AI-ML Lab Programs and Outputs

1. Implement A* Search algorithm.

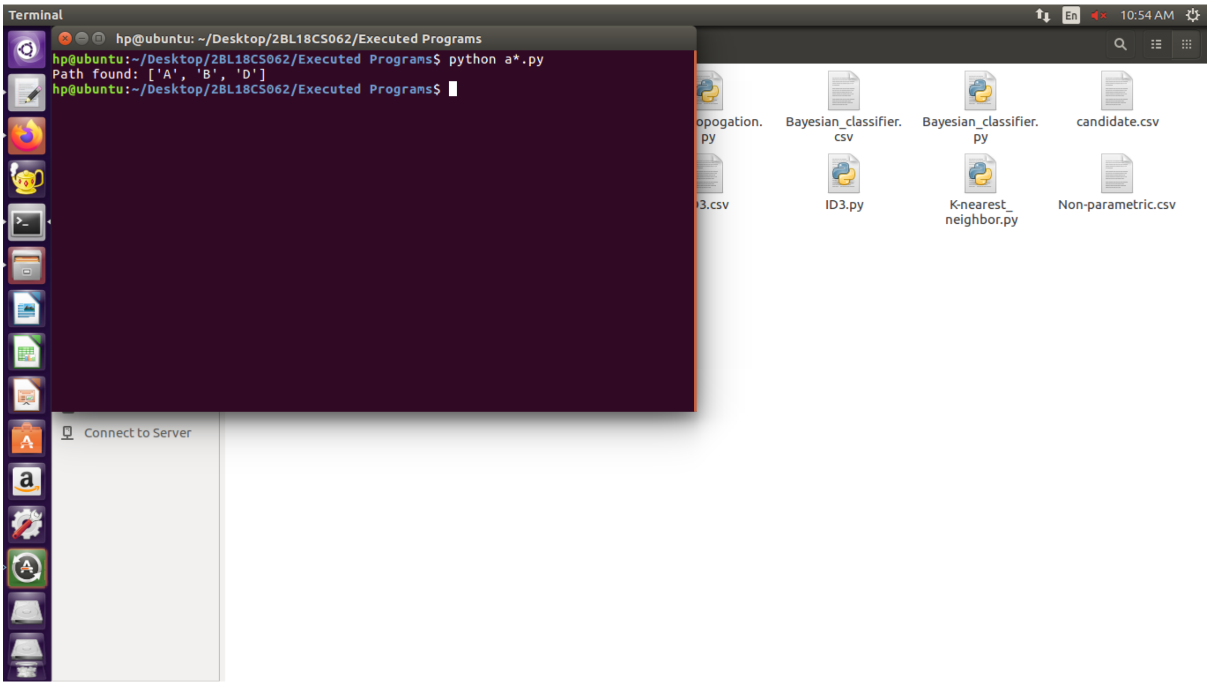
> **Algorithm:**
>
> 01: Create a node containing the goal state node_goal
> 02: Create a node containing the start state node_start
> 03: Put node_start on the open list
> 04: while the OPEN list is not empty
> 05: {
> 06: Get the node off the open list with the lowest f and call it node_current
> 07: if node_current is the same state as node_goal we have found the solution; break from the while loop
> 08: Generate each state node_successor that can come after node_current
> 09: for each node_successor of node_current
> 10: {
> 11: Set the cost of node_successor to be the cost of node_current plus the cost to get to node_successor from node_current
> 12: find node_successor on the OPEN list
> 13: if node_successor is on the OPEN list but the existing one is as good or better than discard this successor and continue
> 14: if node_successor is on the CLOSE list but the existing one is as good or better than discard this successor and continue
> 15: Remove occurrences of node_successor from OPEN and CLOSED
> 16: Set the parent of node_successor to node_current
> 17: Set h to be the estimated distance to node_goal(Using the heuristic function)
> 18: Add node_successor to the OPEN list
> 19: }
> 20: Add node_current to the CLOSED list
> 21: }

Source Code:

```
from collections import deque
class Graph:
    def __init__(self, adjac_lis):
        self.adjac_lis = adjac_lis
    def get_neighbors(self, v):
```

```
        return self.adjac_lis[v]
    def h(self, n):
        H = {
            'A': 1,
            'B': 1,
            'C': 1,
            'D': 1
        }
        return H[n]
    def a_star_algorithm(self, start, stop):
        open_lst = set([start])
        closed_lst = set([])
        poo = {}
        poo[start] = 0
        par = {}
        par[start] = start
        while len(open_lst) > 0:
            n = None
            for v in open_lst:
                if n == None or poo[v] + self.h(v) < poo[n] + self.h(n):
                    n = v;
            if n == None:
                print('Path does not exist!')
                return None
            if n == stop:
                reconst_path = []
                while par[n] != n:
                    reconst_path.append(n)
                    n = par[n]
                reconst_path.append(start)
                reconst_path.reverse()
                print('Path found: {}'.format(reconst_path))
                return reconst_path
            for (m, weight) in self.get_neighbors(n):
                if m not in open_lst and m not in closed_lst:
                    open_lst.add(m)
                    par[m] = n
                    poo[m] = poo[n] + weight
                else:
                    if poo[m] > poo[n] + weight:
                        poo[m] = poo[n] + weight
                        par[m] = n
                        if m in closed_lst:
                            closed_lst.remove(m)
                            open_lst.add(m)
            open_lst.remove(n)
            closed_lst.add(n)

        print('Path does not exist!')
        return None
adjac_lis = {
    'A': [('B', 1), ('C', 3), ('D', 7)],
    'B': [('D', 5)],
    'C': [('D', 12)]
}
graph1 = Graph(adjac_lis)
graph1.a_star_algorithm('A', 'D')
```

Output:



2. Implement AO* Search algorithm.

Algorithm:

OPEN:

It contains the nodes that has been traversed but yet not been marked solvable or unsolvable.

CLOSE:

It contains the nodes that have already been processed.

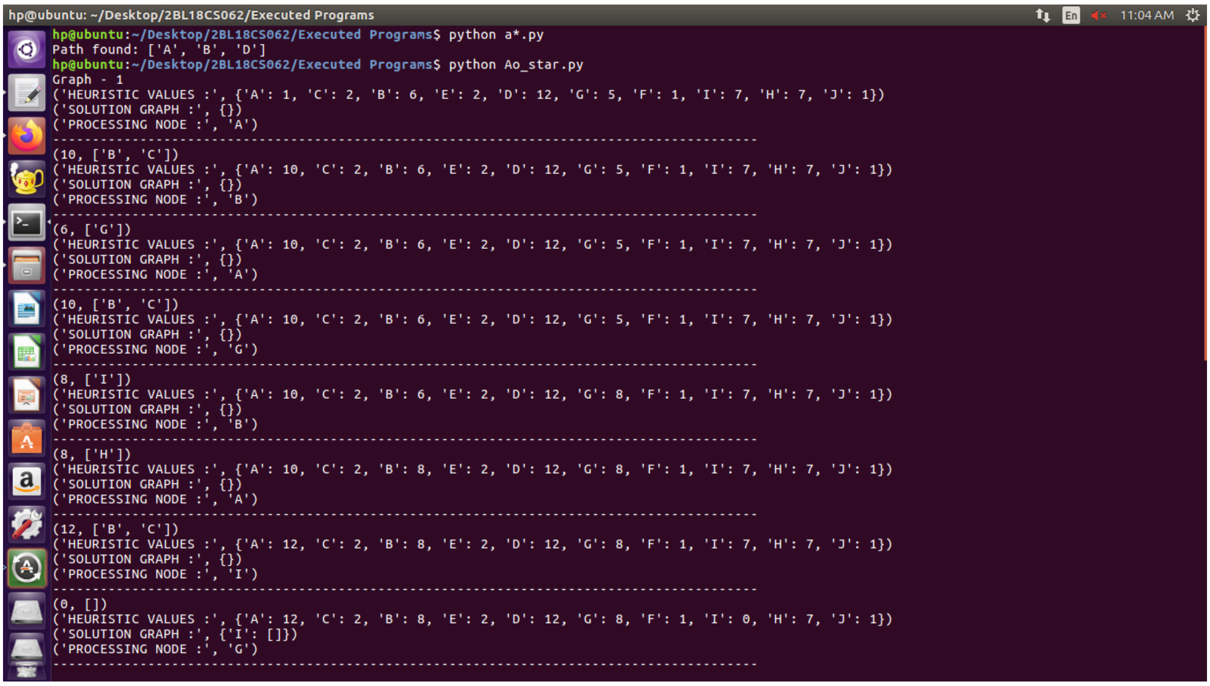
> Step 1: Place the starting node into OPEN.
> Step 2: Compute the most promising solution tree say T0.
> Step 3: Select a node n that is both on OPEN and a member of T0. Remove it from OPEN and place it in CLOSE
> Step 4: If n is the terminal goal node then leveled n as solved and leveled all the ancestors of n as solved. If the starting node is marked as solved then success and exit.
> Step 5: If n is not a solvable node, then mark n as

> unsolvable. If starting node is marked as unsolvable, then return failure and exit.
> Step 6: Expand n. Find all its successors and find their h (n) value, push them into OPEN.
> Step 7: Return to Step 2.
> Step 8: Exit.

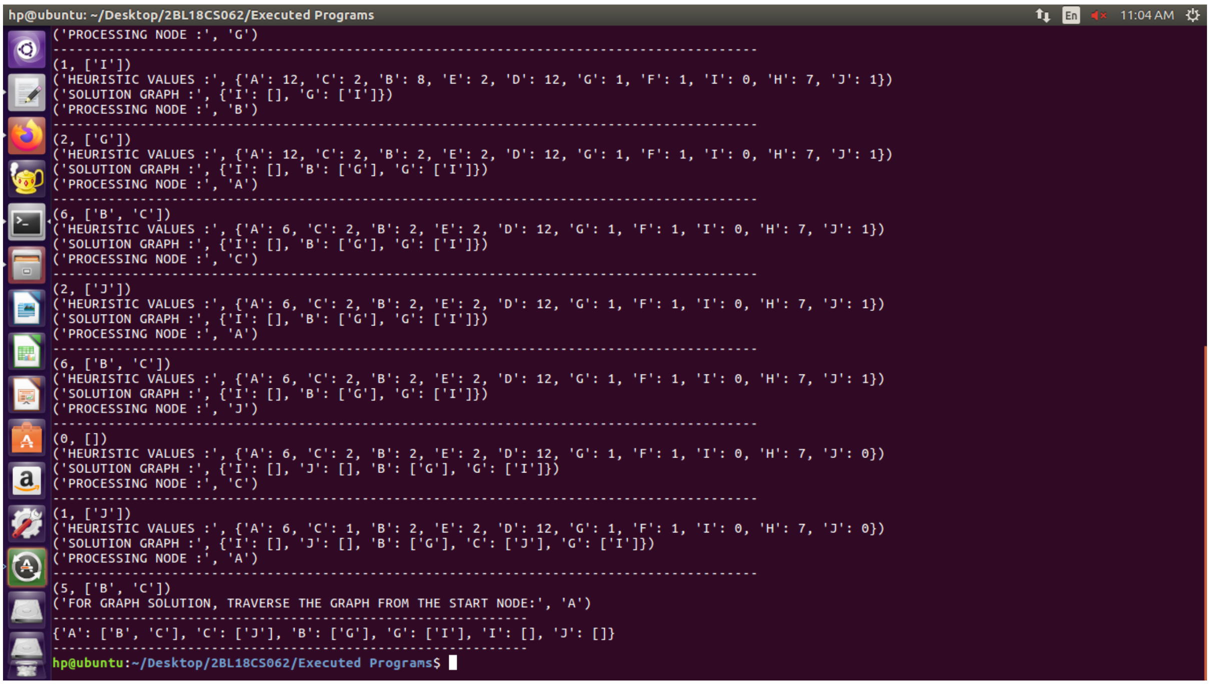
Source Code:

```
class Graph:
    def __init__(self, graph, heuristicNodeList, startNode):
        self.graph = graph
        self.H=heuristicNodeList
        self.start=startNode
        self.parent={}
        self.status={}
        self.solutionGraph={}

    def applyAOStar(self):
        self.aoStar(self.start, False)

    def getNeighbors(self, v):
        return self.graph.get(v,'')

    def getStatus(self,v):
        return self.status.get(v,0)

    def setStatus(self,v, val):
        self.status[v]=val

    def getHeuristicNodeValue(self, n):
        return self.H.get(n,0)

    def setHeuristicNodeValue(self, n, value):
        self.H[n]=value

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

    def computeMinimumCostChildNodes(self, v):
        minimumCost=0
        costToChildNodeListDict={}
        costToChildNodeListDict[minimumCost]=[]
        flag=True
        for nodeInfoTupleList in self.getNeighbors(v):
            cost=0
            nodeList=[]
            for c, weight in nodeInfoTupleList:
                cost=cost+self.getHeuristicNodeValue(c)+weight
                nodeList.append(c)
            if flag==True:
                minimumCost=cost
                costToChildNodeListDict[minimumCost]=nodeList
                flag=False
            else:
                if minimumCost>cost:
```

```
                    minimumCost=cost
                    costToChildNodeListDict[minimumCost]=nodeList
        return minimumCost, costToChildNodeListDict[minimumCost]

    def aoStar(self, v, backTracking):
        print("HEURISTIC VALUES :", self.H)
        print("SOLUTION GRAPH :", self.solutionGraph)
        print("PROCESSING NODE :", v)
        print("-----------------------------------------------------------------------------------------")
        if self.getStatus(v) >= 0:
            minimumCost, childNodeList = self.computeMinimumCostChildNodes(v)
            print(minimumCost, childNodeList)
            self.setHeuristicNodeValue(v, minimumCost)
            self.setStatus(v,len(childNodeList))
            solved=True
            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)
print ("Graph - 1")
h1 = {'A': 1, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1}
graph1 = {
    'A': [[('B', 1), ('C', 1)], [('D', 1)]],
    'B': [[('G', 1)], [('H', 1)]],
    'C': [[('J', 1)]],
    'D': [[('E', 1), ('F', 1)]],
    'G': [[('I', 1)]]
}

G1= Graph(graph1, h1, 'A')
G1.applyAOStar()
G1.printSolution()
```



```
hp@ubuntu: ~/Desktop/2BL18CS062/Executed Programs
hp@ubuntu:~/Desktop/2BL18CS062/Executed Programs$ python a*.py
Path found: ['A', 'B', 'D']
hp@ubuntu:~/Desktop/2BL18CS062/Executed Programs$ python Ao_star.py
Graph - 1
('HEURISTIC VALUES :', {'A': 1, 'C': 2, 'B': 6, 'E': 2, 'D': 12, 'G': 5, 'F': 1, 'I': 7, 'H': 7, 'J': 1})
('SOLUTION GRAPH :', {})
('PROCESSING NODE :', 'A')
------------------------------------------------------------------------------------------
(10, ['B', 'C'])
('HEURISTIC VALUES :', {'A': 10, 'C': 2, 'B': 6, 'E': 2, 'D': 12, 'G': 5, 'F': 1, 'I': 7, 'H': 7, 'J': 1})
('SOLUTION GRAPH :', {})
('PROCESSING NODE :', 'B')
------------------------------------------------------------------------------------------
(6, ['G'])
('HEURISTIC VALUES :', {'A': 10, 'C': 2, 'B': 6, 'E': 2, 'D': 12, 'G': 5, 'F': 1, 'I': 7, 'H': 7, 'J': 1})
('SOLUTION GRAPH :', {})
('PROCESSING NODE :', 'A')
------------------------------------------------------------------------------------------
(10, ['B', 'C'])
('HEURISTIC VALUES :', {'A': 10, 'C': 2, 'B': 6, 'E': 2, 'D': 12, 'G': 5, 'F': 1, 'I': 7, 'H': 7, 'J': 1})
('SOLUTION GRAPH :', {})
('PROCESSING NODE :', 'G')
------------------------------------------------------------------------------------------
(8, ['I'])
('HEURISTIC VALUES :', {'A': 10, 'C': 2, 'B': 6, 'E': 2, 'D': 12, 'G': 8, 'F': 1, 'I': 7, 'H': 7, 'J': 1})
('SOLUTION GRAPH :', {})
('PROCESSING NODE :', 'B')
------------------------------------------------------------------------------------------
(8, ['H'])
('HEURISTIC VALUES :', {'A': 10, 'C': 2, 'B': 8, 'E': 2, 'D': 12, 'G': 8, 'F': 1, 'I': 7, 'H': 7, 'J': 1})
('SOLUTION GRAPH :', {})
('PROCESSING NODE :', 'A')
------------------------------------------------------------------------------------------
(12, ['B', 'C'])
('HEURISTIC VALUES :', {'A': 12, 'C': 2, 'B': 8, 'E': 2, 'D': 12, 'G': 8, 'F': 1, 'I': 7, 'H': 7, 'J': 1})
('SOLUTION GRAPH :', {})
('PROCESSING NODE :', 'I')
------------------------------------------------------------------------------------------
(0, [])
('HEURISTIC VALUES :', {'A': 12, 'C': 2, 'B': 8, 'E': 2, 'D': 12, 'G': 8, 'F': 1, 'I': 0, 'H': 7, 'J': 1})
('SOLUTION GRAPH :', {'I': []})
('PROCESSING NODE :', 'G')
------------------------------------------------------------------------------------------
```



```
hp@ubuntu: ~/Desktop/2BL18CS062/Executed Programs
('PROCESSING NODE :', 'G')
------------------------------------------------------------------------------------------
(1, ['I'])
('HEURISTIC VALUES :', {'A': 12, 'C': 2, 'B': 8, 'E': 2, 'D': 12, 'G': 1, 'F': 1, 'I': 0, 'H': 7, 'J': 1})
('SOLUTION GRAPH :', {'I': [], 'G': ['I']})
('PROCESSING NODE :', 'B')
------------------------------------------------------------------------------------------
(2, ['G'])
('HEURISTIC VALUES :', {'A': 12, 'C': 2, 'B': 2, 'E': 2, 'D': 12, 'G': 1, 'F': 1, 'I': 0, 'H': 7, 'J': 1})
('SOLUTION GRAPH :', {'I': [], 'B': ['G'], 'G': ['I']})
('PROCESSING NODE :', 'A')
------------------------------------------------------------------------------------------
(6, ['B', 'C'])
('HEURISTIC VALUES :', {'A': 6, 'C': 2, 'B': 2, 'E': 2, 'D': 12, 'G': 1, 'F': 1, 'I': 0, 'H': 7, 'J': 1})
('SOLUTION GRAPH :', {'I': [], 'B': ['G'], 'G': ['I']})
('PROCESSING NODE :', 'C')
------------------------------------------------------------------------------------------
(2, ['J'])
('HEURISTIC VALUES :', {'A': 6, 'C': 2, 'B': 2, 'E': 2, 'D': 12, 'G': 1, 'F': 1, 'I': 0, 'H': 7, 'J': 1})
('SOLUTION GRAPH :', {'I': [], 'B': ['G'], 'G': ['I']})
('PROCESSING NODE :', 'A')
------------------------------------------------------------------------------------------
(6, ['B', 'C'])
('HEURISTIC VALUES :', {'A': 6, 'C': 2, 'B': 2, 'E': 2, 'D': 12, 'G': 1, 'F': 1, 'I': 0, 'H': 7, 'J': 1})
('SOLUTION GRAPH :', {'I': [], 'B': ['G'], 'G': ['I']})
('PROCESSING NODE :', 'J')
------------------------------------------------------------------------------------------
(0, [])
('HEURISTIC VALUES :', {'A': 6, 'C': 2, 'B': 2, 'E': 2, 'D': 12, 'G': 1, 'F': 1, 'I': 0, 'H': 7, 'J': 0})
('SOLUTION GRAPH :', {'I': [], 'J': [], 'B': ['G'], 'G': ['I']})
('PROCESSING NODE :', 'C')
------------------------------------------------------------------------------------------
(1, ['J'])
('HEURISTIC VALUES :', {'A': 6, 'C': 1, 'B': 2, 'E': 2, 'D': 12, 'G': 1, 'F': 1, 'I': 0, 'H': 7, 'J': 0})
('SOLUTION GRAPH :', {'I': [], 'J': [], 'B': ['G'], 'C': ['J'], 'G': ['I']})
('PROCESSING NODE :', 'A')
------------------------------------------------------------------------------------------
(5, ['B', 'C'])
('FOR GRAPH SOLUTION, TRAVERSE THE GRAPH FROM THE START NODE:', 'A')
----------------------------------------------------------------
{'A': ['B', 'C'], 'C': ['J'], 'B': ['G'], 'G': ['I'], 'I': [], 'J': []}
----------------------------------------------------------------
hp@ubuntu:~/Desktop/2BL18CS062/Executed Programs$
```

3. 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.

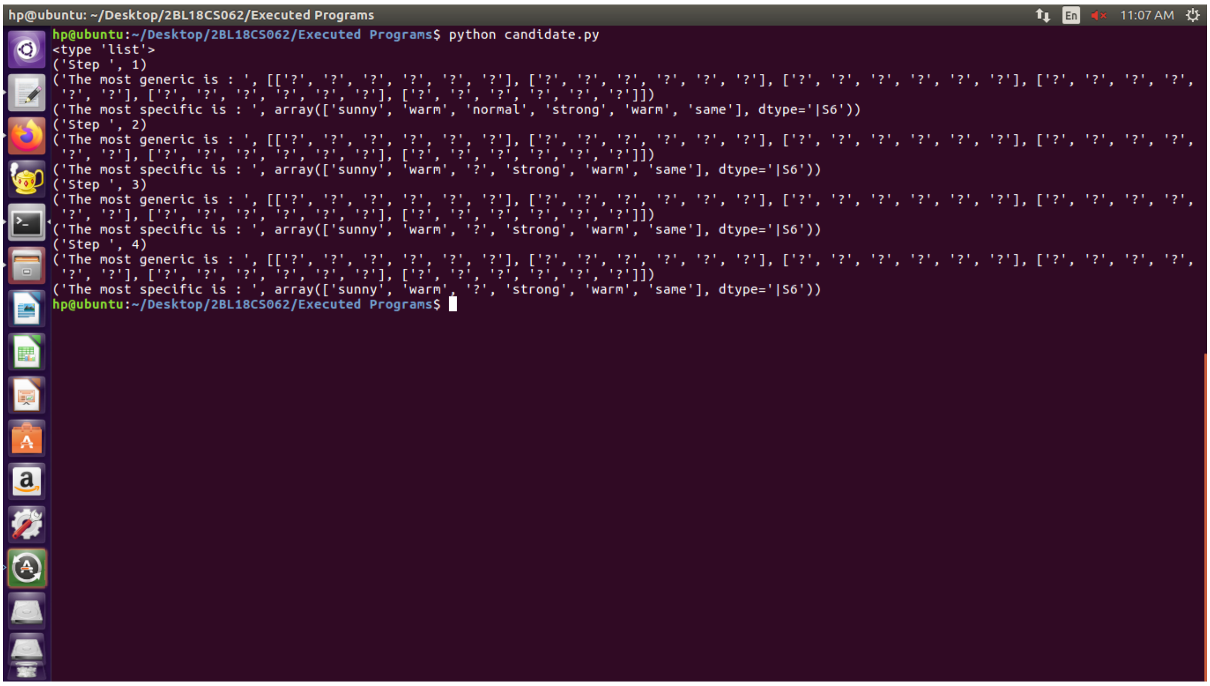
> ### Algorithm:

> For each training example d, do:
>
> If d is positive example
>
> Remove from G any hypothesis h inconsistent with d
>
> For each hypothesis s in S not consistent with d:
>
> Remove s from S
>
> Add to S all minimal generalizations of s consistent with d and having a generalization in G
>
> Remove from S any hypothesis with a more specific h in S
>
> If d is negative example
>
> Remove from S any hypothesis h inconsistent with d
>
> For each hypothesis g in G not consistent with d:
>
> Remove g from G
>
> Add to G all minimal specializations of g consistent with d and having a specialization in S
>
> Remove from G any hypothesis having a more general hypothesis in G

Source code:

```
import csv
import numpy as np
with open('candidate.csv','r') as f:
    reads=csv.reader(f)
    tmp_lst=np.array(list(reads))
concept=np.array(tmp_lst[:,:-1])
target=np.array(tmp_lst[:,-1])
for i in range(len(target)):
    if(target[i]=='yes'):
        specific_h=concept[i]
        break
h=[]
generic_h=[['?' for i in range (len(specific_h))]for i in range (len(specific_h))]
print(type(generic_h))

for i in range(len(target)):
    if(target[i]=='yes'):
        for j in range (len(specific_h)):
            if(specific_h[j]!=concept[i][j]):
                specific_h[j]='?'
                generic_h[j][j]='?'
    else:
        for j in range(len(specific_h)):
            if(specific_h[j]!=concept[i][j]):
                generic_h[j][j]=specific_h[j]
            else:
                generic_h[j][j]='?'
    print("Step ",i+1)
    print("The most generic is : ",generic_h)
    print("The most specific is : ",specific_h)
```

Output:



4. Write a program to demonstrate the working of the decision tree-based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

> **Algorithm:**
>
> ID3(Examples, Target_attribute, Attributes)
>
> Examples are the training examples.
> Target_attribute is the attribute whose value is to be predicted by the tree.
> Attributes is a list of other attributes that may be tested by the learned decision tree.
> Returns a decision tree that correctly classifies the given Examples.
>
> Create a Root node for the tree
> If all Examples are positive, Return the single-node tree Root, with label = +
> If all Examples are negative, Return the single-node tree Root, with label = -
> If Attributes is empty, Return the single-node tree Root,
> with label = most common value of Target_attribute in Examples
>
> Otherwise Begin
> A ← the attribute from Attributes that best* classifies Examples

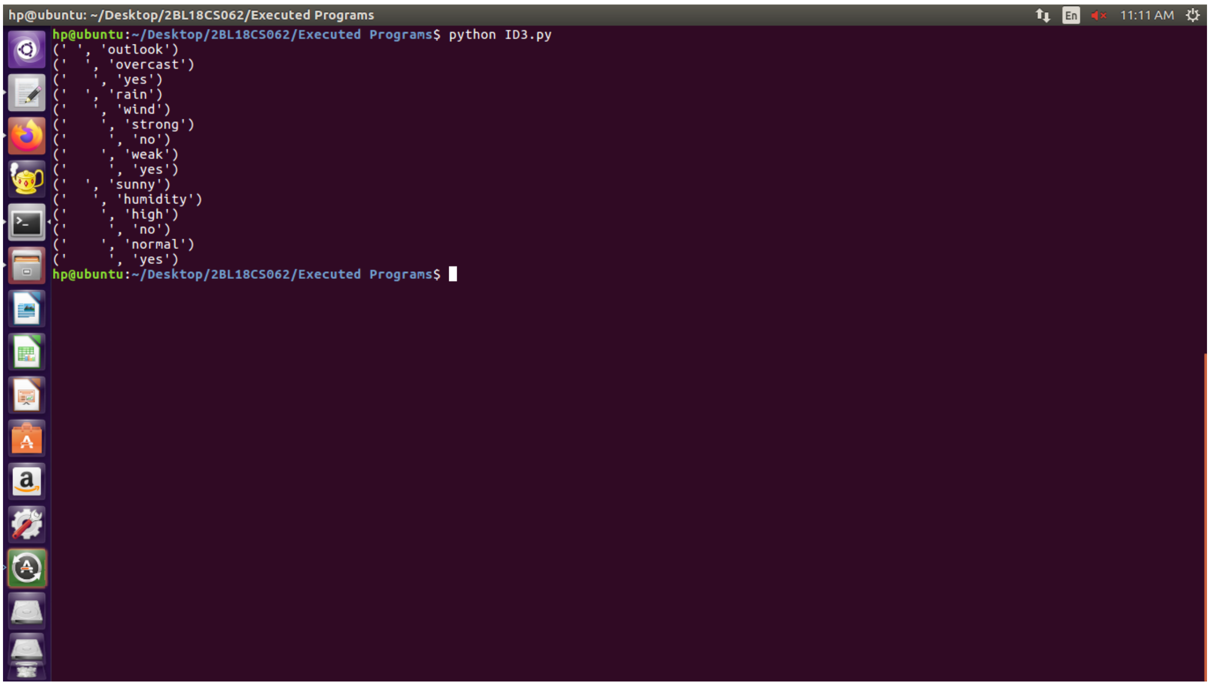
> The decision attribute for Root ← A
> For each possible value, vi, of A,
> Add a new tree branch below Root, corresponding to the test A = vi
> Let Examples vi, be the subset of Examples that have value vi for A
> If Examples vi , is empty
> Then below this new branch add a leaf node with
> label = most common value of Target_attribute in Examples
> Else
> below this new branch add the subtree
> ID3(Examples vi, Target_attribute, Attributes – {A}))
> End
> Return Root

Source code:

```
import numpy as np
import math
import csv
class Node:
  def __init__ (self,attribute):
    self.attribute=attribute
    self.children=[]
    self.answer=" "
def read_data(filename):
  with open(filename,'r') as csvfile:
    datareader=csv.reader(csvfile,delimiter=',')
    headers=next(datareader)
    metadata=[]
    traindata=[]
    for name in headers:
      metadata.append(name)
    for row in datareader:
      traindata.append(row)
  return(metadata,traindata)
def subtables(data,col,delete):
  dict={}
  items=np.unique(data[:,col])
  count=np.zeros((items.shape[0],1),dtype=np.int32)
  for x in range(items.shape[0]):
    for y in range(data.shape[0]):
      if data[y,col]==items[x]:
        count[x]+=1
  for x in range(items.shape[0]):
    dict[items[x]]=np.empty((int (count[x]),data.shape[1]),dtype="|S32")
    pos=0
    for y in range(data.shape[0]):
      if data[y,col]==items[x]:
        dict[items[x]][pos]=data[y]
        pos+=1
    if delete:
      dict[items[x]]=np.delete(dict[items[x]],col,1)
  return items,dict
def entropy(S):
  items=np.unique(S)
  if items.size==1:
    return 0
```

```
  counts = np.zeros((items.shape[0],1))
  sums = 0
  for x in range(items.shape[0]):
    counts[x] = sum(S ==items[x])/(S.size*1.0)
  for count in counts:
    sums +=-1*count*math.log(count,2)
  return sums
def gain_ratio(data,col):
  items,dict=subtables(data,col,delete=False)
  total_size=data.shape[0]
  entropies=np.zeros((items.shape[0],1))
  intrinsic=np.zeros((items.shape[0],1))
  for x in range(items.shape[0]):
    ratio=dict[items[x]].shape[0]/(total_size*1.0)
    entropies[x]=ratio*entropy(dict[items[x]][:,-1])
    intrinsic[x]=ratio*math.log(ratio,2)
    total_entropy=entropy(data[:,-1])
    iv=-1*sum(intrinsic)
    for x in range(entropies.shape[0]):
      total_entropy-=entropies[x]
    return total_entropy/iv
def create_node(data,metadata):
  if(np.unique(data[:,-1])).shape[0]==1:
    node = Node(" ")
    node.answer = np.unique(data[:,-1])[0]
    return node
  gains = np.zeros((data.shape[1]-1,1))
  for col in range(data.shape[1]-1):
    gains[col]=gain_ratio(data,col)
  split=np.argmax(gains)
  node=Node(metadata[split])
  metadata=np.delete(metadata,split,0)
  items,dict=subtables(data,split,delete=True)
  for x in range(items.shape[0]):
    child = create_node(dict[items[x]],metadata)
    node.children.append((items[x],child))
  return node
def empty(size):
  S = " "
  for x in range(size):
    S+=" "
  return S
def print_tree(node,level):
  if node.answer!=" ":
    print(empty(level),node.answer)
    return
  print(empty(level),node.attribute)
  for value,n in node.children:
    print(empty(level+1),value)
    print_tree(n,level+2)
metadata,traindata=read_data("ID3.csv")
data=np.array(traindata)
node=create_node(data,metadata)
print_tree(node,0)
```

Output:



---

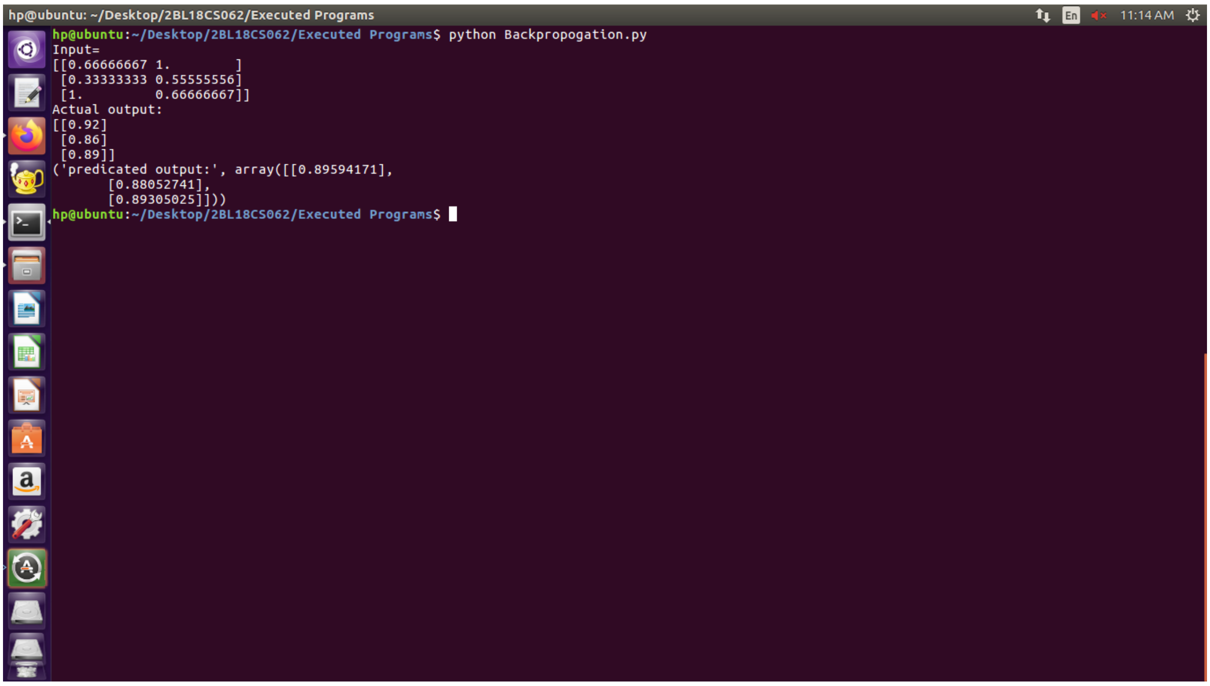
5. Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.

> **Algorithm:**
>
> 1: Run the network forward with your input data to get the network output
> 2: For each output node compute
>
> $$\delta_k = \mathcal{O}_k (1 - \mathcal{O}_k) (\mathcal{O}_k - t_k)$$
>
> 3: For each hidden node calculate
>
> $$\delta_j = \mathcal{O}_j (1 - \mathcal{O}_j) \sum_{k \in K} \delta_k W_{jk}$$
>
> 4: Update the weights and biases as follows
> Given
>
> $$\Delta W = \quad -\eta \delta_\ell \mathcal{O}_{\ell-1}$$
> $$\Delta \theta = \eta \delta_\ell$$
>
> Apply

$$W + \Delta W > W$$
$$\theta + \Delta\theta > \theta$$

Source code:

```
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=0.25
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)
  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
  wh+=X.T.dot(d_hiddenlayer)*lr
print("Input=\n"+str(X))
print("Actual output:\n"+str(y))
print("predicated output:",output)
```



---

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

> **Naïve Bayesian Classifier:**
>
> **Bayes' Theorem is stated as:**
>
> $$P(h \mid D) = \frac{P(D \mid h)P(h)}{P(D)}$$
>
> **Steps to implement Naïve Bayesian Classifier:**
>
> > Step 1: Separate By Class.
> > Step 2: Summarize Dataset.
> > Step 3: Summarize Data By Class.
> > Step 4: Gaussian Probability Density Function.
> > Step 5: Class Probabilities.

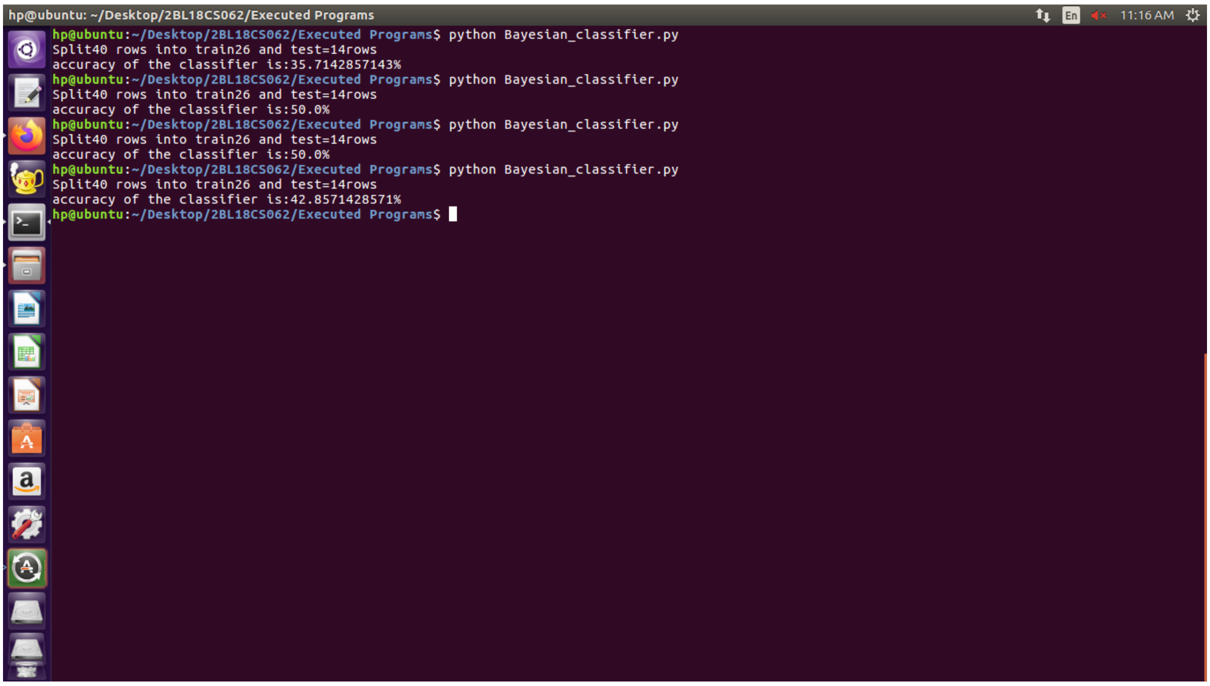
Source code:

```
import csv
import random
```

```
import math
def loadCsv(filename):
  lines = csv.reader(open(filename, "r"));
  dataset = list(lines)
  for i in range(len(dataset)):
    dataset[i] = [float(x) for x in dataset[i]]
  return dataset
def splitDataset(dataset, splitRatio):
  trainSize = int(len(dataset) * splitRatio);
  trainSet = []
  copy = list(dataset);
  while len(trainSet) < trainSize:
    index = random.randrange(len(copy));
    trainSet.append(copy.pop(index))
  return [trainSet, copy]
def separateByClass(dataset):
  separated = {}
  for i in range(len(dataset)):
    vector = dataset[i]
    if (vector[-1] not in separated):
      separated[vector[-1]] = []
    separated[vector[-1]].append(vector)
  return separated
def mean(numbers):
  return sum(numbers)/float(len(numbers))
def stdev(numbers):
  avg = mean(numbers)
  variance = sum([pow(x-avg,2) for x in numbers])/float(len(numbers)-1)
  return math.sqrt(variance)
def summarize(dataset):
  summaries = [(mean(attribute), stdev(attribute)) for attribute in zip(*dataset)];
  del summaries[-1]
  return summaries
def summarizeByClass(dataset):
  separated=separateByClass(dataset)
  summaries={}
  for classValue, instances in separated.items():
    summaries[classValue]=summarize(instances)
  return summaries
def calculateProbability(x,mean,stdev):
  exponent = math.exp(-(math.pow(x-mean,2)/(2*math.pow(stdev,2))))
  return (1 / (math.sqrt(2*math.pi) * stdev)) * exponent
def calculateClassProbabilities(summaries, inputVector):
  probabilities = {}
  for classValue, classSummaries in summaries.items():
    probabilities[classValue] = 1
    for i in range(len(classSummaries)):
      mean, stdev = classSummaries[i]
      x = inputVector[i]
      probabilities[classValue] *= calculateProbability(x, mean,stdev);
    return probabilities
def predict(summaries, inputVector):
  probabilities = calculateClassProbabilities(summaries, inputVector)
  bestLabel, bestProb = None, -1
  for classValue, probability in probabilities.items():
    if bestLabel is None or probability > bestProb:
      bestProb = probability
      bestLabel=classValue
  return bestLabel
def getPredictions(summaries,testSet):
  predictions = []
  for i in range(len(testSet)):
    result = predict(summaries, testSet[i])
    predictions.append(result)
  return predictions
def getAccuracy(testSet, predictions):
  correct = 0
  for i in range(len(testSet)):
    if testSet[i][-1] == predictions[i]:
```

```
        correct += 1
    return (correct/float(len(testSet))) * 100.0
def main():
    filename="Bayesian_classifier.csv"
    splitRatio=0.67
    dataset=loadCsv(filename)
    trainingSet,testSet=splitDataset(dataset,splitRatio)
    print('Split{0} rows into train{1} and test={2}rows'.format(len(dataset),len(trainingSet),len(testSe
t)))
    summaries = summarizeByClass(trainingSet);
    predictions=getPredictions(summaries,testSet)
    accuracy=getAccuracy(testSet,predictions)
    print('accuracy of the classifier is:{0}%'.format(accuracy))
main()
```

Output:



7. Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using the 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.

> ### Algorithm:
>
> ### K-Means:
>
> K-Means clustering algorithm produces a Minimum Variance Estimate (MVE) of the state of the identified clusters in the data.

$$J=\sum_{k \in K} \sum_{i} z_{k}^{i}\left|x_{i}-\mu_{k}\right|^{2}$$

$$\left(\mu_{k}, z_{k}^{i}\right)=\operatorname{argmin}_{\left(z_{k}^{i}, \mu_{k}\right)} J$$

EM Algorithm:

Step 01: Initial guess is made for the model's parameters and a probability distribution is created. This is sometimes called **E-Step** for the expected distribution.
Step 02: Newly observed data is fed into a model.
Step 03: The probability distributed the **E-step** is drawn to include the new data which is sometimes called **M-step**.
Step 04: **Step-02** through **Step-04** are repeated until **S** with normal distribution.
That is:

$$E\left[z_{ji}\right]=\frac{p\left(x=x_{i} \mid \mu=\mu j\right)}{\sum^{2} p\left(x=x_{i} \mid \mu=\mu(n)\right)}$$

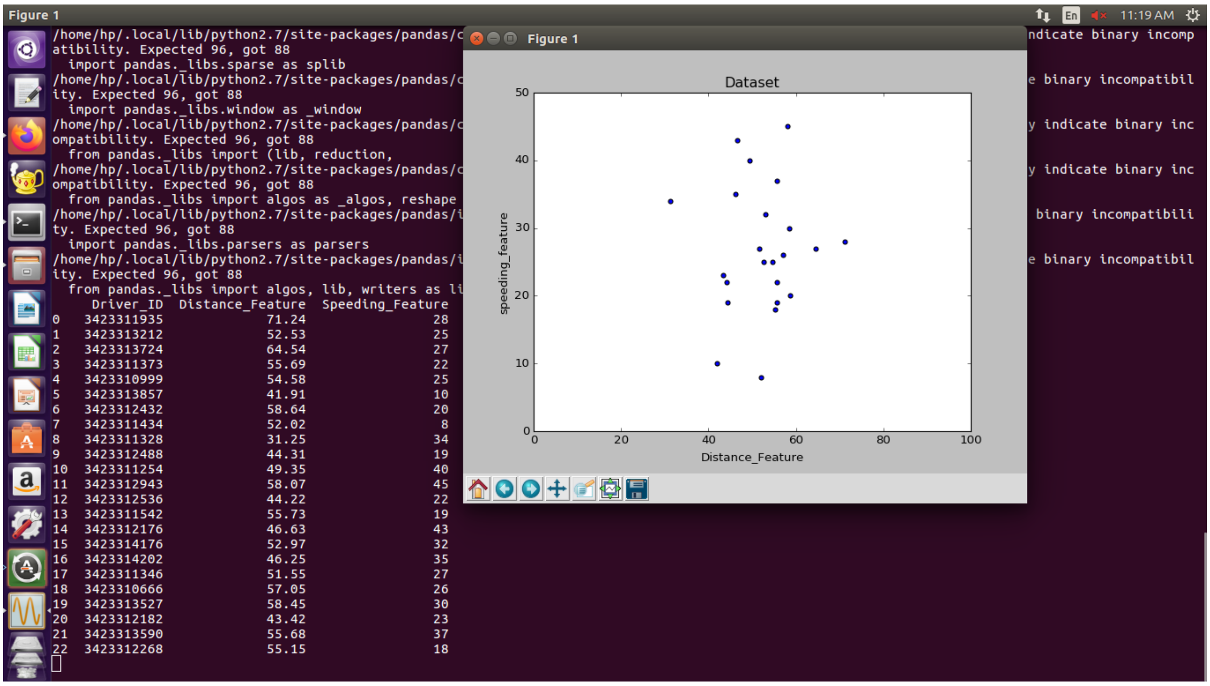
$$\forall \quad \left(\mu_{j} \leftarrow \frac{\sum_{i=1}^{M} E\left[z_{ij}\right] x_{i} .}{\sum_{i=1}^{M} E\left[z_{n}\right] .}\right.$$

Source code:

```
from sklearn.cluster import KMeans

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
data=pd.read_csv("EM_Algorithm.csv")
df1=pd.DataFrame(data)
print(df1)
f1 = df1['Distance_Feature'].values
f2 = df1['Speeding_Feature'].values
X=np.matrix(list(zip(f1,f2)))
plt.plot()
plt.xlim([0, 100])
plt.ylim([0, 50])
plt.title('Dataset')
plt.ylabel('speeding_feature')
plt.xlabel('Distance_Feature')
plt.scatter(f1,f2)
plt.show()
plt.plot()
colors = ['b', 'g', 'r']
markers = ['o', 'v', 's']
kmeans_model = KMeans(n_clusters=3).fit(X)
plt.plot()
```

```
for i, l in enumerate(kmeans_model.labels_):
  plt.plot(f1[i], f2[i], color=colors[l], marker=markers[l],ls='None')
  plt.xlim([0, 100])
  plt.ylim([0, 50])
plt.show()
```

Output:



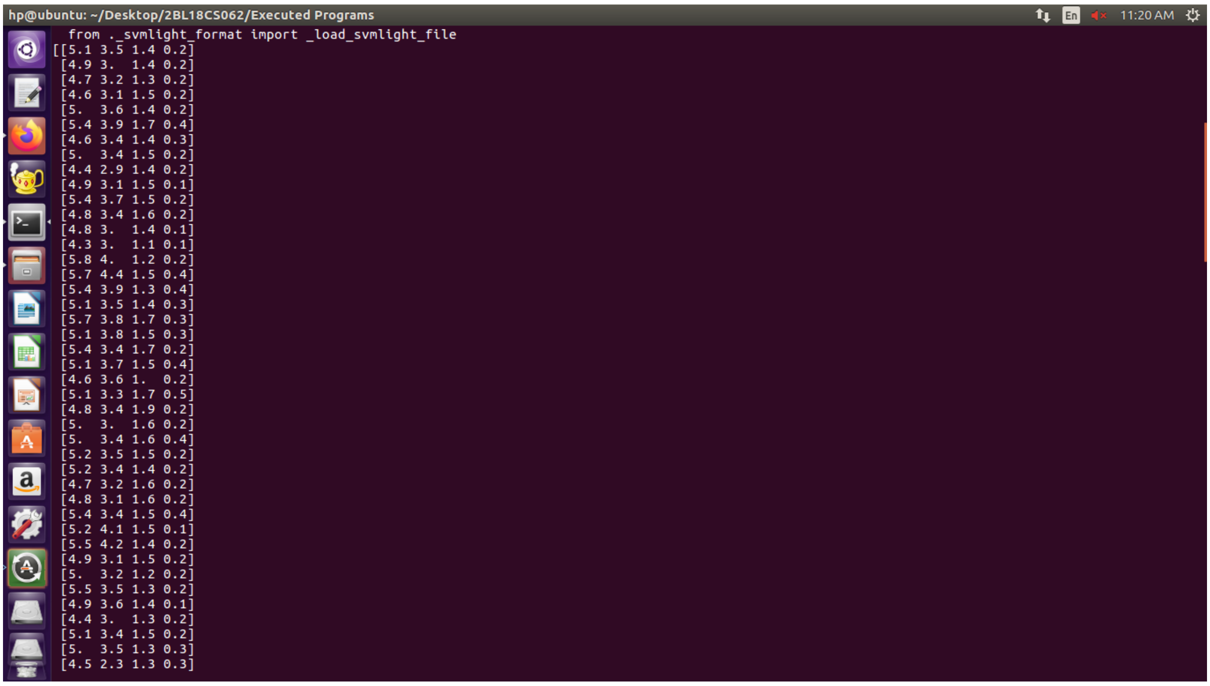
8. Write a program to implement the k-Nearest Neighbor algorithm to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem.

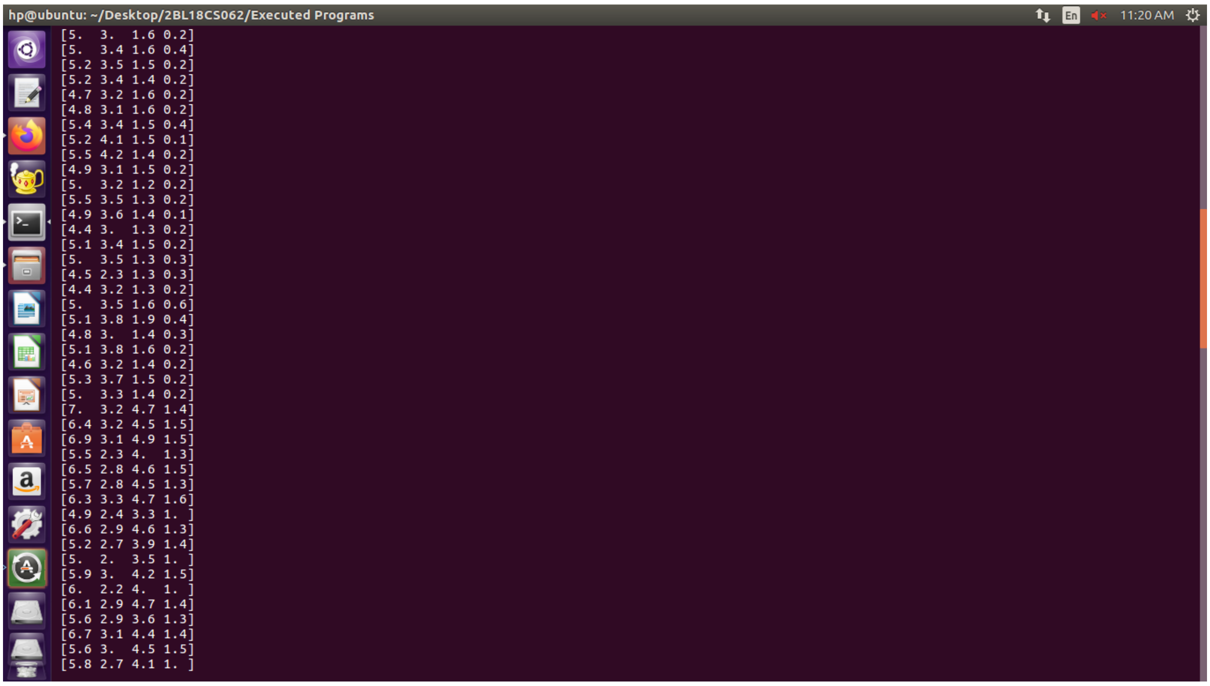
> ### Algorithm:
>
> Step-1: Select the number K of the neighbors
> Step-2: Calculate the Euclidean distance of K number of neighbors
> Step-3: Take the K nearest neighbors as per the calculated Euclidean distance.
> Step-4: Among these k neighbors, count the number of the data points in each category.
> Step-5: Assign the new data points to that category for which the number of the neighbor is maximum.
> Step-6: The model is ready.

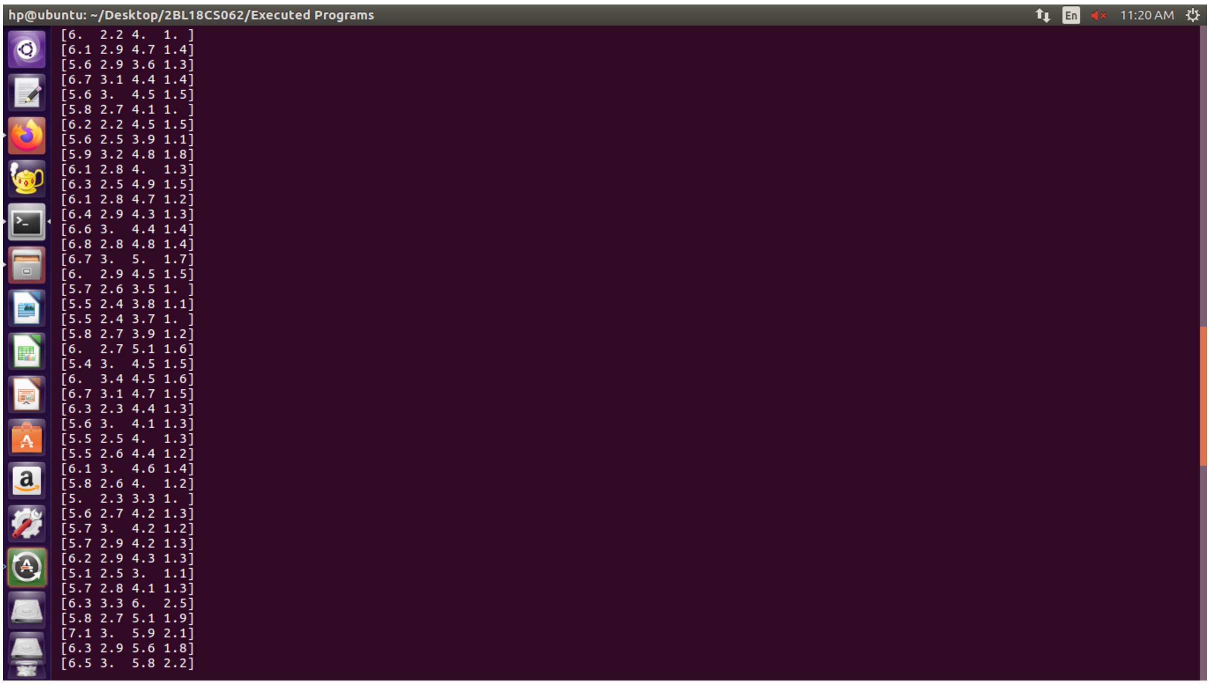
Source code:

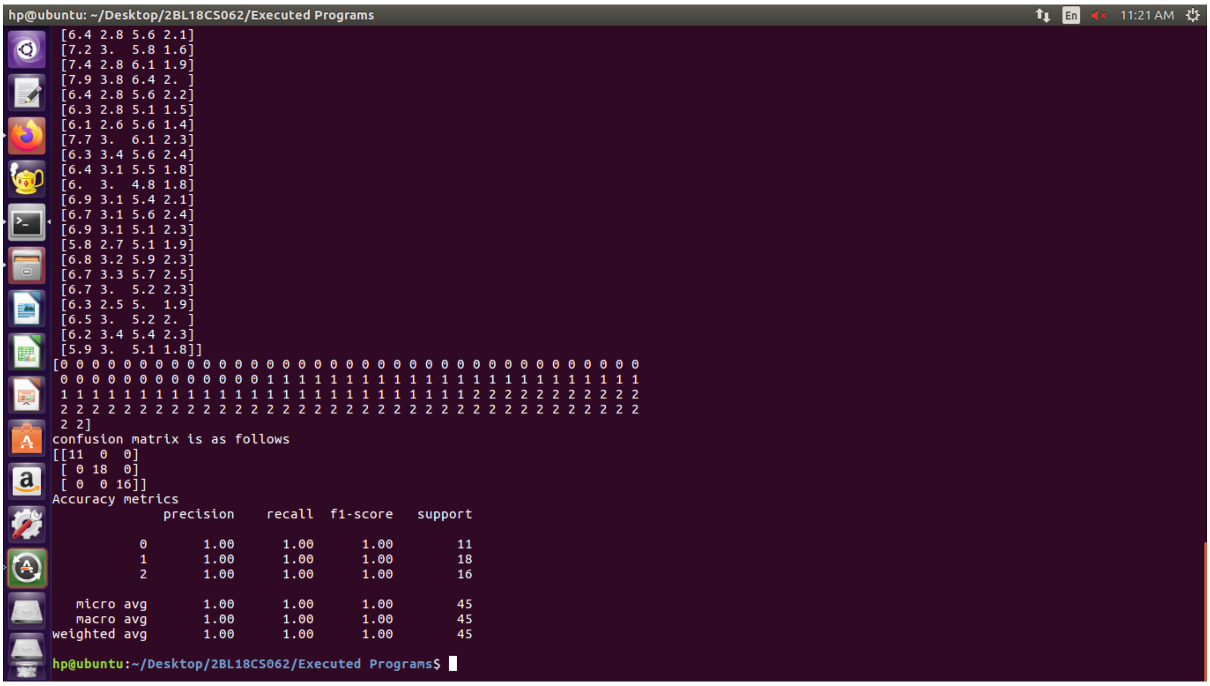
```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix

from sklearn import datasets
iris=datasets.load_iris()
iris_data=iris.data
iris_labels=iris.target
print(iris_data)
print(iris_labels)
x_train, x_test, y_train, y_test=train_test_split(iris_data,iris_labels,test_size=0.30)

classifier=KNeighborsClassifier(n_neighbors=5)
classifier.fit(x_train,y_train)
y_pred=classifier.predict(x_test)
print('confusion matrix is as follows')
print(confusion_matrix(y_test,y_pred))
print('Accuracy metrics')
print(classification_report(y_test,y_pred))
```

Output:




hp@ubuntu: ~/Desktop/2BL18CS062/Executed Programs
11:20 AM
[5. 3. 1.6 0.2]
[5. 3.4 1.6 0.4]
[5.2 3.5 1.5 0.2]
[5.2 3.4 1.4 0.2]
[4.7 3.2 1.6 0.2]
[4.8 3.1 1.6 0.2]
[5.4 3.4 1.5 0.4]
[5.2 4.1 1.5 0.1]
[5.5 4.2 1.4 0.2]
[4.9 3.1 1.5 0.2]
[5. 3.2 1.2 0.2]
[5.5 3.5 1.3 0.2]
[4.9 3.6 1.4 0.1]
[4.4 3. 1.3 0.2]
[5.1 3.4 1.5 0.2]
[5. 3.5 1.3 0.3]
[4.5 2.3 1.3 0.3]
[4.4 3.2 1.3 0.2]
[5. 3.5 1.6 0.6]
[5.1 3.8 1.9 0.4]
[4.8 3. 1.4 0.3]
[5.1 3.8 1.6 0.2]
[4.6 3.2 1.4 0.2]
[5.3 3.7 1.5 0.2]
[5. 3.3 1.4 0.2]
[7. 3.2 4.7 1.4]
[6.4 3.2 4.5 1.5]
[6.9 3.1 4.9 1.5]
[5.5 2.3 4. 1.3]
[6.5 2.8 4.6 1.5]
[5.7 2.8 4.5 1.3]
[6.3 3.3 4.7 1.6]
[4.9 2.4 3.3 1.]
[6.6 2.9 4.6 1.3]
[5.2 2.7 3.9 1.4]
[5. 2. 3.5 1.]
[5.9 3. 4.2 1.5]
[6. 2.2 4. 1.]
[6.1 2.9 4.7 1.4]
[5.6 2.9 3.6 1.3]
[6.7 3.1 4.4 1.4]
[5.6 3. 4.5 1.5]
[5.8 2.7 4.1 1.]


hp@ubuntu: ~/Desktop/2BL18CS062/Executed Programs
11:20 AM
[6. 2.2 4. 1.]
[6.1 2.9 4.7 1.4]
[5.6 2.9 3.6 1.3]
[6.7 3.1 4.4 1.4]
[5.6 3. 4.5 1.5]
[5.8 2.7 4.1 1.]
[6.2 2.2 4.5 1.5]
[5.6 2.5 3.9 1.1]
[5.9 3.2 4.8 1.8]
[6.1 2.8 4. 1.3]
[6.3 2.5 4.9 1.5]
[6.1 2.8 4.7 1.2]
[6.4 2.9 4.3 1.3]
[6.6 3. 4.4 1.4]
[6.8 2.8 4.8 1.4]
[6.7 3. 5. 1.7]
[6. 2.9 4.5 1.5]
[5.7 2.6 3.5 1.]
[5.5 2.4 3.8 1.1]
[5.5 2.4 3.7 1.]
[5.8 2.7 3.9 1.2]
[6. 2.7 5.1 1.6]
[5.4 3. 4.5 1.5]
[6. 3.4 4.5 1.6]
[6.7 3.1 4.7 1.5]
[6.3 2.3 4.4 1.3]
[5.6 3. 4.1 1.3]
[5.5 2.5 4. 1.3]
[5.5 2.6 4.4 1.2]
[6.1 3. 4.6 1.4]
[5.8 2.6 4. 1.2]
[5. 2.3 3.3 1.]
[5.6 2.7 4.2 1.3]
[5.7 3. 4.2 1.2]
[5.7 2.9 4.2 1.3]
[6.2 2.9 4.3 1.3]
[5.1 2.5 3. 1.1]
[5.7 2.8 4.1 1.3]
[6.3 3.3 6. 2.5]
[5.8 2.7 5.1 1.9]
[7.1 3. 5.9 2.1]
[6.3 2.9 5.6 1.8]
[6.5 3. 5.8 2.2]




```
 [6.4 2.8 5.6 2.1]
 [7.2 3.  5.8 1.6]
 [7.4 2.8 6.1 1.9]
 [7.9 3.8 6.4 2. ]
 [6.4 2.8 5.6 2.2]
 [6.3 2.8 5.1 1.5]
 [6.1 2.6 5.6 1.4]
 [7.7 3.  6.1 2.3]
 [6.3 3.4 5.6 2.4]
 [6.4 3.1 5.5 1.8]
 [6.  3.  4.8 1.8]
 [6.9 3.1 5.4 2.1]
 [6.7 3.1 5.6 2.4]
 [6.9 3.1 5.1 2.3]
 [5.8 2.7 5.1 1.9]
 [6.8 3.2 5.9 2.3]
 [6.7 3.3 5.7 2.5]
 [6.7 3.  5.2 2.3]
 [6.3 2.5 5.  1.9]
 [6.5 3.  5.2 2. ]
 [6.2 3.4 5.4 2.3]
 [5.9 3.  5.1 1.8]]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
confusion matrix is as follows
[[11  0  0]
 [ 0 18  0]
 [ 0  0 16]]
Accuracy metrics
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        11
           1       1.00      1.00      1.00        18
           2       1.00      1.00      1.00        16

   micro avg       1.00      1.00      1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45

hp@ubuntu:~/Desktop/2BL18CS062/Executed Programs$
```

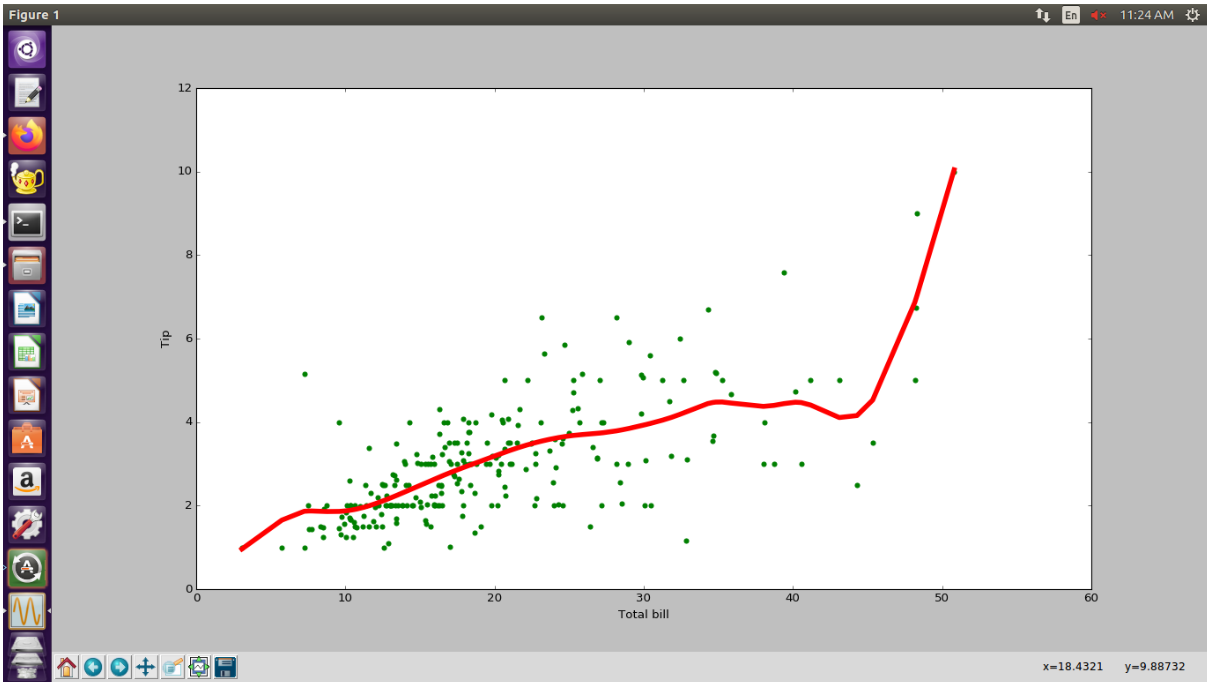
9. Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select the appropriate data set for your experiment and draw graphs.

> ### Algorithm:
>
> 01: Read the Given data Sample to X and the curve (linear or non linear) to Y
>
> 02: Set the value for Smoothening parameter or Free parameter say τ
>
> 03: Set the bias /Point of interest set x0 which is a subset of X
>
> 04: Determine the weight matrix using :
>
> $$w(x, x_o) = e^{-\frac{(x-x_o)^2}{2\tau^2}}$$
>
> 05: Determine the value of model term parameter β using:
>
> $$\hat{\beta}(x_o) = \left(X^T W X\right)^{-1} X^T W y$$
>
> 06: Prediction = x0*β

Source code:

```
from numpy import *
import operator
from os import listdir
```

```
import matplotlib
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np1
import numpy.linalg as np
from scipy.stats.stats import pearsonr

def kernel(point,xmat, k):
  m,n = np1.shape(xmat)
  weights = np1.mat(np1.eye((m)))
  for j in range(m):
    diff = point - X[j]
    weights[j,j] = np1.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 = np1.shape(xmat)
  ypred = np1.zeros(m)
  for i in range(m):
    ypred[i] = xmat[i]*localWeight(xmat[i],xmat,ymat,k)
  return ypred

# load data points
data = pd.read_csv('Non-Parametric.csv')
bill = np1.array(data.total_bill)
tip = np1.array(data.tip)

#preparing and add 1 in bill
mbill = np1.mat(bill)
mtip = np1.mat(tip)
m= np1.shape(mbill)[1]
one = np1.mat(np1.ones(m))
X= np1.hstack((one.T,mbill.T))

#set k here
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=5)
plt.xlabel('Total bill')
plt.ylabel('Tip')
plt.show();
```



The below file contains all the python programs and the respective .csv datasets and also the output printout pdf for 7th and 9th program.

https://drive.google.com/drive/folders/1dtFdFmEc7G23VRtscFxDZMZt5tlazQkX?usp=sharing