Date. .....

Expt. No. .....

ANACONDA (SOFTWARE) INSTALLATION PROCEDURE

Guide to install Python with Anaconda

Step 1: Download Anaconda
Step 2: Download the Python 3 version for windows
Step 3: Double-click on the executable file
Step 4: Click Next-
Step 5: Click on "I agree to the terms and conditions" button
Step 6: Select "Who you Want To Give Anaconda To"
Step 7: Select the installation location
Step 8: Select the environment variables
Step 9: Click Next and then "Finish"
Step 10: See if Python is installed

OUTPUT :

['A', 'F', 'G', 'I', 'J']

Date. .....

Expt. No. .....1..... Page No. 2

Implement A* Search Algorithm.

PROGRAM:

```
def aStarAlgo(start_node, stop_node):
    open_set = set(start_node)
    closed_set = set()
    g = {}  #store distance from starting node.
    parents = {} #parents contains an adjacency map of all nodes

    #distance of starting node from itself is zero
    g[start_node] = 0
    # start_node is root_node i.e. it has no parent nodes
    #so start_node is set to its own parent node
    parents[start_node] = start_node

    while len(open_set) > 0:
        n = None

    #node with lowest f() is found
    for v in open_set:
        if n == None or g[v] + heuristic(v) < g[n] + heuristic(n):
            n = v

    if n == stop_node or Graph_nodes[n] == None:
        pass
    else:
        for (m, weight) in get_neighbours(n):
            #nodes 'm' not in first and last set are added to first.
```

```
        #n is set to its parent.

        if m not in open_set and m not in closed_set:
            open_set.add(m)
            parents[m] = n
            g[m] = g[n] + weight

        #for each node m, compare its distance from start i.e. g(m)
        # from start through n node
        else:
            if g[m] > g[n] + weight:
                #update g[m]
                g[m] = g[n] + weight
                # change parent of m to n
                parents[m] = n

            #if m in closed_set, remove and add to open
            if m in closed_set:
                closed_set.remove(m)
                open_set.add(m)

    if n == None:
        print("path does not exist!");
        return None

    # if the current node is the stop_node
    # then we begin reconstructing the path from it to the start_node
    if n == stop_node:
        path = []
```

```
        while parents[n] != n:
            path.append(n)
            n = parents[n]
        path.append(start_node)
        path.reverse()

        print('path found: {}'.format(path))
        return path

    #remove n from the open_list and add it to the closed_list
    #because all of his neighbours were inspected
    open_set.remove(n)
    closed_set.add(n)

    print("path does not exist!")
    return None

#define function to return neighbour & its distance
#from the passed node
def get_neighbours(v):
    if v in graph_nodes[v]
        return graph_nodes[v]
    else:
        return None

#for simplicity we'll consider heuristic distances given
#and this function returns heuristic distance for all nodes
```

```
def heuristic(n):
    H_dist = {
        'A': 10,
        'B': 8,
        'C': 5,
        'D': 7,
        'E': 3,
        'F': 6,
        'G': 5,
        'H': 3,
        'I': 1,
        'J': 0
    }
return H_dist[n]

#Describe your graph here
Graph_nodes = {
    'A': [('B', 6), ('F', 3)],
    'B': [('C', 3), ('D', 2)],
    'C': [('D', 1), ('E', 5)],
    'D': [('C', 1), ('E', 8)],
    'E': [('I', 5), ('J', 5)],
    'F': [('G', 1), ('H', 7)],
    'G': [('I', 3)],
    'H': [('I', 2)],
    'I': [('E', 5), ('J', 3)],
    'J': None
}
aStarAlgo('A', 'J')
```

Date. .....

Expt. No. 2 Page No. 6

Implementation of AO* Search Algorithm

PROGRAM

```
class Graph:
  def_init_ (self, graph, heuristicNodeList, startNode):
#instantiate graph object with graph topology, heuristic values, start node
      self.graph = graph
      self.H = heuristicNodeList
      self.start = startNode
      self.parent = { }
      self.status = { }
      self.solutionGraph = { }

  def applyAOstar(self): #starts a recursive AO* algorithm
      self.aoStar(self.start, False)

  def getNeighbors(self, v): #get the Neighbors of a given node
      return self.graph.get(v, '')

  def getStatus(self, v): #return the status of a given node
      self.status[v] = val

  def getHeuristicNodeValue(self, n):
      return self.H.get(n, 0) #always return heuristic value of given node

  def printSolution(self):
    print("FOR GRAPH SOLUTION, TRAVERSE THE GRAPH FROM START NODE:",
          self.start)
```

OUTPUT:

Heuristic values : {'A': 1, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5,
'H': 7, 'I': 7, 'J': 1}

Solution graph : { }
Processing Node : A

Heuristic Values : {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5,
'H': 7, 'I': 7, 'J': 1}

Solution Graph = { }
Processing Node : B

Heuristic values : {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5,
'H': 7, 'I': 7, 'J': 1}

Solution Graph = { }
Processing Node : A

Heuristic Values : {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1,
'G': 5, 'H': 7, 'I': 7, 'J': 1}

Solution Graph = { }
Processing Node : G

Heuristic Value : {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 8,
'H': 7, 'I': 7, 'J': 1}

Solution Graph = { }
Processing Node : B

Heuristic Values : {'A': 10, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 8,
'H': 7, 'I': 7, 'J': 1}

Solution Graph = { }
Processing Node : A

Heuristic values : {'A': 12, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 8,
'H': 7, 'I': 0, 'J': 1}

Solution Graph : {'I': []}
Processing Node : G

```
print("- - ---- - -")
print(self.solutionGraph)
print("- - ---- - -")

def computeMinimumCostChildNode(self, v):
#computes the Minimum cost of child Nodes of a given node v
    minimumCost = 0
    costToChildNodeListDict = {}
    costToChildNodeListDict[minimumCost] = []
    flag = True
    for nodeInfoTuplelist in self.getNeighbors(v):
    #iterate over the set of child nodes
        cost = 0
        nodelist = []
        for c, weight in nodeInfoTuplelist:
            cost = cost + self.getHeuristicNodeValue(c) + weight
            nodelist.append(c)

        if flag == True: #initialize Minimum Cost with the cost of
                         first set of child Nodes
            minimumCost = cost
            costToChildNodeListDict[minimumCost] = nodeList
                                    #set the minimum cost child node/s.
            flag = False
        else:        #checking the Minimum cost Node/s with current
            if minimumCost > cost:                Minimum cost
                minimumCost = cost
                costToChildNodeListDict[minimumCost] = nodeList
                                    #set the Minimum cost child node/s.
```

Heuristic values : {'A': 6, 'B': 2, 'C': 1, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 0}

Solution graph = {'J': [], 'G': ['I'], 'B': ['G'], 'J': [], 'C': ['J']}

Processing Node : A

For Graph solution, Traverse the graph from the start Node : A

{'I': [], 'G': ['I'], 'B': ['G'], 'J': [], 'C': ['J'], 'A': ['B', 'C']}

Date. .....

Expt. No. .....2.....

Page No.....8.....

```
        return minimumCost, costToChildNodeListDict[minimumCost]
#return the Minimum Cost and Minimum cost child Node/s.

def aostar(self, v, backTracking):
    #AO* algorithm for a start Node & backTracking status flag
    print("HEURISTIC VALUES :", self.H)
    print("SOLUTION GRAPH :", self.solutionGraph)
    print("PROCESSING NODE :", v)
    print("-------------------")

    if self.getStatus(v) >= 0:
#if status node v >= 0, compute Minimum Cost nodes of v
        minimumCost, childNodeList = self.computeMinimumCostChildNodes(v)
        self.setHeuristicNodeValue(v, minimumCost)
        self.setStatus(v, len(childNodeList))
        solved = True #check the Minimum Cost nodes of v are resolved

    for childNode in childNodeList:
        self.parent[childNode] = v

    if self.getStatus(childNode) != -1:
        solved = solved & False

    if solved == True:
        self.setStatus(v, -1)
        self.solutionGraph[v] = childNodeList

    if v != self.start:
        self.aostar(self.parent[v], True)
```

```
if backTracking == False:
    for childNode in childNodeList:
        self.setStatus(childNode, 0)
        self.aoStar(childNode, False)

h1 = {'A': 1, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7,
      'I': 7, 'J': 1}
graph = {'A': [[('B', 1)], [('C', 1)], [('D', 1)]],
         'B': [[('G', 1)], [('H', 1)]],
         'C': [[('J', 1)]],
         'D': [[('E', 1)], [('F', 1)]],
         'G': [[('F', 1)]]
        }
G1 = Graph(graph, h1, 'A')
G1.applyAOStar()
G1.printSolution()
```

Date. .....

Expt. No. .....3.....

Page No.....

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 hypothesis consistent with the training example.

PROGRAM

```
import numpy as np
import pandas as pd

data = pd.read_csv("enjoysport.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("Initialization of specific_h and general_h")
    print(specific_h)
    general_h = [["?" for i in range(len(specific_h))] for i in
                 range(len(specific_h))]
    print(general_h)
    for i, h in enumerate(concepts):
        print("For loop starts")
        if target[i] == "yes":
            print("if instance is positive")
            for x in range(len(specific_h)):
                if h[x] != specific_h[x]:
                    specific_h[x] = '?'
                    general_h[x][x] = '?'
```

```
if target[i] == "no":
    print("if instance is negative")
    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("Steps of candidate elimination algorithm", i+1)
    print(specific_h)
    print(general_h)
    print("\n")
    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")
```

Date. .....

Expt. No. .....

Page No. .....

PROGRAM

```
import pandas as pd
import math
import numpy as np

data = pd.read_csv("3-dataset.csv")
features = [feat for feat in data]
features.remove("answer")

class Node:
    def __init__(self):
        self.children = []
        self.value = ""
        self.isLeaf = False
        self.pred = ""

def entropy(examples):
    pos = 0.0
    neg = 0.0

    for _, row in examples.iterrows():
        if row["answer"] == "yes":
            pos += 1
        else:
            neg += 1
```

Date. .....

Expt. No. .....

Page No. .....

```
    if pos == 0.0 or neg == 0.0:
        return 0.0
    else:
        p = pos / (pos+neg)
        n = neg / (pos+neg)
        return -(p * math.log(p,2) + n * math.log(n,2))

def info_gain(examples, attr):
    uniq = np.unique(examples[attr])
    gain = entropy(examples)
    for u in uniq:
        subdata = examples[examples[attr] == u]
        sub_e = entropy(subdata)
        gain -= (float(len(subdata))/float(len(examples))) * sub_e
    return gain.

def ID3(examples, attrs):
    root = Node()

    max_gain = 0
    max_feat = ""

    for feature in attrs:
        gain = info_gain(examples, feature)
        if gain > max_gain:
            max_gain = gain
            max_feat = feature.
    root.value = max_feat
    uniq = np.unique(examples[max_feat])
```

Date. .....

Expt. No. .....

Page No. .....

```
    for u in uniq:
        subdata = examples[examples[max_feat] == u]
        if entropy(subdata) == 0.0:
            newNode = Node()
            newNode.isLeaf = True
            newNode.value = u
            newNode.pred = np.unique(subdata["answer"])
            root.children.append(newNode)
        else:
            dummyNode = Node()
            dummyNode.value = u
            new_attrs = attrs.copy()
            new_attrs.remove(max_feat)
            child = ID3(subdata, new_attrs)
            dummyNode.children.append(child)
            root.children.append(dummyNode)
    return root.

def printTree(root: Node, depth=0):
    for i in range(depth):
        print("\t", end=" ")
    print(root.value, end=" ")
    if root.isLeaf:
        print("->", root.pred)
    print()
    for child in root.children:
        printTree(child, depth+1)
root = ID3(data, features)
PrintTree(root)
```

Date. .....

Expt. No. .....

Page No. .....

Build an artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.

PROGRAM

```
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 = 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))
```

Date. .....

Expt. No. .....

Page No. .....

```
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)

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 * hiddenlayer

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)
```

Date. .....

Expt. No. .....

Page No. .....

Write a program to implement the naive Bayesian classifier for a sample training data set stored as a .csv file. Compute the accuracy of the classifier, considering few test data sets.

PROGRAM

```
import pandas as pd

PlayTennis = pd.read_csv("4.csv")
print("Given dataset:\n", PlayTennis, "\n")

from sklearn.preprocessing import LabelEncoder
Le = LabelEncoder()

PlayTennis['Outlook'] = Le.fit_transform(PlayTennis['Outlook'])
PlayTennis['temp'] = Le.fit_transform(PlayTennis['temp'])
PlayTennis['humidity'] = Le.fit_transform(PlayTennis['humidity'])
PlayTennis['wind'] = Le.fit_transform(PlayTennis['wind'])
PlayTennis['play'] = Le.fit_transform(PlayTennis['play'])

print("the encoded dataset is:\n", PlayTennis)

x = PlayTennis.drop(['play'], axis=1)
y = PlayTennis['play']

from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.20)
```

Date. .....

Expt. No. .....

Page No.....

```
print("\n x_train: \n", x_train)
print("\n y_train: \n", y_train)
print("\n x_test: \n", x_test)
print("\n y_test: \n", y_test)

classifier = GaussianNB()
classifier.fit(x_train, y_train)
accuracy = accuracy_score(classifier.predict(x_test), y_test)
print("\n Accuracy is: ", accuracy)
```