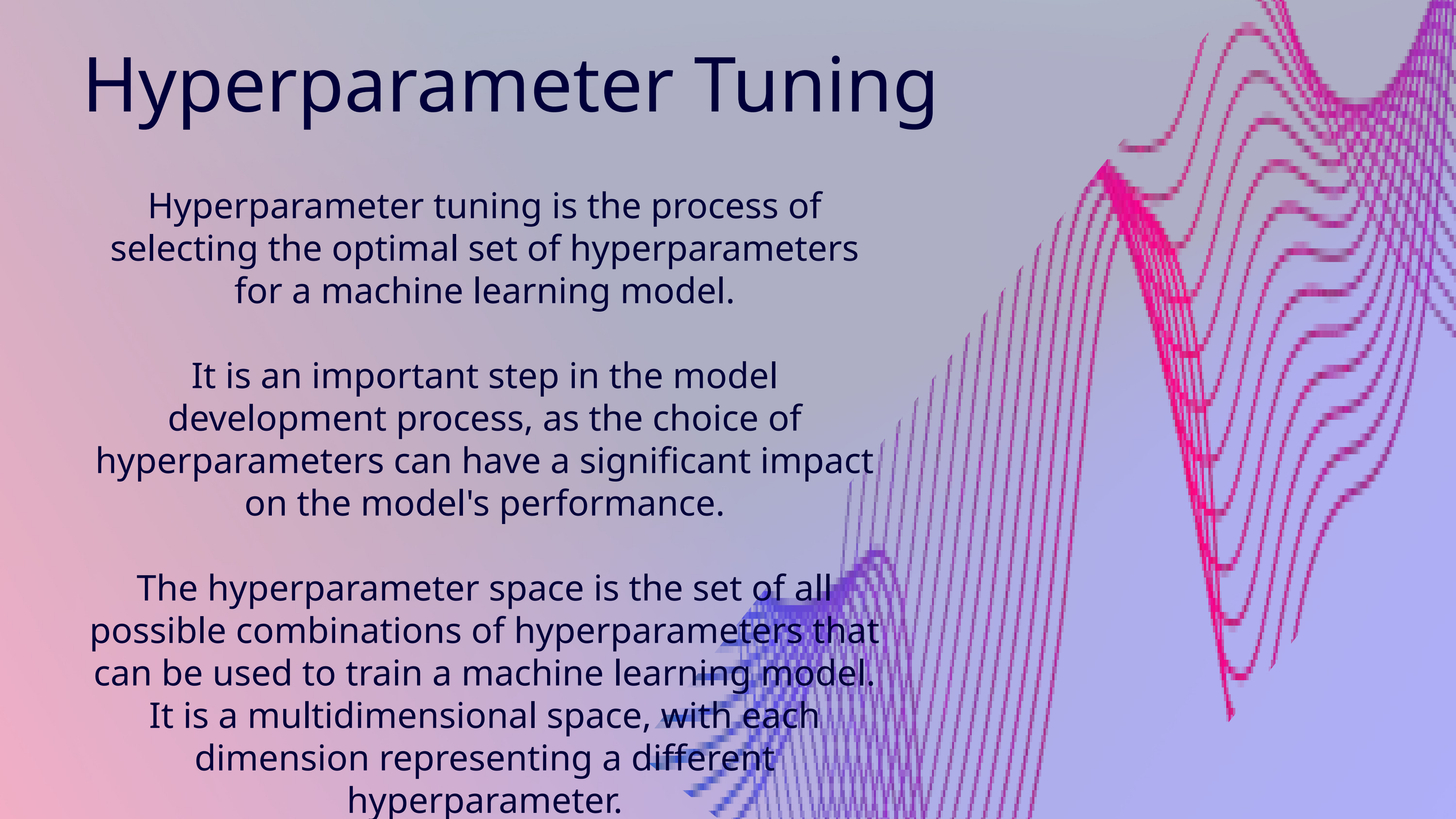
} simultaneous update

Hyperparameter Tuning

Hyperparameter tuning is the process of selecting the optimal set of hyperparameters for a machine learning model.

It is an important step in the model development process, as the choice of hyperparameters can have a significant impact on the model's performance.

The hyperparameter space is the set of all possible combinations of hyperparameters that can be used to train a machine learning model. It is a multidimensional space, with each dimension representing a different hyperparameter.



Methods of Hyperparameter Tuning:

- Grid Search: Grid search is a simple and systematic approach where a predefined set of hyperparameter values is specified, and the model is trained and evaluated for each combination of values.
- Random Search: Random search involves randomly sampling combinations of hyperparameter values from a predefined search space.
- Bayesian Optimization: Bayesian optimization models the unknown objective function (model performance) and uses a probabilistic surrogate to guide the search for optimal hyperparameters, balancing exploration and exploitation.
- Randomized Search Cross-Validation (RSCV): An extension of random search where each random combination of hyperparameters is evaluated using cross-validation.
- Genetic Algorithms: Genetic algorithms apply principles inspired by natural selection to evolve a population of hyperparameter sets over multiple generations, favoring those with better performance.

Optimization Challenges in ML:

- If Regularization is not done, then there is high chance that we may not get the desired results due to the problem of Overfitting and Underfitting.
- If Feature Scaling is not done, then Gradient Descent Algorithm may required more iterations than usual to minimize the cost function.
- Sometime, due to the bad selection of Cost function, we may get multiple local minima, which led to the convergence issues.
- Limited or poor-quality data can pose challenges for optimization.
- Choosing the right set of hyperparameters, such as learning rates, regularization strengths, or the number of layers in a neural network, is challenging and can significantly impact model performance.
- Training complex models, especially deep neural networks, can be computationally intensive and time-consuming, particularly when dealing with large datasets.

Applications of optimization in ML:

- Hyperparameter Tuning: Used to find the optimal values for hyperparameters, such as learning rates.
- Feature Selection and Extraction: Helps to identify the most relevant features for a model.
- Ensemble Learning: Combining multiple models into an ensemble can improve overall performance.
- Anomaly Detection: Optimization is used to define the normal behavior and detect anomalies by minimizing a predefined anomaly score or loss function.
- Recommender Systems: Optimization is employed in collaborative filtering and content-based recommendation systems to minimize the prediction error or loss function.
- Predicting Future Behavior: ML Models are used to predict the future outcomes, or for analyzing the current performance of lets say stock market, movie rating, etc.

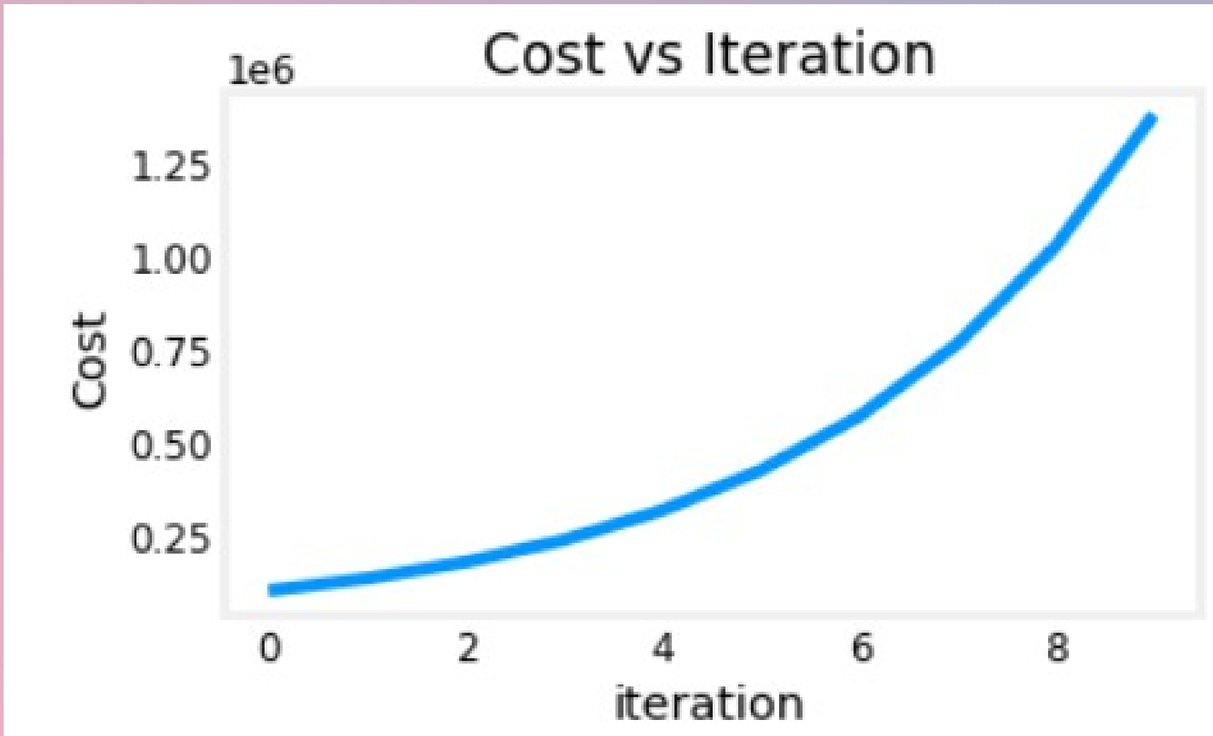
Case Study

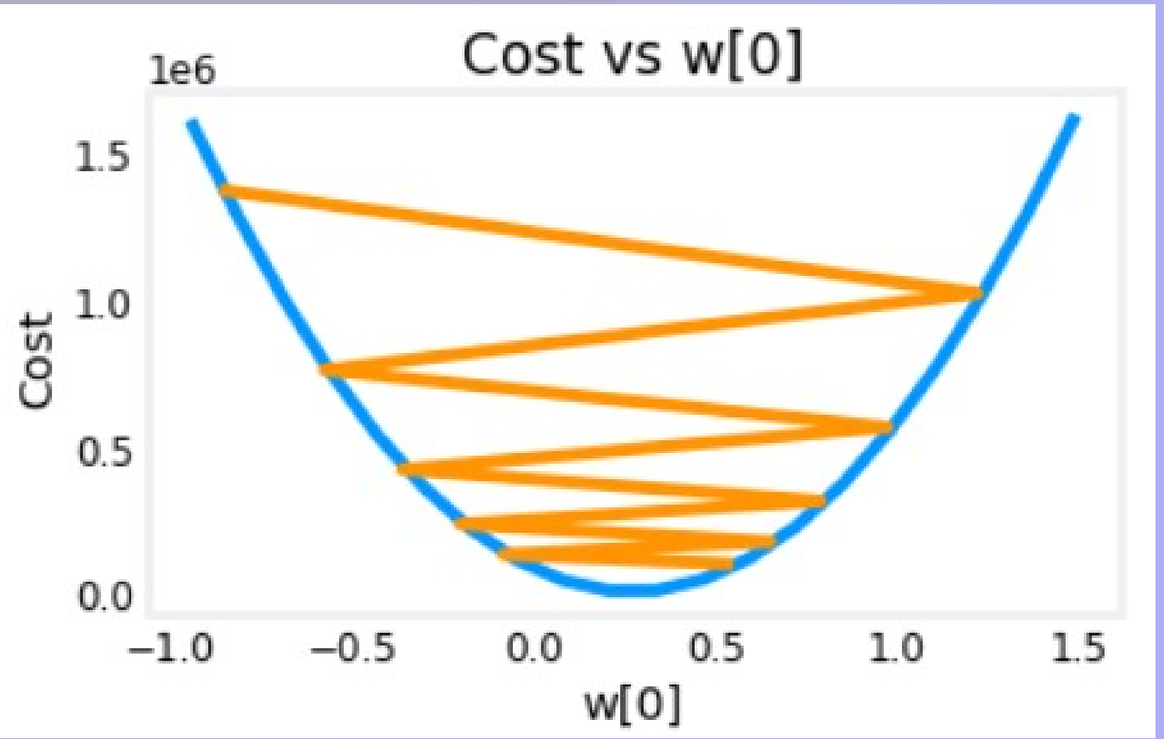
Optimizing Hyperparameter Tuning for Image Classification with Convolutional Neural Networks (CNNs):

This case study demonstrates the importance of hyperparameter tuning in improving the accuracy of CNN models for image classification. The chosen optimization methods and identified hyperparameters provide insights into efficient model training, balancing computational resources and performance gains. The results contribute to the development of a more accurate and robust image classification system for the company's inventory management needs.

Why Hyperparameter Tuning is must?

Say for example, lets talk about Alpha (Learning Rate of Gradient Descent): Choosing correct value of Alpha is important to avoid extra time and computations as well as to avoid the infinite loops and wrong answers.





References

- https://www.seldon.io/machine-learning-optimisation#:~:text=Machine%20learning%20optimisation%20is%20the%20process%20of%20iteratively%20improving%20the,insight%20learned%20from%20training%20data.
- https://towardsdatascience.com/gradient-descent-algorithm-a-deep-dive-cf04e8115f21#:~:text=Gradient%20descent%20(GD)%20is%20an,e.g.%20in%20a%20linear%20regression).
- https://www.analyticsvidhya.com/blog/2020/04/feature-scaling-machine-learning-normalization-standardization/
- https://www.geeksforgeeks.org/ml-feature-scaling-part-2/
- https://scikit-learn.org/stable/auto_examples/preprocessing/plot_scaling_importance.html

THANK YOU!!

Group Members

- 07. Parth Zarekar
- 10. Krishnaraj Thadesar
- 24. Singh Soubhagya
- 25. Sourab Karad

NEURAL NETWORKS

Method in artificial Intelligence that teaches computers to process data in a way that is inspired by human brain.

COMPONENTS OF NEURAL NETWORK

Input Layer

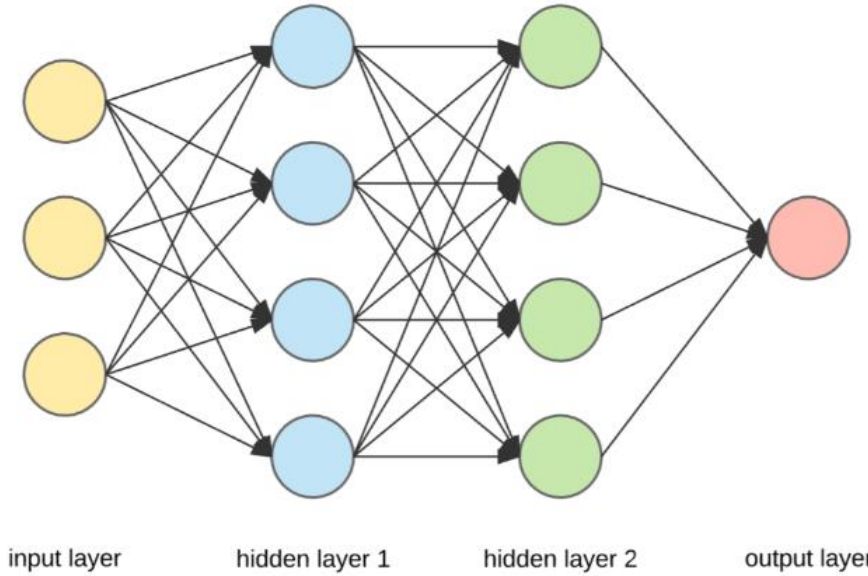
Imagine this as eyes and ears of the network.

Hidden Layers

Hidden layers are where most of the thinking happens

Output Layer

The output Layer produces the final results of the networks calculations



Neurons (Nodes)

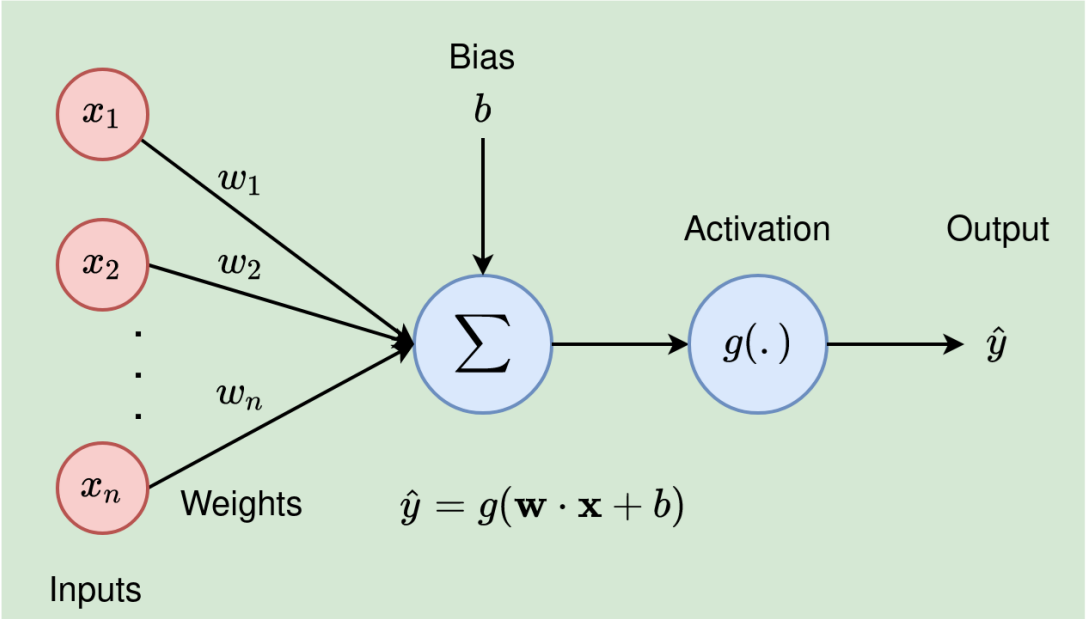
Neurons information from input layer or other neuron and make calculations.

Weights

Neurons assign importance to the information they receive through weights

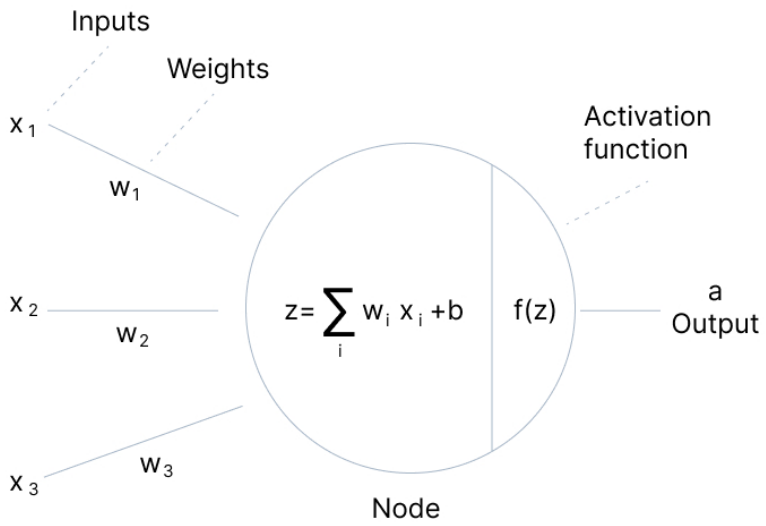
Bias

Biases act like a starting point for each neuron's calculation.



ACTIVATION FUNCTION

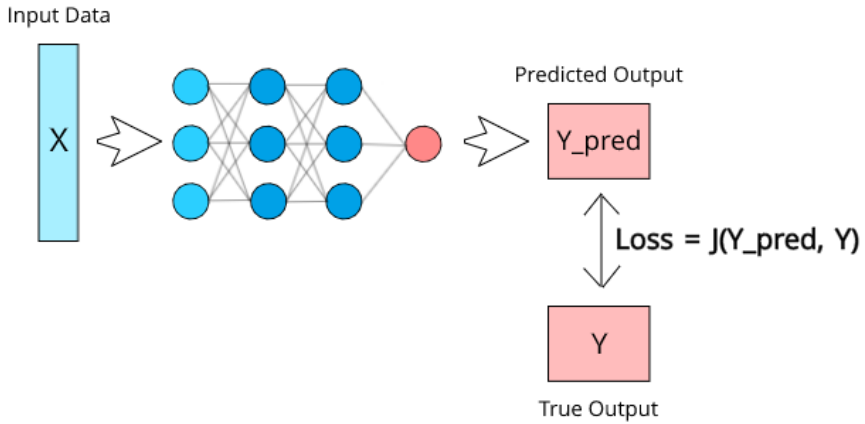
Decides when a neuron should "fire" or activate.

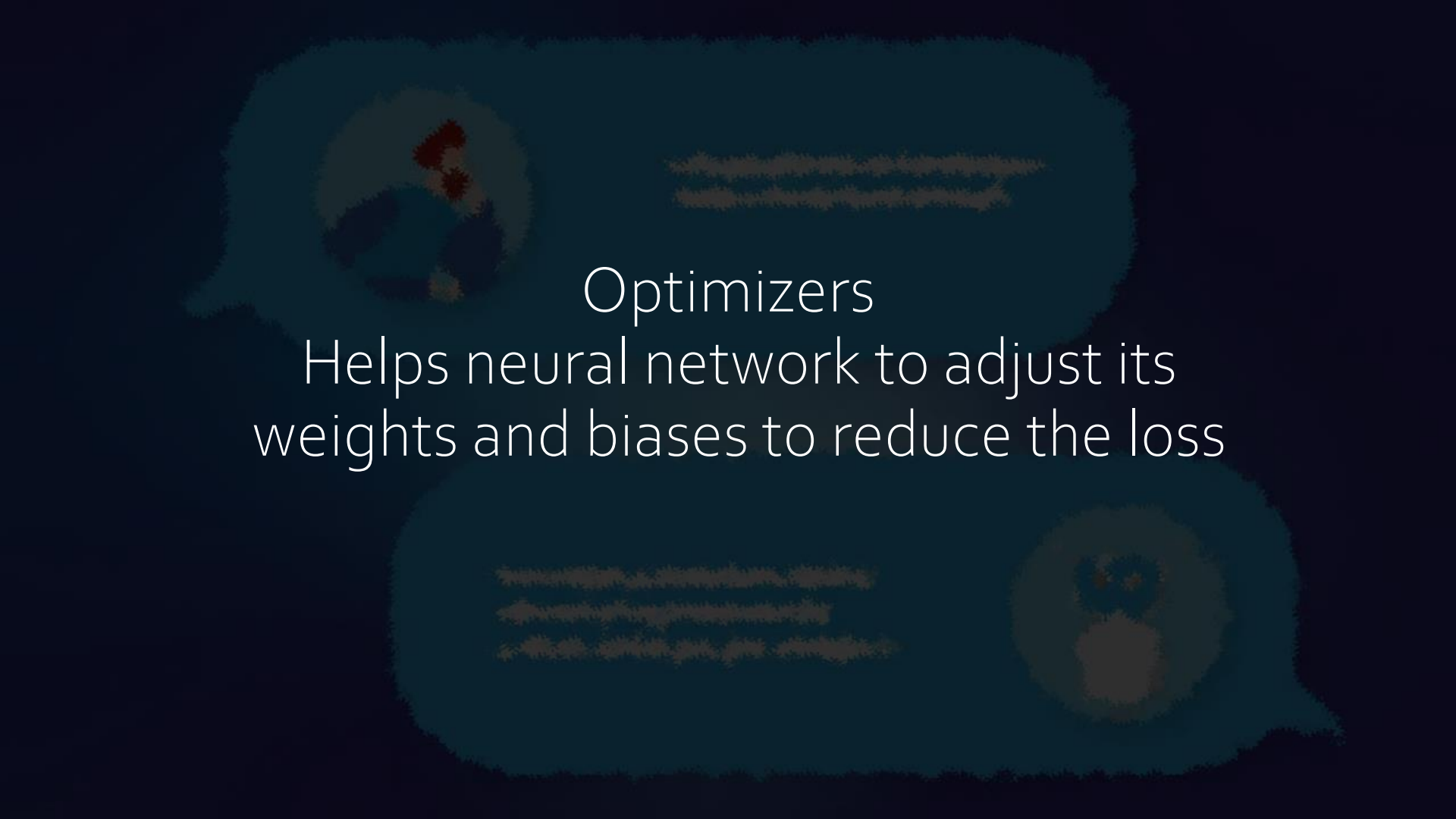


V7 Labs

LOSS FUNCTION

It calculates how far networks calculated output is from actual desired output.




Optimizers
Helps neural network to adjust its weights and biases to reduce the loss

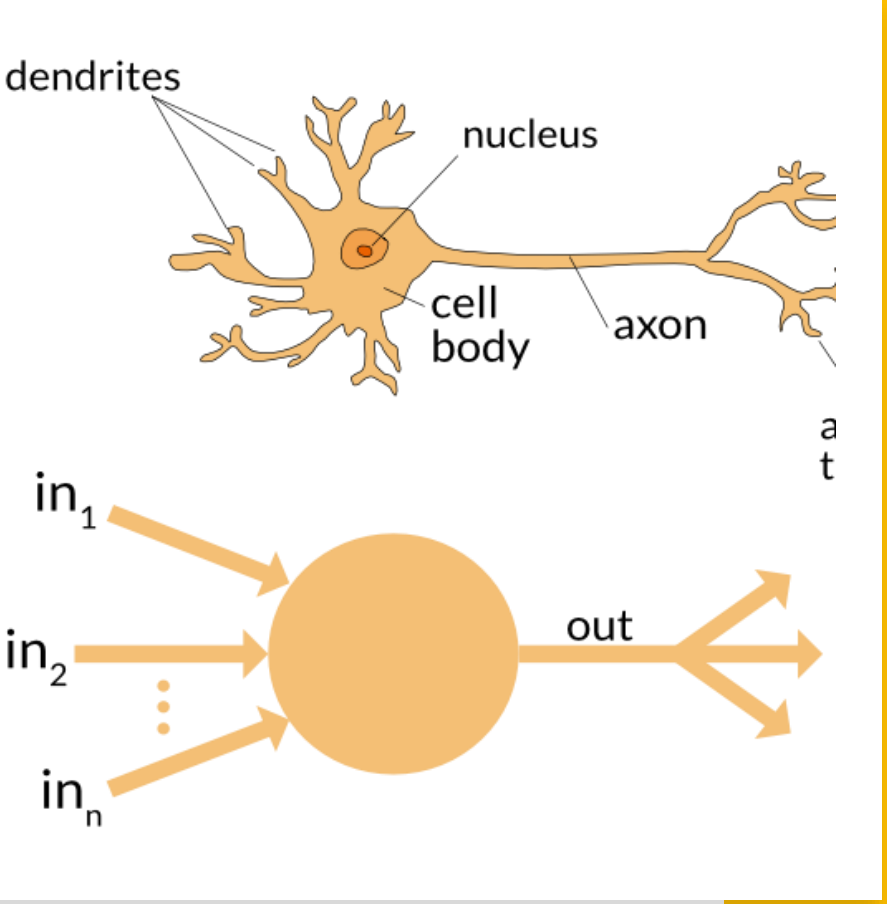

BACK PROPAGATION
It tells network how it should adjust its internal settings to perform better in future

WHAT IS A PERCEPTRON

A perceptron is a fundamental building block in artificial neural networks and serves as a simple model of a biological neuron. It's a binary linear classifier that can be used for various machine learning tasks, such as binary classification.

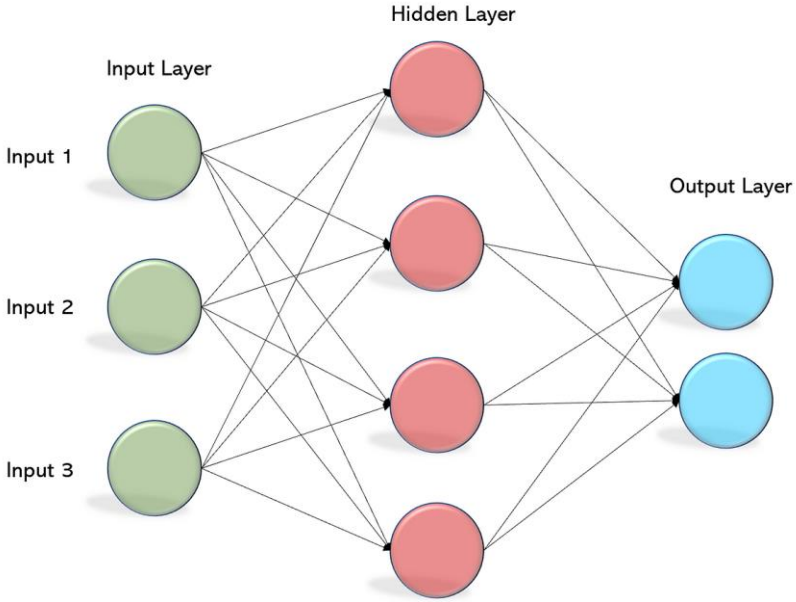
COMPARISON TO A NEURON

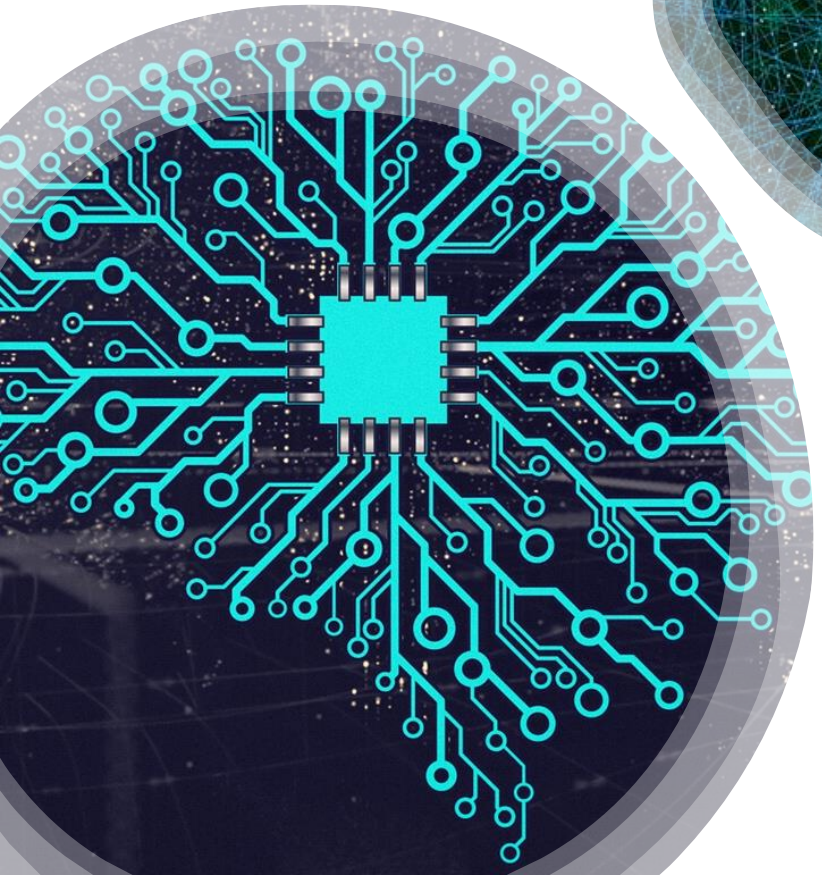
It is a basic unit of an artificial neural network that takes a set of input values, processes them, and produces an output.

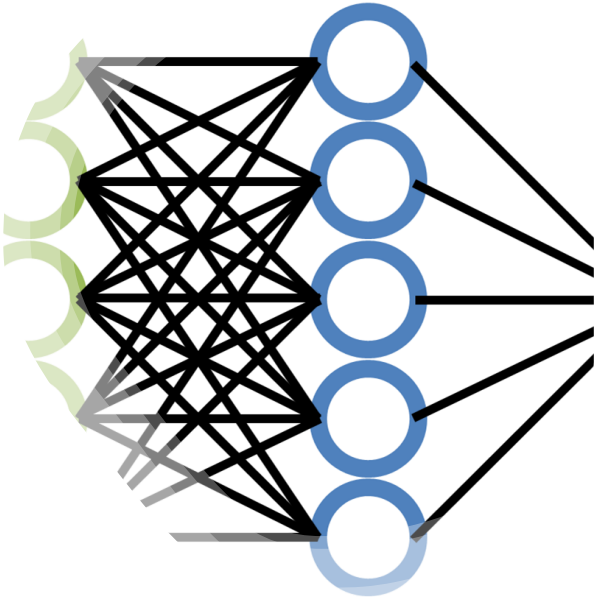


MULTILAYER PERCEPTRON

- Multilayer Perceptrons (MLPs), also known as feedforward neural networks or artificial neural networks, are a type of artificial neural network with multiple layers of interconnected neurons.
- These networks consist of an input layer, one or more hidden layers, and an output layer. Multilayer Perceptrons are designed to handle more complex tasks than single-layer perceptrons, as they can capture non-linear patterns in data.







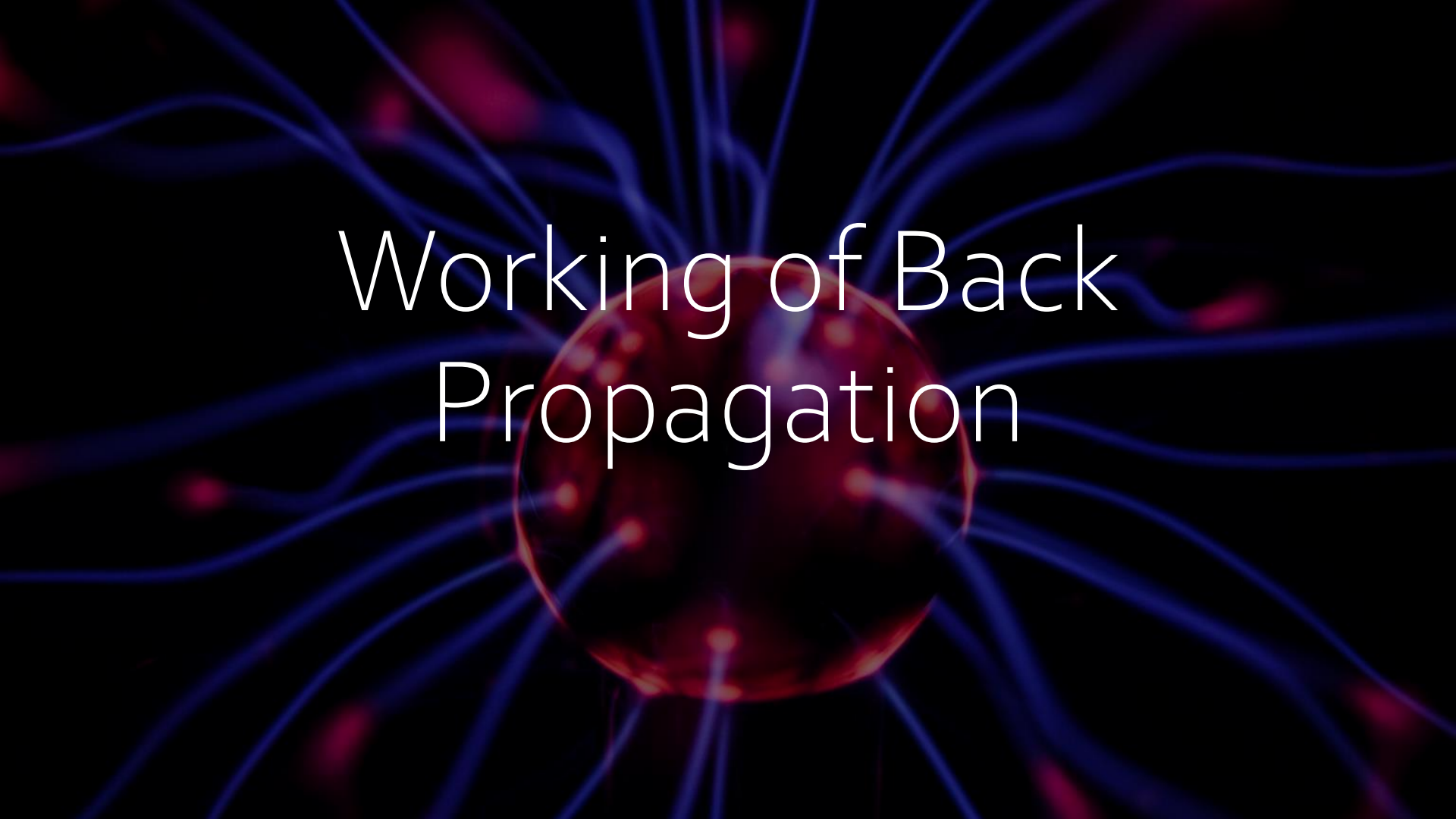
Back Propagation: Unveiling the Inner Workings of Neural Networks

24. Saubhagya Singh

UNDERSTANDING BACK PROPAGATION

Back propagation is a key algorithm for training neural networks. It enables the network to learn from labeled training data and adjust its internal parameters to minimize the error between predicted and actual outputs.

The algorithm works by propagating the error backwards through the network, updating the weights and biases of each neuron based on their contribution to the overall error. This iterative process continues until the network achieves satisfactory performance.


Working of Back
Propagation

Definition of Backpropagation

Backpropagation is a crucial algorithm in artificial neural networks, especially in training deep learning models.

Purpose of Backpropagation

It optimizes neural network weights to minimize the error between predictions and actual target values.

Forward Pass

Input data traverse the network, passing through neurons for weighted summations and activation functions.

Loss Computation

The algorithm quantifies the difference between predictions and ground truth through loss calculation.

Error Propagation

Backpropagation propagates error from output to input layer, computing gradients of loss with respect to each neuron's weights.

Guidance for Weight Adjustments

These gradients serve as guidance for adjusting neuron weights to minimize the loss function.

Gradient Descent Optimization

Weight adjustments are made using gradient descent algorithms to iteratively minimize the loss.

Iterative Process

This process continues until the model's predictions closely align with the target values, enabling accurate predictions.

IMPORTANCE OF BACK PROPAGATION

1. Neural Network Training: Backpropagation is the primary technique for training neural networks, enabling them to make accurate predictions and solve complex tasks.

2. Universal Function Approximation: It allows neural networks to approximate a wide range of complex functions, making them versatile tools for various applications.

3. Feature Learning: Neural networks using backpropagation automatically learn and extract relevant features from raw data, reducing the need for manual feature engineering.

4. Generalization: Backpropagation helps neural networks generalize from training data to make accurate predictions on unseen data, a fundamental aspect of machine learning models.

5. State-of-the-Art Performance: Deep learning models trained with backpropagation have achieved state-of-the-art results in diverse domains, driving research and innovation in machine learning.

ARCHITECTURE OF BACKPROPAGATION

Forward Pass: Input data is passed forward through the neural network layers to generate a prediction.

Backward Pass: Error calculation is performed by comparing the predicted output to the actual output, and this error is propagated backward through the network to adjust weights using the gradient descent algorithm

Gradient Descent

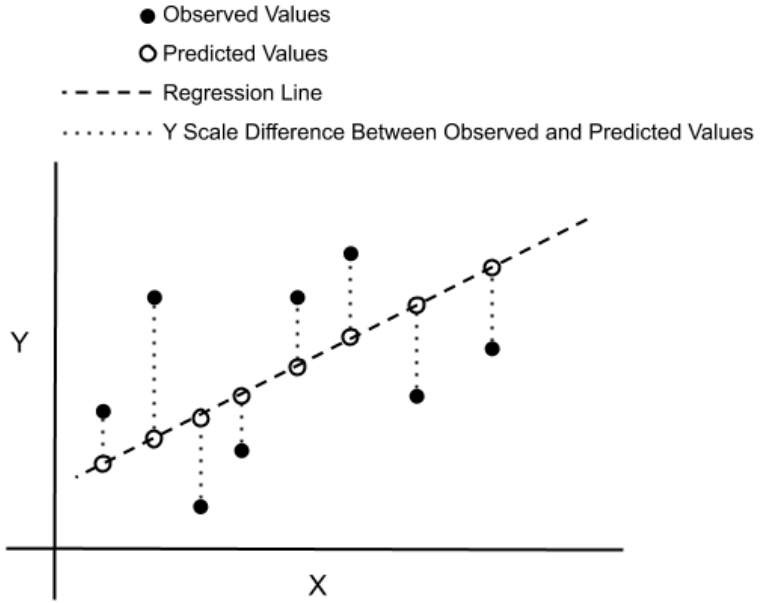
Optimization technique used in backpropagation to minimize the error by adjusting weights. It calculates the gradient of the cost function with respect to each weight.

Activation Functions

Functions used at each neuron to introduce non-linearity. Common activation functions include ReLU, sigmoid, and tanh.

Backpropagation Key Components

Error calculation, chain rule application, weight updates using calculated gradients, learning rate adjustment.



Cost Functions

Mean Error Squared

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

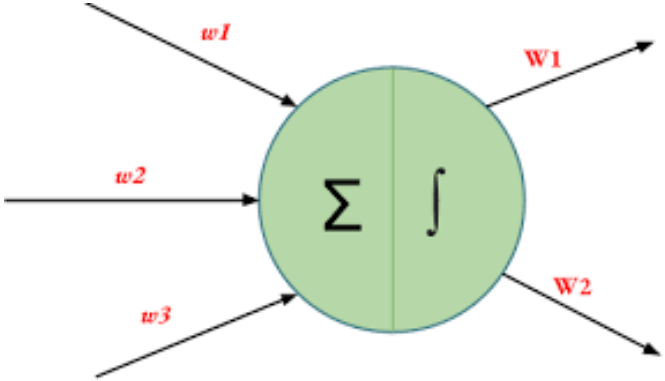
- Mean Squared Error (MSE): Measures the average of the squared differences between predictions and actual values.
- Cross-Entropy Loss: Commonly used in classification problems. It measures the performance of a classification model whose output is a probability value between 0 and 1

ROLE OF COST FUNCTIONS

Cost functions help quantify the error between predicted and actual values. The goal is to minimize this error during training by adjusting network weights.

WEIGHT INITIALIZATION

1. Random Initialization: Weights are initialized randomly to break symmetry among neurons.
2. Random Normal: The weights are initialized from values in a normal distribution.
3. Random Uniform: The weights are initialized from values in a uniform distribution.



$$w_i \sim U[-\sqrt{\frac{\sigma}{fan\_in+fan\_out}}, \sqrt{\frac{\sigma}{fan\_in+fan\_out}}]$$

XAVIER GLOROT INITIALIZATION

- Sets initial weights based on the number of input and output neurons of the layer.
- Xavier/Glorot Initialization often termed as Xavier Uniform Initialization, is suitable for layers where the activation function used is Sigmoid.
- In Xavier/Glorot weight initialization, the weights are assigned from values of a uniform distribution as follows:

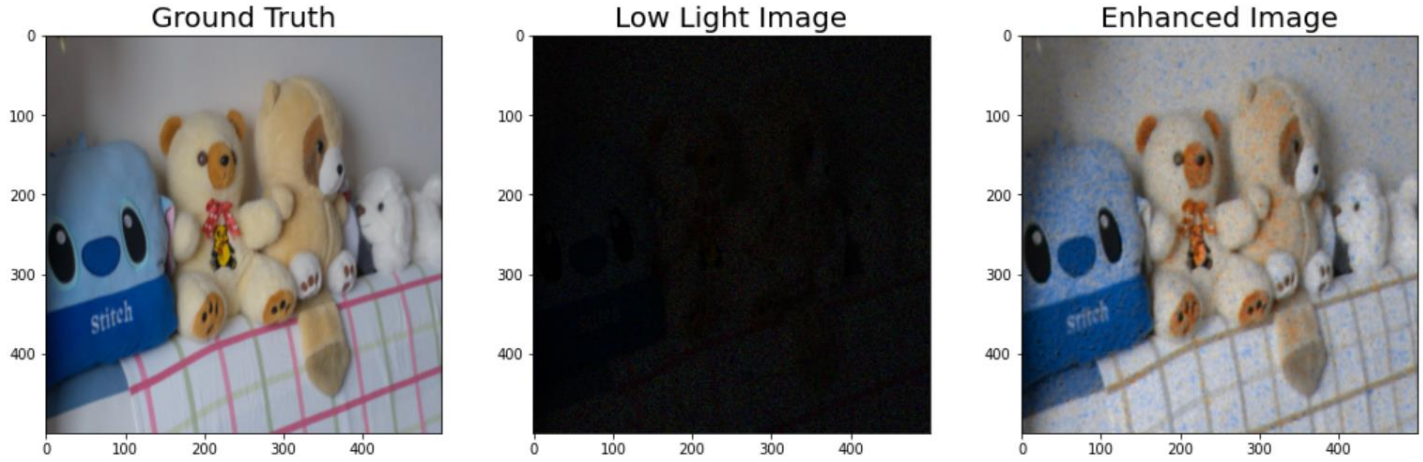
$$w_i \sim U[-\sqrt{\frac{6}{fan\_in}}, \sqrt{\frac{6}{fan\_out}}]$$

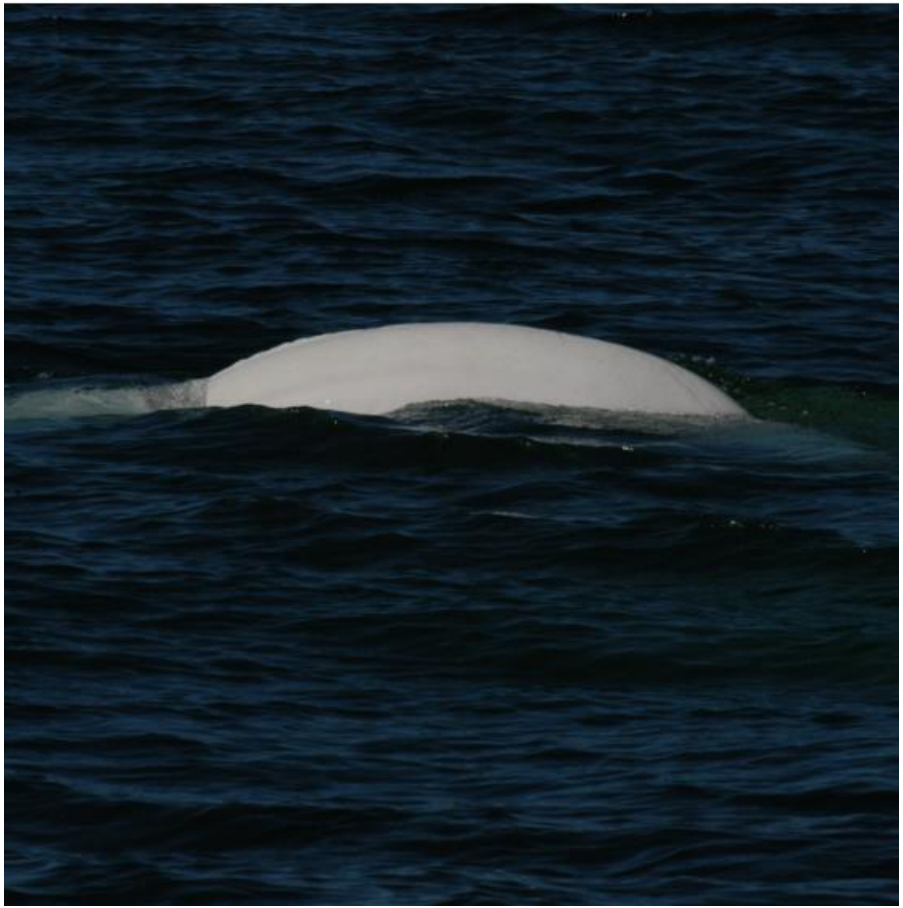
HE INITIALIZATION

- Similar to Xavier, but taking into account the ReLU (rectified linear) activation function.
- In the Uniform weight initialization, the weights are assigned from values of a uniform distribution as follo
- He Uniform Initialization is suitable for layers where ReLU activation function is used

Importance of Weight Initialization

Proper weight initialization helps in convergence speed, prevents vanishing/exploding gradients, and aids in better training of neural networks.


Ground Truth
Low Light Image
Enhanced Image
0
100
200
300
400
stitch



Package Importation

```python
import numpy as np
import pandas as pd
import os
import cv2 as cv
import matplotlib.pyplot as plt
from keras import backend as K
from keras.layers import add, Conv2D,MaxPooling2D,UpSampling2D,Input,BatchNormalization, RepeatVector, Reshape
from keras.models import Model
np.random.seed(1)
```

In [6]:

```python
model = from_pretrained_keras("keras-io/lowlight-enhance-mirnet", compile=False)
```

```
config.json not found in HuggingFace Hub

Downloading: 100% 1.25k/1.25k [00:00<00:00, 50.9kB/s]
```

In [12]:
```
image = keras.preprocessing.image.img_to_array(low_light_img)
```

In [13]:
```
image.shape
```

Out[13]:
```
(256, 256, 3)
```

In [14]:
```
image = image.astype('float32') / 255.0
```

In [15]:
```
image.shape
```

Out[15]:
```
(256, 256, 3)
```

In [18]:
```
output = model.predict(image) # model inference to enhance the low light pics
```

```
2022-04-22 18:23:00.545896: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:185] None
of the MLIR Optimization Passes are enabled (registered 2)
```

In [19]:
```
output_image = output[0] * 255.0
```

In [20]:
```
output_image.shape
```

Out[20]:
```
(256, 256, 3)
```

In [21]:
```
output_image = output_image.clip(0,255)
```

In [25]:

```
output_image
```

Out[25]:

```
array([[[ 75.927765,  90.44433 , 100.36248 ],
        [ 98.26136 , 116.37938 , 143.26581 ],
        [112.696045, 151.7278  , 177.84164 ],
        ...,
        [ 89.94365 , 114.33969 , 135.48743 ],
        [ 91.46919 , 107.76184 , 123.99914 ],
        [ 84.028366, 100.88614 , 123.06505 ]],

       [[ 96.2401  ,  87.456924, 106.73138 ],
        [ 96.12944 ,  95.80934 , 124.13488 ],
        [ 73.13187 , 101.668564, 136.89339 ],
        ...,
        [ 83.40862 , 126.74953 , 133.8411  ],
        [ 83.54708 , 122.20536 , 133.4005  ],
```

Final Image

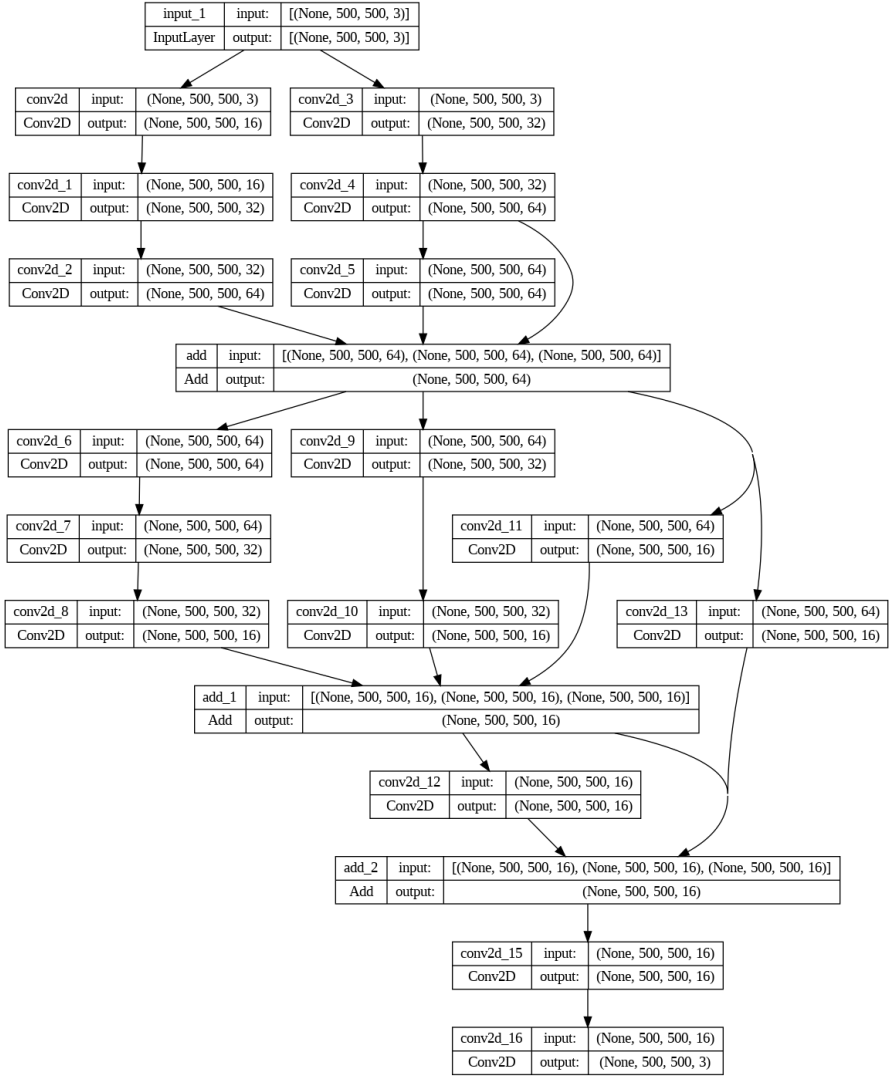
In [28]:
```
Image.fromarray(output_image.astype('uint8'),'RGB')
```

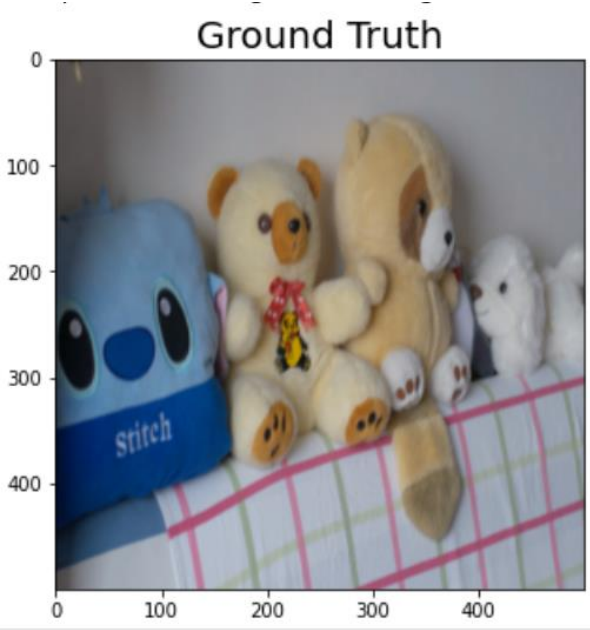
Out[28]:

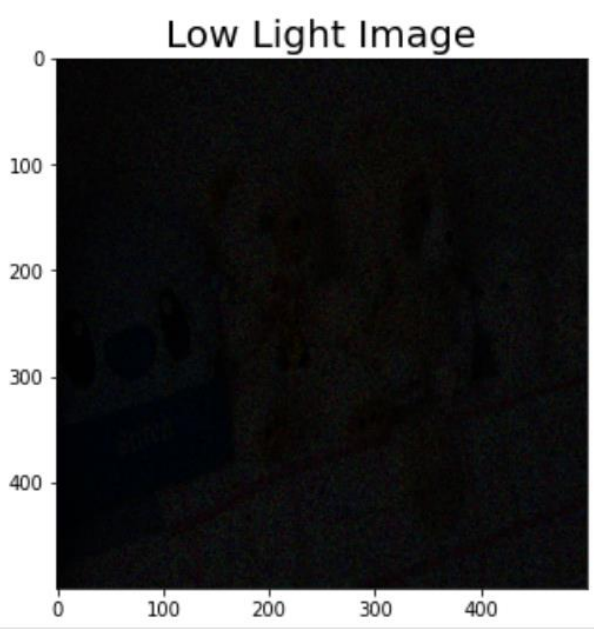
```python
def PreProcessData(ImagePath):
    X_=[]
    y_=[]
    count=0
    for imageName in tqdm(os.listdir(HighPath)):
        count=count+1
        low_img = cv.imread(HighPath + "/" + imageName)
        low_img = cv.cvtColor(low_img, cv.COLOR_BGR2RGB)
        low_img = cv.resize(low_img,(500,500))
        hsv = cv.cvtColor(low_img, cv.COLOR_BGR2HSV) #convert it to hsv
        hsv[...,2] = hsv[...,2]*0.2
        img_1 = cv.cvtColor(hsv, cv.COLOR_HSV2BGR)
        Noisey_img = addNoise(img_1)
        X_.append(Noisey_img)
        y_.append(low_img)
    X_ = np.array(X_)
    y_ = np.array(y_)
```

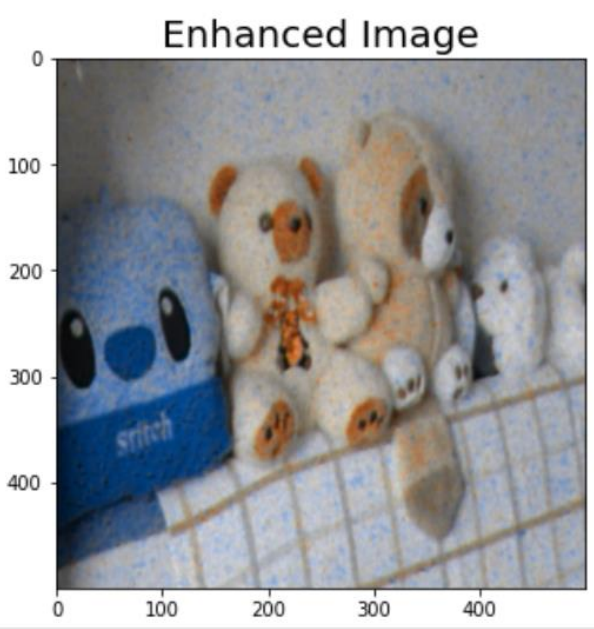
```
K.clear_session()
def InstantiateModel(in_):

    model_1 = Conv2D(16,(3,3), activation='relu',padding='same',strides=1)(in_)
    model_1 = Conv2D(32,(3,3), activation='relu',padding='same',strides=1)(model_1)
    model_1 = Conv2D(64,(2,2), activation='relu',padding='same',strides=1)(model_1)

    model_2 = Conv2D(32,(3,3), activation='relu',padding='same',strides=1)(in_)
    model_2 = Conv2D(64,(2,2), activation='relu',padding='same',strides=1)(model_2)

    model_2_0 = Conv2D(64,(2,2), activation='relu',padding='same',strides=1)(model_2)

    model_add = add([model_1,model_2,model_2_0])
```

```
1  Input_Sample = Input(shape=(500, 500,3))
2  Output_ = InstantiateModel(Input_Sample)
3  Model_Enhancer = Model(inputs=Input_Sample, outputs=Output_)
4  Model_Enhancer.compile(optimizer="adam", loss='mean_squared_error')
5  Model_Enhancer.summary()
```


input_1 | input: | [(None, 500, 500, 3)]
InputLayer | output: | [(None, 500, 500, 3)]
conv2d | input: | (None, 500, 500, 3)
Conv2D | output: | (None, 500, 500, 16)
conv2d_3 | input: | (None, 500, 500, 3)
Conv2D | output: | (None, 500, 500, 32)
conv2d_1 | input: | (None, 500, 500, 16)
Conv2D | output: | (None, 500, 500, 32)
conv2d_4 | input: | (None, 500, 500, 32)
Conv2D | output: | (None, 500, 500, 64)
conv2d_2 | input: | (None, 500, 500, 32)
Conv2D | output: | (None, 500, 500, 64)
conv2d_5 | input: | (None, 500, 500, 64)
Conv2D | output: | (None, 500, 500, 64)
add | input: | [(None, 500, 500, 64), (None, 500, 500, 64), (None, 500, 500, 64)]
Add | output: | (None, 500, 500, 64)
conv2d_6 | input: | (None, 500, 500, 64)
Conv2D | output: | (None, 500, 500, 64)
conv2d_9 | input: | (None, 500, 500, 64)
Conv2D | output: | (None, 500, 500, 32)
conv2d_7 | input: | (None, 500, 500, 64)
Conv2D | output: | (None, 500, 500, 32)
conv2d_11 | input: | (None, 500, 500, 64)
Conv2D | output: | (None, 500, 500, 16)
conv2d_8 | input: | (None, 500, 500, 32)
Conv2D | output: | (None, 500, 500, 16)
conv2d_10 | input: | (None, 500, 500, 32)
Conv2D | output: | (None, 500, 500, 16)
conv2d_13 | input: | (None, 500, 500, 64)
Conv2D | output: | (None, 500, 500, 16)
add_1 | input: | [(None, 500, 500, 16), (None, 500, 500, 16), (None, 500, 500, 16)]
Add | output: | (None, 500, 500, 16)
conv2d_12 | input: | (None, 500, 500, 16)
Conv2D | output: | (None, 500, 500, 16)
add_2 | input: | [(None, 500, 500, 16), (None, 500, 500, 16), (None, 500, 500, 16)]
Add | output: | (None, 500, 500, 16)
conv2d_15 | input: | (None, 500, 500, 16)
Conv2D | output: | (None, 500, 500, 16)
conv2d_16 | input: | (None, 500, 500, 16)
Conv2D | output: | (None, 500, 500, 3)


Ground Truth
0
100
200
300
400
0
100
200
300
400
Stitch


Low Light Image
0
100
200
300
400
0
100
200
300
400


Enhanced Image
0
100
200
300
400
0
100
200
300
400
Stitch

REFERENCES

- Machinelearningmastery.com
- https://www.youtube.com/live/GK5ATUhdH74?si=umyaNz3ZY-VpjK_6
- https://youtu.be/tMjdQLylyGI?si=VWnXSudpp5r4Wbu5
- https://www.geeksforgeeks.org/weight-initialization-techniques-for-deep-neural-networks/
- (4466) Building a Low-Light Enhancement model using Convolutional Neural Networks (CNN) | Tutorial – YouTube
- Low Light Enhancement - Using CNNs - Shared.ipynb - Colaboratory (google.com)
- Low Light Image Enhancement with Keras MIRNet | Kaggle
- https://chat.openai.com

DECISION TREE :
A VISUAL APPROACH TO DECISION-MAKING

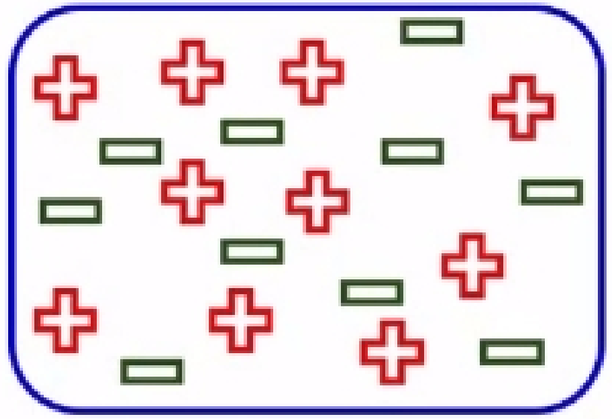
Subject : Artificial Inteligence and machine Learning Techniques

Presented To : Dr. Yogita Hande

Presented By : Vinay More

Course : TY BTECH CSF

What is Decision Tree?



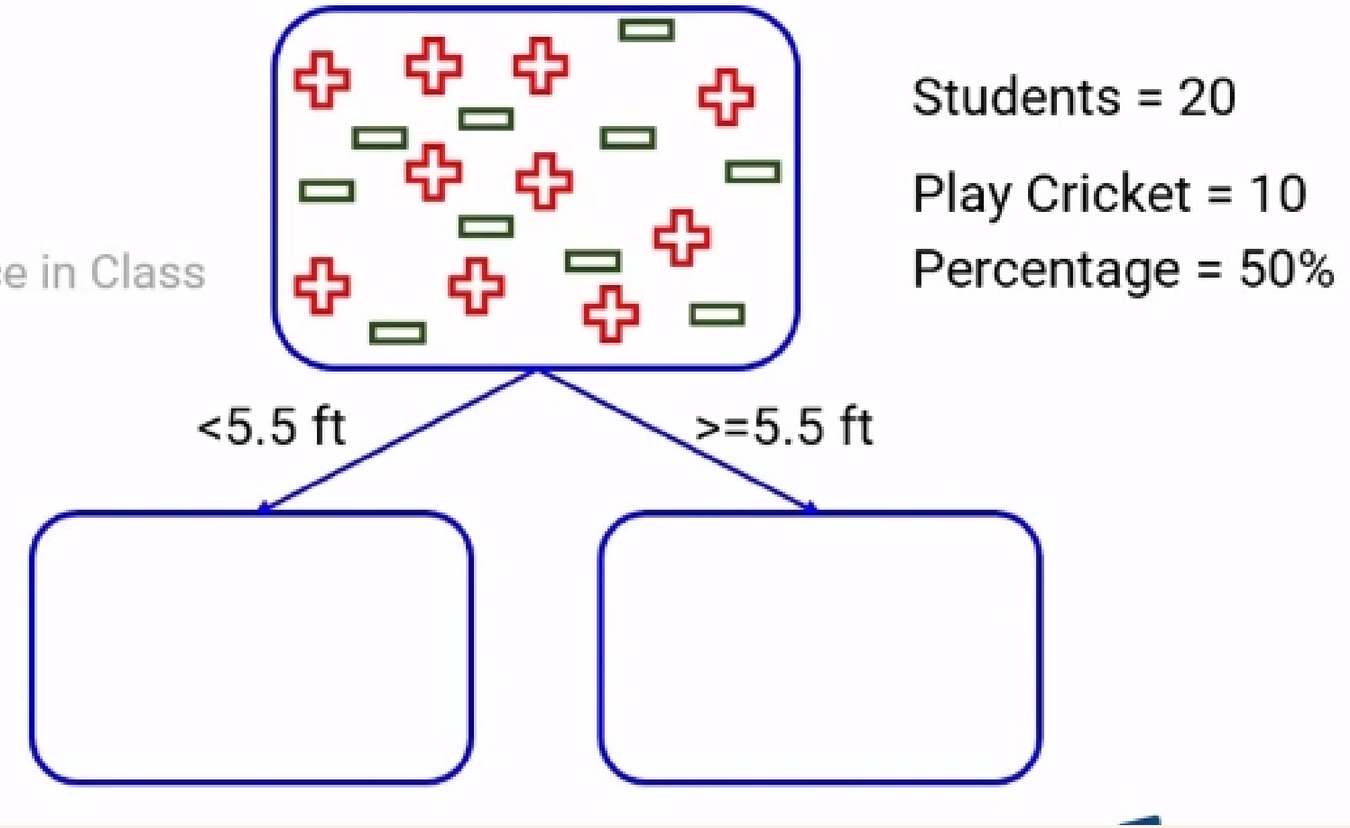
Total number of students = 20

Play cricket = 10

Do not play cricket = 10

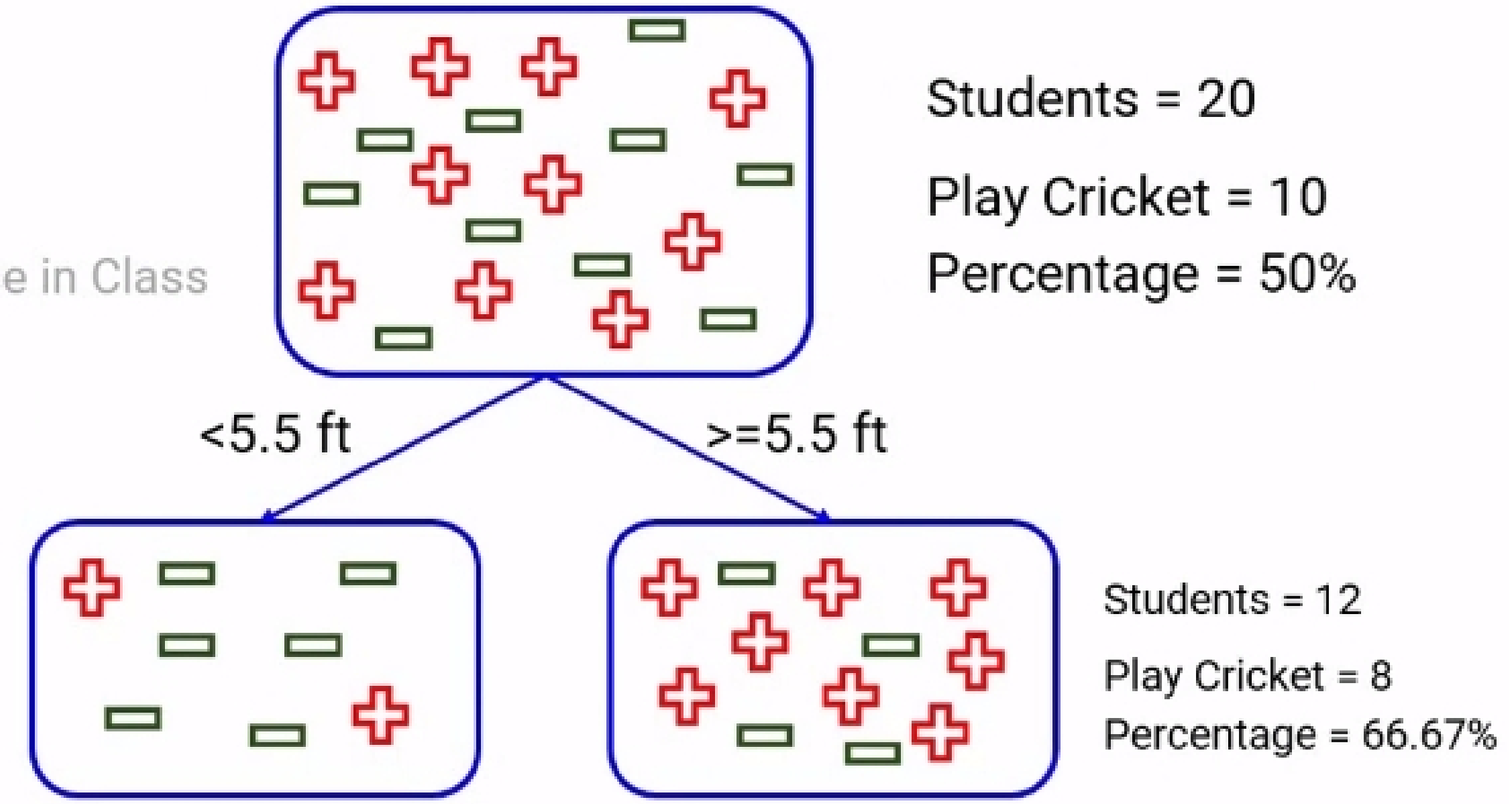
What is Decision Tree?

- Split on Height
- Split on Performance in Class
- Split on Class



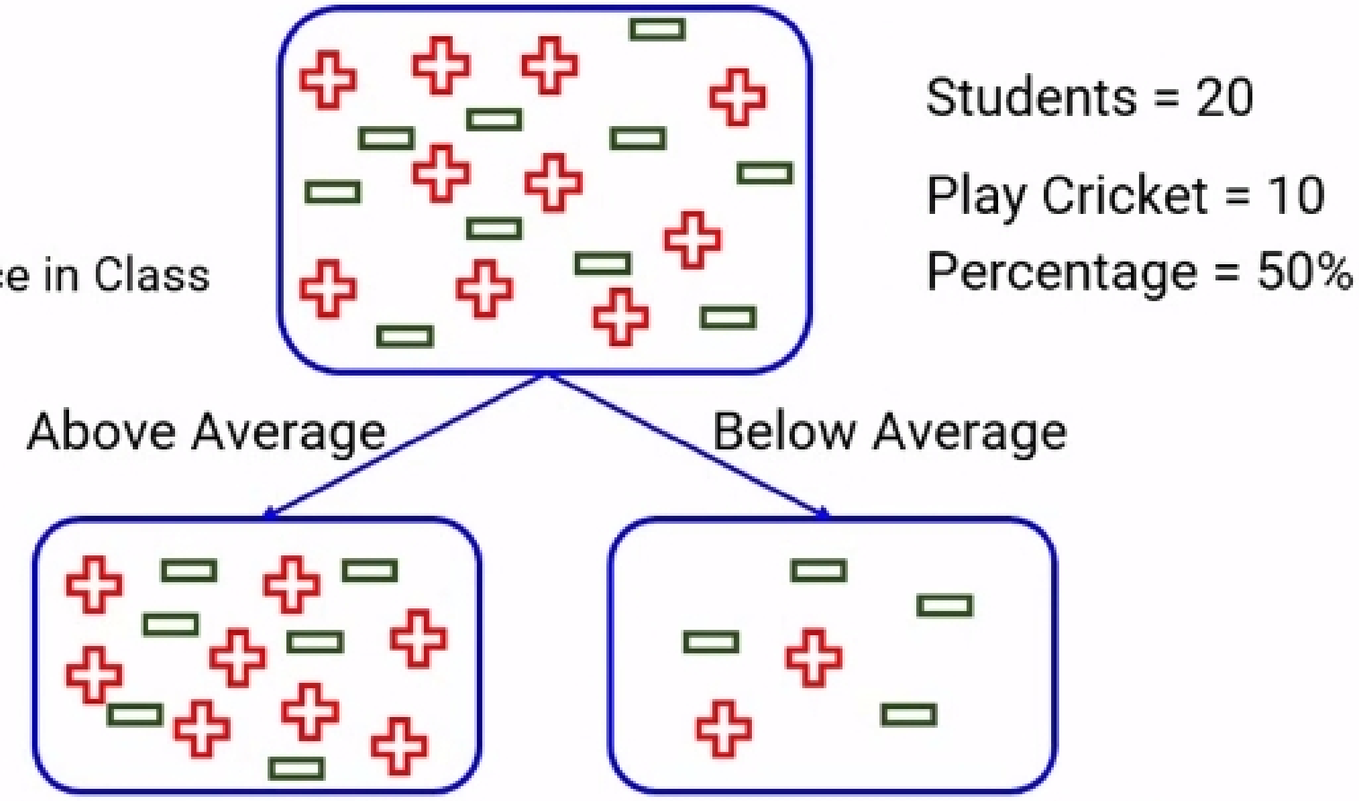
What is Decision Tree?

- Split on Height
- Split on Performance in Class
- Split on Class



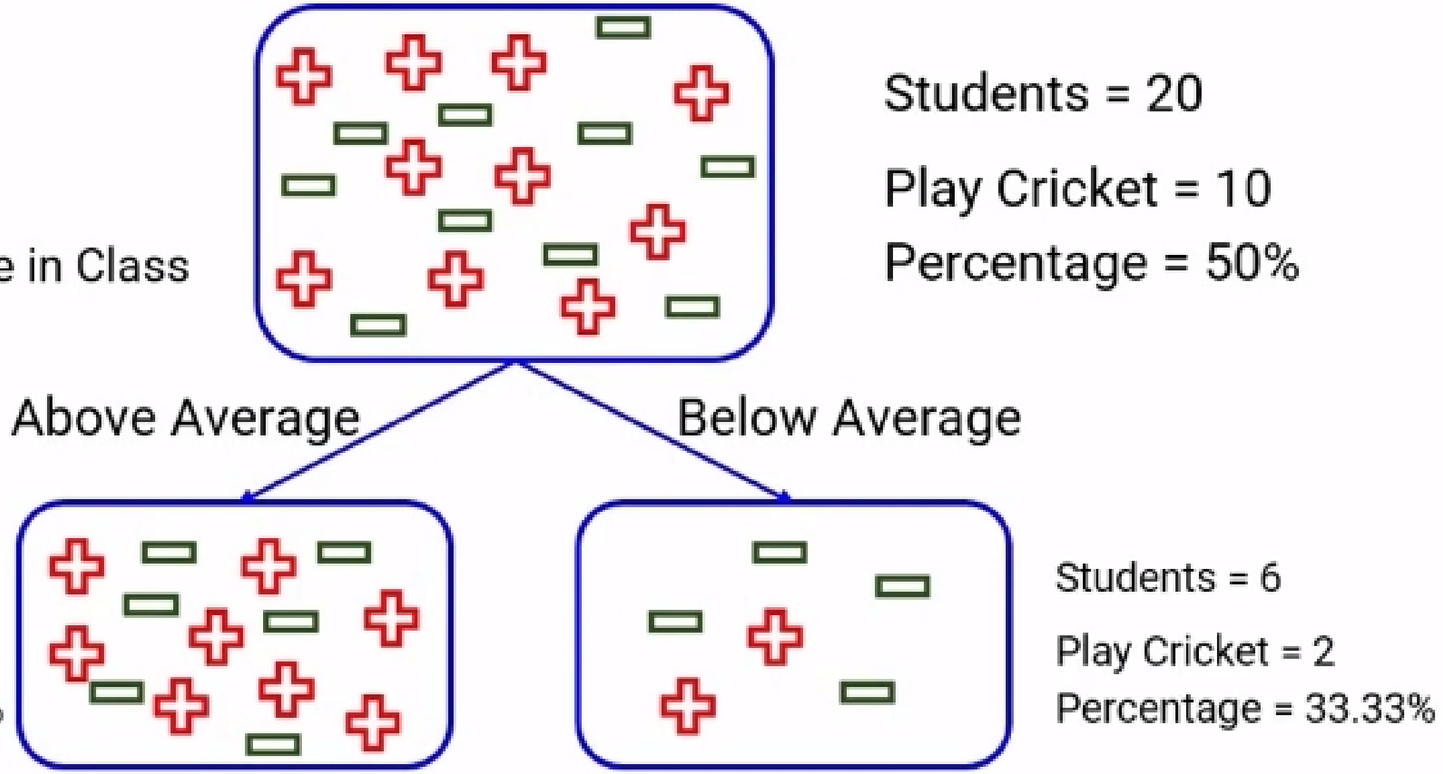
What is Decision Tree?

- Split on Height
- Split on Performance in Class
- Split on Class



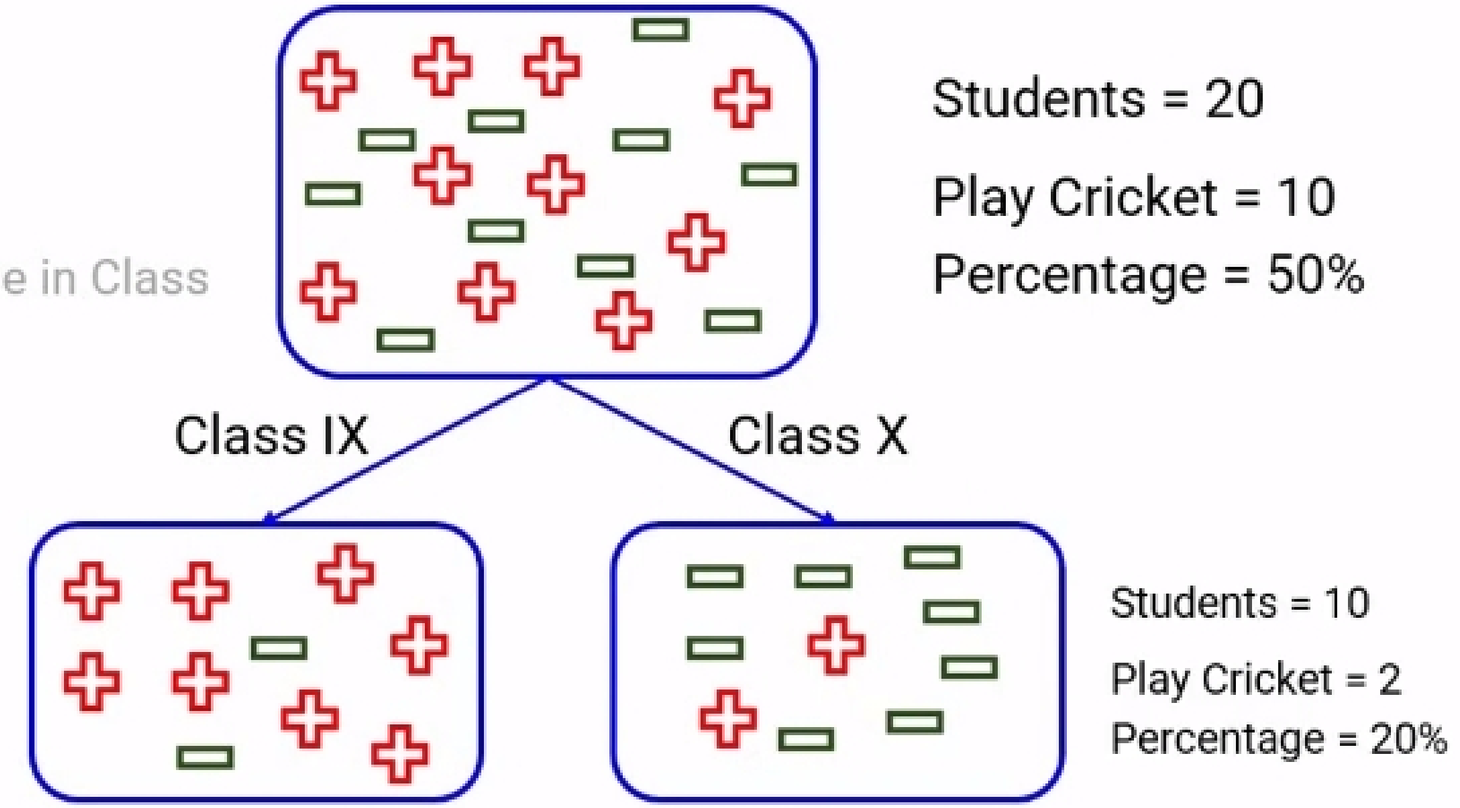
What is Decision Tree?

- Split on Height
- Split on Performance in Class
- Split on Class

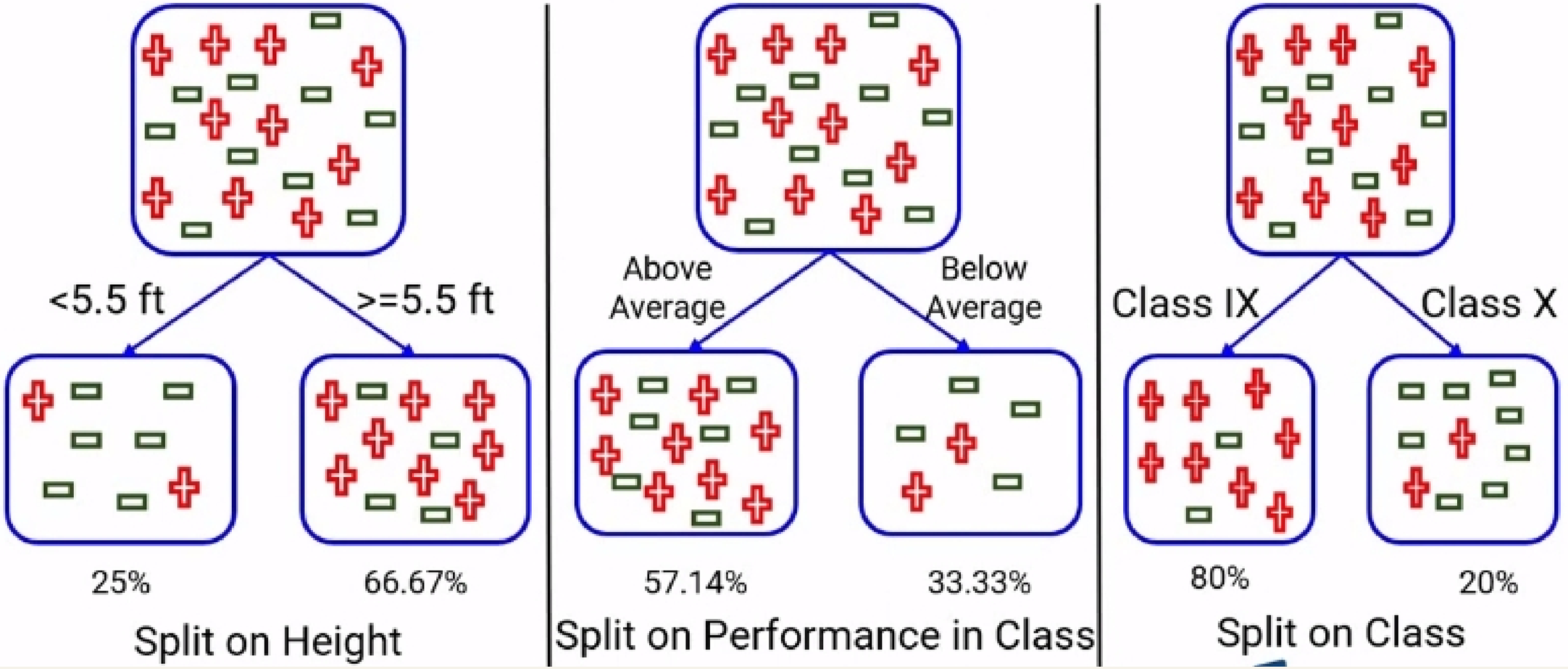


What is Decision Tree?

- Split on Height
- Split on Performance in Class
- Split on Class

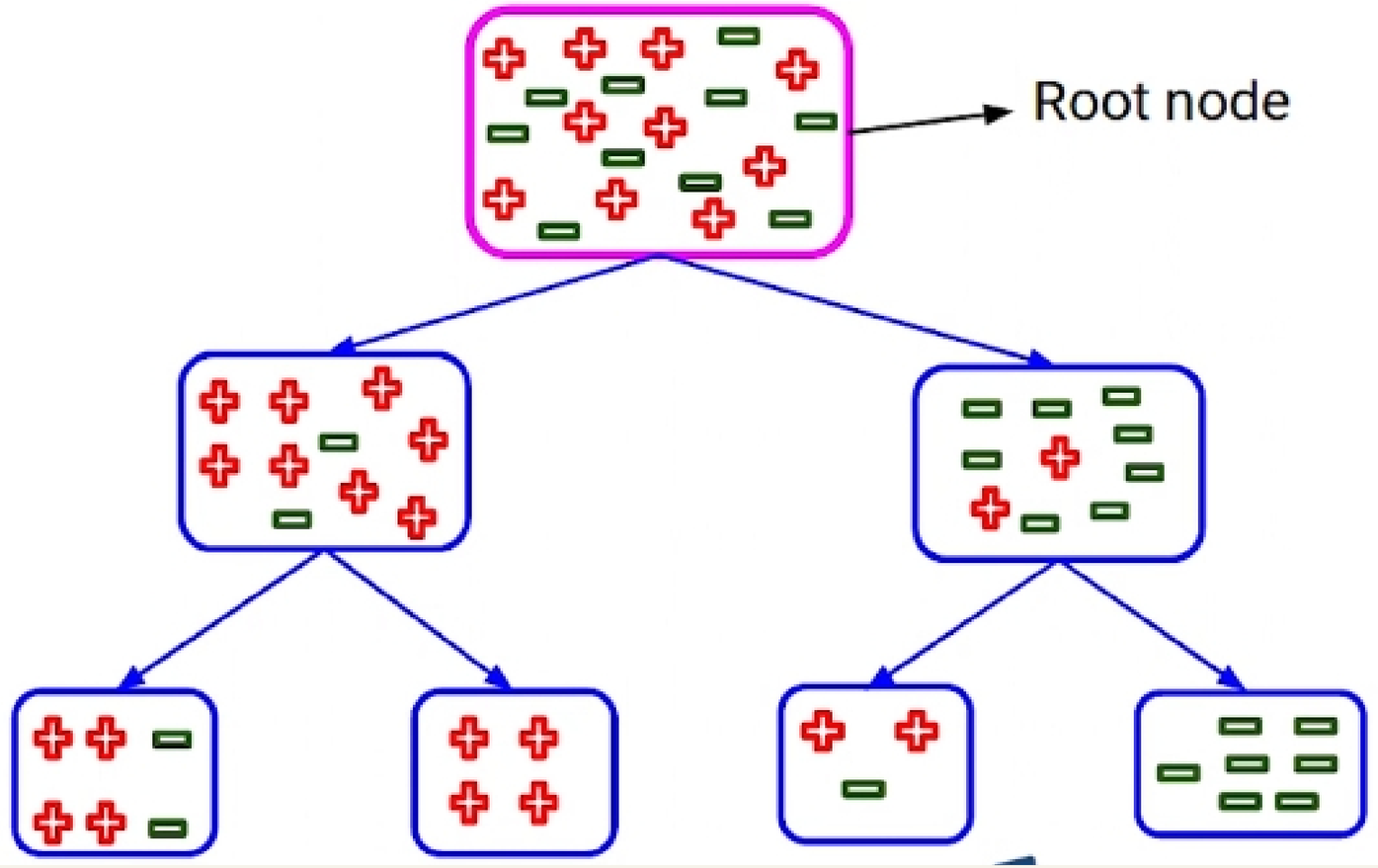


What is Decision Tree?



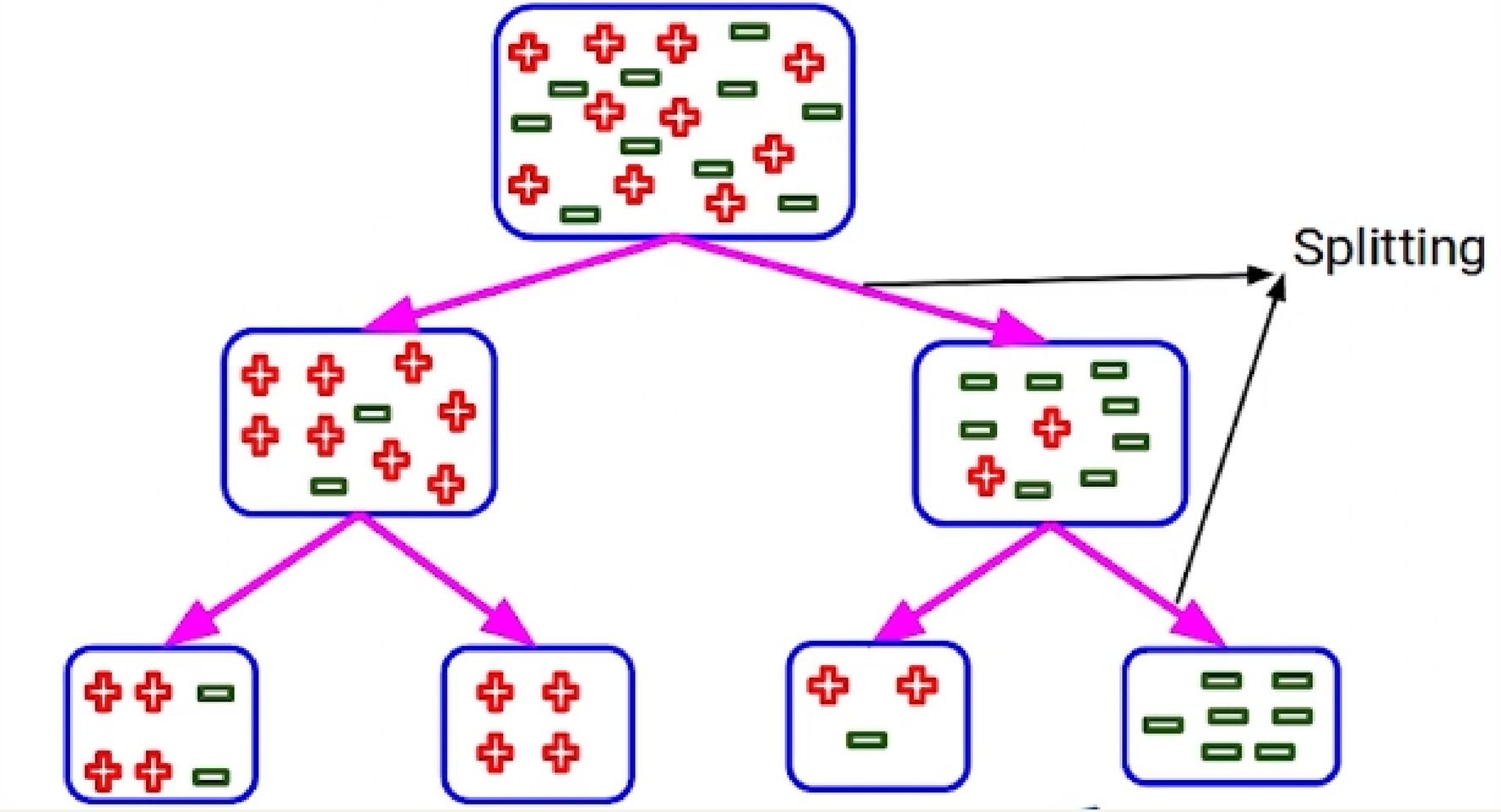
Terminologies related to Decision Trees

- Root Node



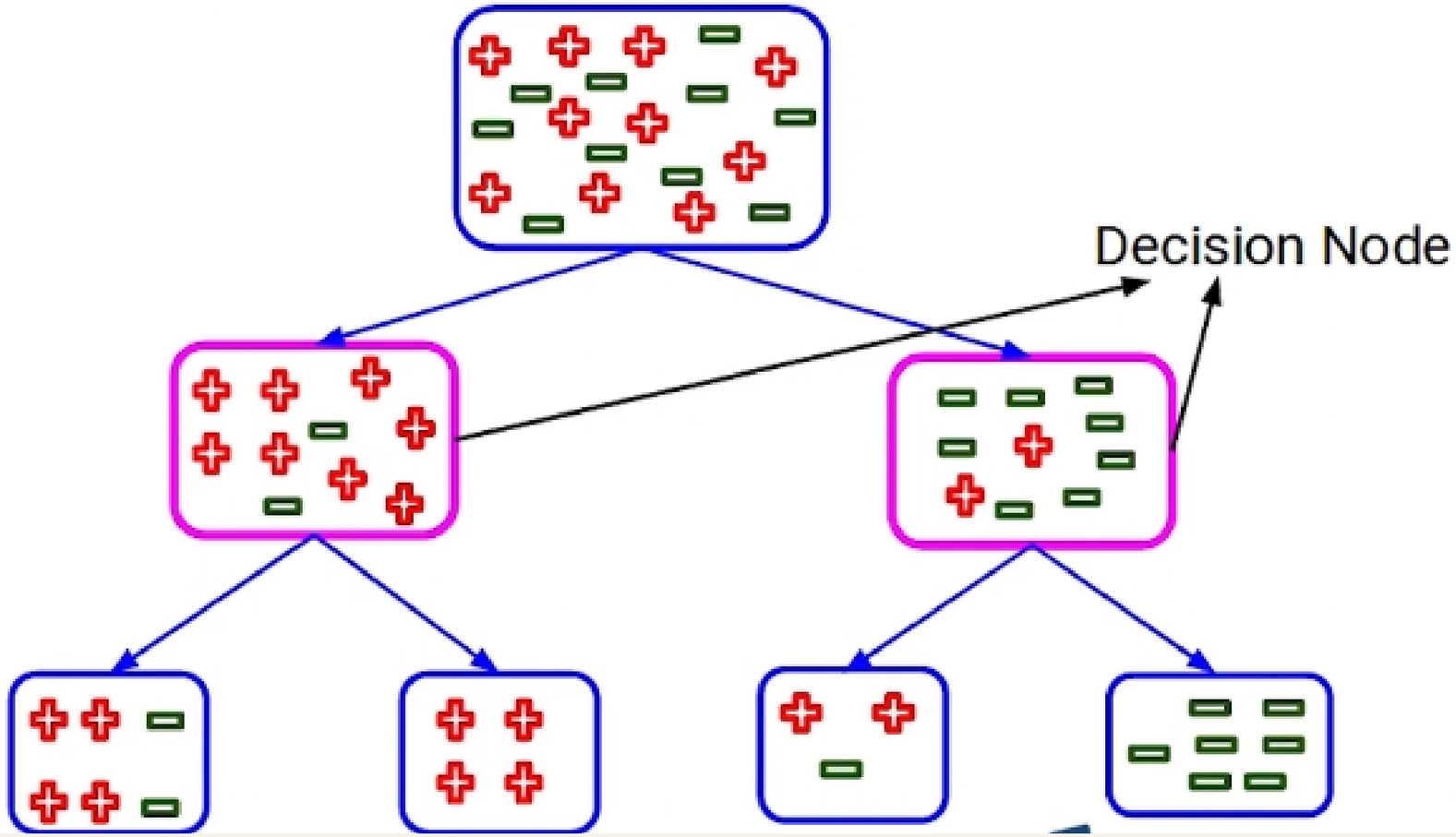
Terminologies related to Decision Trees

- Root Node
- Splitting



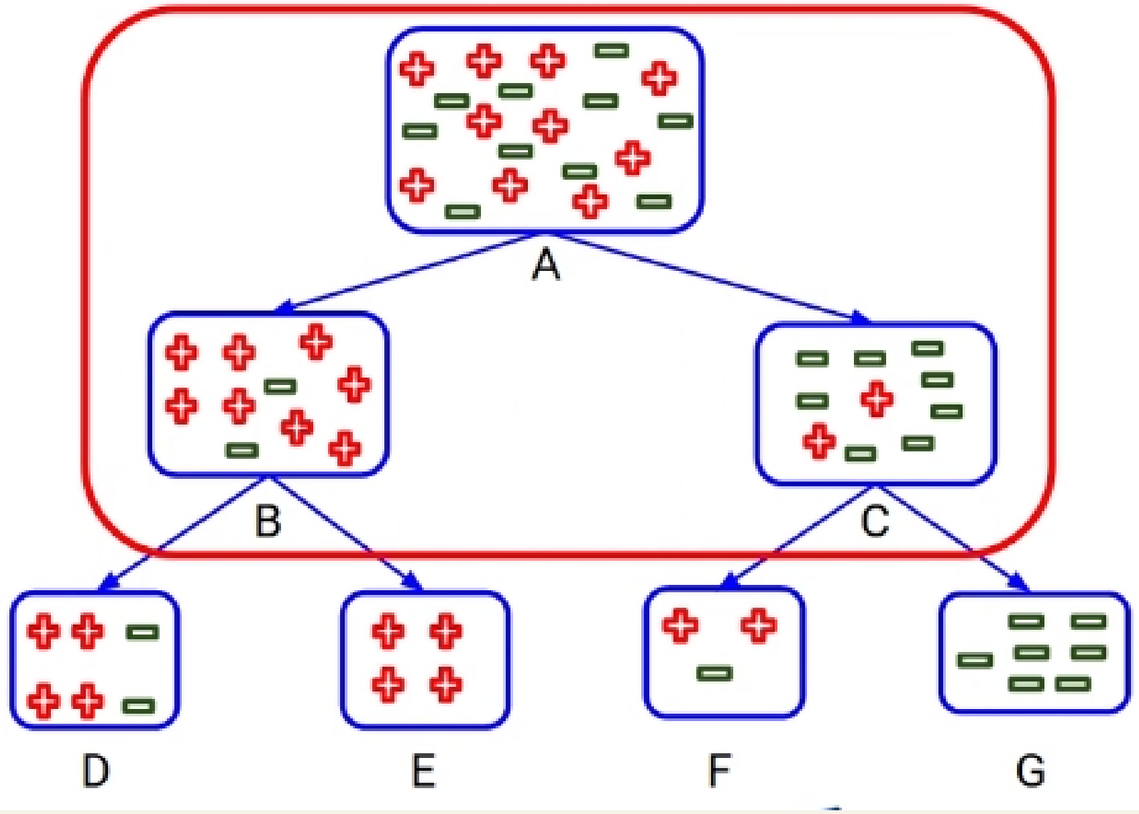
Terminologies related to Decision Trees

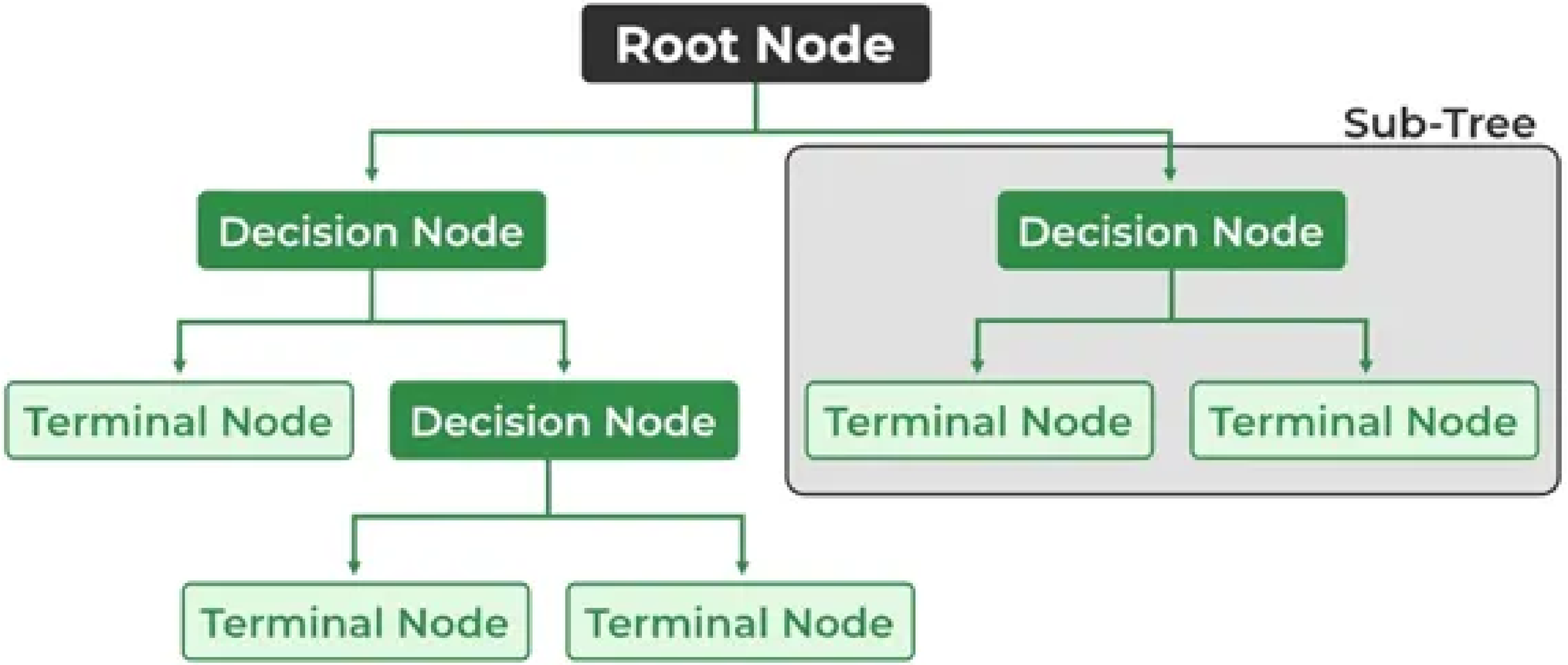
- Root Node
- Splitting
- Decision Node



Terminologies related to Decision Trees

- Root Node
- Splitting
- Decision Node
- Leaf/Terminal Node
- Branch/Sub-Tree
- Parent and Child node




Root Node
Sub-Tree
Decision Node
Decision Node
Terminal Node
Decision Node
Terminal Node
Terminal Node
Terminal Node
Terminal Node

DECISION TREE ALGORITHMS

1.)CART (Classification and Regression Trees):

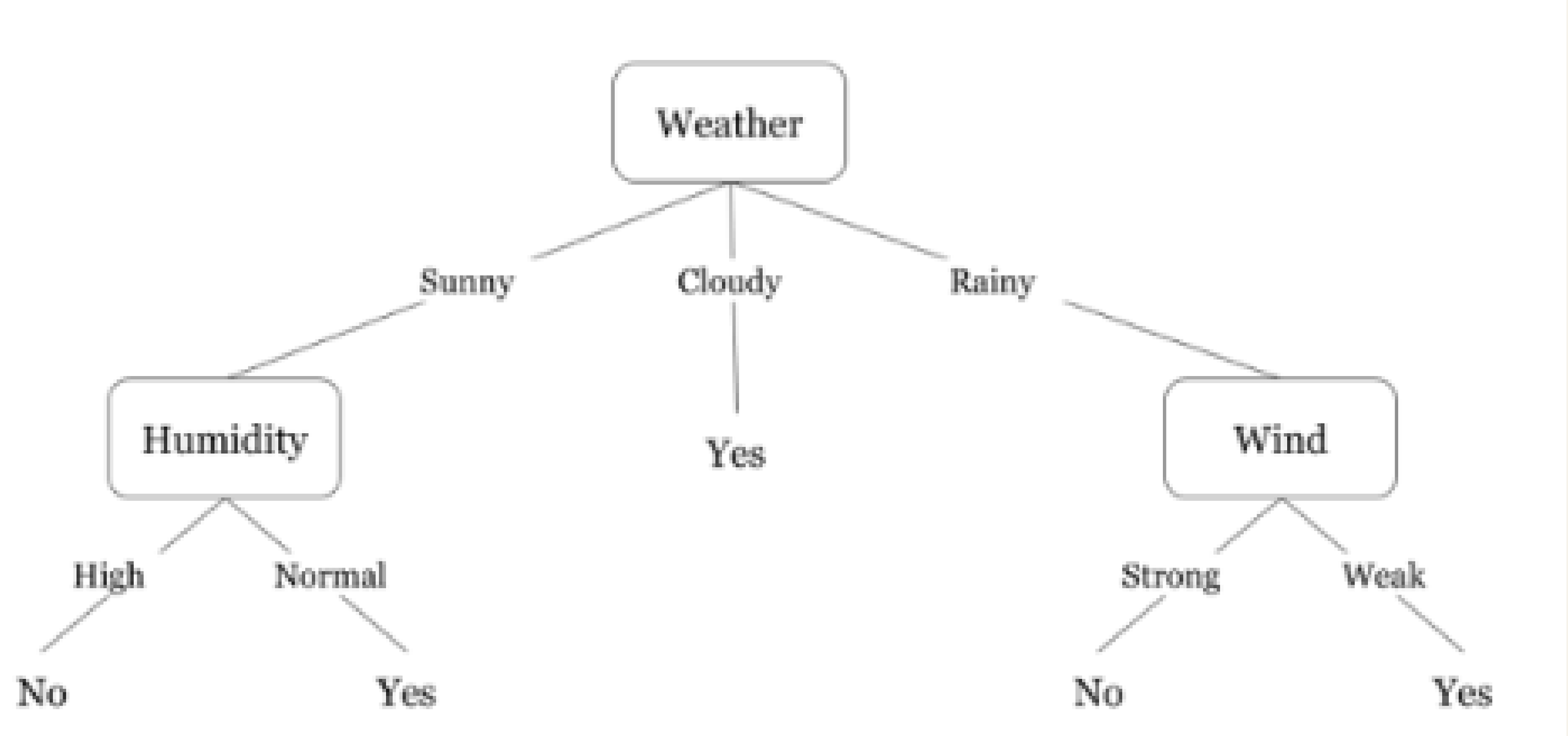
- Purpose: Handles both classification and regression tasks.
- Splitting Criterion: Uses Gini impurity for classification and mean squared error for regression.
- Process: Recursively splits the dataset based on the feature that provides the best impurity reduction.

2.)ID3 (Iterative Dichotomiser 3):

- Purpose: Primarily used for classification tasks.
- Splitting Criterion: Uses information gain to determine the best attribute for splitting.
- Process: Recursively builds the tree by selecting attributes that maximize information gain at each node.

Let's understand decision trees with the help of an example:

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


Weather
Sunny
Cloudy
Rainy
Humidity
Yes
Wind
High
Normal
Strong
Weak
No
Yes
No
Yes

Decision trees are upside down which means the root is at the top and then this root is split into various several nodes. Decision trees are nothing but a bunch of if-else statements in layman terms. It checks if the condition is true and if it is then it goes to the next node attached to that decision.

ENTROPY :

The formula for Entropy is shown below:

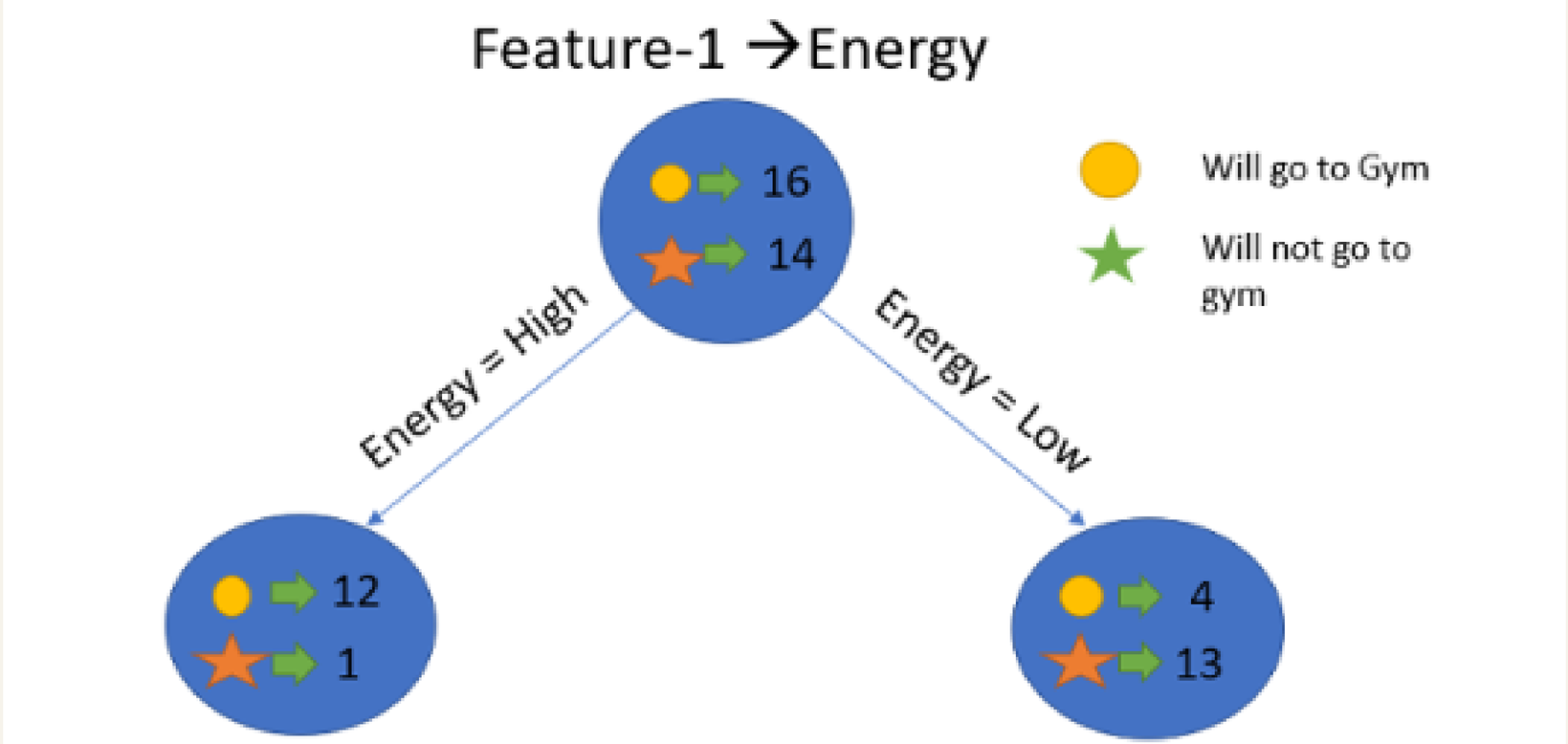
$$E(S) = -p_{(+)}\log p_{(+)} - p_{(-)}\log p_{(-)}$$

iNFORMATION GAIN :

Information gain measures the reduction of uncertainty given some feature and it is also a deciding factor for which attribute should be selected as a decision node or root node.

The formula for Entropy is shown below:

$$Information\ Gain = E(Y) - E(Y|X)$$



Let's calculate the entropy

$$E(Parent) = -\left(\frac{16}{30}\right)\log_2\left(\frac{16}{30}\right) - \left(\frac{14}{30}\right)\log_2\left(\frac{14}{30}\right) \approx 0.99$$

$$E(Parent|Energy = "high") = -\left(\frac{12}{13}\right)\log_2\left(\frac{12}{13}\right) - \left(\frac{1}{13}\right)\log_2\left(\frac{1}{13}\right) \approx 0.39$$

$$E(Parent|Energy = "low") = -\left(\frac{4}{17}\right)\log_2\left(\frac{4}{17}\right) - \left(\frac{13}{17}\right)\log_2\left(\frac{13}{17}\right) \approx 0.79$$

To see the weighted average of entropy of each node we will do as follows:

$$E(Parent|Energy) = \frac{13}{30} * 0.39 + \frac{17}{30} * 0.79 = 0.62$$

Now we have the value of E(Parent) and E(Parent|Energy), information gain will be:

$$\begin{aligned} Information\ Gain &= E(parent) - E(parent|energy) \\ &= 0.99 - 0.62 \\ &= 0.37 \end{aligned}$$

Our parent entropy was near 0.99 and after looking at this value of information gain, we can say that the entropy of the dataset will decrease by 0.37 if we make "Energy" as our root node.

Steps to Calculate Gini Impurity for a Split

Let's now look at the steps to calculate the Gini split.

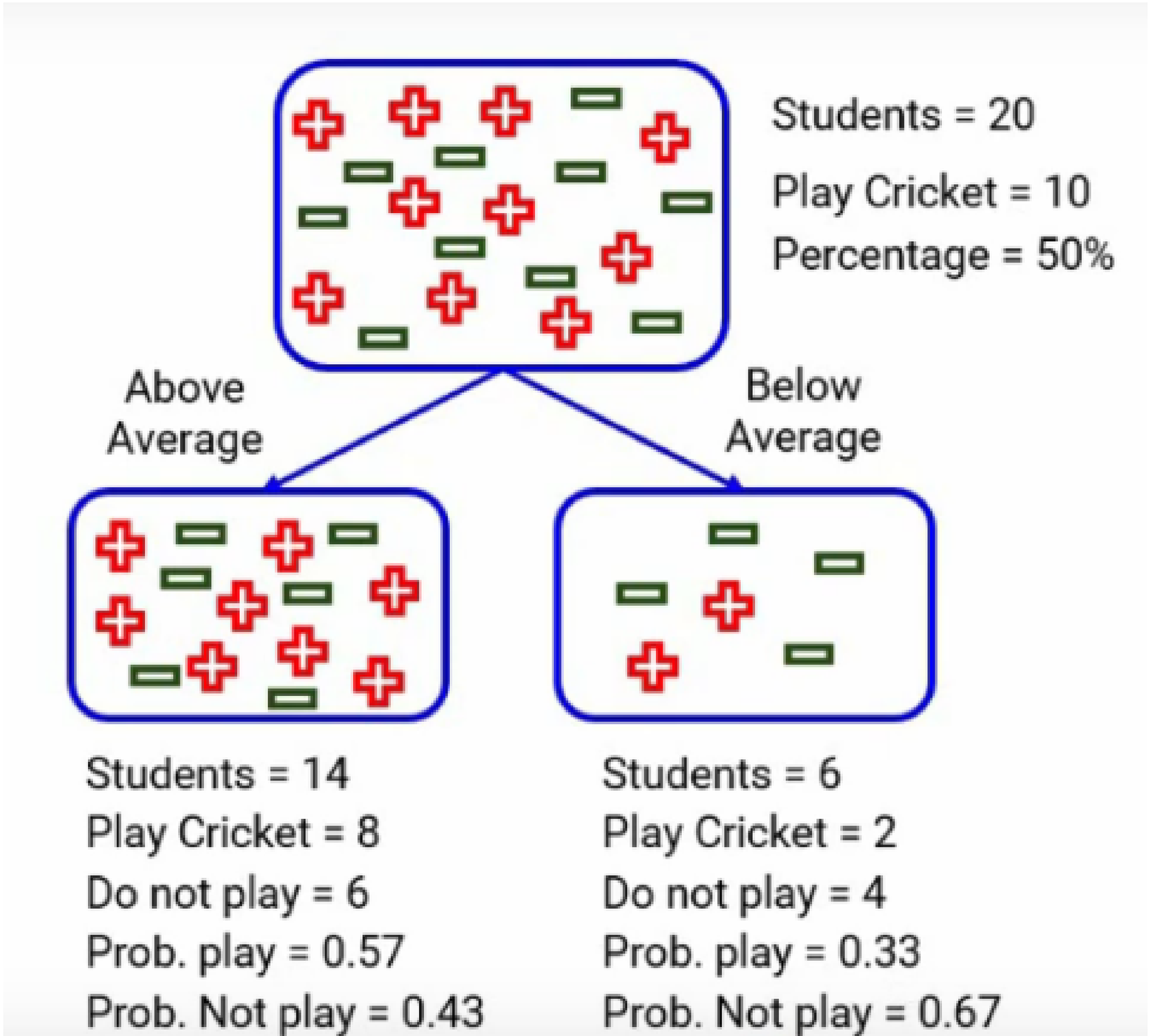
Step 1: Calculate GI for Sub-nodes

First, we calculate the Gini impurity for sub-nodes, as you've already discussed Gini impurity is, and I'm sure you know this by now:

Gini impurity = 1 – Gini

Here is the sum of squares of success probabilities of each class and is given as:

$$Gini = (p_1^2 + p_2^2 + p_3^2 + \ldots + p_n^2)$$


Students = 20
Play Cricket = 10
Percentage = 50%
Above Average
Below Average
Students = 14
Play Cricket = 8
Do not play = 6
Prob. play = 0.57
Prob. Not play = 0.43
Students = 6
Play Cricket = 2
Do not play = 4
Prob. play = 0.33
Prob. Not play = 0.67

Split on Class

Gini Impurity: sub-node Class IX:
1 - [(0.8)*(0.8) + (0.2)*(0.2)] = 0.32

Gini Impurity: sub-node Class X:
1 - [(0.2)*(0.2) + (0.8)*(0.8)] = 0.32

Weighted Gini Impurity: Class:
(10/20)*0.32 + (10/20)*0.32 = 0.32

Now, if we compare the two Gini impurities for each split-

Split	Weighted Gini Impurity
Performance in Class	0.475
Class	0.32

ADVANTAGES OF DECISION TREES

I.) Interpretability:

- Decision trees provide a visually intuitive and transparent representation of decision-making processes.

2.) Handling Diverse Data:

- Accommodates both numerical and categorical data without extensive preprocessing.
- Automatic feature selection capability reduces the need for manual data transformation.

3.)Efficiency and Robustness:

- Efficiently processes large datasets without significant computational overhead.

LIMITATIONS OF DECISION TREES

I.)Overfitting:

- Decision trees can be prone to overfitting, capturing noise in the training data as if it were a real pattern.
- Complex trees with too many branches may perform well on training data but poorly on new, unseen data.
-

2.)Sensitivity to Small Variations:

- Decision trees are sensitive to small changes in the data, which can lead to different tree structures.
- A small change in the input data might result in a completely different set of decisions and outcomes.

THANK YOU

Presented By : Vinay More

MIT WPU| 2025