

B.L.D.E.A’s

Vachana Pitamaha Dr. P. G. Halakatti College of Engineering and Technology,

Vijayapur-586103.

**Department of Computer Science and Engineering**

**Lab Manual**

**“ARTIFICIAL INTELLIGENCE**

**&**

**MACHINE LEARNING LABORATORY”**

**(18CSL76)**

**VII Semester B.E.**

**2018-19**

**Institute Vision**

To emerge as a widely acknowledged centre in technical education and research to cater the need of society with a futuristic outlook.

**Institute Mission**

* To enrich students with the essence of science and engineering knowledge, professional ethics and social values.
* To instill creativity and research temperament to reach the greater heights of professional success.

**Department Vision**

“To provide valuable human resources to the society through Quality Technical Education and Research with moral values”

**Department Mission**

“To educate the students in Computer Science and Engineering by imparting Quality Technical Education and Research to meet the needs of profession and society with ethical values”

**Programme Educational Objectives (PEOs)**

1. A Graduate will be a successful IT professional and function effectively in multidisciplinary domains.
2. A Graduate will have the perspective of lifelong learning for continuous improvement of knowledge in Computer Science & Engineering, higher studies, and research.
3. A Graduate will be able to respond to local, national and global issues by imparting his/her knowledge of Computer Science & Engineering in Educational, Government, Financial and Private sectors.
4. A Graduate will be able to function effectively as an individual, as a team member and as a team leader with highest professional and ethical standards.

**Programme Outcomes (POs)**

**A graduate of the Computer Science and Engineering Program will demonstrate:**

**PO1: Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

**PO2: Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences

**PO3: Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

**PO4: Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

**PO5: Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations

**PO6: The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

**PO7: Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

**PO8: Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

**PO9: Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

**PO10: Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

**PO11: Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one’s own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

**PO12: Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

**Programme Specific Outcomes (PSOs)**

**Graduates will be able to**

1. **Computational skills:** Apply the knowledge of Mathematics and Computational Science to solve societal problems in various domains.
2. **Programming Skills:** Design, Analyze and Implement various algorithms using broad range of programming languages.
3. **Product Development Skills:** Utilize Hardware and Software tools to develop solutions to IT problems.

**ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING LABORATORY (18CSL76)**

**Course Outcomes (COs)**

The student should be able to design and develop Python Code for the

|  |  |
| --- | --- |
| CO476.1 | Best-first search to find the least cost path from a start state to a goal state. |
| CO476.2 | Candidate elimination and ID3 algorithm incrementally builds the version space given a hypothesis space H and a set E |
| CO476.3 | Backpropagation and Bayesian classifier algorithms using .csv files. |
| CO476.4 | Unsupervised machine learning algorithm and classification algorithms. |
| CO476.5 | Regression algorithm. |

**ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING LABORATORY (18CSL76)**

**Correlation of Course Outcomes (CO) with Program Outcomes (PO)**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **COs** |  |  |  |  |  |  |  |  |  |  |  |  |
| CO476.1 | 1 | 1 | 3 | 3 | - | - | - | - | - | - | - | - |
| CO476.2 | 1 | 1 | 3 | 3 | - | - | - | - | - | - | - | - |
| CO476.3 | 2 | 1 | 3 | 3 | - | - | - | - | - | - | - | - |
| CO476.4 | 2 | 1 | 3 | 3 | - | - | - | - | - | - | - | - |
| CO476.5 | 2 | 1 | 3 | 3 | - | - | - | - | - | - | - | - |

**1. Slight (Low) 2. Moderate (Medium) 3. Substantial (High)**

**Correlation of Course Outcomes (CO) with Program Specific Outcomes (PSO)**

|  |  |  |  |
| --- | --- | --- | --- |
| **COs** | **Program Specific Outcomes** | | |
|  |  |  |
| CO476.1 | **3** | **3** | **-** |
| CO476.2 | **3** | **3** | **-** |
| CO476.3 | **3** | **3** | **-** |
| CO476.4 | **3** | **3** | **-** |
| CO476.5 | **3** | **3** | **-** |

**1. Slight (Low) 2. Moderate (Medium) 3. Substantial (High)**

**Mapping of Laboratory Experiments with Course Outcomes**

|  |  |  |
| --- | --- | --- |
| Expt.  No. | Title | COs |
| 1 | Implement A\* Search algorithm. | 1 |
| 2 | Implement AO\* Search algorithm. | 1 |
| 3 | Candidate-Elimination algorithm | 2 |
| 4 | ID3 algorithm | 2 |
| 5 | Backpropagation algorithm | 3 |
| 6 | EM algorithm | 3 |
| 7 | k-Means algorithm | 4 |
| 8 | k-Nearest Neighbour algorithm | 4 |
| 9 | Regression algorithm | 5 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING LABORATORY**  **(Effective from the academic year 2018 -2019) SEMESTER – VII** | | | | |
| **Course Code** | | **18CSL76** | **CIE Marks** | 40 |
| **Number of Contact Hours/Week** | | 0:0:2 | **SEE Marks** | 60 |
| **Total Number of Lab Contact Hours** | | 36 | **Exam Hours** | 03 |
| **Credits – 2** | | | | |
| **Course Learning Objectives:** This course (18CSL76) will enable students to: | | | | |
| * Implement and evaluate AI and ML algorithms in and Python programming language. | | | | |
| **Descriptions (if any):** | | | | |
| **Installation procedure of the required software must be demonstrated, carried out in groups**  **and documented in the journal.** | | | | |
| **Programs List:** | | | | |
| 1. | Implement A\* Search algorithm. | | | |
| 2. | Implement AO\* Search algorithm. | | | |
| 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. | | | |
| 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 toclassify a new sample. | | | |
| 5. | Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the  same using appropriate data sets. | | | |
| 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. | | | |
| 7. | Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using 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. | | | |
| 8. | Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print  both correct and wrong predictions. Java/Python ML library classes can be used for this problem. | | | |
| 9. | Implement the non-parametric Locally Weighted Regressionalgorithm in order to fit data points.  Select appropriate data set for your experiment and draw graphs | | | |
| **Laboratory Outcomes**: The student should be able to: | | | | |
| * Implement and demonstrate AI and ML algorithms. * Evaluate different algorithms. | | | | |
| **Conduct of Practical Examination:** | | | | |
| * Experiment distribution   + For laboratories having only one part: Students are allowed to pick one experiment from the lot with equal opportunity.   + For laboratories having PART A and PART B: Students are allowed to pick one experiment from PART A and one experiment from PART B, with equal opportunity. * Change of experiment is allowed only once and marks allotted for procedure to be made zero of the changed part only. * Marks Distribution *(Courseed to change in accoradance with university regulations)*  1. For laboratories having only one part – Procedure + Execution + Viva-Voce: 15+70+15 =   100 Marks   1. For laboratories having PART A and PART B    1. Part A – Procedure + Execution + Viva = 6 + 28 + 6 = 40 Marks    2. Part B – Procedure + Execution + Viva = 9 + 42 + 9 = 60 Marks | | | | |

**Experiment -1 : A\* Search algorithm**

**A\* Search algorithm is one of the best and popular technique used in path-finding and graph traversals.**

### **Algorithm of A\* search:**

**Step1:** Place the starting node in the OPEN list.

**Step 2:** Check if the OPEN list is empty or not, if the list is empty then return failure and stops.

**Step 3:** Select the node from the OPEN list which has the smallest value of evaluation function (g+h), if node n is goal node then return success and stop, otherwise

**Step 4:** Expand node n and generate all of its successors, and put n into the closed list. For each successor n', check whether n' is already in the OPEN or CLOSED list, if not then compute evaluation function for n' and place into Open list.

**Step 5:** Else if node n' is already in OPEN and CLOSED, then it should be attached to the back pointer which reflects the lowest g(n') value.

**Step 6:** Return to **Step 2**.

**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)],

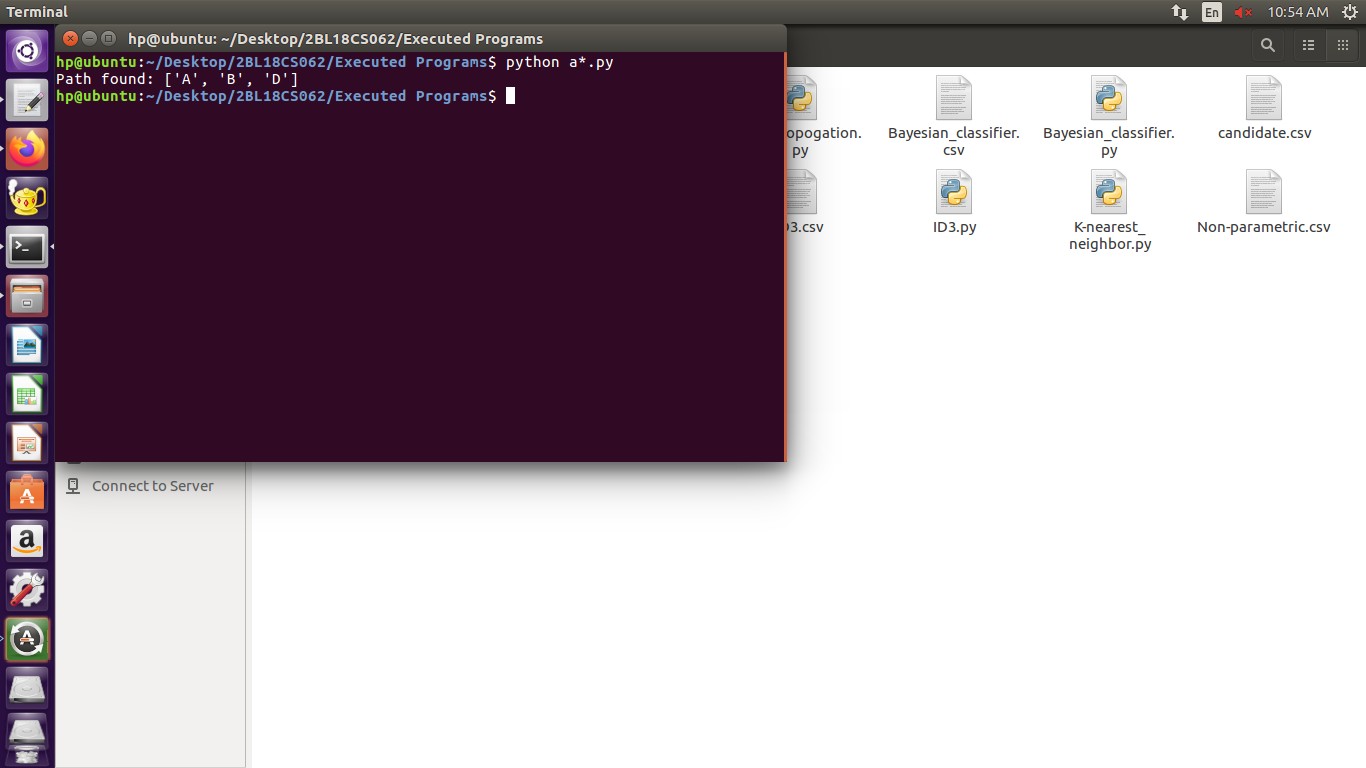
'C': [('D', 12)]

}

graph1 = Graph(adjac\_lis)

graph1.a\_star\_algorithm('A', 'D')

OUTPUT



**Experiment 2: AO\* Search algorithm.**

AO\* algorithm is a best first search algorithm. AO\* algorithm uses the concept of AND-OR graphs to decompose any complex problem given into smaller set of problems which are further solved. AND-OR graphs are specialized graphs that are used in problems that can be broken down into sub problems where AND side of the graph represent a set of task that need to be done to achieve the main goal , whereas the or side of the graph represent the different ways of performing task to achieve the same main goal.

The AO\* algorithm works on the formula given below :

f(n) = g(n) + h(n)

where,

g(n): The actual cost of traversal from initial state to the current state.

h(n): The estimated cost of traversal from the current state to the goal state.

f(n): The actual cost of traversal from the initial state to the goal state.

*AO*\* Algorithm

1. Initialise the graph to start node
2. Traverse the graph following the current path accumulating nodes that have not yet been expanded or solved
3. Pick any of these nodes and expand it and if it has no successors call this value *FUTILITY* otherwise calculate only *f*' for each of the successors.
4. If *f*' is 0 then mark the node as *SOLVED*
5. Change the value of *f*' for the newly created node to reflect its successors by back propagation.
6. Wherever possible use the most promising routes and if a node is marked as *SOLVED* then mark the parent node as *SOLVED*.
7. If starting node is *SOLVED* or value greater than *FUTILITY*, stop, else repeat from 2.

**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)]]

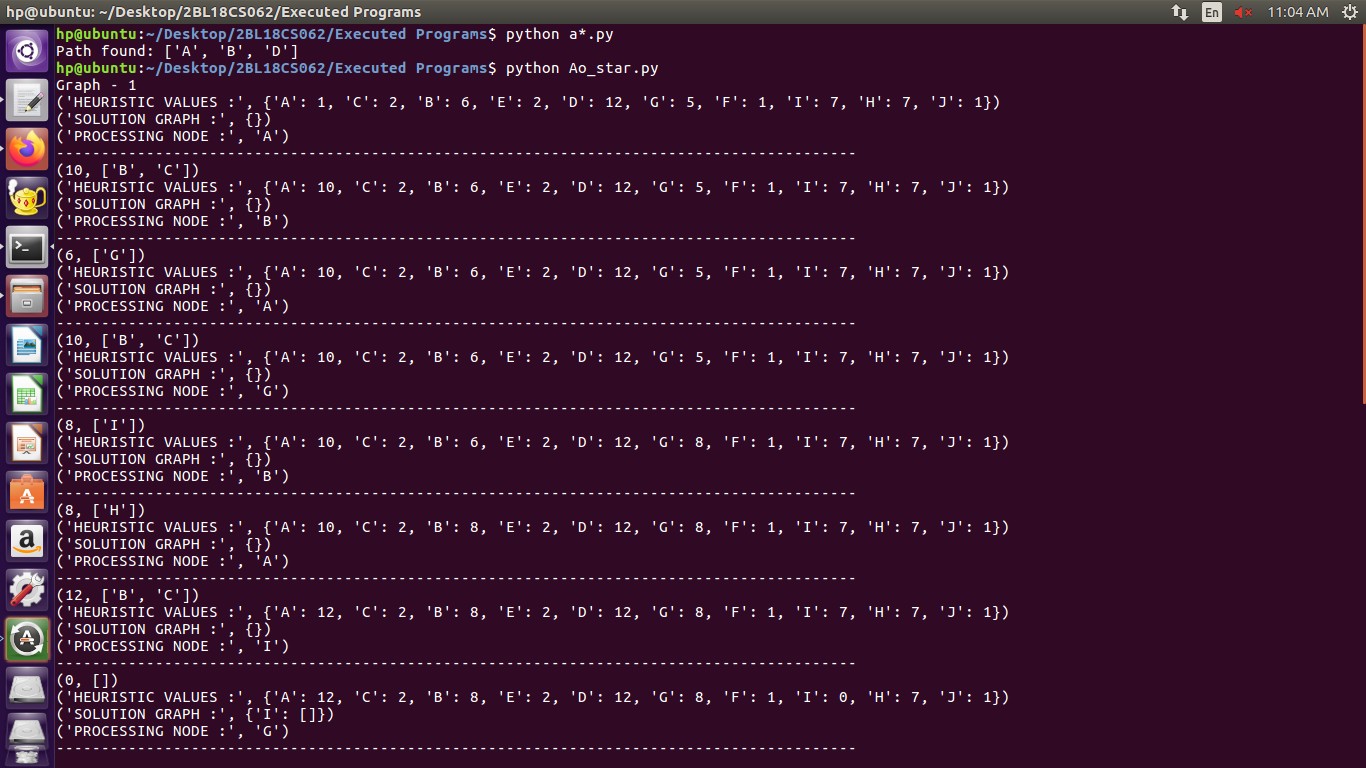
}

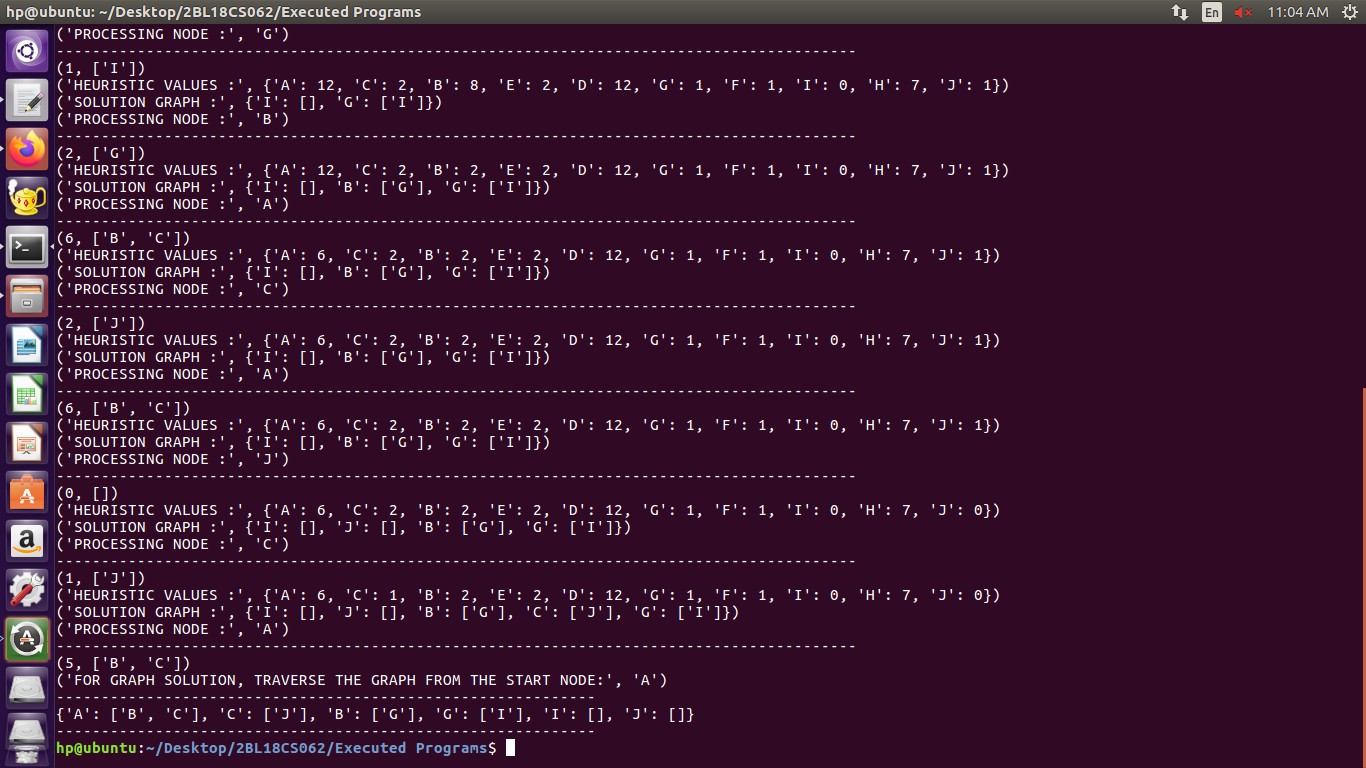
G1= Graph(graph1, h1, 'A')

G1.applyAOStar()

G1.printSolution()

**OUTPUT**





**Experiment 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**.

The candidate elimination algorithm incrementally builds the version space given a hypothesis space H and a set E of examples. The examples are added one by one; each example possibly shrinks the version space by removing the hypotheses that are inconsistent with the example. The candidate elimination algorithm does this by updating the general and specific boundary for each new example.

* You can consider this as an extended form of the Find-S algorithm.
* Consider both positive and negative examples.
* Actually, positive examples are used here as the Find-S algorithm (Basically they are generalizing from the specification).
* While the negative example is specified in the generalizing form.

**Candidate Elimination Algorithm:**

**Step1:** Load Data set

**Step2:** Initialize General Hypothesis and Specific Hypothesis.

**Step3:** For each training example

**Step4:** If example is positive example

if attribute\_value == hypothesis\_value:

Do nothing

else:

replace attribute value with '?' (Basically generalizing it)

**Step5:** If example is Negative example

Make generalize hypothesis more specific.

**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:

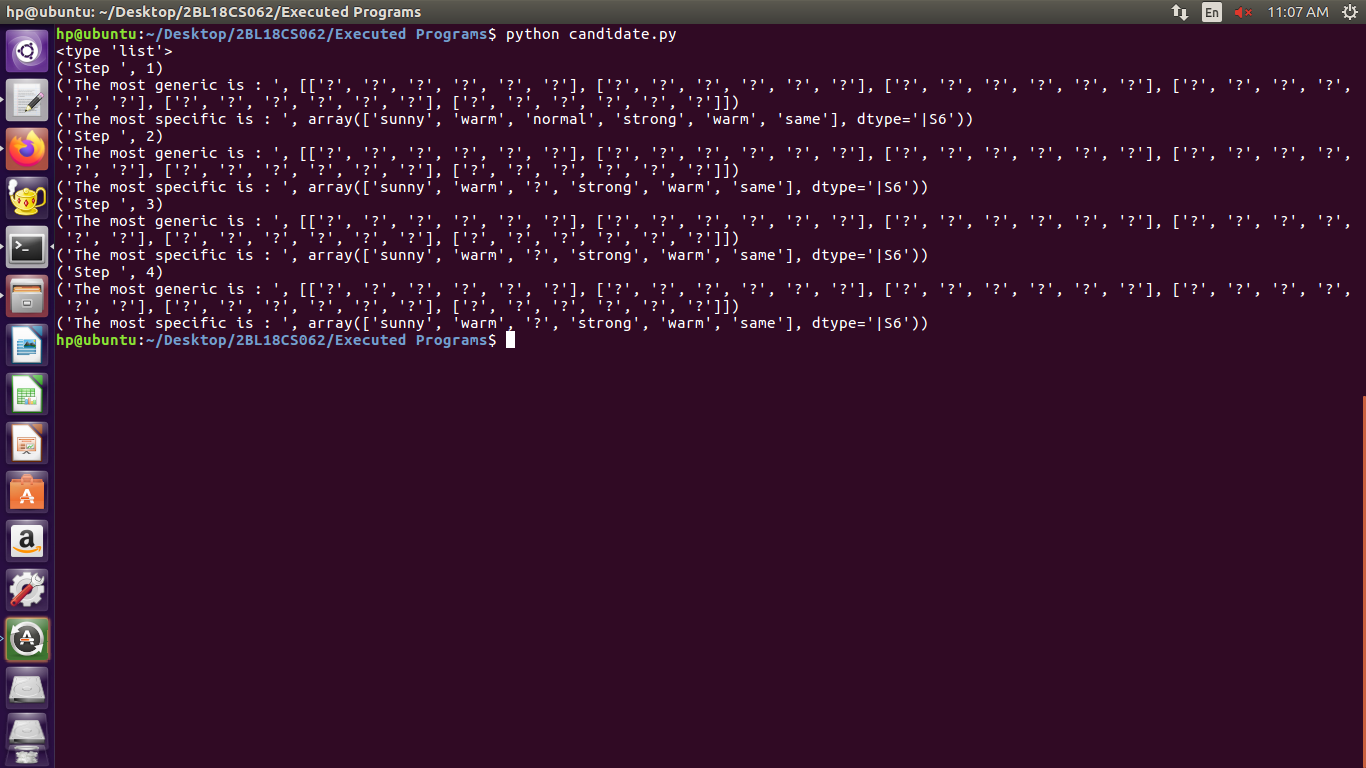
generic\_h[j][j]='?'

print("Step ",i+1)

print("The most generic is : ",generic\_h)

print("The most specific is : ",specific\_h)

**OUTPUT**



**Experiment 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**

**ID3 stands for Iterative Dichotomiser 3**

Algorithm used to generate a decision tree.

ID3 is a precursor to the C4.5

**Decision Tree**

• Classifies data using the attributes

• Tree consists of decision nodes and decision leafs.

• Nodes can have two or more branches which represents the value for the attribute tested.

• Leafs nodes produces a homogeneous result.

**The Process**

• Take all unused attributes and calculates their entropies.

• Chooses attribute that has the lowest entropy is minimum or when information gain is maximum

• Makes a node containing that attribute

**Procedure of ID3 Algorithm**

Calculate entropy for the dataset.

For each node

Calculate entropy for all its categorical values.

Calculate information gain for the node.

Find the node with highest information gain at a particular level.

Repeat steps from 1 to 3 till we reach the leaf node and have created our decision tree.

**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)

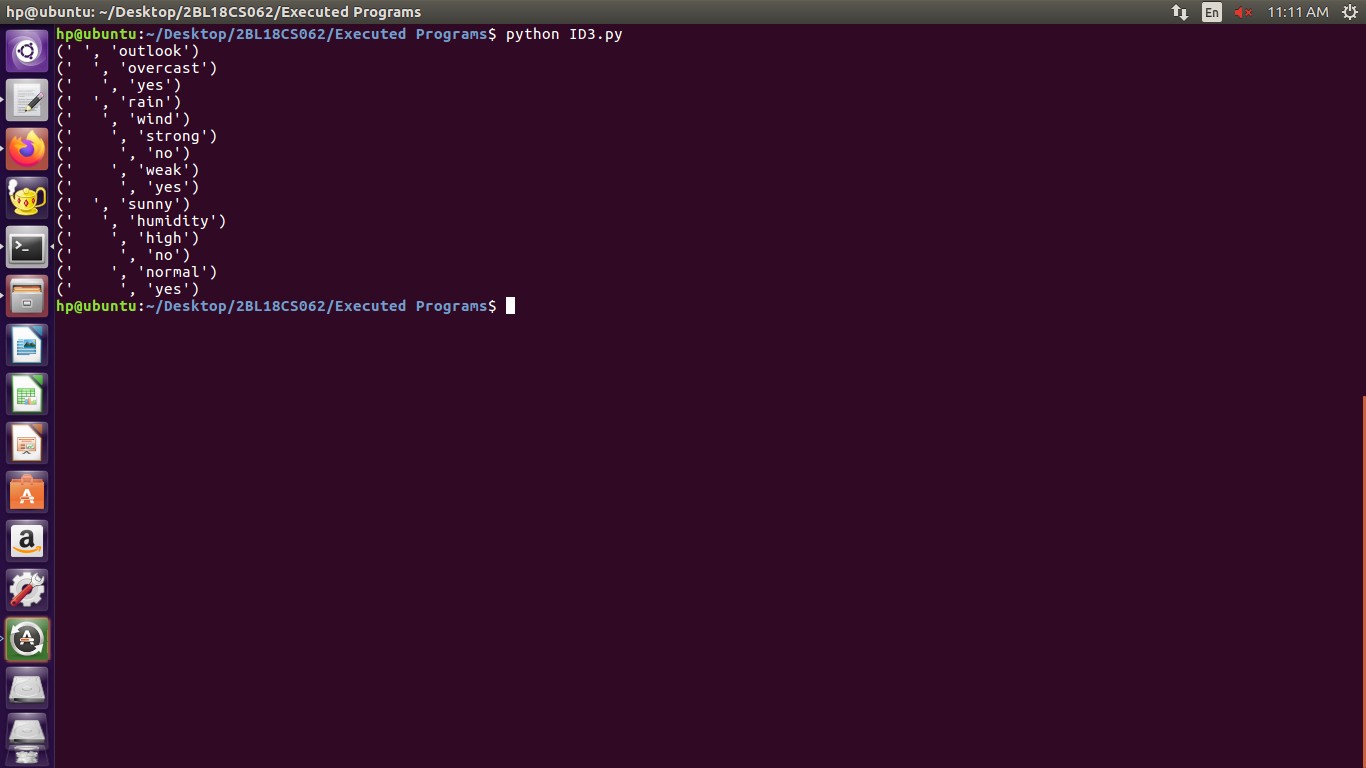
metadata,traindata=read\_data("ID3.csv")

data=np.array(traindata)

node=create\_node(data,metadata)

print\_tree(node,0)

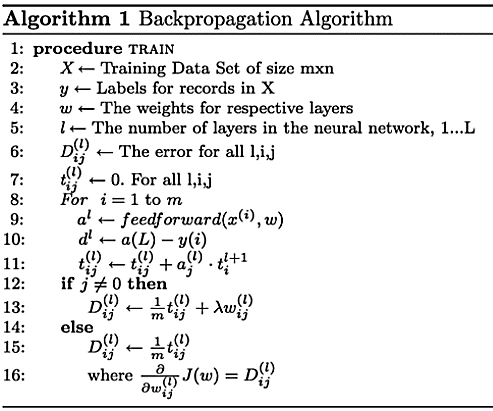
**OUTPUT**



**Experiment 5: Build an Artificial Neural Network by implementing the Back propagation algorithm and test the same using appropriate data sets.**

Backpropagation neural network is used to improve the accuracy of neural network and make them capable of self-learning. Backpropagation means “backward propagation of errors”. Here error is spread into the reverse direction in order to achieve better performance.

Backpropagation is an algorithm for supervised learning of artificial neural networks that uses the gradient descent method to minimize the cost function. It searches for optimal weights that optimize the mean-squared distance between the predicted and actual labels.



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

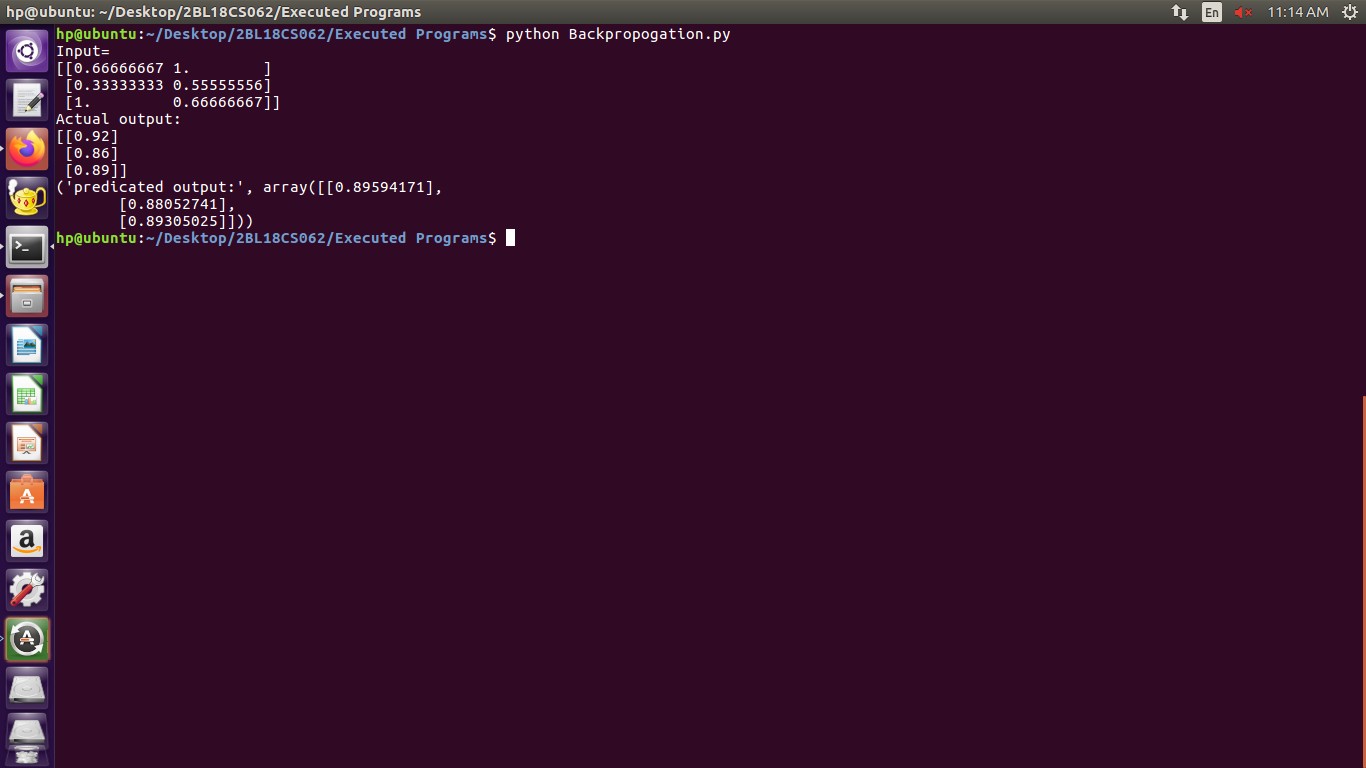
wh+=X.T.dot(d\_hiddenlayer)\*lr

print("Input=\n"+str(X))

print("Actual output:\n"+str(y))

print("predicated output:",output)

**OUTPUT**



**Experiment 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 Bayes algorithm is a supervised learning algorithm, which is based on Bayes theorem and used for solving classification problems.

It is mainly used in text classification that includes a high-dimensional training dataset.

Naïve Bayes Classifier is one of the simple and most effective Classification algorithms which helps in building the fast machine learning models that can make quick predictions.

It is a probabilistic classifier, which means it predicts on the basis of the probability of an object.

Some popular examples of Naïve Bayes Algorithm are spam filtration, Sentimental analysis, and classifying articles.

The Naïve Bayes algorithm is comprised of two words Naïve and Bayes, Which can be described as:

Naïve: It is called Naïve because it assumes that the occurrence of a certain feature is independent of the occurrence of other features. Such as if the fruit is identified on the bases of color, shape, and taste, then red, spherical, and sweet fruit is recognized as an apple. Hence each feature individually contributes to identify that it is an apple without depending on each other.

Bayes: It is called Bayes because it depends on the principle of Bayes' Theorem.

**Bayes' Theorem:**

* Bayes' theorem is also known as **Bayes' Rule** or **Bayes' law**, which is used to determine the probability of a hypothesis with prior knowledge. It depends on the conditional probability.
* The formula for Bayes' theorem is given as:

Naïve Bayes Classifier Algorithm

**Where,**

**P(A|B) is Posterior probability**: Probability of hypothesis A on the observed event B.

**P(B|A) is Likelihood probability**: Probability of the evidence given that the probability of a hypothesis is true.

**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(testSet)))

summaries = summarizeByClass(trainingSet);

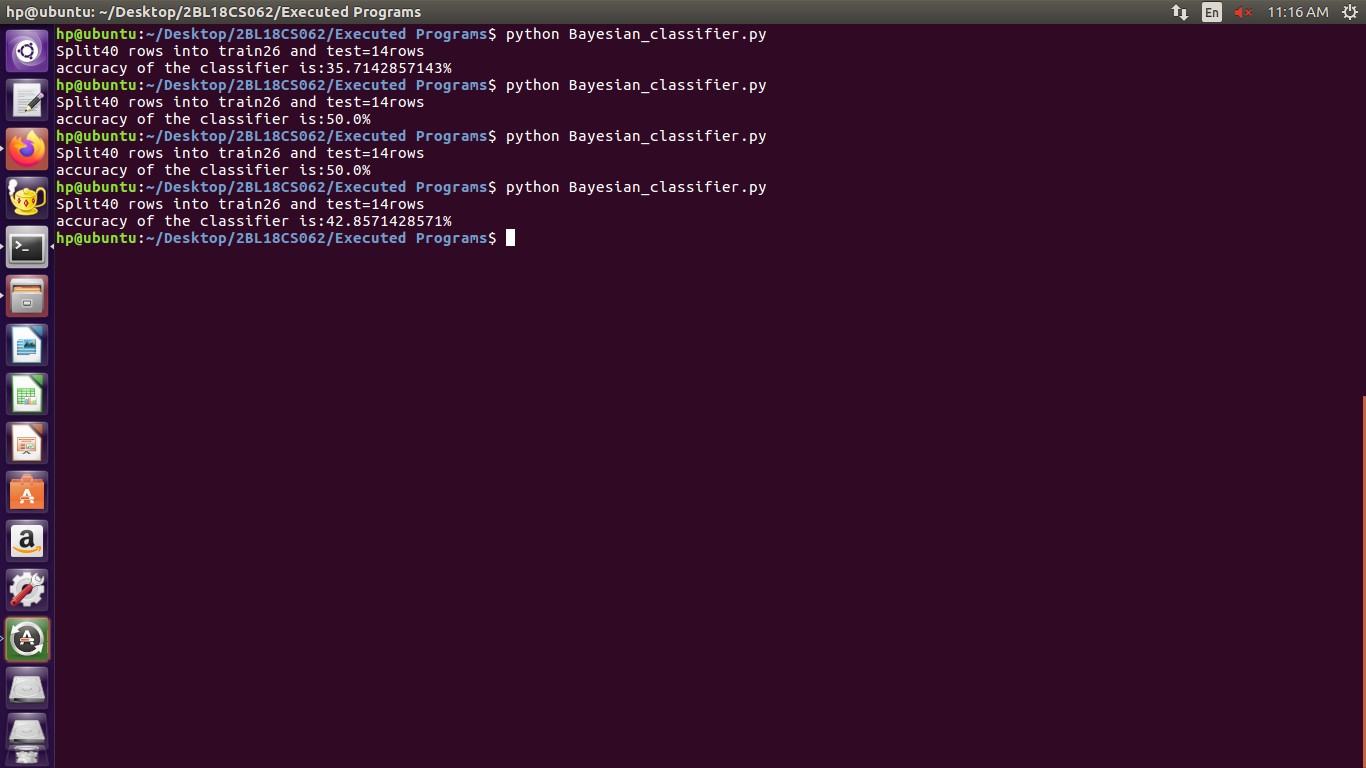
predictions=getPredictions(summaries,testSet)

accuracy=getAccuracy(testSet,predictions)

print('accuracy of the classifier is:{0}%'.format(accuracy))

main()

**OUTPUT**

****

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

**K-Means Clustering Algorithm**

K-Means Clustering is an unsupervised learning algorithm that is used to solve the clustering problems in machine learning or data science. In this topic, we will learn what is K-means clustering algorithm, how the algorithm works, along with the Python implementation of k-means clustering.

**What is K-Means Algorithm?**

K-Means Clustering is an Unsupervised Learning algorithm, which groups the unlabeled dataset into different clusters. Here K defines the number of pre-defined clusters that need to be created in the process, as if K=2, there will be two clusters, and for K=3, there will be three clusters, and so on.

It allows us to cluster the data into different groups and a convenient way to discover the categories of groups in the unlabeled dataset on its own without the need for any training.

It is a centroid-based algorithm, where each cluster is associated with a centroid. The main aim of this algorithm is to minimize the sum of distances between the data point and their corresponding clusters.

The algorithm takes the unlabeled dataset as input, divides the dataset into k-number of clusters, and repeats the process until it does not find the best clusters. The value of k should be predetermined in this algorithm.

The k-means clustering algorithm mainly performs two tasks:

Determines the best value for K center points or centroids by an iterative process.

Assigns each data point to its closest k-center. Those data points which are near to the particular k-center, create a cluster.

Hence each cluster has datapoints with some commonalities, and it is away from other clusters.

**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\_):

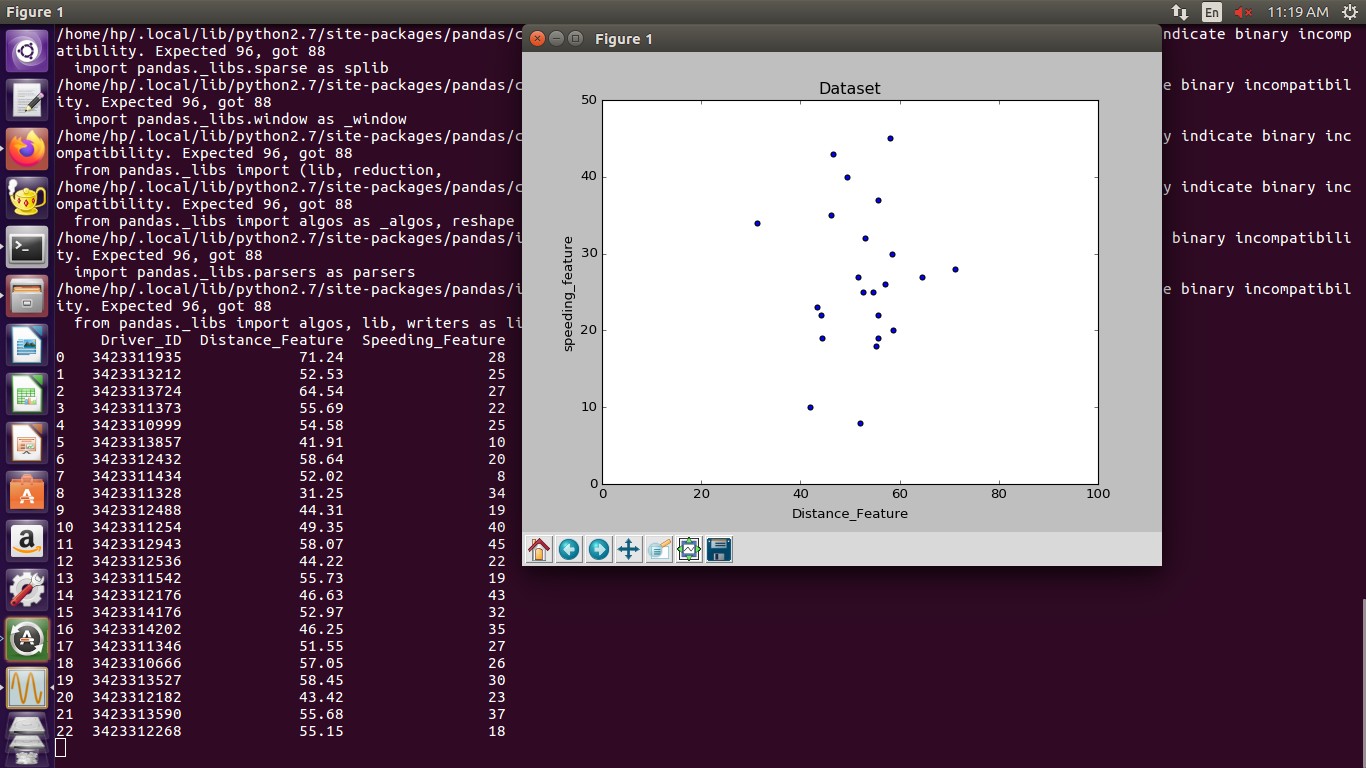
plt.plot(f1[i], f2[i], color=colors[l], marker=markers[l],ls='None')

plt.xlim([0, 100])

plt.ylim([0, 50])

plt.show()

**OUTPUT**



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

K-Nearest Neighbor(KNN) Algorithm for Machine Learning

K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.

K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.

K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.

K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.

K-NN is a non-parametric algorithm, which means it does not make any assumption on underlying data.

It is also called a lazy learner algorithm because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.

KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.

The K-NN working can be explained on the basis of the below 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:** Our 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')

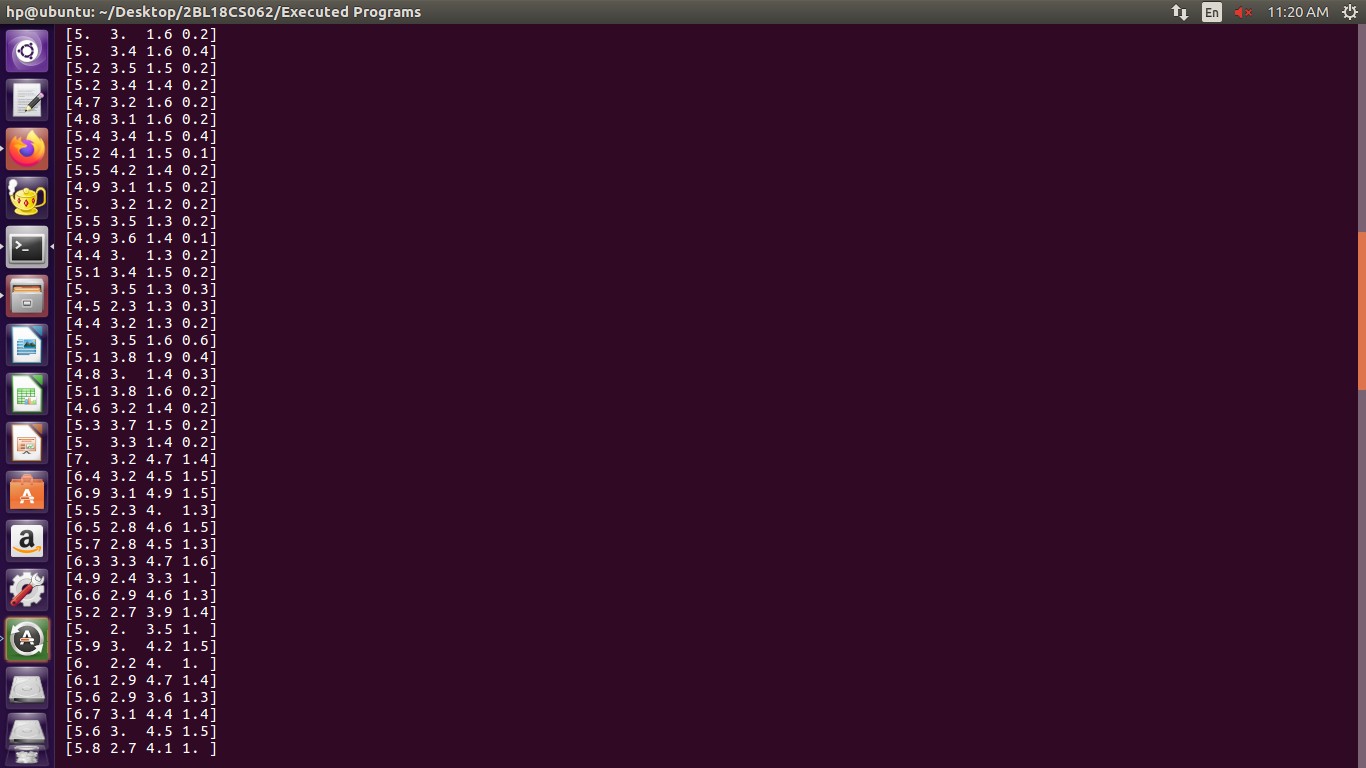
print(confusion\_matrix(y\_test,y\_pred))

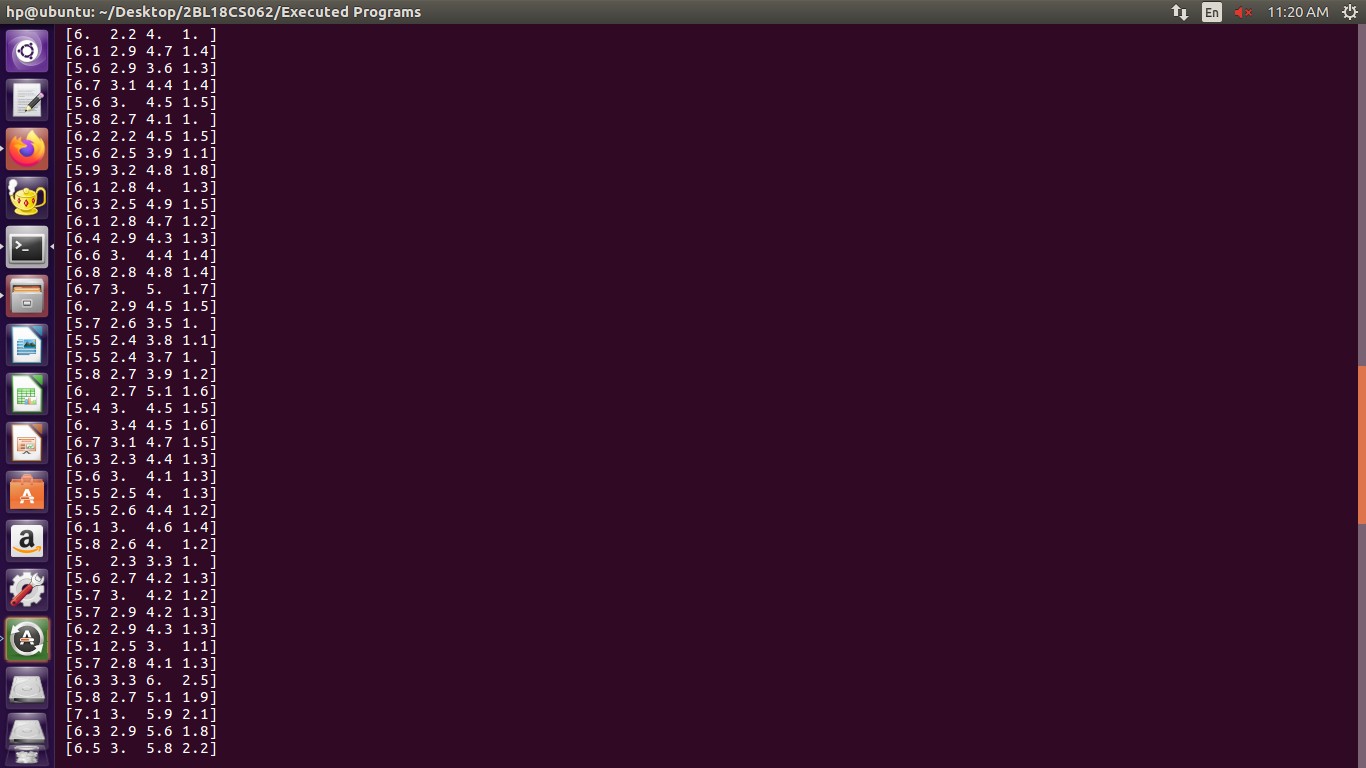
print('Accuracy metrics')

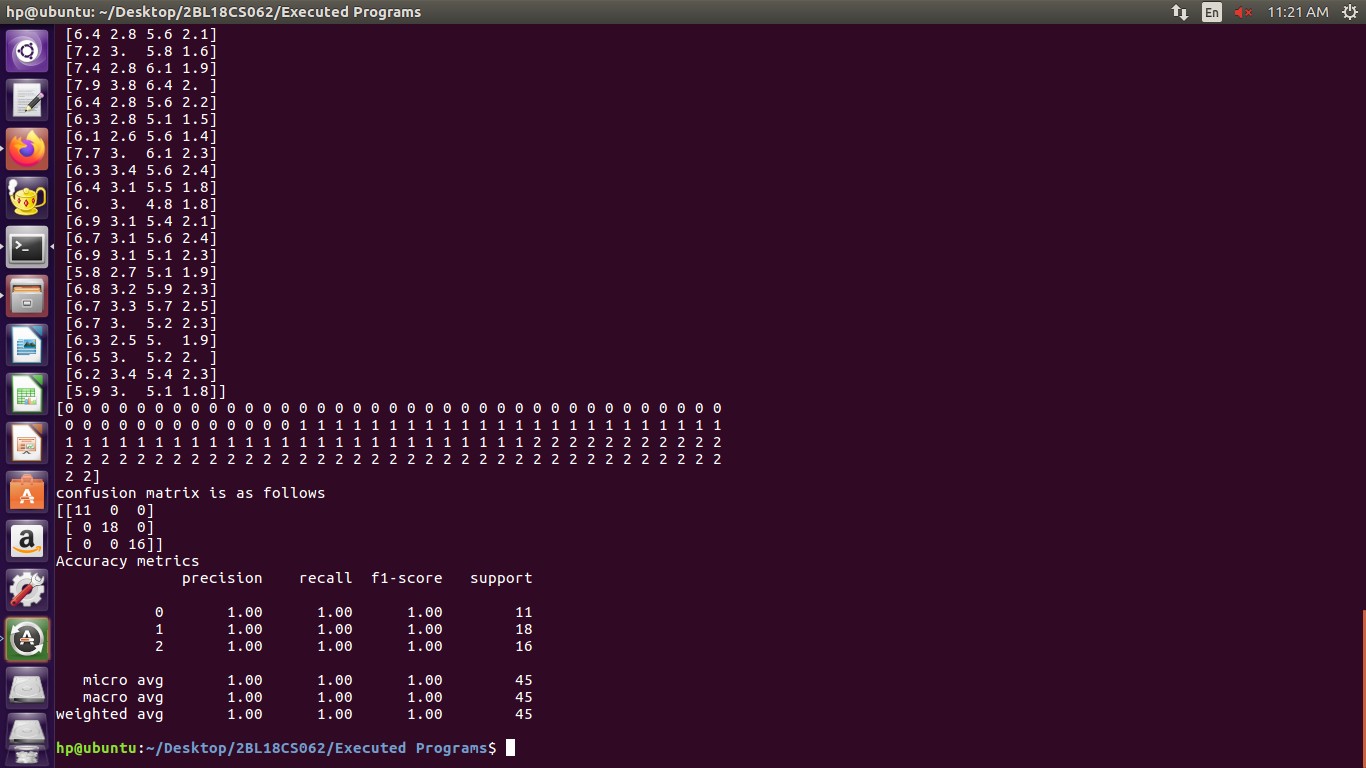
print(classification\_report(y\_test,y\_pred))

**OUTPUT**





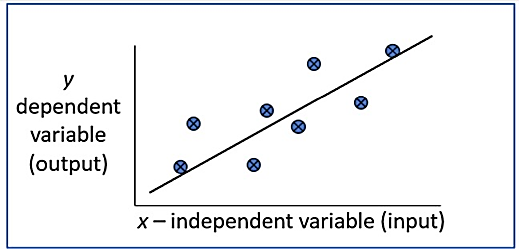




**Experiment 9: Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs**

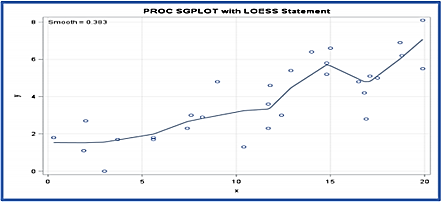
### Regression:

* Regression is a technique from statistics that are used to predict values of the desired target quantity when the target quantity is continuous.
  + In regression, we seek to identify (or estimate) a continuous variable y associated with a given input vector x.
    - y is called the dependent variable.
    - x is called the independent variable.



### Loess/Lowess Regression:

Loess regression is a nonparametric technique that uses local weighted regression to fit a smooth curve through points in a scatter plot.



## Algorithm

1. Read the Given data Sample to X and the curve (linear or non linear) to Y

2. Set the value for Smoothening parameter or Free parameter say τ

3. Set the bias /Point of interest set x0 which is a subset of X

4. Determine the weight matrix using :

Locally Weighted Regression Algorithm in Python - 2

5. Determine the value of model term parameter β using:

Locally Weighted Regression Algorithm in Python -1

6. Prediction = x0\*β

Source Code:

#Importing Required Moudles

import matplotlib.pyplot as plt

import pandas as pd

import numpy as np1

#Calculationg Weight Matrix Using e^(-(x-x0)^2/2\*r^2)

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)) # Using above Formula

return weights

def localWeight(point,xmat,ymat,k):

wei = kernel(point,xmat,k)

W=(X.T\*(wei\*X)).I\*(X.T\*(wei\*ymat.T)) # Calculate Beta(model term parameter) Using β(xo) = (X^T WX)^-1 X^T Wy

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') #Giving Color to Points

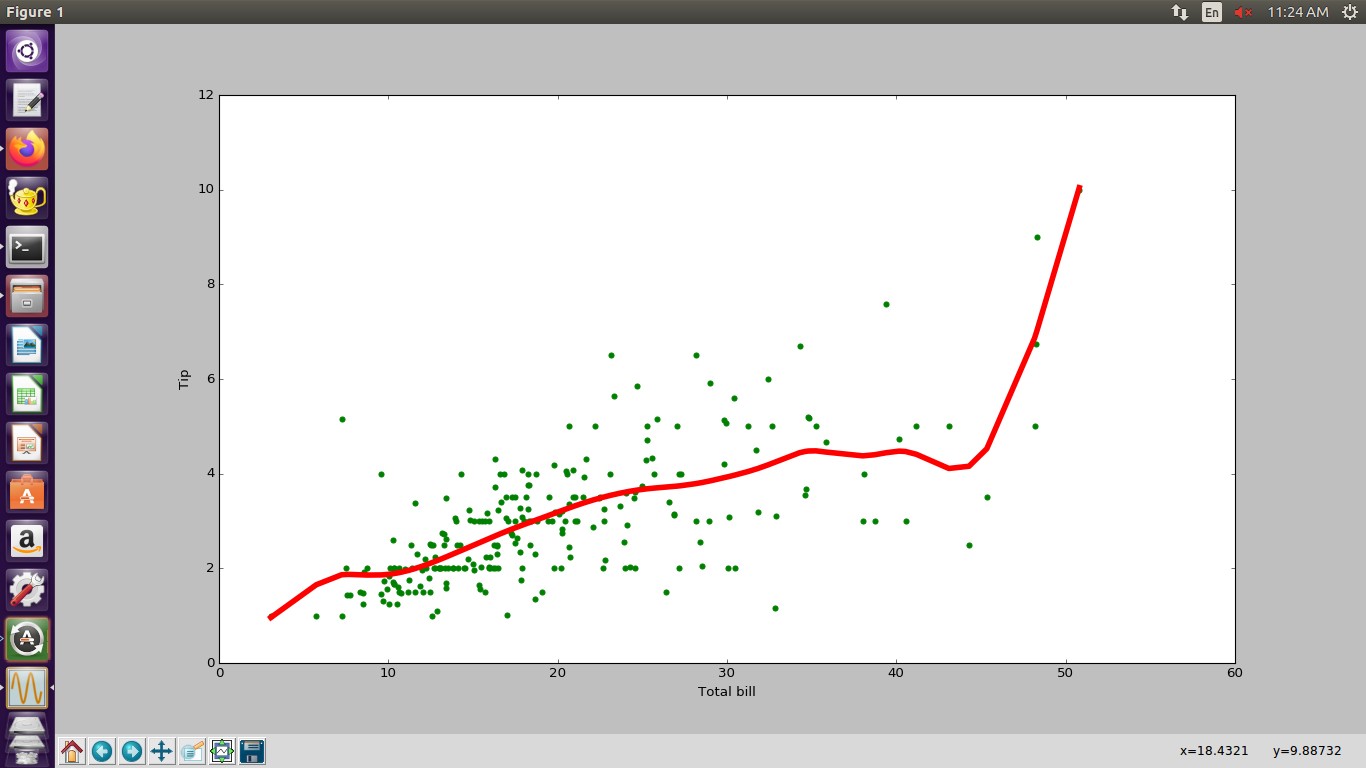
ax.plot(xsort[:,1],ypred[SortIndex], color = 'red', linewidth=5) #Giving Color to line

plt.xlabel('Total bill')

plt.ylabel('Tip')

plt.show()

**OUTPUT**



VIVA VOCE Questions

1. What is artificial intelligence and machine learning?

2. Define supervised learning

3. Define unsupervised learning

4. Define semi supervised learning

5. Define reinforcement learning

6. What do you mean by hypotheses?

7. What is classification?

8. What is clustering? Give Examples.

9. Define precision, accuracy and recall

10. Define entropy

11. Define regression

12. How Knn is different from k-means clustering

13. What is concept learning?

14. Define specific boundary and general boundary

15. Define target function

16. Define decision tree

17. What is ANN ? Give example.

18. Explain gradient descent approximation

19. State Bayes theorem

20. Define Bayesian belief networks

21. Differentiate hard and soft clustering

22. Define variance

23. What is inductive machine learning?

24. Why K nearest neighbor algorithm is lazy learning algorithm

25. Why naïve Bayes is naïve

26. Mention classification algorithms

27. Define pruning

28. Differentiate Clustering and classification

29. Mention clustering algorithms

30. Define Bias.