Late Bhausaheb Hiray S.S Trust's Institute of Computer Application

Bandra (East)

C E R T I F I C A T E

This is to certify that Mr. Nishit Shah of MCA Semester - II with Roll No. **MCA2022051** has completed all practicals of **Artificial Intelligence & Machine Learning LAB** under supervision of Assistant Professor **Aquila Shaikh** and **Dr. Rashmita Pradhan** in this college during the year 2022-2023.

CO	Attendance	Performance during Lab session	Innovation in problem-solving techniques	Mock Viva during Lab session	Journal
CO1					
CO2					
CO3					
CO4					

Subject In-Charge Director

External Examiner

MCAL21 Artificial Intelligence & Machine Learning

INDEX

Sr. No	Practical List	Date	CO	Sign
1	Study of Logical Programming with Prolog	8/02/2023	**CO1**	
2	Study of Python Libraries: a. NumPy b. Pandas	15/02/2023	**CO2**	
3	Study of Python Libraries: a. Matplotlib b. Scikit	14/03/2023	**CO2**	
4	Study of the following algorithms: a. Linear Regression b. Logistic regression	20/03/2023	**CO2**	
5	c. KNN- classification	03/04/2023	**CO2**	
6	Study of dimensionality reduction techniques: a. Feature Extraction b. Normalization	10/04/2023	**CO2,CO3**	
7	Study of Principal Component Analysis	17/04/2023	**CO3**	
8	Implementation of K-Means Clustering	12/05/2023	**CO3**	
9	Implementation of Support Vector Machine	15/05/2023	**CO2,CO3**	
10	Implementation of Bagging Algorithm	16/05/2023	**CO4**	
11	Study of Boosting Algorithm: a. Ada Boost	19/05/2023	**CO4**	
12	b. Stochastic Gradient Boosting c. Voting Ensemble	02/06/2023	**CO4**	
13	Study of Python Flask Library	09/06/2023	**CO4**	

Practical – 1

Aim: - Introduction to basic concepts of python – data types, loops, conditional statements and functions.

Data Types in Python:

Python has several built-in data types, including:

- **Numeric types**: int, float, and complex.
- **Sequence types**: list, tuple, and range.
- **Text type:** str.
- **Mapping type**: dict.
- **Set types**: set, frozenset.
- **Boolean type**: bool.
- **Binary types**: bytes, bytearray, memoryview.

- **Integers (int):**

  Integers are whole numbers (positive, negative, or zero) with no decimal point.

  They can be arbitrarily large or small.

- **Floating-point numbers (float):**

  Floating-point numbers represent real numbers and include a decimal point.

- **Complex numbers (complex):**

  Complex numbers are expressed in the form a + bj, where a and b are real numbers and j is the imaginary unit.

- **Lists (list):**

  Lists are ordered collections of items that can be of any data type. They are mutable, meaning their contents can be changed.

- **Tuples (tuple):**

  Tuples are similar to lists, but they are immutable, meaning their contents cannot be changed after creation.

- **Range**:

  Range is an immutable sequence type used to represent arithmetic progressions.

- **Strings (str):**

Strings are sequences of Unicode characters and are immutable.

- **Dictionaries (dict):**

  Dictionaries are collections of key-value pairs, and the keys must be unique.

- **Sets (set):**

  Sets are unordered collections of unique items.

- **Frozen sets (frozenset):**

  Frozen sets are like sets but are immutable.

- **Booleans (bool):**

  Booleans are either True or False.

- **Bytes:**

  Bytes are immutable sequences of single bytes.

Numbers and Operators :

Operators in Python are symbols that perform operations on one or more operands to produce a result.

▾ Numbers

```
[ ] 23+22
    45

[ ] 2*4
    8

[ ] 23/4
    5.75

[ ] 2**4
    16

[ ] 4%2
    0

[ ] 5%2
    1

[ ] (2+3)*(8+9)
    85
```

Variable Assignment

```
[ ]  x=34
     y=5
     z=x+y
     z
```

```
39
```

String

```
[ ]  a='single quotes'
     b="double quotes"
     c="wrap lot's of other quotes"
     print(a)
     print(b)
     print(c)
```

```
single quotes
double quotes
wrap lot's of other quotes
```

Printing:

```
[1]  a="hello Tarun"
     print(a)
```

```
hello Tarun
```

```
[3]  a
```

```
'hello Tarun'
```

```
[4]  num=33
     name='Tarun'
     print('My number is: {one}, and my name is: {two}'.format(one=num, two=name))
```

```
My number is: 33, and my name is: Tarun
```

Lists:

In Python, a list is a collection of items that are ordered, changeable, and can contain duplicates. Lists are one of the most versatile data structures in Python, and they can be used to store a variety of data types, including integers, strings, and even other lists.

```
[ ] [1,2,3]
```

```
[1, 2, 3]
```

```
[ ] ['hi',1,[1,2]]
```

```
['hi', 1, [1, 2]]
```

```
[ ] my_list = ['a','b','c']
```

```
[ ] my_list.append('d')
```

```
[ ] my_list
```

```
['a', 'b', 'c', 'd']
```

```
[ ] my_list[0]
```

```
'a'
```

```
[ ] my_list[1:]
```

```
['b', 'c', 'd']
```

```
[ ] my_list[:1]
```

```
['a']
```

```
[ ] my_list[0] = 'hello'
```

```
[ ] my_list
```

```
['hello', 'b', 'c', 'd']
```

```
[ ] nest = [1,2,3,[4,5,['target']]]
```

```
[ ] nest[3]
```

```
[4, 5, ['target']]
```

```
[ ] nest[3][2]
```

```
['target']
```

```
[ ] nest[3][2][0]
```

```
'target'
```

```
[ ] d = {'key1':'item1','key2':'item2'}
    d
```

```
{'key1': 'item1', 'key2': 'item2'}
```

```
[ ] d['key1']
```

```
'item1'
```

Booleans

```
[ ] True
```

```
True
```

```
[ ] False
```

```
False
```

Dictionaries:

In Python, a dictionary is a collection of key-value pairs that are unordered, changeable, and indexed. Dictionaries are often used to store and manipulate data in a way that is easy to access and modify.

Each item in a dictionary consists of a key and a value, and these items are separated by a colon. The keys in a dictionary must be unique, but the values can be of any data type.

Tuples:

In Python, a tuple is a collection of items that are ordered and immutable. Tuples are similar to lists, but once a tuple is created, its contents cannot be modified.

```
[20] mytupple=("apple","mango","banana")
     print(mytupple)

     ('apple', 'mango', 'banana')

[21] mytupple[0]

     'apple'
```

Sets:

In Python, a set is a collection of unique and unordered elements. Sets are defined using curly braces or the set() function, and each element in the set is separated by a comma.

```
{1,2,3}

{1, 2, 3}

[ ] {1,2,3,1,2,1,2,3,3,3,3,2,2,2,1,1,2}

    {1, 2, 3}
```

Comparison Operator:

In Python, comparison operators are used to compare two values and return a Boolean value (True or False) based on the result of the comparison.

```
[ ] 2>4

    False

[ ] 1<2

    True

    1>=1

    True

[ ] 1<=4

    True

[ ] 'hi'=='bye'

    False
```

Logical Operators:

In Python, logical operators are used to combine two or more Boolean expressions and return a Boolean value (True or False) based on the result of the combination.
The following logical operators are available in Python:

- and: returns True if both expressions are True
- or: returns True if at least one expression is True
- not: returns the opposite of the expression (True if the expression is False, and False if the expression is True)

```
[ ] (1 > 2) and (2 < 3)

    False

[ ] (1 > 2) or (2 < 3)

    True

[ ] (1 == 2) or (2 == 3) or (4 == 4)

    True
```

The if-else statement

The if-else statement provides an else block combined with the if statement which is executed in the false case of the condition.
If the condition is true, then the if-block is executed. Otherwise, the else-block is executed.

```
[ ] if 1<2:
      print('Yep!')
```

```
Yep!
```

```
[ ] if 1 > 2:
        print('first')
    else:
        print('last')
```

```
last
```

```
if 1 == 2:
    print('first')
elif 3 == 3:
    print('middle')
else:
    print('Last')
```

```
middle
```

Python for Loop

In Python, the for loop is used to run a block of code for a certain number of times. It is used to iterate over any sequences such as list, tuple, string, etc.

```
[ ] seq = [1,2,3,4,5]
```

```
[ ] for item in seq:
        print(item)
```

```
1
2
3
4
5
```

```
for item in seq:
    print('Yep')
```

```
Yep
Yep
Yep
Yep
Yep
```

```
[ ] for jelly in seq:
        print(jelly+jelly)
```

```
2
4
6
8
10
```

Python while Loop

Python while loop is used to run a block code until a certain condition is met.

```
[ ] i = 1
    while i < 5:
        print('i is: {}'.format(i))
        i = i+1
```

```
i is: 1
i is: 2
i is: 3
i is: 4
```

Python for Loop with Python range()

A range is a series of values between two numeric intervals.
We use Python's built-in function range() to define a range of values.

```
[ ] range(5)
```

```
range(0, 5)
```

```
[ ] for i in range(5):
        print(i)
```

```
0
1
2
3
4
```

```
[ ] list(range(5))
```

```
[0, 1, 2, 3, 4]
```

Python List Comprehension:

A Python list comprehension consists of brackets containing the expression, which is executed for each element along with the for loop to iterate over each element in the Python list.
Python List comprehension provides a much shorter syntax for creating a new list based on the values of an existing list.

```
[ ] x = [1,2,3,4]
```

```
out = []
for item in x:
    out.append(item**2)
print(out)
```

```
[1, 4, 9, 16]
```

```
[ ] [item**2 for item in x]
```

```
[1, 4, 9, 16]
```

Python Functions

A function is a collection of related assertions that performs a mathematical, analytical, or evaluative operation. A collection of statements called Python Functions returns the particular task

```
[ ] def my_func(param1='default'):
        """
        Docstring goes here.
        """
        print(param1)
```

```
[ ] my_func
```

```
<function __main__.my_func(param1='default')>
```

```
[ ] my_func()
```

```
default
```

```
my_func('new param')
```

```
new param
```

```
[ ] my_func(param1='new param')
```

```
new param
```

```
[ ] def square(x):
        return x**2
```

```
[ ] out = square(2)
```

```
[ ] print(out)
```

```
4
```

Lambda Function: a lambda function is a special type of function without the function name.

```
[ ] def times2(var):
        return var*2
```

```
times2(2)
```

```
4
```

```
[ ] lambda var: var*2
```

```
<function __main__.<lambda>(var)>
```

Map and Filter

1. The map function takes each item in a given iterable and and includes all of them in a new lazy iterable, transforming each item along the way

2. The filter function doesn't transform the items, but it's selectively picks out which items it should include in the new lazy iterable

```
[ ] seq = [1,2,3,4,5]

[ ] map(times2,seq)

    <map at 0x7f6676e23fa0>

[ ] list(map(times2,seq))

    [2, 4, 6, 8, 10]

[ ] list(map(lambda var: var*2,seq))

    [2, 4, 6, 8, 10]

[ ] filter(lambda item: item%2 == 0,seq)

    <filter at 0x7f6676e23940>

[ ] list(filter(lambda item: item%2 == 0,seq))

    [2, 4]
```

Python Methods:

By definition, a method is a function that is bound to an instance of a class

```
[5] st = 'hello my name is Tarun'

[6] st.lower()

    'hello my name is tarun'

[7] st.upper()

    'HELLO MY NAME IS TARUN'

[8] st.split()

    ['hello', 'my', 'name', 'is', 'Tarun']

[ ] tweet = 'Go Sports! #Sports'

[ ] tweet.split('#')

    ['Go Sports! ', 'Sports']
```

```
tweet.split('#')[1]
```

```
'Sports'
```

```
[ ] d
```

```
{'key1': 'item1', 'key2': 'item2'}
```

```
[ ] d.keys()
```

```
dict_keys(['key1', 'key2'])
```

```
[ ] d.items()
```

```
dict_items([('key1', 'item1'), ('key2', 'item2')])
```

```
[ ] lst = [1,2,3]
```

```
[ ] lst.pop()
```

```
3
```

```
[ ] lst
```

```
[1, 2]
```

```
[ ] 'x' in [1,2,3]
```

```
False
```

```
[ ] 'x' in ['x','y','z']
```

```
True
```

```
[ ]
x="Python "
y="is "
z="Awesome "
print(x,y,z)
```

```
Python  is  Awesome
```

Python Data Types Exercise:

What is 8 to the power of 5?

```
[ ] n = 8**5
print(n)
```

```
32768
```

Split this string into a list

```
[7] s = "Hi there Tushar!"
n = s.split()
print(n)
```

```
['Hi', 'there', 'Tushar!']
```

Use .format() to print the following string:

```
[ ]  planet = "Mars"
     diameter = 15742

     txt = "The diameter of {P} is {D} kilometers".format(P = planet, D = diameter)
     print(txt)
```

```
The diameter of Mars is 15742 kilometers
```

Given this nested list, use indexing to grab the word "hii"

```
[ ]  lst = [1,2,[3,4],[5,[100,200,['hii']],23,11],10,7]
     print(lst[3][1][2][0])

     d = {'k1':[1,2,3,{'tricky':['OK','woman','deception',{'Rule':[1,2,3,'hii']}]}]}
     print(d['k1'][3]['tricky'][3]['Rule'][3])
```

```
hii
hii
```

What is the main difference between a tuple and a list?

The main difference between tuples and lists is that tuples are immutable as opposed to lists which are mutable. Therefore, it is possible to change a list but not a tuple

Create a function that grabs the email website domain from a string in the form: tushar@ves.ac.in So, for example, passing "tushar@ves.ac.in" would return: ves.ac.in

```
[ ]  n = "tushar@ves.ac.in"
     f = "@"
     print(n[n.index('@')+1:])
```

```
ves.ac.in
```

Create a basic function that returns True if the word 'cat' is contained in the input string. Do account for capitalization.

```
[6]  txt = "Hello I am a cat"
     if txt.find("cat")!=-1:
       print("True")
     else:
       print("False")
```

```
True
```

Create a function that counts the number of times the word "cat" occurs in a string. Again ignore edge cases.

```
def countcat(n):
  count = 0
  s = n.split()
  for word in s:
    if word.lower() == "cat":
      count += 1
  return count

print(f'The count of word cat is {countcat("This cat runs faster than the other cat dude!")}')
```

```
The count of word cat is 2
```

Use lambda expressions and the filter() function to filter out words from a list that don't start with the letter 's'.

For example: seq = ['soup','dog','salad','cat','great'] should be filtered down to: ['soup','salad']

```
[ ]  words= ['soup','dog','salad','cat','great']

     filtered_words = list(filter(lambda x: x.startswith('s'), words))

     print(filtered_words)
```

```
['soup', 'salad']
```

Final Problem

You are driving a little too fast, and a police officer stops you. Write a function to return one of 3 possible results: "No ticket", "Small ticket", or "Big Ticket". If your speed is 60 or less, the result is "No Ticket". If speed is between 61 and 80 inclusive, the result is "Small Ticket". If speed is 81 or more, the result is "Big Ticket". Unless it is your birthday (encoded as a Boolean value in the parameters of the function) -- on your birthday, your speed can be 5 higher in all cases.

```
def caught_speeding(speed, is_birthday):
  noticket = 60
  smallticket1 = 61
  smallticket2 = 80
  bigticket = 81
  if(is_birthday):
    noticket +=5
    smallticket1 +=5
    smallticket2 +=5
    bigticket +=5

  if speed <= noticket:
    return "No ticket"
  elif speed >= smallticket1 and speed <= smallticket2:
    return "Small ticket"
  elif speed >= bigticket:
    return "Big ticket"

print(caught_speeding(82,True))
```

```
Small ticket
```

Python Programs:

1 Write a program to find a number is odd or even

```
num=int(input("Enter a Number: "))
if num%2 ==0:
  print("number is a Even ")
else:
  print("number is a Odd ")
```

```
Enter a Number: 15
number is a Odd
```

2.to find if a number is prime or not

```
def is_prime(num):
    if num <= 1:
        return False
    for i in range(2, num):
        if num % i == 0:
            return False
    return True

n = int(input("Enter a number: "))
if is_prime(n):
    print(f"{n} is a prime number.")
else:
    print(f"{n} is not a prime number.")
```

```
Enter a number: 16
16 is not a prime number.
```

3.triangle/pyramid pattern

```
for i in range(8):
  for j in range(i+1):
    print("*",end="")
  print("\r")
```

```
*
**
***
****
*****
******
*******
********
```

4.factorial

```
num=int(input("Enter a Number: "))
fact=1
for i in range(1,num+1):
  fact=fact*i

print(fact)
```

```
Enter a Number: 5
120
```

5.largest of n

```
[ ]  arr=[8,15,54,78,20,96,10,4]
     print("The max number is "+ str(max(arr)))
```

```
The max number is 96
```

6.sum of digits

```
num=int(input("Enter a Number: "))
sum = 0
while (num != 0):
    sum = sum + (num % 10)
    num = num//10


print("Sum of digits : "+str(sum))
```

```
Enter a Number: 548
Sum of digits : 17
```

7.calculator

```
num1=int(input("Enter a Number 1: "))
num2=int(input("Enter a Number 2: "))
op=input("Enter the operation(+,-,/,*): ")
if op =="+":
  print(num1+num2)
elif op=="-":
  print(num1-num2)
elif op=="/":
  print(num1/num2)
else:
  print(num1*num2)
```

```
Enter a Number 1: 10
Enter a Number 2: 5
Enter the operation(+,-,/,*): /
2.0
```

8.gross salary

```
num1=int(input("Enter the Basic Salary: "))
num2=int(input("Enter the HRA : "))
num3=int(input("Enter the Other Allownce: "))
print("Gross Salary : "+str(num1+num2+num3))
```

```
Enter the Basic Salary: 20000
Enter the HRA : 2000
Enter the Other Allownce: 1000
Gross Salary : 23000
```

9.fibonacci

```
a=int(input("Enter the lenght of series :"))
num1=0
num2=1
print(num1)
print(num2)
for i in range(2,a):
  num3=num1+num2
  print(num3)
  num1=num2
  num2=num3
```

```
Enter the lenght of series :10
0
1
1
2
3
5
8
13
21
34
```

10.first n prime numbers

```
def prime(num):
  flag=0
  for i in range(2,num):
    if num%i==0:
      flag=1

  if flag==0:
    print(num)


ran=int(input("Enter the Range: "))
for i in range(1,ran):
  prime(i)
```

```
Enter the Range: 20
1
2
3
5
7
11
13
17
19
```

Conclusion: - I have learned the basic concepts of python data types, operators, conditional statements and successfully implemented it.

Practical 2

Aim :- Introduction to Python Programming: Learn the different libraries - NumPy, Pandas, SciPy, Matplotlib, Scikit Learn.

Theory :- 1.NumPy :-

NumPy is a Python library used for working with arrays. It also has functions for working in the domain of linear algebra, fourier transform, and matrices. NumPy was created in 2005 by Travis Oliphant. It is an open source project and you can use it freely. NumPy stands for Numerical Python. In Python we have lists that serve the purpose of arrays, but they are slow to process.

NumPy aims to provide an array object that is up to 50x faster than traditional Python lists. The array object in NumPy is called ndarray, it provides a lot of supporting functions that make working with ndarray very easy. Arrays are very frequently used in data science, where speed and resources are very important.

Installation :-

If you have Python and PIP already installed on a system, then installation of NumPy is very easy. Install it using this command:

>> pip install numpy **Code :-**

import numpy import numpy

arr = numpy.array([1, 2, 3, 4, 5]) print(arr)

[1 2 3 4 5]
Output :-

2. Pandas:

- Pandas is a Python library used for working with data sets.
- It has functions for analyzing, cleaning, exploring, and manipulating data.
- The name "Pandas" has a reference to both "Panel Data", and "Python Data Analysis" and was created by Wes McKinney in 2008.

Installation of panda:-

If you have Python and PIP already installed on a system, then installation of Pandas is very easy. Install it using this command:

>> pip install pandas

Code:-

```
import pandas as pd
#The head() method returns top 5 rows df=
pd.read_csv("/content/diabetes_m23.csv")
print(df.head())
#There is also a tail() method for viewing the last rows of th e DataFrame. print(df.tail())
#The shape method returns rows & columns of data set print(df.shape)
#Drop Null Values in dataset new_df = df.dropna() print(new_df.to_string())
#The fillna() method allows us to replace empty cells with a v alue print("\n")
df = pd.read_csv("/content/diabetes_m23.csv") df.fillna(130,
inplace = True)
```

Output:-

```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35        0  33.6
1            1       85             66             29        0  26.6
2            8      183             64              0        0  23.3
3            1       89             66             23       94  28.1
4            0      137             40             35      168  43.1

   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
763           10      101             76             48      180  32.9
764            2      122             70             27        0  36.8
765            5      121             72             23      112  26.2
766            1      126             60              0        0  30.1
767            1       93             70             31        0  30.4

     DiabetesPedigreeFunction  Age  Outcome
763                     0.171   63        0
764                     0.340   27        0
765                     0.245   30        0
766                     0.349   47        1
767                     0.315   23        0
```

3. SciPy :-

- SciPy is a scientific computation library that uses NumPy underneath.
- SciPy stands for Scientific Python.

- It provides more utility functions for optimization, stats and signal processing.
- Like NumPy, SciPy is open source so we can use it freely.
- SciPy was created by NumPy's creator Travis Olliphant.

Installation:-

If you have Python and PIP already installed on a system, then installation of Pandas is very easy. Install it using this command:

>> pip install scipy **Code:-**

```
from scipy import constants
#code to find how many cubic metres are in one litre from scipy import constants
print(constants.liter) Output:-
```

```
0.001
```

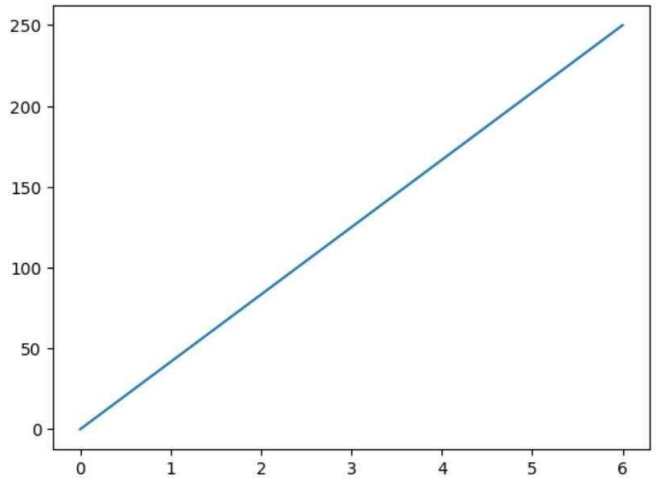
4. Matplotlib:-

- Matplotlib is a low level graph plotting library in python that serves as a visualization utility.
- Matplotlib was created by John D. Hunter. Matplotlib is open source and we can use it freely.
- Matplotlib is mostly written in python, a few segments are written in C, Objective-C and Javascript for Platform compatibility.

Code:-

```
import matplotlib.pyplot as plt
import numpy as np xpoints =
np.array([0, 6]) ypoints =
np.array([0, 250])
plt.plot(xpoints, ypoints)
plt.show()
```

Output:-



5. Scikit Learn:-

Scikit-learn (Sklearn) is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction via a consistence interface in Python. This library, which is largely written in Python, is built upon NumPy, SciPy and Matplotlib.

Code:-

```
#import lib import
pandas as pd
from sklearn.model_selection import train_test_split from
sklearn.tree import DecisionTreeClassifier, plot_tree from
sklearn.metrics import classification_report

#load the data
data = pd.read_csv("titanic_data_m2023.csv")
data.drop(["PassengerId","Name","Ticket","Cabin"] ,axis="columns", inplace=True)

#check for null data
data.isnull().sum()

#handle the data
data.fillna({
        "Age":data["Age"].mean(),
        "Embarked":"C"
        },inplace=True)

#check for null data print(data.isnull().sum())
```

```
#features and target
features = data.drop("Survived", axis="columns") target
= data["Survived"]

#handle cat data
nfeatures = pd.get_dummies(features)
print(nfeatures)

#train and test
x_train, x_test, y_train, y_test = train_test_split(nfeatures, target)

#model
model = DecisionTreeClassifier()
mf = model.fit(x_train, y_train)

#classification report
cr = classification_report(y_test, model.predict(x_test)) print(cr)

#predict
d = [[3,22.000000, 1, 0, 7.2500, 0, 1, 0, 0, 1]]
ans = model.predict(d)
print(ans)
```

Output:-

```
Survived    0
Pclass      0
Sex         0
Age         0
SibSp       0
Parch       0
Fare        0
Embarked    0
dtype: int64
     Pclass        Age  SibSp  Parch     Fare  Sex_female  Sex_male  \
0         3  22.000000      1      0   7.2500           0         1
1         1  38.000000      1      0  71.2833           1         0
2         3  26.000000      0      0   7.9250           1         0
3         1  35.000000      1      0  53.1000           1         0
4         3  35.000000      0      0   8.0500           0         1
..      ...        ...    ...    ...      ...         ...       ...
886       2  27.000000      0      0  13.0000           0         1
887       1  19.000000      0      0  30.0000           1         0
888       3  29.699118      1      2  23.4500           1         0
889       1  26.000000      0      0  30.0000           0         1
890       3  32.000000      0      0   7.7500           0         1

     Embarked_C  Embarked_Q  Embarked_S
0             0           0           1
1             1           0           0
2             0           0           1
3             0           0           1
4             0           0           1
..          ...         ...         ...
886           0           0           1
887           0           0           1
888           0           0           1
889           1           0           0
890           0           1           0

[891 rows x 10 columns]
              precision    recall  f1-score   support

           0       0.83      0.89      0.86       132
           1       0.83      0.74      0.78        91

    accuracy                           0.83       223
   macro avg       0.83      0.82      0.82       223
weighted avg       0.83      0.83      0.83       223
```

Conclusion:-

Introduction of python libraries like NumPy, Pandas, SciPy, Matplotlib, Scikit Learn done successfully.

Akash Gupta **FYMCA(B)** **ROLL NO. 83**

PRACTICAL 3

#Linear Regression

```
import numpy as np import pandas as pd import
matplotlib.pyplot as plt#Linear Regression score =
pd.read_csv("/content/student_scores.csv") score
```

```
   Hours Scores 0
2.5     21
1      5.1     47
2      3.2     27
3      8.5     75
4      3.5     30
5      1.5     20
6      9.2     88
7      5.5     60
8      8.3     81
9      2.7     25
10    7.7     85
11    5.9     62
12    4.5     41
13    3.3     42
14    1.1     17
15    8.9     95
16    2.5     30
17    1.9     24
18    6.1     67
19    7.4     69
20    2.7     30
21    4.8     54
22    3.8     35
23    6.9     76
24    7.8     86
```

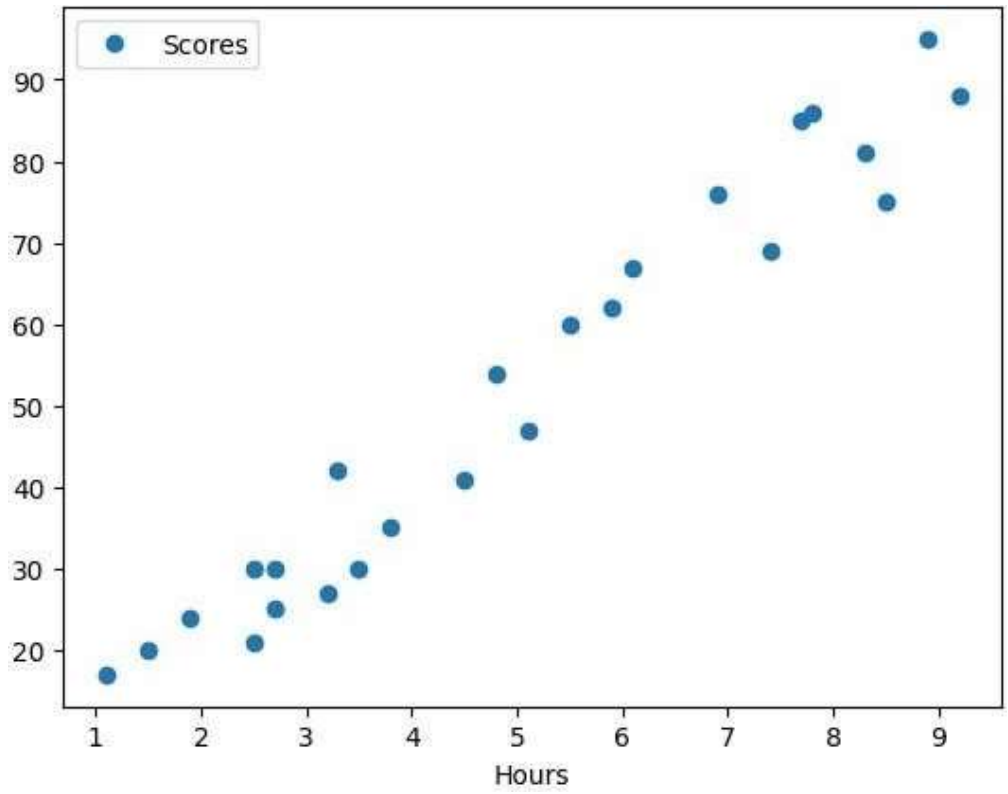
```
score.shape (25,
2)
```

```
score.describe()
```

```
         Hours     Scores
count 25.000000 25.000000 mean
5.012000            51.480000       std
2.525094            25.286887       min
```

1.100000 17.000000 25%
2.700000 30.000000
50% 4.800000 47.000000 75%
7.400000 75.000000 max
9.200000 95.000000

score.plot(x="Hours", y="Scores", style="o") plt.show()



X = score.iloc[:,:-1].values y
= score.iloc[:,1].values

```
from sklearn.linear_model import LinearRegression
```

reg = LinearRegression() reg.fit(X,y)

LinearRegression() print(reg.intercept_)

2.48367340537321

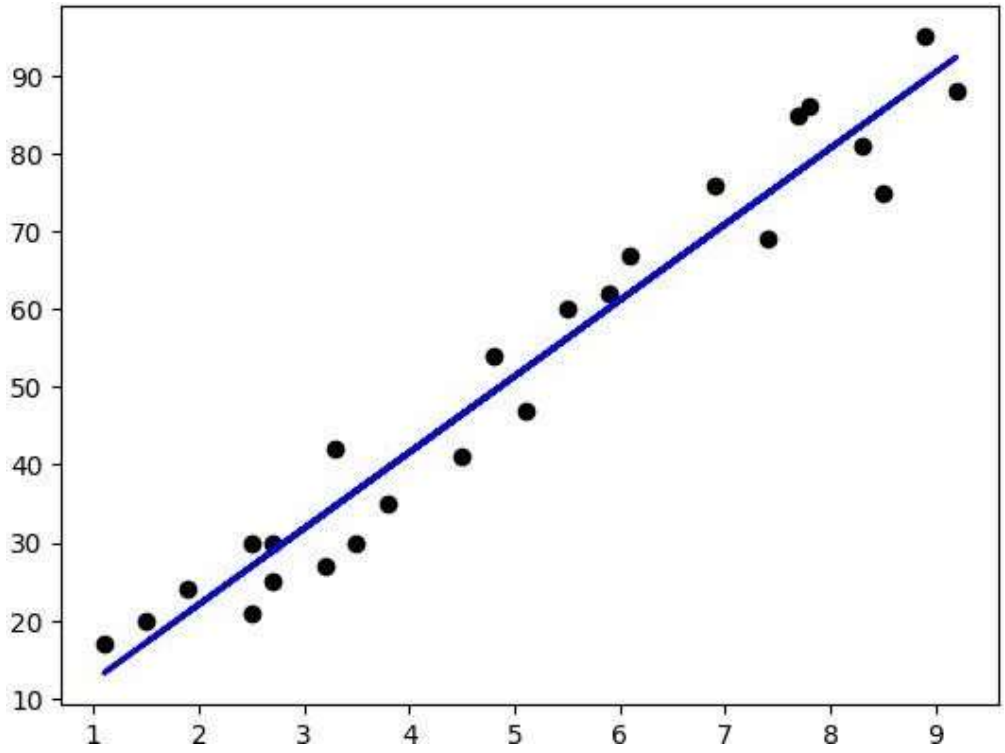
print(reg.coef_) [9.77580339]

y_pred = reg.predict(X)

df = pd.DataFrame({'Actual' : y, 'Predicted' : y_pred}) df

	Actual	Predicted
0	21	26.923182

```
1     47  52.340271
2     27  33.766244
3     75  85.578002
4     30  36.698985
5     20  17.147378
6     88  92.421065
7     60  56.250592
8     81  83.622842
9     25  28.878343
10    85  77.757360
11    62  60.160913
12    41  46.474789
13    42  34.743825
14    17  13.237057
15    95  89.488324
16    30  26.923182
17    24  21.057700
18    67  62.116074
19    69  74.824618
20    30  28.878343
21    54  49.407530
22    35  39.631726
23    76  69.936717 24     86 78.734940 from sklearn.metrics import
      mean_squared_error , r2_score print('MSE : %.2f' % mean_squared_error(y,
      y_pred)) MSE : 28.88 print('r2_score : %.2f' % r2_score(y, y_pred)) r2_score :
      0.95
plt.scatter(X,y, color = 'black') plt.plot(X,y_pred,
color = 'blue', linewidth = 2) plt.show

<function matplotlib.pyplot.show(close=None, block=None)>
```



Logistic Regression

Akash Gupta **FYMCA(B)** **ROLL NO. 83**

```
dataset = pd.read_csv("User_Data.csv")
```

dataset.shape (400, 5) dataset.head()

```
   User ID Gender Age EstimatedSalary Purchased
0 15624510 Male 19              19000        0
1 15810944 Male 35              20000        0
2 15668575 Female 26             43000        0
3 15603246 Female 27             57000        0 4 15804002 Male 19              76000
   0
```

```
x = dataset.iloc[:, [2,3]].values y
= dataset.iloc[:,4].values
print(y)
```

```
[0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0
 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 1 0 1 1 0 0 0 1 0 0 0 1 0 1
1 1 0 0 1 1 0 1 1 0 1 1 0 1 0 0 0 1 1 0 1 1 0 1 0 1 0 1 0 0 1 1 0 1 0 0 1
 1 0 1 1 0 1 1 0 0 1 0 0 1 1 1 1 1 0 1 1 1 1 0 1 1 0 1 0 1 0 1 1 1 1 0 0 0
 1 1 0 1 1 1 1 1 0 0 0 1 1 0 0 1 0 1 0 1 1 0 1 0 1 1 0 1 1 0 0 0 1 1 0 1 0
0 1 0 1 0 0 1 1 0 0 1 1 0 1 1 0 0 1 0 1 0 1 1 1 0 1 0 1 1 1 0 1 1 1 1 0 1
1 1 0 1 0 1 0 0 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 1 0 1 1 1 0 1]
```

```
from sklearn.model_selection import train_test_split xtrain, xtest, ytrain, ytest =
train_test_split(x,y, test_size = .25, random_state = 2)
```

```
from sklearn.preprocessing import StandardScaler
sc_x = StandardScaler() xtrain =
sc_x.fit_transform(xtrain) xtest =
sc_x.transform(xtest) print(xtrain[0:10,:])
```

```
[[ 0.03172569  0.06201266]
 [-0.06539376 -0.50084108]
 [-0.64811043 -1.50805304]
 [ 0.03172569  0.32862759]
 [ 0.32308402  0.09163654]
 [-0.45387154 -1.12294258]
 [-0.74522987 -1.53767692]
 [-0.25963265  -0.64896049]
[-1.13370765    0.50637087]
[-0.06539376 2.22455597]]
```

```
from sklearn.linear_model import LogisticRegression classifier
= LogisticRegression( random_state = 0) classifier.fit(xtrain,
ytrain)

LogisticRegression(random_state=0) y_pred

= classifier.predict(xtest)

from sklearn.metrics import confusion_matrix cm
= confusion_matrix(ytest, y_pred)
print("Confusion Matrix : \n", cm)
```

```
Confusion Matrix :
 [[56 6]
 [13 25]]
```

KNN Classifier

```
headernames = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'Class']

dataset = pd.read_csv("/content/iris.csv", names = headernames) dataset.head()
```

Akash Gupta **FYMCA(B)** **ROLL NO. 83**

	sepal-length	sepal-width	petal-length	petal-width	Class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

```
X = dataset.iloc[:, :-1].values y
= dataset.iloc[:, 4].values

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.40)

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler() scaler.fit(X_train)
X_train = scaler.transform(X_train) X_test
= scaler.transform(X_test)

from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 8)
classifier.fit(X_train, y_train)
KNeighborsClassifier(n_neighbors=8) y_pred =
classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
result = confusion_matrix(y_test, y_pred) print("Confusion Matrix:")
print(result)
result1 = classification_report(y_test, y_pred)
print("Classification Report:",) print (result1)
result2 = accuracy_score(y_test,y_pred) print("Accuracy:",result2)
```

```
Confusion Matrix:
[[23  0  0]
 [ 0 17  0]
 [ 0 2 18]]
Classification Report:
              precision recall f1-score support

   Iris-setosa       1.00      1.00      1.00        23 Iris-versicolor
0.89      1.00      0.94        17 Iris-virginica       1.00      0.90
0.95        20

      accuracy                           0.97        60      macro
avg       0.96      0.97      0.96        60   weighted
avg       0.97      0.97      0.97        60
```

Akash Gupta **FYMCA(B)** **ROLL NO. 83**

Accuracy: 0.9666666666666667

Practical No 4

ID3

ID3 stands for Iterative Dichotomiser 3 and is named such because the algorithm iteratively (repeatedly) dichotomizes(divides) features into two or more groups at each step.

Invented by Ross Quinlan, ID3 uses a **top-down greedy** approach to build a decision tree. In simple words, the **top-down** approach means that we start building the tree from the top and the **greedy** approach means that at each iteration we select the best feature at the present moment to create a node.

Most generally ID3 is only used for classification problems
with nominal features only.

Metrics in ID3

As mentioned previously, the ID3 algorithm selects the best feature at each step while building a Decision tree.
Before you ask, the answer to the question: ‘How does ID3 select the best feature?’ is that ID3 uses **Information Gain** or just **Gain** to find the best feature.

Information Gain calculates the reduction in the entropy and measures how well a given feature separates or classifies the target classes. The feature with the **highest Information Gain** is selected as the **best** one.

In simple words, **Entropy** is the measure of disorder and the Entropy of a dataset is the measure of disorder in the target feature of the dataset. In the case of binary classification (where the target column has only two types of classes) entropy is **0** if all values in the target column are homogenous(similar) and will be **1** if the target column has equal number values for both the classes.

We denote our dataset as **S,** entropy is calculated as:

where,

$$\textbf{Entropy(S)} = -\sum p_i * \log_2(p_i) \; ; \; \textbf{i = 1 to n}$$

n is the total number of classes in the target column (in our case n = 2 i.e YES and NO)

$\mathbf{p_i}$ is the **probability of class 'i'** or the ratio of "*number of rows with class i in the target column"* to the "*total number of rows"* in the dataset.

Information Gain for a feature column **A** is calculated as:

$$\textbf{IG(S, A)} = \textbf{Entropy(S)} - \sum((|S_v| / |S|) * \textbf{Entropy}(S_v))$$

where $\mathbf{S_v}$ is the set of rows in **S** for which the feature column **A** has value **v**, $|S_v|$ is the number of rows in S_v and likewise **|S|** is the number of rows in **S.**

ID3 Steps

1. Calculate the Information Gain of each feature.
2. Considering that all rows don't belong to the same class, split the dataset **S** into subsets using the feature for which the Information Gain is maximum.
3. Make a decision tree node using the feature with the maximum Information gain.
4. If all rows belong to the same class, make the current node as a leaf node with the class as its label.
5. Repeat for the remaining features until we run out of all features, or the decision tree has all leaf nodes.

Implementation on our Dataset

As stated in the previous section the first step is to find the best feature i.e. the one that has the maximum Information Gain**(IG)**. We'll calculate the IG for each of the features now, but for that, we first need to calculate the entropy of **S**

From the total of 14 rows in our dataset **S**, there are **8** rows with the target value **YES** and **6** rows with the target value **NO**. The entropy of **S** is calculated as:

Entropy(S) = — (8/14) * $\log_2$(8/14) — (6/14) * $\log_2$(6/14) = 0.99

Note: If all the values in our target column are same the entropy will be zero (meaning that it has no or zero randomness).

We now calculate the Information Gain for each feature:

IG calculation for Fever:

In this(Fever) feature there are **8** rows having value **YES** and **6** rows having value **NO.**

As shown below, in the **8** rows with **YES for** Fever, there are **6** rows having target value **YES** and **2** rows having target value **NO.**

C4.5 is an algorithm used to generate a decision tree developed by Ross Quinlan.[1] C4.5 is an extension of Quinlan's earlier ID3 algorithm. The decision trees generated by C4.5 can be used for classification, and for this reason, C4.5 is often referred to as a statistical classifier. In 2011, authors of the Weka machine learning software described the C4.5 algorithm as "a landmark decision tree program that is probably the machine learning workhorse most widely used in practice to date".[2]

It became quite popular after ranking #1 in the *Top 10 Algorithms in Data Mining* pre-eminent paper published by Springer LNCS in 2008.[3]

Algorithm[edit]

C4.5 builds decision trees from a set of training data in the same way as ID3, using the concept of information entropy. The training data is a set of already classified samples. Each sample consists of a pdimensional vector , where the represent attribute values or features of the sample, as well as the class in which falls.

At each node of the tree, C4.5 chooses the attribute of the data that most effectively splits its set of samples into subsets enriched in one class or the other. The splitting criterion is the normalized information gain (difference in entropy). The attribute with the highest normalized information gain is chosen to make the decision. The C4.5 algorithm then recurses on the partitioned sublists.

This algorithm has a few base cases.

- All the samples in the list belong to the same class. When this happens, it simply creates a leaf node for the decision tree saying to choose that class.
- None of the features provide any information gain. In this case, C4.5 creates a decision node higher up the tree using the expected value of the class.

- Instance of previously unseen class encountered. Again, C4.5 creates a decision node higher up the tree using the expected value.

Pseudocode

In pseudocode, the general algorithm for building decision trees is:[4]

1. Check for the above base cases.
2. For each attribute *a*, find the normalized information gain ratio from splitting on *a*.
3. Let *a_best* be the attribute with the highest normalized information gain.
4. Create a decision *node* that splits on *a_best*.
5. Recurse on the sublists obtained by splitting on *a_best*, and add those nodes as children of *node*.

Naïve Bayes Classifier Algorithm

- Naïve Bayes algorithm is a supervised learning algorithm, which is based on **Bayes theorem** and used for solving classification problems.
- It is mainly used in *text classification* that includes a high-dimensional training dataset. ○ Naïve Bayes Classifier is one of the simple and most effective Classification algorithms which helps in building the fast machine learning models that can make quick predictions.
- **It is a probabilistic classifier, which means it predicts on the basis of the probability of an object**.
- Some popular examples of Naïve Bayes Algorithm are **spam filtration, Sentimental analysis, and classifying articles**.

Why is it called Naïve Bayes?

The Naïve Bayes algorithm is comprised of two words Naïve and Bayes, Which can be described as:

- **Naïve**: It is called Naïve because it assumes that the occurrence of a certain feature is independent of the occurrence of other features. Such as if the fruit is identified on the bases of color, shape, and taste, then red, spherical, and sweet fruit is recognized as an apple. Hence each feature individually contributes to identify that it is an apple without depending on each other.
- **Bayes**: It is called Bayes because it depends on the principle.

Bayes' Theorem:

- Bayes' theorem is also known as **Bayes' Rule** or **Bayes' law**, which is used to determine the probability of a hypothesis with prior knowledge. It depends on the conditional probability. ○ The formula for Bayes' theorem is given as:

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

Where,

P(A|B) is Posterior probability: Probability of hypothesis A on the observed event B.

P(B|A) is Likelihood probability: Probability of the evidence given that the probability of a hypothesis is true.

P(A) is Prior Probability: Probability of hypothesis before observing the evidence.

P(B) is Marginal Probability: Probability of Evidence.

Working of Naïve Bayes' Classifier:

Working of Naïve Bayes' Classifier can be understood with the help of the below example:

Suppose we have a dataset of **weather conditions** and corresponding target variable "**Play**". So using this dataset we need to decide that whether we should play or not on a particular day according to the weather conditions. So to solve this problem, we need to follow the below steps:

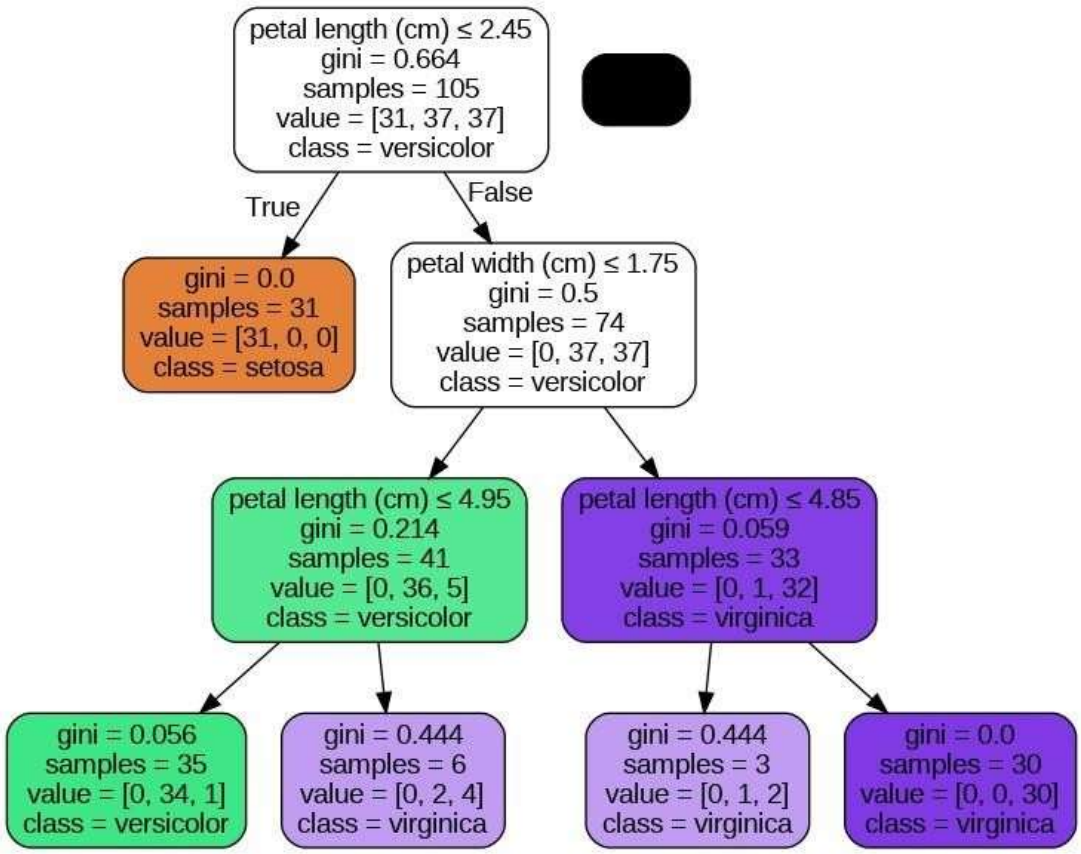
1. Convert the given dataset into frequency tables.
2. Generate Likelihood table by finding the probabilities of given features.
3. Now, use Bayes theorem to calculate the posterior probability.

```
#Classification Trees

from sklearn.datasets import load_iris from sklearn.tree
import DecisionTreeClassifier from
sklearn.model_selection import train_test_split from
sklearn.metrics import accuracy_score

# Load the iris dataset
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, random_state=42) dtc

= DecisionTreeClassifier(max_depth=3)

dtc.fit(X_train, y_train)

y_pred = dtc.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print("Accuracy: {:.2f}%".format(accuracy * 100))
```

```
Accuracy: 100.00% from sklearn.tree
import export_graphviz from
IPython.display import Image import
pydotplus
```

```
dot_data = export_graphviz(dtc, out_file=None,
                    feature_names=iris.feature_names,
class_names=iris.target_names,filled=True, rounded=True,
                                        special_characters=True)

graph = pydotplus.graph_from_dot_data(dot_data)

Image(graph.create_png())
```

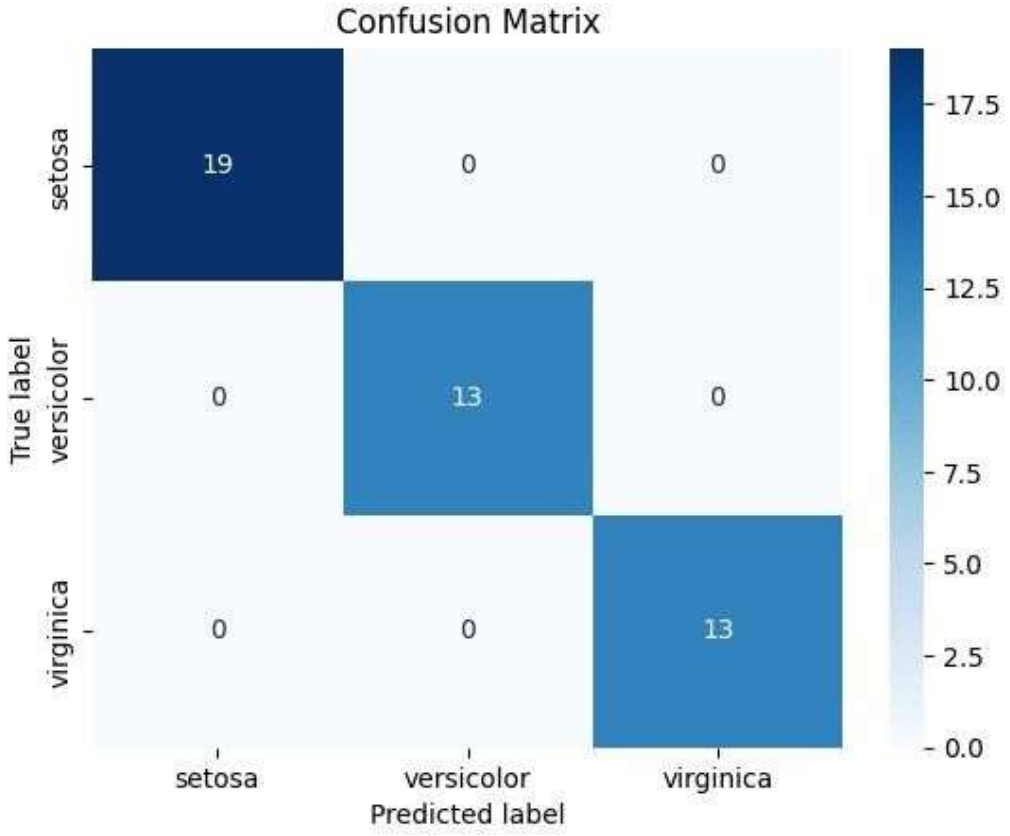



```
from sklearn.model_selection import cross_val_score

scores = cross_val_score(dtc, X, y, cv=10)

print("Accuracy: {:.2f}% (+/- {:.2f})".format(scores.mean() * 100, scores.std() * 2))
```

```
Accuracy: 96.00% (+/- 0.07)
```

```
from sklearn.metrics import confusion_matrix import
seaborn as sns
import matplotlib.pyplot as plt

cm = confusion_matrix(y_test, y_pred)

sns.heatmap(cm, annot=True, cmap="Blues",
```

xticklabels=iris.target_names, yticklabels=iris.target_names) plt.xlabel("Predicted label") plt.ylabel("True label") plt.title("Confusion Matrix") plt.show()

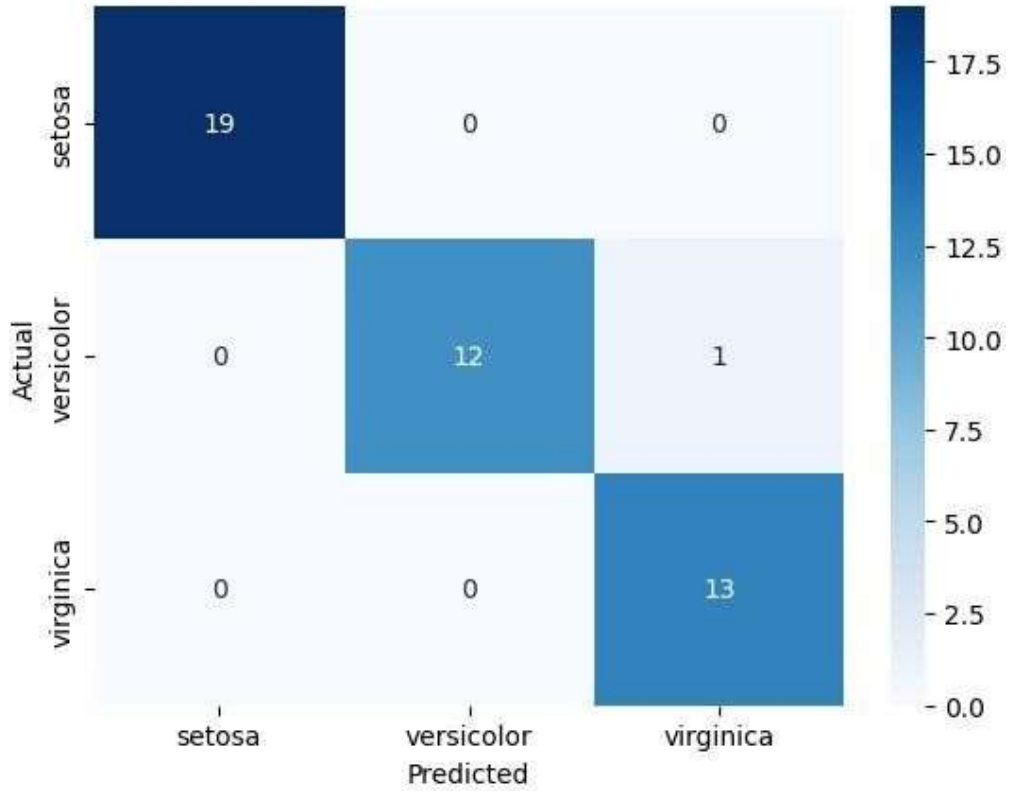



```
#C4.5 from sklearn.tree import
DecisionTreeClassifier from sklearn.datasets
import load_iris
from sklearn.model_selection import train_test_split


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, random_state=42) clf =
DecisionTreeClassifier(criterion='entropy', splitter='best', max_depth=3,
min_samples_split=2,
```

```
                min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None,
random_state=42, max_leaf_nodes=None, min_impurity_decrease=0.0,
class_weight=None, ccp_alpha=0.0)

clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print("Accuracy: {:.2f}%".format(accuracy * 100))
```

Accuracy: 97.78%

```
from sklearn.metrics import confusion_matrix import seaborn as sns
import matplotlib.pyplot as plt

cm = confusion_matrix(y_test, y_pred)

sns.heatmap(cm, annot=True, cmap='Blues', fmt='g',xticklabels=iris.target_names,
yticklabels=iris.target_names) plt.xlabel('Predicted') plt.ylabel('Actual')
plt.show()
```



Naive Bayes Classifier

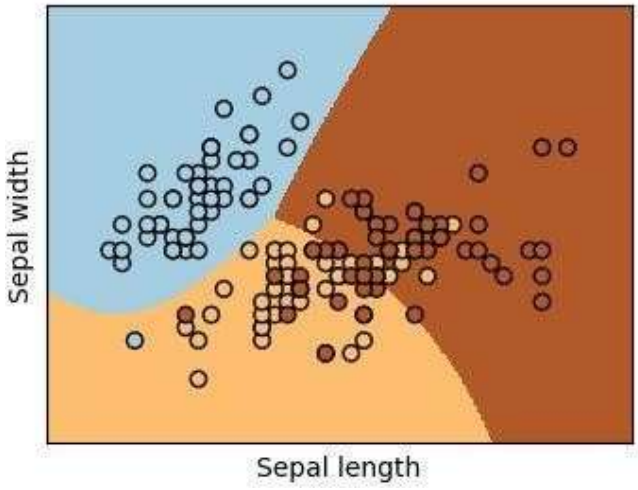
```
from sklearn.datasets import load_iris from
sklearn.model_selection import train_test_split from
sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score


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, random_state=42) gnb

= GaussianNB()
```

```
gnb.fit(X_train, y_train)

y_pred = gnb.predict(X_test)

accuracy = accuracy_score(y_test, y_pred) print("Accuracy:
{:.2f}%".format(accuracy * 100))
```

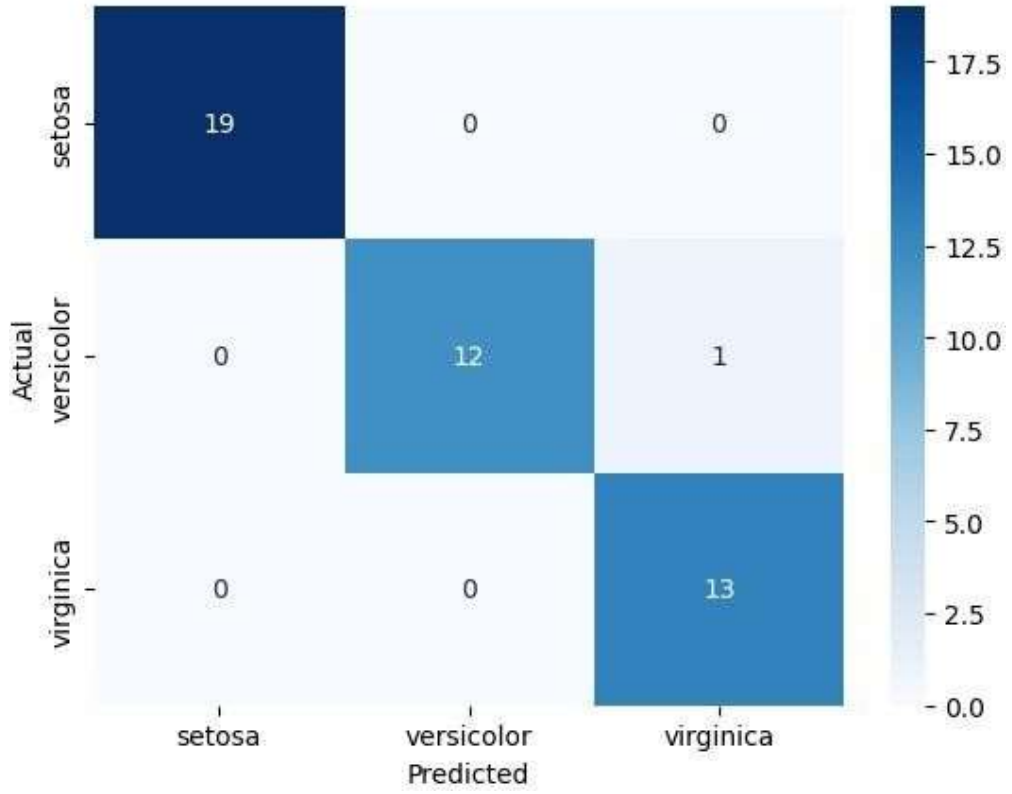
Accuracy: 97.78% **import** numpy **as** np **import** matplotlib.pyplot **as** plt **from** sklearn.datasets **import** load_iris **from** sklearn.model_selection **import** train_test_split

```
from sklearn.naive_bayes import GaussianNB

iris = load_iris() X
= iris.data[:, :2]
y = iris.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42) gnb

= GaussianNB()

gnb.fit(X_train, y_train)

x_min, x_max = X[:, 0].min() - 0.5, X[:, 0].max() + 0.5
y_min, y_max = X[:, 1].min() - 0.5, X[:, 1].max() + 0.5 xx,
yy = np.meshgrid(np.arange(x_min, x_max, 0.01),
np.arange(y_min, y_max, 0.01))


Z = gnb.predict(np.c_[xx.ravel(), yy.ravel()]) Z
= Z.reshape(xx.shape)


plt.figure(1, figsize=(4, 3))
plt.pcolormesh(xx, yy, Z, cmap=plt.cm.Paired)
plt.scatter(X[:, 0], X[:, 1], c=y, edgecolors='k', cmap=plt.cm.Paired) plt.xlabel('Sepal length')
plt.ylabel('Sepal width')

plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
plt.xticks(()) plt.yticks(())
```

plt.show()




```
from sklearn.metrics import confusion_matrix import
seaborn as sns
import matplotlib.pyplot as plt

cm = confusion_matrix(y_test, y_pred)

sns.heatmap(cm, annot=True, cmap='Blues', fmt='g',xticklabels=iris.target_names,
yticklabels=iris.target_names) plt.xlabel('Predicted') plt.ylabel('Actual')
plt.show()
```




```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.naive_bayes import GaussianNB

iris = load_iris()

x_new = np.array([[5.0, 3.5, 1.3, 0.2]]) gnb

= GaussianNB()

gnb.fit(iris.data, iris.target)

y_pred = gnb.predict(x_new)

print("Predicted class label: ", iris.target_names[y_pred[0]]) print("Actual class
label: ", iris.target_names[0])
```

```
Predicted class label: setosa Actual
class label: setosa
```

Conclusion:

In this practical I learned the various classification algorithms.

Practical 5

Aim :- Implementation of K-Means and K-medoid clustering algorithm..

Theory :- K-Means Clustering :-

- K-means clustering is one of the simplest and popular unsupervised machine learning algorithms.
- Typically, unsupervised algorithms make inferences from datasets using only input vectors without referring to known, or labelled, outcomes.
- AndreyBu, who has more than 5 years of machine learning experience and currently teaches people his skills, says that "the objective of K-means is simple: group similar data points together and discover underlying patterns. To achieve this objective, K-means looks for a fixed number (k) of clusters in a dataset."
- A cluster refers to a collection of data points aggregated together because of certain similarities.
- You'll define a target number k, which refers to the number of centroids you need in the dataset. A centroid is the imaginary or real location representing the center of the cluster.
- Every data point is allocated to each of the clusters through reducing the in-cluster sum of squares.
- In other words, the K-means algorithm identifies k number of centroids, and then allocates every data point to the nearest cluster, while keeping the centroids as small as possible.
- The 'means' in the K-means refers to averaging of the data; that is, finding the centroid. ● To process the learning data, the K-means algorithm in data mining starts with a first group of randomly selected centroids, which are used as the beginning points for every cluster, and then performs iterative (repetitive) calculations to optimize the positions of the centroids
- It halts creating and optimizing clusters when either:
- The centroids have stabilized — there is no change in their values because the clustering has been successful.
- The defined number of iterations has been achieved.

Code:- 1) Importing Dataset

```
# importing libraries import
numpy as nm import
matplotlib.pyplot as mtp import
pandas as pd # Importing the
dataset
dataset = pd.read_csv('Mall_Customers.csv')
#dataset dataset
```

Output:-

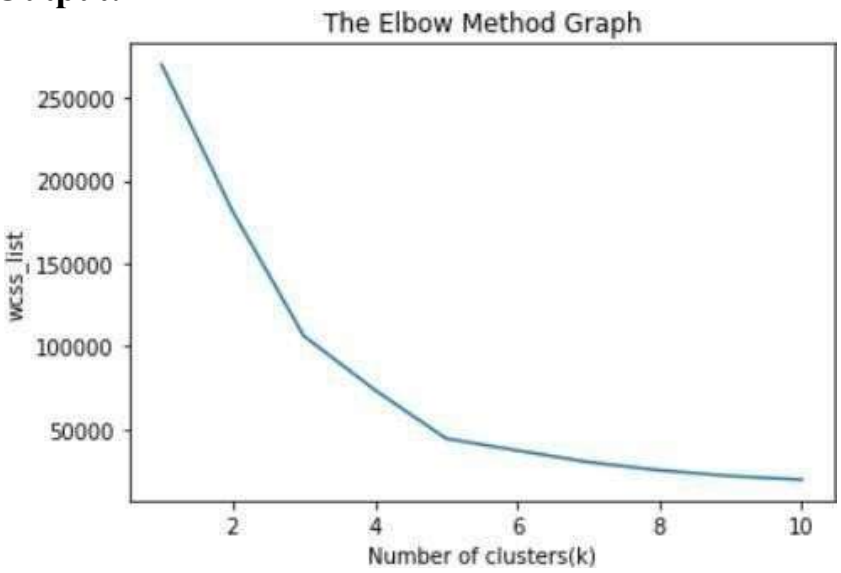
	CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40
...	...	...	...	...	...
195	196	Female	35	120	79
196	197	Female	45	126	28
197	198	Male	32	126	74
198	199	Male	32	137	18
199	200	Male	30	137	83

200 rows × 5 columns

2) Extracting Independent Variables

```
x = dataset.iloc[:, [3, 4]].values
#finding optimal number of clusters using the elbow method
from sklearn.cluster import KMeans
wcss_list= [] #Initializing the list for the values of WCSS

#Using for loop for iterations from 1 to 10.
for i in range(1, 11): kmeans = KMeans(n_clusters=i, init='k-
    means++', random_state= 42) kmeans.fit(x)
    wcss_list.append(kmeans.inertia_)
mtp.plot(range(1, 11), wcss_list) mtp.title('The
Elbow Method Graph') mtp.xlabel('Number of
clusters(k)') mtp.ylabel('wcss_list')
mtp.show()
```

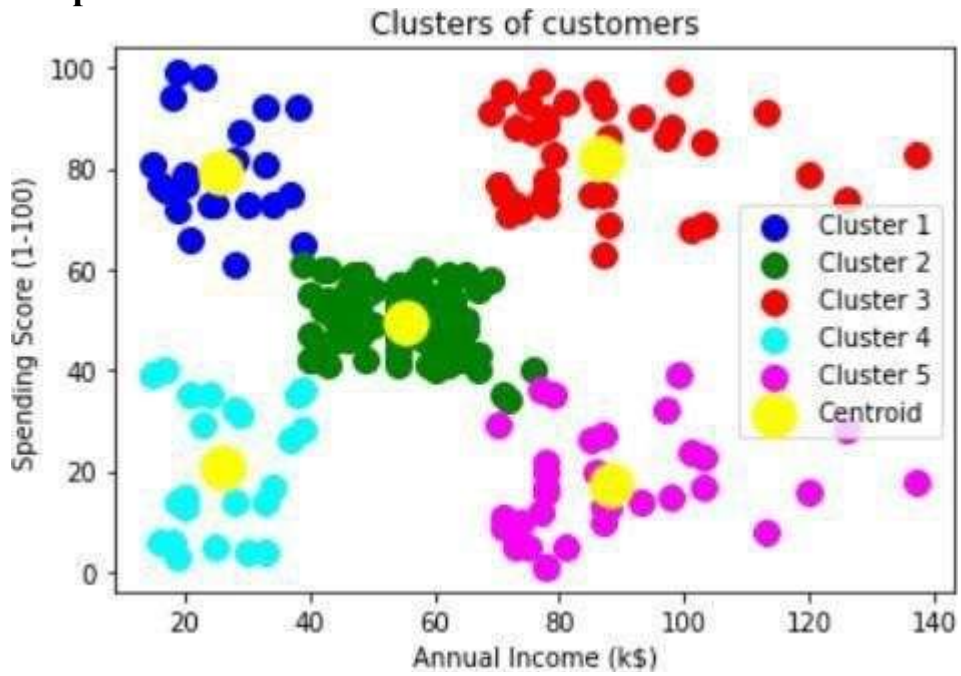
Output:-



3) Training the K-means model on a dataset

```
kmeans = KMeans(n_clusters=5, init='k-means++', random_state= 42)
y_predict= kmeans.fit_predict(x) #visualizing the clusters
mtp.scatter(x[y_predict == 0, 0], x[y_predict == 0, 1], s = 100, c = 'b
lue', label = 'Cluster 1') #for first cluster
mtp.scatter(x[y_predict == 1, 0], x[y_predict == 1, 1], s = 100, c = 'g
reen', label = 'Cluster 2') #for second cluster
mtp.scatter(x[y_predict== 2, 0], x[y_predict == 2, 1], s = 100, c = 're
d', label = 'Cluster 3') #for third cluster
mtp.scatter(x[y_predict == 3, 0], x[y_predict == 3, 1], s = 100, c = 'c
yan', label = 'Cluster 4') #for fourth cluster
mtp.scatter(x[y_predict == 4, 0], x[y_predict == 4, 1], s = 100, c = 'm
agenta', label = 'Cluster 5') #for fifth cluster
mtp.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1
], s = 300, c = 'yellow', label = 'Centroid')
mtp.title('Clusters of customers')
mtp.xlabel('Annual Income (k$)')
mtp.ylabel('Spending Score (1-100)')
mtp.legend() mtp.show()
```

Output:-



K-Medoid

K-medoids clustering is a variant of K-means that is more robust to noises and outliers. Instead of using the mean point as the center of a cluster, K-medoids uses an actual point in the cluster to represent it. Medoid is the most centrally located object of the cluster, with minimum sum of distances to other points. Figure 1 shows the difference between mean and medoid in a 2-D example. The group of points in the right form a cluster, while the rightmost point is an outlier. Mean is greatly influenced by the outlier and thus cannot represent the correct cluster center, while medoid is robust to the outlier and correctly represents the cluster center.

It is a clustering algorithm resembling the K-Means clustering technique. It falls under the category of unsupervised machine learning. It majorly differs from the K-Means algorithm in terms of the way it selects the clusters' centres. The former selects the average of a cluster's points as its centre (which may or may not be one of the data points) while the latter always picks the actual data points from the clusters as their centres (also known as 'exemplars' or 'medoids'). K-Medoids also differ in this respect from the K-Medians algorithm which is the same as K-means, except that it chooses the medians (instead of means) of the clusters as centres.

Code :-

1) Importing Packages & Loading Dataset

```
!pip install scikit-learn-extra
import numpy as np                          #linear algebra
import matplotlib.pyplot as plt
import pandas as pd                         #data processing
```

```
from sklearn.cluster import KMeans from
sklearn_extra.cluster import KMedoids
from sklearn import datasets
#loding Dataset
myiris = datasets.load_iris()
x = myiris.data #we get independent variables y =
myiris.target  #we get target variables myiris
```

Output:-

```
Requirement already satisfied: scipy>=0.19.1 in /usr/local/lib/python3.7/dist-packages (from scikit-learn-extra) (1.4.1)
Requirement already satisfied: numpy>=1.13.3 in /usr/local/lib/python3.7/dist-packages (from scikit-learn-extra) (1.19.5)
Requirement already satisfied: scikit-learn>=0.23.0 in /usr/local/lib/python3.7/dist-packages (from scikit-learn-extra) (0.24.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/dist-packages (from scikit-learn>=0.23.0->scikit-learn-extra) (2.2.0)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages (from scikit-learn>=0.23.0->scikit-learn-extra) (1.0.1)
{'data': array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1],
       [5.4, 3.7, 1.5, 0.2],
       [4.8, 3.4, 1.6, 0.2],
       [4.8, 3. , 1.4, 0.1],
       [4.3, 3. , 1.1, 0.1],
       [5.8, 4. , 1.2, 0.2],
       [5.7, 4.4, 1.5, 0.4],
       [5.4, 3.9, 1.3, 0.4],
```

2) Scaling and Fitting KMedoids

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler().fit(x) x_scaled
= scaler.transform(x)
kMedoids = KMedoids(n_clusters = 3, random_state = 0)
kMedoids.fit(x_scaled)
y_kmed = kMedoids.fit_predict(x_scaled) #to find out the cluster labels
 corresponding to different observations.
```

3) Sillhouete Method to evaluate cluster

```
from sklearn.metrics import silhouette_samples, silhouette_score
kMedoids = KMedoids(n_clusters = 3, random_state = 0)
kMedoids.fit(x_scaled)
y_kmed = kMedoids.fit_predict(x_scaled) silhouette_avg
= silhouette_score(x_scaled, y_kmed)
print(silhouette_avg)                # we get average value of all clusters
```

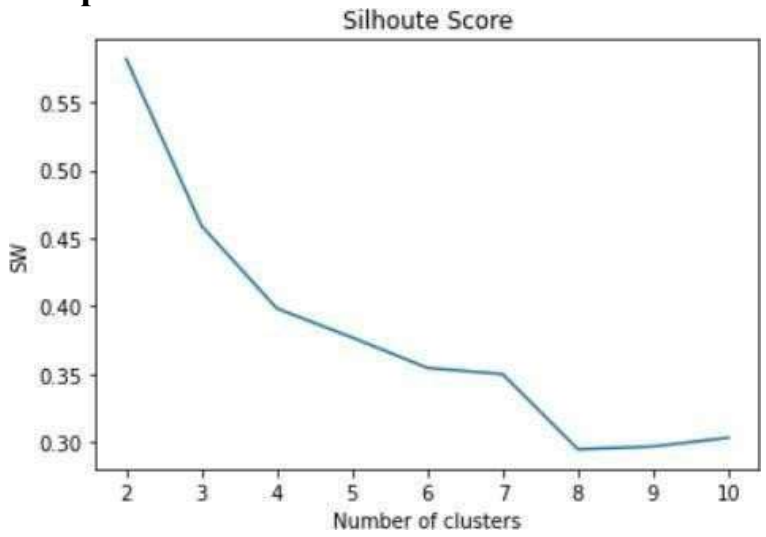
Output:-

```
0.4590416105554613
```

4) Silhouette Width to find number of cluster

```
sw = [] for i
in  range(2,  11):  kMedoids  =  KMedoids(n_clusters  =  i,
random_state = 0) kMedoids.fit(x_scaled)
    y_kmed = kMedoids.fit_predict(x_scaled) silhouette_avg
    = silhouette_score(x_scaled, y_kmed)
       sw.append(silhouette_avg) #different silhouette is calculated fo
r different clusters plt.plot(range(2, 11), sw)
plt.title('Silhoute Score') plt.xlabel('Number of
clusters') plt.ylabel('SW')   #within cluster sum of
squares plt.show()
```

Output:-



5) Computing Purity

```
from sklearn import metrics

def purity_score(y_true, y_pred):
    # compute contingency matrix (also called confusion matrix)
contingency_matrix = metrics.cluster.contingency_matrix(y_true, y_p red)

                                    # return purity return
                                    np.sum(np.amax(contingency_matrix,
                                    axis=0)) / np.sum(conting
ency_matrix)
```

6) How extreme values effect K-Medoid compared to K-means

```
kmeans = KMeans(n_clusters = 3, init = 'random', max_iter = 300, n_init
 = 10, random_state = 0) y_kmeans =
kmeans.fit_predict(x_scaled)
purity_score(y,y_kmeans)
```

Output:-

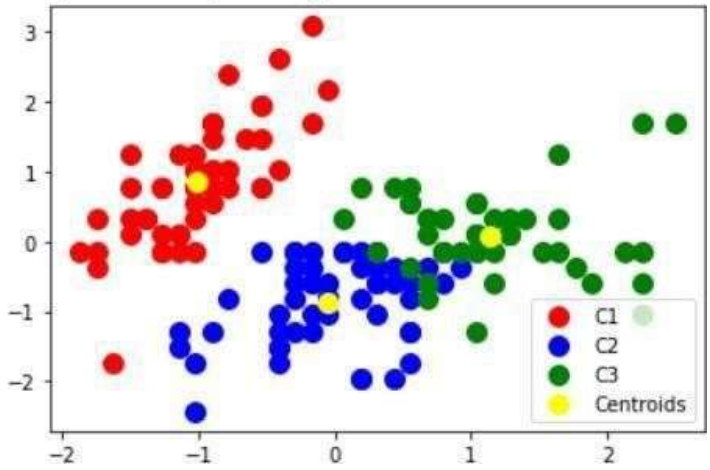
```
0.8333333333333334
```

7) Plotting values

```
plt.scatter(x_scaled[y_kmeans == 0, 0],
x_scaled[y_kmeans == 0, 1], s =
 100, c = 'red', label = 'C1')
plt.scatter(x_scaled[y_kmeans == 1, 0], x_scaled[y_kmeans == 1, 1], s =
 100, c = 'blue', label = 'C2') plt.scatter(x_scaled[y_kmeans == 2, 0],
x_scaled[y_kmeans == 2, 1], s =
 100, c = 'green', label = 'C3')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:,1]
, s = 100, c = 'yellow', label = 'Centroids') plt.legend()
```

Output:-

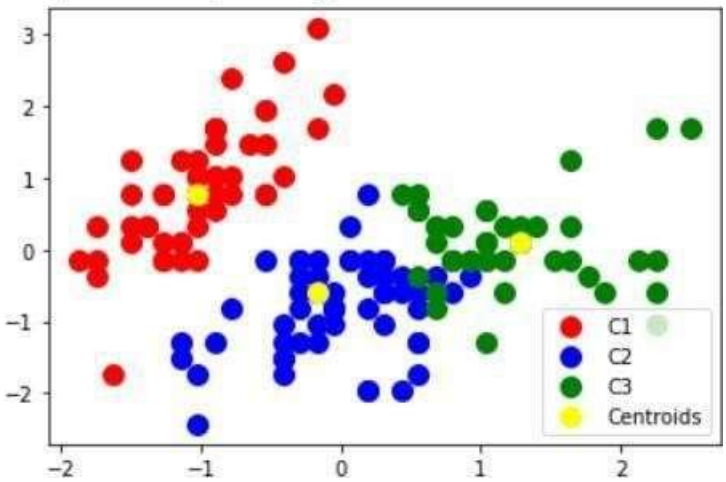
```
<matplotlib.legend.Legend at 0x7f1df1c353d0>
```




```
kmedoids = KMedoids(n_clusters=3, random_state=0).fit(x_scaled)
y_kmed = kmedoids.fit_predict(x_scaled) purity_score(y,y_kmed)
0.84
plt.scatter(x_scaled[y_kmed == 0, 0], x_scaled[y_kmed == 0, 1], s = 100
, c = 'red', label = 'C1')
plt.scatter(x_scaled[y_kmed == 1, 0], x_scaled[y_kmed == 1, 1], s = 100
, c = 'blue', label = 'C2')
plt.scatter(x_scaled[y_kmed == 2, 0], x_scaled[y_kmed == 2, 1], s = 100
, c = 'green', label = 'C3')
plt.scatter(kmedoids.cluster_centers_[:, 0], kmedoids.cluster_centers_[
:,1], s = 100, c = 'yellow', label = 'Centroids')
plt.legend()
```

Output:-

```
<matplotlib.legend.Legend at 0x7f1df1cb8850>
```



8) Adding extreme values

```
import numpy as np
m=np.append(x,[[10,10,10,10],[15,15,15,15],[12,12,12,12]],axis=0)
m.shape
y=np.append(y,[2,2,2]) print(y)
```

```
print("we see 3 observations are added over here.-",m.shape)
```

Output:-

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2]
we see 3 observations are added over here.- (153, 4)
```

```
scaler = StandardScaler().fit(m) x_scaled
= scaler.transform(m)
kmeans = KMeans(n_clusters = 3, init = 'random', max_iter = 300, n_init
 = 10, random_state = 0) y_kmeans =
kmeans.fit_predict(x_scaled)
purity_score(y,y_kmeans)
```

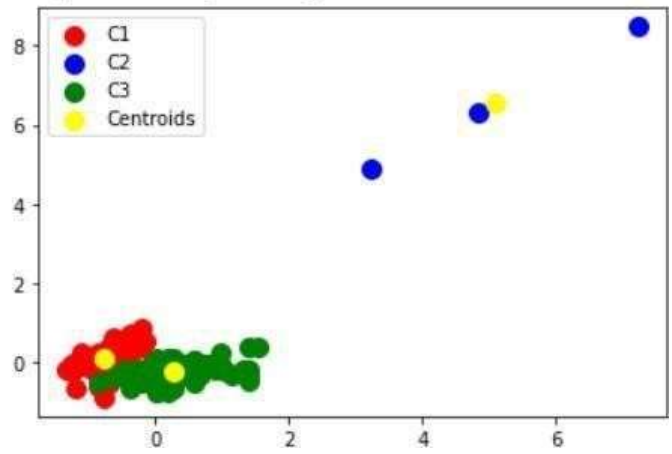
```
0.673202614379085
```

9) Plot

```
plt.scatter(x_scaled[y_kmeans == 0, 0], x_scaled[y_kmeans == 0, 1], s =
 100, c = 'red', label = 'C1') plt.scatter(x_scaled[y_kmeans == 1,
0], x_scaled[y_kmeans == 1, 1], s = 100, c = 'blue', label = 'C2')
plt.scatter(x_scaled[y_kmeans == 2, 0], x_scaled[y_kmeans== 2, 1], s =
 100, c = 'green', label = 'C3') plt.scatter(kmeans.cluster_centers_[:,
0], kmeans.cluster_centers_[:,1]
, s = 100, c = 'yellow', label = 'Centroids') plt.legend()
    #here 3 points formed cluster on their own.
```

Output:-

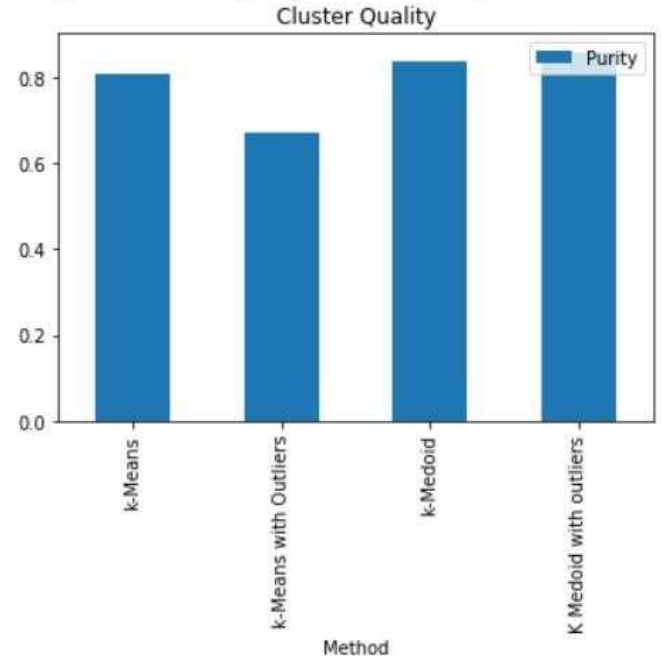
```
<matplotlib.legend.Legend at 0x7f1df1ac1c50>
```




```
data = [['k-Means', 0.81], ['k-Means with Outliers', 0.67], ['k-
Medoid', 0.84],['K Medoid with outliers', 0.86]] df =
pd.DataFrame(data, columns = ['Method', 'Purity'])
df.plot.bar(x='Method',y='Purity',title='Cluster Quality')
```

Output:-

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1df1a4ee10>
```



Conclusion:-

1. K-Medoids are also called PAM(Partition around Medoids)
2. Mean is computed from dataset & median is chosen from dataset.
3. Medoids are representative objects of a dataset or cluster with a dataset whose average dissimilarities to all the objects in the cluster isminimal.

Akash Gupta Div :B Roll No : 83

Practical 6

Aim : Implementation of classifying data using Support vector machines (SVMs)

Theory : Certainly! Here's a theory-based write-up on Support Vector Machines (SVMs) that you can use for reference:

Support Vector Machines (SVMs) are powerful supervised learning models used for classification and regression tasks. They are particularly effective in dealing with complex, highdimensional data and are widely used in various domains such as image recognition, text classification, and bioinformatics.

The main idea behind SVMs is to find an optimal hyperplane that maximally separates the different classes of data points. In a binary classification scenario, the hyperplane is a decision boundary that divides the feature space into two regions, one for each class. SVMs aim to achieve the largest possible margin between the decision boundary and the closest data points from each class. These closest data points, known as support vectors, play a crucial role in defining the hyperplane.

Here are the key concepts and steps involved in SVM:

1. Linear Separability: SVMs assume that the data points are linearly separable, which means that a hyperplane can completely separate the classes. If the data is not linearly separable, SVMs can be extended to handle such cases using techniques like kernel functions.

2. Margin: The margin is the distance between the decision boundary and the closest data points from each class. SVMs aim to maximize this margin, as it represents the confidence of classification. By maximizing the margin, SVMs encourage better generalization and robustness to noise.

3. Support Vectors: Support vectors are the data points that lie on the margin or are misclassified. These points are crucial as they define the hyperplane. SVMs only consider support vectors in the decision-making process, which makes them memory-efficient.

4. Kernel Trick: SVMs can handle nonlinearly separable data by employing kernel functions. Kernel functions transform the input data into a higher-dimensional feature space, where linear separation might be possible. Common kernel functions include linear, polynomial, radial basis function (RBF), and sigmoid. The choice of kernel depends on the problem domain and the characteristics of the data.

5. Regularization: SVMs incorporate a regularization parameter (C) to balance between maximizing the margin and minimizing the classification errors. A larger C value allows for fewer errors on the training set but may lead to overfitting, while a smaller C value increases the

margin but allows more errors. The optimal C value can be determined using techniques like cross-validation.

6. Multi-Class Classification: SVMs are inherently binary classifiers, meaning theyclassify data into two classes. However, SVMs can be extended to handle multi-class classification problems using techniques such as one-vs-one or one-vs-rest approaches. In one-vs-one, SVMs are trained on pairs of classes, while in one-vs-rest, SVMs are trained on each class against the rest.

SVMs have several advantages, such as their ability to handle high-dimensional data, effectiveness in handling non-linearly separable data, and robustness against overfitting. However, they can be computationally expensive for large datasets. Additionally, the choice of the appropriate kernel and the regularization parameter can impact the model's performance, and fine-tuning these hyperparameters is important for optimal results.

In practice, SVMs are implemented in various libraries and frameworks, such as scikit-learn in Python, LIBSVM, and SVMlight. These implementations provide user-friendly interfaces, making it easier to train SVM models, tune hyperparameters, and make predictions on new data.

Support Vector Machines are a powerful and versatile machine learning algorithm with a solid theoretical foundation. Understanding the underlying concepts and steps involved in SVMs can help you effectively apply them to a wide range of classification tasks and derive meaningful insights from your data.

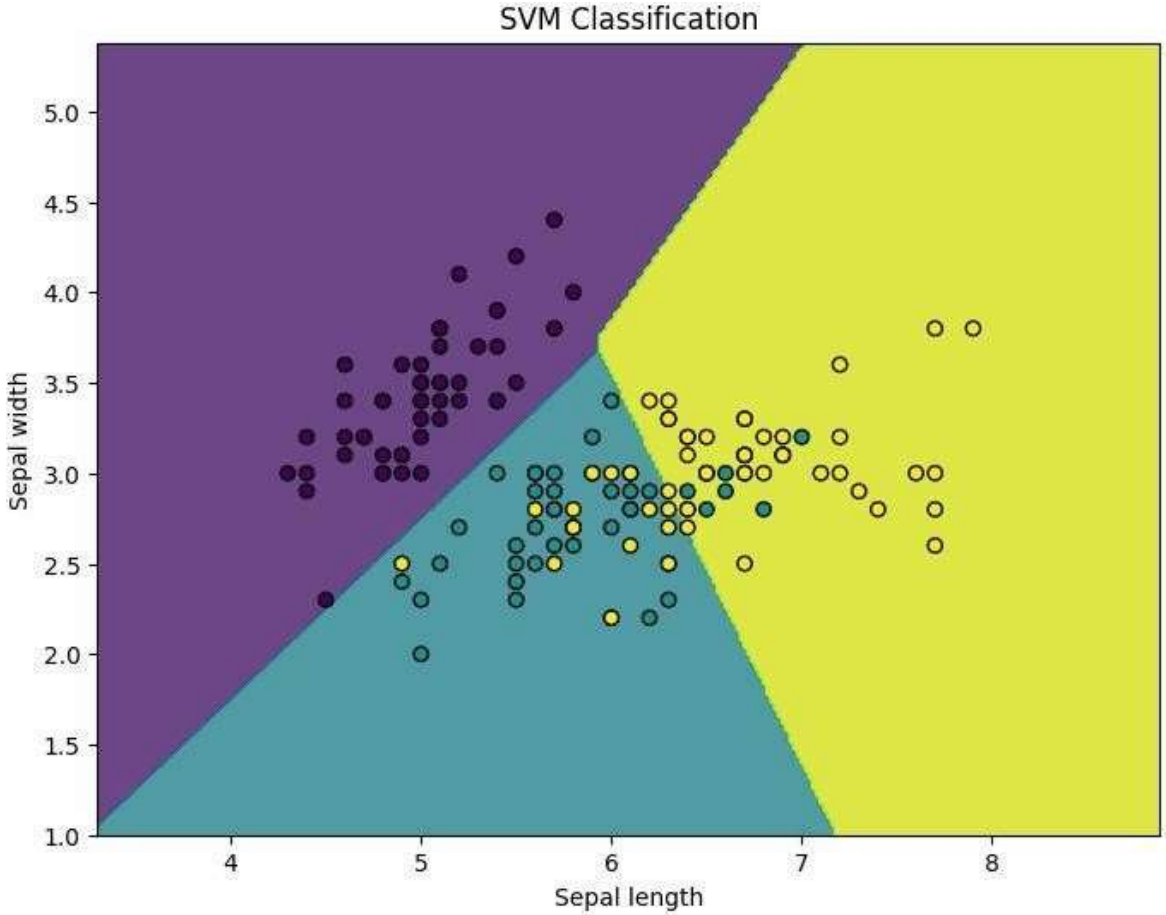
Code :
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score


dataset = datasets.load_iris()

X = dataset.data[:, :2]
y = dataset.target


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
svm = SVC(kernel='linear')
```

```
svm.fit(X_train, y_train)


y_pred = svm.predict(X_test)


accuracy = accuracy_score(y_test, y_pred) print("Accuracy:",
accuracy)


x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1 y_min,
y_max = X[:, 1].min() - 1, X[:, 1].max() + 1

xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.02),
                     np.arange(y_min, y_max, 0.02))

Z = svm.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

plt.figure(figsize=(8, 6)) plt.contourf(xx, yy, Z,
alpha=0.8) plt.scatter(X[:, 0], X[:, 1], c=y,
edgecolors='k') plt.xlabel('Sepal length')
plt.ylabel('Sepal width') plt.title('SVM
Classification')
plt.show()
```

Output :

```
Accuracy: 0.9
```

Akash Gupta Div : B Roll No : 83



Conclusion : Successfully implemented SVM for classifying data.

Akash Gupta Div : B Roll No : 83

Practical 7

Aim : Implementation of classifying data using Non-Linear kernels in Support Vector Machines (SVMs)

Theory :

In Support Vector Machines (SVMs), non-linear kernels are used to handle datasets that are not linearly separable. Linear separation assumes that a straight line or hyperplane can effectively separate the data points into their respective classes. However, many real-world datasets exhibit complex, non-linear relationships among their features, making linear separation insufficient.

Non-linear kernels allow SVMs to transform the original feature space into a higher-dimensional space where linear separation might be possible. Instead of explicitly mapping the data into the higher-dimensional space, kernels implicitly compute the dot product between feature vectors in that space. This technique is known as the "kernel trick" and is computationally efficient compared to explicitly computing the transformed feature vectors.

Here are some commonly used non-linear kernels in SVMs:

1. Radial Basis Function (RBF) Kernel:
- The RBF kernel is one of the most widely used kernels in SVMs. It measures the similarity between data points based on the Euclidean distance.
- The RBF kernel can capture complex non-linear relationships in the data, making it suitable for a wide range of classification problems.
- The RBF kernel is controlled by a parameter called gamma (γ). A higher value of gamma leads to a more flexible decision boundary, which may result in overfitting.

2. Polynomial Kernel:
- The polynomial kernel computes the dot product of feature vectors in polynomial space.
- It captures non-linear relationships by raising the dot product to a specified degree. - The polynomial kernel is controlled by two parameters: the degree of the polynomial (d) and an optional coefficient (c). Higher values of d and c increase the model's complexity.

3. Sigmoid Kernel:
- The sigmoid kernel is based on the sigmoid function, which is commonly used in logistic regression.
- It can capture non-linear relationships and is particularly useful for binary classification problems.

- The sigmoid kernel is controlled by two parameters: gamma (γ) and an optional coefficient (c). Gamma controls the kernel width, while cdetermines the trade-off between the model's complexity and training error.

The choice of the kernel depends on the characteristics of the dataset and the problem at hand. It is crucial to experiment with different kernels and their respective hyperparameters to find the best combination that yields optimal classification results.

One important consideration when using non-linear kernels is the potential for overfitting. Nonlinear kernels can create complex decision boundaries that perfectly fit the training data but may not generalize well to unseen data. Regularization techniques and hyperparameter tuning can help mitigate overfitting.

In practice, SVMs with non-linear kernels are widely used in various domains, including image classification, text analysis, and bioinformatics. They offer the flexibility to handle complex and non-linear relationships in the data, enabling accurate classification in diverse real-world scenarios.

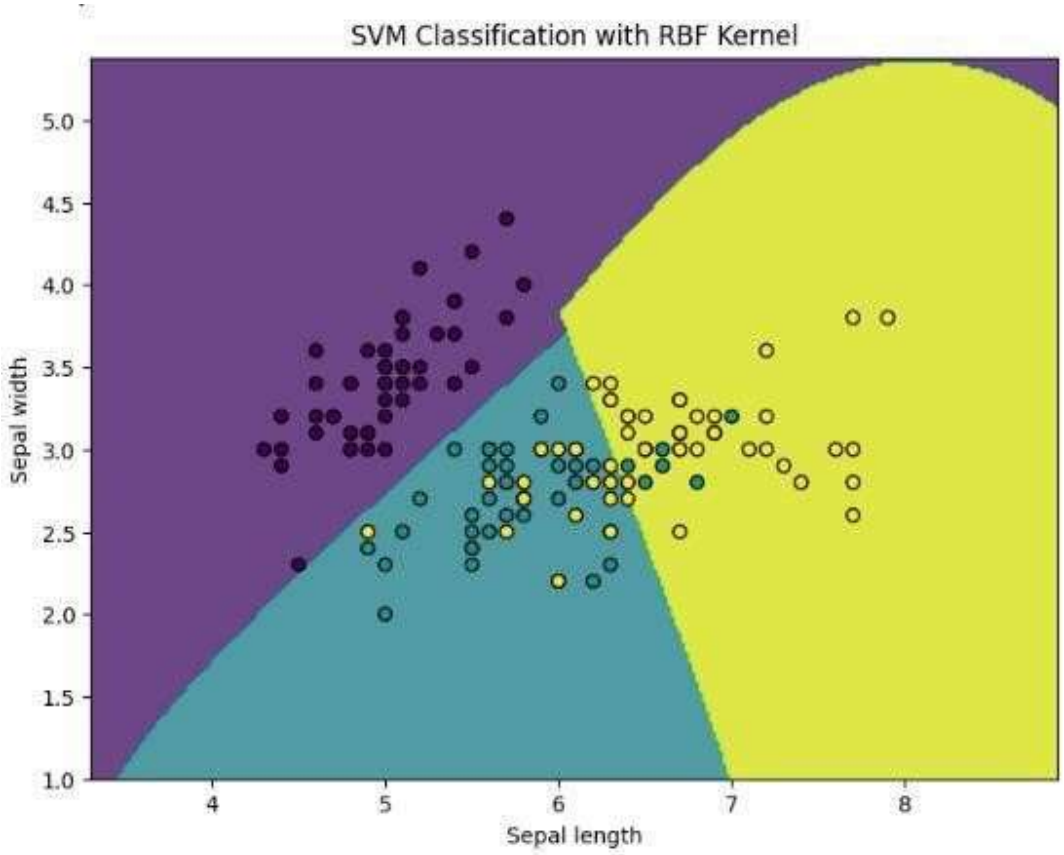
Code :
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score


dataset = datasets.load_iris()


X = dataset.data[:, :2]
y = dataset.target


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


svm = SVC(kernel='rbf')


svm.fit(X_train, y_train)
```

```
y_pred = svm.predict(X_test)


accuracy = accuracy_score(y_test, y_pred) print("Accuracy:",
accuracy)


x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1 y_min,
y_max = X[:, 1].min() - 1, X[:, 1].max() + 1

xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.02),
                     np.arange(y_min, y_max, 0.02))

Z = svm.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

plt.figure(figsize=(8, 6)) plt.contourf(xx, yy, Z,
alpha=0.8) plt.scatter(X[:, 0], X[:, 1], c=y,
edgecolors='k') plt.xlabel('Sepal length')
plt.ylabel('Sepal width') plt.title('SVM
Classification with RBF Kernel') plt.show()
```

Output :

```
Accuracy: 0.9
```

Akash Gupta Div : B Roll No : 83



Conclusion : Successfully implemented Non-linear kernels in Support vector machines.

Akash Gupta Div : B Roll No : 83

Practical 8

Aim : Implementation of bagging Algorithm using Decision Tree and Random Forest

Theory :

Bagging (Bootstrap Aggregating) is an ensemble learning technique that combines multiple classifiers to improve the overall predictive performance. It is a powerful method that helps to reduce variance and increase stability by aggregating the predictions of multiple models trained on different subsets of the data. In Bagging, each classifier is trained independently on a different bootstrap sample of the original dataset.

The Bagging algorithm can be used with various base classifiers, such as Decision Trees and Random Forests. Here, we will discuss how Bagging is implemented using these classifiers.

1. Bagging with Decision Trees:**
Decision Trees are commonly used as the base classifiers in Bagging. In this approach, multiple Decision Tree classifiers are trained on different bootstrap samples of the training data. Each Decision Tree is built using a subset of the features, selected randomly at each node of the tree. The final prediction is made by aggregating the predictions of all individual Decision Trees (e.g., through majority voting for classification or averaging for regression).

Bagging with Decision Trees offers several advantages. It helps to reduce overfitting and improve the model's generalization by combining the predictions of multiple trees. Additionally, it allows capturing complex decision boundaries and handling non-linear relationships between features and targets.

2. Bagging with Random Forest:**
Random Forest is an extension of Bagging that further enhances the performance of Decision Trees. It introduces additional randomness by selecting a random subset of features at each split, resulting in diverse and uncorrelated trees. This randomness helps to reduce the correlation between the individual trees and increase the overall model's accuracy.

Random Forest builds an ensemble of Decision Trees using the Bagging technique. Each tree is constructed by considering a random subset of features at each node, making the trees less correlated and more diverse. During prediction, each tree in the ensemble provides its prediction, and the final prediction is determined by aggregating the individual predictions (e.g., through majority voting for classification or averaging for regression).

Random Forest offers several advantages over individual Decision Trees. It helps to handle high-dimensional datasets, reduce overfitting, and improve the model's generalization. Random

Forests are also robust to outliers and noisy data, making them suitable for a wide range of classification and regression tasks.

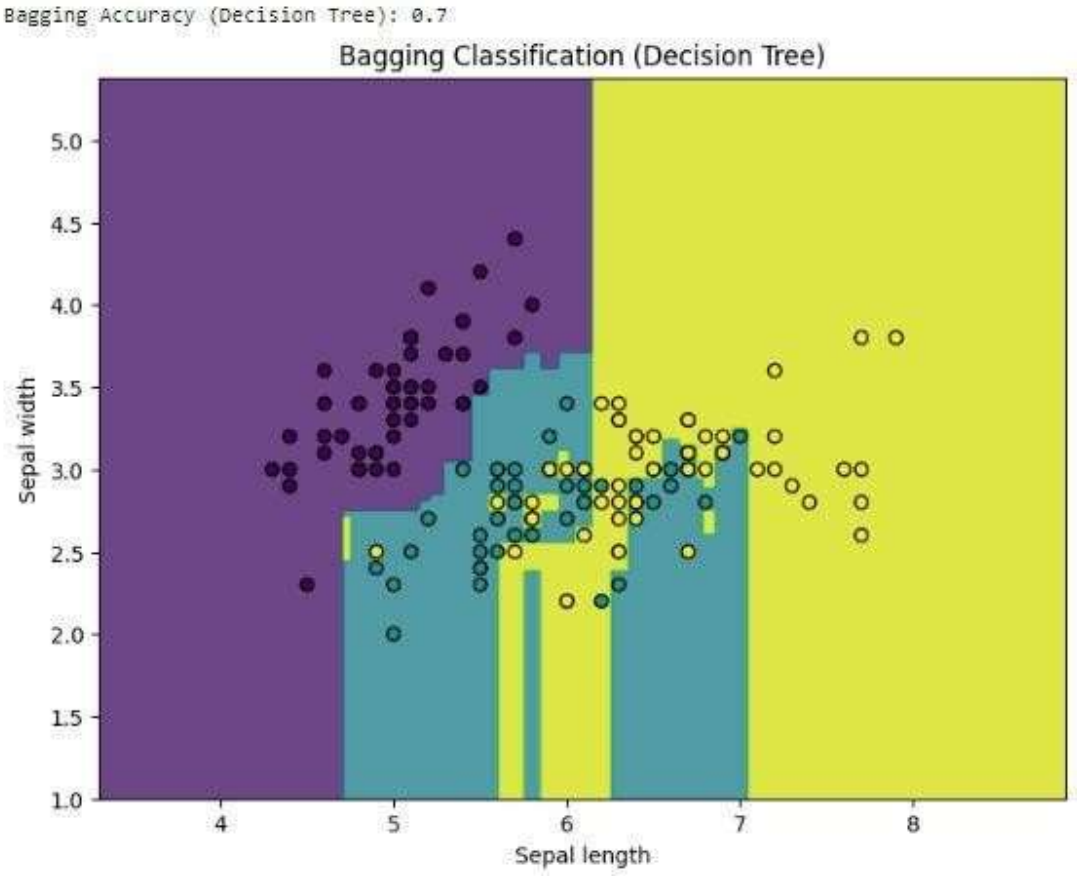
Both Bagging with Decision Trees and Random Forests are powerful ensemble learning techniques that enhance the predictive performance of individual classifiers. They offer improved accuracy, robustness, and generalization capabilities by aggregating multiple models trained on different subsets of the data.

Code :

```
import numpy as np import matplotlib.pyplot as plt
from sklearn.datasets import load_iris from
sklearn.model_selection import train_test_split from
sklearn.tree import DecisionTreeClassifier from
sklearn.ensemble import BaggingClassifier from
sklearn.metrics import accuracy_score


dataset = load_iris()
X, y = dataset.data[:, :2], dataset.target


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


base_classifier = DecisionTreeClassifier()

bagging_classifier = BaggingClassifier(base_classifier, n_estimators=10)


bagging_classifier.fit(X_train, y_train)


y_pred = bagging_classifier.predict(X_test)


accuracy = accuracy_score(y_test, y_pred) print("Bagging
Accuracy (Decision Tree):", accuracy)


x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
```

```
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1

xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.02),
np.arange(y_min, y_max, 0.02))

Z = bagging_classifier.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

plt.figure(figsize=(8, 6)) plt.contourf(xx, yy, Z,
alpha=0.8) plt.scatter(X[:, 0], X[:, 1], c=y,
edgecolors='k') plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.title('Bagging Classification (Decision Tree)') plt.show()
```

Output :

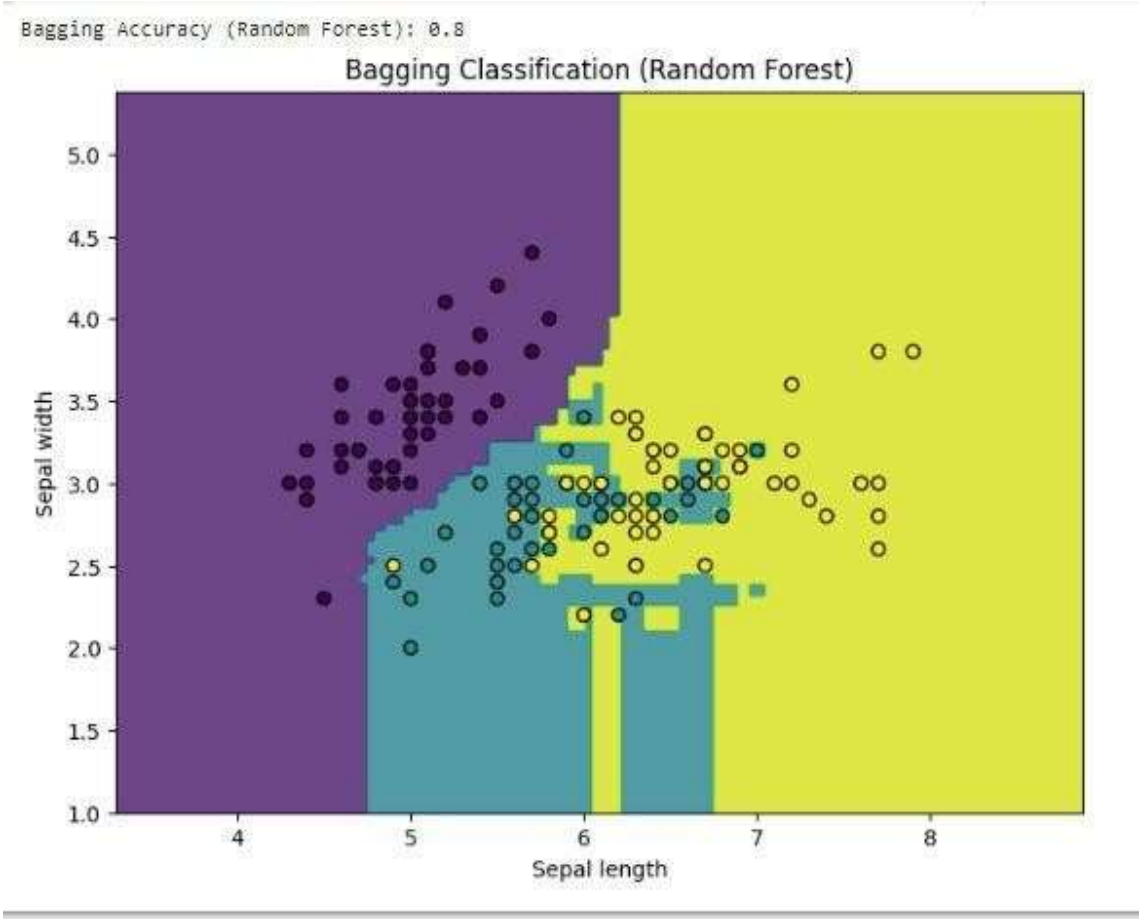


Code :

```
import numpy as np import
matplotlib.pyplot as plt from
sklearn.datasets import
load_iris from
```

```
sklearn.model_selection
import train_test_split from
sklearn.ensemble import
BaggingClassifier from
sklearn.ensemble import
RandomForestClassifier
from sklearn.metrics import
accuracy_score

dataset = load_iris()
X, y = dataset.data[:, :2], dataset.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

base_classifier = RandomForestClassifier(n_estimators=10)

bagging_classifier = BaggingClassifier(base_classifier, n_estimators=10)

bagging_classifier.fit(X_train, y_train)

y_pred = bagging_classifier.predict(X_test)

accuracy = accuracy_score(y_test, y_pred) print("Bagging
Accuracy (Random Forest):", accuracy)

x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1 y_min,
y_max = X[:, 1].min() - 1, X[:, 1].max() + 1

xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.02),
np.arange(y_min, y_max, 0.02))

Z = bagging_classifier.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

plt.figure(figsize=(8, 6)) plt.contourf(xx, yy, Z,
alpha=0.8) plt.scatter(X[:, 0], X[:, 1], c=y,
edgecolors='k') plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.title('Bagging Classification (Random Forest)') plt.show()
```

Output :



Conclusion : Successfully implemented bagging using Decision tree and random forest.

Practical 9

Aim : Implementation of boosting algorithms : Adaboost , Stochastic Gradient Boosting, Voting Ensemble

Theory :

1. AdaBoost (Adaptive Boosting):
AdaBoost is an ensemble learning algorithm that combines multiple weak classifiers to create a strong classifier. Here's a basic implementation outline:

1. Initialize the weights of training samples uniformly.
2. For each boosting round:
a. Train a weak classifier on the weighted training data.

b. Compute the weak classifier's error rate and its weight in the final strong classifier.
c. Update the sample weights based on the misclassified samples.
3. Combine the weak classifiers' predictions using their weights to obtain the final prediction.

The implementation involves training weak classifiers such as decision stumps (simple decision trees with a single split) and updating sample weights. The process iteratively focuses on misclassified samples, assigning them higher weights to improve their importance in subsequent rounds.

2. Stochastic Gradient Boosting (Gradient Boosting Machines):
Stochastic Gradient Boosting is a variant of gradient boosting that combines multiple weak learners in a sequential manner. Here's a simplified implementation outline:

1. Initialize the model with a constant value (e.g., the mean of the target variable).
2. For each boosting round:
a. Compute the negative gradient (residuals) of the loss function with respect to the current model's predictions.
b. Fit a weak learner (e.g., decision tree) to the negative gradient.
c. Update the model by adding a fraction of the weak learner's predictions (learning rate * weak learner's predictions).
3. Repeat steps 2a-2c until the desired number of weak learners is reached.

The implementation involves gradient computation, fitting weak learners to the negative gradient, and updating the model using a learning rate.

3. Voting Ensemble:
Voting ensemble is a technique that combines the predictions of multiple individual models to make a final prediction. Here's a general implementation outline for a majority voting ensemble:

1. Train individual models on different subsets of the training data or using different algorithms.
2. For classification:
a. Each model predicts the class label for a given input.
b. The class labels are tallied, and the class with the majority of votes is chosen as the final prediction.
3. For regression:
a. Each model predicts a continuous value for a given input.
b. The average or median of the predicted values is taken as the final prediction.

The implementation involves training individual models and aggregating their predictions using majority voting (for classification) or averaging/median (for regression).

Akash Gupta Div : B Roll No : 83

Code : AdaBoost

```
import numpy as np
import pandas as pd
from sklearn.ensemble import AdaBoostClassifier
from sklearn.datasets import load_iris from
sklearn.model_selection import train_test_split from
sklearn.metrics import accuracy_score from
sklearn.impute import SimpleImputer from
sklearn.preprocessing import OneHotEncoder


iris = load_iris()

data = pd.DataFrame(data=np.c_[iris['data'], iris['target']], columns=iris['feature_names'] +
['target'])
X = data.drop('target', axis=1)
y = data['target']


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


imputer = SimpleImputer(strategy='mean')
X_train = imputer.fit_transform(X_train)
X_test = imputer.transform(X_test)




ada_boost = AdaBoostClassifier(n_estimators=50)

ada_boost.fit(X_train, y_train)

predictions = ada_boost.predict(X_test)


accuracy = accuracy_score(y_test, predictions)
print("Accuracy:", accuracy) import seaborn as
sns import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, predictions)
plt.figure(figsize=(8, 6))
```

```
sns.heatmap(cm, annot=True, cmap='Blues', fmt='d', cbar=False)
plt.title('Confusion Matrix') plt.xlabel('Predicted Labels')
plt.ylabel('True Labels') plt.show()
```

Output :

```
Accuracy: 1.0
```

Stochastic Gradient Boosting

```
import numpy as np import
pandas as pd import seaborn
as sns import
matplotlib.pyplot as plt
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.datasets import load_iris from
sklearn.model_selection import train_test_split from
sklearn.metrics import accuracy_score, confusion_matrix from
sklearn.impute import SimpleImputer

iris = load_iris()
data = pd.DataFrame(data=np.c_[iris['data'], iris['target']], columns=iris['feature_names'] +
['target'])
X = data.drop('target', axis=1) y
= data['target']


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


imputer = SimpleImputer(strategy='mean')
X_train = imputer.fit_transform(X_train)
X_test = imputer.transform(X_test)

stochastic_boost = GradientBoostingClassifier(n_estimators=50, subsample=0.8) # Adjust the
subsample value

stochastic_boost.fit(X_train, y_train)

predictions = stochastic_boost.predict(X_test)

accuracy = accuracy_score(y_test, predictions) print("Accuracy:",
accuracy)

cm = confusion_matrix(y_test, predictions)
```

```
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, cmap='Blues', fmt='d', cbar=False)
plt.title('Confusion Matrix') plt.xlabel('Predicted Labels')
plt.ylabel('True Labels') plt.show()
```

Output :

```
Accuracy: 1.0
```

Voting Ensemble :

```
import numpy as np import pandas as pd import seaborn
as sns import matplotlib.pyplot as plt from
sklearn.ensemble import VotingClassifier from
sklearn.ensemble import AdaBoostClassifier from
sklearn.ensemble import GradientBoostingClassifier from
sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split from
sklearn.metrics import accuracy_score, confusion_matrix from
sklearn.impute import SimpleImputer

iris = load_iris()

data = pd.DataFrame(data=np.c_[iris['data'], iris['target']], columns=iris['feature_names'] +
['target'])
X = data.drop('target', axis=1)
y = data['target']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

imputer = SimpleImputer(strategy='mean')
X_train = imputer.fit_transform(X_train)
X_test = imputer.transform(X_test)

ada_boost = AdaBoostClassifier(n_estimators=50)
stochastic_boost = GradientBoostingClassifier(n_estimators=50, subsample=0.8)

voting_ensemble = VotingClassifier(estimators=[('ada_boost', ada_boost), ('stochastic_boost',
stochastic_boost)])

voting_ensemble.fit(X_train, y_train)

predictions = voting_ensemble.predict(X_test)
```

```
accuracy = accuracy_score(y_test, predictions) print("Accuracy:",
accuracy)

cm = confusion_matrix(y_test, predictions)

plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, cmap='Blues', fmt='d', cbar=False)
plt.title('Confusion Matrix') plt.xlabel('Predicted Labels')
plt.ylabel('True Labels') plt.show()
```

Output :

```
Accuracy: 1.0
```

Conclusion :
Successfully implemented Boosting algorithms.

Practical 10

Aim : Implementation of dimensionality reduction techniques : features extraction and selection , Normalization, Transformation , Principal Component analysis

Theory :
Dimensionality reduction techniques play a crucial role in machine learning and data analysis to handle high-dimensional data effectively. Here is an overview of the implementation of some common dimensionality reduction techniques:

1. Feature Extraction and Selection:
- Feature extraction involves transforming the original features into a new set of lower-dimensional features. Techniques like Principal Component Analysis (PCA), Linear Discriminant Analysis (LDA), and t-distributed Stochastic Neighbor Embedding (t-SNE) can be used.
- Feature selection aims to identify the most relevant features from the original feature set. Methods like Recursive Feature Elimination (RFE), L1 regularization (Lasso), and mutual information-based selection can be employed.

2. Normalization:
- Normalization is used to scale the features to a standard range, ensuring that they have similar magnitudes. Common normalization techniques include Min-Max scaling, z-score normalization (StandardScaler), and robust scaling (RobustScaler).
- Implementation depends on the programming language or library used. For example, in Python with scikit-learn, you can import the appropriate scaler class and apply it to your data.

3. Transformation:
- Transformation techniques, such as logarithmic transformation or power transformation, can be applied to adjust the distribution of features to meet specific assumptions of certain models (e.g., normality assumption for linear regression).
- Implementation involves applying the desired transformation function to the feature values. Libraries like NumPy or pandas in Python can be used for this purpose.

4. Principal Component Analysis (PCA):
- PCA is a widely used technique for dimensionality reduction. It identifies the principal components that capture the maximum variance in the data.
- Implementation typically involves using a library like scikit-learn in Python. You can import the PCA class, fit it to your data, and then transform your data using the learned transformation.

Code :

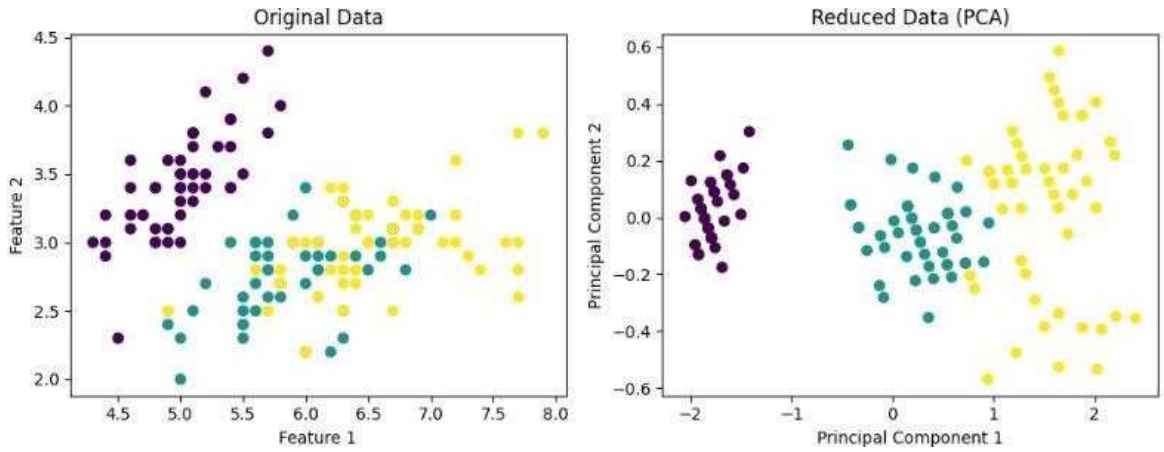
```
import numpy as np import matplotlib.pyplot as plt from
sklearn.decomposition import PCA from sklearn.preprocessing import
MinMaxScaler from sklearn.feature_selection import SelectKBest,
mutual_info_classif from sklearn.preprocessing import
PowerTransformer from sklearn.datasets import load_iris

# Load the dataset data
= load_iris()

# Separate the features (X) and the target variable (y)
X = data.data y = data.target

# Feature extraction using SelectKBest with mutual information
selector = SelectKBest(score_func=mutual_info_classif, k=2) # Select top 2 features X_selected
= selector.fit_transform(X, y)

# Normalization using Min-Max scaling scaler
= MinMaxScaler()
X_normalized = scaler.fit_transform(X_selected)

# Transformation using PowerTransformer
transformer = PowerTransformer(method='yeo-johnson') # Can also use 'box-cox'
X_transformed = transformer.fit_transform(X_normalized)

# Principal Component Analysis (PCA) for dimensionality reduction pca
= PCA(n_components=2) # Reduce to 2 principal components
```

```
X_pca = pca.fit_transform(X_transformed)

# Print the original and reduced data
print("Original Data:") print(X)
print("\nReduced Data (PCA):")
print(X_pca)

# Visualize the original and reduced data plt.figure(figsize=(10,
4))
plt.subplot(1, 2, 1) plt.scatter(X[:,
0], X[:, 1], c=y)
plt.title("Original Data")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")

plt.subplot(1, 2, 2) plt.scatter(X_pca[:,
0], X_pca[:, 1], c=y) plt.title("Reduced
Data (PCA)") plt.xlabel("Principal
Component 1") plt.ylabel("Principal
Component 2")

plt.tight_layout() plt.show()
```

Output :

```
Original Data:
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5.  3.4 1.5 0.2]
 [4.4 2.9 1.4 0.2]
 [4.9 3.1 1.5 0.1]]
Reduced Data (PCA):
[[-1.85817383e+00 -3.14644996e-03]
 [-1.85817383e+00 -3.14644996e-03]
 [-1.89178500e+00  3.04647128e-02]
 [-1.82423795e+00 -3.70823307e-02]
 [-1.85817383e+00 -3.14644996e-03]
 [-1.56869616e+00  8.07836291e-02]
 [-1.76480254e+00  9.02248466e-02]
 [-1.82423795e+00 -3.70823307e-02]
 [-1.85817383e+00 -3.14644996e-03]
 [-1.91764953e+00 -1.30493909e-01]
 [-1.82423795e+00 -3.70823307e-02]
 [-1.78997936e+00 -7.13409191e-02]] \
```

Akash Gupta Div : B Roll No : 83



Conclusion : Successfully implemented Dimensionality reduction techniques .

Akash Gupta Div : B Roll No : 83

Practical No 11

Aim : implementation of dimensionality reduction techniques : features extraction and selection , Normalization, Transformation , Principal Component analysis

Theory :
Dimensionality reduction techniques play a crucial role in machine learning and data analysis to handle high-dimensional data effectively. Here is an overview of the implementation of some common dimensionality reduction techniques:

1. Feature Extraction and Selection:
- Feature extraction involves transforming the original features into a new set of lower-dimensional features. Techniques like Principal Component Analysis (PCA), Linear Discriminant Analysis (LDA), and t-distributed Stochastic Neighbor Embedding (t-SNE) can be used.
- Feature selection aims to identify the most relevant features from the original feature set. Methods like Recursive Feature Elimination (RFE), L1 regularization (Lasso), and mutual information-based selection can be employed.

2. Normalization:
- Normalization is used to scale the features to a standard range, ensuring that they have similar magnitudes. Common normalization techniques include Min-Max scaling, z-score normalization (StandardScaler), and robust scaling (RobustScaler).
- Implementation depends on the programming language or library used. For example, in Python with scikit-learn, you can import the appropriate scaler class and apply it to your data.

3. Transformation:
- Transformation techniques, such as logarithmic transformation or power transformation, can be applied to adjust the distribution of features to meet specific assumptions of certain models (e.g., normality assumption for linear regression).
- Implementation involves applying the desired transformation function to the feature values. Libraries like NumPy or pandas in Python can be used for this purpose.

4. Principal Component Analysis (PCA):
- PCA is a widely used technique for dimensionality reduction. It identifies the principal components that capture the maximum variance in the data.
- Implementation typically involves using a library like scikit-learn in Python. You can import the PCA class, fit it to your data, and then transform your data using the learned transformation.

Code :

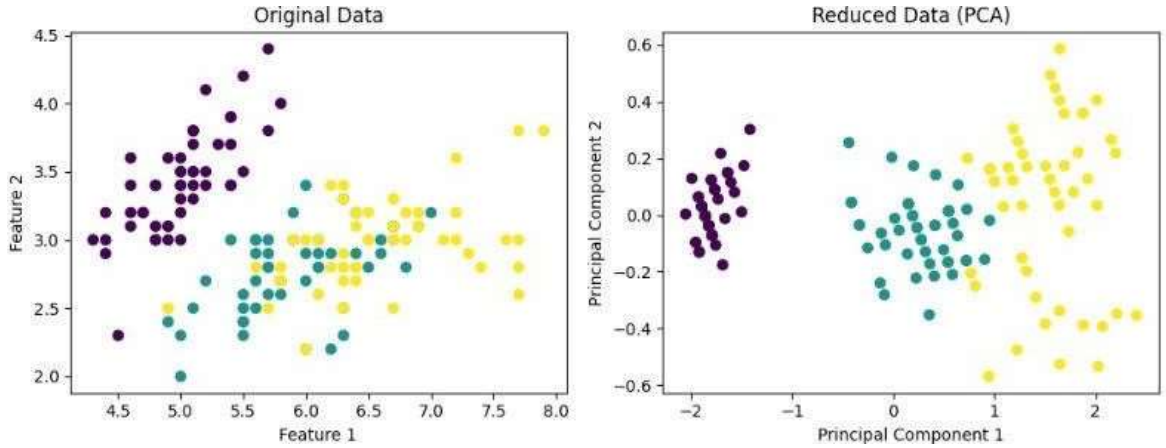
```
import numpy as np import matplotlib.pyplot as plt from
sklearn.decomposition import PCA from sklearn.preprocessing import
MinMaxScaler from sklearn.feature_selection import SelectKBest,
mutual_info_classif from sklearn.preprocessing import
PowerTransformer from sklearn.datasets import load_iris

# Load the dataset data
= load_iris()

# Separate the features (X) and the target variable (y)
X = data.data y = data.target

# Feature extraction using SelectKBest with mutual information
selector = SelectKBest(score_func=mutual_info_classif, k=2) # Select top 2 features X_selected
= selector.fit_transform(X, y)

# Normalization using Min-Max scaling scaler
= MinMaxScaler()
X_normalized = scaler.fit_transform(X_selected)

# Transformation using PowerTransformer
transformer = PowerTransformer(method='yeo-johnson') # Can also use 'box-cox'
X_transformed = transformer.fit_transform(X_normalized)

# Principal Component Analysis (PCA) for dimensionality reduction pca
= PCA(n_components=2) # Reduce to 2 principal components
X_pca = pca.fit_transform(X_transformed)

# Print the original and reduced data
print("Original Data:") print(X)
print("\nReduced Data (PCA):")
print(X_pca)

# Visualize the original and reduced data
plt.figure(figsize=(10, 4)) plt.subplot(1,
2, 1) plt.scatter(X[:, 0], X[:, 1], c=y)
plt.title("Original Data")
```

```
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")

plt.subplot(1, 2, 2) plt.scatter(X_pca[:,
0], X_pca[:, 1], c=y) plt.title("Reduced
Data (PCA)") plt.xlabel("Principal
Component 1") plt.ylabel("Principal
Component 2")

plt.tight_layout() plt.show()
```

Output :

```
Original Data:
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5.  3.4 1.5 0.2]
 [4.4 2.9 1.4 0.2]
 [4.9 3.1 1.5 0.1]
Reduced Data (PCA):
[[-1.85817383e+00 -3.14644996e-03]
 [-1.85817383e+00 -3.14644996e-03]
 [-1.89178500e+00  3.04647128e-02]
 [-1.82423795e+00 -3.70823307e-02]
 [-1.85817383e+00 -3.14644996e-03]
 [-1.56869616e+00  8.07836291e-02]
 [-1.76480254e+00  9.02248466e-02]
 [-1.82423795e+00 -3.70823307e-02]
 [-1.85817383e+00 -3.14644996e-03]
 [-1.91764953e+00 -1.30493909e-01]
 [-1.82423795e+00 -3.70823307e-02]
 [-1.78997936e+00 -7.13409191e-02]] \
```

Akash Gupta Div : B Roll No : 83



Conclusion : Successfully implemented Dimensionality reduction techniques .

Practical: 12

Aim : Deployment of Machine Learning Model using Flask Trained Model

Theory:

1. Iris Dataset: The Iris dataset is a popular dataset in machine learning and statistics. It consists of measurements of sepal and petal dimensions for three different species of Iris flowers: Setosa, Versicolor, and Virginica. Each sample in the dataset has four features: sepal length, sepal width, petal length, and petal width. The goal is to train a model that can classify new Iris flowers based on their measurements.

2. Decision Tree Classifier: A decision tree is a supervised machine learning algorithm that can be used for classification tasks. It builds a tree-like model of decisions and their possible consequences. In this project, a decision tree classifier is used to classify Iris flowers based on their measurements. The classifier learns from the training data, which consists of labeled samples with known flower species, and then makes predictions for new, unseen data.

3. Flask: Flask is a web framework for building Python web applications. It provides tools and libraries for handling HTTP requests, rendering HTML templates, and managing routes. In this project, Flask is used to create a simple web application that allows users to input Iris flower measurements and obtain predictions from the trained decision tree classifier.

Code :

app.py

```
from flask import Flask, render_template, request
import iris_model app = Flask(_name_)
@app.route('/', methods=['GET', 'POST']) def
basic():     if request.method == 'POST':
sepal_length = request.form['sepallength']
sepal_width = request.form['petalwidth']
petal_length = request.form['petallength']
petal_width = request.form['petalwidth']
      y_pred = [[sepal_length, sepal_width, petal_length, petal_width]]
trained_model = iris_model.training_model()        prediction_value =
trained_model.predict(y_pred)        setosa = 'The flower is classified
as Setosa'        versicolor = 'The flower is classified as Versicolor'
virginica = 'The flower is classified as Virginica'        if
prediction_value == 0:            return render_template('index.html',
setosa=setosa)
      elif prediction_value == 1:            return
render_template('index.html', versicolor=versicolor)        else:
```

```
        return render_template('index.html', virginica=virginica)
return render_template('index.html') if__name___==
'_ main_ ':     app.run(debug=True)
```

iris_model.py

```
import pandas as pd from
sklearn.datasets import load_iris data
= load_iris()
# print(data.DESCR)
X = pd.DataFrame(data.data, columns=(data.feature_names))
y = pd.DataFrame(data.target, columns=['Target']) from
sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1, test_size=0.3)
from sklearn.tree import DecisionTreeClassifier def training_model():     model =
DecisionTreeClassifier()     trained_model = model.fit(X_train, y_train)     return
trained_model
```

iris.css

```
html {  font-size:
14px;
}
@media (min-width: 768px) {
  html {     font-
size: 16px;
  }
}
.container {
  max-width: 960px;
}
.card-deck .card {
  min-width: 220px;
}
```

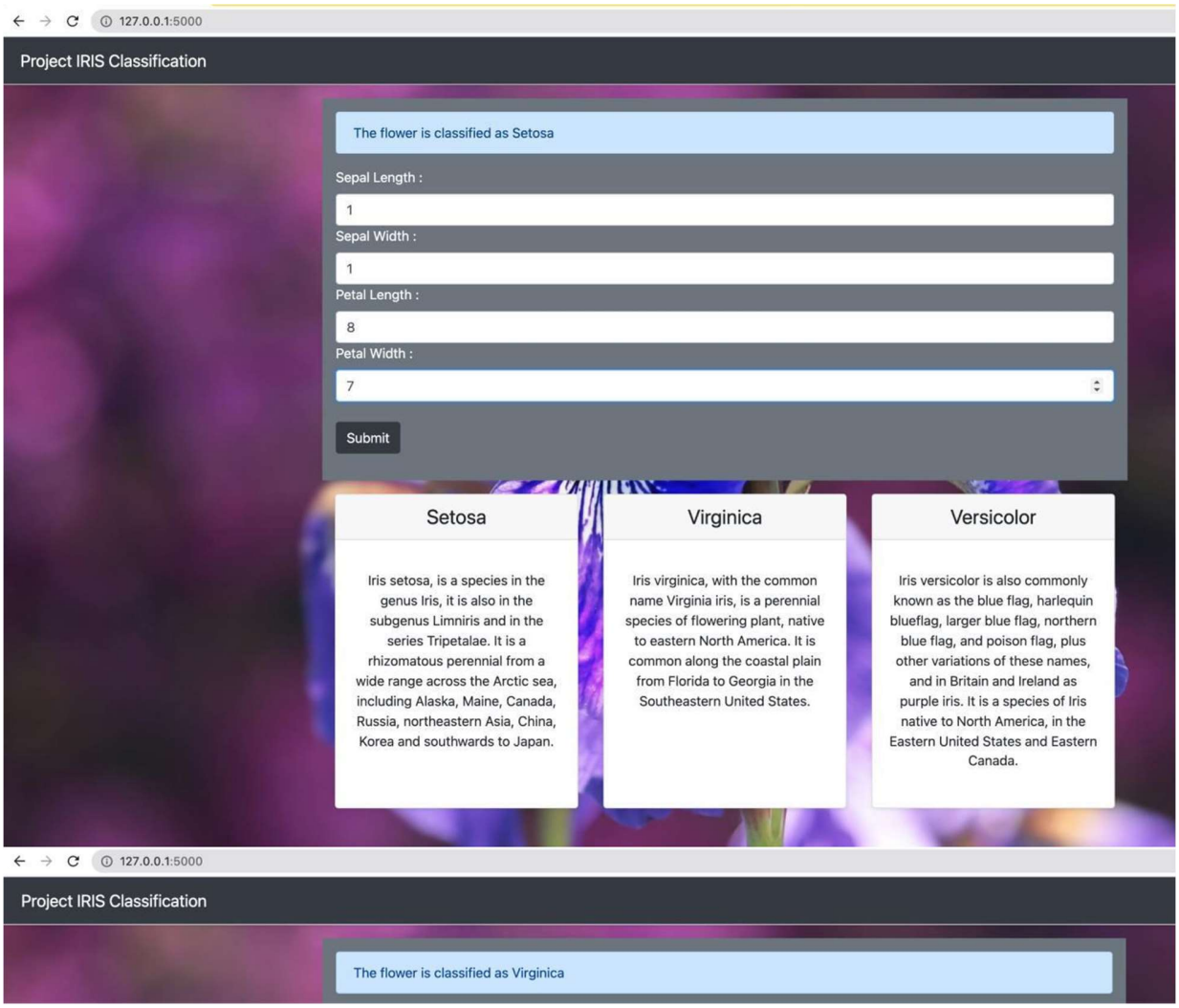
index.html

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
<title>Iris Project</title>
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/css/bootstrap.min.css"
integrity="sha384-9aIt2nRpC12Uk9gS9baDl411NQApFmC26EwAOH8WgZl5MYYxFfc+NcPb1dKGj7Sk"
crossorigin="anonymous">
```

```
    <style>
      .bd-placeholder-img {
font-size: 1.125rem;         text-
anchor: middle;         -webkit-user-
select: none;
        -moz-user-select: none;         -ms-
user-select: none;         user-select:
none;
      }
      body {
         background-image: url('../static/iris-flowers.jpg');
      }
      @media (min-width: 768px) {         .bd-placeholder-
img-lg {
         font-size: 3.5rem;
       }
      }
    </style>
    <!-- Custom styles for this template -->
    <link href="../static/css/iris.css" rel="stylesheet">
  </head>
<body>
    <div class="d-flex flex-column flex-md-row align-items-center p-3 px-md-4 mb-3 bg-dark text-white
border-bottom shadow-sm">
    <h5 class="my-0 mr-md-auto font-weight-normal">Project IRIS Classification</h5>
</div>

    <div class="container p-3 my-3 bg-secondary text-white">
      {% if setosa %}
        <div class="alert alert-primary">{{setosa}}</div>
      {% elif versicolor %}
        <div class="alert alert-primary">{{versicolor}}</div>
      {% elif virginica %}
        <div class="alert alert-primary">{{virginica}}</div>
      {% endif %}
    <form class="form-group" action="/" method="POST">
      <label for="sepallength">Sepal Length :</label>
      <input class="form-control" type="number" name="sepallength">
      <label for="sepalwidth">Sepal Width :</label>
      <input class="form-control" type="number" name="sepalwidth">
      <label for="petallength">Petal Length :</label>
      <input class="form-control" type="number" name="petallength">
      <label for="petalwidth">Petal Width :</label>
      <input class="form-control" type="number" name="petalwidth">
    </br>
      <button class="btn btn-dark" type="submit" name="submit">Submit</button>
</form>
    </div>
```

```
  <div class="container">
    <div class="card-deck mb-3 text-center">
      <div class="card mb-4 shadow-sm">
      <div class="card-header">
        <h4 class="my-0 font-weight-normal">Setosa</h4>
      </div>
      <div class="card-body">
        <ul class="list-unstyled mt-3 mb-4">
          <li>Iris setosa, is a species in the genus Iris, it is also in the subgenus Limniris and in the series Tripetalae. It is a rhizomatous perennial from a wide range across the Arctic sea, including Alaska, Maine, Canada, Russia, northeastern Asia, China, Korea and southwards to Japan.</li>
        </ul>
      </div>
      </div>
      <div class="card mb-4 shadow-sm">
      <div class="card-header">
        <h4 class="my-0 font-weight-normal">Virginica</h4>
      </div>
      <div class="card-body">
        <ul class="list-unstyled mt-3 mb-4">
        <li>Iris virginica, with the common name Virginia iris, is a perennial species of flowering plant, native to eastern North America. It is common along the coastal plain from Florida to Georgia in the Southeastern United States.</li>
        </ul>
      </div>
      </div>
      <div class="card mb-4 shadow-sm">
      <div class="card-header">
        <h4 class="my-0 font-weight-normal">Versicolor</h4>
      </div>
      <div class="card-body">
        <ul class="list-unstyled mt-3 mb-4">
        <li>Iris versicolor is also commonly known as the blue flag, harlequin blueflag, larger blue flag, northern blue flag, and poison flag, plus other variations of these names, and in Britain and Ireland as purple iris. It is a species of Iris native to North America, in the Eastern United States and Eastern Canada.</li>
        </ul>
      </div>
    </div>
  </div>

  </div>
</body>
</html>
```

Output :



Conclusion:

The project demonstrates the implementation of a basic web application for Iris flower classification using Flask and a decision tree classifier. By following the steps outlined in the code, the application allows users to input values for sepal length, sepal width, petal length, and petal width through a web form. The trained decision tree classifier then predicts the class of the Iris flower based on the input values.

The project showcases how machine learning algorithms can be integrated into web applications to provide interactive and user-friendly interfaces for making predictions. It highlights the use of Flask as a web framework for handling HTTP requests and rendering dynamic HTML templates.

However, it's important to note that this project is a simplified example for educational purposes. In real-world scenarios, more advanced machine learning algorithms, data preprocessing techniques, and model evaluation methods may be necessary for achieving higher accuracy and

robustness. Additionally, the project can be extended by incorporating more features, improving the user interface, and deploying the application to a production environment.

PRACTICAL - 13

AIM:

Implementation of logic programming using PROLOG-DFS for the water jug problem.

THEORY:

Prolog:

Prolog is a logic programming language. It has important role in artificial intelligence. Unlike many other programming languages, Prolog is intended primarily as a declarativeprogramming language. In prolog, logic is expressed as relations (called as Facts and Rules). Core heart of prolog lies at the logic being applied. Formulation or Computation is carried out by running a query over these relations.

Q1 . Perform addition, subtraction, multiplication and division operations on 2 numbers.

```
sum(X,Y,Z):- Z is X+Y.

diff(X,Y,Z):-Z is X-Y.

mult(X,Y,Z):-Z is X*Y.

divide(X,Y,Z):-Z is X/Y. ouput:
```

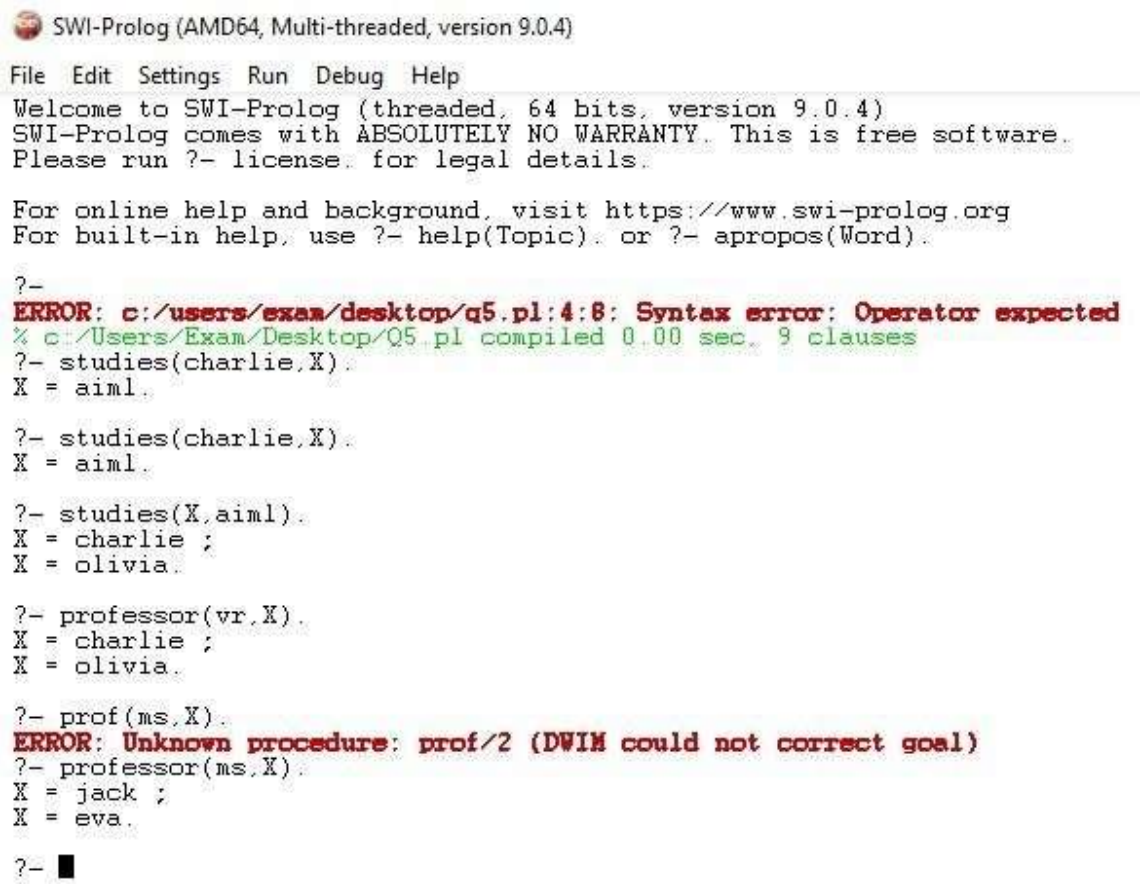
SWI-protog (AMD64, MuKi-threaded, version *9.0A)*

Fife Edit Settings Run Debug Help

```
Welcome to SWI-Prolog (threaded. 64 bits. version 9.0.4)
SWI-Prolog comes with ABSOlUTElY NO WARRANTY  This is lree soltware.
Please run 7- license. 1or legal details.

For online help and background. visit https://www.swi-prolog.org
For built-in help. use 7- help(Topic). or 7- apropos(Uord)


X c:/Users/Exam/Desktop/Q1 pl compiled 0 00 sec. 4 clauses
7- sum(10.20.Z)
Z = 30

7- dill(20.5.Z)
Z = 15

7- multi(2.3.Z)
     multi(2.3.Z)
ERROR: Syntax error: Operator expected
ERROR: multi(2,3,Z)
RBBO8.     boro
RIDDJR:
nulti(2,3,Z)
2- multi(2.3.Z)
Correct to: ”mult(2.3.Z)”2 yes
Z = 6

7- devide(90.5.Z)
Correct to: "divide(90.5.Z)"7 yes
Z = 18
```

2. Write a Prolog program to implement max(X, Y, M) so that M is the maximum of two numbers X and Y.

Commands



3. Write a Prolog program to implement power (Num,Pow, Ans) : where Num is raised to the power Pow to get Ans.

power(Num,Pow,Ans):-Ans is Num^Pow.

4. Given the following facts and rules:
    a. Facts:
        i. food(burger). ii. food(sandwich). iii. food(pizza). iv. lunch(sandwich).
        v. dinner(pizza).
    b. Rule: Every food is a meal
        i. meal(X) :- food(X).

    Write the following queries:

    1. Is pizza a food? Output:
        ?- food(pizza).

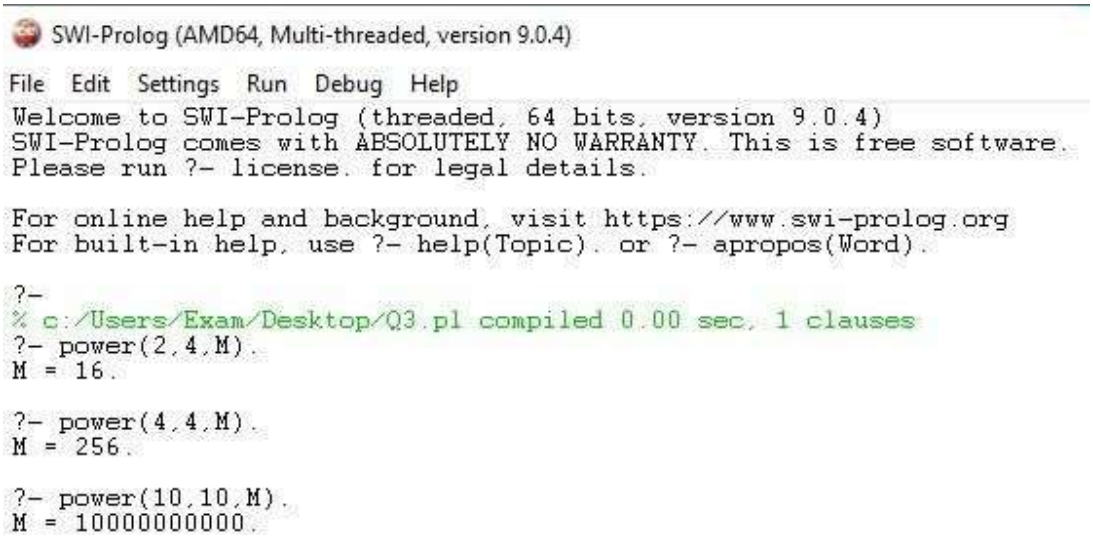
    2. Which food is both meal and food? output
        ?- meal(X),food(X).

    3. Which food is both meal and lunch?
        Output:
        ?- meal(X),lunch(X). X
        = sandwich ;

    4. Is sandwich a dinner? Output:

dinner(sandwich).

False.



5. Given the following facts and rules:
    a. Facts:
        i. studies(charlie, aiml). ii. studies(olivia, aiml). iii. studies(jack, dmba). iv. studies (john,dmba). v. studies(eva,dmba). vi. studies(arthur, mfcs). vii. teaches(vr, aiml).
        viii. teaches(ms, dmba).
        ix. teaches(snd, mfcs).
    b. Rule: X is a professor of Y if X teaches C and Y studies C.
        i. professor(X, Y) :- teaches(X, C), studies(Y, C)

Write the following queries:

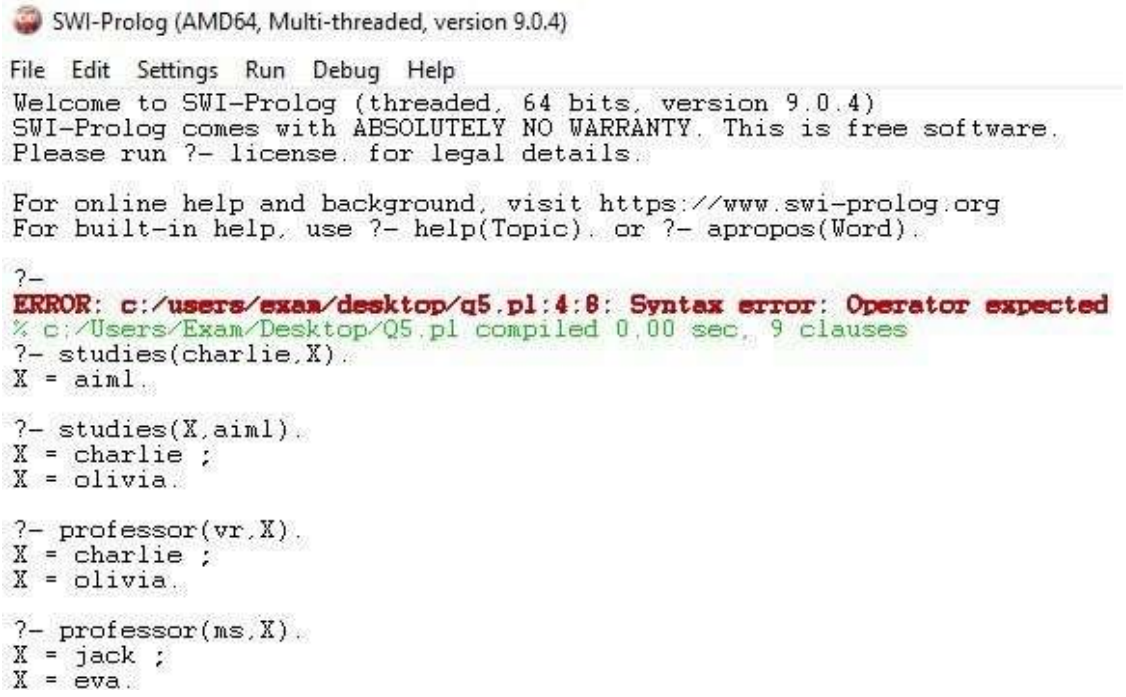
1. What does charlie study? Output:

?- studies(charlie,X). X
= aiml.

2. Which student has taken aiml?
?- studies(X,aiml).
X = charlie ; X =
olivia.

3. Who are the students of professorvr?
Output:
?- prof(vr,X).
X = charlie ; X
= olivia.

4. Who are the students of professorms?
Output:
?- prof(ms,X). X
= jack ;
X = john ; X
= eva.

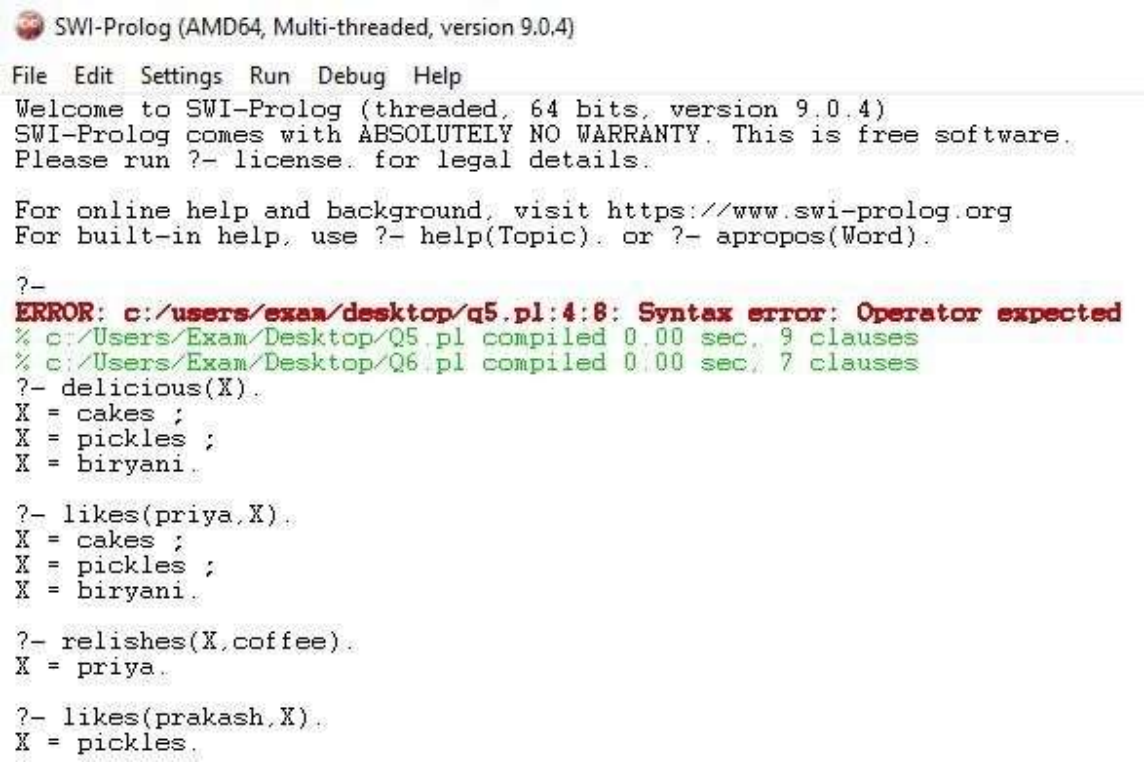


6 Given the following facts and rules:

delicious(cakes).
delicious(pickles).
delicious(biryani). spicy(pickles).
relishes(priya,coffee).
likes(priya,Y):- delicious(Y).
likes(prakash,Y):- spicy(Y).

Answer the following queries:

1. Which food items are delicious?
   ?- delicious(X).
   X = cakes ;
   X = pickles ; X
   = biryani.
2. Which food Priya likes? Output:
   ?- likes(priya,X). X
   = cakes ;
   X = pickles ; X
   = biryani.
3. Who relishes coffee? Output:
   ?- relishes(X,coffee).
   X = priya.
4. What does Prakash like?
   Output:
   ?- likes(prakash,X).
   X = pickles.



Conclusion :

Successfully implemented Logic Programming using PROLOG