## Program 1(A\*)

class Node():

def init (self,parent=None,position=None): self.parent=parent

self.position=position self.g=0

self.h=0 self.f=0

def eq (self,other):

return self.position==other.position

def astar(maze,start,end): start\_node=Node(None,start) start\_node.g=start\_node.h=start\_node.f=0 end\_node=Node(None,end) end\_node.g=end\_node.h=end\_node.f=0 open\_list=[]

closed\_list=[] open\_list.append(start\_node)

while len(open\_list)>0: current\_node=open\_list[0] current\_index=0

for index,item in enumerate(open\_list): if item.f<current\_node.f:

current\_node=item current\_index=index

open\_list.pop(current\_index) closed\_list.append(current\_node) if current\_node==end\_node:

path=[] current=current\_node

while current is not None: path.append(current.position) current=current.parent

return path[::-1] children=[]

for new\_position in [(0,-1),(0,1),(-1,0),(1,0),(-1,-1),(- 1,1),(1,-1),(1,1)]:

node\_position=(current\_node.position[0]+new\_position[0],cu rrent\_node.position[1]+new\_position[1])

if node\_position[0]>(len(maze)-

1) or node\_position[0]<0 or node\_position[1]>(len(maze[len(maz e)-1])-1) or node\_position[1]<0:

continue

if maze[node\_position[0]][node\_position[1]]!=0: continue

new\_node=Node(current\_node,node\_position) children.append(new\_node)

for child in children:

for closed\_child in closed\_list: if child==closed\_child:

continue child.g=current\_node.g+1 child.h=((child.position[0]-

end\_node.position[0])\*\*2)+((child.position[1]- end\_node.position[1])\*\*2)

child.f=child.g+child.h for open\_node in open\_list:

if child==open\_node and child.g>open\_node.g: continue

open\_list.append(child)

def main(): maze=[[0,0,0,0,1,0],

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

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

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

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

[0,0,0,0,0,0]]

graph=[[0,1,0,0,0,0],

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

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

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

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

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

start=(0,0) end=(5,5) end1=(5,5)

path=astar(maze,start,end) print("Path: ",path) path1=astar(graph,start,end) print("Path1: ",path1)

if name ==' main ': main()

## Program 3(Candidate elimination)

import pandas as pd import numpy as np

data=pd.DataFrame(data=pd.read\_csv("/content/enjoySport\_IS027. csv"))

concepts=np.array(data.iloc[:,0:-1]) print(concepts) target=np.array(data.iloc[:,-1]) print(target)

def learn(concepts,target): specific\_h=concepts[0].copy() print("initialize specific\_h and general\_h") print(specific\_h)

general\_h=[["?" for i in range(len(specific\_h))] for i in ra nge(len(specific\_h))]

print(general\_h)

for i,h in enumerate(concepts): if target[i]=="yes":

for x in range(len(specific\_h)): if h[x]!=specific\_h[x]:

specific\_h[x]='?' general\_h[x][x]='?'

if target[i]=="no":

for x in range(len(specific\_h)): if h[x]!=specific\_h[x]:

general\_h[x][x]=specific\_h[x] else:

general\_h[x][x]='?'

print("Specific Boundary after ",i+1," Instance is ",speci fic\_h)

print("Generic Boundary after ",i+1," Instance is ",genera l\_h)

print("\n")

indices=[i for i,val in enumerate(general\_h) if val==['?','? ','?','?','?','?']]

for i in indices: general\_h.remove(['?','?','?','?','?','?'])

return specific\_h,general\_h

s\_final,g\_final=learn(concepts,target) print("Final Specific\_h: ",s\_final,sep="\n") print("Final General\_h: ",g\_final,sep="\n")

**Program 4(ID3)**

import numpy as np

import pandas as pd

def entropy(target\_col):

  val,counts = np.unique(target\_col,return\_counts = True)

  ent=sum((counts[i]/np.sum(counts))\*np.log2( counts[i]/np.sum(counts))

  for i in range(len(val)))

  return ent

def infoGain(data,features,target):

  te = entropy(data[target])

  val,counts = np.unique(data[features],return\_counts = True)

  eg = sum((counts[i]/sum(counts)) \* entropy(data[data[features] == val[i]][target] )

  for i in range(len(val)))

  InfoGain = te-eg

  return InfoGain

def ID3(data,features,target,pnode):

 if len(np.unique(data[target])) == 1:

  return np.unique(data[target])[0]

 elif len(features) == 0:

  return pnode

 else:

  pnode=np.unique(data[target])[np.argmax(np.unique(data[target])[1])]

  IG = [infoGain(data,f,target) for f in features]

  index = np.argmax(IG)

  col = features[index]

  tree = {col:{}}

  features = [f for f in features if f!=col]

  for val in np.unique(data[col]):

    sub\_data = data[data[col]==val].dropna()

    subtree = ID3(sub\_data,features,target,pnode)

    tree[col][val] = subtree

  return tree

data = pd.read\_csv('ID3.csv')

testData = data.sample(frac = 0.1)

data.drop(testData.index,inplace = True)

print(data)

target = 'answer'

features = data.columns[data.columns!=target]

tree = ID3(data,features,target,None)

print (tree)

test = testData.to\_dict('records')[0]

print(test,'=>', test['answer'])

# Program 5(ANN)

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=7000 lr=0.1

inputlayer\_neurons=2 hiddenlayer\_neurons=3 output\_neurons=1

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

bh=np.random.uniform(size=(1,hiddenlayer\_neurons)) wout=np.random.uniform(size=(hiddenlayer\_neurons,output\_neuron s))

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)

outinp1=np.dot(hlayer\_act,wout) outinp=outinp1+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 bout+=np.sum(d\_output,axis=0,keepdims=True)\*lr wh+=X.T.dot(d\_hiddenlayer)\*lr

print("Input: \n"+str(X)) print("Actual output: \n"+str(y)) print("Predicted output: \n",output)

**Program 6(naïve Bayesian)**

import csv

import pandas as pd

import numpy as np

from sklearn.naive\_bayes import GaussianNB

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

x = np.array(data.iloc[:,0:-1])

y = np.array(data.iloc[:,-1])

print(data.head())

model = GaussianNB()

model.fit(x,y)

predicted= model.predict([[6,149,78,35,0,34,0.625,54]])

print("Predicted Value:", predicted)

## Program 7(EM)

import matplotlib.pyplot as plt from sklearn.cluster import KMeans import pandas as pd

import numpy as np

names = ['Sepal\_Length','Sepal\_Width','Petal\_Length','Petal\_Wi dth', 'Class']

dataset = pd.read\_csv("8dataset.csv", names=names) X = dataset.iloc[:, :-1]

label = {'Iris-setosa': 0,'Iris-versicolor': 1, 'Iris- virginica': 2}

y = [label[c] for c in dataset.iloc[:, -1]] model = KMeans(n\_clusters=3)

model.fit(X) plt.figure(figsize=(14,14))

colormap = np.array(['red', 'lime', 'black']) plt.subplot(2, 2, 1)

plt.scatter(X.Petal\_Length, X.Petal\_Width, c=colormap[y], s=40

)

plt.title('Real Clusters') plt.xlabel('Petal Length') plt.ylabel('Petal Width') plt.subplot(2, 2, 2)

plt.scatter(X.Petal\_Length, X.Petal\_Width, c=colormap[model.la bels\_], s=40)

plt.title('K-Means Clustering') plt.xlabel('Petal Length') plt.ylabel('Petal Width')

from sklearn import preprocessing scaler = preprocessing.StandardScaler() scaler.fit(X)

xsa = scaler.transform(X)

xs = pd.DataFrame(xsa, columns = X.columns) from sklearn.mixture import GaussianMixture gmm = GaussianMixture(n\_components=3) gmm.fit(xs)

gmm\_y = gmm.predict(xs) plt.subplot(2, 2, 3)

plt.scatter(X.Petal\_Length, X.Petal\_Width, c=colormap[gmm\_y], s=40)

plt.title('GMM Clustering') plt.xlabel('Petal Length') plt.ylabel('Petal Width')

print('Observation: The GMM using EM algorithm based clusterin g matched the true labels more closely than the Kmeans.')

# Program 8(KNN)

from sklearn.model\_selection import train\_test\_split from sklearn.neighbors import KNeighborsClassifier from sklearn import datasets

iris=datasets.load\_iris() print("Iris Data set loaded...")

x\_train, x\_test, y\_train, y\_test = train\_test\_split(iris.data, iris.target,test\_size=0.2)

print("Dataset is split into training and testing...") print("Size of trainng data and its label",x\_train.shape,y\_tra in.shape)

print("Size of testing data and its label",x\_test.shape, y\_tes t.shape)

classifier = KNeighborsClassifier(n\_neighbors=1) classifier.fit(x\_train, y\_train) y\_pred=classifier.predict(x\_test)

print("Results of Classification using K-nn with K=1 ") for r in range(0,len(x\_test)):

print(" Sample:", str(x\_test[r]), " Actual-

label:", str(y\_test[r]), " Predicted-label:",str(y\_pred[r])) print("Classification Accuracy :" , classifier.score(x\_test,y\_ test));

# Program 9(Regression)

import matplotlib.pyplot as plt import pandas as pd

import numpy as np

def kernel(point, xmat, k): m,n = np.shape(xmat)

weights = np.mat(np.eye((m))) for j in range(m):

diff = point - X[j]

weights[j,j] = np.exp(diff\*diff.T/(-2.0\*k\*\*2)) return weights

def localWeight(point, xmat, ymat, k): wei = kernel(point,xmat,k)

W = (X.T\*(wei\*X)).I\*(X.T\*(wei\*ymat.T)) return W

def localWeightRegression(xmat, ymat, k): m,n = np.shape(xmat)

ypred = np.zeros(m) for i in range(m):

ypred[i] = xmat[i]\*localWeight(xmat[i],xmat,ymat,k) return ypred

data = pd.read\_csv('data10\_tips.csv') bill = np.array(data.total\_bill)

tip = np.array(data.tip)

mbill = np.mat(bill) mtip = np.mat(tip)

m= np.shape(mbill)[1] one = np.mat(np.ones(m))

X = np.hstack((one.T,mbill.T))

ypred = localWeightRegression(X,mtip,2) SortIndex = X[:,1].argsort(0)

xsort = X[SortIndex][:,0]

fig = plt.figure()

ax = fig.add\_subplot(1,1,1) ax.scatter(bill,tip, color='green')

ax.plot(xsort[:,1],ypred[SortIndex], color = 'red', linewidth= 10)

plt.xlabel('Total bill') plt.ylabel('Tip') plt.show()