A star

def astaralgo(start\_node,stop\_node):

open\_set = set(start\_node)

close\_set = set()

g = {}

parent = {}

g[start\_node] = 0

parent[start\_node] = start\_node

while len(open\_set) > 0:

n = None

for v in open\_set:

if n == None or g[v]+heuristic(v) < g[n] + heuristic(v):

n = v

if n == stop\_node or graph\_node[n] == None:

pass

else:

for (m,weight) in get\_negh(n):

if m not in open\_set and m not in close\_set:

open\_set.add(m)

parent[m] = n

g[m] = g[n] + weight

else:

if g[m] > g[n]+weight:

g[m] = g[n] + weight

parent[m] = n

if m in close\_set:

close\_set.remove(m)

open\_set.add(m)

if n == stop\_node:

path = []

while parent[n] != n:

path.append(n)

n = parent[n]

path.append(start\_node)

path.reverse()

print("Path found: {}".format(path))

return path

open\_set.remove(n)

close\_set.add(n)

print('Path does not exist!')

return None

def heuristic(n):

H\_dist = {'A': 11,'B': 6,'C': 99,'D': 1,'E': 7,'G': 0}

return H\_dist[n]

def get\_negh(v):

if v in graph\_node:

return graph\_node[v]

else:

return None

graph\_node = {'A':[('B',2),('E',3)],

'B':[('C',1),('G',9)],

'C':None,

'D':[('G',1)],

'E':[('D',6)]

}

astaralgo('A', 'E')

Candidate elimination

import numpy as np

import pandas as pd

data = pd.DataFrame(pd.read\_csv("data.csv"))

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

print(concepts)

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

print(target)

def learn(concept,target):

specific\_h = concepts[0].copy()

print("Initialization of specific\_h and general\_h")

print("Specific\_h : ",specific\_h)

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

print()

print("General\_h : ",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("\nSteps of candidate elimination algorithm",i+1)

print("\nSpecific\_h : ",specific\_h)

print("\nGeneral\_h : ",general\_h)

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()

print("Final Specific\_h : ",s\_final,sep = "\n")

print("Final Genaral\_h : ",g\_final,sep = "\n")

Back Propogation

import numpy as np

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

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

Z=np.amax(X,axis=0)

X=X/Z

Y=Y/100

def sigmoid(X):

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

def derivatives\_sigmoid(X):

return X\*(1-X)

epoch=5000

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):

hinpl=np.dot(X,wh)

hinp=hinpl+bh

hlayer\_act=sigmoid(hinp)

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

outinp=outinpl+bout

output=sigmoid(outinp)

E0=Y-output

outgrad=derivatives\_sigmoid(output)

d\_output= E0\*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("Input: \n" + str(X))

print("Actual Output: \n" + str(Y))

print("Predicted Output: \n" ,output)