Training Manual on

Application of Machine Learning Technique in Agriculture

Compiled and Edited by

Hasan Md. Hamidur Rahman

Course Director, Computer and GIS Unit

Hasan Mahmud

Senior System Analyst, Computer and GIS Unit

Al-Helal

Programmer, Computer and GIS Unit

Rashedul Islam

Asst. Programmer, Computer and GIS Unit

Computer & GIS Unit
Bangladesh Agricultural Research Council

Published by

Computer and GIS Unit, BARC Farmgate, Dhaka 1215, Bangladesh

Date of Publication

May 2025

Contributors:

1. Mr. Hasan Md. Hamidur Rahman, Director, Computer and GIS Unit, BARC
2. Professor Dr. Syed Shahadat Hossain, Institute of Statistical Research and Training (ISRT) University of Dhaka, Bangladesh
3. Dr. Md. Zulfiker Mahmud, Professor, Department of CSE, Jagannath University
4. Hasan Mahmud, Senior System Analyst, Computer and GIS Unit, BARC
5. Mr. Al-Helal, Programmer (Com & GIS), BARC
6. Mr. Aditya Rajbongshi, Lecturer, Gazipur Digital University, Bangladesh
7. Mr. Md. Rasel Biswas, Lecturer Institute of Statistical Research and Training (ISRT) University of Dhaka, Bangladesh

Designed by

Mohammad Nazmul Islam, Graphics Designer, BARC

Funded by

Program on Agricultural and Rural Transformation for Nutrition Entrepreneurship and Resilience (PARTNER), BARC Component

Computer & GIS Unit
Bangladesh Agricultural Research Council

Table of Content

Introduction to Machine Learning, Deep Learning, Data Science, and Big Data Analytics—

Syed Shahadat Hossain, Ph.D.
shahadat@isrt.ac.bd
Professor
Institute of Statistical Research and Training
University of Dhaka, Banglades

Data Science
Dictionary meanings of the word Data Science:
UNCOUNTABLE NOUN Data science is the scientific analysis of large amounts of information held on computers, done for example in order to learn about people's shopping or voting behaviour. The social network has its own in-house team working on data science. *[Collins Languages]*

Data Science
Data science is the study of data to extract meaningful insights from data generated perpetually in diverse domains of life. A multidisciplinary approach combines principles and practices from the fields of mathematics, statistics, artificial intelligence and computer engineering to analyze large amounts of data.

Mainly evolved to support the understanding and interpretation of the large amounts of data which has been being amassed from the processes of reporting, registering, recording (manual/digital/automated) over time.

This helps data scientists to ask and answer questions like what happened, why it happened, what will happen, and what can be done with the results.

The term **Data Science** was first used in the early 1960s to describe a new profession that would support analyzing the then hugely amassed data particularly from businesses. (At the time, there was no way of predicting the truly massive amounts of data over the next sixty years.)

In 1962, John Tukey wrote a paper titled The Future of Data Analysis and described a shift in the world of statistics, saying, *"… as I have watched mathematical statistics evolve, I have had cause to wonder and to doubt…I have come to feel that my central interest is in data analysis…"*

In 1977, The International Association for Statistical Computing (IASC) was formed with a mission statement emphasizing linking traditional statistical methodology, modern computer technology, and the knowledge of domain experts.

Subject matter: Data

Modern stalwarts in the development of the subject of Data Science is
In 1999, in Mining Data for Nuggets of Knowledge, Jacob Zahavi pointed out the need for new tools beyond conventional statistical methods.

In 2015, declared as a a landmark year for AI, using Deep Learning techniques, Google's speech recognition, Google Voice, experienced a dramatic performance jump of 49 percent.
2023 experiences the magic of NLPM and ChatGPT.

John Tukey (1915-2000) contributed greatly to statistical practice and data analysis in general. In fact, some regard John Tukey as the father of Data Science. At the very least, he pioneered many of the key foundations of what came later to be known as Data Science. He was credited with coining two terms used in the computer industry: the first was "software," or the programs used to run a computer, and the second was "bit," a contraction for binary digit.

Subject matter: Analysis

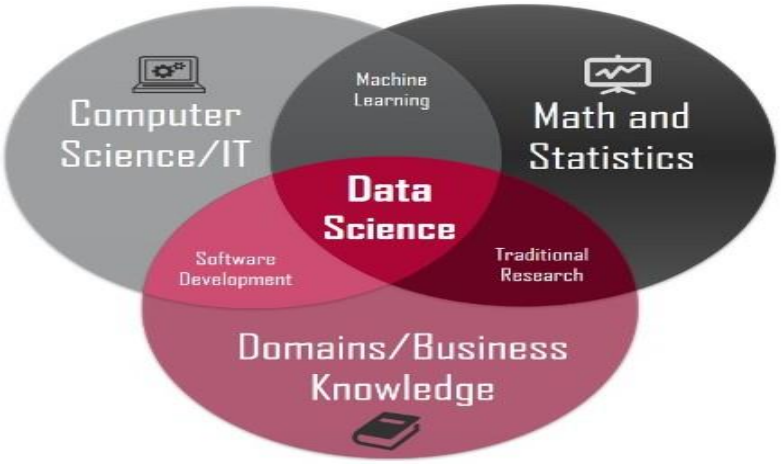
Statistics	Data Science /Applied Statistics
Computation Scientific	Statistical Softwares Tools Calculators, Excel, Python, R, SQL,
Statistical	Mostly by retrieving Softwares like C++, algorithms and pakages R, Stata, SPSS, SAS available on internet/cloud

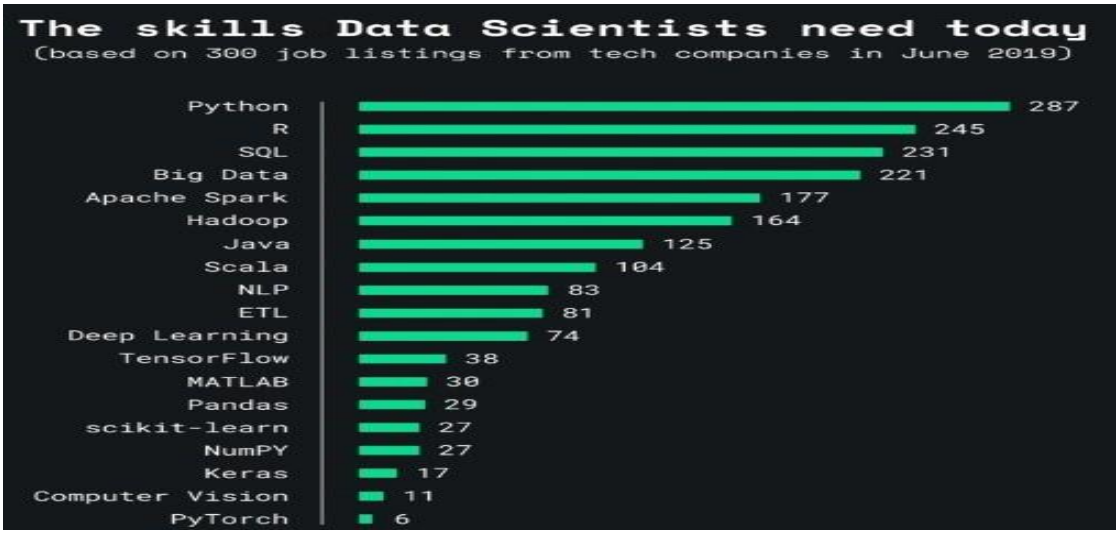
Subject matter: Analysis

Statistics	Data Science /Applied Statistics Decision Probabilistic, Mostly deterministic, Making States level of Rarely states level Confidence of Credibility and validity

Careers in Data Science

Skills used for Careers in Data Science





Data Set

Data Base

A database is an organized collection of structured information, or data, typically stored electronically in a computer system.

A database is usually controlled by a database management system (DBMS).

Together, the data and the DBMS, along with the applications that are associated with them, are referred to as a database system, often shortened to just database.

Metadata

DATA:

Data can be stored in a database which has one dimension – a list, or two dimensions – a grid made up of rows and columns. It can also be stored in multi-dimensional databases which can take the form of a grid, with three axes, or even more complex permutations.

More complex dimensional structures typically allow for more connections to be observed between the data objects which are being analyzed.

Data Governance

Anonymization

When carrying out scientific data analysis using personal data (data which identifies a person), anonymization refers to the process of removing or obfuscating indicators in the data which show who it specifically refers to.

This is not always as simple as it sounds as people can be identified by more than just their name. Properly anonymized data is no longer considered "personal" and there are commonly less legal and ethical restrictions on how it can be used.

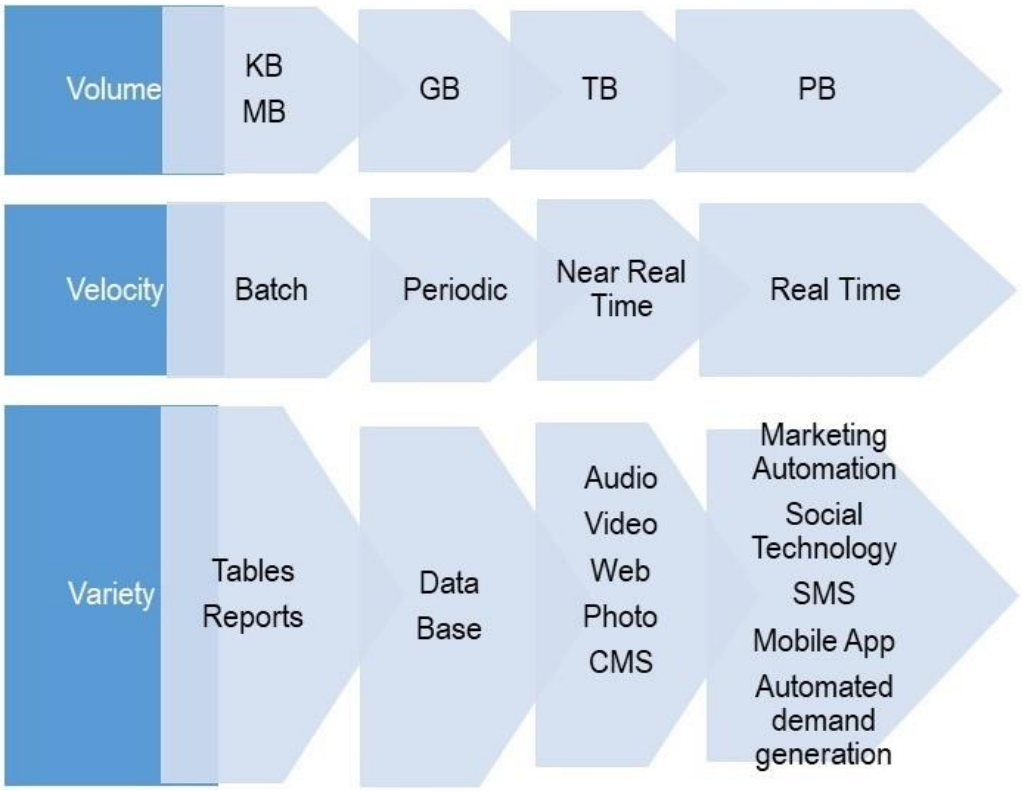
Big Data

Big Data: Big Data is the buzzword term which has come to represent the vast increase in the amount of data which has become available in recent years, particularly as the world has increasingly become online and connected through the internet.

This data is distinguished from data previously available not just by its size but also the high speed at which it is generated, and the large variations in the forms it can take.

It greatly expands the potential of what can be achieved with data science, which was previously hampered by slow computer processing speeds and the difficulty of capturing accurate information in large volumes, before the widespread digitization.

Big.



Structured and unstructured Data

Structured data is highly organized and formatted so that it's easily searchable in relational databases. Unstructured data is the polar opposite of structured data. It is scattered and variable and has no predefined format or organization, making it much more difficult to collect, process, and analyze. Unstructured data.

Features of Structured Data

This data is more finite and sorted into data arrays.
Structured data is most often categorized as quantitative data, and it's the type of data most of us are used to working with.

Structured data is highly organized and fits neatly within fixed fields and columns in relational databases and spreadsheets. This is easily understood by machine language.

Structured data can be easily fed into machine learning models as input datasets without any or trimming.

It does not require any AI or ML expertise to work with structured data. A person with good product information and basic data science knowledge can work with it.

Features of Structured Data

Structured data is stored evenly in data warehouses or spreadsheets. The specific and organized nature makes it easy to manipulate and query it.

Structured data predates unstructured data, so more analytics tools are available to measure and analyze it.

The data is of higher quality, consistency, and usability than unstructured data.
Examples of structured data includes:

E-commerce: Review data, pricing data, and SKU number of commodities
Healthcare: hospital administration, pharmacy, and patient data and medical history of patients.

Banking: Financial transaction details like bank ad beneficiary, account details, sender or receiver information.

Features of Unstructured Data

Unstructured data is scattered and variable and has no predefined format or organization, making it much more difficult to collect, process, and analyze.

This is highly complex, qualitative, and unorganized. It does not conform to any one particular standard. This data can be numerical, alphabetical, boolean or a mix of all of them.

It cannot be processed and analyzed using conventional data tools and methods.
Unstructured data is easily available and has enough insights businesses can collect to learn about their product response.

Features of Unstructured Data

Unstructured data can be stored on shared or hybrid cloud servers with minimal expenditure on database management.

Unstructured data is in its native format, so data scientists or engineers do not define it until needed. Hence minor alterations to the database do not impact cost, time, or resources.
Examples of unstructured data includes:

Rich media: Social media, entertainment, surveillance, satellite information, geospatial data, weather forecasting, podcasts Documents: Invoices, records, web history, emails, productivity applications

Media and entertainment data, surveillance data, geospatial data, audio, weather data

Descriptive:

Descriptive analysis examines data to gain insights into what happened or what is happening in the data environment.

It is characterized by data visualizations such as pie charts, bar charts, line graphs, tables, or generated narratives.

For example, a flight booking service may record data like the number of tickets booked each day. Descriptive analysis will reveal booking spikes, booking slumps, and high-performing months for this service.

Diagnostic:

Diagnostic analysis is a deep-dive or detailed data examination to understand why something happened.

It is characterized by techniques such as drill-down, data discovery, data mining, and correlations.

Multiple data operations and transformations may be performed on a given data set to discover unique patterns in each of these techniques.

For example, the flight service might drill down on a particularly high-performing month to better understand the booking spike. This may lead to the discovery that many customers visit a particular city to attend a monthly sporting event.

A Business Model Perspective

Predictive.

Predictive analysis uses historical data to make accurate forecasts about data patterns that may occur in the future.

It is characterized by techniques such as machine learning, forecasting, pattern matching, and predictive modeling. In each of these techniques, computers are trained to reverse engineer causality connections in the data.

For example, the flight service team might use data science to predict flight booking patterns for the coming year at the start of each year. The computer program or algorithm may look at past data and predict booking spikes for certain destinations in May. Having anticipated their customer's future travel requirements, the company could start targeted advertising for those cities from February

Prescriptive analysis

Prescriptive analytics takes predictive data to the next level. It not only predicts what is likely to happen but also suggests an optimum response to that outcome.

It can analyze the potential implications of different choices and recommend the best course of action. It uses graph analysis, simulation, complex event processing, neural networks, and recommendation engines from machine learning.

Prescriptive analysis (cont.)

Back to the flight booking example, prescriptive analysis could look at historical marketing campaigns to maximize the advantage of the upcoming booking spike. A data scientist could project booking outcomes for different levels of marketing spend on various marketing channels. These data forecasts would give the flight booking company greater confidence in their marketing decisions.

O – Obtain data

Data can be pre-existing, newly acquired, or a data repository downloadable from the internet.

Data scientists can extract data from internal or external databases, company CRM software, web server logs, social media or purchase it from trusted third-party sources.

S – Scrub data

Data scrubbing, or data cleaning, is the process of standardizing the data according to a predetermined format.
It includes handling missing data, fixing data errors, and removing any data outliers. Some examples of data scrubbing are:

Changing all date values to a common standard format. Fixing spelling mistakes or additional spaces.

Fixing mathematical inaccuracies or removing commas from large numbers.

E – Explore data

Data exploration is preliminary data analysis that is used for planning further data modeling strategies.

Data scientists gain an initial understanding of the data using descriptive statistics and data visualization tools.

Then they explore the data to identify interesting patterns that can be studied or actioned.

M – Model data

Software and machine learning algorithms are used to gain deeper insights, predict outcomes, and prescribe the best course of action.
Machine learning techniques like association, classification, and clustering are applied to the training data set.
The model might be tested against predetermined test data to assess result accuracy. The data model can be fine-tuned many times to improve result outcomes.

N – Interpret results

Data scientists work together with analysts and businesses to convert data insights into action.

They make diagrams, graphs, and charts to represent trends and predictions.
Data summarization helps stakeholders understand and implement results effectively.

Data Mining

The process of examining a set of data to determine relationships between variables which could affect outcomes – generally at large scale and by machines.

Data mining is an older term used by computer scientists and in business to describe the basic function of a data scientist or a data science initiative.

Algorithm

Repeatable sets of instructions which people or machines can use to process data. Typically, algorithms are constructed by feeding data into them and adjusting variables until a desired outcome is achieved.

Thanks to breakthroughs in AI such as machine learning and neural networks today machines generally do this, as they can do it far more quickly than any human.

Classification

The ability to use data (about an object, event or anything else) to determine which of a number of predetermined groups an item belongs in.

For a basic example, an image recognition analysis might classify all shapes with four equal sides as squares, and all shapes with three sides as triangles.

Decision

A basic decision-making structure which can be used by a computer to understand and classify information. By asking a series of questions about each data item fed into them, outputs are channeled along different branches leading to different outcomes, typically labelling or classification of the piece of data.

Clustering

Clustering is also about grouping objects together but it differs because it is used when there are no predetermined groups.

Objects (or events) are clustered together due to similarities they share and algorithms determine what that common relationship between them may be.

Clustering is a data science technique which makes unsupervised learning possible.

Random Forest

A random forest is a method of statistical analysis which involves taking the output of a large number of decision trees (see above) and analyzing them together, to provide a more complex and detailed understanding or classification of data than would be possible with just one tree.

As with decision trees this is a technique that has been around in statistics for a long time but modern computers allow for far more complex trees and forests, leading to more accurate predictions.

Artificial Intelligence

Today's AIs are built on concepts developed through the study and application of data science.

One way to categorize the latest wave of intelligent machines is as machines which are capable of carrying out data science for themselves.
Rather than simply process the data they are fed, in the way they are told, they can learn it and adapt to become better at processing it.

This is how Google Translate becomes better at understanding language, and how autonomous cars will navigate areas they have never visited before.

Machine Learning

It is a field of artificial intelligence (AI) that focuses on enabling computers to learn from data without being explicitly programmed. It involves creating algorithms that can analyze patterns in data and build models for specific tasks, allowing them to make predictions and improve their performance over time.

Key aspects of machine learning

Learning from data: ML algorithms learn from large datasets to identify patterns and make predictions.

Prediction and classification: ML algorithms can be used for tasks like classifying data, predicting outcomes, and making recommendations.

Automation: It automates the process of building analytical models, reducing human intervention.

Improvement over time: As ML algorithms are trained on more data,

Machine Learning (Cont.)

Various types of ML:

Supervised learning: works with Labeled datasets (input-output pairs) Unsupervised learning: works with Unlabeled datasets (raw data)

Reinforcement learning: where an agent learns to make decisions by interacting with an environment and receiving feedback in the form of rewards or penaltie

Machine Learning (Cont.)

Feature	Supervised Learning	Unsupervised Learning
Data	Labeled datasets (input-output pairs)	Unlabeled datasets (raw data)
Objective	Prediction, classification, or regression of known outcomes	Pattern discovery, clustering, anomaly detection, etc.
Training	Learns from labeled examples to make predictions	Discovers patterns without explicit supervision
Complexity	Generally less complex than unsupervised learning	More complex, often requires powerful algorithms
Evaluation	Accuracy based on predictions against known labels	Internal metrics based on discovered patterns or structures
Resource	Requires time and effort to label data	May be more efficient with large, unlabeled datasets
Applications	Spam detection, image recognition, price prediction	Customer segmentation, recommendation systems, anomaly detection

Machine Learning (Cont.)

Examples of machine learning applications:

Medical diagnosis: Analyzing medical images and data to detect diseases.
Weather forecasting: Predicting weather patterns and conditions. Fraud detection: Identifying fraudulent transactions and activities.

Robotics: Developing robots that can perform tasks in dynamic environments.
Game playing: Creating AI players that can beat human opponents in games like Go and chess.

Personalized recommendations: Suggesting products, content, or services based on user preferences.

Self-driving cars: Developing autonomous driving systems.

Deep learning

Is a subfield of artificial intelligence (AI) and machine learning that uses artificial neural networks to analyze data and solve complex problems. It involves training these networks with vast amounts of data to enable them to learn patterns and make predictions, similar to how the human brain learns. Deep learning is used in various applications, including image recognition, natural language processing, and speech recognition.

Basic Python Tutorial

Mr. Hasan Mahmud
Senior System Analyst (Com & GIS), BARC
Email: hasan.mahmud@barc.gov.bd

Python is a high-level, interpreted, general-purpose programming language known for its simplicity, readability, and versatility. It was created by Guido van Rossum and first released in 1991. Python supports multiple programming paradigms, including procedural, object-oriented, and functional programming.

Python is often described as a programming language that is:

- Beginner-friendly
- Open-source and free
- Readable and clean in syntax
- Extremely versatile

Python Basics

A variable is a name that refers to a value stored in memory. Think of it as a labeled box that holds information you want to use later.

A data type in Python (or any programming language) defines what kind of value a variable can hold and what operations can be performed on that value.

Variable Assignment In Python, variables are created the moment you assign a value using the = (assignment) operator.

Data types and Assignments

Data Type	Description	Example
int	Integer numbers	`x = 10`
float	Floating-point (decimal) numbers	`pi = 3.14`
bool	Boolean values	`flag = True`
str	Text (string)	`name = "Alice"`
list	Ordered, mutable collection	`nums = [1, 2, 3]`
tuple	Ordered, immutable collection	`coords = (10.0, 20.0)`
dictionary	Unordered key-value collection	`{"wheat": 30, "rice": 40, "maize": 35}`
range	Sequence of numbers	`r = range(5)`

Code (Operators)

```
# Operators
a = 10
b = 3
print("Initial Value: a = ", a)
print("Initial Value: b =", b)
# Arithmetic Operators
print("Arithmetic Operators:")
print("a + b =", a + b)
print("a - b =", a - b)
print("a * b =", a * b)
print("a / b =", a / b)
print("a // b =", a // b)
print("a % b =", a % b)
print("a ** b =", a ** b)

# Comparison Operators
print("\nComparison Operators:")
print("a == b:", a == b)
print("a != b:", a != b)
print("a < b:", a < b)
print("a > b:", a > b)
print("a <= b:", a <= b)
print("a >= b:", a >= b)

# Assignment Operators
print("\nAssignment Operators:")
x = 5
print("\nInitial Value: x = ", x)
x += 2
print("x += 2:", x)
x -= 1
```

```
print("x -= 1:", x)
x *= 3
print("x *= 3:", x)
x /= 2
print("x /= 2:", x)
x %= 4
print("x %= 4:", x)

# Logical Operators
print("\nLogical Operators:")
x = True
y = False
print(f"\nInitial Value: x = {x} and y = {y}")
print("x and y:", x and y)
print("x or y:", x or y)
print("not x:", not x)
```

Code Output (Operators)

```
Initial Value: a =  10
Initial Value: b = 3
Arithmetic Operators:
a + b = 13
a - b = 7
a * b = 30
a / b = 3.3333333333333335
a // b = 3
a % b = 1
a ** b = 1000

Comparison Operators:
a == b: False
a != b: True
```

a < b: False

a > b: True

a <= b: False

a >= b: True

Assignment Operators:

Initial Value: x = 5

x += 2: 7

x -= 1: 6

x *= 3: 18

x /= 2: 9.0

x %= 4: 1.0

Logical Operators:

Initial Value: x = True and y = False

x and y: False

x or y: True

not x: False

Why Casting is Needed (Code)

```
# Cannot Add !!
x = "123"
y = x+1   # ✕ Error (Quiz-Why?)
```

```
TypeError                                 Traceback (most recent call last)
<ipython-input-2-4504a196927f> in <cell line: 0>()
      1 # Cannot Add !!
      2 x = "123"
----> 3 y = x+1   # ✕ Error (Quiz-Why?)
```

TypeError: can only concatenate str (not "int") to str

Casting (Code)

```
# Casting (2 possible scenarios of addition whichever we want)
x = "123"
y1 = int(x)+1
print(f"Type of y1: {type(y1)}, value of y1: {y1}")
y2 = x+"1"
print(f"Type of y2: {type(y2)}, value of y2: {y2}")
```

Casting (Output)

```
Type of y1: <class 'int'>, value of y1: 124
Type of y2: <class 'str'>, value of y2: 1231
```

Rules for Naming Variables

1. Must start with a letter (a–z, A–Z) or an underscore (_).
2. Can contain letters, numbers, and underscores.
3. Case-sensitive (Name and name are different).
4. Cannot use Python keywords (like if, while, class, etc.).

Keywords are reserved words that have special meaning. You cannot use them as variable names, function names, or identifiers. They are used to define the syntax and structure of Python code.

and as assert async await break class continue def del elif else except finally for from global if import in is lambda None nonlocal not or pass raise return try while with yield TRUE FALSE

Code

```
# Valid Names:
user_name = "Bob"
_user2 = 45

# Invalid Names: ✕ Error (Quiz-Why?)
1user = "Alice"   # Cannot start with a digit
class = "Math"    # Cannot use keywords
```

Code (Data Structure)

```
# 1. List: Creating, Indexing, Slicing
crops = ["wheat", "rice", "maize", "jute"]
print("All crops:", crops)
print("First crop:", crops[0])              # Indexing
print("Middle crops:", crops[1:3])      # Slicing

# 2. List Methods: append(), remove(), sort()
crops.append("mustard")                 # Add crop
print("\nUpdated Crops after adding Mustard:", crops)
crops.remove("jute")                      # Remove crop
print("\nUpdated Crops after removing Jute:", crops)
crops.sort()                                 # Sort crops
print("\nUpdated Crops after sorting:", crops)

# 3. Tuple: Basics and Immutability
crop_info = ("wheat", 20, 30)           # (crop, ideal_temp, ideal_moisture)
print("\nCrop Info (Tuple):", crop_info)

#crop_info[0] = "rice"  # ✕ Error (Quiz-Why?)

# 4. Dictionary: Key-Value Pairs
yield_data = {
    "wheat": 2800,
    "rice": 3500,
    "maize": 3200
}
print("Rice yield:", yield_data["rice"])  # Accessing value
yield_data["barley"] = 2700              # Adding new key-value pair
print("Yield Data (Dict):", yield_data)
```

```
# 5. Set: Unique Collections
pests = {"aphid", "armyworm", "mite"}
pests.add("cricket")
pests.add("aphid")  # Duplicate will not be added
print("Pests (Set):", pests)
```

Code Output (Data Structure)

```
All crops: ['wheat', 'rice', 'maize', 'jute']
First crop: wheat
Middle crops: ['rice', 'maize']
Updated Crops after adding Mustard: ['wheat', 'rice', 'maize', 'jute', 'mustard']
Updated Crops after removing Jute: ['wheat', 'rice', 'maize', 'mustard']
Updated Crops after sorting: ['maize', 'mustard', 'rice', 'wheat']
Crop Info (Tuple): ('wheat', 20, 30)
Rice yield: 3500
Yield Data (Dict): {'wheat': 2800, 'rice': 3500, 'maize': 3200, 'barley': 2700}
Pests (Set): {'cricket', 'aphid', 'armyworm', 'mite'}
```

Code Block & Indentation

- Statements don't require semicolons (;) or curly braces ({}).
- Instead of braces {}, Python uses whitespace indentation to separate code blocks.
- Colons (:) are used to start code blocks (e.g., if, for, def).
- Indentation is mandatory and is used to define blocks of code.
- Standard Practice: Use 2-4 spaces per indentation level (preferred). Do not mix tabs and spaces.

Comments

1. Single-line comments

   Start with a # symbol.

   Anything after # on that line is ignored by Python.

   Typically used to explain a single line or step.

2. Multiple-line comments

   Use triple quotes (''' or """) for multi-line documentation.

   Often used for documenting functions, classes, or modules.

Code (A simple function calling)

```
def greet(name):
    '''Define a simple function to
    illustrate code block concept'''

    print("Hello, " + name + "!")

# Create a variable to store the user's name
user_name = input("What is your name? ")

# Call the function with the user's name
greet(user_name)
```

Code Output (A simple function calling)

```
What is your name? Sumon
Hello, Sumon!
```

Control Statements (If, Elif, Else)

The condition must be a boolean expression (something that evaluates to True or False).

The code block under if is indented (usually 2-4 spaces).

We can have multiple elif blocks.

Only the first True condition's block is executed.

Structure

```
if condition1:
    # code to run if condition1 is True
elif condition2:
    # code to run if condition2 is True
else:
    # code to run if none of the above are True
```

Code (Control Flow)

```
print("Agricultural Decision Support:\n")
soil_moisture = int(input("What is your soil moisture percentage (0-100): "))

# Using if, elif, else with comparison and logical operators
if soil_moisture < 20:
    print("Soil is too dry. Immediate irrigation needed.")
elif 20 <= soil_moisture <= 40:
    print("Soil moisture is moderate.")
else:
    print("Soil moisture is sufficient. No irrigation needed now.")
```

Code Output (Control Flow)

```
Agricultural Decision Support:

What is your soil moisture percentage (0-100): 25
Soil moisture is moderate.
```

Function

A function is a reusable block of code that performs a specific task.

Characteristics

Starts with def followed by the function name.

The function body must be indented.

Function names should follow following naming rules:

- Lowercase
- Words separated by underscores (snake_case)
- Should not start with a number

Parameters are optional (can be zero or more).

Use return to send a result back (optional).

Can return multiple values as tuple

Default value for parameter can be set

Structure

```
def function_name(parameters):
  #code block
  return result
```

Code (Function)

```
def irrigation_advice(soil_moisture, crop_type='rice'):
    advice =""
    # Irrigation decision
    if soil_moisture < 20:
        advice += "Soil is too dry. Immediate irrigation needed."
    elif 20 <= soil_moisture <= 40:
        advice += "Soil moisture is moderate."
        # Nested condition: decision based on crop type
        if crop_type == "rice":
```

```
        advice += "Irrigation is recommended for rice."
    elif crop_type == "wheat":
        advice += "Monitor moisture; irrigation may be needed soon for wheat."
    else:
        advice += "Check crop-specific moisture requirements."
else:
    advice += "Soil moisture is sufficient. No irrigation needed now."

return advice
```

Code (Function Calling)

```
print(irrigation_advice(25, 'rice'))
print(irrigation_advice(25))  # Default Value Demonstration
```

Code Output (Function Calling)

```
Soil moisture is moderate. Irrigation is recommended for rice.
Soil moisture is moderate. Irrigation is recommended for rice.
```

Loop

A **loop** is a logical structure in programming that allows a set of instructions to be executed **repeatedly** until a specific condition is met or no longer valid. It automates repetitive tasks without the need to write the same instructions multiple times.

Importance of Loops

- **Efficiency**
  Loops let you perform repetitive actions without rewriting code, saving time and reducing clutter.
- **Automation**
  In real-world tasks like processing multiple files, calculating statistics, or managing sensors in agriculture, loops help automate work that involves repetition.
- **Flexibility**
  You can easily adjust how many times an action happens, based on data or changing conditions.
- **Scalability**
  Loops allow your program to handle larger datasets or longer tasks without extra effort — it works the same whether there are 5 or 5,000 items.

- **Error Reduction**
  Repeating actions through loops helps reduce the chance of human error compared to copying and pasting code multiple times.

- **Logical Flow**
  Loops support logical operations like checking conditions or performing tasks at regular intervals (e.g., monitoring soil moisture every hour).

for Loop

A for loop is used when you know in advance how many times you want to repeat a task. It is typically used to iterate over a sequence like a list, range of numbers, or items in a collection.

Example:

A farmer wants to inspect 10 plants in a row — one by one. Since the number is fixed (10), a for loop is the most suitable.

while Loop

A while loop is used when the number of repetitions is not known beforehand. The loop continues as long as a certain condition remains true.

Example:

A sensor monitors soil moisture and continues irrigation until the moisture level is sufficient. Since the number of times it runs is not fixed, a while loop fits better

Code (for Loop – Simple)

```
farms = [
   ("Farm A", "wheat", 35),
   ("Farm B", "rice", 15),
   ("Farm C", "maize", 45),
   ("Farm D", "wheat", 25),
]
print("Irrigation Decision System\n")
print(f"Type of farms: {type(farms)}")
print(farms)
for farm in farms:
  print(f"\nType of farm: {type(farm)}")
  print(farm)
```

Code Output (for Loop – Simple)

```
Irrigation Decision System

Type of farms: <class 'list'>
[('Farm A', 'wheat', 35), ('Farm B', 'rice', 15), ('Farm C', 'maize', 45), ('Farm D', 'wheat', 25)]

Type of farm: <class 'tuple'>
('Farm A', 'wheat', 35)

Type of farm: <class 'tuple'>
('Farm B', 'rice', 15)

Type of farm: <class 'tuple'>
('Farm C', 'maize', 45)

Type of farm: <class 'tuple'>
('Farm D', 'wheat', 25)
```

Code (for Loop – Advanced)

```
farms = [
    ("Farm A", "wheat", 35),
    ("Farm B", "rice", 15),
    ("Farm C", "maize", 45),
 #   ("Farm D", "rice", -1),  # Error in Data. Check the output by uncommenting this line
    ("Farm E", "wheat", 25),
]

print("Irrigation Decision System\n")
```

```
# Loop through farm data
for farm in farms:
    name, crop, moisture = farm

    # Check for faulty moisture value
    if moisture < 0:
        print(f"Error in data for {name}. Stopping analysis.")
        break  # Stop loop if error in data found

    # Skip farms growing crops not monitored
    if crop not in ["wheat", "rice"]:
        print(f"Skipping {name}: crop '{crop}' not monitored.")
        continue  # Skip this farm

    # Provide irrigation advice
    print(f"\nAnalyzing {name} ({crop}):")
    if moisture < 20:
        print("  ➤ Soil too dry. Irrigation needed.")
    elif moisture <= 40:
        print("  ➤ Moderate moisture. Monitor closely.")
    else:
        print("  ➤ Sufficient moisture. No action needed.")
print("\n Farm loop completed.\n")
```

Code Output (for Loop – Advanced)

```
Irrigation Decision System

Analyzing Farm A (wheat):
  ➤ Moderate moisture. Monitor closely.
```

Analyzing Farm B (rice):

➤ Soil too dry. Irrigation needed.

Skipping Farm C: crop 'maize' not monitored.

Analyzing Farm E (wheat):

➤ Moderate moisture. Monitor closely.

Farm loop completed.

Code (while Loop)

```
soil_moisture = 20  # in percent

while soil_moisture < 30:
    print(f"Soil moisture is {soil_moisture}%. Irrigating...")
    if soil_moisture < 27:
        soil_moisture += 3  # simulated standard irrigation increasing moisture
    else:
        soil_moisture += 1  # simulated less irrigation increasing moisture
print(f"Soil moisture is now {soil_moisture}%. Irrigation complete.")
```

Code Output (while Loop)

Soil moisture is 20%. Irrigating...

Soil moisture is 23%. Irrigating...

Soil moisture is 26%. Irrigating...

Soil moisture is 29%. Irrigating...

Soil moisture is now 30%. Irrigation complete.

Installing and Importing Libraries

A library is a collection of packages and modules.

We install the library (e.g., pandas) using pip, and it may include multiple packages/modules.

To use it in our code, we must first import it.

Concept	Description	Example
Library	Collection of tools	pandas
Package	Organized directory of modules	pandas.io, matplotlib.pyplot
Module	A single .py file	math, datetime

Code (Libraries)

```
!pip install geopandas
import geopandas as gpd
```

Code Output (Libraries)

```
Collecting geopandas
  Downloading geopandas-1.0.1-py3-none-any.whl.metadata (2.2 kB)
Requirement already satisfied: numpy>=1.22 in /usr/local/lib/python3.11/dist-packages (from geopandas) (2.0.2)
……..
```

Code (Read from File and Basic Aggregation using pandas)

```
import pandas as pd

from google.colab import drive
drive.mount('/content/drive')

# Read the agricultural data from CSV previously uploaded to Google Drive
df = pd.read_csv("/content/drive/MyDrive/Training.csv")
# Replace the file-path with your actual file path copied from drive

# Show the first few rows
print("Original Data:")
print(df.head())
```

```
# 1. Total yield per district
total_yield_by_district = df.groupby("district")["reported_yield"].sum().reset_index()
print("\nTotal Yield by district:")
print(total_yield_by_district)

# 2. Average yield per crop
average_yield_by_crop = df.groupby("crop")["reported_yield"].mean().reset_index()
print("\nAverage Yield by Crop:")
print(average_yield_by_crop)

# 3. Maximum yield by crop
max_yield_by_crop = df.groupby("crop")["reported_yield"].max().reset_index()
print("\nMaximum Yield by Crop:")
print(max_yield_by_crop)
```

Code Output (Read from File and Basic Aggregation using pandas)

Original Data:

	Field_ID	year	crop_season	crop	district	cropcut_yield	\
0	1	2020-21	Boro	Rice	Dinajpur	6472.38	
1	2	2020-21	Boro	Rice	Dinajpur	7529.07	
2	3	2020-21	Boro	Rice	Dinajpur	7421.51	
3	4	2020-21	Boro	Rice	Dinajpur	7194.77	
4	5	2020-21	Boro	Rice	Dinajpur	7313.95	

	reported_yield
0	6373.52
1	8646.25
2	8522.73
3	8152.17
4	8399.21

Total Yield by district:

	district	reported_yield
0	Dinajpur	4726693.88
1	Faridpur	6433138.21
2	Jashore	3531942.73
3	Rajshahi	3945320.83
4	Rangpur	4040708.88

Average Yield by Crop:

	crop	reported_yield
0	Rice	6699.665209
1	Wheat	3772.183627

Maximum Yield by Crop:

	crop	reported_yield
0	Rice	9074.78
1	Wheat	5122.17

Code (File Writing)

```
total_yield_by_district.to_csv('/content/drive/MyDrive/output.csv', index=False)


with open('/content/drive/MyDrive/output.csv', 'a') as f:
    f.write('\n\n')

average_yield_by_crop.to_csv('/content/drive/MyDrive/output.csv', mode='a', index=False)
```

Object Oriented Programming (OOP)

1. Class Definition

A class is a blueprint for creating objects.

It defines attributes (variables) and methods (functions) that the objects created from the class will have.

◈ **init** is a constructor method that runs when an object is created.

◈ self refers to the instance of the class.

2. Object Creation

An object is an instance of a class. It contains state (attributes) and behavior (methods).

3. Method Calling

Methods of the class are called using the object.

Code (class)

```
# Class definition
class Person:
    # Constructor method (called when object is created)
    def __init__(self, name, age):
        self.name = name  # instance variable
        self.age = age

    # Instance method
    def introduce(self):
        print(f"Hi, I'm {self.name} and I'm {self.age} years old.")

    # Another method
    def birthday(self):
        self.age += 1
        print(f"Happy Birthday, {self.name}! You are now {self.age}.")
```

Code Output (class)

```
# Object creation
person1 = Person("Alice", 30)
person2 = Person("Bob", 25)
```

```
# Method calling
person1.introduce()    # Output: Hi, I'm Alice and I'm 30 years old.
person2.introduce()    # Output: Hi, I'm Bob and I'm 25 years old.

person1.birthday()     # Output: Happy Birthday, Alice! You are now 31.
```

```
Hi, I'm Alice and I'm 30 years old.
Hi, I'm Bob and I'm 25 years old.
Happy Birthday, Alice! You are now 31.
```

4 Important Concepts of OOP

1. Encapsulation

Bundles data (attributes) and methods that operate on the data into one unit (a class).

Protects internal object state by using access modifiers (private, public, etc. in some languages).

2. Inheritance

A class (child) can inherit attributes and methods from another class (parent).

Promotes code reusability.

3. Polymorphism

Allows methods to have different behaviors based on the object that is calling them.

Two types: method overriding (in child class) and method overloading (same method name with different parameters, depending on the language).

4. Abstraction

Hides complex implementation details and shows only the essential features of the object.

Often achieved using abstract classes or interfaces (in languages like Java, C++).

Code (OOP Concepts)

```
from abc import ABC, abstractmethod

# Abstraction + Encapsulation
class Crop(ABC):
    def __init__(self, name, water_required):
        self.__name = name                  # Encapsulated (private)
        self.__water_required = water_required

    def get_name(self):                     # Public getter
        return self.__name

    def get_water_required(self):
        return self.__water_required

    @abstractmethod
    def grow(self):
        pass

# Inheritance
class Wheat(Crop):
    def grow(self):                         # 3. Polymorphism (Overriding)
        print(f"{self.get_name()} needs {self.get_water_required()}L of water to grow. Growing wheat...")

class Rice(Crop):
    def grow(self):
        print(f"{self.get_name()} needs {self.get_water_required()}L of water to grow. Growing rice...")

# Polymorphism in action
def start_farming(crop: Crop):
    crop.grow()
```

```
#  Object creation
wheat = Wheat("Wheat", 50)
rice = Rice("Rice", 80)

#  Using polymorphism
start_farming(wheat)
start_farming(rice)
```

Code Output (OOP Concepts)

```
Wheat needs 50L of water to grow. Growing wheat...
Rice needs 80L of water to grow. Growing rice...
```

Python for Data Analysis – Beginner's Guide with Detailed Explanations

Mr. Al-Helal
Programmer (Com & GIS)
Bangladesh Agricultural Research Council
Email: al.helal@barc.gov.bd

- **Loops (for, while)**

Loops are essential in programming to perform repetitive tasks efficiently. Instead of writing the same code multiple times, you use loops to execute it repeatedly.

- For loop: Best used when the number of repetitions is known ahead of time. It iterates over a sequence like a list, range, or string.
- While loop: Runs as long as a given condition is true. Useful when the number of iterations depends on a condition rather than a fixed number.

Common use cases:
- Processing every item in a dataset.
- Repeating tasks until a certain state is reached.
- Automating repetitive actions in data manipulation.

Example:
```
# For loop
for i in range(1, 6):
    print(f"{i} squared is {i*i}")

# While loop
x = 1
while x <= 5:
    print("Counting:", x)
    x += 1
```

- **Functions**

Functions help organize code into reusable blocks. Instead of writing the same code multiple times, you can define a function once and call it whenever needed. Functions can accept inputs (parameters) and return outputs.

Benefits:
- Code reuse
- Improved readability

- Easier debugging and maintenance

Example:

```
def greet(name):
    return f"Hello, {name}!"

print(greet("Al-Helal"))
```

- **Data Structures**

Python provides various built-in data structures to store and organize data:

- List: Ordered, mutable collection (e.g., a list of fruits).
- Dictionary: Key-value pairs for fast lookup (e.g., person details).
- Tuple: Ordered but immutable collection (fixed data).
- Set: Unordered collection of unique items (no duplicates).

These structures are the foundation for data manipulation in Python.

Example:

```
# List
fruits = ["apple", "banana", "cherry"]
fruits.append("orange")

# Dictionary
person = {"name": "Helal", "age": 35}
print(person.get("name"))

# Tuple
coordinates = (10, 20)

# Set
unique_numbers = set([1, 2, 2, 3, 3])
print(unique_numbers)
```

- **File Handling**

Working with files is crucial for reading data stored in text or CSV files and writing outputs or logs.

- Writing: You can create or overwrite files with text data.
- Reading: You can open existing files and extract information.

This is especially important in data analysis when working with datasets saved in files.

Example:
```
# Writing to file
with open("data.txt", "w") as f:
    f.write("Python is fun.")

# Reading from file
with open("data.txt", "r") as f:
    print(f.read())
```

5. NumPy

NumPy is a fundamental library for numerical computing in Python. It offers:

- Efficient arrays for large datasets.
- Mathematical functions for statistics and algebra.
- Operations on multi-dimensional arrays.

NumPy is the base for many other scientific libraries.

Example:
```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])
print("Mean:", np.mean(arr))
print("Standard Deviation:", np.std(arr))

# 2D array
matrix = np.array([[1, 2], [3, 4]])
print("Matrix:\n", matrix)
```

6. Pandas

Pandas is a powerful library for handling tabular data (like Excel or CSV files). It provides:

- DataFrames: 2D labeled data structures with columns.
- Easy filtering, grouping, and aggregation.
- Functions to clean and manipulate data.

It's essential for real-world data analysis.

Example:

```
import pandas as pd

df = pd.DataFrame({
   'Name': ['Alice', 'Bob', 'Cathy'],
   'Age': [25, 30, 22]
})

# View first rows
print(df.head())

# Filter rows where Age > 23
print(df[df['Age'] > 23])
```

7. Visualization

Visualization helps us understand data better by showing patterns and trends.

- Matplotlib: Core plotting library for charts.
- Seaborn: Built on Matplotlib, offers attractive statistical plots.

Common plots: line charts, histograms, scatterplots.

Example:

```
import matplotlib.pyplot as plt
import seaborn as sns

# Line chart
plt.plot([1, 2, 3], [2, 4, 6])
plt.title("Simple Line Chart")
plt.xlabel("X")
plt.ylabel("Y")
plt.show()

# Histogram
data = [10, 20, 20, 30, 30, 30]
sns.histplot(data, bins=3)
plt.title("Histogram")
plt.show()
```

8. Data Cleaning

Real-world data is often messy with missing or incorrect values. Data cleaning is the process of fixing or removing these issues before analysis.

Common tasks:
- Removing rows with missing values.
- Filling missing values with defaults or averages.
- Renaming columns for clarity.

Example:
```
data = {'Name': ['A', 'B', None], 'Age': [25, None, 30]}
df = pd.DataFrame(data)

# Drop rows with missing values
print(df.dropna())

# Fill missing with 0
df_filled = df.fillna(0)
print(df_filled)

# Rename column
df.rename(columns={'Name': 'Full Name'}, inplace=True)
```

9. Basic Statistics

Statistics help summarize and understand data. Pandas provides easy functions for common stats like max, min, mean, and descriptive summaries.
Example:
```
print("Max Age:", df['Age'].max())
print("Min Age:", df['Age'].min())
print("Mean Age:", df['Age'].mean())
print("Summary:\n", df.describe())
```

11.Basic Machine Learning

Machine learning enables computers to learn patterns from data and make predictions. Scikit-learn is a popular Python library for machine learning.

Steps:
- Load a dataset.
- Split into training and testing parts.
- Train a model (e.g., Random Forest).

- Predict on test data.
- Evaluate results.

Example:

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report

iris = load_iris()
X = iris.data
y = iris.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

model = RandomForestClassifier()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred)
```

Exploratory Data Analysis (EDA): RMSE, Accuracy, Confusion Matrix

Mr. Al-Helal
Programmer (Com & GIS)
Bangladesh Agricultural Research Council
Email: al.helal@barc.gov.bd

Understanding the Confusion Matrix in Machine Learning

Machine learning models are increasingly used in various applications to classify data into different categories. However evaluating the performance of these models is crucial to ensure their accuracy and reliability. One essential tool in this evaluation process is the confusion matrix. In this article we will work on confusion matrix, its significance in machine learning and how it can be used to improve the performance of classification models.

Understanding Confusion Matrix

A **confusion matrix** is a simple table that shows how well a classification model is performing by comparing its predictions to the actual results. It breaks down the predictions into four categories: correct predictions for both classes (**true positives** and **true negatives**) and incorrect predictions (**false positives** and **false negatives**). This helps you understand where the model is making mistakes, so you can improve it.

The matrix displays the number of instances produced by the model on the test data.

- **True Positive (TP):** The model correctly predicted a positive outcome (the actual outcome was positive).
- **True Negative (TN):** The model correctly predicted a negative outcome (the actual outcome was negative).
- **False Positive (FP):** The model incorrectly predicted a positive outcome (the actual outcome was negative). Also known as a Type I error.
- **False Negative (FN):** The model incorrectly predicted a negative outcome (the actual outcome was positive). Also known as a Type II error.

A **confusion matrix** helps you see how well a model is working by showing correct and incorrect predictions. It also helps calculate key measures like **accuracy**, **precision**, and **recall**, which give a better idea of performance, especially when the data is imbalanced.

Metrics based on Confusion Matrix Data

1. Accuracy

Accuracy measures how often the model's predictions are correct overall. It gives a general idea of how well the model is performing. However, accuracy can be misleading, especially with imbalanced datasets where one class dominates. For example, a model that predicts the majority class correctly most of the time might have high accuracy but still fail to capture important details about other classes.

$$\textbf{Accuracy=TP+TN/TP+TN+FP+FN}$$

2. Precision

Precision focuses on the quality of the model's positive predictions. It tells us how many of the instances predicted as positive are actually positive. Precision is important in situations where false positives need to be minimized, such as detecting spam emails or fraud.

$$\textbf{Precision=TP/TP+FP}$$

3. Recall

Recall measures how well the model identifies all actual positive cases. It shows the proportion of true positives detected out of all the actual positive instances. High recall is essential when missing positive cases has significant consequences, such as in medical diagnoses.

$$\textbf{Recall=TP/TP+FN}$$

4. F1-Score

F1-score combines precision and recall into a single metric to balance their trade-off. It provides a better sense of a model's overall performance, particularly for imbalanced datasets. The F1 score is helpful when both false positives and false negatives are important, though it assumes precision and recall are equally significant, which might not always align with the use case.

$$\textbf{F1-Score=2×Precision×Recall / Precision+Recall}$$

5. Specificity

Specificity is another important metric in the evaluation of classification models, particularly in binary classification. It measures the ability of a model to correctly identify negative instances. Specificity is also known as the True Negative Rate. Formula is given by:

Specificity=TN/TN+FP

6. Type 1 and Type 2 error

- **Type 1 error**
- A **Type 1 Error** occurs when the model incorrectly predicts a positive instance, but the actual instance is negative. This is also known as a **false positive**. Type 1 Errors affect the **precision** of a model, which measures the accuracy of positive predictions.

$$\textbf{Type 1 Error=FP/FP+TN}$$

- **Type 2 error**

  A **Type 2 Error** occurs when the model fails to predict a positive instance, even though it is actually positive. This is also known as a **false negative**. Type 2 Errors impact the **recall** of a model, which measures how well the model identifies all actual positive cases.

$$\textbf{Type 2 Error=FN/TP+FN}$$

Example: A diagnostic test is used to detect a particular disease in patients.

- **Type 1 Error (False Positive):** This occurs when the test predicts a patient has the disease (positive result), but the patient is actually healthy (negative case).
- **Type 2 Error (False Negative):** This occurs when the test predicts the patient is healthy (negative result), but the patient actually has the disease (positive case).

Confusion Matrix For binary classification

A 2X2 Confusion matrix is shown below for the image recognition having a Dog image or Not Dog image:

	Predicted	**Predicted**
Actual	True Positive (TP)	False Negative (FN)
Actual	False Positive (FP)	True Negative (TN)

- **True Positive (TP):** It is the total counts having both predicted and actual values are Dog.
- **True Negative (TN):** It is the total counts having both predicted and actual values are Not Dog.
- **False Positive (FP):** It is the total counts having prediction is Dog while actually Not Dog.
- **False Negative (FN):** It is the total counts having prediction is Not Dog while actually, it is Dog.

Example: Confusion Matrix for Dog Image Recognition with Numbers

Index	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Actual	Dog	Dog	Dog	Not Dog	Dog	Not Dog	Dog	Dog	Not Dog	Not Dog
Predicted	Dog	Not Dog	Dog	Not Dog	Dog	Dog	Dog	Dog	Not Dog	Not Dog
Result	TP	FN	TP	TN	TP	FP	TP	TP	TN	TN

- Actual Dog Counts = 6
- Actual Not Dog Counts = 4
- True Positive Counts = 5
- False Positive Counts = 1
- True Negative Counts = 3
- False Negative Counts = 1

		Predicted	
		Dog	**Not Dog**
Actual	**Dog**	True Positive (TP =5)	False Negative (FN =1)
	Not Dog	False Positive (FP=1)	True Negative (TN=3)

Implementation of Confusion Matrix for Binary classification using Python

Step 1: Import the necessary libraries

```
import numpy as np
from sklearn.metrics import confusion_matrix,classification_report
import seaborn as sns
import matplotlib.pyplot as plt
```

Step 2: Create the NumPy array for actual and predicted labels

- **actual:** represents the true labels or the actual classification of the items. In this case, it's a list of 10 items, where each entry is either 'Dog' or 'Not Dog'.
- **predicted:** represents the predicted labels or the classification made by the model.

```
actual    = np.array(
 ['Dog','Dog','Dog','Not Dog','Dog','Not Dog','Dog','Dog','Not Dog','Not Dog'])
predicted = np.array(
 ['Dog','Not Dog','Dog','Not Dog','Dog','Dog','Dog','Dog','Not Dog','Not Dog'])
```

Step 3: Compute the confusion matrix

- **confusion_matrix:** This function from **sklearn.metrics** computes the confusion matrix, which is a table used to evaluate the performance of a classification algorithm.
- It compares actual and predicted to generate a matrix

```
cm = confusion_matrix(actual,predicted)
```

Step 4: Plot the confusion matrix with the help of the seaborn heatmap

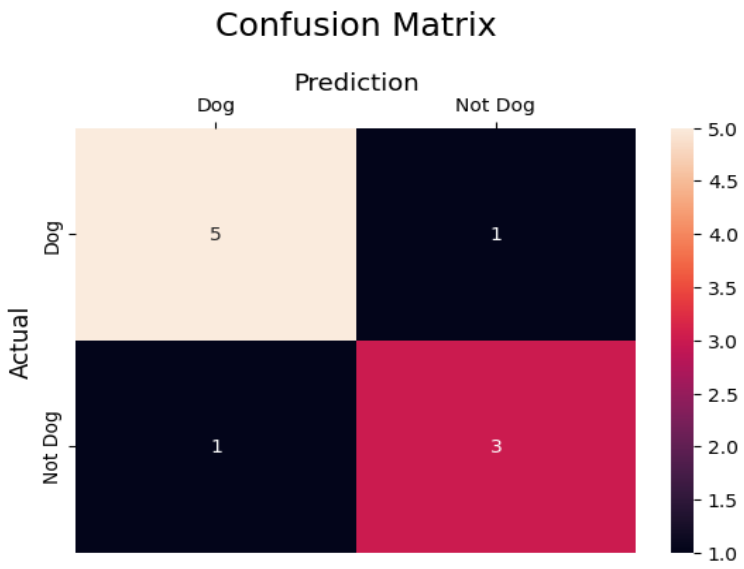
- **sns.heatmap:** This function from **Seaborn** is used to create a heatmap of the confusion matrix.
- **annot=True:** Displays the numerical values in each cell of the heatmap.

```
sns.heatmap(cm,
```

```
        annot=True,
        fmt='g',
        xticklabels=['Dog','Not Dog'],
        yticklabels=['Dog','Not Dog'])
plt.ylabel('Actual', fontsize=13)
plt.title('Confusion Matrix', fontsize=17, pad=20)
plt.gca().xaxis.set_label_position('top')
plt.xlabel('Prediction', fontsize=13)
plt.gca().xaxis.tick_top()

plt.gca().figure.subplots_adjust(bottom=0.2)
plt.gca().figure.text(0.5, 0.05, 'Prediction', ha='center', fontsize=13)
plt.show()
```

Output:



Visualizing the Confusion Matrix

Step 5: Classifications Report based on Confusion Metrics

```
print(classification_report(actual, predicted))
```

Output:

	precision	***recall***	***f1-score***	***support***
Dog	*0.83*	*0.83*	*0.83*	*6*
Not Dog	*0.75*	*0.75*	*0.75*	*4*
accuracy	*0.80*	*10*		
macro avg	*0.79*	*0.79*	*0.79*	*10*
weighted avg	*0.80*	*0.80*	*0.80*	*10*

Confusion Matrix For Multi-class Classification

In **multi-class classification** the confusion matrix is expanded to account for multiple classes.

- **Rows** represent the actual classes (ground truth).
- **Columns** represent the predicted classes.
- Each cell in the matrix shows how often a specific actual class was predicted as another class.

For example, in a 3-class problem, the confusion matrix would be a 3×3 table, where each row and column corresponds to one of the classes. It summarizes the model's performance across all classes in a compact format.

Lets consider that example:

Example: Confusion Matrix for Image Classification (Cat, Dog, Horse)

	Predicted Cat	**Predicted Dog**	**Predicted Horse**
Actual Cat	True Positive (TP)	False Negative (FN)	False Negative (FN)
Actual Dog	False Negative (FN)	True Positive (TP)	False Negative (FN)
Actual Horse	False Negative (FN)	False Negative (FN)	True Positive (TP)

- The definitions of all the terms (TP, TN, FP and FN) are the same as described in the previous example.

Example with Numbers:

Let's consider the scenario where the model processed 30 images:

	Predicted Cat	**Predicted Dog**	**Predicted Horse**
Actual Cat	8	1	1
Actual Dog	2	10	0
Actual Horse	0	2	8

In this scenario:

- **Cats:** 8 were correctly identified, 1 was misidentified as a dog, and 1 was misidentified as a horse.

- **Dogs:** 10 were correctly identified, 2 were misidentified as cats.
- **Horses:** 8 were correctly identified, 2 were misidentified as dogs.

To calculate true negatives, we need to know the total number of images that were NOT cats, dogs, or horses. Let's assume there were 10 such images, and the model correctly classified all of them as "not cat," "not dog," and "not horse."
Therefore:

- **True Negative (TN) Counts:** 10 (for each class, as the model correctly identified each non-cat/dog/horse image as not belonging to that class)

Implementation of Confusion Matrix for Multi-Class classification using Python

Step 1: Import the necessary libraries

```
import numpy as np
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay,
classification_report
import matplotlib.pyplot as plt
```

Step 2: Create the NumPy array for actual and predicted labels

- **y_true:** List of true labels.
- **y_pred:** List of predicted labels by the model.
- **classes:** A list of class names: 'Cat', 'Dog' and 'Horse'.

```
y_true = ['Cat'] * 10 + ['Dog'] * 12 + ['Horse'] * 10
y_pred = ['Cat'] * 8 + ['Dog'] + ['Horse'] + ['Cat'] * 2 + ['Dog'] * 10 + ['Horse'] * 8 + ['Dog'] *
2
classes = ['Cat', 'Dog', 'Horse']
```

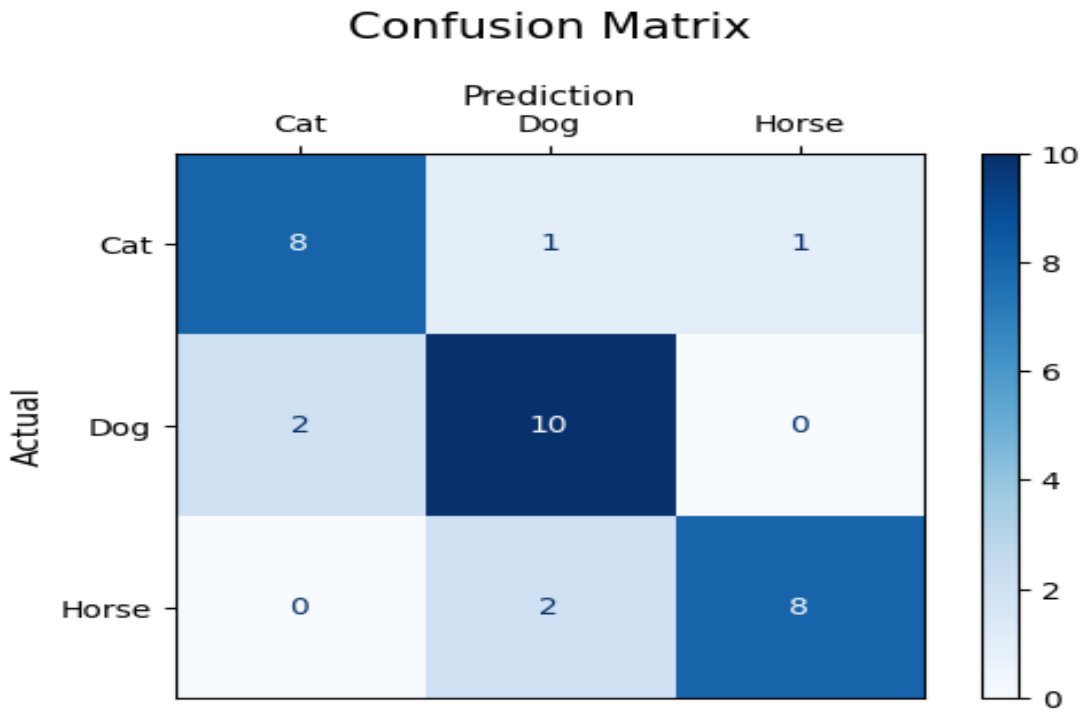
Step 3: Generate and Visualize the Confusion Matrix

- **ConfusionMatrixDisplay:** Creates a display object for the confusion matrix.
- **confusion_matrix=cm:** Passes the confusion matrix (cm) to display.
- **display_labels=classes:** Sets the labels (['Cat', 'Dog', 'Horse']) for the confusion matrix.

```
cm = confusion_matrix(y_true, y_pred, labels=classes)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=classes)
disp.plot(cmap=plt.cm.Blues)
plt.title('Confusion Matrix', fontsize=15, pad=20)
plt.xlabel('Prediction', fontsize=11)
plt.ylabel('Actual', fontsize=11)
plt.gca().xaxis.set_label_position('top')
plt.gca().xaxis.tick_top()
plt.gca().figure.subplots_adjust(bottom=0.2)
plt.gca().figure.text(0.5, 0.05, 'Prediction', ha='center', fontsize=13)

plt.show()
```

Output:



Display the confusion matrix

Step 4: Print the Classification Report

```
print (classification_report(y_true, y_pred, target_names=classes))
```

Output:

	Precision	***recall***	***f1-score***	***support***
Cat	*0.80*	*0.80*	*0.80*	*10*
Dog	*0.77*	*0.83*	*0.80*	*12*
Horse	*0.89*	*0.80*	*0.84*	*10*
Accuracy	*0.81*	*32*		
macro avg	*0.82*	*0.81*	*0.81*	*32*
weighted avg	*0.82*	*0.81*	*0.81*	*32*

Confusion matrix is a valuable tool for evaluating how well a classification model works. It provides clear insights into important metrics like accuracy, precision, and recall by analyzing correct and incorrect predictions (true positives, true negatives, false positives, and false negatives). This article explained these metrics with examples and showed how to create confusion matrices in Python for both binary and multi-class classification. By understanding and using these metrics, practitioners can make better decisions about model performance, especially when dealing with imbalanced datasets.

Supervised Learning – Classification Algorithms

Mr. Aditya Rajbongshi
Lecturer, Gazipur Digital University, Bangladesh

Image Classification using Machine Learning Classifier

In a society where digital images have become prominent in our daily lives, the volumes of data it has generated over the decades are massive. Computer Vision is a field of AI which uses a lot of data, mainly for image detection, recognition, and classification. Image Classification uses Machine Learning algorithms to analyze the presence of items in a picture and to categorize the picture. This particular task forms the basis of Computer Vision and Image Recognition. Machines don't analyze a picture as a whole. They only analyze a picture through pixel patterns or vectors. They will then categorize and assign labels to the elements they detect and classify them depending on the different rules that have been set up when configuring the algorithm. Classifiers' task is to take an input (a photograph for example) and to output a class label (extract the image features and predict the category from them).

The following steps are used to classify the image using machine learning classifier:

- Image acquisition
- Image preprocessing
- Feature extraction
- Splitting the featured dataset
- Train and Test the model
- Model evaluation

The description of those steps allow with the implementation code is presented in below:

- **Image acquisition**

Image acquisition for **agriculture image classification** involves collecting visual data from crops, soil, fields, or agricultural products to be used in machine learning models for tasks such as:

- **Crop classification**
- **Weed detection**
- **Disease identification**
- **Yield estimation**
- **Soil analysis**

- **Image preprocessing**

Capture RGB image with a smartphone, resize to 224×224, apply CLAHE for contrast enhancement, convert to HSV color space, normalize pixel values, and save preprocessed image for training/inference.

- **Feature extraction**

Feature extraction is the process of identifying and quantifying meaningful information from images to use in classification tasks. For agricultural image classification (e.g., plant disease detection, soil type classification), texture and statistical features are especially powerful. After extracting relevant features from agricultural images using Gray Level Co-occurrence Matrix (GLCM) and statistical, the resulting feature vectors are organized into a dataset for machine learning. Each feature vector represents an individual image, and includes descriptors such as contrast, correlation, energy, homogeneity (from GLCM), along with mean, standard deviation, skewness, kurtosis, and entropy (from statistical analysis).

- **Splitting the featured dataset**

To evaluate the performance of classification algorithms, the dataset is split into two subsets: training and testing. The training set is used to train the model to learn patterns in the data, while the testing set is used to evaluate its performance on unseen examples. A typical split ratio is 80:20, where 80% of the data is used for training and 20% for testing. This ensures that the model is trained on a sufficiently large portion of the data while retaining enough samples for a meaningful evaluation. The splitting process is often performed using functions such as train_test_split() from the scikit-learn library to ensure randomness and class balance.

- **Train and Test the model**

Once the features are extracted from agricultural images using GLCM and statistical methods, the dataset is divided into training and testing sets, typically in an 80:20 ratio. This ensures that the model is trained on a majority of the data while being evaluated on unseen examples to assess its generalization capability.

The training phase involves feeding the training feature vectors and their corresponding class labels into a machine learning algorithm, such as Support Vector Machine (SVM), Random Forest, or K-Nearest Neighbors (KNN). During this phase, the model learns the relationships and patterns within the data, adjusting its internal parameters to minimize classification errors. Following training, the model enters the testing phase, where it is applied to the feature vectors in the testing set.

- **Model evaluation**

The predicted labels are compared to the actual labels to evaluate the model's performance. Common evaluation metrics include accuracy, precision, recall, and F1-score.

Implementation

Feature Extraction:

The collected image are resized into 300×300 pixels. The mapping of color intensity is applied for enhancing the contrast of images that presented in figure 1. Then the color images are segmented into various clusters. This segmentation is performed by using k-means clustering which is shown in figure 2. K-means clustering outperforms other algorithms which are utilized for segmentation. We extracted the feature vectors from each cluster that is shown in figure 3. After completion of segmentation, we have extracted feature vectors from each segmented image that have been utilized for the training of the classifiers.

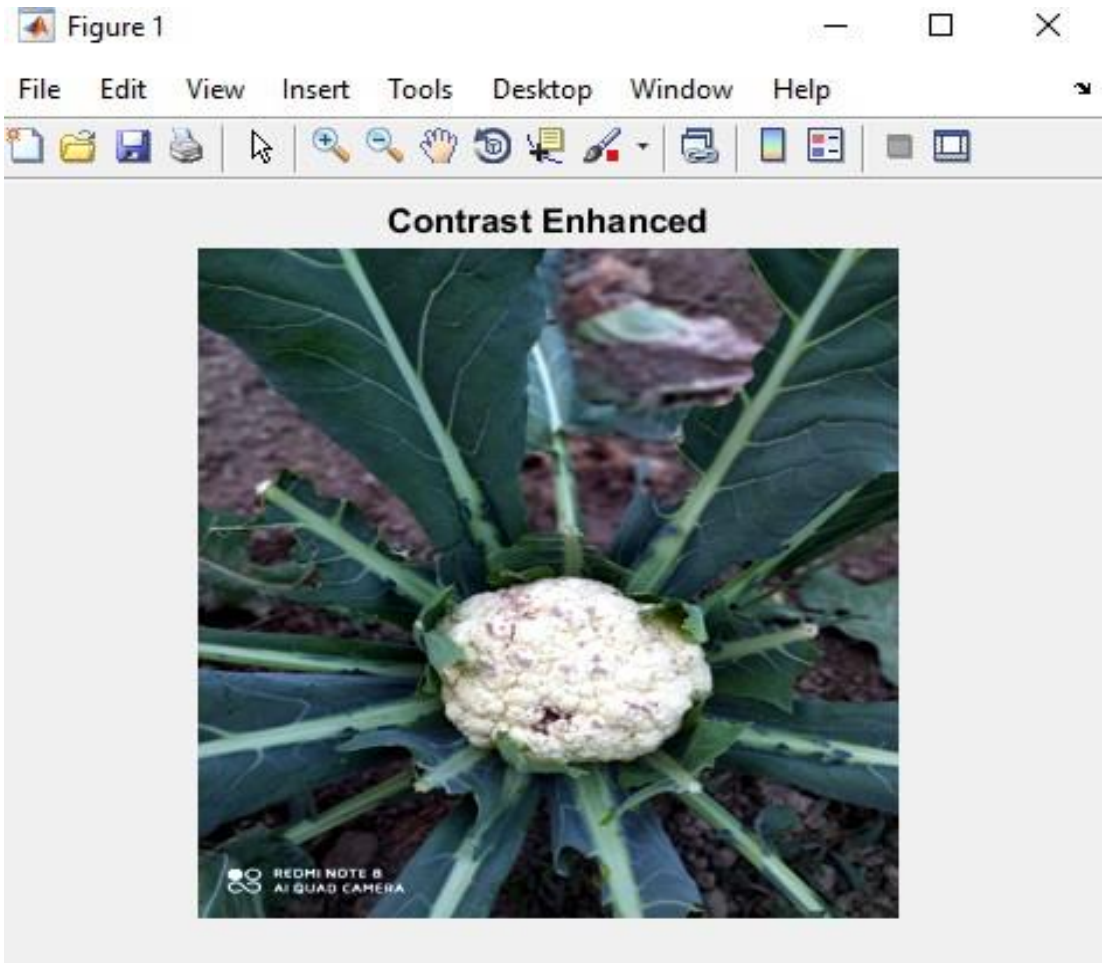


Fig 1: Enhancing the contrast of original image.

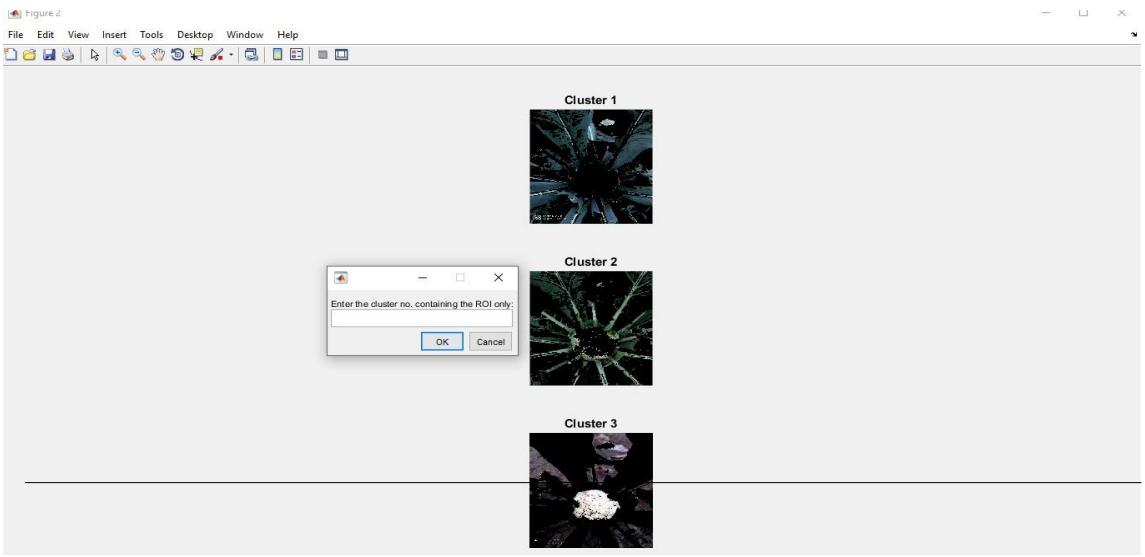


Fig 2: Segment the image into 3 clusters.

Editor - Detect.m | Variables - value

value

419x13 double

	1	2	3	4	5	6	7	8	9	10	11	12	13
1	0.4023	0.7919	0.4033	0.9310	25.8317	39.1192	3.6357	9.6760	1.3728e+03	1.0000	5.3440	1.5877	255
2	0.2744	0.9636	0.5564	0.9429	32.7589	67.4427	3.4784	8.8274	3.3763e+03	1.0000	6.7349	2.2486	255
3	0.4580	0.7985	0.5618	0.9292	18.3365	38.1094	2.6140	7.6836	1.3676e+03	1.0000	7.0116	2.1828	255
4	0.4934	0.9376	0.5347	0.9444	34.6171	69.0232	3.4984	8.9160	4.2392e+03	1.0000	6.1834	2.1185	255
5	0.6628	0.7752	0.5037	0.9250	22.1792	42.9567	2.9226	8.3167	1.7791e+03	1.0000	6.4549	2.0379	255
6	0.4297	0.8482	0.4234	0.9345	27.8682	45.7003	3.5827	9.4462	1.9214e+03	1.0000	5.5294	1.7517	255
7	0.8169	0.8701	0.3838	0.9143	40.7707	63.6143	3.9887	9.5480	3.5214e+03	1.0000	5.2545	1.6847	255
8	0.9335	0.6920	0.5619	0.9190	19.4684	43.3438	2.4698	7.4079	1.7822e+03	1.0000	8.8310	2.4861	255
9	0.6725	0.8039	0.4025	0.9165	31.1311	49.5148	3.6091	9.3669	2.2180e+03	1.0000	4.7323	1.5749	255
10	0.9311	0.9427	0.3368	0.9027	77.8256	98.2970	4.3988	9.8423	5.7397e+03	1.0000	1.6259	0.6484	255
11	0.8956	0.6026	0.6391	0.9428	15.3875	38.2821	2.6216	7.4557	1.4087e+03	1.0000	18.6043	3.7071	255
12	0.2668	0.8460	0.5379	0.9519	17.8507	33.8677	2.7428	7.9255	1.0227e+03	1.0000	6.2935	1.9602	255

Fig 3: Extracted feature vectors for each clusters.

Source code:

- ✓ **Import the necessary package:**

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import preprocessing
from sklearn.model_selection import GridSearchCV,RandomizedSearchCV
from sklearn.preprocessing import OneHotEncoder,OrdinalEncoder,StandardScaler
from sklearn.model_selection import train_test_split,cross_val_score
from sklearn.metrics import confusion_matrix,classification_report,ConfusionMatrixDisplay
from sklearn.linear_model import LogisticRegression,RidgeClassifier,LassoCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import
RandomForestClassifier,AdaBoostClassifier,GradientBoostingClassifier
from sklearn.metrics import
confusion_matrix,classification_report,ConfusionMatrixDisplay,accuracy_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from xgboost import XGBClassifier
from sklearn.neighbors import KNeighborsClassifier
```

- ✓ **Mount the drive**

```
from google.colab import drive
drive.mount('/content/drive')
```

- ✓ **Read the CSV file**

```
import pandas as pd
data = pd.read_csv("/content/drive/MyDrive/Workshop on AI in
Agriculture/data10L.csv", encoding='latin1')
```

- ✓ **Show the dataset of first 5 and last 5 records**

```
data.head()
data.tail()
data.info()
```

	Contrast	RMS	Correlation	Skewness	Kurtosis	Variance	Standard_Deviation	Entropy	Energy	Mean	Output
0	0.456388	8.611277	0.823309	3.358529	15.266355	1294.854341	39.105572	3.609497	0.573108	18.732925	1
1	0.523621	9.975618	0.853721	2.849428	12.042898	2004.468146	47.067766	4.170216	0.420138	27.064723	1
2	0.481740	9.922909	0.884276	2.386336	8.903400	2443.327884	51.480671	3.996901	0.459804	29.477890	1
3	0.097457	6.516737	0.987480	1.956509	5.570027	2657.972212	63.630226	2.839669	0.540382	31.075862	1
4	0.184712	8.610412	0.984210	1.267840	3.075216	4034.105273	79.741854	3.922062	0.386375	50.705765	1

	Contrast	RMS	Correlation	Skewness	Kurtosis	Variance	Standard_Deviation	Entropy	Energy	Mean	Output
398	0.157950	10.167802	0.967364	1.251144	3.333108	3483.192466	70.972414	0.519481	0.339875	50.122040	0
399	0.250123	10.993690	0.953120	1.214767	3.372392	3311.585020	63.674366	1.341504	0.315570	47.370056	0
400	0.815119	10.586401	0.886777	1.108409	2.885659	4917.459593	74.053102	2.163527	0.314780	54.032267	0
401	0.277083	8.017354	0.935475	2.004759	5.990507	2971.255470	57.670040	2.985550	0.537572	29.107671	0
402	0.520987	9.580801	0.883559	1.589998	4.534865	3198.609204	59.604628	3.807573	0.409606	36.777445	0

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 403 entries, 0 to 402
Data columns (total 11 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Contrast            403 non-null    float64
 1   RMS                 403 non-null    float64
 2   Correlation         403 non-null    float64
 3   Skewness            403 non-null    float64
 4   Kurtosis            403 non-null    float64
 5   Variance            403 non-null    float64
 6   Standard_Deviation  403 non-null    float64
 7   Entropy             403 non-null    float64
 8   Energy              403 non-null    float64
 9   Mean                403 non-null    float64
 10  Output              403 non-null    int64
dtypes: float64(10), int64(1)
memory usage: 34.8 KB
```

✓ **Data Preprocessing**

```
data.isnull().sum()
label_encoder = preprocessing.LabelEncoder()
for column in data.columns[1:]:
data[column] =  label_encoder.fit_transform(data[column])
data.head()
```

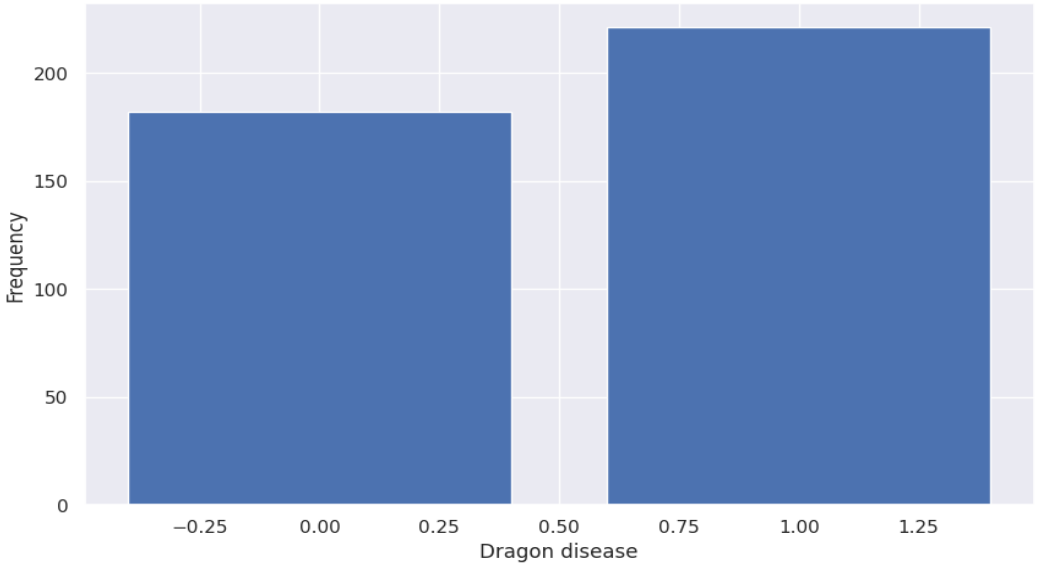
Contrast	0
RMS	0
Correlation	0
Skewness	0
Kurtosis	0
Variance	0
Standard_Deviation	0
Entropy	0
Energy	0
Mean	0
Output	0

dtype: int64

	Contrast	RMS	Correlation	Skewness	Kurtosis	Variance	Standard_Deviation	Entropy	Energy	Mean	Output
0	0.456388	205	14	394	395	46	25	318	338	29	1
1	0.523621	300	38	385	388	124	84	362	154	125	1
2	0.481740	295	74	350	363	176	132	351	188	166	1
3	0.097457	47	400	253	218	200	275	226	311	187	1
4	0.184712	204	399	96	83	337	381	346	122	325	1

✓ **Visualized the ratio of class**

```
result_counts = data['Output'].value_counts()
plt.figure(figsize=(10, 6))
plt.bar(result_counts.index, result_counts.values)
plt.xlabel('Dragon disease')
plt.ylabel('Frequency')
plt.xticks(rotation=0)
plt.show()
```



✓ **Visualize the heat map**

```
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize=(20, 20), dpi = 200)
sns.set(font_scale=2)
sns.heatmap(data.corr(), annot=True, cmap="YlGnBu")
plt.show()
```

✓ **Split the dataset**

```
X = data.drop('Output', axis=1)
y = data['Output']
```

✓ **Train and test the model**

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)
models = {'Logistic Regression': LogisticRegression(max_iter=1000),
        'K-Nearest Neighbors': KNeighborsClassifier(),
        'Decision Tree': DecisionTreeClassifier(),
        'AdaBoost': AdaBoostClassifier(),
        'Random Forest': RandomForestClassifier(),
        'Support Vector Machine': SVC()}
for name, model in models.items():
```

```
print(name)
print('---')
model.fit(X_train, y_train)
score = model.score(X_test, y_test)
print("Model score: ", score)
y_pred = model.predict(X_test)
```

Evaluate the model

```
cm = confusion_matrix(y_test, y_pred)
print('Confusion Matrix: \n',cm)
report = classification_report(y_test, y_pred)
print("Classification report:\n", report)
testing_accuracy = cm.diagonal().sum() / cm.sum()
sensitivity = cm[1, 1] / (cm[1, 0] + cm[1, 1])
specificity = cm[0, 0] / (cm[0, 1] + cm[0, 0])
fpr = cm[0, 1] / (cm[0, 1] + cm[0, 0])
fnr = cm[1, 0] / (cm[1, 0] + cm[1, 1])
fdr = cm[0, 1] / (cm[0, 1] + cm[1, 1])
npv = cm[0, 0] / (cm[0, 0] + cm[1, 0])
print("Testing accuracy: ", testing_accuracy)
print("Sensitivity: ", sensitivity)
print("Specificity: ", specificity)
print("False positive rate: ", fpr)
print("False negative rate: ", fnr)
print("False discovery rate: ", fdr)
print("Negative predictive value: ", npv)
```

Unsupervised Learning: Concepts & Applications

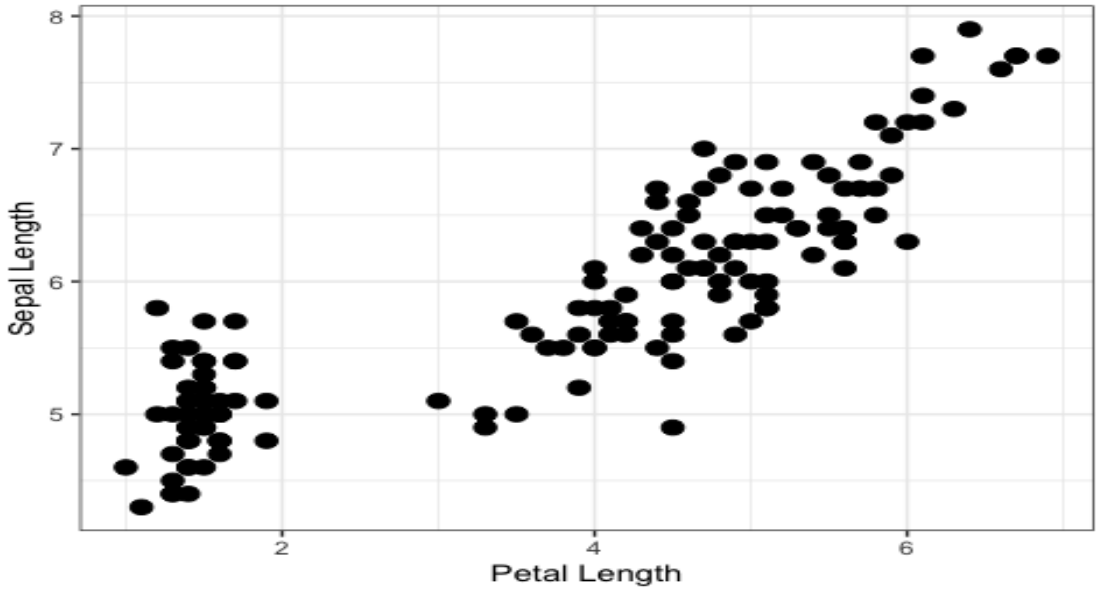
Mr. Md. Rasel Biswas
Lecturer, Institute of Statistical Research and Training (ISRT)
University of Dhaka, Bangladesh
Email: rbiswas1@isrt.ac.bd

Linear Regression

- Linear regression is a simple and widely used technique in machine learning and statistics to understand the relationship between two or more variables.
- It helps us predict one variable (called the target or response) using some other variables (called the feature/input/predictors).

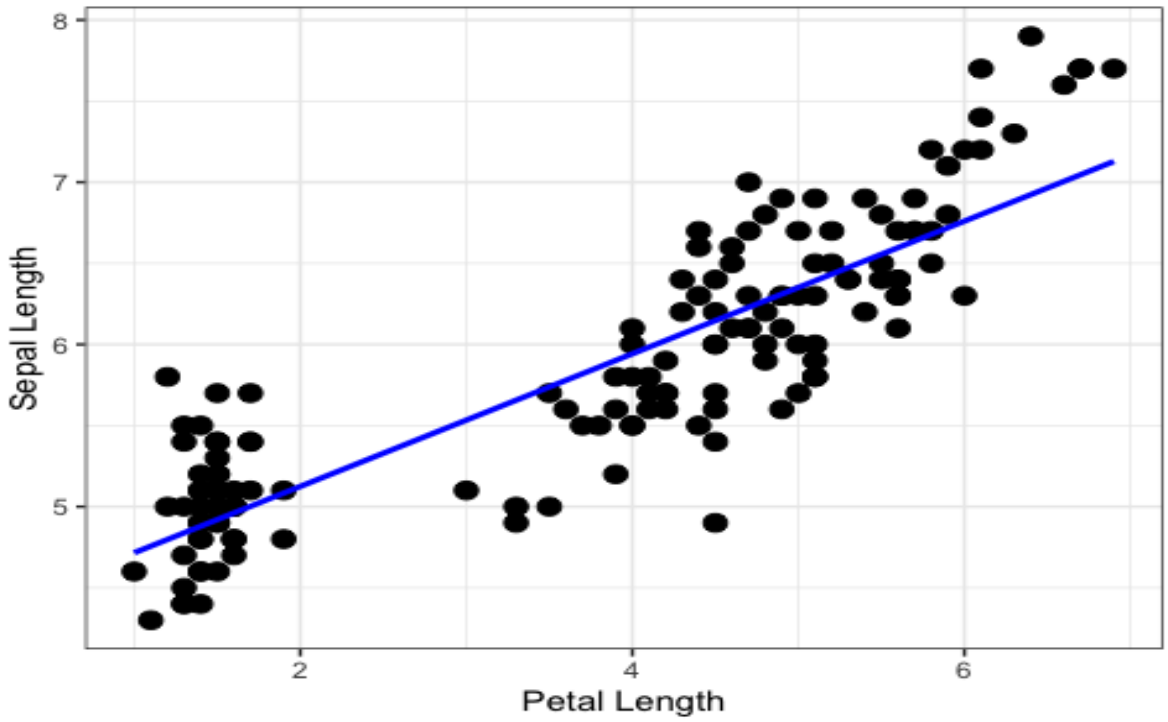
Scatter Plot

- A scatter plot is a type of graph that shows the relationship between two numerical variables using **dots**.
- Each dot represents one observation (data point), with:
    - the x-axis showing one variable (e.g., Sepal Length),
    - and the y-axis showing another variable (e.g., Petal Length).



- A scatter plot shows whether the points follow a pattern — like a straight line going upward or downward.

- If the dots form a roughly straight path, we might say: "There's a linear relationship between these variables."
- Linear regression takes that idea further by finding the best-fitting straight line through the scatter plot.



Equation of a Regression Line

- Recall, from algebra, the equation of the line is $y = mx + c$.
    - **Intercept (c)**: where the line crosses the y-axis (when x = 0)
    - **Slope (m)**: the "rise over the run" - how much y changes for each one unit change in x
- Statisticians may have different notations:

$$E(y|x) = \beta_0 + \beta_1 x$$

    - β_0 = intercept
    - β_1 = slope

Simple Linear Regression

$$y = \beta_0 + \beta_1 x + \varepsilon$$

where β_0 is intercept, β_1 is slope, and ε is the error term.

- **Assumptions**:
    - Linearity
    - Independence
    - Homoscedasticity
    - Normality of residuals

- The fitted model is

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x$$

Interpretation of the parameter estimates:

- **intercept**: $\hat{\beta}_0$ is the estimated average value of y when the value of x is zero (if $x = 0$ is in the range of observed x values)
- **slope**: $\hat{\beta}_1$ is the estimated changes in the average value of y for additional one unit increase in x.

Example

- Suppose we wish to model the linear relationship between the TV advertising budget and sales
- We can write this as $Y = \beta_0 + \beta_1 X + \epsilon$, where Y represents sales and X represents TV budget

```
advertising = read_csv("Advertising.csv")
model1 <- lm(sales ~ TV, data = advertising)
broom::tidy(model1)
```

```
# A tibble: 2 × 5
```

term	estimate	std.error	statistic	p.value
<chr>	<dbl>	<dbl>	<dbl>	<dbl>
1 (Intercept)	7.03	0.458	15.4	1.41e-35
2 TV	0.0475	0.00269	17.7	1.47e-42

Estimated Regression Model

- The estimated intercept $\hat{\beta}_0 = 7.03$ and slope $\hat{\beta}_1 = 0.05$.
- The estimated regression model is

$$\hat{Y} = 7.03 + 0.05X.$$

- Interpretation:
    - $\hat{\beta}_0 = 7.03$: When the TV advertising budget is zero we can expect sales to be \$7030 (remember we are operating in units of 1,000).
    - $\hat{\beta}_1 = 0.05$: For every \$1,000 increase in the TV advertising budget we expect the average increase in sales to be 50 units.

Multiple Linear Regression

- Multiple linear regression is an extension of simple linear regression.
- In general, suppose that we have p distinct predictors. Then the multiple linear regression model takes the form.

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p + \epsilon,$$

where

- X_j represents the j-th predictor
- β_j represents the regression coefficient of the predictor X_j
- We interpret β_j as the average effect on Y of a one unit increase in X_j, holding all other predictors fixed

---

- Previously, we have examined the relationship between **sales** and **TV**.
- We may want to know whether the varables **radio** and **newspaper** have any effect on sales.
- We now extend the analysis of the **advertising data** in order to accommodate these two variables.

```
# To run multiple linear regression model
model2 <- lm(sales ~ TV + radio + newspaper,
             data = advertising)
broom::tidy(model2)
```

```
# A tibble: 4 × 5
  term        estimate std.error statistic  p.value
  <chr>          <dbl>     <dbl>     <dbl>    <dbl>
1 (Intercept)  2.94      0.312       9.42  1.27e-17
2 TV           0.0458    0.00139    32.8   1.51e-81
3 radio        0.189     0.00861    21.9   1.51e-54
4 newspaper   -0.00104   0.00587    -0.177 8.60e- 1
```

- The estimated model is

$$\text{Sales} = 2.94 + 0.046 \times \text{TV} + 0.189 \times \text{radio} - 0.001 \times \text{news}$$

- **Interpretation of model parameters**
    - The estimated coefficent of TV suggests that for every USD 1,000 increase in TV advertising budget, holding all other predictors constant, we can expect an increase of 46 sales units, on average.
    - The estimated coefficient of newspaper suggests that for every 1,000 increase in Radio advertising budget, holding all other predictors constant, we can expect an increase of 189 sales units, on average.

Model Diagnostics

- Residuals

$$\hat{\epsilon} = y - \hat{y}$$

- Residuals are used for model diagnostics

Test for significance of single parameter

```
broom::tidy(model2)
```

```
# A tibble: 4 × 5
  term        estimate std.error statistic  p.value
  <chr>          <dbl>     <dbl>     <dbl>    <dbl>
1 (Intercept)  2.94      0.312       9.42  1.27e-17
2 TV           0.0458    0.00139    32.8   1.51e-81
3 radio        0.189     0.00861    21.9   1.51e-54
4 newspaper   -0.00104   0.00587    -0.177 8.60e- 1
```

- With $\alpha = 0.05$, the variables TV and radio have significant effect on sales.

Model summary

```
summary(model2)
```

```
Call:
lm(formula = sales ~ TV + radio + newspaper, data = advertising)

Residuals:
    Min      1Q  Median      3Q     Max
-8.8277 -0.8908  0.2418  1.1893  2.8292

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  2.938889   0.311908   9.422   <2e-16 ***
TV           0.045765   0.001395  32.809   <2e-16 ***
radio        0.188530   0.008611  21.893   <2e-16 ***
```

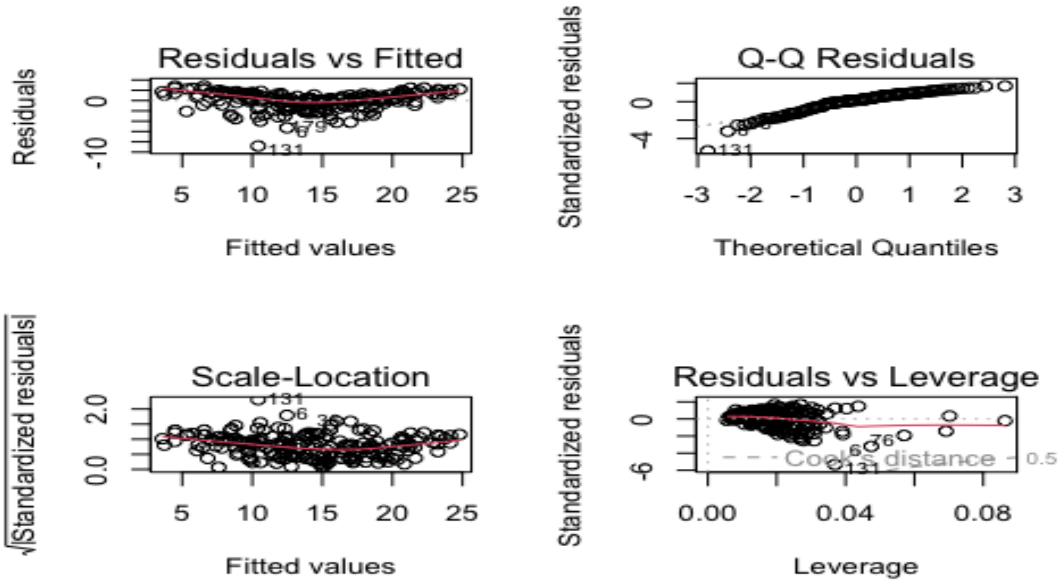
```
newspaper   -0.001037   0.005871  -0.177     0.86
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.686 on 196 degrees of freedom
Multiple R-squared:  0.8972,    Adjusted R-squared:  0.8956
F-statistic: 570.3 on 3 and 196 DF,  p-value: < 2.2e-16
```

Assessing Model Accuracy: R^2 Statistic

- The R^2 statistic represents the proportion of variance explained and so it always takes on a value between 0 and 1
- The adjusted R^2 is a modified version of R^2 that has been adjusted for the number of predictors in the model.
- In our model adjusted $R^2 = 0.90$ which means 90% of total variability of sales can be explained using the fitted model.
- However, a higher values of R^2 doesn't mean the model is good.
- **F-statistic**: the F-statistic (statistic) in model 2 is larger than model 1. Here larger is better and suggests that model 2 provides a better "goodness-of-fit".

```
par(mfrow=c(2, 2))
plot(model2)
```



Unsupervised Learning: Concepts & Applications

Machine Learning

- **Machine Learning** is a subfield of artificial intelligence that allows systems to learn and improve from experience without being explicitly programmed.
- Imagine teaching a child to recognize animals by showing them lots of pictures — ML works in a similar way.

Two Main Types of Machine Learning

1. Supervised Learning

This is like learning with a teacher.

- In supervised learning, the computer is trained using data that is already labeled — meaning, we tell the computer the *right answers* during training.
- **Goal**: Learn the relationship between input (features) and output (label) so it can predict the output for new, unseen inputs.

Example:

  - **Data**: You have a list of houses (inputs: size, number of bedrooms, location) and the price of each house (output: price).
  - **Task**: Teach the model to predict the price of a new house.

- **Common Algorithms**: Linear regression, Logistic regression, Decision trees, Random forests, Support vector machines, Neural networks
- **Applications**: Email spam detection (spam or not spam), Medical diagnosis (disease or no disease), Stock price prediction

---

2. Unsupervised Learning

This is like learning without a teacher.

- In unsupervised learning, the data has no labels — the computer tries to find hidden patterns or groupings all by itself.
- **Goal**: Explore the structure or distribution in the data without any predefined output.

- **Example**:
    - Data: A bunch of customer profiles (age, income, spending habits), but no info about customer types.
    - Task: Group similar customers together (called clustering) without knowing what types to expect.

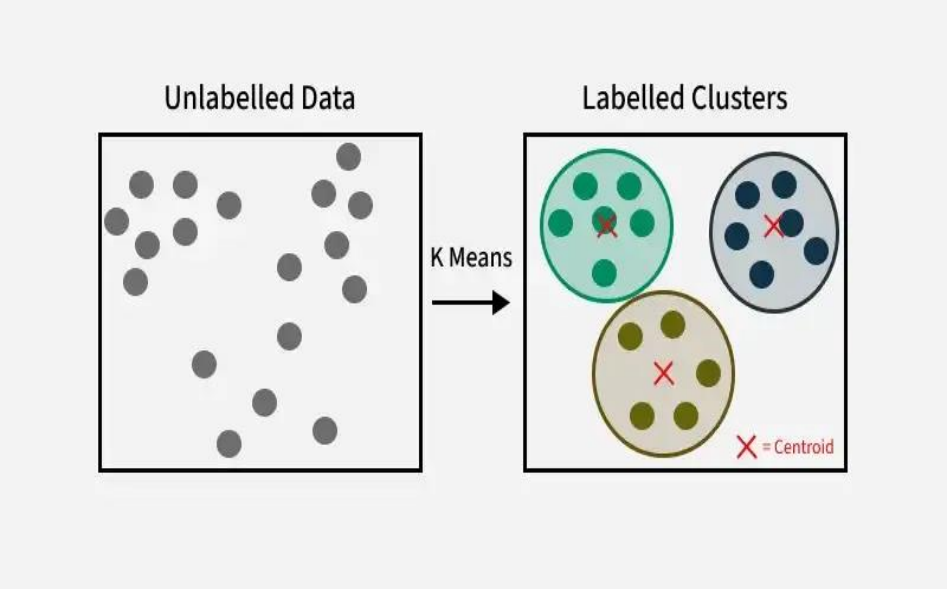
- **Common Algorithms**: K-means clustering, Hierarchical clustering, Principal Component Analysis (PCA), t-SNE, Autoencoders
- **Applications**: Market segmentation (grouping customers), Anomaly detection (fraud detection), Recommender systems (like Netflix or Amazon)

Clustering Techniques

- Clustering techniques are a set of unsupervised machine learning methods used to group similar data points together based on their characteristics or features — without using predefined labels. The goal is to identify natural groupings or structures in data.
- Popular clustering algorithms are
    1. **K-Means Clustering**, and
    2. **Hierarchical Clustering**

1. K-Means Clustering

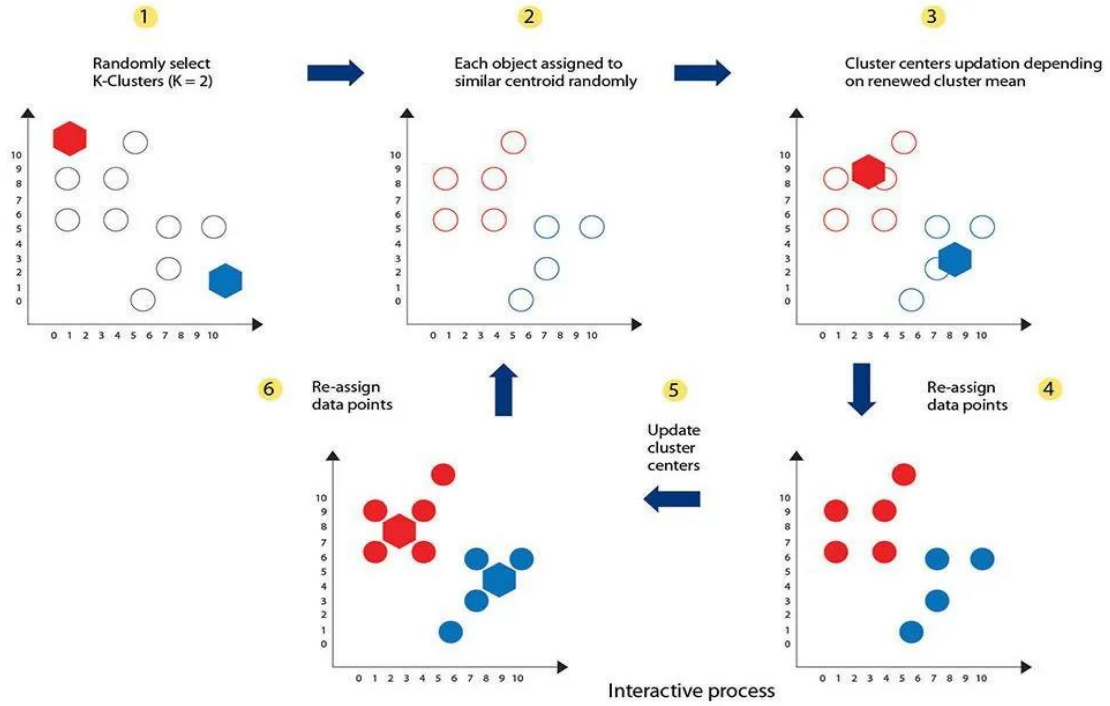
- K-Means divides a dataset into K clusters, where each data point belongs to the cluster with the nearest mean (also called the centroid).



How K-Means Works (Step-by-Step)

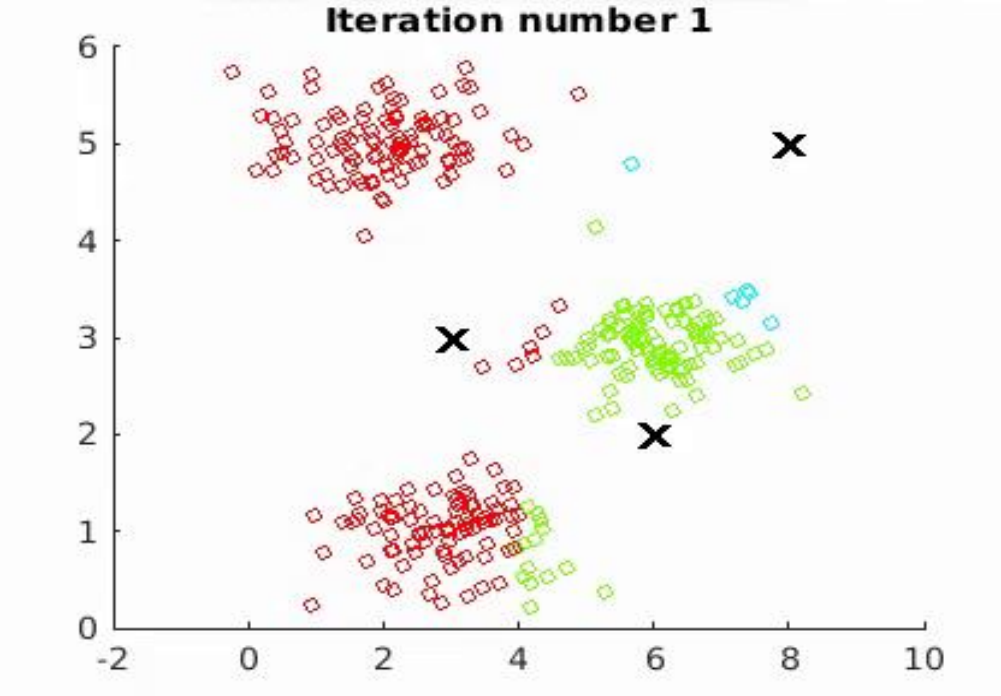
1. Choose number of clusters K
2. Initialize K centroids randomly
3. Assign each data point to the nearest centroid
4. Recalculate centroids as the mean of assigned points
5. Repeat steps 3–4 until convergence

Schematic of K-Means

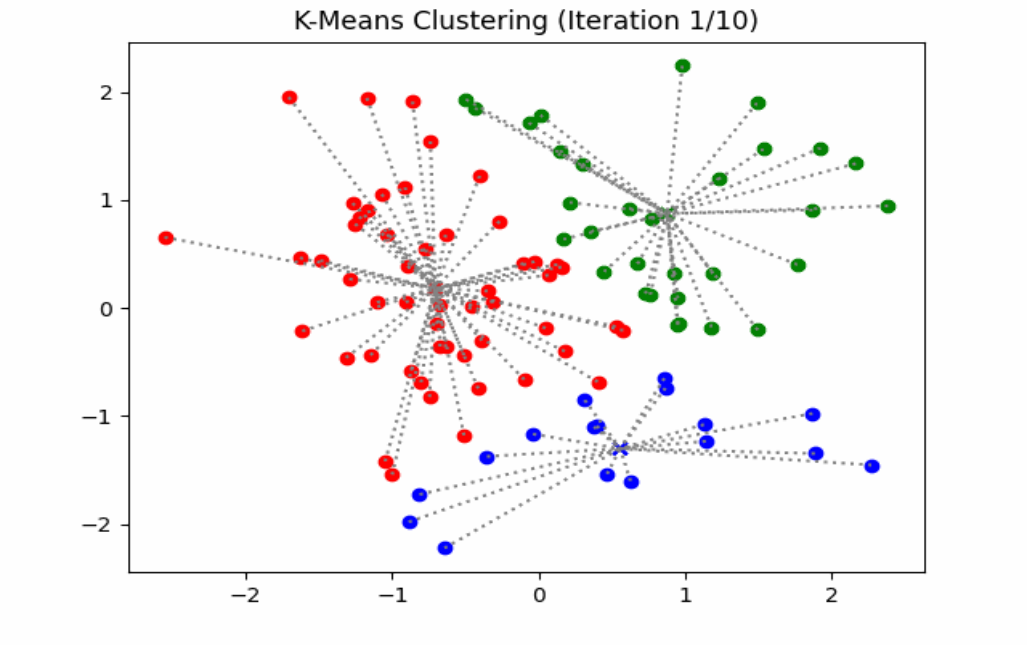


https://medium.com/@atakanerdogan305

Movement of the Centroids



https://medium.com/@atakanerdogan305



https://medium.com/@TechTushaar

2. Hierarchical Clustering

- Hierarchical Clustering groups data by either merging smaller clusters (agglomerative) or splitting larger clusters (divisive), forming a tree-like structure called a dendrogram.

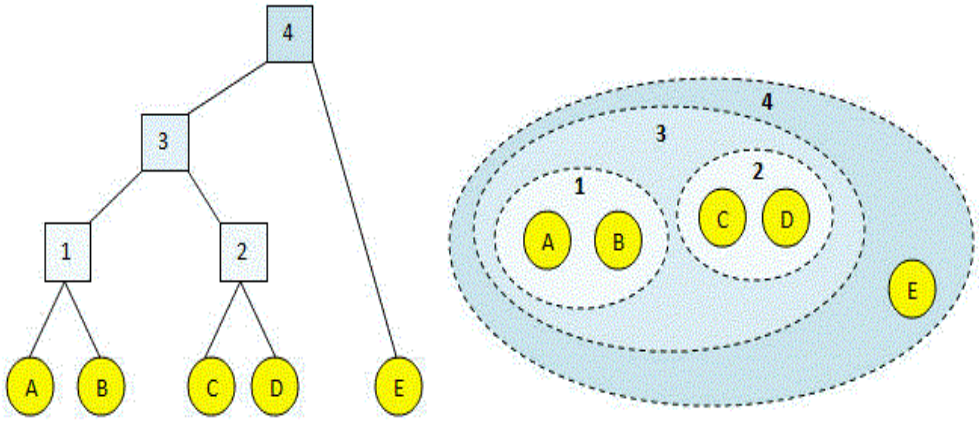
- It does not require specifying the number of clusters. you can choose clusters by cutting the dendrogram at a chosen level.

Types of Hierarchical Clustering

- Agglomerative (Bottom-Up)
    - Starts with each data point as its own cluster.
    - At each step, the closest two clusters are merged.
    - Continues merging until all points are in one cluster.
- Divisive (Top-Down)
    - Starts with all data points in a single cluster.
    - At each step, the most dissimilar points are split into smaller clusters.
    - Continues splitting until each point is its own cluster

Distance metrics

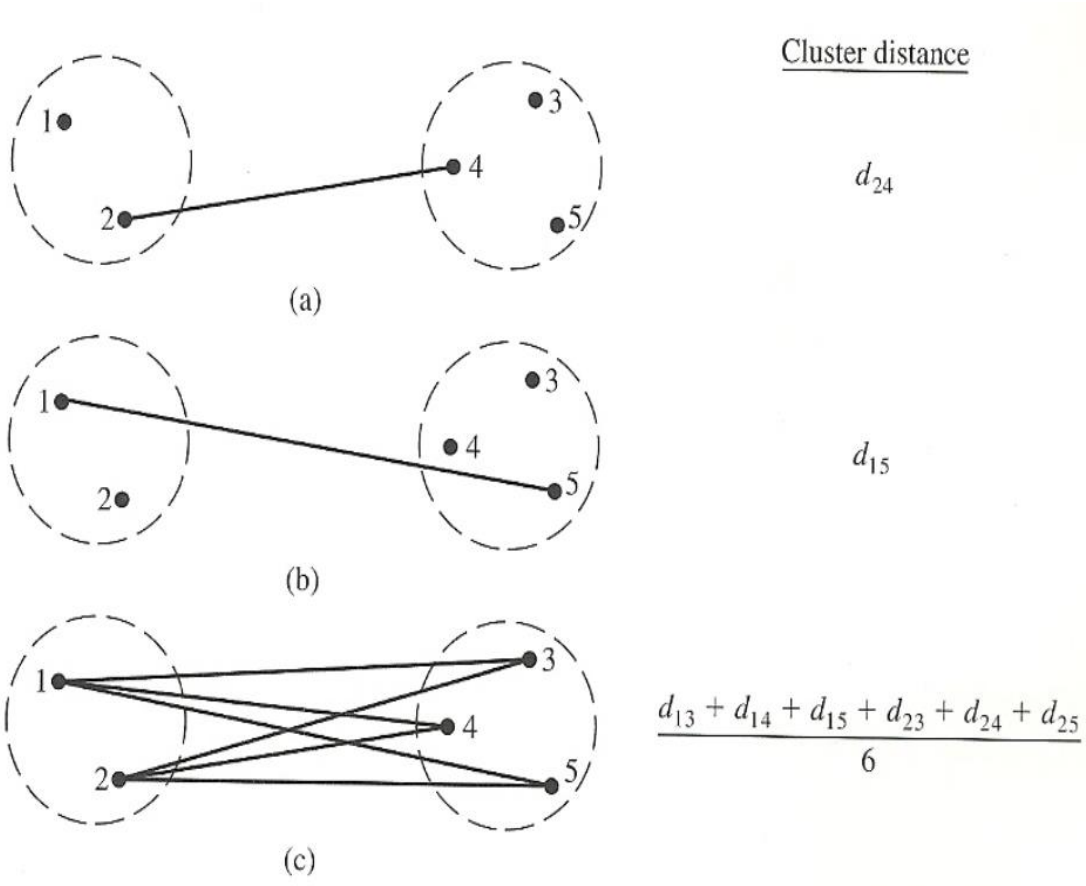
- Agglomerative clustering relies on a distance metric, such as **Euclidean distance**, to determine the similarity between clusters and decide which ones to merge at each step.



https://medium.com/@khushivirpariya

Linkage Criteria

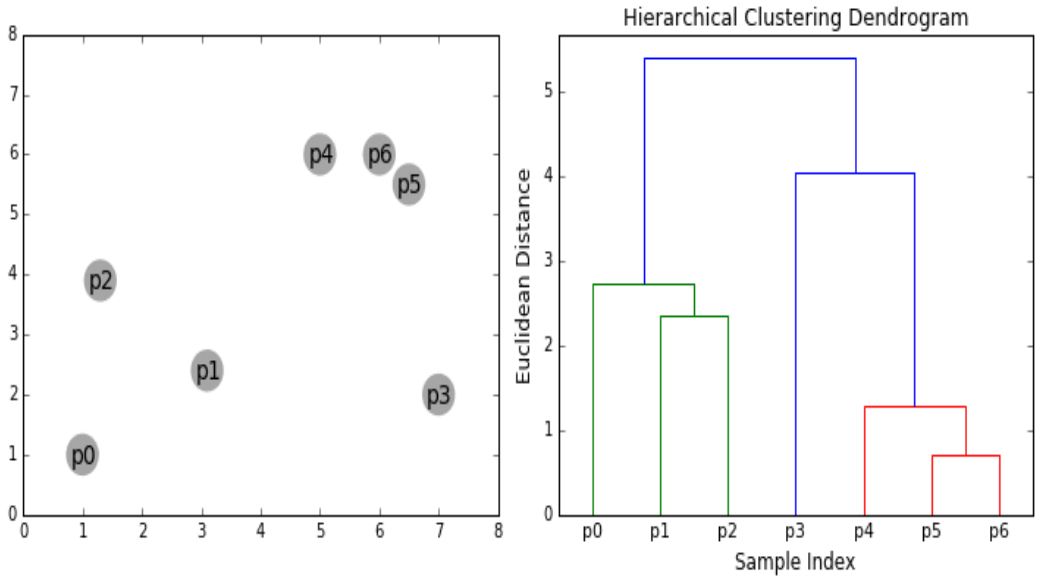
- Single Linkage: Minimum distance
- Complete Linkage: Maximum distance
- Average Linkage: Mean distance
- Ward's Method: Minimize total within-cluster variance



Intercluster distance for (a) single linkage, (b) complete linkage, and (c) average linkage. Source: Johnson and Wichern (2007).

Dendrogram

- Visual representation of merge/split steps
- Cut the dendrogram at a certain height to choose number of clusters



https://dashee87.github.io

Summary

- Unsupervised learning uncovers hidden patterns
- K-means is efficient but requires specifying K
- Hierarchical clustering is computationally expensive, but no need to pre-specify number of clusters

Application using python

```
# Import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans, AgglomerativeClustering
from scipy.cluster.hierarchy import dendrogram, linkage

# Load the iris dataset
iris = load_iris()
```

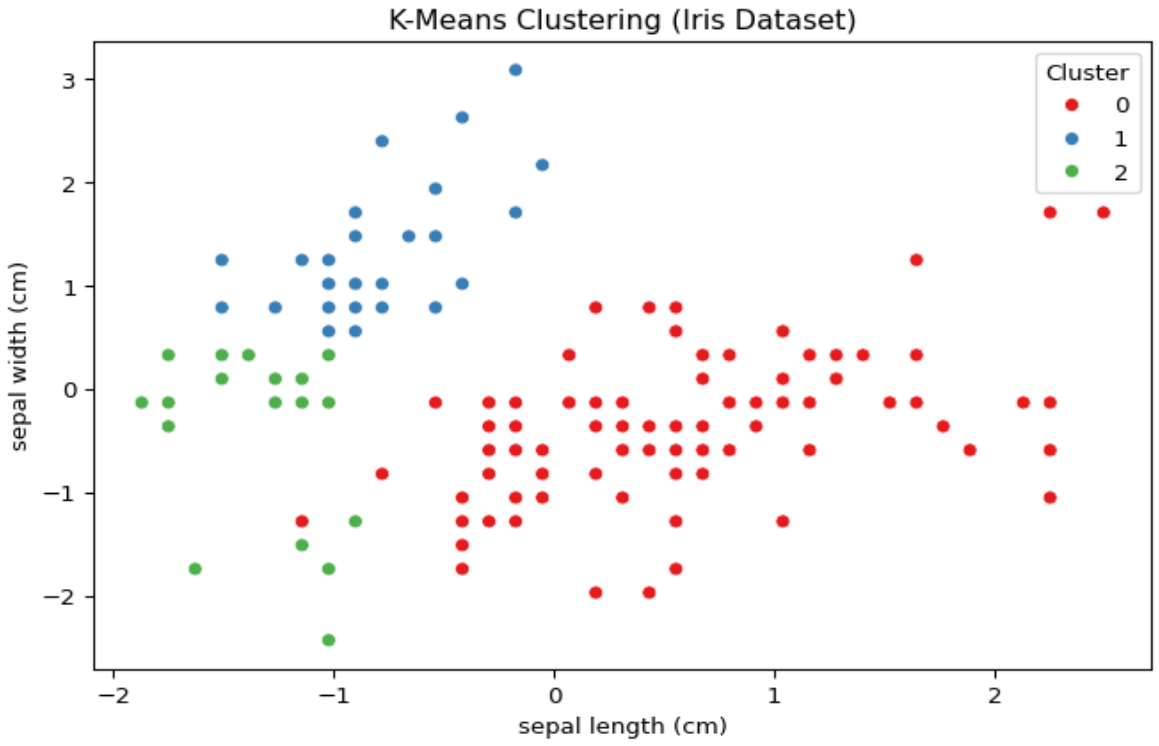
```
X = iris.data
y = iris.target
features = iris.feature_names

# Standardize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Create a DataFrame
df = pd.DataFrame(X_scaled, columns=features)
```

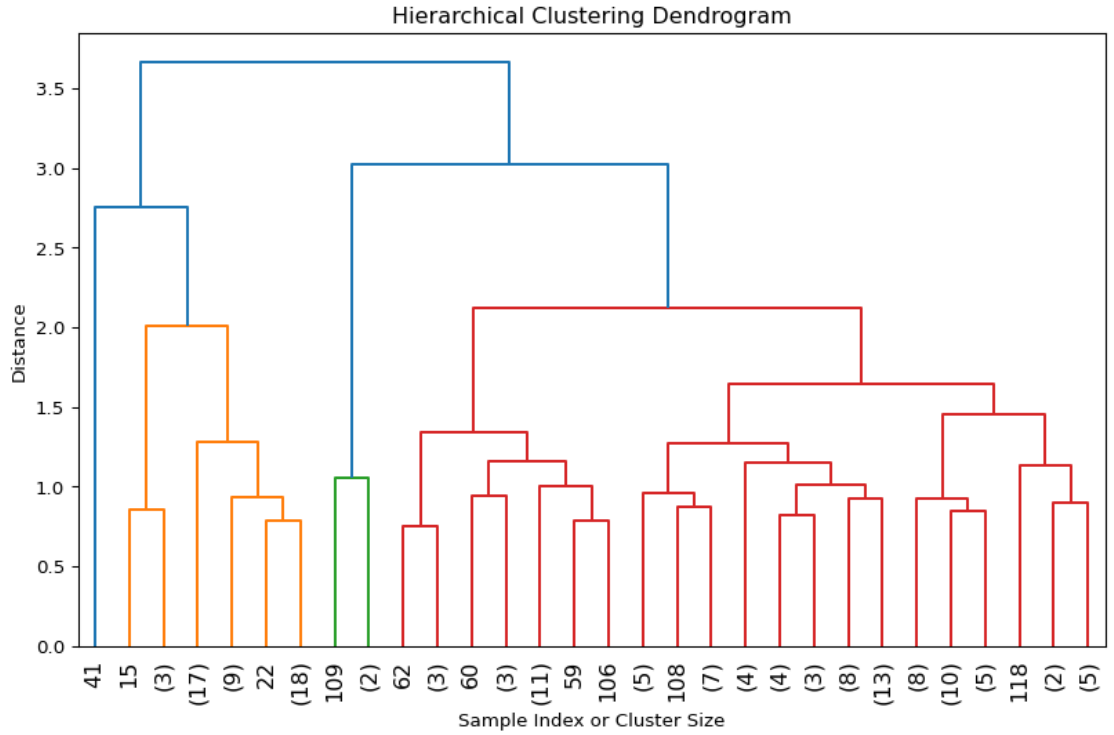
```
# K-MEANS CLUSTERING
kmeans = KMeans(n_clusters=3, random_state=42)
df['kmeans_cluster'] = kmeans.fit_predict(X_scaled)

# Plot K-Means Clustering result
plt.figure(figsize=(8, 5))
sns.scatterplot(x=df[features[0]], y=df[features[1]], hue=df['kmeans_cluster'], palette='Set1')
plt.title('K-Means Clustering (Iris Dataset)')
plt.xlabel(features[0])
plt.ylabel(features[1])
plt.legend(title='Cluster')
plt.show()
```



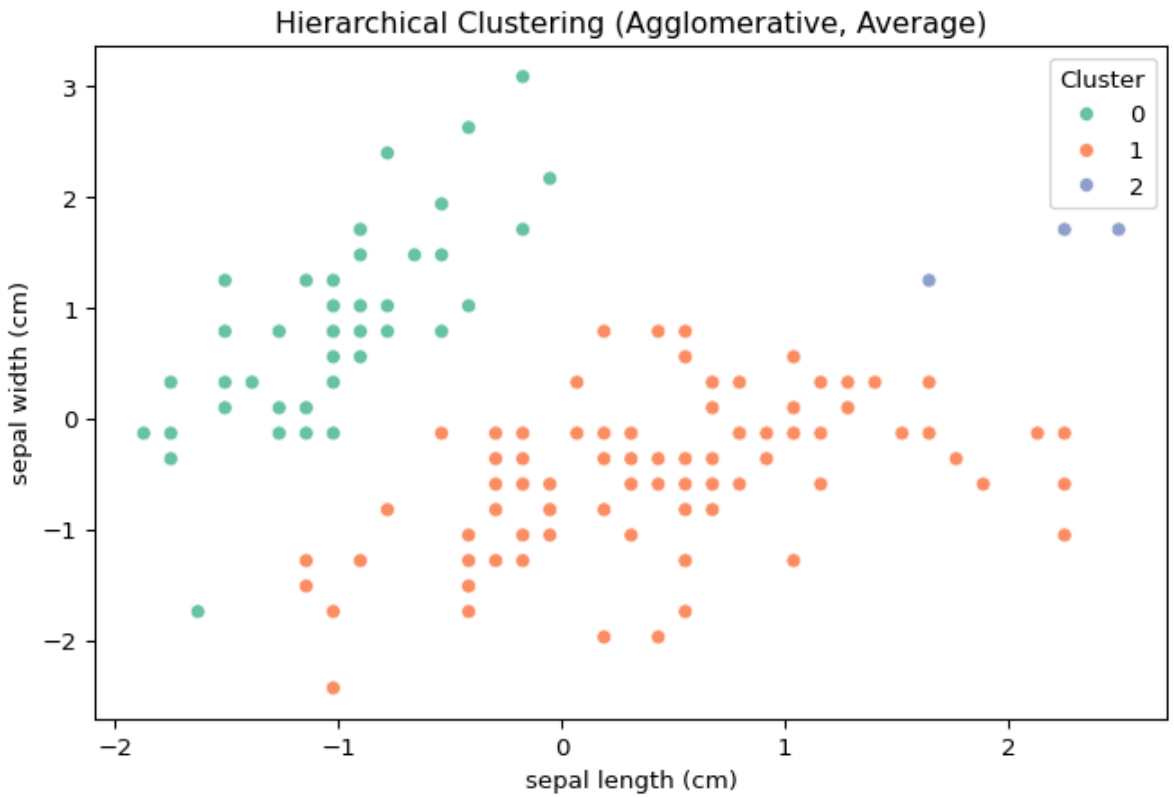
```
# HIERARCHICAL CLUSTERING
# Create linkage matrix
Z = linkage(X_scaled, method='average')

# Plot dendrogram
plt.figure(figsize=(10, 6))
dendrogram(Z, truncate_mode='lastp', p=30, leaf_rotation=90., leaf_font_size=12.)
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('Sample Index or Cluster Size')
plt.ylabel('Distance')
plt.show()
```



```
# Apply Agglomerative Clustering (Average linkage, 3 clusters)
agg = AgglomerativeClustering(n_clusters=3, linkage='average')
df['hierarchical_cluster'] = agg.fit_predict(X_scaled)

# Plot Agglomerative Clustering result
plt.figure(figsize=(8, 5))
sns.scatterplot(x=df[features[0]], y=df[features[1]], hue=df['hierarchical_cluster'], palette='Set2')
plt.title('Hierarchical Clustering (Agglomerative, Average)')
plt.xlabel(features[0])
plt.ylabel(features[1])
plt.legend(title='Cluster')
plt.show()
```


Hierarchical Clustering (Agglomerative, Average)
Cluster
0
1
2
sepal width (cm)
sepal length (cm)
3
2
1
0
−1
−2
−2
−1
0
1
2

Introduction to Neural Networks (ANN)
Dr. Md. Zulfiker Mahmud
Professor, Department of CSE, Jagannath University
Email: zulfiker@cse.jnu.ac.bd

Deep Learning & AI
Artificial Neural Networks (ANNs)

$$y_1 = \sigma(w_1 \cdot x_1 + w_2 \cdot x_2 + w_3 \cdot x_3 + b) = \sigma(\sum_i w_i \cdot x_i + b)$$

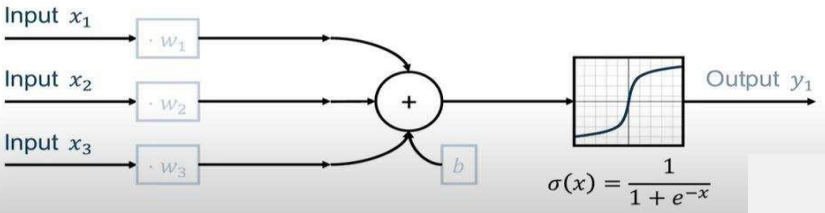


Fig: Artificial Neuron (Single Layer Perceptron)

Deep Learning & AI
Neuron in Human Brain

A human neuron is an incredibly complex biological cell with numerous structures like dendrites, an axon, the soma (cell body), and synaptic terminals. The human brain is estimated to contain approximately **86 billion** neurons.

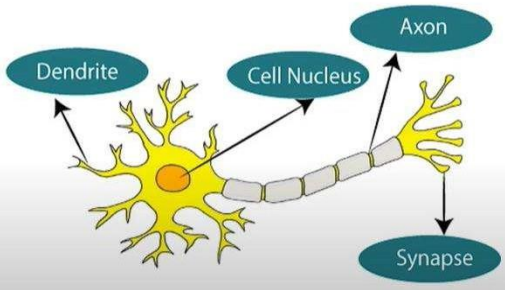


Fig: A Neuron

Deep Learning & AI

Artificial Neural Networks (ANNs)

- An artificial neuron is a simple computational model that includes input weights, an a
  an output.
- It processes information using mathematical functions, taking numeric inputs, multiplying them by weights, summing them up, adding a bias, and passing the result through an activation function.

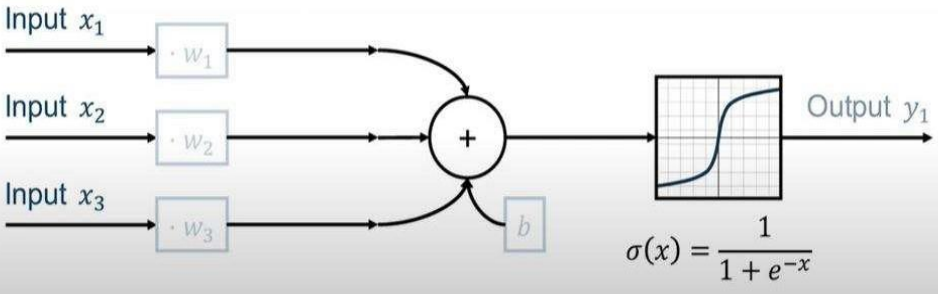
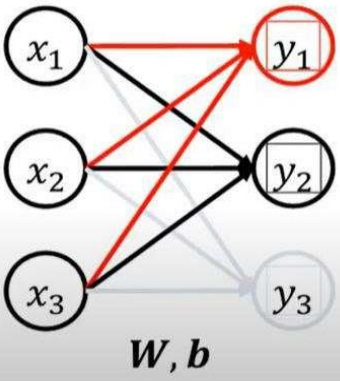


Fig: Artificial Neuron (Single Layer Perceptron)

Deep Learning & AI

Artificial Neural Networks (ANNs)

We can combine multiple perceptrons to create a layer.



We can thus rewrite the three computations as:

$$\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}$$

Or in a more simplified form:

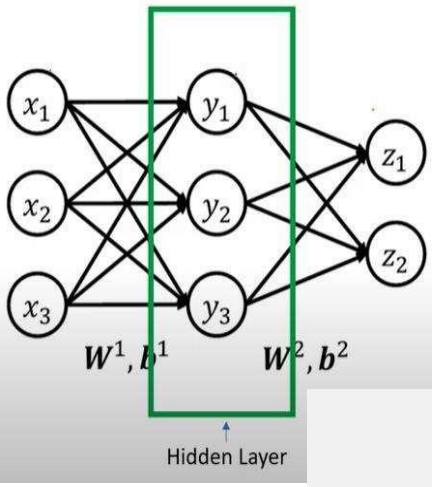
$$y = \sigma(W \cdot x + b)$$

Deep Learning & AI

Artificial Neural Networks (ANNs)

We call "hidden layer" any layer in between the input and the output layers.

- ❖ X: Input Layer
- ❖ Y: Hidden Layer
- ❖ Z: Output Layer



Deep Learning & AI

Artificial Neural Networks (ANNs)

We can chain multiple layers, with each output being the input of the next:

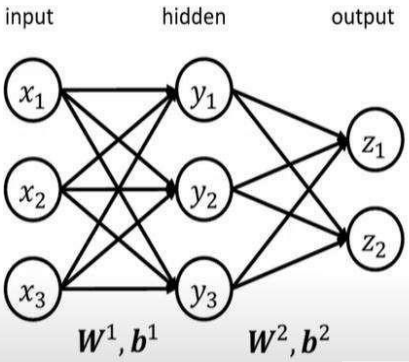
$$y = \sigma(W^1 \cdot x + b^1)$$

$$z = \sigma(W^2 \cdot y + b^2)$$

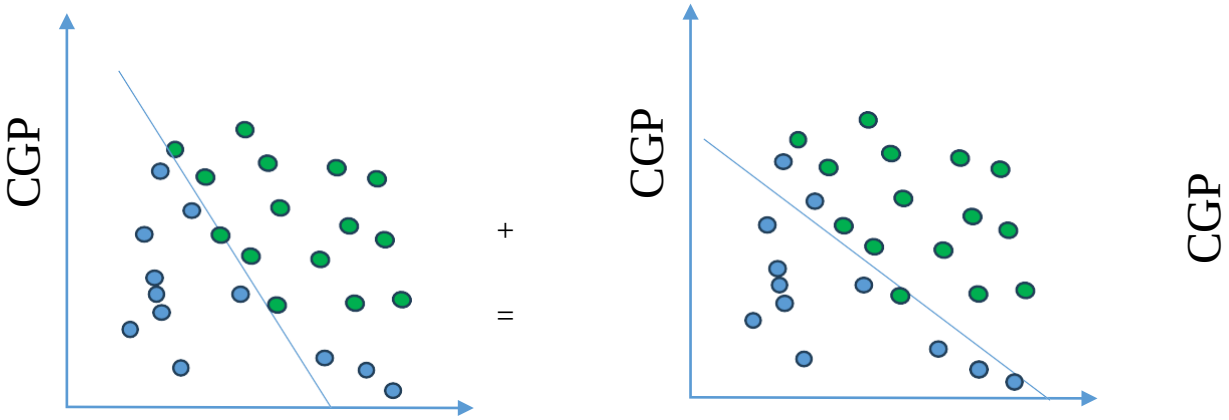
Combining these leads us to:

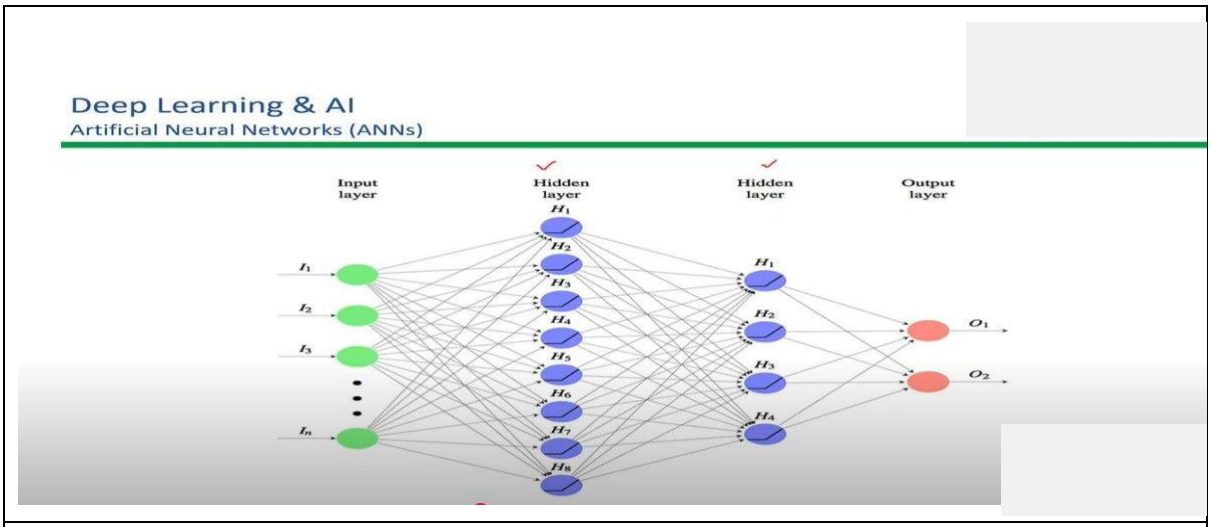
$$z = \sigma(W^2 \cdot \underbrace{\sigma(W^1 \cdot x + b^1)}_{y} + b^2)$$

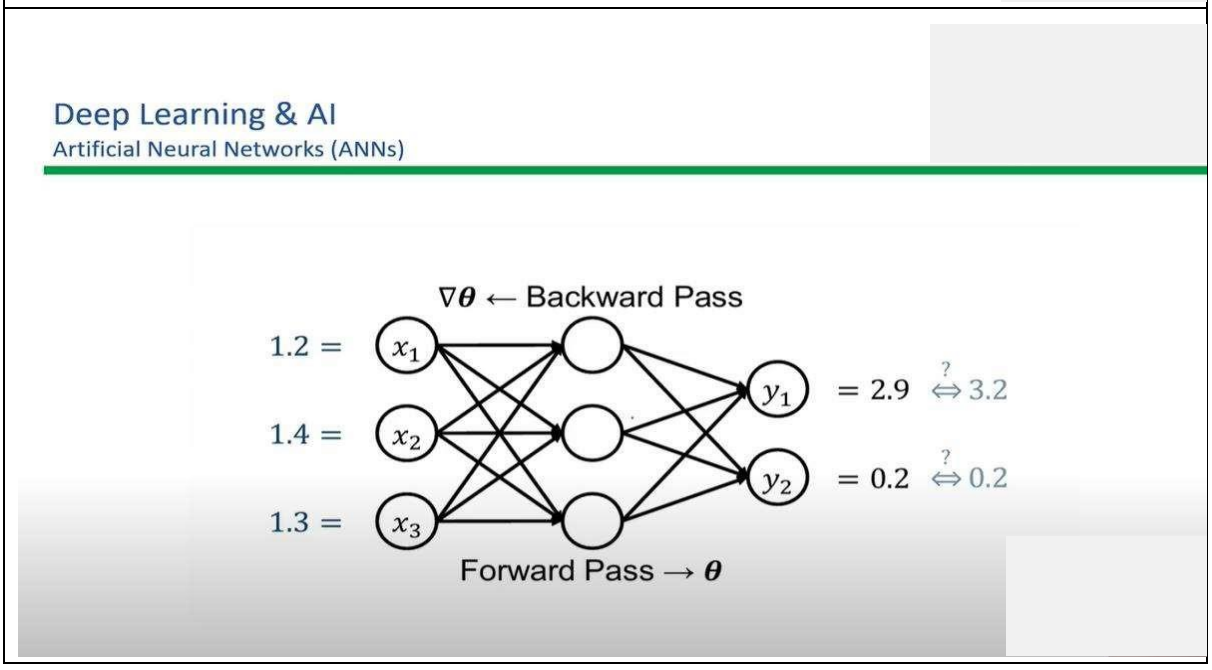
- Each layer has its own set of parameters (weights W^i and bias b^i)
- The underlying computation is a matrix multiplication described by $y^{i+1} = \sigma(W^i \cdot y^i + b^i)$



How ANN can capture non-linear pattern?







Deep Learning & AI
Artificial Neural Networks (ANNs)

Forward Pass:

- The input values ($x_1 = 1.2, x_2 = 1.4, x_3 = 1.3$) are fed into the network.
- These inputs are processed by the hidden layer(s) through a series of weighted sums followed by activation functions (not shown in detail in the image).
- The resulting values are then passed to the output layer, producing the outputs ($y_1 = 2.9$ and $y_2 = 0.2$).

The forward pass is essentially the computation that happens when the network makes predictions. It starts from the input layer and propagates through the hidden layers and finally to the output layer.

Deep Learning & AI
Artificial Neural Networks (ANNs)

Backward Pass:

- This phase is about learning from errors and updating the weights (θ) in the network.
- The true target values for the outputs are compared with the predicted values from the forward pass (3.2 for y_1 and 0.2 for y_2).
- The differences between the predicted values and true values ($2.9 - 3.2 = -0.3$ for y_1 and $0.2 - 0.2 = 0$ for y_2) represent the errors.
- These errors are then used to calculate the gradients of the loss function with respect to the weights ($\nabla\theta$).

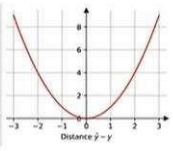
The backward pass involves calculating the gradient of the loss function with respect to each weight in the network (this gradient tells us how to change the weights to decrease the error). Then, the weights are updated typically using an optimization algorithm like gradient descent.

The loss function is a comparison metric between the predicted outputs $\hat{y}_i$ and the expected outputs y_i.

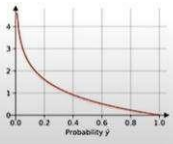
The choice of the loss function usually depends on the type of problem:

- For regression, a common metric is the mean squared error
- For classification, a common metric is the cross entropy

$$MSE(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$$



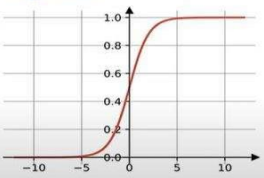
$$CE(\hat{\boldsymbol{y}}, \boldsymbol{y}) = -\sum_{i} y_i \log(\hat{y}_j)$$



Activation Functions

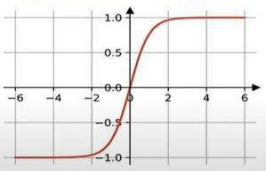
It is a mathematical function that takes the **weighted sum of the inputs** and **bias** as input and then generates an output, typically used to **add non-linearity** to the model.

Sigmoid



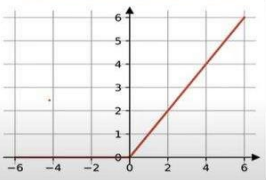
$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Hyperbolic Tangent



$$\sigma(x) = \tanh(x)$$

Rectified Linear Unit



$$\sigma(x) = \max(x, 0)$$

Activation Functions

Why are activation functions important?

- **Non-Linearity:** Without non-linear activation functions, a neural network would essentially become a linear regression model, incapable of handling the complexities of most real-world data, which often are non-linear.
- **Control of Signal Flow:** Activation functions decide whether a neuron should be activated or not, effectively allowing the network to make complex decisions and learn from the data.
- **Learning Complex Patterns:** The introduction of non-linear activation functions allows the neural network to learn complex patterns in the data by stacking layers of neurons, each possibly using different activation functions.
- **Backpropagation:** Activation functions that are differentiable play a vital role in backpropagation, the training process for neural networks. The derivative of the activation function is used to update the weights of the neurons.

Activation Functions

Common types of activation functions:

- **Sigmoid or Logistic Function:** It squashes the input values into a range between **0 and 1**, which can represent probabilities (Binary class).
- **Tanh Function:** It scales the input values between **-1 and 1**, which is a zero-centered range, often leading to better training performance for multi-layer networks.
- **ReLU (Rectified Linear Unit):** It provides a linear response for positive inputs and **zero for negative inputs**, which allows for faster training and addresses the vanishing gradient problem to some extent.
- **Softmax:** Often used in the output layer for multi-class classification problems. The output of the softmax function for each class is in the range (0,1), and the **sum** of all the output probabilities for **all classes will be 1.**

Activation Functions

ReLU (Rectified Linear Unit):

- **Formula:** $\mathrm{ReLU}(x) = \max(0, x)$
- **Description:** ReLU is widely used because it helps mitigate the vanishing gradient problem, allowing models to learn faster and more effectively. It is simple and computationally efficient.

Activation Functions

ReLU (Rectified Linear Unit):

- **Formula:** $\mathrm{ReLU}(x) = \max(0, x)$
- **Description:** ReLU is widely used because it helps mitigate the vanishing gradient problem, allowing models to learn faster and more effectively. It is simple and computationally efficient.

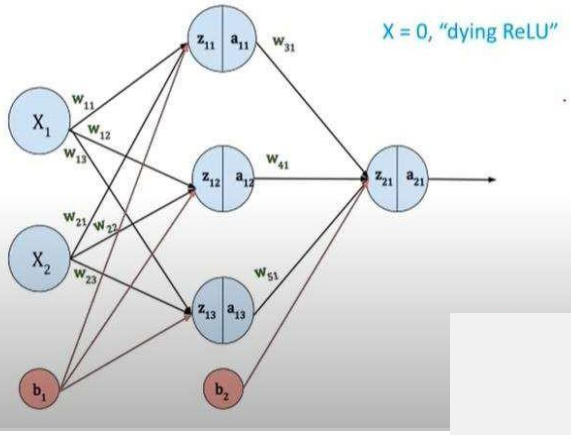
Variants: Due to some of the **limitations** of the standard ReLU, several variants have been proposed:

- **Leaky ReLU:** It allows a small, non-zero, constant gradient ϵ when the unit is not active and x is less than zero ($f(x) = \epsilon x$ for $x < 0$).
- **Parametric ReLU (PReLU):** It generalizes the leaky ReLU by allowing the gradient during the non-active phase to be learned during training.
- **Exponential Linear Unit (ELU):** It also allows for a small gradient when x is negative, but instead being linear, the negative part is an exponential decay.

Activation Functions

This means that the output is the input itself if the input is greater than zero, and zero if the input is zero or negative. Let's compute the ReLU activation for the following input values: $-3, -1, 0, 2, 4$.

1. For $x = -3$:
$f(-3) = \max(0, -3) = 0$
2. For $x = -1$:
$f(-1) = \max(0, -1) = 0$
3. For $x = 0$:
$f(0) = \max(0, 0) = 0$
4. For $x = 2$:
$f(2) = \max(0, 2) = 2$
5. For $x = 4$:
$f(4) = \max(0, 4) = 4$



Activation Functions

Leaky ReLU Function:

- **Function:** $f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{if } x \leq 0 \end{cases}$
  - Here, α is a small positive coefficient, such as 0.01.
- **Output**: Positive inputs yield positive outputs (just like ReLU), but negative inputs result in small negative outputs instead of zeros because of the multiplication by the small coefficient α.

For example, if $\alpha = 0.01$ and $x = -5$, then:

- $f(x) = 0.01 \times -5 = -0.05$

Activation Functions

Key Differences

- **Output Range**:
  - **ReLU**: $[0, \infty)$
  - **Leaky ReLU**: $(-\infty, \infty)$, although negative values are very small
- **Gradient**:
  - **ReLU**: The gradient is 1 for positive inputs and 0 for negative inputs.
  - **Leaky ReLU**: The gradient is 1 for positive inputs and α (a small value) for negative inputs, which helps to keep information flowing through the network even when the input is negative.

Activation Functions

Tanh (Hyperbolic Tangent):

- **Formula**: $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
- **Description**: Tanh squashes the outputs to the range between -1 and 1, which can be more desirable than ReLU in certain cases, particularly if the model needs to handle negative outputs naturally.

- **Usage in Neural Networks:** It was particularly popular in the hidden layers of traditional neural networks and is still widely used in the **gates of LSTM** (Long Short-Term Memory) and **GRU** (Gated Recurrent Unit) cells in recurrent neural networks, where the regulation of information flow benefits from the symmetric properties of the functior
- **Disadvantages:** Vanishing Gradients Problem & More Comp

Activation Functions

ELU (Exponential Linear Unit):

- **Formula**: $\mathrm{ELU}(x) = \begin{cases} x & \text{if } x \geq 0 \\ \alpha(e^x - 1) & \text{if } x < 0 \end{cases}$
- **Description**: ELU is similar to ReLU but includes a nonzero gradient for negative values, which helps reduce the vanishing gradient effect, leading to better performance and faster learning.

ELU: Outputs a smooth, non-linear response for negative inputs, with the output curving towards $-\alpha$ as x decreases.

Leaky ReLU: Provides a simple, linear response for negative inputs, proportionally scaled down by α.

Activation Functions

Sigmoid: The sigmoid function, also known as the logistic function, is a classical activation function used in neural networks, particularly in the output layer of binary classification models.

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Disadvantages: Vanishing Gradient Problem, Non-zero-centered Output, Computational Complexity is High.

Activation Functions

Softmax Function: The Softmax function is defined as follows for a vector **z** of raw class scores from the final layer of a model:

$$\text{softmax}(\mathbf{z})_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

where:

- z_i is the score for class i,
- e^{z_i} is the exponential function applied to z_i,
- the denominator is the sum of exponential scores for all classes in the vector $\mathbf{z}$.

Activation Functions

$$\text{softmax}(\mathbf{z})_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

Let's assume-
- Score for 1st Class: 2.0
- Score for 2nd Class: 1.0
- Score for 3rd Class: 0.5

- $e^{2.0} \approx 7.389$
- $e^{1.0} \approx 2.718$
- $e^{0.5} \approx 1.649$

Sum of exponentiated scores: $7.389 + 2.718 + 1.649 \approx 11.756$

Now, computing the probabilities:

- **Probability for Class 1:** $\frac{7.389}{11.756} \approx 0.628$
- **Probability for Class 2:** $\frac{2.718}{11.756} \approx 0.231$
- **Probability for Class 3:** $\frac{1.649}{11.756} \approx 0.140$

These probabilities sum to approximately 1.0, as expected.

Activation Functions

In regression problems where the goal is to predict a **continuous output** value, the activation function used in the output layer is typically a **linear activation function** or no activation function at all. The equation for the linear activation function, commonly used in the output layer of regression problems, is simply:

$$f(x) = x$$

Properties:

- **Input**: x (input value)
- **Output**: x (output is the same as the input)

This signifies that the output of the activation function is directly equal to the input value (x). It doesn't introduce any modification to the weighted sum of the inputs.

Activation Functions

Activation Function	Formula	Typical Usage
ReLU	$\text{ReLU}(x) = \max(0, x)$	Hidden layers (common for general purposes)
Sigmoid	$\sigma(x) = \frac{1}{1+e^{-x}}$	Output layer (binary classification)
Tanh	$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	Hidden layers (range between -1 and 1)
Softmax	$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$	Output layer (multi-class classification)
Leaky ReLU	$\text{LeakyReLU}(x) = \max(\alpha x, x)$	Hidden layers (variation of ReLU)

Neural Networks

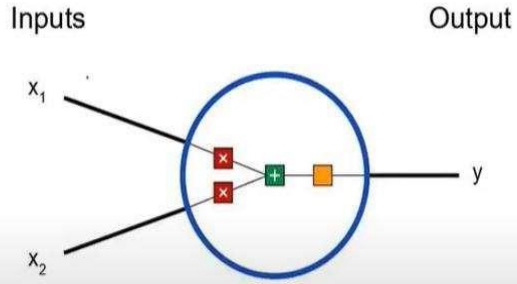
Epoch refers to one **complete pass** through the **entire training dataset**. During an epoch, the network will see every sample in the dataset once, allowing it to learn from the data by adjusting its weights based on the loss gradient.

- **Forward Pass:** Each input sample from the dataset is passed through the network to generate a prediction.
- **Loss Calculation:** The prediction is compared to the true value using a loss function, which quantifies the difference or error.
- **Backpropagation:** The gradient of the loss function is computed concerning each weight in the network. This involves applying the chain rule from calculus to find out how changes in weights affect the output error.
- **Weight Update:** The weights are then updated using an optimization algorithm, typically gradient descent or one of its variants (like Adam or RMSprop). The optimization algorithm adjusts the weights in a direction that minimizes the loss.
- **Repeat for All Batches:** Steps 1 to 4 are repeated for each batch in the training dataset until the entire
been processed.

How Does an Artificial Neuron Learn?

Neural Networks

First, we must talk about neurons, the basic unit of a neural network. A neuron takes inputs, does some math with them and produces one output. Here's what a 2-input neuron looks like:



In neural network training, the weights are typically initialized randomly and then adjusted durir process through backpropagation. Backpropagation is an iterative algorithm that updates the we network based on the error between the predicted output and the actual output. By minimizing network can learn to make more accurate predictions.

Neural Networks

Batch Size:

The **epochs=10** means that the neural network will train on the entire training dataset for **10 iterations**. During each epoch, the neural network will update its weights multiple times using backpropagation and stochastic gradient descent (or other optimization algorithms) until it has seen all the training examples. The number of weight updates during an epoch depends on the **batch_size**, which is another hyperparameter that determines how many samples are used to update the weights in each iteration.

For example: if we have a training dataset of 1000 samples and set the batch size to 100, the neural network will update its weights 10 times during an epoch (since 1000/100 = 10). During each weight update, the neural network will calculate the gradient of the loss function concerning the weights and use this gradient to adjust the weights in the direction that reduces the loss.

After 10 epochs, the neural network will have updated its weights 10 times on the entire training datase hopefully learned to make accurate predictions on new data.

Neural Networks

- Assume we have a 2-input neuron that uses the sigmoid activation function and has the following parameters:
  - w = [0, 1]
  - b = 0.5
- Now, let's give the neuron an input of x = [2, 3]. We'll use the dot product to write things more concisely:
  - (w.x) + b = ((w1*x1) + (w2*x2)) + b
    - = (0*2 + 1*3 + 0.5)
    - = 3.5

So, y = f ((x1*w1)+(x2*w2)+b)
=f(3.5)
~0.97

Note: $f(z) = \frac{1}{1+e^{-z}}$

$$f(3.5) = \frac{1}{1+e^{-3.5}} = \frac{1}{1+e^{-3.5}} \approx \frac{1}{1+0.0302} \approx \frac{1}{1.0302} \approx 0.9707$$

The neuron outputs given the inputs x=[2,3]. This process of passing inputs forward to get output is known as feedforward. That's it!

Neural Networks

- Assume we have a 2-input neuron that uses the sigmoid activation function and has the following parameters:
  - w = [0, 1]
  - b = 0.5
- Now, let's give the neuron an input of x = [2, 3]. We'll use the dot product to write things more concisely:
  - (w.x) + b = ((w1*x1) + (w2*x2)) + b
    - = (0*2 + 1*3 + 0.5)
    - = 3.5

So, y = f ((x1*w1)+(x2*w2)+b) #sigmoid
=f(3.5)
~0.97
~1

The neuron outputs given the inputs x=[2,3]. This process of passing inputs forward to get output is known as feedforward. That's it!

Image/Data Classification using RNN & CNN

Dr. Md. Zulfiker Mahmud
Professor, Department of CSE, Jagannath University
Email: zulfiker@cse.jnu.ac.bd

Convolution Neural Network (CNN)

Computer Vision Fundamentals

Computer vision is a field of artificial intelligence that enables computers and systems to derive meaningful information from **digital images**, **videos**, and **other visual inputs**, and to act or make recommendations based on that information.

The key aspects of computer vision, summarized as main points:

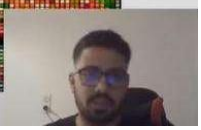
- **Image Recognition:** Identifying objects, people, and other elements within images.
- **Object Detection:** Recognizing and locating objects within an image using bounding boxes or other markers.
- **Image Segmentation:** Dividing an image into parts to simplify the analysis, often used in applications like medical imaging.
- **Pattern Recognition:** Recognizing patterns in visual data, such as shapes or movements.
- **Scene Reconstruction:** Reconstructing a 3D scene from images, used in augmented reality and robotics.
- **Video Tracking:** Tracking objects or individuals across a video sequence.
- **Image Restoration:** Restoring or enhancing the quality of degraded images.

Image Concepts

Image: In deep learning and computer vision, an image is a **digital representation of visual information**—like a photograph or a frame from a video—formatted in a way that can be processed by algorithms. The total number of pixels that represent the image is 1024x768 = 7,86,432. From **black at the lowest value (0)** to **white** at the **highest value (255)**, with shades of gray in between proportional to the value.

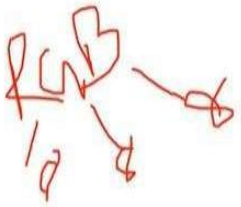
Image Size: 1024×768

Image Size: 1024×768



Pixel Concepts

A pixel, short for "picture element," is the **smallest unit of information in an image** or display. Each pixel represents a single point in the image. Pixels are arranged in a **grid pattern**. When viewed together at the proper distance, they form the complete image that our eyes perceive. An image with a resolution of 1920x1080 has 2,073,600 pixels.



24-bit Color (True Color):

- **Usage**: Most commonly used in JPEG images, web graphics, television screens, computer monitors, and smartphones.
- **Advantages**: Offers a good balance between color variety and file size, providing sufficient color depth for most practical applications without consuming as much storage or bandwidth as higher bit depths.

The term "True Color" is used to describe a color depth that allows for a representation of 24-bit color information. This configuration provides a total of 16,777,216 color variations. In True Color, each color component (Red, Green, and Blue) is allocated 8 bits, which allows each component 256 different intensity levels, leading to 256×256×256=16,777,216 possible colors.

Pixel Concepts

A pixel, short for "picture element," is the **smallest unit of information in an image** or display. Each pixel represents a single point in the image. Pixels are arranged in a **grid pattern**. When viewed together at the proper distance, they form the complete image that our eyes perceive. An image with a resolution of 1920x1080 has 2,073,600 pixels.

8-bit Grayscale:

For tasks where color information **might not add significant value**, converting images to **8-bit grayscale** can reduce computational requirements and streamline the learning process. This is common in tasks like text recognition, certain types of medical imaging, or when training efficiency is a priority.

Pixel Concepts

Color Representation:

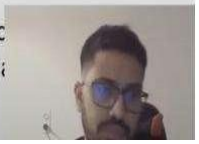
- **8-bit Grayscale**: These images contain only shades of gray, meaning that each pixel carries information only about intensity or luminance, without any color data. The values range from 0 (black) to 255 (white), providing a total of 256 different shades of gray.

- **24-bit True Color**: True Color images use three color channels (Red, Green, and Blue), with each channel represented by 8 bits. This setup allows each channel to display 256 different intensity levels, combining to offer over 16 million possible colors (256 x 256 x 256).

Data Complexity and Size:

- **8-bit Grayscale**: In an 8-bit grayscale image, there is only one channel. These images are simpler and require less storage space and bandwidth because they consist of a single channel. An 8-bit grayscale image uses one byte per pixel.

- **24-bit True Color**: In a 24-bit color image, there is a total of three channels. These images are more c requiring three times the data for each pixel compared to grayscale images (since there are three cha using one byte). This makes them larger and more resource-intensive to process and store.



Pixel Concepts

1-bit (Monochrome):

- **Value Range:** 0-1
- **Colors:** Black and White (2 colors)

2-bit color:

- **Value Range:** 0-3
- **Colors:** 4 different colors or shades

3-bit color:

- **Value Range:** 0-7
- **Colors:** 8 different colors or shades

4-bit color:

- **Value Range:** 0-15
- **Colors:** 16 different colors or shades (often seen in early computer graphics)

Pixel Concepts

48-bit color (used in professional photography):

- **Value Range per Channel**: 0-65535
- **Total Colors**: This effectively allows trillions of different color nuances (important in high-end photography and printing).

64-bit color (HDR and advanced graphics):

- **Value Range per Channel**: 0-65535 for RGB and alpha
- **Total Colors**: This provides deep color with high dynamic range capabilities.

Working with Videos

❖ **Frame:** A "frame" in video and animation refers to a single still image in a sequence of images that constitute a video or animated content. When multiple frames are displayed rapidly, they create the illusion of motion.

❖ **Frame Rate:** It is measured in frames per second (FPS) and indicates the frequency at which these frames are displayed or projected.

Standard Frame Rates:

- **24 FPS**: Traditionally used in cinema. It provides a film-like motion blur that feels natural due to its closeness to how the human eye perceives movement.
- **30 FPS**: Common in television broadcasting in the NTSC (National Television System Committee) format. It offers slightly smoother motion compared to 24 FPS.
- **25 FPS**: Standard for television in countries using the PAL (Phase Alternating Line) and SECAM (Sequential Couleur Avec Memoire) formats.
- **60 FPS and higher**: Used in high-definition TV, live sports broadcasting, video games, and we streaming to produce very smooth motion effects.

Working with Videos

1. Object Detection and Tracking:

- **Moderate to High FPS**: For tasks like real-time object tracking in video feeds or sports analytics, a higher frame rate (30-60 FPS) can be beneficial. It provides more temporal information and smoother transitions, which can improve the accuracy and robustness of tracking algorithms.

2. Action Recognition:

- **Variable FPS**: The necessary frame rate can vary depending on the speed of the actions being analyzed. Faster actions might require higher frame rates to capture relevant motion details effectively. Typically, 30 FPS is a good starting point, but more dynamic scenes might benefit from up to 60 FPS.

Working with Videos

3. Behavior Analysis:

- **Lower to Moderate FPS**: For general behavior analysis, such as monitoring crowd movements or analyzing consumer behavior in stores, lower frame rates (15-30 FPS) might be sufficient. These applications often prioritize broader trends over split-second details.

4. Surveillance:

- **Lower FPS Sufficient**: In surveillance, especially in scenarios with less dynamic scenes, lower frame rates (5-15 FPS) are often adequate. This helps save on storage and processing while still capturing enough temporal information for activity detection and person identification.

5. Medical Imaging and Analysis:

- **Specific FPS Requirements**: Certain medical applications, such as analyzing heart rate or other quick physiological changes, may require very specific frame rates defined by the speed of the physiological events.

Convolutional Neural Network (CNN)

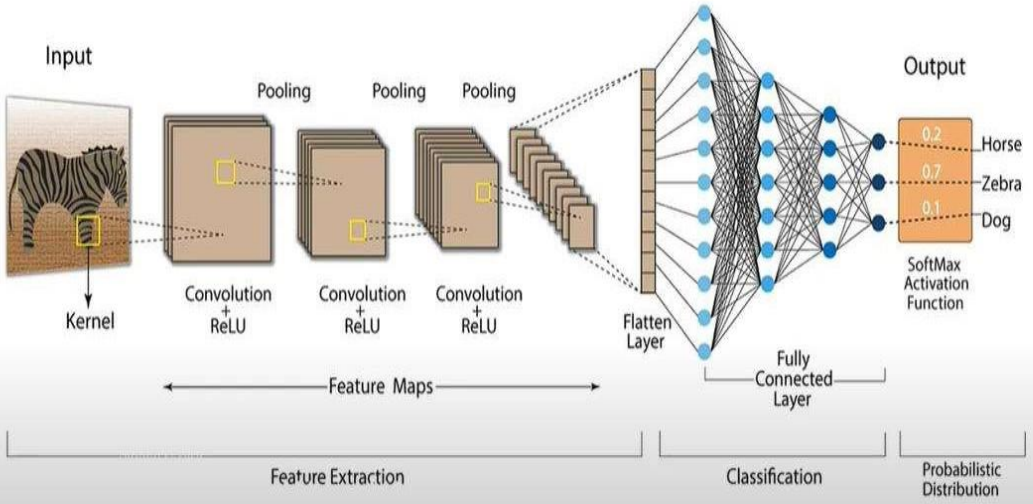


Fig: CNN Architecture

Convolutional Neural Network (CNN)

- **Input:** The process begins with an input image, which in this case is a **zebra.**
- **Padding**: Before applying the convolution operation, padding might be added around the border of the input image. Padding is used to **preserve the original size** of the image after convolution.
  - **Zero Padding (most common)**: Adds zeros around the border of the image.
  - **Reflective or Replication Padding**: Copies the edge values or reflects them around the border.
- **Convolution Layer:** To create feature maps, small matrices called **kernels** or filters are applied to the input image. These filters detect features such as edges and textures by sliding across the image. Each application of the filter results in a new feature map. (Next Slide)
- **ReLU Activation:** After convolution, the feature maps are passed through a Rectified Linear Unit (ReLU) activation function to introduce non-linearity, making the network capable of learning more complex patterns. It does this by turning all negative values to zero.
- **Pooling Layer:** After the ReLU activation, a pooling layer (typically max pooling) simplifies the output by reducing its dimensions but retaining the most important information. It selects the maximum value from a group of pixels in a feature map.

Convolutional Neural Network (CNN)

- **Flatten Layer:** After several convolution and pooling layers, high-level reasoning in the neural network occurs. The data is flattened into a single vector to prepare for the fully connected layer, which needs a flat input. (Next Slide)
- **Fully Connected Layer:** This layer is a deep neural network that uses the flattened data from the previous layers. Every neuron in this layer is connected to all the activations in the previous layer, and it's where the actual classification process begins to take shape.
- **Output Layer with SoftMax Function:** The last step in the process involves the SoftMax activation function, which is applied in the output layer. This function converts the output into a probability distribution, representing the likelihood of the input image belonging to one of the predefined classes (like horse, zebra, or dog).
- **Classification Output:** The output is a probabilistic distribution where each class (type of animal in this case) is assigned a probability score, indicating the confidence of the network in predicting each class.

More About Convolution Layer in CNN

Suppose, your input image has **three** channels (such as an RGB image), and you use a single filter, you **do not** get three separate feature maps for each channel. Instead, you get a **single feature map** from the convolution operation. This is how it works:

- **Multi-channel Filter:** Each filter in a convolution layer is three-dimensional when working with multi-channel images like RGB images. The depth of the filter matches the number of channels in the input image. So, for an RGB image, each filter has three layers—one for each channel (Red, Green, Blue).
- **Convolution Operation:** During the convolution operation, each layer of the filter is applied to the corresponding channel of the image. This means the first layer of the filter processes the Red channel, the second layer processes the Green channel, and the third layer processes the Blue channel.
- **Summation:** The results of these three convolutions (one per channel) are then summed up to produce a single output value. This summation results in a single feature map per filter, even though the filter is applied to multiple channels.

Convolutional Neural Network (Part-02)

Topics: Convolution & ReLU

Convolutional Neural Network (CNN)

The output computation now only depends on a subset of inputs:

$$y_{11} = w_{11}x_{11} + w_{12}x_{12} + w_{13}x_{13} + w_{21}x_{21} + w_{22}x_{22} + w_{23}x_{23} + w_{31}x_{31} + w_{32}x_{32} + w_{33}x_{33}$$

$$O = \frac{M-F+2P}{S} + 1$$

x_{11}	x_{12}	x_{13}	x_{14}	x_{15}
x_{21}	x_{22}	x_{23}	x_{24}	x_{25}
x_{31}	x_{32}	x_{33}	x_{34}	x_{35}
x_{41}	x_{42}	x_{43}	x_{44}	x_{45}
x_{51}	x_{52}	x_{53}	x_{54}	x_{55}

Input Image (5*5)

$*$

w_{11}	w_{12}	w_{13}
w_{21}	w_{22}	w_{23}
w_{31}	w_{32}	w_{33}

Kernel or Filter (3*3)

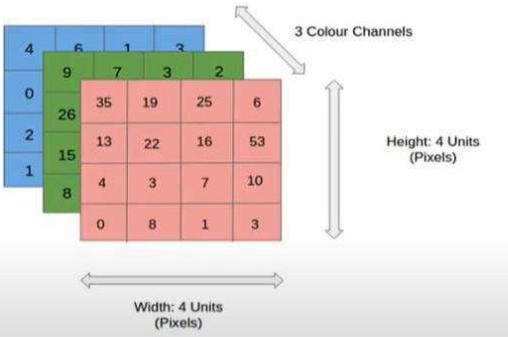
$=$

y_{11}		

Feature Map (3*3)

Input Layer

Fig: RGB Image

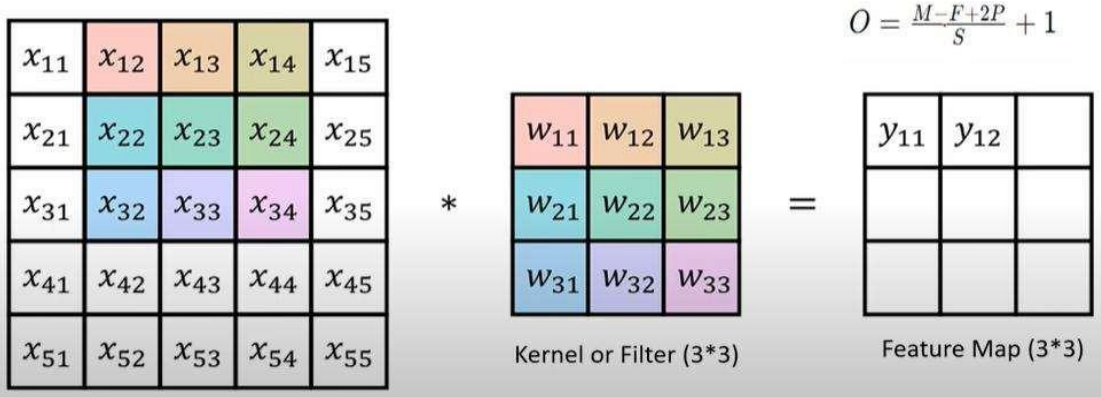


Convolutional Neural Network (CNN)

The output computation now only depends on a subset of inputs:

$$y_{12} = w_{11}x_{12} + w_{12}x_{13} + w_{13}x_{14} + w_{21}x_{22} + w_{22}x_{23} + w_{23}x_{24} + w_{31}x_{32} + w_{32}x_{33} + w_{33}x_{34}$$

$$O = \frac{M-F+2P}{S} + 1$$

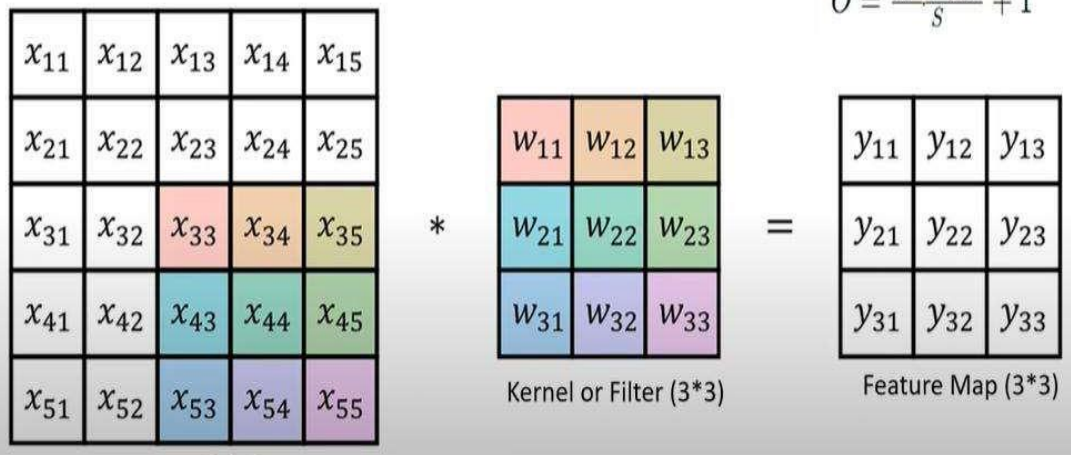


Convolutional Neural Network (CNN)

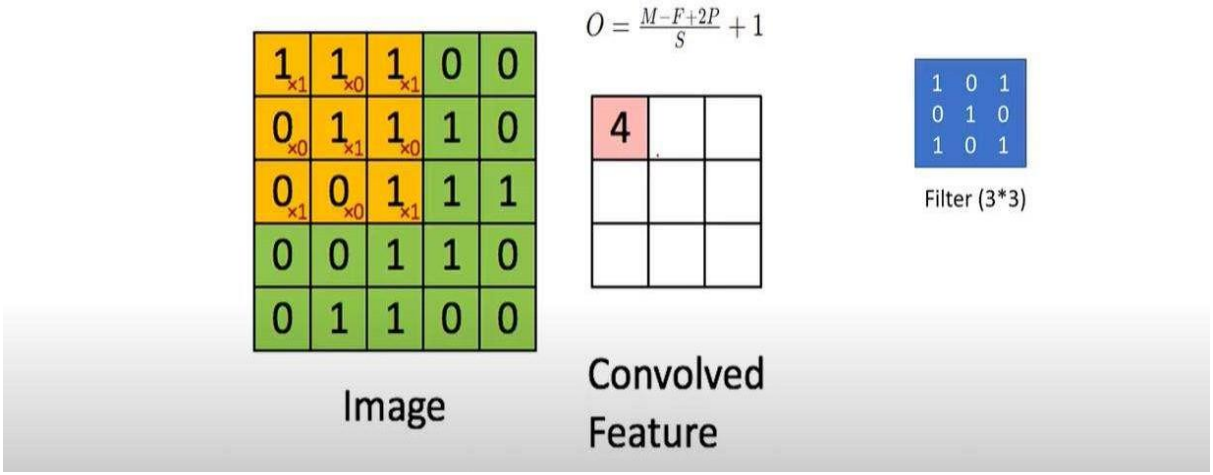
The output computation now only depends on a subset of inputs:

$$y_{33} = w_{11}x_{33} + w_{12}x_{34} + w_{13}x_{35} + w_{21}x_{43} + w_{22}x_{44} + w_{23}x_{45} + w_{31}x_{53} + w_{32}x_{54} + w_{33}x_{55}$$

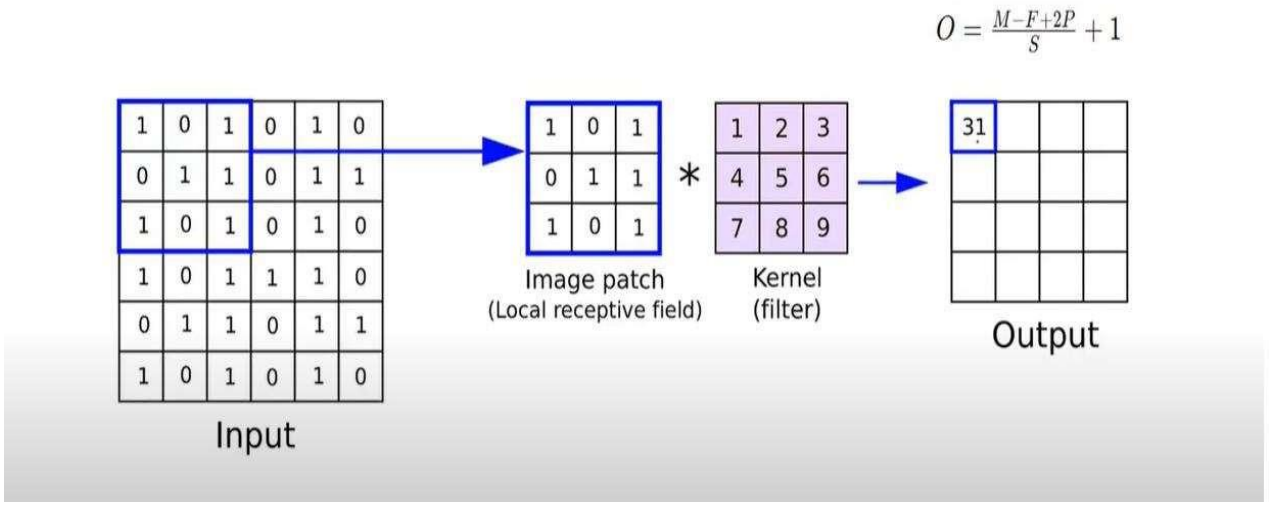
$$O = \frac{M-F+2P}{S} + 1$$



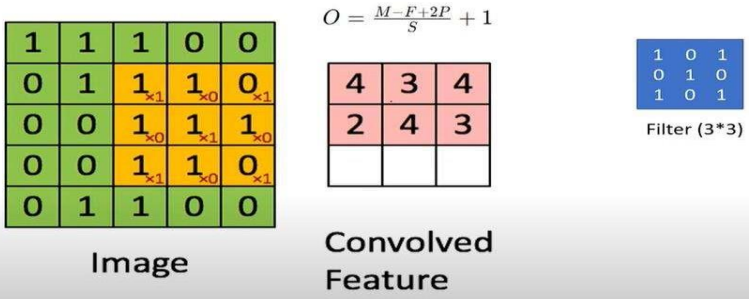
Convolutional Neural Network (CNN)



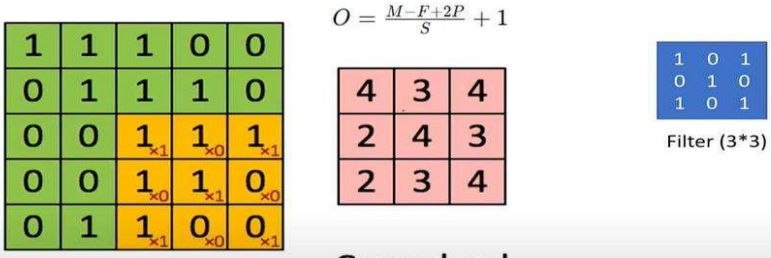
Convolutional Neural Network (CNN)



Convolutional Neural Network (CNN)



Convolutional Neural Network (CNN)



Convolutional Neural Network (CNN)

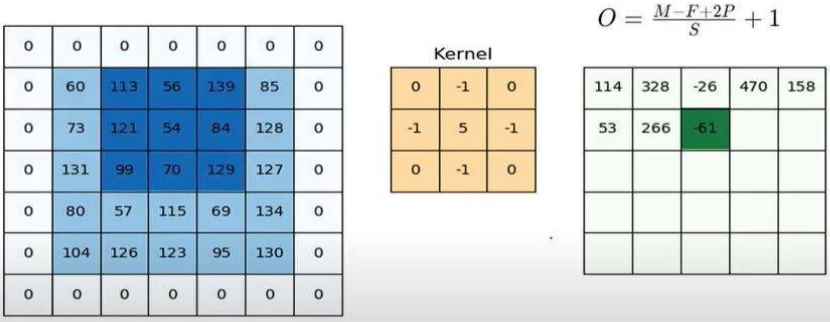


Fig: Zero-Padding in CNN

Convolutional Neural Network (CNN)

0	0	0	0	0	0	0
0	60	113	56	139	85	0
0	73	121	54	84	128	0
0	131	99	70	129	127	0
0	80	57	115	69	134	0
0	104	126	123	95	130	0
0	0	0	0	0	0	0

Kernel

0	-1	0
-1	5	-1
0	-1	0

$$O = \frac{M-F+2P}{S} + 1$$

114	328	-26	470	158
53	266	-61	-30	344
403	116	-47	295	244
108	-135	256	-128	344

Fig: Zero-Padding in CNN

Convolutional Neural Network (CNN)

Input

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

★

Kernel

0	-1	0
-1	5	-1
0	-1	0

Bias

+ 5 =

Output

$$O = \frac{M-F+2P}{S} + 1$$

11	12
15	16

★ Cross-Correlation

1 X 0 + 5 X -1 + 9 X 0 + 2 X -1 + 6 X 5 + 10 X -1 + 3 X 0 + 7 X -1 + 11 X 0 + 5 = 11

Convolutional Neural Network (CNN)

Input

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

★

Kernel

0	-1	0
-1	5	-1
0	-1	0

Bias

+ 5 =

Output

$$O = \frac{M-F+2P}{S} + 1$$

11	12
15	16

★ Cross-Correlation

5 X 0 + 9 X -1 + 13 X 0 + 6 X -1 + 10 X 5 + 14 X -1 + 7 X 0 + 11 X -1 + 15 X 0 + 5 = 15

Input

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

★

Kernel

0	-1	0
-1	5	-1
0	-1	0

Bias

+ 5 =

Output

$$O = \frac{M-F+2P}{S} + 1$$

11	12
15	16

★ Cross-Correlation

1 X 0 + 5 X -1 + 9 X 0 + 2 X -1 + 6 X 5 + 10 X -1 + 3 X 0 + 7 X -1 + 11 X 0 + 5 = 11

-1	-1	1
0	1	-1
0	1	1

Kernel Channel #1

1	0	0
1	-1	-1
1	0	-1

Kernel Channel #2

0	1	1
0	1	0
1	-1	1

Kernel Channel #3

308 + −498 + 164 + 1 = −25

Bias = 1

Output

- **Prepare Input:** The input matrix includes pixel values or feature values arranged in a grid.
- **Position the Kernel:** The kernel (or filter) is a smaller matrix that you slide over the input matrix. Start at the **top-left** corner of the input and move the kernel over every valid position.
- **Perform Element-wise Multiplication:** For each position of the kernel, multiply each element of the kernel with the corresponding element of the input matrix that it covers.
- **Sum the Results:** After multiplying, sum up all the products obtained in the previous step to get a single output.
- **Adding Bias:** Add a bias term to the scalar value from the previous step. The bias is a single value that is learned during the training of the network. Adding a bias can help the model learn better by adjusting the output.
- **Generate Output Matrix:** The scalar result from each position of the kernel (after adding the bias) forms one element of the output matrix (or feature map).
- **Repeat for Multiple Filters:** If the CNN uses multiple filters, repeat steps 2-7 for each filter to produce multiple output matrices. Each filter can detect different features in the input, such as edges, textures, or other patterns.

Calculating Output Dimensions:

To find the dimensions of the output feature map (width and height), you can use the following formulas:

1. **Output Height (OH):**

$$OH = \left\lfloor \frac{H + 2P - F}{S} + 1 \right\rfloor$$

2. **Output Width (OW):**

$$OW = \left\lfloor \frac{W + 2P - F}{S} + 1 \right\rfloor$$

Here,
W: The width of the input volume.
H: The height of the input volume.
D: The depth of the input volume (number of input channels).
K: The number of filters.
F: The spatial size (height and width) of the filter/kernel.
P: The amount of padding applied to the width and height of the input.
S: The stride, which is the step size the filter moves across the input.

3. **Output Depth (OD):**
   - The output depth is equal to the number of filters K used in the convolutional layer.

Suppose you have an input volume of size 32×32×3 (width x height x depth), and you apply a convolutional layer with the following parameters:

- Filter size: 5×5
- Padding: 1 (applied to all sides)
- Stride: 1
- Number of filters: 10

Using the formulas, you would calculate the output dimensions as follows:

- **Output Height**:

$$OH = \left\lfloor \frac{32 + 2 \times 1 - 5}{1} + 1 \right\rfloor = 29$$

- **Output Height:**

$$OH = \left\lfloor \frac{32 + 2 \times 1 - 5}{1} + 1 \right\rfloor = 29$$

- **Output Width:**

$$OW = \left\lfloor \frac{32 + 2 \times 1 - 5}{1} + 1 \right\rfloor = 29$$

- **Output Depth:** 10 (since there are 10 filters)

So, the output feature map would have dimensions 29×29×10.

1	14	-9	4
-2	-20	10	6
-3	3	11	1
2	54	-2	80

Fig: Feature Map (**Before** ReLU)

ReLU

1	14	0	4
0	0	10	6
0	3	11	1
2	54	0	80

Fig: Feature Map (**After** ReLU)

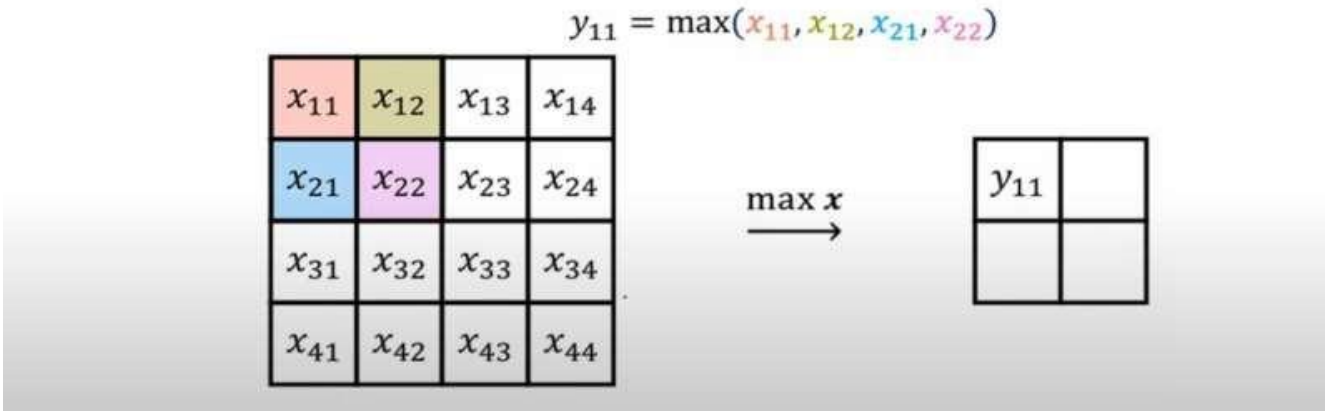
CNN: Pooling Layer

In a convolutional neural network (CNN), the pooling layer plays a crucial role in **reducing** the spatial dimensions (width and height) of the input volume for the subsequent layers.

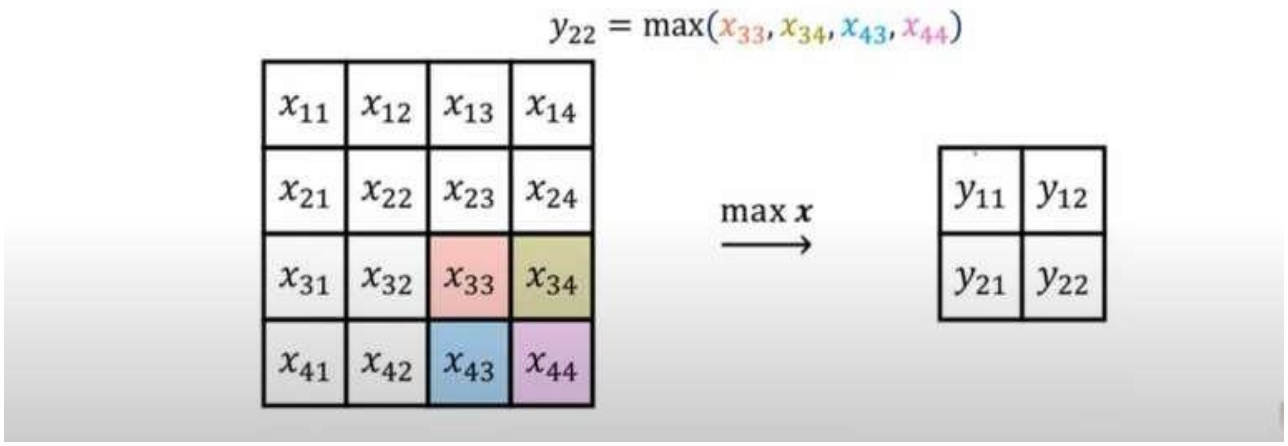
The most common types of pooling are:

- **Max Pooling:** Outputs the maximum value from each patch of the feature map.
- **Average Pooling:** Outputs the average value from each patch.

The pooling operation is introduced to reduce the number of computations in a CNN. From a theoretical perspective, higher representations do not require high spatial resolution.



The pooling operation is introduced to reduce the number of computations in a CNN. From a theoretical perspective, higher representations do not require high spatial resolution.



The pooling operation is introduced to reduce the number of computations in a CNN. From a theoretical perspective, higher representations do not require high spatial resolution.

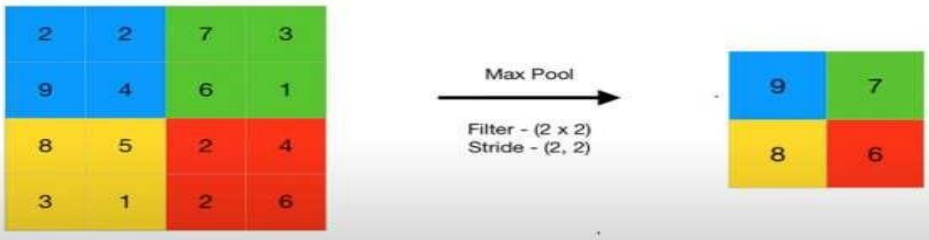


Fig: Max Pooling in CNN

The pooling operation is introduced to reduce the number of computations in a CNN. From a theoretical perspective, higher representations do not require high spatial resolution.

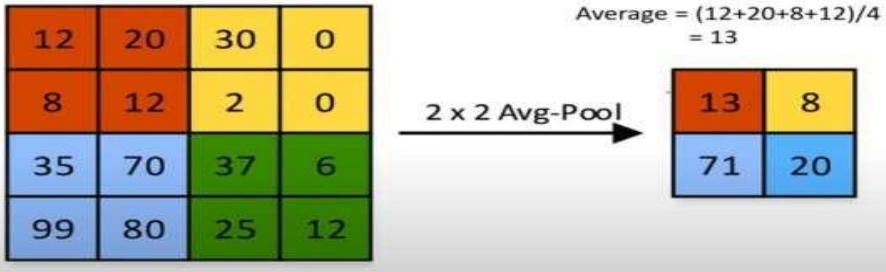


Fig: Average Pooling in CNN

More About Flatten Layer in CNN

Suppose, you will provide a **20x20** 2D data array to a **flatten layer**, it will convert this 2D array into a 1D vector as 400 individual elements.

How does it work when you use 100 neurons for a 400-element 1D input vector?

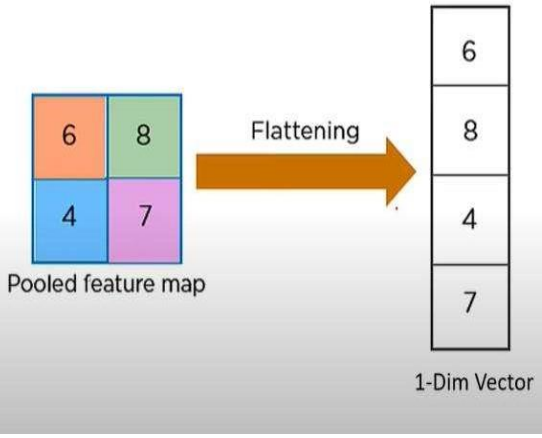
- **All Connections:** Each of the neurons is connected to every element of the input vector. For example, 100 neurons would each connect to all 400 elements from the input.
- **Weighted Sum and Bias:** Every connection has a weight. Each neuron computes a sum of the input elements multiplied by these weights, plus a bias term. This integrates information from all 400 points.

Mathematically: $\sum_{j=1}^{400} w_{ij} \cdot x_j + b_i$, where b_i is the bias for neuron i.

- **Activation Function:** This sum is then processed through an activation function, adding non-linearity and allowing the network to learn complex patterns.
- **The output of the Layer:** The output is a new vector, the size of which equals the number of neurons (for example 100). This output represents a condensed form of the input, capturing essential features.

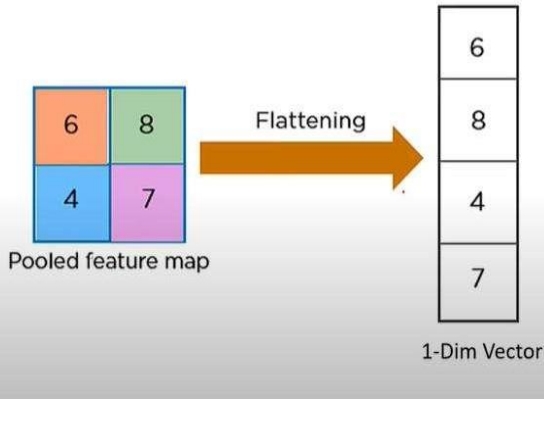
CNN: Flatten Layer

The flatten layer typically appears after the convolutional and pooling layers in convolutional neural network (CNN) architectures. The main **responsibility** of the flatten layer in a convolutional neural network (CNN) is to **reshape the multi-dimensional output** from the preceding convolutional or pooling layers into a **single, one-dimensional vector**.



CNN: Flatten Layer

The flatten layer typically appears after the convolutional and pooling layers in convolutional neural network (CNN) architectures. The main **responsibility** of the flatten layer in a convolutional neural network (CNN) is to **reshape the multi-dimensional output** from the preceding convolutional or pooling layers into a **single, one-dimensional vector**.



CNN: Dense Layer

Question: If a convolutional neural network has a flatten layer that receives a **10x10 dimensional feature map**, resulting in 100 flattened values, how are these values processed in the subsequent fully connected layer? Specifically, does each neuron in the next layer receive all 100 values, or does each value get assigned to a different neuron?

Answer: In a fully connected layer that follows a flatten layer in a convolutional neural network, **each neuron in the fully connected layer receives all 100 values** from the flattened 10x10 dimensional feature map. The fully connected layer operates such that each neuron is connected to every single input from the previous layer (in this case, the 100 flattened values).

CNN: Dimensions

Output dimensions describe the **size of the data tensor that results from a layer** within a CNN. This typically includes:

- **Width** and **Height**: These dimensions can change based on the type of layer (convolutional, pooling), the kernel size, the stride, and the padding used.
- **Depth** (or Channels): This dimension often changes in convolutional layers depending on the number of filters used. It stays the same through pooling layers unless pooling is done separately across channels.

Question: Consider the input tensor of dimensions 10×10×10 where the last dimension is the channel dimension. The following operations are applied:

1. 3×3 convolution (40 channels) with stride 1 and padding 1 for each dimension.
2. ReLU activation.
3. 3×3 max pooling with stride 1 and padding 1 for each dimension.
4. 3×3 convolution (20 channels) with stride 1 and padding 1 for each dimension.
5. ReLU activation.
6. 2×2 max pooling with stride 2 and padding 1 for each dimension.

What are the dimensions of the output tensor after each operation? Report the name of the operation and the output dimensions as width x height x channels. There are 6 operations in total.

1. 3×3 Convolution (40 channels) with stride 1 and padding 1

- **Kernel Size:** 3×3
- **Stride:** 1
- **Padding:** 1
- **Input Dimensions:** $10 \times 10 \times 10$

$$\text{Output Width} = \left\lfloor \frac{10-3+2\times1}{1} \right\rfloor + 1 = 10$$
$$\text{Output Height} = \left\lfloor \frac{10-3+2\times1}{1} \right\rfloor + 1 = 10$$
$$\text{Output Channels} = 40$$

Dimension: 10 x 10 x 40

2. ReLU Activation

- Does not change dimensions.
- **Output Dimensions:** $10 \times 10 \times 40$

3. 3×3 Max Pooling with stride 1 and padding 1

- **Kernel Size:** 3×3
- **Stride:** 1
- **Padding:** 1

$$\text{Output Width} = \left\lfloor \frac{10-3+2\times1}{1} \right\rfloor + 1 = 10$$
$$\text{Output Height} = \left\lfloor \frac{10-3+2\times1}{1} \right\rfloor + 1 = 10$$
$$\text{Output Channels} = 40 \text{ (Channel dimension remains unchanged in pooling)}$$

Dimension: 10 x 10 x 40

4. 3×3 Convolution (20 channels) with stride 1 and padding 1

- **Kernel Size**: 3×3
- **Stride**: 1
- **Padding**: 1
- **Input Dimensions**: $10 \times 10 \times 40$

$$\text{Output Width} = \left\lfloor \frac{10-3+2\times1}{1} \right\rfloor + 1 = 10$$
$$\text{Output Height} = \left\lfloor \frac{10-3+2\times1}{1} \right\rfloor + 1 = 10$$
$$\text{Output Channels} = 20$$

Dimension: 10 x 10 x 20

5. ReLU Activation

- Does not change dimensions.
- **Output Dimensions**: $10 \times 10 \times 20$

6. 2×2 Max Pooling with stride 2 and padding 1

- **Kernel Size**: 2×2
- **Stride**: 2
- **Padding**: 1

$$\text{Output Width} = \left\lfloor \frac{10-2+2\times1}{2} \right\rfloor + 1 = 6$$
$$\text{Output Height} = \left\lfloor \frac{10-2+2\times1}{2} \right\rfloor + 1 = 6$$
$$\text{Output Channels} = 20 \text{ (Channel dimension remains unchanged in pooling)}$$

Dimension: 6 x 6 x 20

$$\text{Output Dimension} = \left\lfloor \frac{\text{Input Dimension} - \text{Kernel Size} + 2 \times \text{Padding}}{\text{Stride}} \right\rfloor + 1$$

Dimensions After Each Operation:

1. **After 3×3 Convolution**: $10 \times 10 \times 40$
2. **After ReLU Activation**: $10 \times 10 \times 40$
3. **After 3×3 Max Pooling**: $10 \times 10 \times 40$
4. **After 3×3 Convolution**: $10 \times 10 \times 20$
5. **After ReLU Activation**: $10 \times 10 \times 20$
6. **After 2×2 Max Pooling**: $6 \times 6 \times 20$

CNN: Counting the Parameters

Parameters reflect the model's learning capacity and complexity. More parameters can mean a more powerful model, but also one that is more prone to overfitting and is computationally more expensive to train and run.

Note: In typical Convolutional Neural Network (CNN) architectures, the **convolutional layers** are primarily responsible for adding parameters, which are the learnable weights and biases of the model.

Question: Consider the input tensor of dimensions **10×10×10** where the last dimension is the channel dimension. The following operations are applied:

1. **3×3 convolution (40 channels) with stride 1 and padding 1 for each dimension.**
2. ReLU activation.
3. 3×3 max pooling with stride 1 and padding 1 for each dimension.
4. 3×3 convolution (20 channels) with stride 1 and padding 1 for each dimension.
5. ReLU activation.
6. 2×2 max pooling with stride 2 and padding 1 for each dimension.

What is the total parameter count?

$$\text{Number of Parameters} = (\text{Kernel Height} \times \text{Kernel Width} \times \text{Input Channels} + 1) \times \text{Output Channels}$$

First 3x3 Convolution (40 channels):

- **Kernel Height**: 3
- **Kernel Width**: 3
- **Input Channels**: 10 (initial input channels)
- **Output Channels**: 40

$$\text{Parameters} = (3 \times 3 \times 10 + 1) \times 40 = (90 + 1) \times 40 = 3640$$

$$\text{Number of Parameters} = (\text{Kernel Height} \times \text{Kernel Width} \times \text{Input Channels} + 1) \times \text{Output Channels}$$

Second 3x3 Convolution (20 channels):

- The input to this layer is the output of the first max pooling layer, which maintains 40 channels.
- **Output Channels**: 20

$$\text{Parameters} = (3 \times 3 \times 40 + 1) \times 20 = (360 + 1) \times 20 = 7220$$

Total Parameters

Now, to find the total number of parameters in the CNN:

$$\text{Total Parameters} = 3640(\text{First Conv}) + 7220(\text{Second Conv}) = 10860$$

Thus, the CNN has a total of 10,860 parameters, solely from the convolutional layers as pooling layers and ReLU activations do not add any learnable parameters.

Question: Consider the input tensor of dimensions **10×10×10** where the last dimension is the channel dimension. The following operations are applied:

1. 3×3 convolution (40 channels) with stride 1 and padding 1 for each dimension.
2. ReLU activation.
3. 3×3 max pooling with stride 1 and padding 1 for each dimension.
4. 3×3 convolution (20 channels) with stride 1 and padding 1 for each dimension.
5. ReLU activation.
6. 2×2 max pooling with stride 2 and padding 1 for each dimension.

What is the total parameter count?

If you are adding another convolutional layer with 10 output channels following the previous convolutional layer that had 20 output channels, you would again use the formula to calculate the parameters. This layer also uses 3x3 kernels:

Parameters for the Third Convolutional Layer:

- **Kernel Size**: 3×3
- **Input Channels**: 20 (output of the second convolution layer)
- **Output Channels**: 10
- **Bias**: 1 bias per output channel (10 in total)

Formula and Calculation:

$$\text{Number of Parameters} = (\text{Kernel Height} \times \text{Kernel Width} \times \text{Input Channels} + 1) \times \text{Output Channels}$$
$$\text{Number of Parameters} = (3 \times 3 \times 20 + 1) \times 10$$
$$\text{Number of Parameters} = (180 + 1) \times 10$$
$$\text{Number of Parameters} = 181 \times 10$$
$$\text{Number of Parameters} = 1810$$

Computer Vision

Computer vision is a field of artificial intelligence (AI) and computer science that focuses on enabling computers to interpret and understand **visual information** from the world. By processing and analyzing **images and videos**, computer vision systems aim to automate tasks that the human visual system can do.

Here are the main fields of application for computer vision:

1. Autonomous Vehicles
2. Medical Imaging
3. Surveillance and Security
4. Retail and E-commerce
5. Robotics
6. Augmented Reality (AR) and Virtual Reality (VR)
7. Agriculture
8. Manufacturing

Working Area	Purpose and Main Goal	Common Methods
Image Classification	Assign a label to an entire image.	CNNs (ResNet, VGG, etc.)
Object Detection	Identify and locate objects within an image.	Faster R-CNN, YOLO, SSD
Semantic Segmentation	Classify each pixel in the image into a category.	FCN, DeepLab, PSPNet
Instance Segmentation	Identify and describe each object instance in the image.	Mask R-CNN, SOLO
Panoptic Segmentation	Unified segmentation of both background and foreground objects.	Panoptic FPN, Panoptic-DeepLab
Image Generation and Synthesis	Generate new images from scratch or transform images.	GANs, VAEs
Image Reconstruction	Restore images from degraded or incomplete forms.	Super-resolution, De DnCNN

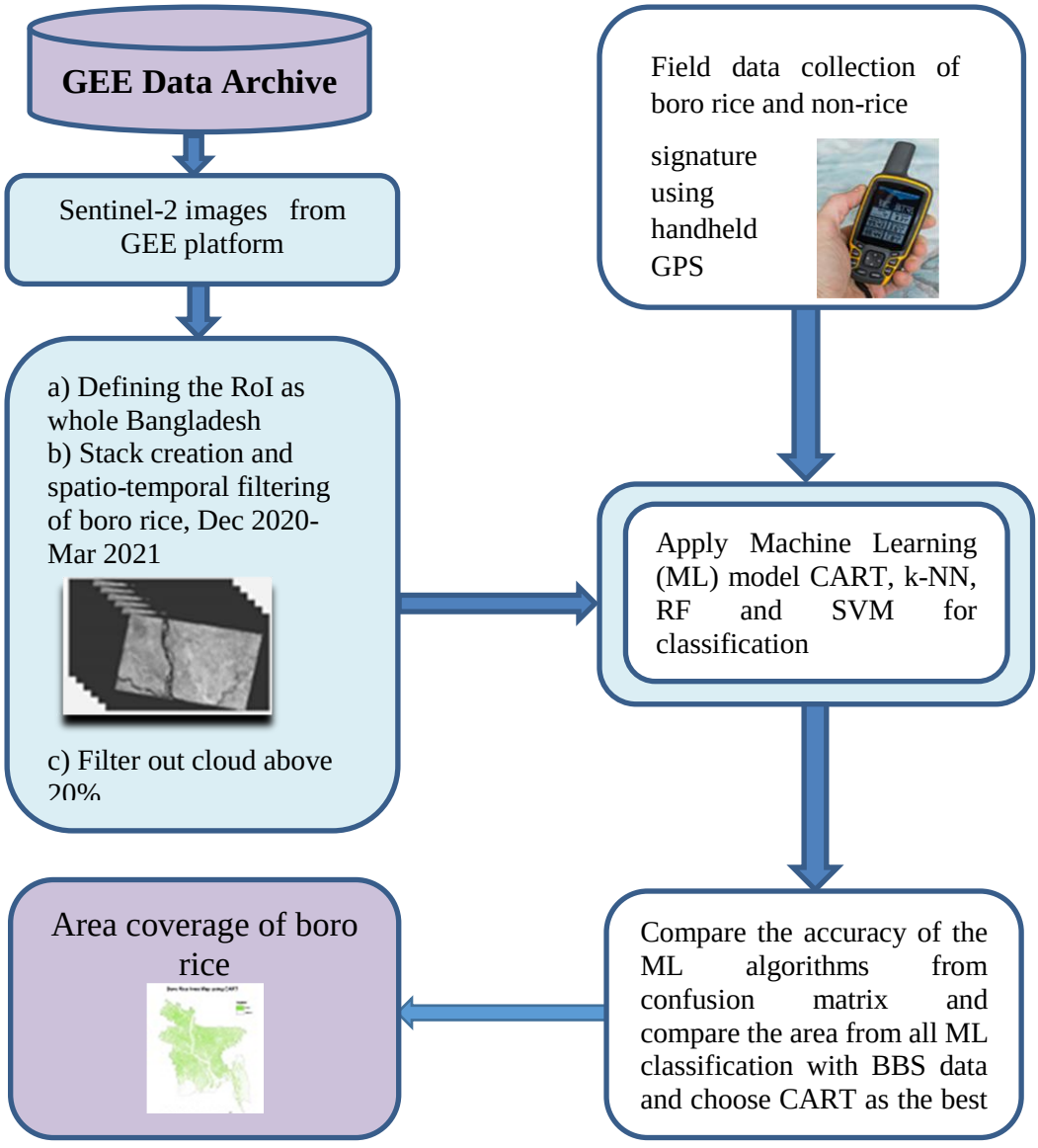
Working Area	Purpose and Main Goal	Common Methods
Image Retrieval	Find and retrieve images from a database based on a query.	Content-based image retrieval (CBIR), deep learning-based retrieval
Pose Estimation	Detect the pose or orientation of objects, often humans.	OpenPose, PoseNet
Action Recognition	Recognize and classify actions in videos or sequences of images.	3D CNNs, Two-Stream Networks
3D Computer Vision	Understand and interpret 3D structures from images or video.	Stereo vision, depth estimation, 3D reconstruction
Optical Character Recognition (OCR)	Recognize and extract text from images.	Tesseract, deep learning-based OCR
Visual Tracking	Track objects as they move across frames in a video.	KLT tracker, Siamese networks

Boro Rice Area Estimation with Google Earth Engine (GEE)

Hasan Md. Hamidur Rahman, Director (Computer & GIS Unit)
Bangladesh Agricultural Research Council
Email: h.rahman@barc.gov.bd

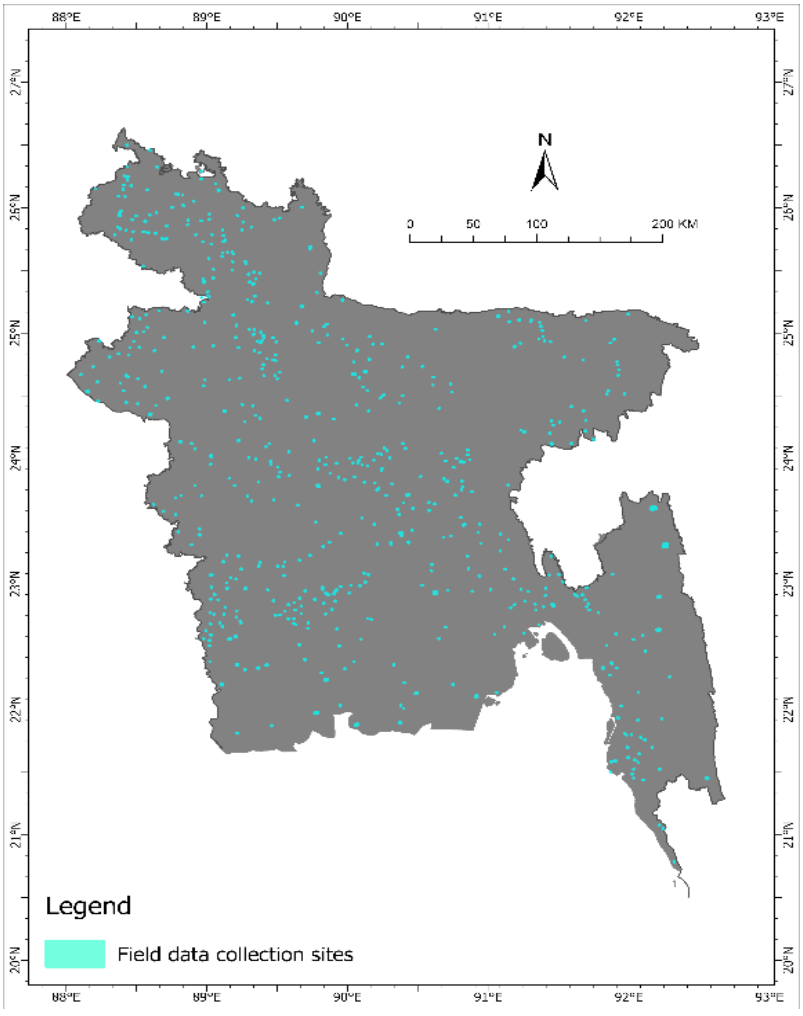
Steps for Boro rice area estimation

Flowchart of Boro rice area delineation:

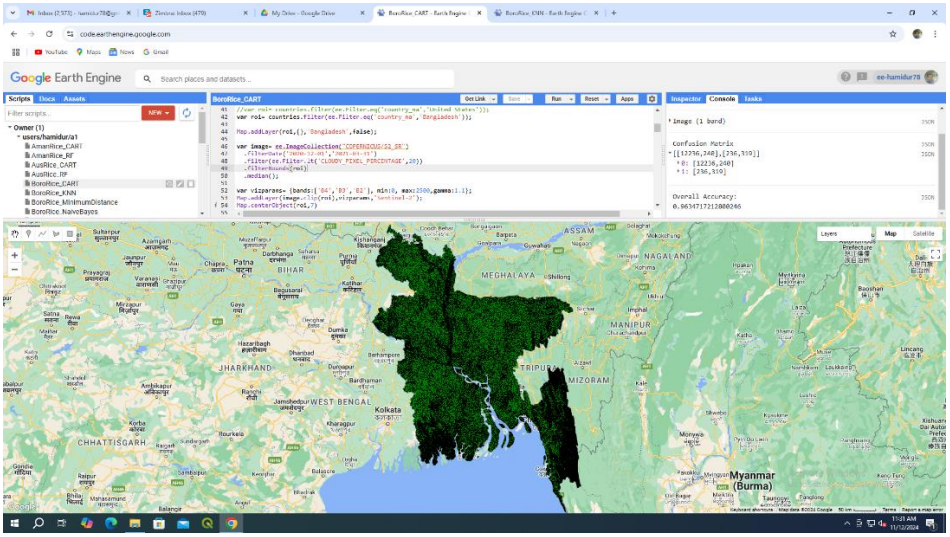


Field/ Signature data collection and processing

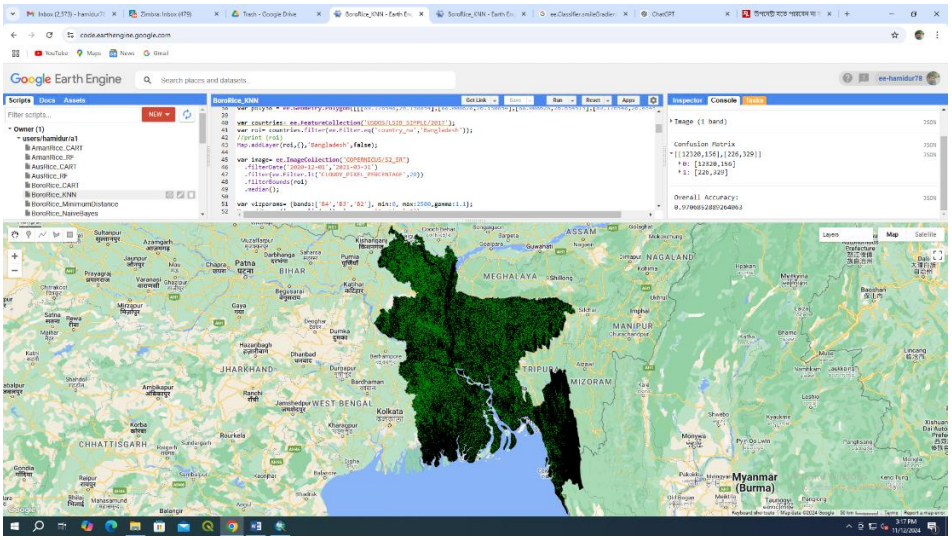
1. Field Data Collection through GPS device
2. The field data collected during boro seasons are compiled into an Excel sheet. The validated data is then saved as a CSV file, which serves as the attribute file.
3. Simultaneously, the GPX file from the handheld GPS devices is downloaded and imported into Google Earth Pro to verify the survey locations.
4. Polygons are drawn using the GPS points and digital photos for most of the fields.
5. The file is exported in KML format and opened in QGIS, where it is converted into a shapefile.
6. The shapefile is then joined with the attribute file (CSV) to merge the land types and other parameters.
7. Non-rice land types are generalized into a single class called 'non-rice'. In the shapefile, there are two columns named 'id' and 'feature_name', where the value '0' represents 'non-rice' and '1' represents ‘rice’ features.
8. The generalized shapefile is imported into the Google Earth Engine (GEE) asset, where it is used as the signature data in the machine learning (ML) model.



Screen shot of boro rice classified GEE raster data (CART)



Screen shot of boro rice classified GEE raster data (k-NN)



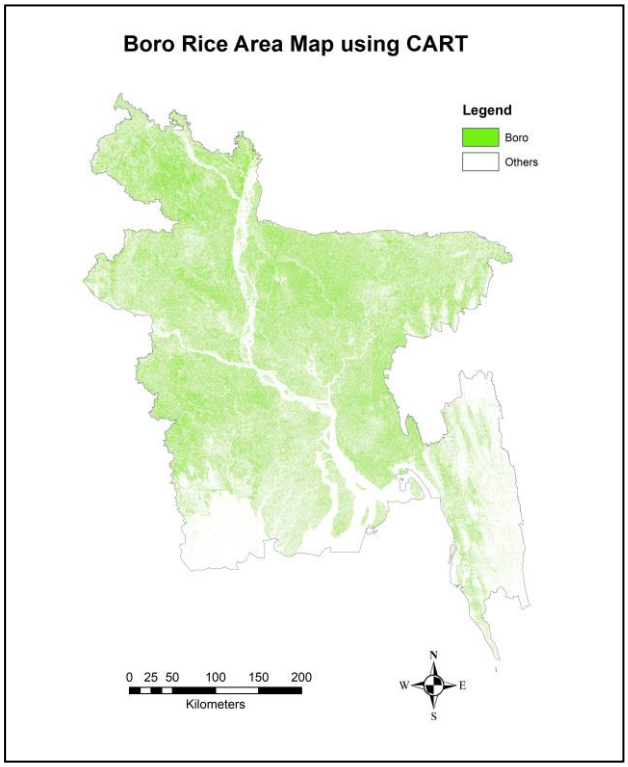
Steps to process data in ArcGIS

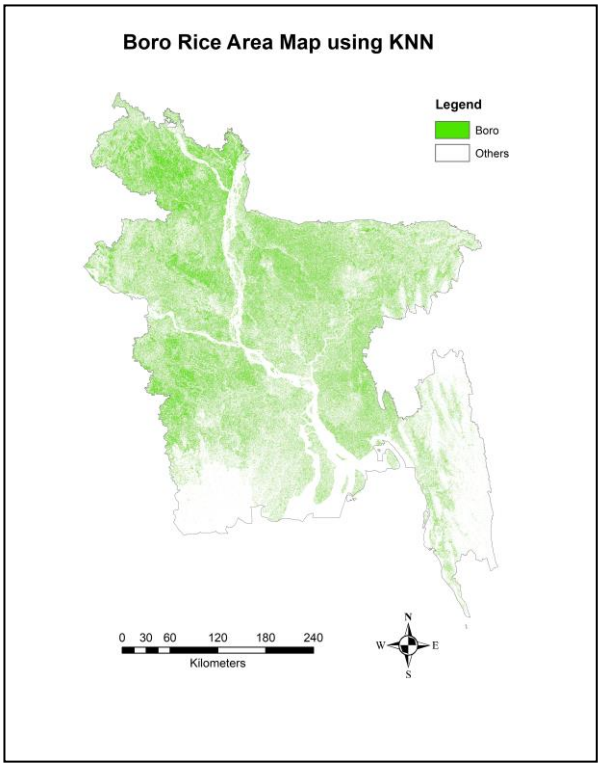
1. Boro Classified GEE Raster Data
2. Boro Reclassified Mosaiced Raster Data
3. Resampled Raster Data
4. Boro Final Area Result.

ArcGIS Tool for Raster data processing:

- Arc Tool Box-> Spatial Analysis Tools->Reclass->Reclassify
- Arc Tool Box-> Data Management Tools->Raster->Raster Dataset->Mosaic to New Raster
- Arc Tool Box-> Data Management Tools->Raster->Raster Processing->Resample

Boro rice area map





Boro rice area comparison

Sl.	Classification Algorithm	Pixel count (10m x 10m)	Area in Sq. Meter	Area in Sq. km	Area in Ha	Classification Accuracy of Algorithm (%)	Area Accuracy with respect to BBS (%)
1	**CART**	419957945	41995794500	41996	**4,199,579**	**96**	**88**
2	**k-NN**	384058603	38405860300	38406	**3,840,586**	97	**80**
				BBS Statistics 2020-21:	**4,786,621**	-	-

Conclusion

The integration of Machine Learning (ML) techniques in agriculture holds immense potential to transform the way we produce, manage, and forecast crop outcomes. This five-day training has introduced participants to essential ML concepts, data analysis tools, and real-world agricultural applications using Python and R programming. From foundational knowledge in data science and exploratory data analysis to hands-on experience with classification, regression, clustering, and ensemble learning methods such as Random Forests, participants have gained practical skills necessary for implementing intelligent decision-making systems in agriculture. The inclusion of neural networks and Google Earth Engine (GEE) has further empowered participants to explore advanced solutions such as image-based crop health detection and large-scale land monitoring. It is hoped that this training will not only strengthen participants' technical capabilities but also inspire them to apply these tools to address agricultural challenges in Bangladesh. By leveraging data-driven approaches, we can enhance crop productivity, ensure sustainability, and support policy-making that benefits farmers and the agricultural ecosystem.

The journey does not end here—continuous learning, experimentation, and collaboration will be key to driving meaningful change in the field of agricultural informatics.