CHAPTER 2

INTRODUCTION

2.1 Python Introduction

Python is basically a high-level programming language. It is a dynamic and free open-source language in nature. Moreover, it uses an interpreter for converting the source code into machine code. Furthermore, it supports both object-oriented programming as well as procedure-oriented programming. It is such a language that is highly readable and uses English keywords.

Python is a general purpose, high-level, interpreted programming language developed by Guido van Rossum in the late 1980s at the National Research Institute for Mathematics and Computer Science in the Netherlands.

Python is one of the most popular and widely used programming language used for set of tasks including console based, GUI based, web programming and data analysis. Python is a easy to learn and simple programming language so even if you are new to programming, you can learn python without facing any problems.

2.1.1 History

Guido Van Rossum developed Python in the middle of the eighties and nineties. He developed it at the National Research Institute for Mathematics and Computer Science in the Netherlands.

It is derived from many languages like C, C++, Algol-68, ABC, Unix shell, etc. Moreover, it is copyrighted.

- Python laid its foundation in the late 1980s.
- The implementation of Python was started in December 1989 by Guido Van Rossum at CWI in Netherland.
- In February 1991, Guido Van Rossum published the code (labeled version 0.9.0) to alt.source

- Python 2.0 added new features such as list comprehensions, garbage collection systems.

- On December 3, 2008, Python 3.0 (also called "Py3K") was released. It was designed to rectify the fundamental flaw of the language.

- ABC programming language is said to be the predecessor of Python language, which was capable of Exception Handling and interfacing with the Amoeba Operating System.

2.1.2 Features

Python provides many useful features which make it popular and valuable from the other programming languages. It supports object-oriented programming, procedural programming approaches and provides dynamic memory allocation. We have listed below a few essential features.

- **Easy to Learn and Use**

  Python is easy to learn as compared to other programming languages. Its syntax is straightforward and much the same as the English language. There is no use of the semicolon or curly-bracket, the indentation defines the code block. It is the recommended programming language for beginners.

- **Expressive Language**

  Python can perform complex tasks using a few lines of code. A simple example, the hello world program you simply type print("Hello World"). It will take only one line to execute, while Java or C takes multiple lines.

- **Interpreted Language**

  Python is an interpreted language; it means the Python program is executed one line at a time. The advantage of being interpreted language, it makes debugging easy and portable.

- **Cross-platform Language**

  Python can run equally on different platforms such as Windows, Linux, UNIX, and Macintosh, etc. So, we can say that Python is a portable language. It enables

programmers to develop the software for several competing platforms by writing a program only once.

- **Free and Open Source**

  Python is freely available for everyone. It is freely available on its official website www.python.org. It has a large community across the world that is dedicatedly working towards make new python modules and functions. Anyone can contribute to the Python community. The open-source means, "Anyone can download its source code without paying any penny."

- **Object-Oriented Language**

  Python supports object-oriented language and concepts of classes and objects come into existence. It supports inheritance, polymorphism, and encapsulation, etc. The object-oriented procedure helps to programmer to write reusable code and develop applications in less code.

- **Extensible**

  It implies that other languages such as C/C++ can be used to compile the code and thus it can be used further in our Python code. It converts the program into byte code, and any platform can use that byte code.

- **Large Standard Library**

  It provides a vast range of libraries for the various fields such as machine learning, web developer, and also for the scripting. There are various machine learning libraries, such as Tensor flow, Pandas, Numpy, Keras, and Pytorch, etc. Django, flask, pyramids are the popular framework for Python web development.

- **GUI Programming**

  Support Graphical User Interface is used for the developing Desktop application. PyQT5, Tkinter, Kivy are the libraries which are used for developing the web application.

- **Integrated**

  It can be easily integrated with languages like C, C++, and JAVA, etc. Python runs code line by line like C,C++ Java. It makes easy to debug the code.

- **Embeddable**

  The code of the other programming language can use in the Python source code. We can use Python source code in another programming language as well. It can embed other language into our code.

- **Dynamic Memory Allocation**

  In Python, we don't need to specify the data-type of the variable. When we assign some value to the variable, it automatically allocates the memory to the variable at run time. Suppose we are assigned integer value 15 to x, then we don't need to write int x = 15. Just write x = 15.

2.1.3 Applications of Python

Here is a list of popular applications of Python:

- **Web Development:** Using frameworks like Django, Flask, and FastAPI.
- **Data Science and Analytics:** For data manipulation and visualization with libraries like Pandas, NumPy, and Matplotlib.
- **Machine Learning and AI:** With powerful libraries like TensorFlow, PyTorch, and scikit-learn.
- **Game Development:** Utilizing libraries like Pygame.
- **Desktop GUI Applications:** Using Tkinter, PyQt, and Kivy.
- **Web Scraping:** Through tools like Beautiful Soup and Scrapy.
- **Automation/Scripting:** Writing scripts for task automation and system administration.
- **Network Programming:** This is used to build networked applications using libraries such as Socket and Twisted.

- **Cybersecurity and Penetration Testing:** Employed to write security tools and scripts.
- **Finance and Fintech:** For quantitative analysis, financial modeling, and algorithmic trading.
- **Blockchain Development:** Creating smart contracts and blockchain solutions.
- **Education:** Widely used as a teaching tool in computer science courses due to its simplicity.
- **Cloud Computing:** Employed in cloud environments for automation, serverless applications, and more.
- **Audio and Video Processing:** With libraries like OpenCV and PyDub.
- **Scientific Computing:** Using libraries like SciPy for complex mathematical and scientific tasks.
- **Chatbots and Conversational Interfaces:** Using libraries like ChatterBot.
- **E-commerce Solutions:** Backend development for online stores and inventory management systems.
- **Business Applications:** Developing enterprise software, ERP systems, and other business solutions.
- **IoT (Internet of Things):** Integrating with hardware and IoT platforms for smart applications.

2.2 Fundamentals of Python Programming

2.2.1 Python Syntax

In the simplest words, Syntax is the arrangement of words and phrases to create well-formed sentences in a language. In the case of a computer language, the syntax is the structural arrangement of comments, variables, numbers, operators, statements, loops, functions, classes, objects, etc. which helps us understand the meaning or semantics of a computer language

2.2.2 Comments

A comment is a part of the coding file that the programmer does not want to execute, rather the programmer uses it to either explain a block of code or to avoid the execution of a specific part of code while testing.

Single-Line Comments:

To write a comment just adds'#' at the start of the line.

Example:

#This is a 'Single-Line Comment

Multi-Line Comments

To write multi-line comments you can use '#' at each line or you can use the multiline string.

Example 1 : The use of '#'.

#It will execute a block of code if a specified condition is true. #If the codition is false than it will execute another block of code. **Example 2 : The use of Multiline String.**

"""This is an if-else statement.

It will execute a block of code if a specified condition is true.

If the condition is false then it will execute another block of code."""

2.2.3 Variables

Variables are containers that store information that can be manipulated and referenced later by the programmer within the code. In python, the programmer does not need to declare the variable type explicitly, we just need to assign the value to the variable.

In Python, a variable is a container for storing a value. You can create a variable by assigning a value to it using the "=" operator. The value can be of any data type, such as a string, integer, or floating-point number. Once a variable is assigned a value, it can be used throughout your program.

Example:

```
name = "L K Technologies"  #type str age = 20  #type int

passed = True        #type bool
```

It is always advisable to keep variable names descriptive and to follow a set of conventions while creating variables:

- Variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable name must start with a letter or the underscore character.
- Variables are case sensitive.
- Variable name cannot start with a number

2.2.4 Data Types

Python has several built-in data types, including strings, integers, and floating-point numbers. In Python, data types refer to the type of value a variable holds. Python has several built-in data types:

Python Data Types	Classes	Description
Numeric	int, float, complex	Holds numeric values
String	str	Holds sequence of characters
Sequence	list, tuple, range	Holds collection of items
Mapping	dict	Holds data in key-value pair form
Boolean	bool	Holds either True or False
Set	set, frozenset	Hold collection of unique items

In Python, numeric data type is used to hold numeric values. Integers, floating-point numbers and complex numbers fall under Python numbers category. They are defined as int, float and complex classes in Python.

Python List Data Type

List is an ordered collection of similar or different types of items separated by commas and enclosed within brackets [ ]. For example,

languages = ["Swift", "Java", "Python"]

Python Tuple Data Type

Tuple is an ordered sequence of items same as a list. The only difference is that tuples are immutable. Tuples once created cannot be modified.

In Python, we use the parentheses () to store items of a tuple. For example, product = ('Xbox', 499.99)

Python String Data Type

String is a sequence of characters represented by either single or double quotes. For example,

name = 'Python' print(name)

message = 'Python for beginners' print(message)

Python Set Data Type

Set is an unordered collection of unique items. Set is defined by values separated by commas inside braces { }. For example,

create a set named student_id student_id = {112, 114, 116, 118, 115} # display student_id elements print(student_id)

display type of student_id print(type(student_id))

Python Dictionary Data Type

Python dictionary is an ordered collection of items. It stores elements in key/value pairs. Here, keys are unique identifiers that are associated with each value.

Let's see an example,

```
# create a dictionary named capital_city

capital_city = {'Nepal': 'Kathmandu', 'Italy': 'Rome', 'England': 'London'}

 print(capital_city)
```

2.2.5 Input function

Programs usually request for some user input to serve its function (e.g. calculators asking for what numbers to use, to add/subtract etc.). In Python, we request user input using the input() function.

Syntax

```
message = input("This is where you type in your input request: ")
```

2.2.6 Python Operators

Python operators are symbols that perform an operation on one or more operands. An operand is a variable or a value on which we perform the operation.

Python operators are categorized in the following categories −

- Arithmetic Operators
- Comparison (Relational) Operators
- Assignment Operators
- Logical Operators
- Bitwise Operators
- Membership Operators
- Identity Operators

2.2.7 Control Statements

Control statements direct the flow of a Python program's execution. They introduce logic, decision-making, and looping. Key control statements in Python include if/else, for/while, break/continue, and pass.

Flow control in Python is the process of controlling and directing the progression of program execution by utilizing conditional statements, iterative loops, and reusable functions. It helps govern a program's logic and coherence, permitting code to execute only when certain predefined conditions have been satisfied. In particular, flow control allows a program to make decisions: thinking: and perform repetitive tasks.

Conditional statements like if-else tests enable a program to evaluate expressions and execute code only if certain conditions are true or false. Loops like for and while loops enable a program to reiterate through a block of code multiple times.

if Statement

if statement in Python evaluates whether a condition is true or false. It contains a logical expression that compares data, and a decision is made based on the result of the comparison.

Syntax of the if Statement

if expression:

statement(s) to be executed

Example

discount = 0

amount = 1200

Check he amount value if amount > 1000:

discount = amount * 10 / 100 print("amount = ", amount - discount)

if –else Statement

Condition checking and if-else statements are vital parts of programming because they control the flow of execution . The if-else statement in Python enables decision-making in code. It determines whether to execute a block of statements based on the result of condition check. The statements in the if block runs if the condition is true . Otherwise, the else block's statements run.

Syntax of if-else Statement

```
if condition:
    statement(s)
else:
    statement(s)
```

Example

```
credit_score = 720
if credit_score > 700:
    print("Congratulations, your loan application has been approved!")
else:
    print("We're sorry, your loan application has been rejected.")
```

if elif else Statement

The if-elif statement checks multiple conditions. If Test Expression1 is true, its body runs. Else, Test Expression2 is matched. If true, elif1 runs. Else, Test Expression 3 is checked. If true, elif2 runs. Else, . Any code below the if-elif statement then executes.

Syntax of Python if elif else Statement

```
if condition1:
    statement1
elif condition2:
    statement2
else:
    statement3
```

Example

```
# Get input from the user

hours_worked = int(input("Enter the number of hours worked: ")) # Define the hourly rate
and calculate the payment amount hourly_rate = 10  # $10 per hour

if hours_worked <= 0: payment_amount = 0

elif hours_worked <= 40:

    payment_amount = hours_worked * hourly_rate else:

# For hours worked above 40, apply 1.5 times the hourly rate for overtime overtime_hours =
hours_worked - 40

    payment_amount = 40 * hourly_rate + overtime_hours * hourly_rate * 1.5 # Display the
payment amount to the user

print("The payment amount is $", payment_amount)
```

Loops

Python loops allow us to execute a statement or group of statements multiple times.

In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on. There may be a situation when you need to execute a block of code several number of times.

Programming languages provide various control structures that allow for more complicated execution paths.

For loop

The for Loop in Python emphasizes over and navigates through a sequence (list, tuple, string) or other iterable objects. In other words, it rehashes the body of the Loop for each item within the sequence. More particularly, a for Loop permits you to execute a piece of code numerous times and is convenient after you have a piece of code that you need to rehash "n" a number of times.

```
if hours_worked <= 0:
    payment_amount = 0
    elif hours_worked <= 40:
        payment_amount = hours_worked * hourly_rate
        else:
    # For hours worked above 40, apply 1.5 times the hourly rate for overtime
    overtime_hours = hours_worked - 40
        payment_amount = 40 * hourly_rate + overtime_hours * hourly_rate * 1.5
        # Display the payment amount to the user
    print("The payment amount is $", payment_amount)
```

Loops

Python loops allow us to execute a statement or group of statements multiple times.

In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on. There may be a situation when you need to execute a block of code several number of times.

Programming languages provide various control structures that allow for more complicated execution paths.

For loop

The for Loop in Python emphasizes over and navigates through a sequence (list, tuple, string) or other iterable objects. In other words, it rehashes the body of the Loop for each item within the sequence. More particularly, a for Loop permits you to execute a piece of code numerous times and is convenient after you have a piece of code that you need to rehash "n" a number of times.

Syntax

for variable in sequence: statements

Example

looping through a list

numbers = [1,2,3,4,5]

for number in numbers:

print(number)

This code uses a for Loop to iterate through a list of numbers and print each one. The Loop specifies "number" as the iteration variable, referencing each item in the list. The Loop executes the code within it for each list item. Here, the code prints the iteration variable's value.

While Loops in Python

A while loop in Python continually executes a code block if a specified condition is true. The Loop will run the code block repeatedly until the condition becomes false. Once the condition is false, the program exits the while loop and continues.The condition for the Loop can be any expression that evaluates to true or false. True is any non-zero value. The Loop can also be terminated early using the break statement. While loops are a helpful control structure that allows a program to execute a set of statements multiple times. They run the code block within the Loop repeatedly until the defined condition is no longer met.

Syntax

while expression:

statement(s)

Example

Initialize counter counter = 0

While loop while counter < 10:

print(counter)

```
# Increment counter

 counter += 1
```

While Loop with Else

Let's further discuss while Loop with else. Let's check the syntax. A while loop with else executes a block of code as long as a given condition is true. Once the condition becomes false, the else block of code is executed. The else block of code is executed only if the condition is false. This is useful for ensuring that certain code is executed at least once or after the while loop ends.

Syntax

```
while <condition>:

    <code to execute> else:

<code to execute>
```

Example

```
i = 0

while (i <= 100 and i % 7 != 0):

i += 1

else:

if (i % 7 == 0):

  print("The first number divisible by 7 is", i) else:

print("No number is divisible by 7")
```

This code uses a while loop with an else statement to find the first number divisible by 7 between 0 and 100. The while loop checks if the value of i is less than or equal to 100 and is not divisible by 7, and if it is not, then it increments the value of i by 1. The else statement checks if the value of i is divisible by 7; if it is, then it prints the value of i. If the while loop fails to find any number divisible by 7, then the else statement prints that no number is divisible by 7.

2.2.8 Functions

A function is a block of code that performs a specific task whenever it is called. In bigger programs, where we have large amounts of code, it is advisable to create or use existing functions that make the program flow organized and neat.

There are two types of functions:

- built-in functions
- user-defined functions

1. Built-in functions:

These functions are defined and pre-coded in python. Some examples of built-in functions are as follows:

min(), max(), len(), sum(), type(), range(), dict(), list(), tuple(), set(), print(), etc.

2. User-defined functions:

We can create functions to perform specific tasks as per our needs. Such functions are called user-defined functions.

Syntax:

def function_name(parameters):

Code and Statements

- Create a function using the def keyword, followed by a function name, followed by a paranthesis (()) and a colon(:).
- Any parameters and arguments should be placed within the parentheses.
- Rules to naming function are similar to that of naming variables.
- Any statements and other code within the function should be indented.

2.2.9 Modules

Python modules are python files that contain python code that we can use within our python files ensuring simplicity and code reusability.

Here are some popular python built-in modules:

csv, datetime, json, math, random, sqlite3, statistics, tkinter, turtle, etc.

Example: importing math module:

```
import math
print("Sin(0) =", math.sin(0))
print("Sin(30) =", math.sin(math.pi/6))
print("Sin(45) =", math.sin(math.pi/4))
print("Sin(60) =", math.sin(math.pi/3))
print("Sin(90) =", math.sin(math.pi/2))
```

Creating and using module:

Write code and save file with extension .py .

Example: creating file module.py and writing some code:

```
def add(a, b):
    return (a+b)
def sub(a, b):
    return (a-b)
def mul(a, b):
    return (a*b)
def div(a, b):
    return (a/b)
```

Now we can use this code in a different python file by simply using the import statement.

Example: creating file calc.py and importing module.py import module

Now we can write code to call the functions from module.py in calc.py Example:

```
import module
num1 = int(input("Enter first number:"))
num2 = int(input("Enter second number:"))
print("Addition", module.add(num1, num2))
print("Subtraction", module.sub(num1, num2))
 print("Multiplication", module.mul(num1, num2))
 print("Division", module.div(num1, num2))
```

2.2.10 Python OOPs Concepts

A class is a blueprint or a template for creating objects while an object is an instance or a copy of the class with actual values.

Creating a Class:

Create a class using the class keyword. Example:

```
class Details:
name = "Simran" age = 20
```

Creating an Object:

Now create object of the class. Example:

```
obj1 = Details() print(obj1.name) print(obj1.age)
```

Now we can print values:

Example:

class Details:

name = "Simran" age = 20

obj1 = Details() print(obj1.name) print(obj1.age)

CHAPTER 3

MACHINE LEARNING

3.1 Introduction

Machine Learning is a fantastic new branch of science that is slowly taking over day-to-day life. From targeted ads to even cancer cell recognition, machine learning is everywhere. The high-level tasks performed by simple code blocks raise the question, "How is machine learning done?".

Machine learning is the process of making systems that learn and improve by themselves, by being specifically programmed.

The ultimate goal of machine learning is to design algorithms that automatically help a system gather data and use that data to learn more. Systems are expected to look for patterns in the data collected and use them to make vital decisions for them.

In general, machine learning is getting systems to think and act like humans, show human-like intelligence, and give them a brain. In the real world, there are existing machine learning models capable of tasks like :

- Separating spam from actual emails, as seen in Gmail
- Correcting grammar and spelling mistakes, as seen in autocorrect

Thanks to machine learning, the world has also seen design systems capable of exhibiting uncanny human-like thinking, which performs tasks like:

- Object and image recognition
- Detecting fake news
- Understanding written or spoken words
- Bots on websites that interact with humans, like humans
- Self-driven cars

3.2 Machine Learning Steps

A machine learning system builds prediction models, learns from previous data, and predicts the task of imparting intelligence to machines seems daunting and impossible. But it is actually really easy. It can be broken down into 7 major steps :

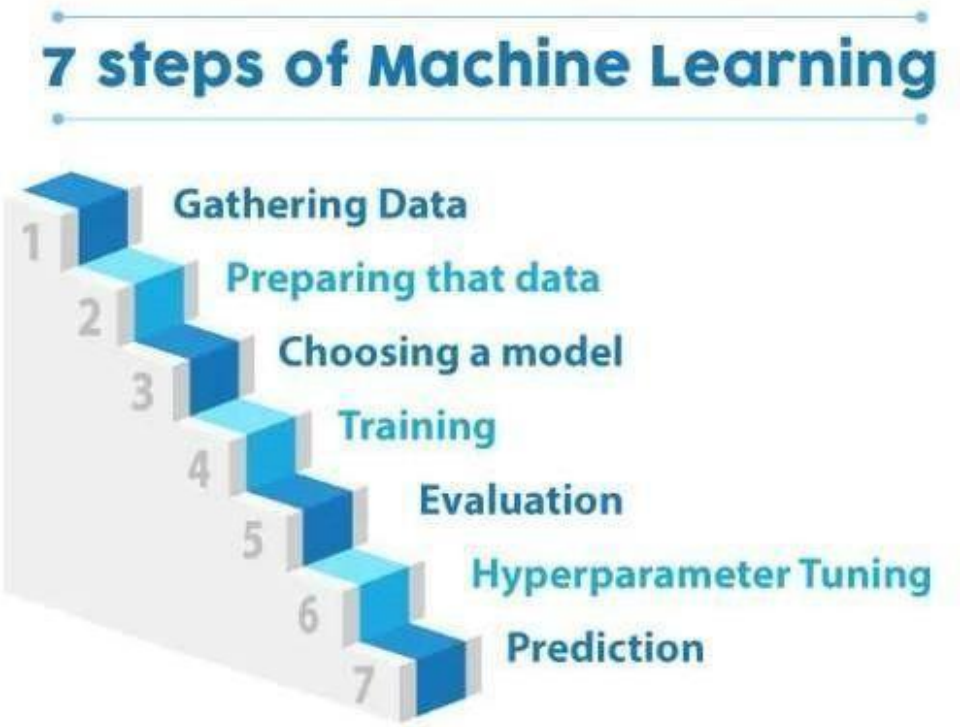

Fig 3.2.1 : Machine Learning Steps

1. Collecting Data:

As you know, machines initially learn from the data that you give them. It is of the utmost importance to collect reliable data so that your machine learning model can find the correct patterns. The quality of the data that you feed to the machine will determine how accurate your model is. If you have incorrect or outdated data, you will have wrong outcomes or predictions which are not relevant.

Make sure you use data from a reliable source, as it will directly affect the outcome of your model. Good data is relevant, contains very few missing and repeated values, and has a good representation of the various subcategories/classes present.

2. Preparing the Data:

After you have your data, you have to prepare it. You can do this by :

- Putting together all the data you have and randomizing it. This helps make sure that data is evenly distributed, and the ordering does not affect the learning process.
- Cleaning the data to remove unwanted data, missing values, rows, and columns, duplicate values, data type conversion, etc. You might even have to restructure the dataset and change the rows and columns or index of rows and columns.
- Visualize the data to understand how it is structured and understand the relationship between various variables and classes present.
- Splitting the cleaned data into two sets - a training set and a testing set. The training set is the set your model learns from. A testing set is used to check the accuracy of your model after training.

3. Choosing a Model:

A machine learning model determines the output you get after running a machine learning algorithm on the collected data. It is important to choose a model which is relevant to the task at hand. Over the years, scientists and engineers developed various models suited for different tasks like speech recognition, image recognition, prediction, etc. Apart from this, you also have to see if your model is suited for numerical or categorical data and choose accordingly.

4. Training the Model:

Training is the most important step in machine learning. In training, you pass the prepared data to your machine learning model to find patterns and make predictions. It results in the model learning from the data so that it can accomplish the task set. Over time, with training, the model gets better at predicting.

5. Evaluating the Model:

After training your model, you have to check to see how it's performing. This is done by testing the performance of the model on previously unseen data. The unseen data used is the testing set that you split our data into earlier. If testing was done on the same data which is

used for training, you will not get an accurate measure, as the model is already used to the data, and finds the same patterns in it, as it previously did. This will give you disproportionately high accuracy.

When used on testing data, you get an accurate measure of how your model will perform and its speed.

6. Parameter Tuning:

Once you have created and evaluated your model, see if its accuracy can be improved in any way. This is done by tuning the parameters present in your model. Parameters are the variables in the model that the programmer generally decides. At a particular value of your parameter, the accuracy will be the maximum. Parameter tuning refers to finding these values.

7. Making Predictions

In the end, you can use your model on unseen data to make predictions accurately.

3.3 Features of Machine Learning:

- Machine learning uses data to detect various patterns in a given dataset.
- It can learn from past data and improve automatically.
- It is a data-driven technology.
- Machine learning is much similar to data mining as it also deals with the huge amount of the data.

3.4 Classification of Machine Learning

1. Supervised Machine Learning

Supervised learning is a type of machine learning where the algorithm is trained on a labeled dataset. In this process, the system receives input-output pairs, where the input data is paired with the correct output label. The goal is for the machine learning model to learn the relationship between the input and output during training. Once the training is complete, the model is tested on new data to assess its ability to make accurate predictions. Essentially, supervised learning involves learning a mapping from inputs to outputs, and its success relies on guidance or supervision, much like a student learns under the guidance of a teacher. An example of supervised learning is spam filtering, where the system learns to classify emails as either spam or not based on labeled examples. Supervised learning algorithms are generally divided into two main categories:

- Classification
- Regression

2. Unsupervised learning:

Unsupervised learning is a type of machine learning where the model is trained on data that is not labeled or categorized. In this approach, the machine is given input data without predefined outcomes, and it must analyze and learn patterns from the data on its own. The primary goal of unsupervised learning is to find structure or relationships within the data, such as grouping similar data points together or identifying hidden patterns. Unlike supervised learning, there is no "correct" output provided, and the machine explores the data to discover useful insights. Unsupervised learning algorithms are typically divided into two main categories:

- **Clustering**
- **Association**

3. Reinforcement Learning

Reinforcement learning is a type of machine learning where an agent learns by interacting with its environment. The agent takes actions and receives feedback in the form of rewards or penalties, depending on whether the action is right or wrong. This feedback helps the agent improve its performance over time. The goal of the agent is to maximize the cumulative reward by learning from the consequences of its actions. Reinforcement learning allows the agent to explore and learn from its environment autonomously. A great example of reinforcement learning is a robotic dog that learns to move its limbs or perform tasks by trial and error, gradually improving its movements based on the feedback it receives.

3.5 Machine Learning Algorithms

Here's a brief explanation of the commonly used machine learning algorithms:

1. **Linear Regression**: A simple algorithm used to predict a continuous output variable based on one or more input features. It assumes a linear relationship between the input variables and the output.

2. **Logistic Regression**: A classification algorithm used for binary classification problems. It predicts the probability of a data point belonging to a particular class using a logistic (sigmoid) function to output values between 0 and 1.

3. **Decision Tree**: A model that splits data into subsets based on feature values. It makes decisions by following a tree-like structure, where each node represents a decision based on a feature, and leaves represent the final prediction.

4. **SVM (Support Vector Machine)**: A powerful algorithm used for classification and regression. SVM tries to find the hyperplane that best separates data points of different classes while maximizing the margin between them.

5. **Naive Bayes**: A probabilistic classifier based on Bayes' Theorem. It assumes that the features are independent and calculates the probability of each class given the input features, often used for text classification problems like spam detection.

6. **K-nearest Neighbors (K-NN)**: A simple algorithm that classifies a data point based on the majority class of its 'K' nearest neighbors in the feature space. It's a lazy learner that doesn't build an explicit model but stores the entire training dataset for prediction.

7. **K-Means**: A clustering algorithm that groups data into K clusters based on feature similarity. The algorithm iteratively assigns each data point to the nearest cluster centroid and then updates the centroids until convergence.

8. **Random Forest**: An ensemble learning method that creates multiple decision trees during training and outputs the mode (for classification) or mean (for regression) of the individual tree predictions. It reduces overfitting by averaging multiple models.

Gradient Boosting Algorithms: A family of algorithms (like GBM, XGBoost, LightGBM, and CatBoost) that combine multiple weak learners (typically decision trees) to create a strong predictive model. They work by building trees sequentially, each correcting the errors of the previous one.

9. **GBM (Gradient Boosting Machine)**: A specific type of gradient boosting algorithm that builds decision trees sequentially, where each tree aims to correct the residual errors of the previous trees. It minimizes a loss function using gradient descent.

10. **XGBoost**: A highly efficient, scalable, and popular gradient boosting algorithm known for its performance in machine learning competitions. It optimizes the boosting process using techniques like regularization and parallelization.

11. **LightGBM**: A gradient boosting framework that is particularly efficient for large datasets and focuses on optimizing the performance of decision trees by using techniques like histogram-based algorithms and leaf-wise growth.

12. **CatBoost**: A gradient boosting algorithm designed to handle categorical features without needing manual preprocessing. It is known for its speed, accuracy, and ease of use in dealing with real-world data, especially with categorical variables.

    These algorithms provide a wide range of tools for different types of machine learning tasks, from classification to regression to clustering.

CHAPTER 4

PYTHON LIBRARIES

Machine learning is a science of programming the computer by which they can learn from different types of data. According to machine learning's definition of Arthur Samuel - "Field of study that gives computers the ability to learn without being explicitly programmed". The concept of machine learning is basically used for solving different types of life problems.In previous days, the users used to perform tasks of machine learning by manually coding all the algorithms and using mathematical and statistical formulas.

This process was time-consuming, inefficient, and tiresome compared to Python libraries, frameworks, and modules. But in today's world, users can use the Python language, which is the most popular and productive language for machine learning. Python has replaced many languages as it is a vast collection of libraries, and it makes work easier and simpler.

Python libraries are collections of modules that contain useful codes and functions, eliminating the need to write them from scratch. There are tens of thousands of Python libraries that help machine learning developers, as well as professionals working in data science, data visualization, and more.

Python is the preferred language for machine learning because its syntax and commands are closely related to English, making it efficient and easy to learn. Compared with C++, R, Ruby, and Java, Python remains one of the simplest languages, enabling accessibility, versatility, and portability. It can operate on nearly any operating system or platform.

4.1 Python libraries for machine learning

If you’re working with machine learning and deep learning projects, there are thousands of Python libraries to choose from, and they can vary in size, quality, and diversity. Here is a curated list of the best Python libraries to help you get started on your machine learning journey. This list is based on popularity, derived from their reputation among Python library users.

1. **NumPy**

NumPy is an open-source numerical and popular Python library. It can be used to perform a variety of mathematical operations on arrays and matrices. It's one of the most used scientific computing libraries, and it's often used by scientists for data analysis. Additionally, its ability to process multidimensional arrays—handling linear algebra and Fourier transformation—makes it ideal for machine learning and artificial intelligence (AI) projects.

Compared with regular Python lists, NumPy arrays require significantly less storage area. They're also much faster and more convenient to use than the former. NumPy allows you to manipulate the data in the matrix and to transpose and reshape it. Compiled, Numpy's capabilities help you improve the performance of your machine learning model without much hassle.

2. **SciPy**

SciPy is a free and open-source library that's based on NumPy. It can be used to perform scientific and technical computing on large sets of data. Similar to NumPy, SciPy comes with embedded modules for array optimization and linear algebra. It's considered a foundational Python library due to its critical role in scientific analysis and engineering.

SciPy depends greatly on NumPy for its array manipulation subroutines and includes all of NumPy's functions. However, it adds to them to make them full-fledged scientific tools that are still user-friendly.

SciPy is ideal for image manipulation and provides basic processing features of non-scientific high-level mathematical functions. It's easy to use and fast to execute. It also includes high-level commands that play a role in data visualization and manipulation.

3. **Scikit-Learn**

Based on NumPy and SciPy, scikit-learn is a free Python library that's often considered a direct extension of SciPy. It was specifically designed for data modeling and developing machine learning algorithms, both supervised and unsupervised.

Thanks to its simple, intuitive, and consistent interface, scikit-learn is both beginner and user-friendly. Although the use of scikit-learn is limited because it only excels in data modeling, it does an excellent job at allowing users to manipulate and share data however they need.

4. **Theano**

Theano is a numerical computation Python library made specifically for machine learning. It allows for efficient definition, optimization, and evaluation of mathematical expressions and matrix calculations to employ multidimensional arrays to create deep learning models. It's a highly specific library and almost exclusively used by ML and DL developers and programmers.

Theano supports integration with NumPy, and when used with a graphics processing unit (GPU) rather than a central processing unit (CPU), it performs data-intensive computations 140 times faster. Additionally, Theano has built-in validation and unit testing tools to avoid bugs and errors later on in the code.

5. **TensorFlow**

TensorFlow is a free and open-source Python library that specializes in differentiable programming. The library offers a collection of tools and resources that help make building DL and ML models and neural networks straightforward for beginners and professionals. TensorFlow's architecture and framework are flexible and allow it to run on several computational platforms such as CPU and GPU. However, it performs its best when working on a tensor processing unit (TPU).

TensorFlow can be used to implement reinforcement-learning in ML and DL models and allows you to directly visualize your machine learning models with its built-in tools. TensorFlow isn't limited to working on desktop devices. It lets you create and train smart models on servers and smartphones.

6. Keras

Keras is an open-source Python library designed for developing and evaluating neural networks within deep learning and machine learning models. It can run on top of Theano and TensorFlow, making it possible to start training neural networks with a little code. The Keras library is modular, flexible, and extensible, making it beginner- and user-friendly. It also offers a fully functioning model for creating neural networks as it integrates with objectives, layers, optimizers, and activation functions.

Keras framework is flexible and portable, allowing it to operate in multiple environments and work on both CPUs and GPUs. It allows for fast and efficient prototyping, research work, and data modeling and visualization. Keras also has one of the widest ranges when it comes to data types because it can work on text images and images to train models.

7. PyTorch

PyTorch is an open-source machine learning Python library that's based on the C programming language framework, Torch. PyTorch qualifies as a data science library and can integrate with other similar Python libraries such as NumPy. It's able to seamlessly create computational graphs that can be changed anytime while the Python program is running. It's mainly used in ML and DL applications such as computer vision and natural language processing.

PyTorch is known for its high speeds of execution even when it's handling heavy and extensive graphs. It's also highly flexible, which allows it to operate on simplified processors in addition to CPUs and GPUs. PyTorch comes with a collection of powerful APIs that lets you expand on the PyTorch library, as well as a natural language toolkit for smoother processing. It's compatible with Python's IDE tools, which makes for an easy debugging process.

8. Pandas

Pandas is a data science and analysis Python library that allows developers to build intuitive and seamless high-level data structures. Built on top of NumPy, Pandas is responsible for preparing data sets and points for machine training. Pandas uses two types of data structures, one-dimensional (series) and two-dimensional (DataFrame), which, together,

The Pandas library is flexible and can be used in tandem with other scientific and numerical libraries. Its data structures are easy to use because they're highly descriptive, quick, and compliant. With Pandas, you can manipulate data functionality by grouping, integrating, and re-indexing it using minimal commands.

9. Matplotlib

Matplotlib is a data visualization library that's used for making plots and graphs. It's an extension of SciPy and is able to handle NumPy data structures as well as complex data models made by Pandas. Although its expertise is limited to 2D plotting, Matplotlib can produce high-quality and publish-ready diagrams, graphs, plots, histograms, error charts, scatter plots and bar charts.

Matplotlib is intuitive and easy to use, making it a great choice for beginners. It's even easier to use for people with preexisting knowledge in various other graph-plotting tools. It offers GUI toolkit support, including wxPython, Tkinter, and Qt.

10. Beautiful Soup

Beautiful Soup is a Python package used for web scraping and data collection that parses XML and HTML documents and prepares them for manipulation. It creates a parse tree for all the parsed pages of a website that can then be used to seamlessly extract the web content's data from HTML. Thanks to its versatility and the type of data it's able to scrape, Beautiful Soup is used by data scientists and analysts as well as by ML and DL developers looking for data to train their programs.

Beautiful Soup is incredibly fast and efficient at doing its job and doesn't require extensive hardware resources to function. It's extremely lenient and works with a variety of websites and encoded data types. Beautiful Soup is easy to use even for absolute Python beginners thanks to its simplistic code, comprehensive documentation, and active online community.

11. Scrapy

Scrapy is a free and open-source web scraping Python library. It's designed for large- scale web scraping. It comes included with all the tools needed to extract data from websites and process them into use-ready states. In addition to web scraping and

crawling, Scrapy also allows you to use APIs to extract data directly from websites that offer it.

One of Scrapy's biggest advantages is its incredible data scraping speeds in relation to its efficient CPU and memory use. Scrapy's spiders make parallel requests to the website and don't have a long wait line. In addition to being easily extendable, Scrapy is extremely beginner- and user-friendly thanks to its strong community of developers and sufficient documentation.

12. Seaborn

Seaborn is an open-source data visualization and plotting Python library. It's based on the plotting library Matplotlib and includes the extensive data structures of Pandas. On its own, Seaborn provides a high-level and feature-heavy interface to draw accurate and informative statistical graphs. It's used in ML and DL projects because of its ability to generate sensible plots of learning and execution data.

Seaborn produces the most visually appealing and attractive graphs and plots, making it perfect for use in publications and marketing. Additionally, Seaborn allows you to create extensive graphs with little code and simple commands, so it can help save time and effort on your behalf.

13. PyCaret

PyCaret is an open-source Python machine learning library that's based on the Caret machine learning library written in R. PyCaret offers features that automate and simplify standard practices and ML programs. It allows ML developers to spot-check a myriad of standard ML and DL algorithms on a classification or regression data set with a single command.

PyCaret is moderately easy to use even though there's a learning curve. It's important to note that PyCaret is low-code, which makes it a low-energy and efficient library to use. In addition to its ability to compare different machine learning models for you, PyCaret has simple commands or basic data processing and feature engineering.

14. OpenCV

OpenCV is a library consisting of various programming functions, which makes it useful for real-time computer vision programs. It's able to process a variety of visual inputs from image and video data and identify objects, faces, and handwriting.

OpenCV was designed with computational efficiency in mind. The library takes full advantage of its multicore processing functions to allow for a strong focus on real-time data processing in applications. It also has a supportive and active online community that keeps it going.

4.2 Flask Overview with Libraries:

Flask is a lightweight Python web framework that enables developers to build web applications with ease. It is often called a "micro-framework" because it focuses on simplicity and minimalism while allowing the addition of external libraries for extended functionality. Key libraries used with Flask include:

- Flask: The core web framework that handles routing, request/response management, and server setup.
- Jinja2: A templating engine used to render dynamic HTML pages.
- Werkzeug: A toolkit for building web applications, providing utilities for HTTP request handling, routing, and session management.
- Flask-SQLAlchemy: An extension to integrate SQL databases using SQLAlchemy ORM for easier data management.
- Flask-WTF: Simplifies form handling and validation by integrating Flask with WTForms.
- Flask-Login: Manages user sessions, allowing you to implement user authentication and security in Flask apps.
- Flask-Mail: Used for sending emails within a Flask application.

4.3 Tkinter Overview with Libraries:

Tkinter is the standard Python library for building graphical user interfaces (GUIs) in desktop applications. It provides a simple way to create windows, buttons, menus, and other interactive elements for desktop applications. Tkinter is built on the Tk GUI toolkit and is part of the Python standard library. Key components and libraries include:

Tkinter: The core library for creating GUIs, allowing you to design windows, dialogs, buttons, labels, and other widgets.

ttk: A themed widget set that allows you to create modern-looking user interfaces with Tkinter.

Tkinter.messagebox: A module to display message boxes for alerts, confirmations, and user notifications.

Tkinter.filedialog: Provides dialogs for file selection, allowing users to open or save files from within the application.

PIL/Pillow: A library for image processing, often used alongside Tkinter for handling image display in GUI applications.

CHAPTER 5

MACHINE LEARNING REGRESSION

5.1 Machine learning Regression

Regression analysis is a fundamental component of machine learning, widely employed for predicting continuous outcomes based on input features. In Python, implementing regression models is facilitated by various libraries, with Scikit-learn being a prominent choice. Scikit-learn offers a comprehensive suite of regression algorithms, including Linear Regression, Ridge Regression, and Lasso Regression. For instance, a simple linear regression model can be created by splitting data into training and testing sets, training the model, and evaluating its performance using metrics such as mean squared error.Additionally, Stats models is another library in Python, focusing on statistical models. It provides functionalities for estimating and testing different statistical models, including Ordinary Least Squares (OLS) regression. OLS regression allows for in-depth analysis with detailed statistics and hypothesis tests, providing insights into the relationships between variables.

5.2 Types of regression

5.2.1 LINEAR REGRESSION

Linear regression is one of the most basic types of regression in machine learning. The linear regression model consists of a predictor variable and a dependent variable related linearly to each other. In case the data involves more than one independent variable, then linear regression is called multiple linear regression models.

Example:

```
import matplotlib.pyplot as plt x  = [5,7,8,7,2,17,2,9,4,11,12,9,6]

y =  [99,86,87,88,111,86,103,87,94,78,77,85,86]

plt.scatter(x, y)

plt.show()
```

Result:

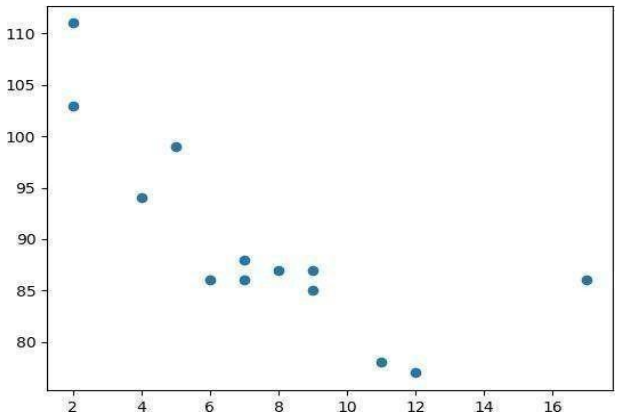

fig 5.2.1 linear regression graph

5.2.2 MULTILINEAR REGRESSION

Multiple linear regression is an extension of simple linear regression where you have more than one independent variable. In simple terms, it helps you model the relationship between a dependent variable and multiple independent variables.

Explanation:

1. Variables:

- Dependent Variable (\(y \)): This is the variable you are trying to predict.
- Independent Variables (\(x_1, x_2, \ldots, x_n \)): These are the variables that you believe have an impact on the dependent variable.

2. Equation:

The equation for multiple linear regression is an extension of the simple linear regression equation:

\[y = b_0 + b_1 \cdot x_1 + b_2 \cdot x_2 + \ldots + b_n \cdot x_n \] Here,

\(y \) is the dependent variable,

- \(x_1, x_2, \ldots, x_n \) are the independent variables,
- \(b_0 \) is the y-intercept,

3. Goal:

The goal of multiple linear regression is to find the values of \(b_0, b_1, b_2, \ldots, b_n \) that minimize the difference between the predicted values and the actual values in your dataset.

4. Python Implementation:

In Python, you would use a library like `scikit-learn` to create a multiple linear regression model. You provide the model with your dataset containing multiple independent variables and the corresponding dependent variable.

```python

from sklearn.linear_model import LinearRegression

# Create a linear regression model model = LinearRegression()
# Fit the model to the data (X represents multiple independent variables) model.fit(X, y) # Make predictions using the trained model y_pred = model.predict(X_new)
```

Overall, multiple linear regression is a powerful tool when you have more than one factor influencing the variable you want to predict. It helps you understand the combined impact of multiple variables on the outcome.

5.2.3 POLYNOMIAL REGRESSION

If your data points clearly will not fit a linear regression (a straight line through all data points), it might be ideal for polynomial regression.
Polynomial regression, like linear regression, uses the relationship between the variables x and y to find the best way to draw a line through the data points.

Example:

```
import matplotlib.pyplot as plt

x = [1,2,3,5,6,7,8,9,10,12,13,14,15,16,18,19,21,22]

y =[100,90,80,60,60,55,60,65,70,70,75,76,78,79,90,99,99,100]

plt.scatter(x, y)

plt.show()
```

Result:

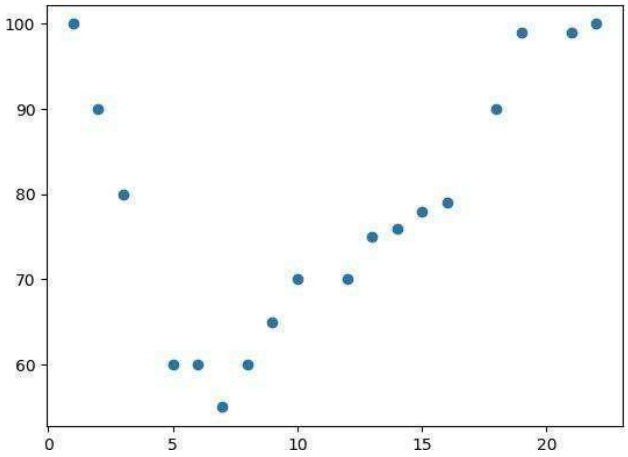

Fig 5.2.2 polynomial regression

5.2.4 LOGISTIC REGRESSION

Logistic regression is one of the types of regression analysis technique, which gets used when the dependent variable is discrete. Example: 0 or 1, true or false, etc. This means the target variable can have only two values, and a sigmoid curve denotes the relation between the target variable and the independent variable.

logit(p) = ln(p/(1-p)) = b0+b1X1+b2X2+b3X3….+bkXk where p is the probability of occurrence of the feature.

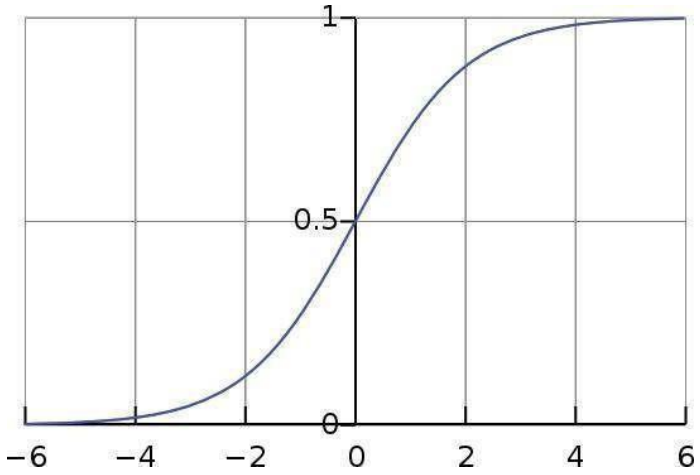

Fig 5.2.3 Logistic Regression

5.2.5 DECISION MAKING REGRESSION

is a decision-making tool that uses a flowchart-like tree structure or is a model of decisions and all of their possible results, including outcomes, input costs, and utility. Decision-tree algorithm falls under the category of supervised learning algorithms. It works for both continuous as well as categorical output variables.

The branches/edges represent the result of the node and the nodes have either:

1. Conditions [Decision Nodes]
2. Result [End Nodes]

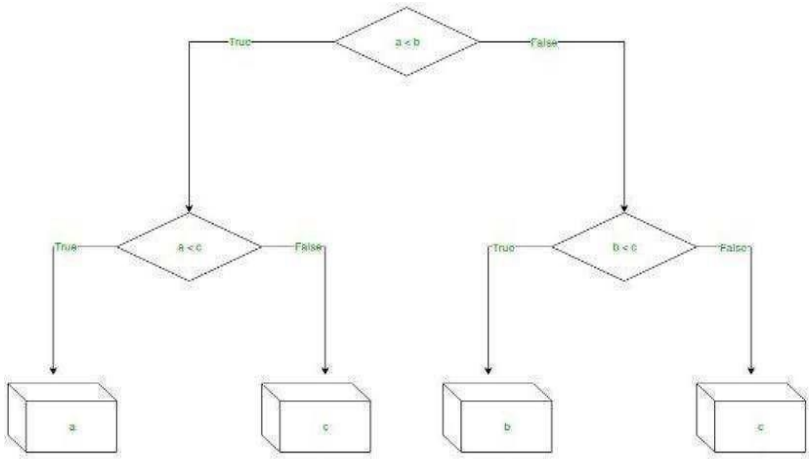

Fig 5.2.4 Decision making

5.2..6 RANDOM FOREST REGRESSION

Random forest is also a widely-used algorithm for non-linear regression in Machine Learning. Unlike decision tree regression (single tree), a random forest uses multiple decision trees for predicting the output. Random data points are selected from the given dataset (say k data points are selected), and a decision tree is built with them via this algorithm. Several decision trees are then modeled that predict the value of any new data point.

Since there are multiple decision trees, multiple output values will be predicted via a random forest algorithm. You have to find the average of all the predicted values for a

new data point to compute the final output. The only drawback of using a random forest algorithm is that it requires more input in terms of training. This happens due to the large number of decision trees mapped under this algorithm, as it requires more computational power.

Example:

```
Import numpy as nprid
      =np.arange(min(X),max(X),0.01)X_grid=X_grid.reshape(len(X_grid),1)
plt.scatter(X,y, color='blue') #plotting real points

plt.plot(X_grid,regressor.predict(X_grid),color='green')  #plotting for predict points
plt.title("Random Forest Regression Results")

plt.xlabel('Position level')

plt.ylabel('Salary')
```

Output:

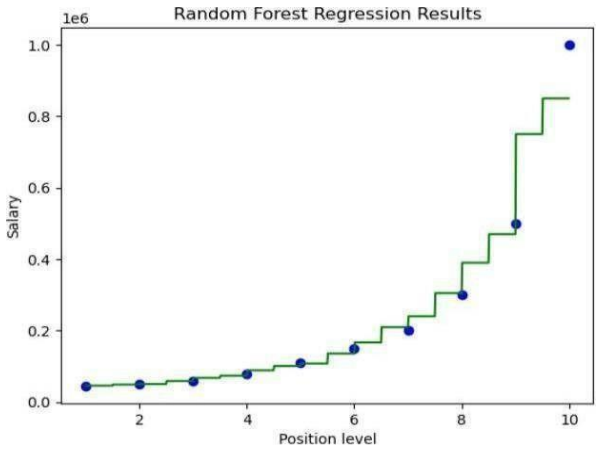

Fig 5.2.5 random forest regression

5.2.7 K Nearest Neighbours(KNN) METHOD

As we saw above, the KNN algorithm can be used for both classification and regression problems. The KNN algorithm uses **_feature similarity**‘ to predict the values of any new data points. This means that the new point is assigned a value based on how closely it resembles the points in the training set. From our example, we know that ID11 has height and age similar to ID1 and ID5, so the weight would also approximately be the same.

Had it been a classification problem, we would have taken the mode as the final prediction. In this case, we have two values of weight – 72 and 77. Any guesses on how the final value will be calculated? The average of the values is taken to be the final prediction.

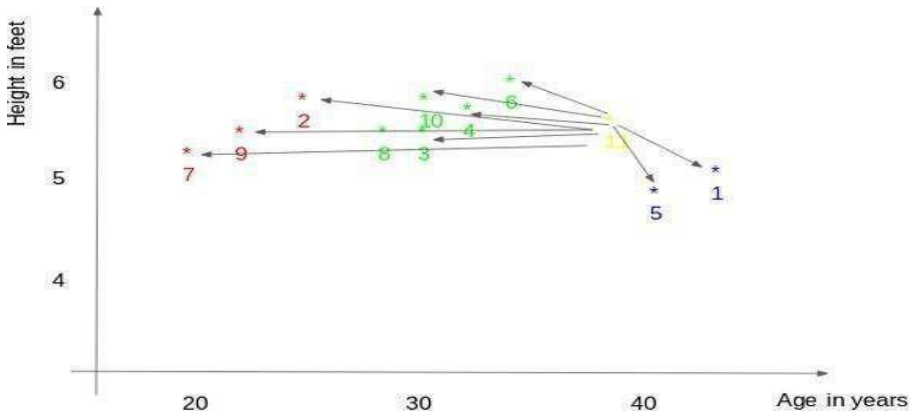

Fig 5.2.6 KNN algorithm graph

CHAPTER 6

WEATHER FORCAST USING MACHINE LEARNING

6.1 OBJECTIVES

The primary objective of this project is to develop a user-friendly weather application that can provide weather data for major Indian cities. The app will aim to deliver the following functionalities:

1. **City Selection**:

   The user should be able to select from a predefined list of major Indian cities using a drop-down combo box.

2. **Weather Data Display**:

   Upon selecting a city, the application should retrieve and display the weather data, including temperature, weather description, humidity, wind speed, and the source of the data.

3. **Weather Icon Display**:

   The app should show weather icons corresponding to the weather description (e.g., sun, rain, clouds).

4. **Error Handling**:

   The app should handle cases where the user doesn't select a city or when there is no data available for the selected city by showing appropriate error messages.

5. **Graphical User Interface (GUI)**:

   The app should have an intuitive and visually appealing interface, with color-coded messages based on the weather condition (e.g., blue for rain, yellow for sunny weather).

6. **Static Data (Fallback Data)**:

   Since the project is designed for demonstration purposes, it uses static weather data for

different cities, simulating how a real-time weather API might be used.

7. **Aesthetic Appeal**:

   The application will aim for a clean, minimalist, and user-friendly design using the Tkinter library's widgets.

6.2 METHODOLOGY

The development of the weather application follows a simple and structured methodology that involves the following steps:

1. **Requirements Gathering**:

   The first step is to understand the problem and identify the features required in the application. In this case, the application should allow users to select a city and retrieve the weather details.

2. **Data Modeling**:

   Predefined weather data is used to simulate how the application would behave in a real-world scenario. This data is structured in dictionaries for easy access.

3. **GUI Design**:

   Tkinter was chosen for building the GUI due to its simplicity and ease of integration with Python. The design focused on ensuring the application is intuitive, visually appealing, and user-friendly.

4. **Implementation of Core Features**:

   The core feature of fetching weather data from the fallback system is implemented using the get_weather() function, which checks if the city exists in the data.

   The show_weather() function handles the logic for displaying the weather report.

5. **Error Handling**:

Error handling was added to ensure that the app behaves predictably when a user does not select a city or when no data is available for a given city.

6.3 IMPLEMENTATION

The implementation of the weather application involves multiple stages, starting from designing the graphical user interface (GUI) and implementing the logic for displaying weather information. Below are the key implementation steps:

1. **Importing Necessary Libraries**:

The application uses the **Tkinter** library for GUI development, **messagebox** for pop-up alerts, and **ttk** for the dropdown widget.

```
import tkinter as tk

from tkinter import messagebox, ttk
```

2. **Defining Data**:

- The cities of India are defined in the INDIAN_CITIES list.
- Predefined weather data is stored in a dictionary FALLBACK_DATA, where each city is associated with its weather information, including temperature, humidity, description, and wind speed.
- Weather conditions are mapped to corresponding icons in the WEATHER_ICONS dictionary.

3. **Weather Data Retrieval**:

The get_weather() function is used to fetch weather data for the selected city. It retrieves the relevant data from the FALLBACK_DATA dictionary. This is a static fallback data system, simulating how a real-world app would use an API to fetch live weather information.

4. **Displaying the Weather Data**:

- When a user selects a city and clicks the "Get Weather" button, the show_weather() function is invoked.
- The selected city is checked for its existence in the fallback data.
- If found, the corresponding weather information (temperature, description, wind speed, humidity, and source) is displayed in a formatted string.
- Weather icons are also mapped to the weather descriptions for visual enhancement.

5. **GUI Development**:

- A window (root) is created using Tkinter, and various widgets (such as labels, combobox, and buttons) are added to the interface.
- he combo box (ttk.Combobox) allows users to choose a city from the predefined list.
- A button labeled "Get Weather" triggers the weather display when clicked.
- The weather details, including city, temperature, humidity, wind speed, and description, are shown on the GUI with visual feedback (color coding).

6. **Message Box and Error Handling**:

- The weather data is displayed using messagebox.showinfo(), which pops up with the weather information.
- If no city is selected or if there's an error in fetching the data, appropriate error messages are shown using messagebox.showerror().

7. **Styling the Application**:

- The weather report is color-coded based on weather conditions:
  - Light blue for rain or cloudy conditions.
  - Yellow for cle ar sky and sunny weather.

6.4 SNAPSHOTS

Fig 6.4.1: View without entering data

6.4.2: View with entering data

6.5 CODE

```
import tkinter as tk
from tkinter import messagebox, ttk
# List of major Indian cities
INDIAN_CITIES = [
    "Mumbai", "Delhi", "Bengaluru", "Hyderabad", "Ahmedabad",
    "Chennai", "Kolkata", "Pune", "Jaipur", "Lucknow"
]
# Pre-loaded fallback weather data for selected cities
FALLBACK_DATA = {
    "Mumbai": {"temp": 30, "desc": "Clear Sky", "humidity": 70, "wind_speed": 4.5},
    "Delhi": {"temp": 32, "desc": "Haze", "humidity": 40, "wind_speed": 3.2},
    "Bengaluru": {"temp": 27, "desc": "Partly Cloudy", "humidity": 60, "wind_speed": 2.8},
    "Hyderabad": {"temp": 28, "desc": "Sunny", "humidity": 55, "wind_speed": 3.0},
    "Ahmedabad": {"temp": 33, "desc": "Hot", "humidity": 45, "wind_speed": 4.0},
    "Chennai": {"temp": 31, "desc": "Humid", "humidity": 80, "wind_speed": 5.1},
    "Kolkata": {"temp": 29, "desc": "Rainy", "humidity": 85, "wind_speed": 4.8},
    "Pune": {"temp": 26, "desc": "Cool Breeze", "humidity": 50, "wind_speed": 3.5},
    "Jaipur": {"temp": 34, "desc": "Hot and Dry", "humidity": 30, "wind_speed": 3.8},
  "Lucknow": {"temp": 28, "desc": "Cloudy", "humidity": 65, "wind_speed": 3.3},
}

# Map weather conditions to icons
WEATHER_ICONS = {
  "clear sky": "🟡",
  "few clouds": "🟡 ",
```

```
    "scattered clouds": " ",
    "broken clouds": " ",
    "shower rain": " ",
    "rain": " ",
    "thunderstorm": " ",
    "snow": " ",
    "mist":
    "haze": " ",
    "sunny": " ",
    "partly cloudy": " ",
    "hot": " ",
    "humid": "     ",
    "rainy": " ",
    "cool breeze": " ",
    "hot and dry": " ",
    "cloudy": " ",
}


    """Fetch weather data only from fallback data."""
    # Check if city exists in fallback data
    if city in FALLBACK_DATA:
        fallback = FALLBACK_DATA[city]
        return {
            "city": city,
            "country": "India",
```

```
            "temp": fallback["temp"],
            "desc": fallback["desc"],
            "humidity": fallback["humidity"],
            "wind_speed": fallback["wind_speed"],
            "source": "Fallback Data"
        }
    else:
            return None


def show_weather():
    """Display weather and show city name in a message box."""
    selected_city = city_combo.get()

    if selected_city != "Select City":
        # Fetch and display weather from fallback data
        weather_data = get_weather(selected_city)

        if weather_data:
            # Get weather details
            icon = WEATHER_ICONS.get(weather_data["desc"].lower(), "")
            weather_message = (
                f"{icon} {weather_data['desc'].capitalize()}\n"
                f"City: {weather_data['city']}, {weather_data['country']}\n"
                f"Temperature: {weather_data['temp']}°C\n"
                f"Humidity: {weather_data['humidity']}%\n"
                f"Wind Speed: {weather_data['wind_speed']} m/s\n"
```

```
    Data Source: {weather_data['source']}"
)

# Style message box based on weather
if "rain" in weather_data["desc"].lower() or "shower" in weather_data["desc"].lower():
    msg_bg = "#74b9ff"  # Light blue for rain
elif "cloud" in weather_data["desc"].lower():
    msg_bg = "#b2bec3"  # Grey for clouds
elif "sunny" in weather_data["desc"].lower() or "clear" in weather_data["desc"].lower():
    msg_bg = "#ffeaa7"  # Yellow for clear sky
elif "hot" in weather_data["desc"].lower():
    msg_bg = "#ff7675"  # Red for hot weather
```

CONCLUSION

In The weather application developed using Python's Tkinter library is a simple yet effective tool that allows users to check weather conditions for various major cities in India. The application demonstrates the power of Python for creating interactive graphical user interfaces (GUIs), providing users with an engaging way to get weather information. The application integrates predefined weather data for a selection of cities, dynamically displays the corresponding weather conditions, and adapts the user interface to reflect different weather scenarios through colorful styling. This report provides an in-depth analysis of the features, structure, and overall functionality of the application, highlighting the technical aspects involved in its creation.

The weather app’s design and functionality are driven by a set of well-defined goals: to create a user-friendly interface that offers a seamless experience for retrieving and viewing weather data. Tkinter, Python's standard library for building GUIs, provides a robust framework for designing and handling the application's user interface. The application is structured around a simple layout with labels, a dropdown menu, and a button to fetch the weather data. These elements are easily recognizable and intuitive for users of all technical backgrounds, ensuring that the primary goal of usability is met.

The interface is centered on a drop-down menu, allowing users to select a city from a list of major Indian cities. The list includes cities like Mumbai, Delhi, Bengaluru, Hyderabad, Ahmedabad, Chennai, Kolkata, Pune, Jaipur, and Lucknow. The cities chosen reflect a broad range of geographical locations, ensuring that users from various parts of India can easily access weather information relevant to their region. By selecting a city, the user can retrieve weather data such as temperature, humidity, wind speed, and a description of the weather conditions.

BIBILIOGRAPHY

- https://www.w3schools.com/python/
- https://docs.python.org/3/tutorial/index.html
- https://www.tutorialspoint.com/python/index.htm
- https://www.python.org/about/gettingstarted/
- https://www.w3schools.com/python/python_ml_getting_started.asp
- https://www.tutorialspoint.com/machine_learning_with_python/index.htm
- https://realpython.com/tutorials/machine-learning/