Lab Report

Neural Networks and Machine Learning (IT 903)

Department of Information Technology

Gauhati University Institute of Science & Technology

Gauhati University, GNB Nagar,Ghy-14

SUBMITTED BY:

NAME: PARTHIV SARMA

ROLL NO: 190103016

BRANCH: COMPUTER SCIENCE AND ENGINEERING (CSE)

SEMESTER: 6TH

EXPERIMENT NO. 1: Python program to print "Hello Python" with user input to exit.

AIM: To print 'Hello Python'.

THEORY: In this program, we have used the built-in print() function to print the string Hello Python on our screen. A string is a sequence of characters. In Python, strings are enclosed inside single quotes, double quotes, or triple quotes.

- The following program displays a string "Hello Python" in the output by using the print() function.
- The `input()` function allows user input.
- The python sys module provides functions and variables which are used to manipulate different parts of the Python Runtime Environment.

Program Code:

```
[ ]  # Q1. Write a program to print "Hello Python"
```

Double-click (or enter) to edit

```
print ("Hello Python")

import sys;
#This module provides access to some variables used or maintained by the
#interpreter and to functions that interact strongly with the interpreter.

x='hey';
sys.stdout.write(x + '\n');
```

```
Hello Python
hey
```

EXPERIMENT NO. 2: Python program to print constant and variables.

AIM: To write a Python program to show the assignments of constants and variables and print them.

THEORY:

A *Python variable* is a reserved memory location to store values. . Values assigned to a variable can be changed later in the program whereas a constant is a type of variable whose value cannot be changed.

EXPLANATION:

The following program shows the difference between constants and variables by assigning different values to them and printing their values.

- In the code below, 'counter', 'miles' and 'name' are constants as their values are assigned only once and are not changed later.
- On the other hand, 'a', 'b', and 'c' are variables as the values assigned to them are changed later in the program.
- Next, the variable 'a' is deleted using 'del' keyword.

CODE:

```
[ ]

    # Constant

    counter =10
    miles = 1000.0
    name = 'Parthiv'
    print(counter)
    print(miles)
    print(name)

    #variables
    a=b=c=1
    a,b,c=1,'Parthiv','Sarma'
    print(a)
    print(b)
    print(c)

    #delete variable
    del a
```

OUTPUT:

```
10
1000.0
Parthiv
1
Parthiv
Sarma
```

EXPERIMENT NO. 3: Write a python program to print strings, updating strings, string formatting operator and triple quote.

AIM:Python program to create, format, modify strings.

THEORY:

The following program assigns values to strings and prints their values.

- A string 'str' is created by enclosing characters inside double-quotes.
- The values present at various indices of a string are printed using print(str[index]).
- A range of characters of the string is returned by using the slice syntax. This is done by specifying the start index and the end index, separated by a colon, to return a part of the string, e.g: print(str[2:9]).
- Similarly, print(str[:4]) prints the string from the beginning till the 4^{th} index of the string. And print(str[4:]) prints the string from 4^{th} index till the end of the string.
- The print(str * 2) function prints the same string twice.
- The print(str + " Sarma") statement is used to concatenate two different strings.

The program also shows how to update a string, and the use of String formatting operator and Triple Quotes in strings.

- '%' is used for string format. '%d' it will format a number for display, & '%s' will format a string for display.
- Triple quotes are used to allow strings to span multiple lines, including newlines, tabs, and any other special characters. The syntax for triple quotes consists of three consecutive single or double quotes.

CODE:

```python
#Q3)

#string
str = "Parthiv Sarma"
print(str)
print(str[0])
print(str[2])
print(str[4:])
print(str[2:9])
print(str * 2)
print(str + " Sarma")

# updating string
var1 = "Hello Parthiv"
print(var1)
print("Updating string :-", var1[:6] + "Sarma")


# string formatting operator
print("I am %s and age is %d" % ("Parthiv",21))

# Triple quote


para_string = """ This is an example of triple qoutes, which
helps to write multiple lines, tab \t and
also newlines exclusively given or wrote like this \n is also dispalyed"""
print(para_string)
```

Output:

```
Parthiv Sarma
P
r
hiv Sarma
rthiv S
Parthiv SarmaParthiv Sarma
Parthiv Sarma Sarma
Hello Parthiv
Updating string :- Hello Sarma
I am Parthiv and age is 21
 This is an example of triple qoutes, which
helps to write multiple lines, tab       and
also newlines exclusively given or wrote like this
 is also dispalyed
```

EXPERIMENT NO. 4: Python program to create & update list & tuples.

AIM:Python program to create & update list & tuples.

THEORY:

Lists in Python are those data types that allow us to work with multiple elements at once. A tuple in Python is similar to a list. The difference between the two is that we cannot change the elements of a tuple once it is assigned whereas we can change the elements of a list.

EXPLANATON:

Two lists, namely 'blist' and 'tlist' are created by placing elements inside square brackets [], separated by commas.

- The lists are printed using the print() function, e.g: print(blist).
- We use the index operator [] to access an item in a list, e.g. print(blist[0]).
- We access a range of items in a list by using the slicing operator :, e.g print(blist[2:]).
- We use the assignment operator '=' to change an item or a range of items, e.g list[2] = 2003.
- The '+' operator is used to concatenate two lists, e.g: print(blist + tlist).
- The 'del' statement is used to delete one or more items from a list, or even the entire list, e.g. del list[1].

The program also shows how to create a tuple, displaying tuple items and adding two tuples.

- Two tuples namely 'btupl' and 'ttupl' are created by placing all the items (elements) inside parentheses (), separated by commas.
- The tuples are printed using the print() function, e.g: print(ttupl).

- We use the index operator [] to access an item in a tuple, e.g. print(tttupl[1]).
- We access a range of items in a tuple by using the slicing operator :, e.g print(btupl[1:]).
- The '+' operator is used to concatenate two tuples, e.g: print(btupl + ttupl).
- Unlike lists, tuples are immutable. This means that elements of a tuple cannot be changed once they have been assigned. we cannot delete or remove items from a tuple. Deleting a tuple entirely, however, is possible using the keyword 'del'.

CODE:

```
  #Q4)
  #list
blist = [ "XYZ", "ABC", "PQR", '23', "33", "43" ]
tlist = [ "a", "b" ]

print(blist)
print(blist[0])
print(blist[2:])
print(tlist[1])
print(tlist)
print(blist + tlist)

# updating list


list = ["mango", "apple", 500, 600]
print(list)
print("Value available at index 2:")
print(list[2])
list[2] = 700
print("New value available at index 2:")
print(list[2])
del list[1]
print("after deleting value at index 1 the list looks like this:")
print(list)
```

```python
#tuple
btupl = ("parthiv", "sarma", 22, 28)
ttupl = (22, 28)

print(btupl)
print(ttupl)
print(btupl[1:])
print(ttupl[1])
print(btupl + ttupl)

# deleting elements of tuple is not permitted
```

Output:

```
['XYZ', 'ABC', 'PQR', '23', '33', '43']
XYZ
['PQR', '23', '33', '43']
b
['a', 'b']
['XYZ', 'ABC', 'PQR', '23', '33', '43', 'a', 'b']
['mango', 'apple', 500, 600]
Value available at index 2:
500
New value available at index 2:
700
after deleting value at index 1 the list looks like this:
['mango', 700, 600]
('parthiv', 'sarma', 22, 28)
(22, 28)
('sarma', 22, 28)
28
('parthiv', 'sarma', 22, 28, 22, 28)
```

EXPERIMENT NO. 5: Python program to create & update dictionaries.

AIM: : To write a Python program to print a dictionary and update a dictionary.

THEORY:Python dictionary is an unordered collection of items. Each item of a dictionary has a key/value pair. Dictionaries are optimized to retrieve values when the key is known.

EXPLANATION:

- A dictionary named ‘dict’ is created by placing curly braces {} and assigning two keys ‘one’ and ‘2’ and their respective values
- print(dict) is used to print the entire dictionary.
- A dictionary uses keys to access items, e.g.: print(dict[‘2’])
- All the keys and the values of the dictionary are printed separately using print(dict.keys) and print(dict.values) functions.
- A dictionary named ‘tinydict’ is created by placing items inside curly braces {} separated by a comma

tinydict['field']= 'Speech' creates a new key:value pair {‘field’: ‘Speech’} in the dictionary ‘tinydict’

CODE:

```
#q5)



dict = {}
dict['one'] = "This is value for one"
dict['2'] = "This is value for 2"
tinydict = {'name':'Parthiv', 'course':'B.TECH', 'rollno':'3016', 'Branch':'CSE'}

print(dict)
print(dict["one"])
print(dict['2'])
print(dict.values())
print(dict.keys())
print(tinydict)
print(tinydict.keys())
print(tinydict.values())


# updating dictionary

tinydict['field']= 'Grass'
tinydict['second_field']='Water'

print(tinydict)
```

Output:

```
{'one': 'This is value for one', '2': 'This is value for 2'}
This is value for one
This is value for 2
dict_values(['This is value for one', 'This is value for 2'])
dict_keys(['one', '2'])
{'name': 'Parthiv', 'course': 'B.TECH', 'rollno': '3016', 'Branch': 'CSE'}
dict_keys(['name', 'course', 'rollno', 'Branch'])
dict_values(['Parthiv', 'B.TECH', '3016', 'CSE'])
{'name': 'Parthiv', 'course': 'B.TECH', 'rollno': '3016', 'Branch': 'CSE', 'field': 'Grass', 'second_field': 'Water'}
```

EXPERIMENT NO. 6: Python program to implement while loop and for loop

AIM: Program to implement loops (while loop and for loop).

THEORY A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string). With the while loop we can execute a set of statements as long as a condition is true.

PROCEDURE:

- The while loop in Python is used to iterate over a block of code as long as the test expression (condition) is true.
- The statement 'while condition < 6: print(condition)' is used to print the value of 'condition' as long as it is less than 6.
- The for loop in Python is used to iterate over a sequence (list, tuple, string) or other iterable objects.
- The statement 'for x in egList' iterates over all the items in the list 'exampleList', and the 'print(x)' function is used to print each item.

Similarly, the statement 'for x in range(1,11)' is used to generate numbers from 1 to 10

CODE:

```
#Q6)


# While loop
m = 1

while m < 6:
  print(m)
  m += 1

print("---------")
# For loop
egList = [2,3,11,46,70]

for x in egList:
    print(x)

for x in range(1,11):
    print(x)
```

Output:

```
1
2
3
4
5
---------
2
3
11
46
70
1
2
3
4
5
6
7
8
9
10
```

EXPERIMENT NO. 7: Python program to demonstrate if-else statements.

AIM: Python program to demonstrate if-else statements.

THEORY: The if-else statement evaluates test expression and will execute the body of if only when the test condition is True.

If the condition is False, the body of else is executed. Indentation is used to separate the blocks.

CODE:

```python
# q7)

# If-else statement
a = 50
b = 100

if a > b:
    print('a is greater than y')
else:
    print('b is greater than x')


#  "elif" statement

p = 11
q = 22
r = 33

if p > q:
    print('p is greater than q')
elif p < r:
    print('p is less than r')
else:
    print('if and elif never run')
```

Output:

```
b is greater than x
p is less than r
```

EXPERIMENT NO. 8: Python program for implementing functions

AIM: To write a Python program for implementing functions.

THEORY: In Python, a function is a group of related statements that performs a specific task. Functions help break our program into smaller and modular chunks. As our program grows larger and larger, functions make it more organized and manageable. Furthermore, it avoids repetition and makes the code reusable.

Explanation:

- A function 'example' is created which prints a predefined value.
- The function is later called. It always produces the same output.
- A function 'simple_addition(num1,num2)' is created which takes two parameters 'num1' and 'num2', & prints their sum as output.
- This function produces different outputs for different values of parameters passed.

CODE:

```python
# q8)

def example():
    print('this is an example of how to write functions')
    c= 10 + 20
    print(c)

def simple_addition(n1,n2):
    ans = n1 + n2
    print('n1 is:', n1)
    print(ans)

# simple function
example()

# function with parameters
simple_addition(11,9)

# If you have a lot of parameters where it might be difficult to remember their order
# you could do something like:
#simple_addition(n2=4,n1=5)
```

Output:

```
this is an example of how to write functions
30
n1 is: 11
20
n1 is: 5
9
```

EXPERIMENT NO. 9: Python program to insert or write data into file.

AIM:Python program to insert or write data into file.

THEORY: Python provides inbuilt functions for creating, writing files. To write to an existing file, we must add a parameter to the open() function:

"a" - Append - will append to the end of the file

"w" - Write - will overwrite any existing content

EXPLANATION:

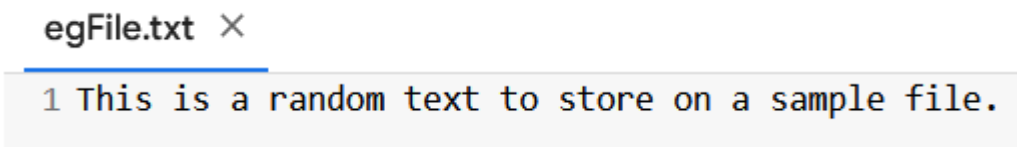
The following program shows the process of accessing an external text file and then writing some content in it using various python functions.

- Initially, a string named 'text' is created.
- An external file is opened using 'open('exampleFile.txt','w')' and this is saved in a variable 'saveFile'. The 'w' is used to get access for writing in the file.
- Then we write some text in it using saveFile.write(text).
- saveFile.close() is used to close the file.

CODE:

```python
# Q9)

text = 'This is a random text to store on a sample file.'

# notifies Python that you are opening this file, with the intention to write
saveFile = open('egFile.txt','w')

# actually writes the information
saveFile.write(text)

# It is important to remember to actually close the file, otherwise it will
# hang for a while and could cause problems in your script
saveFile.close()
```

Output:



EXPERIMENT NO. 10: Python program to read data from file.

AIM: To write a Python program to read content from an external file.

THEORY:

In Python, to open an external file, we use the built-in open() function.
The open() function returns a file object, which has a read() method for reading the content of the file.

EXPLANATION:

The following program shows the process of accessing an external text file and then reading it's inner text, and displaying it as output, by using various python functions. The program also displays the file's content as a python list.

- An external file is opened using 'open('egFile.txt','r').read' and is saved in a variable 'readMe'. The 'r' is used to get access for only reading the file.
- The print(readMe) function is used to print the entire content of the file.
- The statement 'open('egFile.txt','r').readlines()' will read the file in a Python list.

```
#Q10)

# similar syntax as you've seen, 'r' for read. You can just throw a .read() at
# the end, and you get:
readMe = open('egFile.txt','r').read()
print(readMe)


# this will instead read the file into a python list.
readMe = open('egFile.txt','r').readlines()
print(readMe)
```

Output:

```
This is a random text to store on a sample file.
['This is a random text to store on a sample file.']
```

EXPERIMENT NO. 11: Python program for implementation of classes and objects.

AIM: To create object class.

THEORY:

An object is simply a collection of data (variables) and methods (functions) that act on those data. Similarly, a class is a blueprint for that object. An object is also called an instance of a class and the process of creating this object is called instantiation. Classes are created by keyword class. Attributes are the variables that belong to a class. Attributes are always public and can be accessed using the dot (.) operator.

EXPLANATION:

- The following program creates a class *Student* as the base class.
- The __init__(self, name, scholarship) function gets called whenever a new object of the *Student* class is instantiated.
- The statement 'Student.stdcount += 1' increases the number of employees everytime an object is created.
- The function displayCount(self) prints the total number of employees.
- The function displayStudent(self) prints the name and salary of a particular *Employee* object.

Two objects std1 and std2 of *Student* class are created and the functions are called.

CODE:

```python
#Q11)

class Student:
   'Common base class for all employees'
   stdcount = 0

   def __init__(self, name, scholarship):
      self.name = name
      self.scholarship = scholarship
      Student.stdcount += 1

   def displayCount(self):
     print("Total Students %d" % Student.stdcount)

   def displayStudent(self):
      print("Name : ", self.name,  ", scholarship earned: ", self.scholarship)

"This would create first object of Student class"
std1 = Student("Bhhupen", 80000)
"This would create second object of Student class"
std2 = Student("Kuarnjo", 50000)
std1.displayStudent()
std2.displayStudent()
print("Total students %d" % Student.stdcount)
```

Output:

```
Name :  Bhhupen , scholarship earned:  80000
Name :  Kuarnjo , scholarship earned:  50000
Total students 2
```

EXPERIMENT NO. 12: Python program to plot a graph using Matplotlib.

AIM To write a Python program for plotting graphs using matplotlib.

THEORY:

Pyplot is an API (Application Programming Interface) for Python's matplotlib that effectively makes matplotlib a viable open source alternative to MATLAB. Matplotlib is a library for data visualization, typically in the form of plots, graphs and charts.

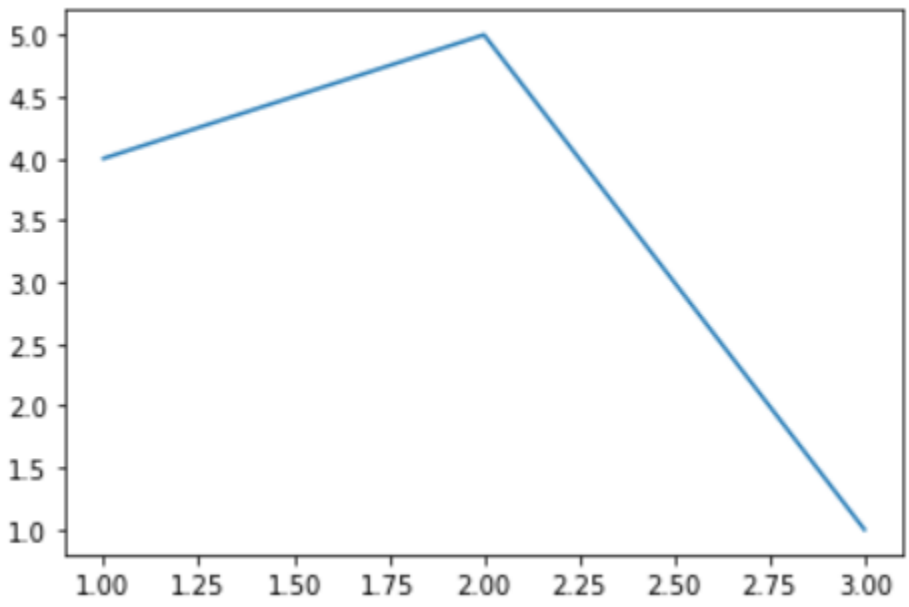
EXPLANATION:

- Initially, *pyplot()* is imported as *plt.*
- The pyplot *plot()* command is directed to the *matplotlib.axes.Axes.plot* function in the backend layer. It provides a unified interface for different types of plots.
- The pyplot *show()* command is directed to the *matplotlib.figure.Figure.show()* function in the backend layer.

CODE:

```
[ ]  #q12)

     import matplotlib

     #Importing pyplot
     from matplotlib import pyplot as plt

     #Plotting to our canvas
     plt.plot([1,2,3],[4,5,1])

     #Showing what we plotted
     plt.show()
```

Output:



EXPERIMENT NO. 13: Python program to classify data using Support Vector Classification(SVC).

AIM OF THE EXPERIMENT: To classify data using Support Vector Classification(SVC).

THEORY:

Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane

CODE:

```
#13)


from sklearn import svm


X = [[0, 0], [1, 1]]
y = [0, 1]

print(X)
print(y)

# SVC is for support vector classification .
clf = svm.SVC(kernel='rbf', max_iter=100)


# fit the model
clf.fit(X, y)

#After being fitted, the model can then be used to predict new values:
p=clf.predict([[2., 2.]])
print(p)
```

Output:

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

EXPERIMENT NO. 14: To demonstrate Scalar Vector Machine regression in Python

AIM: The aim of the program is to demonstrate Scalar Vector Machine regression in Python.

THEORY: Support Vector Regression is **a supervised learning algorithm that is used to predict discrete values**. Support Vector Regression uses the same principle as the SVMs. The basic idea behind SVR is to find the best fit line. In SVR, the best fit line is the hyperplane that has the maximum number of points.

CODE:

```
#14)
import numpy as np
from sklearn.svm import SVR
import matplotlib.pyplot as plt
# ##############################################################################
# Generate sample data
X = np.sort(5 * np.random.rand(40, 1), axis=0)
y = np.sin(X).ravel()
print(X)
print(y)
# ##############################################################################
# Add noise to targets
y[::5] += 3 * (0.5 - np.random.rand(8))
print(y)
# ##############################################################################
# Fit regression model
svr_rbf = SVR(kernel='rbf', C=1e3, gamma=0.1)
svr_lin = SVR(kernel='linear', C=1e3)
svr_poly = SVR(kernel='poly', C=1e3, degree=2)
y_rbf = svr_rbf.fit(X, y).predict(X)
y_lin = svr_lin.fit(X, y).predict(X)
y_poly = svr_poly.fit(X, y).predict(X)
# ##############################################################################
```

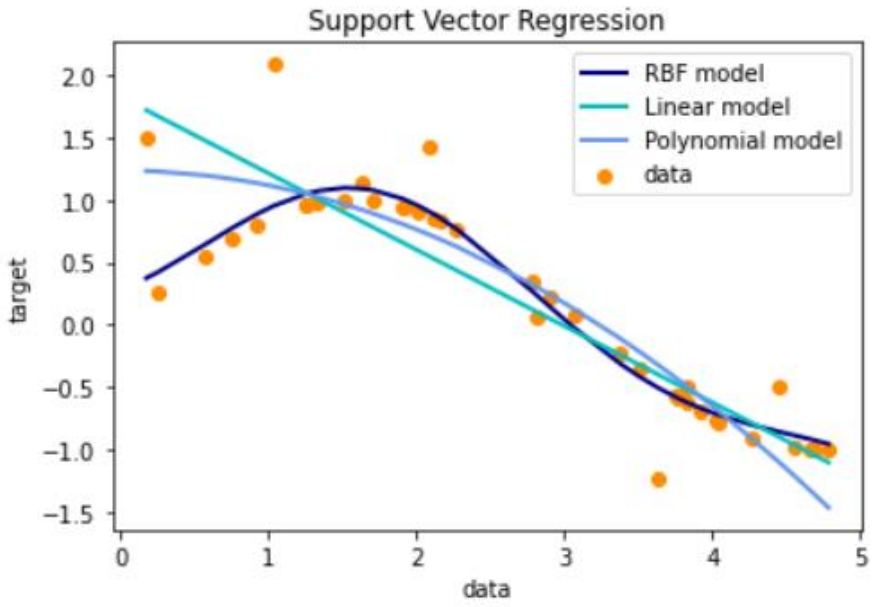
```
# Look at the results
lw = 2
plt.scatter(X, y, color='darkorange', label='data')
plt.plot(X, y_rbf, color='navy', lw=lw, label='RBF model')
plt.plot(X, y_lin, color='c', lw=lw, label='Linear model')
plt.plot(X, y_poly, color='cornflowerblue', lw=lw, label='Polynomial model')
plt.xlabel('data')
plt.ylabel('target')
plt.title('Support Vector Regression')
plt.legend()
plt.show()
```

Output:

```
[[0.17686457]
 [0.25495553]
 [0.57748374]
 [0.75130287]
 [0.92411108]
 [1.03543068]
 [1.257446  ]
 [1.25841592]
 [1.33645381]
 [1.51647498]
 [1.63966287]
 [1.71222563]
 [1.90280244]
 [1.92332788]
 [2.00760931]
 [2.08569176]
 [2.11800465]
 [2.16703862]
 [2.27374338]
 [2.77569564]
 [2.8195132 ]
 [2.90863778]
 [3.06777951]
 [3.37662845]
 [3.50710161]
 [3.63343576]
 [3.7516108 ]
 [3.76567148]
```

```
 [3.77361663]
 [3.82163541]
 [3.83082841]
 [3.91906871]
 [4.01661784]
 [4.03337224]
 [4.27203988]
 [4.45491131]
 [4.55683757]
 [4.66466611]
 [4.69451915]
 [4.78302606]]
[ 0.17594392  0.25220238  0.54591745  0.68259148  0.79808546  0.86008215
  0.95130618  0.95160471  0.97266722  0.99852496  0.99762964  0.99001554
  0.94539037  0.93850164  0.90610454  0.87034422  0.85398037  0.82745154
  0.76294033  0.35778707  0.31653977  0.23085358  0.07374613 -0.23287779
 -0.35742468 -0.47225132 -0.57288234 -0.58434999 -0.59077898 -0.62882627
 -0.63594757 -0.70148288 -0.76755965 -0.77819061 -0.90460291 -0.96703535
 -0.98792625 -0.99886148 -0.99984034 -0.99750624]
[ 1.49960757  0.25220238  0.54591745  0.68259148  0.79808546  2.0883026
  0.95130618  0.95160471  0.97266722  0.99852496  1.1426275   0.99001554
  0.94539037  0.93850164  0.90610454  1.42774098  0.85398037  0.82745154
  0.76294033  0.35778707  0.06523145  0.23085358  0.07374613 -0.23287779
 -0.35742468 -1.2342567  -0.57288234 -0.58434999 -0.59077898 -0.62882627
 -0.49921145 -0.70148288 -0.76755965 -0.77819061 -0.90460291 -0.49791707
 -0.98792625 -0.99886148 -0.99984034 -0.99750624]
```

Visual Representation:


Support Vector Regression
RBF model
Linear model
Polynomial model
data
target
data
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
0
1
2
3
4
5

EXPERIMENT NO. 15: Python program for plotting graphs for IRIS dataset using matplotlib.

AIM OF THE EXPERIMENT: To write a Python program for plotting graphs for IRIS dataset using matplotlib.

THEORY:

Matplotlib.pyplot library is most commonly used in Python in the field of machine learning. It helps in plotting the graph of a large dataset. Not only this also helps in classifying different dataset. It can plot graphs both in 2d and 3d format. It has a feature of legend, label, grid, graph shape, grid and many more that make it easier to understand and classify the dataset.

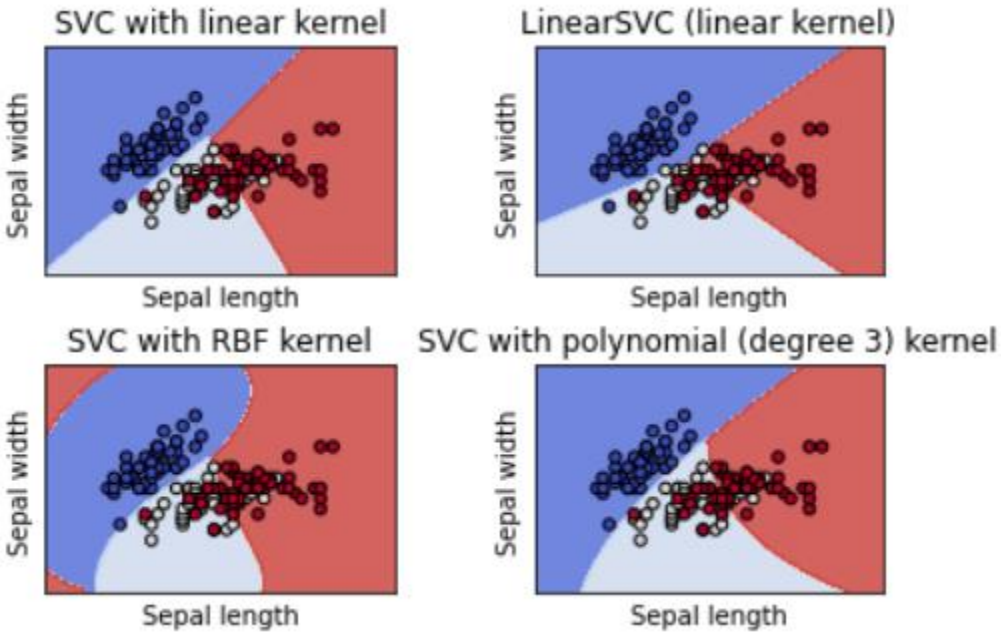
CODE:

```
#Q15
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm, datasets

def make_meshgrid(x, y, h=.02):
    """Create a mesh of points to plot in

    Parameters
    ----------
    x: data to base x-axis meshgrid on
    y: data to base y-axis meshgrid on
    h: stepsize for meshgrid, optional
    Returns
    -------
    xx, yy : ndarray
    """
    x_min, x_max = x.min() - 1, x.max() + 1
    y_min, y_max = y.min() - 1, y.max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))
    return xx, yy
```

```
def plot_contours(ax, clf, xx, yy, **params):
    """Plot the decision boundaries for a classifier.

    Parameters
    ----------
    ax: matplotlib axes object
    clf: a classifier
    xx: meshgrid ndarray
    yy: meshgrid ndarray
    params: dictionary of params to pass to contourf, optional
    """
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    out = ax.contourf(xx, yy, Z, **params)
    return out


# import some data to play with
iris = datasets.load_iris()

# Take the first two features. We could avoid this by using a two-dim dataset
X = iris.data[:, :2]
y = iris.target

# we create an instance of SVM and fit out data. We do not scale our
# data since we want to plot the support vectors
C = 1.0  # SVM regularization parameter
models = (svm.SVC(kernel='linear', C=C),
          svm.LinearSVC(C=C),
          svm.SVC(kernel='rbf', gamma=0.7, C=C),
          svm.SVC(kernel='poly', degree=3, C=C))
models = (clf.fit(X, y) for clf in models)
```

```
# title for the plots
titles = ('SVC with linear kernel',
          'LinearSVC (linear kernel)',
          'SVC with RBF kernel',
          'SVC with polynomial (degree 3) kernel')

# Set-up 2x2 grid for plotting.
fig, sub = plt.subplots(2, 2)
plt.subplots_adjust(wspace=0.4, hspace=0.4)

X0, X1 = X[:, 0], X[:, 1]
xx, yy = make_meshgrid(X0, X1)

for clf, title, ax in zip(models, titles, sub.flatten()):
    plot_contours(ax, clf, xx, yy,
                  cmap=plt.cm.coolwarm, alpha=0.8)
    ax.scatter(X0, X1, c=y, cmap=plt.cm.coolwarm, s=20, edgecolors='k')
    ax.set_xlim(xx.min(), xx.max())
    ax.set_ylim(yy.min(), yy.max())
    ax.set_xlabel('Sepal length')
    ax.set_ylabel('Sepal width')
    ax.set_xticks(())
    ax.set_yticks(())
    ax.set_title(title)

plt.show()
```

Output:


SVC with linear kernel
LinearSVC (linear kernel)
SVC with RBF kernel
SVC with polynomial (degree 3) kernel
Sepal width
Sepal length

TENSORBOARD:

EXPERIMENT NO. 1: To add two constant tensors and print their result and build the computation graph.

AIM OF THE EXPERIMENT: To use TensorFlow to add two constant tensors and print their result and build the computation graph.

THEORY:

TensorBoard provides the visualization and tooling needed for machine learning experimentation: Tracking and visualizing metrics such as loss and accuracy, Visualizing the model graph (ops and layers), Viewing histograms of weights, biases, Displaying images, text, and audio data, and much more. A graph contains tensors and operations.

EXPLANATION:

- To initiate a graph, a session is created which runs the graph. In other words, a graph provides a schema whereas a session processes a graph to compute values(tensors).
- tf.constant() creates a constant tensor from a tensor-like object.
- tf.Session() initiates a TensorFlow Graph object in which tensors are processed through operations (or ops).
- The tf.summary()module provides APIs for writing summary data. This data can be visualized in TensorBoard, the visualization toolkit that comes with TensorFlow.

CODE:

```
import numpy as np

import warnings
warnings.filterwarnings("ignore")

# import tensorflow as tf
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()

#*******************************************************
# visualzing graph with tensorboard


a = tf.constant(2)
b = tf.constant(3)
x = tf.add(a,b)

print(a)
print(b)
print(x)

sess = tf.Session()
writer = tf.summary.FileWriter('./graphs', sess.graph)
writer.add_graph(tf.get_default_graph())
print(sess.run(x))
```

Output:

```
Tensor("Const_2:0", shape=(), dtype=int32)
Tensor("Const_3:0", shape=(), dtype=int32)
Tensor("Add_1:0", shape=(), dtype=int32)
5
```

EXPERIMENT NO. 2: Visualizing graphs with TensorBoard

AIM OF THE EXPERIMENT: To visualize graphs with TensorBoard, with graphs nodes displaying names given by the user.

THEORY:

TensorBoard's Graphs dashboard is a powerful tool for examining TensorFlow models. We can view a conceptual graph of our model's structure and ensure it matches our intended design. We can also view an op-level graph to understand how TensorFlow understands our program. Examining the op-level graph can give us insight as to how to change our model. For example, we can redesign our model if training is progressing slower than expected.

EXPLANATION:

- tf.constant([2,2], name='a') and tf.constant([3,6], name='b') are used to create two tensors a and b from tensor-like objects.
- tf.add(a,b, name='x') returns the addition of a and b TensorFlow. Tensor objects element wise.
- tf.Session() initiates a TensorFlow Graph object in which tensors are processed through operations.
- The FileWriter class provides a mechanism to create an event file in a given directory and add summaries and events to it.

Code:

```
#2

import numpy as np

import warnings
warnings.filterwarnings("ignore")
# import tensorflow as tf
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()

#*****************************************************
# visualzing graph with tensorboard, with graphs node displaying name given by user


a = tf.constant([2,2], name='a')
b = tf.constant([3,6], name='b')
x = tf.add(a,b, name='x')


print(a)
print(b)
print(x)

sess = tf.compat.v1.Session()
writer = tf.summary.FileWriter('./graphs', sess.graph)
writer.add_graph(tf.get_default_graph())
print(sess.run(x))
```

Output:

```
Tensor("a:0", shape=(2,), dtype=int32)
Tensor("b:0", shape=(2,), dtype=int32)
Tensor("x:0", shape=(2,), dtype=int32)
[5 8]
```

EXPERIMENT NO. 3: Visualizing the computational graph.

AIM OF THE EXPERIMENT: To visualize the computational graph

THEORY:

TensorBoard's Graphs dashboard is a powerful tool for examining TensorFlow models. We can view a conceptual graph of our model's structure and ensure it matches our intended design. We can also view an op-level graph to understand how TensorFlow understands our program. Examining the op-level graph can give us insight as to how to change our model. For example, we can redesign our model if training is progressing slower than expected.

EXPLANATION:

- tf.constant() creates a constant tensor from a tensor-like object.
- tf.add() performs an element-wise addition over two Tensor objects.
- Similarly, tf.multiply() performs an element-wise multiplication over two Tensor objects.
- tf.summary()module provides APIs for writing summary data.

CODE:

```python
#Q3

import numpy as np
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()

#*******************************************************
# visualzing graph with tensorboard

a = tf.constant([2.0], name='a')
b = tf.constant([3.0], name='b')
x = tf.add(a,b, name='x')
c = tf.constant([4.0], name='c')
y = tf.multiply(x,c)

print(a)
print(b)
print(x)
print(c)
print(y)
sess = tf.Session()
writer = tf.summary.FileWriter('./graphs', sess.graph)
writer.add_graph(tf.get_default_graph())
print(sess.run(y))
```

Output:

```
Tensor("a_1:0", shape=(1,), dtype=float32)
Tensor("b_1:0", shape=(1,), dtype=float32)
Tensor("x_1:0", shape=(1,), dtype=float32)
Tensor("c:0", shape=(1,), dtype=float32)
Tensor("Mul:0", shape=(1,), dtype=float32)
[20.]
```

EXPERIMENT NO. 4: To multiply matrices in TensorFlow and visualize their computational graph.

AIM OF THE EXPERIMENT: To multiply two constant matrices in TensorFlow and visualize their computational graph using TensorBoard.

THEORY:

Tensorflow provides a range of functions to deal with data structures. Tensor multiplication is an important part of building machine learning models, as all data (images or sounds) is represented in the form of tensors.

EXPLANATION:

Two types of tensor multiplication are used:

- tf.multiply() : used to perform element-wise multiplication
- tf.matmul**()** : used to perform dot product (matrix multiplication) between two tensors

CODE:

```
#Q4

import numpy as np
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()

#*******************************************************
# visualzing graph with tensorboard

mat1 = tf.constant([[2., 3.]], name='mat1')

mat2 = tf.constant([[4.],[5.]], name='mat2')

product = tf.matmul(mat1, mat2, name='product')


print(mat1)
print(mat2)
print(product)

sess = tf.Session()
writer = tf.summary.FileWriter('./graphs', sess.graph)
writer.add_graph(tf.get_default_graph())
print(sess.run(product))
```

Output:

```
Tensor("mat1:0", shape=(1, 2), dtype=float32)
Tensor("mat2:0", shape=(2, 1), dtype=float32)
Tensor("product:0", shape=(1, 1), dtype=float32)
[[23.]]
```

TENSORFLOW:

EXPERIMENT NO. 1: Using constant variables in Tensorflow

AIM OF THE EXPERIMENT: To demonstrate the use of constant variables in TensorFlow.

THEORY:

TensorFlow provides a collection of workflows to develop and train models using Python or JavaScript, and to easily deploy in the cloud, on-prem, in the browser, or on-device no matter what language you use. Variables are created and tracked via the tf.Variable class. A tf.Variable represents a tensor whose value can be changed by running ops on it. Specific ops allow us to read and modify the values of this tensor. Higher level libraries like tf.keras use tf.Variable to store model parameters.

CODE:

```
#1

import numpy as np
import tensorflow as tf

a = tf.constant(3.0, dtype=tf.float32)
print(a)
cool_numbers  = tf.Variable([3.14159, 2.71828], tf.float32)
print(cool_numbers)
```

Output:

```
Tensor("Const:0", shape=(), dtype=float32)
<tf.Variable 'Variable:0' shape=(2,) dtype=float32>
```

EXPERIMENT NO. 2: Building a simple computational graph and perform Tensor computations.

AIM OF THE EXPERIMENT: To build a simple computational graph and perform Tensor computations.

THEORY:

The function tf.random_uniform() outputs random values from a uniform distribution. The generated values follow a uniform distribution in the range [minval, maxval]. The lower bound minval is included in the range, while the upper bound maxval is excluded

CODE:

```
#2)
import numpy as np
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()

#*******************************************************
#*******************************************************
# Let's build a simple computational graph
a = tf.constant(3.0, dtype=tf.float32)
b = tf.constant(4.0) # also tf.float32 implicitly
total = a + b
print(a)
print(b)
print(total)
#*******************************************************
# The following code creates a tf.Session object and then invokes its run method to evaluate the total tensor we created
sess = tf.Session()
print(sess.run(total))
#You can pass multiple tensors to tf.Session.run. The run method transparently handles any combination of tuples or dictionaries, as in the following example:

print(sess.run({'ab':(a, b), 'total':total}))

#During a call to tf.Session.run any tf.Tensor only has a single value.
#For example, the following code calls tf.random_uniform to produce a tf.Tensor that generates a random 3-element vector (with values in [0,1)):

vec = tf.random_uniform(shape=(3,))
out1 = vec + 1
out2 = vec + 2
print(sess.run(vec))
print(sess.run(vec))
print(sess.run((out1, out2)))
```

Output:

```
Tensor("Const_1:0", shape=(), dtype=float32)
Tensor("Const_2:0", shape=(), dtype=float32)
Tensor("add:0", shape=(), dtype=float32)
7.0
{'ab': (3.0, 4.0), 'total': 7.0}
[0.98725986 0.47965372 0.14174414]
[0.6700454  0.9542326  0.32715726]
(array([1.0009034, 1.459045 , 1.5015006], dtype=float32), array([2.0009034, 2.459045 , 2.5015006], dtype=float32))
```

EXPERIMENT NO. 3: To show the use of Fethcing in TensorFlow.

AIM: To show the use of Fethcing in TensorFlow.

THEORY:

We can fetch the tensor values we want to print with tf.Session.run(). The values are returned as a NumPy array and can be printed or logged with Python statements. This is the simplest and easiest approach, with the biggest drawback being that the computation graph executes all the dependent paths, starting from the fetched tensor, and if those paths include the training operations, then it advances one step or one epoch. Therefore, most of the time we would not call tf.Session.run() to fetch tensors in the middle of the graph, but we would execute the whole graph and fetch all the tensors, the ones we need to debug along with the ones we do not need to debug.

CODE:

```
#3)!/usr/bin/python

import numpy as np
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()

x = tf.constant([[37.0, -23.0], [1.0, 4.0]])
w = tf.Variable(tf.random_uniform([2, 2]))
y = tf.matmul(x, w)
output = tf.nn.softmax(y)
init_op = w.initializer
print(x)
print(w)
print(y)
print(output)
print(init_op)

with tf.Session() as sess:
  # Run the initializer on `w`.
  sess.run(init_op)
  # Evaluate `output`. `sess.run(output)` will return a NumPy array containing
  # the result of the computation.
  print(sess.run(output))
  # Evaluate `y` and `output`. Note that `y` will only be computed once, and its
  # result used both to return `y_val` and as an input to the `tf.nn.softmax()`
  # op. Both `y_val` and `output_val` will be NumPy arrays.
  y_val, output_val = sess.run([y, output])
```

Output:

```
[ ]  Tensor("Const_5:0", shape=(2, 2), dtype=float32)
     <tf.Variable 'Variable_1:0' shape=(2, 2) dtype=float32_ref>
     Tensor("MatMul:0", shape=(2, 2), dtype=float32)
     Tensor("Softmax:0", shape=(2, 2), dtype=float32)
     name: "Variable_1/Assign"
     op: "Assign"
     input: "Variable_1"
     input: "random_uniform/RandomUniform"
     attr {
       key: "T"
       value {
         type: DT_FLOAT
       }
     }
     attr {
       key: "_class"
       value {
         list {
           s: "loc:@Variable_1"
         }
       }
     }
     attr {
       key: "use_locking"
       value {
         b: true
       }
     }
     attr {
       key: "validate_shape"
       value {
         b: true
       }
     }


    [[1.0000000e+00 8.9083445e-12]
     [4.2143669e-02 9.5785636e-01]]
```

EXPERIMENT NO. 4: Use of feeding in TensorFlow.

AIM OF THE EXPERIMENT: To show the use of feeding in TensorFlow.

THEORY: A placeholder is simply a variable that we will assign data to at a later date. It allows us to create our operations and build our computation graph, without needing the data. tf.placeholder() is used to create a placeholder variable. feed_dict allows us to feed values to the TensorFlow placeholder.

CODE:

```
#4)

import numpy as np
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
#*****************************************************
# Feeding
x = tf.placeholder(tf.float32)
y = tf.placeholder(tf.float32)
z = x + y


sess = tf.Session()
print(sess.run(z, feed_dict={x: 3, y: 4.5}))
print(sess.run(z, feed_dict={x: [1, 3], y: [2, 4]}))
```

Output:

```
7.5
[3. 7.]
```

EXPERIMENT NO. 5: Use of Feeding in TensorFlow.

AIM OF THE EXPERIMENT: To demonstrate feeding in TensorFlow.

THEORY:

A placeholder is simply a variable that we will assign data to at a later date. It allows us to create our operations and build our computation graph, without needing the data. tf.placeholder() is used to create a placeholder variable. feed_dict allows us to feed values to the TensorFlow placeholder. tf.square() is used to compute element-wise squares.

CODE:

```
[ ]  #5)!/usr/bin/python

     import numpy as np
     import tensorflow.compat.v1 as tf
     tf.disable_v2_behavior()
     #****************************************************
     # Define a placeholder that expects a vector of three floating-point values,
     # and a computation that depends on it.
     x = tf.placeholder(tf.float32, shape=[3])
     y = tf.square(x)

     with tf.Session() as sess:
       # Feeding a value changes the result that is returned when you evaluate `y`.
       print(sess.run(y, {x: [1.0, 2.0, 3.0]}))  # => "[1.0, 4.0, 9.0]"
       print(sess.run(y, {x: [0.0, 0.0, 5.0]}))  # => "[0.0, 0.0, 25.0]"

       # Raises `tf.errors.InvalidArgumentError`, because you must feed a value for
       # a `tf.placeholder()` when evaluating a tensor that depends on it.
       # sess.run(y)

       # Raises `ValueError`, because the shape of `37.0` does not match the shape
       # of placeholder `x`.
       # sess.run(y, {x: 37.0})
```

Output:

```
[1. 4. 9.]
[ 0.  0. 25.]
```

EXPERIMENT NO. 6: Demonstration of Metadata in TensorFlow.

AIM OF THE EXPERIMENT: To demonstrate metadata in TensorFlow.

THEORY:

MetaData in TensorFlow is used to extract runtime statistics of the graph execution. It adds information about the time of execution and memory consumption to the event files and allows us to see this information in TensorBoard. MetaData also stores information like run times, memory consumption.

CODE:

```
#6)

import numpy as np
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
#*******************************************************

y = tf.matmul([[37.0, -23.0], [1.0, 4.0]], tf.random_uniform([2, 2]))

with tf.Session() as sess:
  # Define options for the `sess.run()` call.
  options = tf.RunOptions()
  options.output_partition_graphs = True
  options.trace_level = tf.RunOptions.FULL_TRACE

  # Define a container for the returned metadata.
  metadata = tf.RunMetadata()

  sess.run(y, options=options, run_metadata=metadata)

  # Print the subgraphs that executed on each device.
  print(metadata.partition_graphs)

  # Print the timings of each operation that executed.
  print(metadata.step_stats)
```

Output:

```
[node {
  name: "MatMul_1/a"
  op: "Const"
  device: "/job:localhost/replica:0/task:0/device:CPU:0"
  attr {
    key: "_XlaHasReferenceVars"
    value {
      b: false
    }
  }
  attr {
    key: "dtype"
    value {
      type: DT_FLOAT
    }
  }
```

```
  attr {
    key: "value"
    value {
      tensor {
        dtype: DT_FLOAT
        tensor_shape {
          dim {
            size: 2
          }
          dim {
            size: 2
          }
        }
        tensor_content:
"\000\000\024B\000\000\270\301\000\000\200?\000\000\200@"
      }
    }
  }
}
node {
  name: "random_uniform_1/shape"
  op: "Const"
  device: "/job:localhost/replica:0/task:0/device:CPU:0"
  attr {
    key: "_XlaHasReferenceVars"
    value {
      b: false
    }
  }
  attr {
    key: "dtype"
    value {
      type: DT_INT32
    }
  }
  attr {
    key: "value"
    value {
      tensor {
        dtype: DT_INT32
        tensor_shape {
          dim {
            size: 2
          }
        }
        int_val: 2
      }
    }
  }
}
node {
  name: "random_uniform_1/RandomUniform"
  op: "RandomUniform"
  input: "random_uniform_1/shape"
  device: "/job:localhost/replica:0/task:0/device:CPU:0"
  attr {
    key: "T"
```

```
      value {
        type: DT_INT32
      }
    }
    attr {
      key: "_XlaHasReferenceVars"
      value {
        b: false
      }
    }
    attr {
      key: "dtype"
      value {
        type: DT_FLOAT
      }
    }
    attr {
      key: "seed"
      value {
        i: 0
      }
    }
    attr {
      key: "seed2"
      value {
        i: 0
      }
    }
  }
  node {
    name: "MatMul_1"
    op: "MatMul"
    input: "MatMul_1/a"
    input: "random_uniform_1/RandomUniform"
    device: "/job:localhost/replica:0/task:0/device:CPU:0"
    attr {
      key: "T"
      value {
        type: DT_FLOAT
      }
    }
    attr {
      key: "_XlaHasReferenceVars"
      value {
        b: false
      }
    }
    attr {
      key: "transpose_a"
      value {
        b: false
      }
    }
    attr {
      key: "transpose_b"
      value {
        b: false
```

```
    }
  }
}
node {
  name: "_retval_MatMul_1_0_0"
  op: "_Retval"
  input: "MatMul_1"
  device: "/job:localhost/replica:0/task:0/device:CPU:0"
  attr {
    key: "T"
    value {
      type: DT_FLOAT
    }
  }
  attr {
    key: "_XlaHasReferenceVars"
    value {
      b: false
    }
  }
  attr {
    key: "index"
    value {
      i: 0
    }
  }
}
library {
}
versions {
  producer: 987
}
]
dev_stats {
  device: "/job:localhost/replica:0/task:0/device:CPU:0"
  node_stats {
    node_name: "_SOURCE"
    all_start_micros: 1653289222301502
    op_start_rel_micros: 1
    op_end_rel_micros: 1
    all_end_rel_micros: 3
    timeline_label: "_SOURCE = NoOp()"
    scheduled_micros: 11528999741310140
    all_start_nanos: 1653289222301502848
    op_start_rel_nanos: 255
    op_end_rel_nanos: 341
    all_end_rel_nanos: 2798
    scheduled_nanos: -6917744332399411440
  }
  node_stats {
    node_name: "MatMul_1/a"
    all_start_micros: 1653289222301534
    op_start_rel_micros: 4
    op_end_rel_micros: 14
    all_end_rel_micros: 36
    output {
      tensor_description {
```

```
        dtype: DT_FLOAT
        shape {
          dim {
            size: 2
          }
          dim {
            size: 2
          }
        }
        allocation_description {
          requested_bytes: 16
          allocator_name: "cpu"
          ptr: 128587968
        }
      }
    }
    timeline_label: "MatMul_1/a = Const()"
    scheduled_micros: 11528999741355049
    memory_stats {
      persistent_memory_size: 16
    }
    all_start_nanos: 1653289222301534774
    op_start_rel_nanos: 4083
    op_end_rel_nanos: 13762
    all_end_rel_nanos: 35355
    scheduled_nanos: -6917744332354502440
  }
  node_stats {
    node_name: "random_uniform_1/shape"
    all_start_micros: 1653289222301572
    op_start_rel_micros: 1
    op_end_rel_micros: 2
    all_end_rel_micros: 3
    output {
      tensor_description {
        dtype: DT_INT32
        shape {
          dim {
            size: 2
          }
        }
        allocation_description {
          requested_bytes: 8
          allocator_name: "cpu"
          ptr: 128584960
        }
      }
    }
    timeline_label: "random_uniform_1/shape = Const()"
    scheduled_micros: 11528999741419978
    memory_stats {
      persistent_memory_size: 8
    }
    all_start_nanos: 1653289222301572990
    op_start_rel_nanos: 322
    op_end_rel_nanos: 1177
    all_end_rel_nanos: 2477
```

```
      scheduled_nanos: -6917744332289573440
    }
    node_stats {
      node_name: "random_uniform_1/RandomUniform"
      all_start_micros: 1653289222301577
      op_start_rel_micros: 1
      op_end_rel_micros: 92
      all_end_rel_micros: 98
      memory {
        allocator_name: "cpu"
        total_bytes: 16
        peak_bytes: 16
        live_bytes: 16
        allocation_records {
          alloc_micros: 1653289222301652
          alloc_bytes: 16
        }
        allocation_records {
          alloc_micros: 1653289222301735
          alloc_bytes: -16
        }
      }
      output {
        tensor_description {
          dtype: DT_FLOAT
          shape {
            dim {
              size: 2
            }
            dim {
              size: 2
            }
          }
          allocation_description {
            requested_bytes: 16
            allocated_bytes: 16
            allocator_name: "cpu"
            allocation_id: 1
            has_single_reference: true
            ptr: 129360064
          }
        }
      }
      timeline_label: "random_uniform_1/RandomUniform =
RandomUniform(random_uniform_1/shape)"
      scheduled_micros: 11528999741425323
      memory_stats {
      }
      all_start_nanos: 1653289222301577896
      op_start_rel_nanos: 677
      op_end_rel_nanos: 91589
      all_end_rel_nanos: 97456
      scheduled_nanos: -6917744332284228440
    }
    node_stats {
      node_name: "MatMul_1"
      all_start_micros: 1653289222301681
```

```
op_end_rel_micros: 49
all_end_rel_micros: 55
memory {
  allocator_name: "cpu"
  total_bytes: 16
  peak_bytes: 16
  live_bytes: 16
  allocation_records {
    alloc_micros: 1653289222301692
    alloc_bytes: 16
  }
}
output {
  tensor_description {
    dtype: DT_FLOAT
    shape {
      dim {
        size: 2
      }
      dim {
        size: 2
```

Experiment: To create a ID3 Decision Tree using Information Gain for the following the given dataset.

AIM OF THE EXPERIMENT: To create a Decision Tree using Information Gain for the following the given dataset using Python.

Day	Outlook	Temp	Humidity	Wind	Tennis?
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

CODE:

```
importpandasaspd
importnumpyasnp
frompprintimportpprint

dataset = pd.read_csv('tennis.csv',
                      names=['Day', 'Outlook', 'Temp', 'Humidity', 'Wind',
'Tennis'])

dataset.drop('Day', axis=1, inplace=True)


defentropy(target_col):
```

```
    elements, counts = np.unique(target_col, return_counts=True)
    entropy = np.sum([(-counts[i]/np.sum(counts))*np.log2(counts[i] /
                                                          np.sum(counts))
foriinrange(len(elements))])
    returnentropy


defInfoGain(data, split_attribute_name, target_name="Tennis"):

    total_entropy = entropy(data[target_name])


    vals, counts = np.unique(data[split_attribute_name], return_counts=True)

    Weighted_Entropy = np.sum([(counts[i]/np.sum(counts))*entropy(data.where(
        data[split_attribute_name] == vals[i]).dropna()[target_name])
foriinrange(len(vals))])

    Information_Gain = total_entropy - Weighted_Entropy
    returnInformation_Gain


defID3(data, originaldata, features, target_attribute_name="Tennis",
parent_node_class=None):
     iflen(np.unique(data[target_attribute_name])) <= 1:
        returnnp.unique(data[target_attribute_name])[0]

    eliflen(data) == 0:

returnnp.unique(originaldata[target_attribute_name])[np.argmax(np.unique(original
data[target_attribute_name], return_counts=True)[1])]

    eliflen(features) == 0:
        returnparent_node_class


    else:
        parent_node_class = np.unique(data[target_attribute_name])[np.argmax(
            np.unique(data[target_attribute_name], return_counts=True)[1])]

        item_values = [InfoGain(data, feature, target_attribute_name)
                       forfeatureinfeatures]
        best_feature_index = np.argmax(item_values)
        best_feature = features[best_feature_index]
```

```
        tree = {best_feature: {}}

        features = [iforiinfeaturesifi != best_feature]


        forvalueinnp.unique(data[best_feature]):
            value = value
            sub_data = data.where(data[best_feature] == value).dropna()

            subtree = ID3(sub_data, dataset, features,
                          target_attribute_name, parent_node_class)

            tree[best_feature][value] = subtree

        return(tree)


defpredict(query, tree, default=1):

    forkeyinlist(query.keys()):
        ifkeyinlist(tree.keys()):

            try:
                result = tree[key][query[key]]
        except:
                returndefault

            result = tree[key][query[key]]

            ifisinstance(result, dict):
                returnpredict(query, result)
            else:
                returnresult


deftrain_test_split(dataset):
    training_data = dataset.iloc[:11].reset_index(drop=True)
    testing_data = dataset.iloc[11:].reset_index(drop=True)
    returntraining_data, testing_data


training_data = train_test_split(dataset)[0]
testing_data = train_test_split(dataset)[1]


deftest(data, tree):
```

```
    queries = data.iloc[:, :-1].to_dict(orient="records")

    predicted = pd.DataFrame(columns=["predicted"])

    foriinrange(len(data)):
        predicted.loc[i, "predicted"] = predict(queries[i], tree, 1.0)
    print('The prediction accuracy is: ',
          (np.sum(predicted["predicted"] == data["Tennis"])/len(data))*100, '%')


tree = ID3(training_data, training_data, training_data.columns[:-1])
pprint(tree)
test(testing_data, tree)
```

Output:

```
{'Outlook': {'Outlook': 'Tennis',
             'Overcast': 'Yes',
             'Rain': {'Wind': {'Strong': 'No', 'Weak': 'Yes'}},
             'Sunny': {'Temp': {'Cool': 'Yes', 'Hot': 'No', 'Mild': 'No'}}}}
The prediction accuracy is:  75.0 %
```

Experiment: To make gates using mccullochpitts neuron.

AIM:To show basic logic gates using mccullochpitts neuron.

Code

AND GATE:

Q. Implement AND Gate using mcculloch pitts neuron

```python
import numpy as np

def threshold_function(in_vector, threshold):
  if in_vector >= threshold:
    return 1
  else:
    return 0

input_pairs = np.array([[0,0],[0,1],[1,0],[1,1]])
print(f'Input Array:\n {input_pairs}')

weights = np.array([1,1])
print(f'Weights:\n {weights}')

bias = np.array([-1,-1,-1,-1])
matrix_multiply = input_pairs @ weights
total_input = matrix_multiply + bias
print(f'Total input:\n {total_input}')

T = 1
print(f'AND gate || Threshold: {T}')

for i in range(0,4):
  output = threshold_function(total_input[i], T)
  print(f'Output: {output}')
```

Output:

```
Input Array:
 [[0 0]
 [0 1]
 [1 0]
 [1 1]]
Weights:
 [1 1]
Total input:
 [-1  0  0  1]
AND gate || Threshold: 1
Output: 0
Output: 0
Output: 0
Output: 1
```

NOR GATE:

```
[ ]  import numpy as np

     def threshold_function(in_vector, threshold):
       if in_vector >= threshold:
         return 1
       else:
         return 0

     input_pairs = np.array([[0,0],[0,1],[1,0],[1,1]])
     print(f'Input Array:\n {input_pairs}')

     weights = np.array([-1,-1])
     print(f'Weights:\n {weights}')

     bias = np.array([1,1,1,1])
     matrix_multiply = input_pairs @ weights
     total_input = matrix_multiply + bias
     print(f'Total input:\n {total_input}')

     T = 1
     print(f'NOR gate || Threshold: {T}')

     for i in range(0,4):
       output = threshold_function(total_input[i], T)
       print(f'Output: {output}')
```

OUTPUT:

```
Input Array:
 [[0 0]
 [0 1]
 [1 0]
 [1 1]]
Weights:
 [-1 -1]
Total input:
 [ 1  0  0 -1]
NOR gate || Threshold: 1
Output: 1
Output: 0
Output: 0
Output: 0
```

- Q. Implement XOR Gate using mcculloch pitts neuron

```
[ ]  import numpy as np

     def threshold_function(in_vector, threshold):
       if in_vector >= threshold:
         return 1
       else:
         return 0

     def AND(in_vector):
       weights = np.array([1,1])
       bias = np.array([-1,-1,-1,-1])
       matrix_multiply = input_pairs @ weights
       total_input = matrix_multiply + bias

       return total_input

     def NOR(in_vector):
       weights = np.array([-1,-1])
       bias = np.array([1,1,1,1])
       matrix_multiply = input_pairs @ weights
       total_input = matrix_multiply + bias

       return total_input

     input_pairs = np.array([[0,0],[0,1],[1,0],[1,1]])
     print(f'Input Array:\n {input_pairs}')
```

```
and_input = AND(input_pairs)
and_output=[0,0,0,0]

T=1
for i in range(0,4):
  and_output[i] = threshold_function(and_input[i], T)

nor_input = NOR(input_pairs)
nor_output=[0,0,0,0]

for i in range(0,4):
  nor_output[i] = threshold_function(nor_input[i], T)

inputs = np.array([[and_output[0],nor_output[0]],[and_output[1],nor_output[1]], [and_output[2],nor_output[2]],[and_output[3],nor_output[3]]])
weights = np.array([1,1])
print(f'Weights:\n {weights}')

bias = np.array([-1,-1,-1,-1])
matrix_multiply = inputs @ weights
total_input = matrix_multiply + bias
print(f'Total input:\n {total_input}')

T = 0
print(f'XOR gate || Threshold: {T}')

for i in range(0,4):
  output = threshold_function(total_input[i], T)
  print(f'Output: {output}')
```

OUTPUT:

```
Input Array:
 [[0 0]
 [0 1]
 [1 0]
 [1 1]]
Weights:
 [1 1]
Total input:
 [ 0 -1 -1  0]
XOR gate || Threshold: 0
Output: 1
Output: 0
Output: 0
Output: 1
```