1. Write a Program to Implement Tic-Tac-Toe game using Python

def print\_board(board):

for row in board:

print("|".join(row))

print("."\*5)

def check\_winner(board,player):

#check rows

for row in board:

if all(cell==player for cell in row):

return True

#check columns

for col in range(3):

if all(board[row][col]==player for row in range(3)):

return True

#check diagonals

if all (board[i][i]==player for i in range(3) or all (board[i][2-i]==player for i in range(3))):

return True

return False

def is\_board\_full(board):

for row in board:

for cell in row:

if cell==" ":

return False

return True

def main():

board=[[" " for \_ in range(3)]for \_ in range(3)]

players=["X","O"]

turn=0

print("welcome to Tic-Tac-Toe")

print\_board(board)

while True:

player=players[turn%2]

print(f"player{player}'s turn")

row=int(input("enter row(0,1,or 2):"))

col=int(input("enter column(0,1,or 2):"))

if board[row][col]==" ":

board[row][col]=player

print\_board(board)

if check\_winner(board,player):

print(f"player{player}wins!")

break

elif is\_board\_full(board):

print("it's draw!")

break

turn+=1

else:

print("That cell is already ocucupied.Try again")

if \_\_name\_\_=="\_\_main\_\_":

main()

Output:-

welcome to Tic-Tac-Toe  
 | |  
.....  
 | |  
.....  
 | |  
.....  
playerX's turn  
enter row(0,1,or 2):0  
enter column(0,1,or 2):0  
X| |  
.....  
 | |  
.....  
 | |  
.....  
playerO's turn  
enter row(0,1,or 2):0  
enter column(0,1,or 2):1  
X|O|  
.....  
 | |  
.....  
 | |  
.....  
playerX's turn  
enter row(0,1,or 2):0  
enter column(0,1,or 2):2  
X|O|X  
.....  
 | |  
.....  
 | |  
.....  
playerO's turn  
enter row(0,1,or 2):1  
enter column(0,1,or 2):0  
X|O|X  
.....  
O| |  
.....  
 | |  
.....  
playerX's turn  
enter row(0,1,or 2):1  
enter column(0,1,or 2):1  
X|O|X  
.....  
O|X|  
.....  
 | |  
.....  
playerO's turn  
enter row(0,1,or 2):1  
enter column(0,1,or 2):2  
X|O|X  
.....  
O|X|O  
.....  
 | |  
.....  
playerX's turn  
enter row(0,1,or 2):2  
enter column(0,1,or 2):0  
X|O|X  
.....  
O|X|O  
.....  
X| |  
.....  
playerO's turn  
enter row(0,1,or 2):2  
enter column(0,1,or 2):1  
X|O|X  
.....  
O|X|O  
.....  
X|O|  
.....  
playerX's turn  
enter row(0,1,or 2):2  
enter column(0,1,or 2):2  
X|O|X  
.....  
O|X|O  
.....  
X|O|X  
.....  
PlayerXwins!

2. Write a Program to Implement Water-Jug problem using Python

x=0

y=0

m=4

n=3

print("initial state =(0,0)")

print("capacitioes =(4,3)")

print("goal state =(2,y)")

while(x!=2):

r=int(input("enter the rule:"))

if(r==1):

x=m

elif(r==2):

y=n

elif(r==3):

x=0

elif(r==4):

y=0

elif(r==5):

t=n-y

y=n

x-=t

elif(r==6):

t=m-x

x=m

y-=t

elif(r==7):

y+=x

x=0

elif(r==8):

x+=y

y=0

else:

print("invalid rule")

print(x,y)

Output :-

initial state =(0,0)  
capacitioes =(4,3)  
goal state =(2,y)  
enter the rule:2  
0 3  
enter the rule:8  
3 0  
enter the rule:2  
3 3  
enter the rule:6  
4 2  
enter the rule:3  
0 2  
enter the rule:8  
2 0

3. Write a Program to implement 8-Puzzle problem using Python.

import numpy as np

import pandas as pd

import os

def bfs(src,target):

queue=[]

queue.append(src)

exp=[]

while len(queue)>0:

source=queue.pop(0)

exp.append(source)

print(source)

if source==target:

print("success")

return

poss\_moves\_to\_do=[]

poss\_moves\_to\_do=possible\_moves(source,exp)

for move in poss\_moves\_to\_do:

if move not in exp and move not in queue:

queue.append(move)

def possible\_moves(state,visited\_states):

b=state.index(0)

d=[]

if b not in [0,1,2]:

d.append('u')

if b not in[6,7,8]:

d.append('d')

if b not in[0,3,6]:

d.append('l')

if b not in[2,5,8]:

d.append('r')

poss\_moves\_it\_can=[]

for i in d:

poss\_moves\_it\_can.append(gen(state,i,b))

return[move\_it\_can for move\_it\_can in poss\_moves\_it\_can if move\_it\_can not in visited\_states]

def gen(state,m,b):

temp=state.copy()

if m=='d':

temp[b+3],temp[b]=temp[b],temp[b+3]

if m=='u':

temp[b-3],temp[b]=temp[b],temp[b-3]

if m=='l':

temp[b-1],temp[b]=temp[b],temp[b-1]

if m=='r':

temp[b+1],temp[b]=temp[b],temp[b+1]

return temp

src=[1,2,3,4,5,6,0,7,8]

target=[1,2,3,4,5,6,7,8,0]

bfs(src,target)

Output :-

[1, 2, 3, 4, 5, 6, 0, 7, 8]  
[1, 2, 3, 0, 5, 6, 4, 7, 8]  
[1, 2, 3, 4, 5, 6, 7, 0, 8]  
[0, 2, 3, 1, 5, 6, 4, 7, 8]  
[1, 2, 3, 5, 0, 6, 4, 7, 8]  
[1, 2, 3, 4, 0, 6, 7, 5, 8]  
[1, 2, 3, 4, 5, 6, 7, 8, 0]  
success

4. Write a Program to Implement AO\* Algorithm using Python.

M='D'

I='C'

H='B'

A='A'

L='AND'

F='OR'

E=len

D=print

def J(n):

R=False

I=True

global X

D('Expanding Node:',n)

G=[]

H=[]

if n in B:

if L in B[n]:

G=B[n][L]

if F in B[n]:

H=B[n][F]

if E(G)==0 and E(H)==0:

return

M=R

Q={}

while not M:

if E(Q)==E(G)+E(H):

S,T=O(G,H,{})

M=I

P(n,S)

C[n]=T

continue

V,A=O(G,H,Q)

N=R

if E(A)>1:

if A[0] in B:

N=I

J(A[0])

if A[1] in B:

N=I

J(A[1])

elif A in B:

N=I

J(A)

if N:

V,W=O(G,H,{})

if A==W:

M=I

P(n,V)

C[N]=A

else:

M=I

P(n,V)

C[n]=A

Q[A]=1

return K(n)

def O(and\_nodes,or\_nodes,marked):

G=marked

for B in and\_nodes:

if not B[0]+B[1] in G:

A=0

A=A+K(B[0])+K(B[1])+2

C[B[0]+B[1]]=A

for D in or\_nodes:

if not D in G:

A=0

A=A+K(D)+1

C[D]=A

e=999999

H=None

for F in C:

if C[F]<e:

E=C[F]

H=F

return [E,H]

def K(n):

return N[n]

def P(n,cost):

N[n]=cost

return

def G(node):

B='->'

A=node

D(C[A],end='')

A=C[A]

if E(str(A))>1:

if A[0] in C:

D(B,end='')

G(A[0])

if A[1] in C:

D(B,end='')

G(A[1])

elif A in C:

D(B,end='')

G(A)

N={A:-1,H:4,I:2,M:3,'E':6,'F':8,'G':2,'H':0,'I':0,'J':0}

B={A:{L:[(I,M)],F:[H]},

H:{F:['E','F']},

I:{F:['G'],L:[('H','I')]},

M:{F:['J']}}

C={}

Q=J(A)

D("Nodes which give optimal cost are")

G(A)

D("\n optimal cost is::",Q)

Output :-

Expanding Node: A  
Expanding Node: B  
Nodes which give optimal cost are  
F->9  
 optimal cost is:: 9

5. Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file.

import csv

num\_attribute=6

a=[]

with open('s.csv','r') as file:

reader=csv.reader(file)

a=list(reader)

hypothesis=a[1][:-1]

for i in a:

if i[-1]=='yes':

for j in range(num\_attribute):

if i[j]!=hypothesis[j]:

hypothesis[j]='?'

print(hypothesis)

print("\n the maximally specific hyothesis for a given training examples\n")

print(hypothesis)

Output :-

['sunny', 'warm', '?', 'strong', '?', '?']  
  
 the maximally specific hyothesis for a given training examples  
  
['sunny', 'warm', '?', 'strong', '?', '?']

6. For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.

import csv

with open("s.csv") as f:

csv\_file=csv.reader(f)

data=list(csv\_file)

s=data[1][:-1]

g=[['?'for i in range(len(s))]for j in range(len(s))]

for i in data:

if i[-1]=='yes':

for j in range(len(s)):

if i[j]!=s[j]:

s[j]='?'

g[j][j]='?'

elif i[-1]=='no':

for j in range(len(s)):

if i[j]!=s[j]:

g[j][j]=s[j]

else:

g[j][j]='?'

print("\n steps of candidate elimination algorithm",data.index(i)+1)

print(s)

print(g)

gh=[]

for i in g:

for j in i:

if j!='?':

gh.append(i)

break

print("\n final specific hypothesis:\n",s)

print("\n final general hypothesis:\n",gh)

Output :-

steps of candidate elimination algorithm 1  
['sunny', 'warm', 'normal', 'strong', 'warm', 'same']  
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]  
  
 steps of candidate elimination algorithm 2  
['sunny', 'warm', 'normal', 'strong', 'warm', 'same']  
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]  
  
 steps of candidate elimination algorithm 3  
['sunny', 'warm', '?', 'strong', 'warm', 'same']  
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]  
  
 steps of candidate elimination algorithm 4  
['sunny', 'warm', '?', 'strong', 'warm', 'same']  
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'same']]  
  
 steps of candidate elimination algorithm 5  
['sunny', 'warm', '?', 'strong', '?', '?']  
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]  
  
 final specific hypothesis:  
 ['sunny', 'warm', '?', 'strong', '?', '?']  
  
 final general hypothesis:  
 [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]

7. Write a program to demonstrate the working of the decision tree basedID3 algorithm.

import pandas as pd

from sklearn.tree import DecisionTreeClassifier

from sklearn.preprocessing import LabelEncoder

pd.options.mode.copy\_on\_write=True

data=pd.read\_csv('m.csv')

print('first 5 values of data are:\n',data.head())

x=data.iloc[:,:-1]

y=data.iloc[:,-1]

print("\n values of x:\n",x.head())

print('\n first 5 values of train output:\n',y.head())

le\_Outlook=LabelEncoder()

le\_Temperature=LabelEncoder()

le\_Humidity=LabelEncoder()

le\_Windy=LabelEncoder()

le\_PlayTennis=LabelEncoder()

x['Outlook']=le\_Outlook.fit\_transform(x['Outlook'])

x['Temperature']=le\_Temperature.fit\_transform(x['Temperature'])

x['Humidity']=le\_Humidity.fit\_transform(x['Humidity'])

x['Windy']=le\_Windy.fit\_transform(x['Windy'])

x.columns=['Outlook','Temperature','Humidity','Windy']

y=le\_PlayTennis.fit\_transform(y)

print('\nnow the rain data is:\n',x.head())

print('\n now the train output is:\n',y)

classifier=DecisionTreeClassifier()

classifier.fit(x,y)

inp=["Overcast","Cool","Normal","Strong"]

inp\_df=pd.DataFrame([inp],columns=['Outlook','Temperature','Humidity','Windy'])

inp\_df['Outlook']=le\_Outlook.transform(inp\_df['Outlook'])

inp\_df['Temperature']=le\_Temperature.transform(inp\_df['Temperature'])

inp\_df['Humidity']=le\_Humidity.transform(inp\_df['Humidity'])

inp\_df['Windy']=le\_Windy.transform(inp\_df['Windy'])

y\_pred=classifier.predict(inp\_df)

predicted\_label=le\_PlayTennis.inverse\_transform(y\_pred)[0]

print("\n for input {0} ,we obtain {1} ".format(inp,predicted\_label))

Output :-

first 5 values of data are:  
 Outlook Temperature Humidity Windy PlayTennis  
0 Sunny Hot High Weak No  
1 Sunny Hot High Strong No  
2 Overcast Hot High Weak Yes  
3 Rain Mild High Weak Yes  
4 Rain Cool Normal Weak Yes  
  
 values of x:  
 Outlook Temperature Humidity Windy  
0 Sunny Hot High Weak  
1 Sunny Hot High Strong  
2 Overcast Hot High Weak  
3 Rain Mild High Weak  
4 Rain Cool Normal Weak  
  
 first 5 values of train output:  
 0 No  
1 No  
2 Yes  
3 Yes  
4 Yes  
Name: PlayTennis, dtype: object  
  
now the rain data is:  
 Outlook Temperature Humidity Windy  
0 2 1 0 1  
1 2 1 0 0  
2 0 1 0 1  
3 1 2 0 1  
4 1 0 1 1  
  
 now the train output is:  
 [0 0 1 1 1 0 1 0 1 1 1 1 1 0]  
  
 for input ['Overcast', 'Cool', 'Normal', 'Strong'] ,we obtain Yes

8. Use an appropriate data set for building the decision tree and apply this knowledge to classify anew sample. Build an Artificial Neural Network by implementing the Back propagation algorithm and test the same using appropriate datasets.

import numpy as np

X=np.array(([2,9],[1,5],[3,6]),dtype=float)

y=np.array(([92],[86],[89]),dtype=float)

x=X/np.amax(X,axis=0)

y=y/100

def sigmoid(X):

return 1/(1+np.exp(-X))

def derivatives\_sigmoid(X):

return X\*(1-X)

epoch=5

lr=0.1

inputlayer\_neurons=2

hiddenlayer\_neurons=3

output\_neurons=1

wh=np.random.uniform(size=(inputlayer\_neurons,hiddenlayer\_neurons))

bh=np.random.uniform(size=(1,hiddenlayer\_neurons))

wout=np.random.uniform(size=(hiddenlayer\_neurons,output\_neurons))

bout=np.random.uniform(size=(1,output\_neurons))

for i in range(epoch):

hinp1=np.dot(X,wh)

hinp=hinp1+bh

hlayer\_act=sigmoid(hinp)

outlinep1=np.dot(hlayer\_act,wout)

outinp=outlinep1+bout

output=sigmoid(outinp)

EO=y-output

outgrad=derivatives\_sigmoid(output)

d\_output=EO\*outgrad

EH=d\_output.dot(wout.T)

hiddengrad=derivatives\_sigmoid(hlayer\_act)

d\_hiddenlayer=EH\*hiddengrad

wout+=hlayer\_act.T.dot(d\_output)\*lr

wh+=x.T.dot(d\_hiddenlayer)\*lr

print("\n------epoch",i+1,"starts-----\n")

print("input:\n"+str(X))

print("actual output:\n"+str(y))

print("predicted output:\n",output)

print("----epoch-",i+1,"ends-----\n")

Output :-

------epoch 1 starts-----  
  
input:  
[[2. 9.]  
 [1. 5.]  
 [3. 6.]]  
actual output:  
[[0.92]  
 [0.86]  
 [0.89]]  
predicted output:  
 [[0.92420224]  
 [0.91025558]  
 [0.92643799]]  
----epoch- 1 ends-----  
  
  
------epoch 2 starts-----  
  
input:  
[[2. 9.]  
 [1. 5.]  
 [3. 6.]]  
actual output:  
[[0.92]  
 [0.86]  
 [0.89]]  
predicted output:  
 [[0.92407415]  
 [0.910118 ]  
 [0.92631242]]  
----epoch- 2 ends-----  
  
  
------epoch 3 starts-----  
  
input:  
[[2. 9.]  
 [1. 5.]  
 [3. 6.]]  
actual output:  
[[0.92]  
 [0.86]  
 [0.89]]  
predicted output:  
 [[0.92394623]  
 [0.90998062]  
 [0.92618702]]  
----epoch- 3 ends-----  
  
  
------epoch 4 starts-----  
  
input:  
[[2. 9.]  
 [1. 5.]  
 [3. 6.]]  
actual output:  
[[0.92]  
 [0.86]  
 [0.89]]  
predicted output:  
 [[0.92381847]  
 [0.90984345]  
 [0.92606178]]  
----epoch- 4 ends-----  
  
  
------epoch 5 starts-----  
  
input:  
[[2. 9.]  
 [1. 5.]  
 [3. 6.]]  
actual output:  
[[0.92]  
 [0.86]  
 [0.89]]  
predicted output:  
 [[0.92369089]  
 [0.90970649]  
 [0.9259367 ]]  
----epoch- 5 ends-----

9. Write a program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set. You can use Java/Python ML library classes/API

import pandas as pd

from pgmpy.estimators import MaximumLikelihoodEstimator

from pgmpy.models import BayesianNetwork

from pgmpy.inference import VariableElimination

data=pd.read\_csv(r'heart1.csv')

print("first few rows of the dataset:")

print(data.head())

print("\n columns in dataset:")

print(data.columns)

model=BayesianNetwork([('age','heartdisease'),

('gender','heartdisease'),

('family','heartdisease'),

('diet','heartdisease'),

('lifestyle','heartdisease'),

('cholestrol','heartdisease')])

model.fit(data,estimator=MaximumLikelihoodEstimator)

infer=VariableElimination(model)

q1=infer.query(variables=['heartdisease'],evidence={'cholestrol':2})

print("\n query for 'heartdisease' given 'cholestrol=2':")

print(q1)

q2=infer.query(variables=['heartdisease'],evidence={'diet':1})

print("\n query for 'heartdisease' given 'diet=1':")

print(q2)

Output :-

first few rows of the dataset:  
 age gender family diet lifestyle cholestrol heartdisease  
0 0 0 1 1 3 0 1  
1 0 1 1 1 3 0 1  
2 1 0 0 0 2 1 1  
3 4 0 1 1 3 2 0  
4 3 1 1 0 0 2 0  
  
 columns in dataset:  
Index(['age', 'gender', 'family', 'diet', 'lifestyle', 'cholestrol',  
 'heartdisease'],  
 dtype='object')  
  
 query for 'heartdisease' given 'cholestrol=2':  
+-----------------+---------------------+  
| heartdisease | phi(heartdisease) |  
+=================+=====================+  
| heartdisease(0) | 0.5072 |  
+-----------------+---------------------+  
| heartdisease(1) | 0.4928 |  
+-----------------+---------------------+  
  
 query for 'heartdisease' given 'diet=1':  
+-----------------+---------------------+  
| heartdisease | phi(heartdisease) |  
+=================+=====================+  
| heartdisease(0) | 0.4952 |  
+-----------------+---------------------+  
| heartdisease(1) | 0.5048 |  
+-----------------+---------------------+

10. Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program.

from sklearn.cluster import KMeans

from sklearn import preprocessing

from sklearn.mixture import GaussianMixture

from sklearn.datasets import load\_iris

import sklearn.metrics as sm

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

dataset=load\_iris()

X=pd.DataFrame(dataset.data)

X.columns=['sepal\_length','sepal\_width','petal\_length','petal\_width']

y=pd.DataFrame(dataset.target)

y.columns=['targets']

plt.figure(figsize=(14,7))

colormap=np.array(['red','lime','black'])

plt.subplot(1,3,1)

plt.scatter(X.petal\_length,X.petal\_width,c=colormap[y.targets],s=40)

plt.title('real')

plt.subplot(1,3,2)

model=KMeans(n\_clusters=3)

model.fit(X)

predY=np.choose(model.labels\_,[0,1,2]).astype(np.int64)

plt.scatter(X.petal\_length,X.petal\_width,c=colormap[predY],s=40)

plt.title('KMeans')

scaler=preprocessing.StandardScaler()

scaler.fit(X)

xsa=scaler.transform(X)

xs=pd.DataFrame(xsa,columns=X.columns)

gmm=GaussianMixture(n\_components=3)

gmm.fit(xs)

y\_cluster\_gmm=gmm.predict(xs)

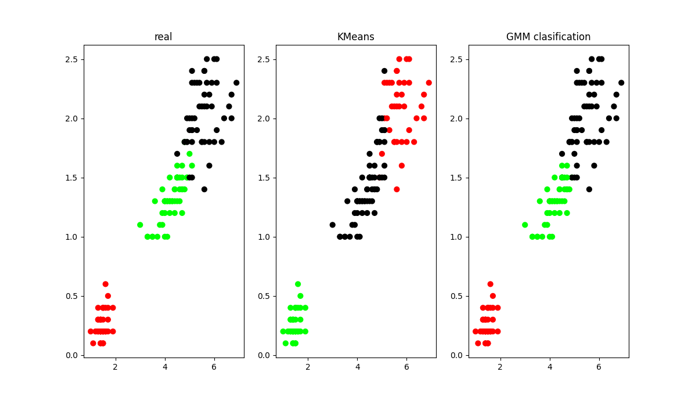
plt.subplot(1,3,3)

plt.scatter(X.petal\_length,X.petal\_width,c=colormap[y\_cluster\_gmm],s=40)

plt.title('GMM clasification')

plt.show()

Output :-



11. Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem.

from sklearn.model\_selection import train\_test\_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import classification\_report,confusion\_matrix

from sklearn import datasets

iris=datasets.load\_iris()

x=iris.data

y=iris.target

print('feature names:',iris.feature\_names)

print('classes:0-iris-setosa,1-iris-versicolour,2-iris-virginica')

x\_train,x\_test,y\_train,y\_test=train\_test\_split(x,y,test\_size=0.3,random\_state=42)

classifier=KNeighborsClassifier(n\_neighbors=5)

classifier.fit(x\_train,y\_train)

y\_pred=classifier.predict(x\_test)

print('confusion matrix:')

print(confusion\_matrix(y\_test,y\_pred))

print('classification report:')

print(classification\_report(y\_test,y\_pred))

Output :-

feature names: ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']  
classes:0-iris-setosa,1-iris-versicolour,2-iris-virginica  
confusion matrix:  
[[19 0 0]  
 [ 0 13 0]  
 [ 0 0 13]]  
classification report:  
 precision recall f1-score support  
  
 0 1.00 1.00 1.00 19  
 1 1.00 1.00 1.00 13  
 2 1.00 1.00 1.00 13  
  
 accuracy 1.00 45  
 macro avg 1.00 1.00 1.00 45  
weighted avg 1.00 1.00 1.00 45

12. Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs

from math import ceil

import numpy as np

from scipy import linalg

import matplotlib.pyplot as plt

def lowess(x, y, f, iterations):

n = len(x)

r = int(ceil(f \* n))

h = np.array([np.sort(np.abs(x - xi))[r] for xi in x])

w = np.clip(np.abs((x[:, None] - x[None, :]) / h), 0.0, 1.0)

w = (1 - w \* 3) \* 3

yest = np.zeros(n)

delta = np.ones(n)

for iteration in range(iterations):

for i in range(n):

weights = delta \* w[:, i]

b = np.array([np.sum(weights \* y), np.sum(weights \* y \* x)])

A = np.array([[np.sum(weights), np.sum(weights \* x)],

[np.sum(weights \* x), np.sum(weights \* x \* x)]])

beta = linalg.solve(A, b)

yest[i] = beta[0] + beta[1] \* x[i]

residuals = y - yest

s = np.median(np.abs(residuals))

delta = np.clip(residuals / (6.0 \* s), -1, 1)

delta = (1 - delta \* 2) \* 2

return yest

n = 100

x = np.linspace(0, 2 \* np.pi, n)

y = np.sin(x) + 0.3 \* np.random.randn(n)

f = 0.25

iterations = 3

yest = lowess(x, y, f, iterations)

plt.figure(figsize=(10, 6))

plt.plot(x, y, "r.", label='Data')

plt.plot(x, yest, "b-", label='Lowess Smoothing')

plt.xlabel('x')

plt.ylabel('y')

plt.title('Lowess Smoothing')

plt.legend()

plt.show()

Output :-

