**AIML Termworks**

***Part A***

**Termwork 1: Implement DFID algorithm and compare its performance with DFS and BFS algorithm**

graph={

'A':['B','C'],

'B':['D','E'],

'C':['G'],

'D':[],

'E':['F'],

'G':[],

'F':[]

}

def iterativeDDFS(curnode,dnode,graph,maxd):

for i in range(maxd):

if DFS(curnode,dnode,graph,i):

return True

def DFS(curnode,dnode,graph,maxd):

print("Checking for GoalNode",curnode)

if curnode==dnode:

return True

if maxd<=0:

return False

for node in graph[curnode]:

if DFS(node,dnode,graph,maxd-1):

return True

return False

if not iterativeDDFS('A','F',graph,4):

print("Path does not exist")

else:

print("A path exists")

**OUTPUT**

C:\Users\Harshita\PycharmProjects\pythonProject1\venv\Scripts\python.exe C:/Users/Harshita/PycharmProjects/pythonProject1/main.py

Checking for GoalNode A

Checking for GoalNode A

Checking for GoalNode B

Checking for GoalNode C

Checking for GoalNode A

Checking for GoalNode B

Checking for GoalNode D

Checking for GoalNode E

Checking for GoalNode C

Checking for GoalNode G

Checking for GoalNode A

Checking for GoalNode B

Checking for GoalNode D

Checking for GoalNode E

Checking for GoalNode F

A path exists

**Termwork 2: Implement Best-First Search algorithm**

SuccList = {'S': [['A', 3], ['B', 6], ['C', 5]],

'A': [['E', 8], ['D', 9]],

'B': [['G', 14], ['F', 12]],

'C': [['H', 7]],

'H': [['J', 6], ['I', 5]],

'I': [['M', 2], ['L', 10], ['K', 1]]}

Start = input("Enter Source node >> ").upper()

Goal = input("Enter Goal node >> ").upper()

Closed = list()

SUCCESS = True

FAILURE = False

State = FAILURE

def GOALTEST(N):

if N == Goal:

return True

else:

return False

def MOVGEN(N):

New\_list = list()

if N in SuccList.keys():

New\_list = SuccList[N]

return New\_list

def APPEND(L1, L2):

New\_list = list(L1)+list(L2)

return New\_list

def SORT(L):

L.sort(key = lambda x: x[1])

return L

def BestFirstSearch():

OPEN = [[Start, 5]]

CLOSED = list()

global State

global Closed

i = 1

while (len(OPEN) != 0) and (State != SUCCESS):

print("\n<<<<<<<<<<----({})---->>>>>>>>>>\n".format(i))

N = OPEN[0]

print("N: ", N)

del OPEN[0] #delete the node we picked

if GOALTEST(N[0]) == True:

State = SUCCESS

CLOSED = APPEND(CLOSED, [N])

print("CLOSED: ", CLOSED)

else:

CLOSED = APPEND(CLOSED, [N])

print("CLOSED: " ,CLOSED)

CHILD = MOVGEN(N[0])

print("CHILD: ", CHILD)

for val in OPEN:

if val in CHILD:

CHILD.remove(val)

for val in CLOSED:

if val in CHILD:

CHILD.remove(val)

OPEN = APPEND(CHILD, OPEN)

print("Unsorted OPEN: ", OPEN)

SORT(OPEN)

print("Sorted OPEN=", OPEN)

Closed = CLOSED

i += 1

return State

result = BestFirstSearch()

print("Best First Search Path >>>>> {} <<<<{}>>>>".format(Closed, result))

**OUTPUT**

C:\Users\Harshita\PycharmProjects\pythonProject1\venv\Scripts\python.exe C:/Users/Harshita/PycharmProjects/pythonProject1/main.py

Enter Source node >> S

Enter Goal node >> I

<<<<<<<<<<----(1)---->>>>>>>>>>

N: ['S', 5]

CLOSED: [['S', 5]]

CHILD: [['A', 3], ['B', 6], ['C', 5]]

Unsorted OPEN: [['A', 3], ['B', 6], ['C', 5]]

Sorted OPEN= [['A', 3], ['C', 5], ['B', 6]]

<<<<<<<<<<----(2)---->>>>>>>>>>

N: ['A', 3]

CLOSED: [['S', 5], ['A', 3]]

CHILD: [['E', 8], ['D', 9]]

Unsorted OPEN: [['E', 8], ['D', 9], ['C', 5], ['B', 6]]

Sorted OPEN= [['C', 5], ['B', 6], ['E', 8], ['D', 9]]

<<<<<<<<<<----(3)---->>>>>>>>>>

N: ['C', 5]

CLOSED: [['S', 5], ['A', 3], ['C', 5]]

CHILD: [['H', 7]]

Unsorted OPEN: [['H', 7], ['B', 6], ['E', 8], ['D', 9]]

Sorted OPEN= [['B', 6], ['H', 7], ['E', 8], ['D', 9]]

<<<<<<<<<<----(4)---->>>>>>>>>>

N: ['B', 6]

CLOSED: [['S', 5], ['A', 3], ['C', 5], ['B', 6]]

CHILD: [['G', 14], ['F', 12]]

Unsorted OPEN: [['G', 14], ['F', 12], ['H', 7], ['E', 8], ['D', 9]]

Sorted OPEN= [['H', 7], ['E', 8], ['D', 9], ['F', 12], ['G', 14]]

<<<<<<<<<<----(5)---->>>>>>>>>>

N: ['H', 7]

CLOSED: [['S', 5], ['A', 3], ['C', 5], ['B', 6], ['H', 7]]

CHILD: [['J', 6], ['I', 5]]

Unsorted OPEN: [['J', 6], ['I', 5], ['E', 8], ['D', 9], ['F', 12], ['G', 14]]

Sorted OPEN= [['I', 5], ['J', 6], ['E', 8], ['D', 9], ['F', 12], ['G', 14]]

<<<<<<<<<<----(6)---->>>>>>>>>>

N: ['I', 5]

CLOSED: [['S', 5], ['A', 3], ['C', 5], ['B', 6], ['H', 7], ['I', 5]]

Best First Search Path >>>>> [['S', 5], ['A', 3], ['C', 5], ['B', 6], ['H', 7]] <<<<True>>>>

Process finished with exit code 0

**Termwork 3: Implementation of AND/OR/NOT Gate using single layer perceptron**

# importing Python library

import numpy as np

# define Unit Step Function

def unitStep(v):

if v >= 0:

return 1

else:

return 0

# design Perceptron Model

def perceptronModel(x, w, b):

v = np.dot(w, x) + b

y = unitStep(v)

return y

# OR Logic Function

# w1 = 1, w2 = 1, b = -0.5

def OR\_logicFunction(x):

w = np.array([1, 1])

b = -0.5

return perceptronModel(x, w, b)

#NOT Logic Function

# w = -1, b = 0.5

def NOT\_logicFunction(x):

w = -1

b = 0.5

return perceptronModel(x, w, b)

# AND Logic Function

# w1 = 1, w2 = 1, b = -1.5

def AND\_logicFunction(x):

w = np.array([1, 1])

b = -1.5

return perceptronModel(x, w, b)

def XOR\_logicFunction(x):

y1 = AND\_logicFunction(x)

y2 = OR\_logicFunction(x)

y3 = NOT\_logicFunction(y1)

final\_x = np.array([y2, y3])

finalOutput = AND\_logicFunction(final\_x)

return finalOutput

# testing the Perceptron Model

test1 = np.array([0, 1])

test2 = np.array([1, 1])

test3 = np.array([0, 0])

test4 = np.array([1, 0])

test5 = np.array(1)

test6 = np.array(0)

test7 = np.array([0, 1])

test8 = np.array([1, 1])

test9 = np.array([0, 0])

test10 = np.array([1, 0])

test11 = np.array([0, 1])

test12 = np.array([1, 1])

test13 = np.array([0, 0])

test14 = np.array([1, 0])

print("OR({}, {}) = {}".format(0, 1, OR\_logicFunction(test1)))

print("OR({}, {}) = {}".format(1, 1, OR\_logicFunction(test2)))

print("OR({}, {}) = {}".format(0, 0, OR\_logicFunction(test3)))

print("OR({}, {}) = {}".format(1, 0, OR\_logicFunction(test4)))

print("\n")

print("NOT({}) = {}".format(1, NOT\_logicFunction(test5)))

print("NOT({}) = {}".format(0, NOT\_logicFunction(test6)))

print("\n")

print("AND({}, {}) = {}".format(0, 1, AND\_logicFunction(test7)))

print("AND({}, {}) = {}".format(1, 1, AND\_logicFunction(test8)))

print("AND({}, {}) = {}".format(0, 0, AND\_logicFunction(test9)))

print("AND({}, {}) = {}".format(1, 0, AND\_logicFunction(test10)))

print("\n")

print("XOR({}, {}) = {}".format(0, 1, XOR\_logicFunction(test11)))

print("XOR({}, {}) = {}".format(1, 1, XOR\_logicFunction(test12)))

print("XOR({}, {}) = {}".format(0, 0, XOR\_logicFunction(test13)))

print("XOR({}, {}) = {}".format(1, 0, XOR\_logicFunction(test14)))

print("\n")

**OUTPUT**

C:\Users\Harshita\PycharmProjects\pythonProject1\venv\Scripts\python.exe C:/Users/Harshita/PycharmProjects/pythonProject1/main.py

OR(0, 1) = 1

OR(1, 1) = 1

OR(0, 0) = 0

OR(1, 0) = 1

NOT(1) = 0

NOT(0) = 1

AND(0, 1) = 0

AND(1, 1) = 1

AND(0, 0) = 0

AND(1, 0) = 0

XOR(0, 1) = 1

XOR(1, 1) = 0

XOR(0, 0) = 0

XOR(1, 0) = 1

**Termwork 4: Implementation of XOR Gate using**

**a) Multi-layer perceptron/Error back propagation**

import numpy as np

#np.random.seed(0)

def sigmoid (x):

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

def sigmoid\_derivative(x):

return x \* (1 - x)

#Input datasets

inputs = np.array([[0,0],[0,1],[1,0],[1,1]])

expected\_output = np.array([[0],[1],[1],[0]])

epochs = 10000

lr = 0.1

inputLayerNeurons, hiddenLayerNeurons, outputLayerNeurons = 2,2,1

#Random weights and bias initialization

hidden\_weights = np.random.uniform(size=(inputLayerNeurons,hiddenLayerNeurons))

hidden\_bias =np.random.uniform(size=(1,hiddenLayerNeurons))

output\_weights = np.random.uniform(size=(hiddenLayerNeurons,outputLayerNeurons))

output\_bias = np.random.uniform(size=(1,outputLayerNeurons))

print("Initial hidden weights: ",end='')

print(\*hidden\_weights)

print("Initial hidden biases: ",end='')

print(\*hidden\_bias)

print("Initial output weights: ",end='')

print(\*output\_weights)

print("Initial output biases: ",end='')

print(\*output\_bias)

#Training algorithm

for \_ in range(epochs):

#Forward Propagation

hidden\_layer\_activation = np.dot(inputs,hidden\_weights)

hidden\_layer\_activation += hidden\_bias

hidden\_layer\_output = sigmoid(hidden\_layer\_activation)

output\_layer\_activation = np.dot(hidden\_layer\_output,output\_weights)

output\_layer\_activation += output\_bias

predicted\_output = sigmoid(output\_layer\_activation)

#Backpropagation

error = expected\_output - predicted\_output

d\_predicted\_output = error \* sigmoid\_derivative(predicted\_output)

error\_hidden\_layer = d\_predicted\_output.dot(output\_weights.T)

d\_hidden\_layer = error\_hidden\_layer \* sigmoid\_derivative(hidden\_layer\_output)

#Updating Weights and Biases

output\_weights += hidden\_layer\_output.T.dot(d\_predicted\_output) \* lr

output\_bias += np.sum(d\_predicted\_output,axis=0,keepdims=True) \* lr

hidden\_weights += inputs.T.dot(d\_hidden\_layer) \* lr

hidden\_bias += np.sum(d\_hidden\_layer,axis=0,keepdims=True) \* lr

print("Final hidden weights: ",end='')

print(\*hidden\_weights)

print("Final hidden bias: ",end='')

print(\*hidden\_bias)

print("Final output weights: ",end='')

print(\*output\_weights)

print("Final output bias: ",end='')

print(\*output\_bias)

print("\nOutput from neural network after 10,000 epochs: ",end='')

print(\*predicted\_output)

**OUTPUT**

C:\Users\Harshita\PycharmProjects\pythonProject1\venv\Scripts\python.exe C:/Users/Harshita/PycharmProjects/pythonProject1/main.py

Initial hidden weights: [0.3189444 0.21252212] [0.60957507 0.0087231 ]

Initial hidden biases: [0.30953359 0.54906098]

Initial output weights: [0.49361275] [0.85381414]

Initial output biases: [0.01259271]

Final hidden weights: [5.8082732 3.54400035] [5.79814636 3.54227662]

Final hidden bias: [-2.34403935 -5.40193325]

Final output weights: [7.21326209] [-7.77631086]

Final output bias: [-3.24653175]

Output from neural network after 10,000 epochs: [0.0659985] [0.93689166] [0.93692689] [0.06959378]

Process finished with exit code 0

**Termwork 5: Implement Hebbian learning rule and Correlation learning rule**

#xi=[ip1,ip2]

x1=[1,1]

x2=[1,-1]

x3=[-1,1]

x4=[-1,-1]

xilist=[x1,x2,x3,x4]

y=[1,-1,-1,-1]

w1=w2=bw=0

b=1

def heb\_learn():

global w1,w2,bw

print("dw1\tdw2\tdb\tw1\tw2\tb")

i=0

for xi in xilist:

dw1=xi[0]\*y[i]

dw2=xi[1]\*y[i]

db=y[i]

w1=w1+dw1

w2=w2+dw2

bw+=db

print(dw1,dw2,db,w1,w2,bw,sep='\t')

i+=1

print("Learning...")

heb\_learn()

print("Learning completed")

print("Output of AND gate using obtained w1,w2,bw:")

print("x1\tx2\ty")

for xi in xilist:

print(xi[0],xi[1],1 if w1\*xi[0]+w2\*xi[1]+b\*bw>0 else -1,sep='\t')

**OUTPUT**

C:\Users\Harshita\PycharmProjects\pythonProject1\venv\Scripts\python.exe C:/Users/Harshita/PycharmProjects/pythonProject1/main.py

Learning...

dw1 dw2 db w1 w2 b

1 1 1 1 1 1

-1 1 -1 0 2 0

1 -1 -1 1 1 -1

1 1 -1 2 2 -2

Learning completed

Output of AND gate using obtained w1,w2,bw:

x1 x2 y

1 1 1

1 -1 -1

-1 1 -1

-1 -1 -1

***Part B***

**Termwork 1: Implement Find-S algorithms**

import pandas as pd

import numpy as np

#to read the data in the csv file

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

print(data,"n")

#making an array of all the attributes

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

print("n The attributes are: ",d)

#segragating the target that has positive and negative examples

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

print("n The target is: ",target)

#training function to implement find-s algorithm

def train(c,t):

for i, val in enumerate(t):

if val == "Yes":

specific\_hypothesis = c[i].copy()

break

for i, val in enumerate(c):

if t[i] == "Yes":

for x in range(len(specific\_hypothesis)):

if val[x] != specific\_hypothesis[x]:

specific\_hypothesis[x] = '?'

else:

pass

return specific\_hypothesis

#obtaining the final hypothesis

print("n The final hypothesis is:",train(d,target))

**OUTPUT**

C:\Users\Harshita\PycharmProjects\pythonProject1\venv\Scripts\python.exe C:/Users/Harshita/PycharmProjects/pythonProject1/main.py

[['sunny', 'warm', 'normal', 'strong', 'warm', 'same', 'yes'], ['sunny', 'warm', 'high', 'strong', 'warm', 'same', 'yes'], ['rainy', 'cold', 'high', 'strong', 'warm', 'change', 'no'], ['sunny', 'warm', 'high', 'strong', 'cool', 'change', 'yes']]

The total number of training instances are : 4

The initial hypothesis is :

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

Instance 1 is ['sunny', 'warm', 'normal', 'strong', 'warm', 'same', 'yes'] and is Positive Instance

The hypothesis for the training instance 1 is: ['sunny', 'warm', 'normal', 'strong', 'warm', 'same']

Instance 2 is ['sunny', 'warm', 'high', 'strong', 'warm', 'same', 'yes'] and is Positive Instance

The hypothesis for the training instance 2 is: ['sunny', 'warm', '?', 'strong', 'warm', 'same']

Instance 3 is ['rainy', 'cold', 'high', 'strong', 'warm', 'change', 'no'] and is Negative Instance Hence Ignored

The hypothesis for the training instance 3 is: ['sunny', 'warm', '?', 'strong', 'warm', 'same']

Instance 4 is ['sunny', 'warm', 'high', 'strong', 'cool', 'change', 'yes'] and is Positive Instance

The hypothesis for the training instance 4 is: ['sunny', 'warm', '?', 'strong', '?', '?']

The Maximally specific hypothesis for the training instance is ['sunny', 'warm', '?', 'strong', '?', '?']

Process finished with exit code 0

**Termwork 2: Build a linear regression model housing price**

import sys

import subprocess

subprocess.check\_call([sys.executable, '-m','pip','install','sklearn'])

import pandas as pd

import numpy as np

from sklearn import linear\_model

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

from sklearn.datasets import load\_boston

boston = load\_boston()

print(boston)

df\_x = pd.DataFrame(boston.data,columns = boston.feature\_names)

df\_y = pd.DataFrame(boston.target)

df\_x.describe()

reg = linear\_model.LinearRegression()

x\_train,x\_test, y\_train, y\_test = train\_test\_split(df\_x , df\_y,test\_size=0.33, random\_state = 42)

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

print(reg.coef\_)

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

print(y\_pred)

y\_pred[2]

y\_test[0]

print(np.mean((y\_pred-y\_test)\*\*2))

from sklearn.metrics import mean\_squared\_error

print(mean\_squared\_error(y\_test,y\_pred))

**Termwork 3: Implement spam detection using Naïve Bayes Algorithm**

import numpy as np

import pandas as pd

emails = pd.read\_csv('/home/lab2/Downloads/emails.csv')

def process\_email(text):

text = text.lower()

return list(set(text.split()))

emails['words'] = emails['text'].apply(process\_email)

num\_emails = len(emails)

num\_spam = sum(emails['spam'])

print('Number of emails:', num\_emails)

print('Number of spam emails:', num\_spam)

print()

print('probability of spam:', num\_spam/num\_emails)

print()

model = {}

for index, email in emails.iterrows():

for word in email['words']:

if word not in model:

model[word] = {'spam': 1, 'ham': 1}

if word not in model:

if email['spam']:

model[word]['spam'] += 1

else:

model[word]['ham'] +1

def predict\_bayes(word):

word = word.lower()

num\_spam\_with\_word = model[word]['spam']

num\_ham\_with\_word = model[word]['ham']

return 1.0\*num\_spam\_with\_word/(num\_spam\_with\_word + num\_ham\_with\_word)

print('Prediction using Bayes for word sale', predict\_bayes('sale'))

print('Prediction using bayes for word lottery', predict\_bayes('lottery'))

def predict\_naive\_bayes(email):

total = len(emails)

num\_spam = sum(emails['spam'])

num\_ham = total - num\_spam

email = email.lower()

words = set(email.split())

spams = [1.0]

hams = [1.0]

for word in words:

if word in model:

spams.append(model[word]['spam']/num\_spam\*total)

hams.append(model[word]['ham']/num\_ham\*total)

prod\_spams = np.compat.long(np.prod(spams)\*num\_spam)

prod\_hams = np.compat.long(np.prod(hams)\*num\_ham)

return prod\_spams/(prod\_spams + prod\_hams)

print('Prodication using NaiveBayes for word lottery sale', predict\_naive\_bayes('lottery sale'))

print('Prodication using NaiveBayes for word lottery sale', predict\_naive\_bayes('asdfgh'))

print('Prodication using NaiveBayes for word lottery sale', predict\_naive\_bayes('Hi Mom, How are you?'))

import numpy as np

import pandas as pd

emails = pd.read\_csv('/home/lab2/Downloads/emails.csv')

def process\_email(text):

text = text.lower()

return list(set(text.split()))

emails['words'] = emails['text'].apply(process\_email)

num\_emails = len(emails)

num\_spam = sum(emails['spam'])

print('Number of emails:', num\_emails)

print('Number of spam emails:', num\_spam)

print()

print('probability of spam:', num\_spam/num\_emails)

print()

model = {}

for index, email in emails.iterrows():

for word in email['words']:

if word not in model:

model[word] = {'spam': 1, 'ham': 1}

if word not in model:

if email['spam']:

model[word]['spam'] += 1

else:

model[word]['ham'] +1

def predict\_bayes(word):

word = word.lower()

num\_spam\_with\_word = model[word]['spam']

num\_ham\_with\_word = model[word]['ham']

return 1.0\*num\_spam\_with\_word/(num\_spam\_with\_word + num\_ham\_with\_word)

print('Prediction using Bayes for word sale', predict\_bayes('sale'))

print('Prediction using bayes for word lottery', predict\_bayes('lottery'))

def predict\_naive\_bayes(email):

total = len(emails)

num\_spam = sum(emails['spam'])

num\_ham = total - num\_spam

email = email.lower()

words = set(email.split())

spams = [1.0]

hams = [1.0]

for word in words:

if word in model:

spams.append(model[word]['spam']/num\_spam\*total)

hams.append(model[word]['ham']/num\_ham\*total)

prod\_spams = np.compat.long(np.prod(spams)\*num\_spam)

prod\_hams = np.compat.long(np.prod(hams)\*num\_ham)

return prod\_spams/(prod\_spams + prod\_hams)

print('Prodication using NaiveBayes for word lottery sale', predict\_naive\_bayes('lottery sale'))

print('Prodication using NaiveBayes for word lottery sale', predict\_naive\_bayes('asdfgh'))

print('Prodication using NaiveBayes for word lottery sale', predict\_naive\_bayes('Hi Mom, How are you?'))