krunal dhavle

TyCS-713

Artificial Intelligence

Name : Krunal Dhavle

Class : TYCS

Roll No : 713

Subject : Artificial Intelligence Practicals

Teacher : Manali Patil Maam

**INDEX**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **SR NO** | **DATE** | **PRACTICAL AIM** | **PG. NO** | **REMARK** |
| 1 | 24/08/20 | Implement Breadth first search algorithm for Romanian map problem or any other map. | 3-4 |  |
| 2 | 31/08/20 | Implement Iterative deep depth first search for Romanian map problem or any other map. | 5-6 |  |
| 3 | 07/09/20 | Implement A\* search algorithm for Romanian map problem or any other map. | 7 |  |
| 4 | 05/10/20 | Implement recursive best-first search algorithm for Romanian map problem or any other map. | 8-10 |  |
| 5 | 09/09/20 | Implement decision tree learning algorithm for the restaurant waiting problem. | 11-12 |  |
| 6 | 11/10/20 | Implement feed forward back propagation neural network learning algorithm for the restaurant waiting problem. | 13-15 |  |
| 7 | 2/11/20 | Implement Adaboost ensemble learning algorithm for the restaurant waiting problem | 16 |  |
| 8 | 23/11/20 | Implement Naive Bayes’ learning algorithm for the restaurant waiting problem. | 17-18 |  |

**Date: 24/08/2020**

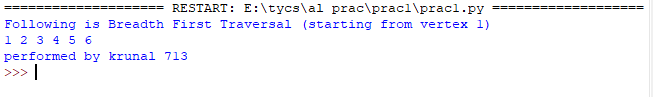
**Practical no 1**

**AIM**: Implement Breadth first search algorithm for Romanian map problem or any other map.

**CODE:-**

|  |
| --- |
| from collections import defaultdict  class Graph:  def \_\_init\_\_(self):  self.graph = defaultdict(list)  def addEdge(self,u,v):  self.graph[u].append(v)  def BFS(self, s):  visited = [False] \* (len(self.graph))  queue = []  queue.append(s)  visited[s-1] = True  while queue:  s = queue.pop(0)  print (s, end = " ")  for i in self.graph[s]:  if visited[i-1] == False:  queue.append(i)  visited[i-1] = True  g = Graph()  g.addEdge(1, 2)  g.addEdge(1, 3)  g.addEdge(2, 1)  g.addEdge(2, 4)  g.addEdge(2, 5)  g.addEdge(3, 1)  g.addEdge(3, 5)  g.addEdge(4, 2)  g.addEdge(4, 6)  g.addEdge(5, 2)  g.addEdge(6, 5)  print ("Following is Breadth First Traversal"  " (starting from vertex 1)")  g.BFS(1)  print("\nperformed by krunal 713") |

**OUTPUT:-**



**Date:31/08/2020**

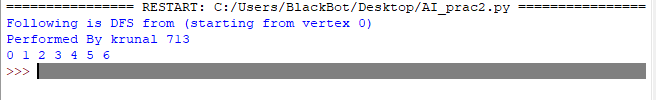
**Practical no 2**

**AIM**: Implement Iterative deep depth first search for Romanian map problem or any other map

**CODE:**-

|  |
| --- |
| from collections import defaultdict  class Graph:  def \_\_init\_\_(self):  self.graph = defaultdict(list)  def addEdge(self, u, v):  self.graph[u].append(v)  def DFSUtil(self, v, visited):  visited[v] = True  print(v, end = ' ')  for i in self.graph[v]:  if visited[i] == False:  self.DFSUtil(i, visited)  def DFS(self, v):  visited = [False] \* (max(self.graph)+1)  self.DFSUtil(v, visited)  g = Graph()  g.addEdge(0, 1)  g.addEdge(0, 2)  g.addEdge(1, 2)  g.addEdge(2, 0)  g.addEdge(2, 3)  g.addEdge(3, 3)  g.addEdge(3, 4)  g.addEdge(4, 4)  g.addEdge(4, 5)  g.addEdge(5, 4)  g.addEdge(5, 5)  g.addEdge(4, 6)  g.addEdge(5, 6)  g.addEdge(6, 6)  print("Following is DFS from (starting from vertex 0)")  print("Performed By krunal 713")  g.DFS(0) |

**OUTPUT:-**

****

**Date:07/09/2020**

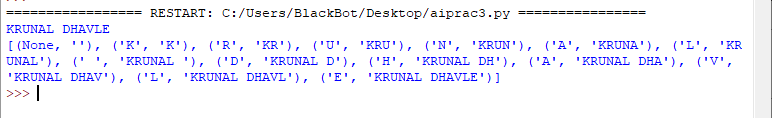
**Practical no 3**

**AIM**: Implement A\* search algorithm for Romanian map problem or any other map.

**CODE:**

|  |
| --- |
| from simpleai.search import SearchProblem, astar  GOAL = 'KRUNAL DHAVLE'  class HelloProblem(SearchProblem):  def actions(self, state):  if len(state) < len(GOAL):  return list(' ABCDEFGHIJKLMNOPQRSTUVWXYZ')  else:  return []  def result(self, state, action):  return state + action  def is\_goal(self, state):  return state == GOAL  def heuristic(self, state):    wrong = sum([1 if state[i] != GOAL[i]  else 0  for i in range(len(state))])  missing = len(GOAL) - len(state)  return wrong + missing  problem = HelloProblem(initial\_state='')  result = astar(problem)  print(result.state)  print(result.path()) |

**OUTPUT:**

****

**Date:05/10/2020**

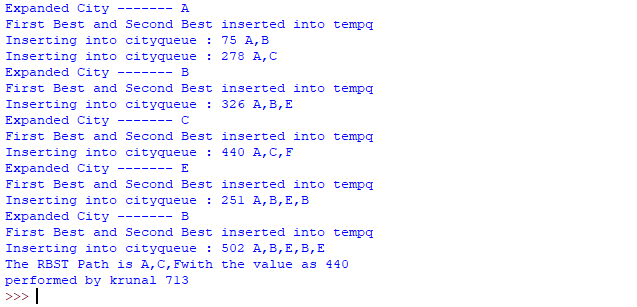
**Practical no 4**

**AIM**: Implement recursive best-first search algorithm for Romanian map problem.

**CODE**:

|  |
| --- |
| import queue as q  dict\_hn={  'A':336,  'B':0,  'C':160,  'D':242,  'E':161,  'F':176  }  dict\_gn={  'A':{'B':75,'C':118},  'B':{'A':85,'D':211,'E':90},  'C':{'A':120,'F':146},  'D':{'B':75},  'E':{'B':86},  'F':{'C':99}  }  def get\_fn(citystr):  cities=citystr.split(',')  hn=0  gn=0  ctr=0  while ctr!=len(cities)-1:  gn=gn+dict\_gn[cities[ctr]][cities[ctr+1]]  ctr+=1  hn=dict\_hn[cities[len(cities)-1]]  return hn+gn  def expand(mycities,cityq,goal):  tot,citystr=mycities  cities=citystr.split(',')  city2expand=cities[len(cities)-1]  if(city2expand==goal):  ans="The RBST Path is "+citystr+"with the value as "+str(tot);  while not cityq.empty():  cityq.get()  return ans  print("Expanded City -------",city2expand)  tempq=q.PriorityQueue()  for city in dict\_gn[city2expand]:  tempq.put((get\_fn(citystr+','+city),citystr+','+city))  print('First Best and Second Best inserted into tempq')  ctr=1  if(cityq.empty()):  while not tempq.empty():  if ctr==1 or ctr==2:  tempgn,tempcitystr=tempq.get()  print('Inserting into cityqueue :',tempgn,tempcitystr)  cityq.put((tempgn,tempcitystr))  ctr=ctr+1  else:  #pass  tempq.get()  else:  fn=0  citystr=""  fn=getSecondBest(cityq,fn,citystr)  while not tempq.empty():  if ctr==1 or ctr==2:  tempgn,tempcitystr=tempq.get()  if tempgn>ctr:  if ctr==1:  print('Inserting into cityqueue :',tempgn,tempcitystr)  cityq.put((tempgn,tempcitystr))  ctr=3  continue  else:  #break  print("Inserting into CityQueue:",tempgn,citystr)  cityq.put((tempgn,tempcitystr))  ctr+=1  else:  tempq.get()  while not tempq.empty():  tempq.get()  def getSecondBest(cityq,fn,citystring):  fn,citystring=cityq.get()  cityq.put((fn,citystring))  return fn  def main():  start="A"  goal="F"  cityq=q.PriorityQueue()  cityq.put((get\_fn(start),start))  while not cityq.empty():  mycities=cityq.get()  ans=expand(mycities,cityq,goal)  print(ans)  print('performed by krunal 713')  main() |

**OUTPUT:**

****

**Date:09/09/2020**

**Practical no 5**

**AIM:** Implement decision tree learning algorithm for the restaurant waiting problem.

**STEPS:**

**Step1**: Download the graph viz file from below link and extract it.

https://graphviz.gitlab.io/\_pages/Download/windows/graphviz-2.38.zip

**Step2:** Install the sklearn , ipython and pydotplus packages.First copy the path of script in python

folder and then change the path of cmd.

**Step3**: Now install the packages by writing pip install and the packages name.

**Step4**: Next you have to change the environment variable. Copy the path of graphiz..Then go to

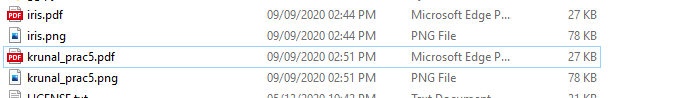
environment and add new path.

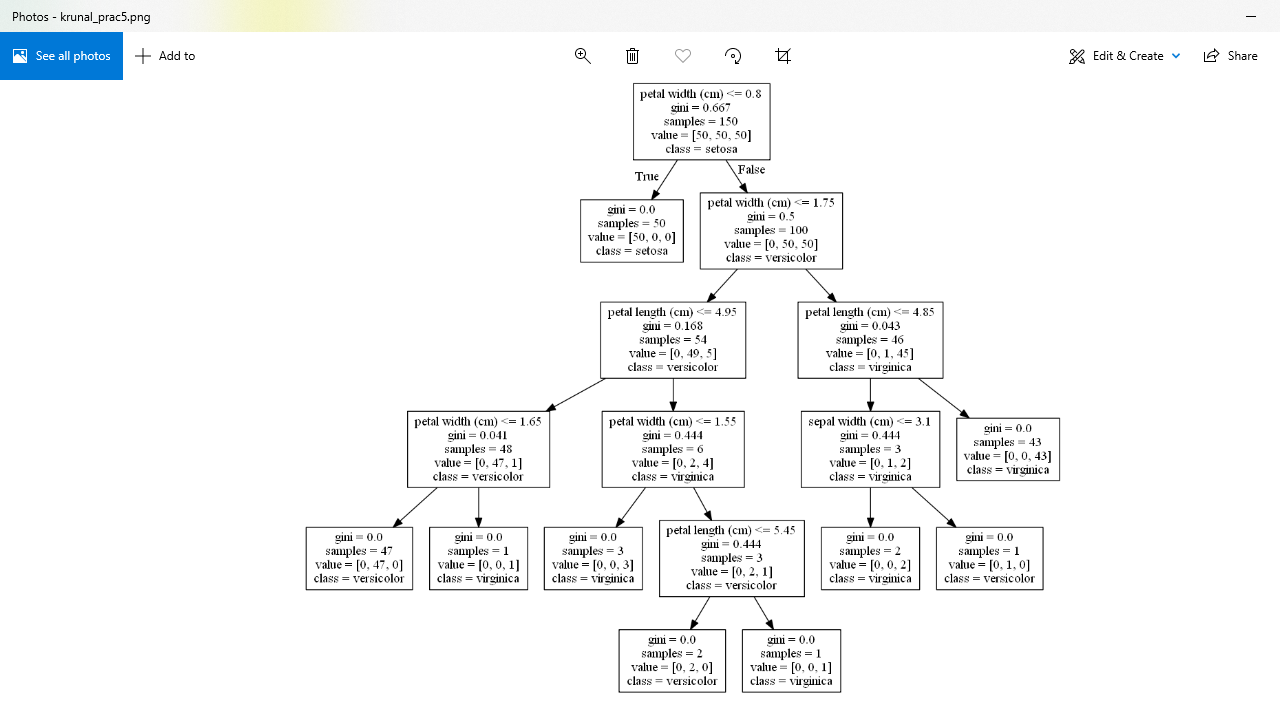
**Step5**: After all this is done write the code and run it . output will be in pdf and png format.

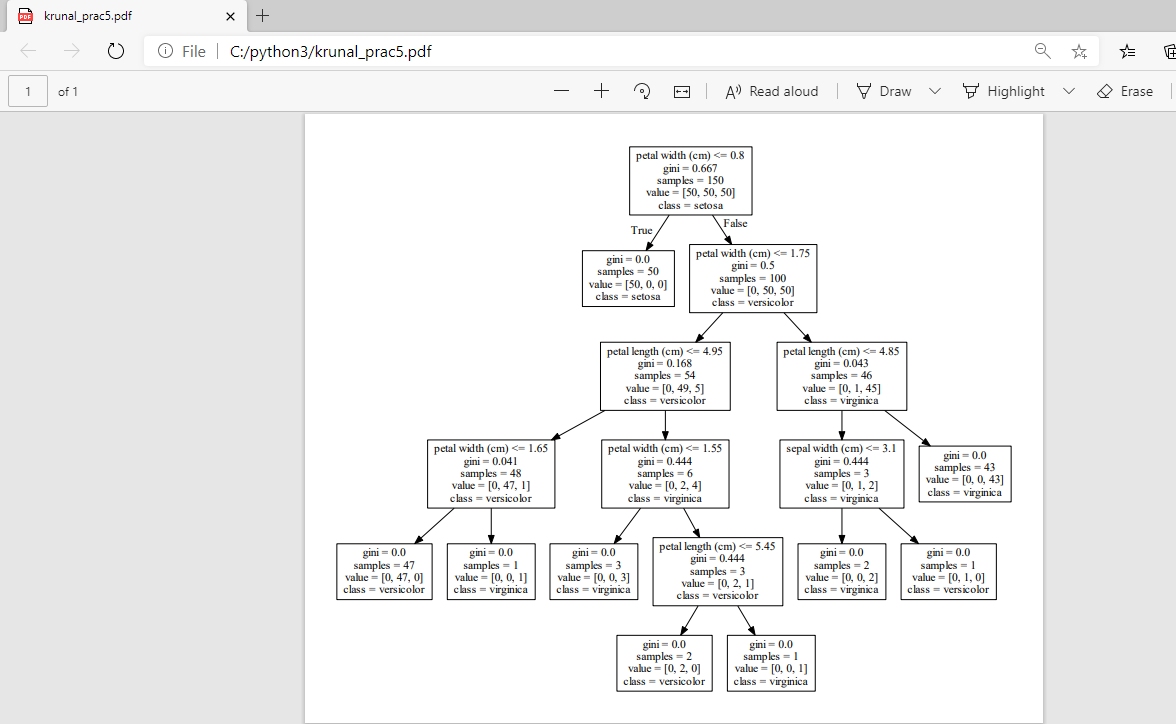
**CODE:**

|  |
| --- |
| from sklearn.tree import DecisionTreeClassifier  from sklearn import datasets  from IPython.display import Image  from sklearn import tree  import os #only for windows  import pydotplus  os.environ['PATH'] += os.pathsep+ "C:/graphviz-2.38/release/bin/"  iris=datasets.load\_iris()  x=iris.data  y=iris.target  clf=DecisionTreeClassifier(random\_state=0)  model=clf.fit(x,y)  dot\_data=tree.export\_graphviz(clf,out\_file=None,feature\_names=iris.feature\_nam  es,class\_names=iris.target\_names)  graph =pydotplus.graph\_from\_dot\_data(dot\_data)  Image(graph.create\_png())  graph.write\_pdf("krunal\_prac5.pdf")  graph.write\_png("krunal\_prac5.png") |

**OUTPUT:**







**Date:11/10/2020**

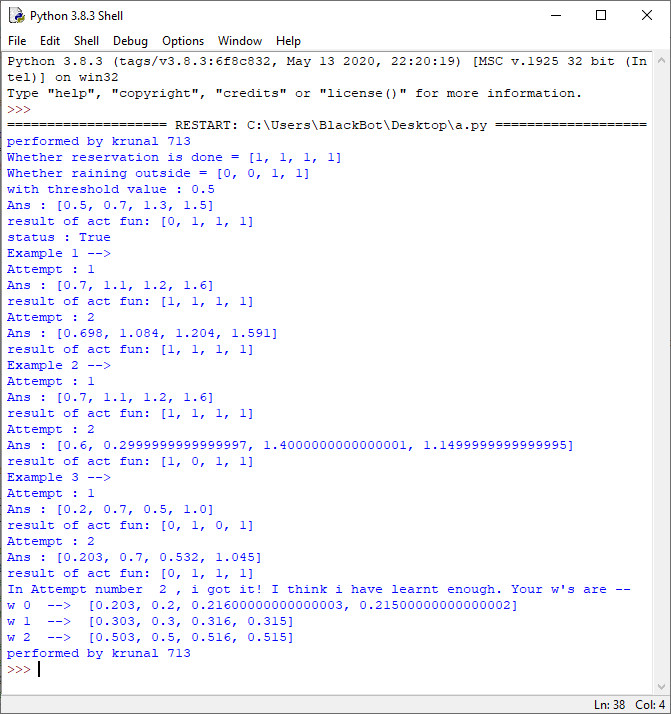
**Practical no 6**

**AIM**: Implement feed forward back propagation neural network learning algorithm for the restaurant waiting problem.

**CODE:**

|  |
| --- |
| Implement feed forward back propagation neural network learning algorithm for the restaurant waiting problem  class Perceptron : # With 2 inputs and 1 output  def \_\_init\_\_(self, a,b, c, tval):  self.x = a # input vector  self.result = b # activation result  self.cresult = c # summation result  self.threshold = tval # threshold value used by activation function  self.w = []  def h(self, tw): # calculating summation(hypothesis function)  hresult= []  for i in range(0 , len(self.result)):  hresult.append(0)  #print("index - ", i, ";", hresult)  for j in range(0,len(tw)):  #print("i=",i, ",j=",j)  hresult[i] = hresult[i] + ( tw[j][i]\*self.x[j][i] )  return hresult  def checkthreshold(self, hresult): # applying activation function on summation result using threshold value  #flag = True  actfun =[]  for i in range(0 , len(self.result)) :  if (hresult[i] <= self.threshold ):  actfun.append(0)  else :  actfun.append( 1)  print("Ans :", hresult)  print("result of act fun:", actfun)  for i in range(0 , len(self.x)) :  if (actfun[i] != self.result[i]) :  return False  return True    def training(self, tw, alpha): #passing w vector and alpha value  i=1  while i<=2 : # Max 100 attempts  print("Attempt :", i)  hresult = self.h(tw)    if(self.checkthreshold(hresult)) : #if training result matches the test result  self.w = tw    print("In Attempt number ", i, ", i got it! I think i have learnt enough. Your w's are --" )  for x in range(0,len(self.w)):  print("w", x, " --> ", self.w[x])  break  i = i +1  # Changing values of w to reduce error/loss using batch gradient descent learning rule given on page 721 eqn 18.6  for j in range(0,len(self.result)) :  for k in range(0, len(tw)):  sum = 0  for n in range(0, len(tw)):  sum = sum + (self.cresult[j] - hresult[j]) \*self.x[n][j]  tw[k][j] = tw[k][j] + alpha\*sum  if(i>=100):  print("I am exhausted, tried 100 iterations! plz change something else...")  a = [ [1,1,1,1], [0,0,1,1] , [0,1,0,1] ] # x vector, x0 is dummy  b = [0,1,1,1] # result of activation function  c = [0.5, 0.7, 1.3, 1.5] # sample h values  p = Perceptron(a,b,c, 0.5) # threshold = 0.5  print("Whether reservation is done =", p.x[0])  print("Whether raining outside =", p.x[1])  print("with threshold value :", p.threshold)  r = p.h([ [0.5,0.5,0.5,0.5], [0.8, 0.8, 0.8, 0.8], [0.2, 0.2, 0.2, 0.2]])  print("status :", p.checkthreshold(r))  print("Example 1 -->") #with alpha as 0.01, you will not get result  p.training( [ [0.7,0.7,0.7,0.7], [0.5, 0.5, 0.5, 0.5], [0.4, 0.4, 0.4, 0.4]], 0.01)  print("Example 2 -->") #with alpha as 0.5, you will not get result  p.training( [ [0.7,0.7,0.7,0.7], [0.5, 0.5, 0.5, 0.5], [0.4, 0.4, 0.4, 0.4]], 0.5)  print("Example 3 -->")  p.training( [ [0.2,0.2,0.2,0.2], [0.3, 0.3, 0.3, 0.3], [0.5, 0.5, 0.5, 0.5]], 0.01) |

**OUTPUT :**

****

**Date:02/11/2020**

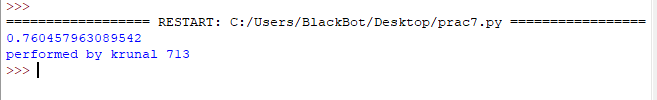
**Practical no 7**

**AIM**: Implement Adaboost ensemble learning algorithm for the restaurant waiting problem Or any other problem.

**CODE:**

|  |
| --- |
| import pandas  from sklearn import model\_selection  from sklearn.ensemble import AdaBoostClassifier  url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"  names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']  dataframe = pandas.read\_csv(url, names=names)  array = dataframe.values  X = array[:,0:8]  Y = array[:,8]  seed = 7  num\_trees = 30  kfold = model\_selection.KFold(n\_splits=10)  model = AdaBoostClassifier(n\_estimators=num\_trees, random\_state=seed)  results = model\_selection.cross\_val\_score(model, X, Y, cv=kfold)  print(results.mean()) |

**OUTPUT:**

****

**Date:23/11/2020**

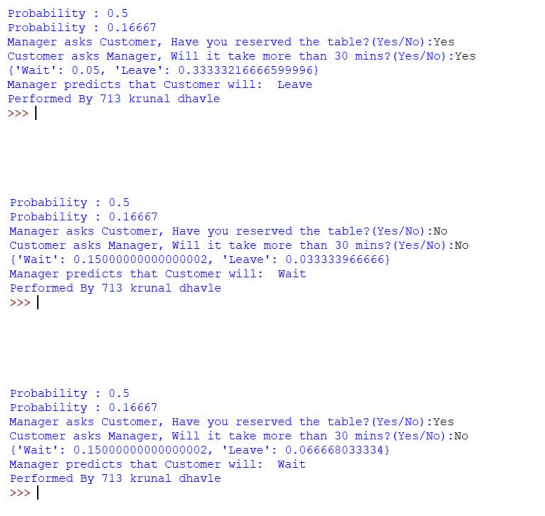
**Practical no 8**

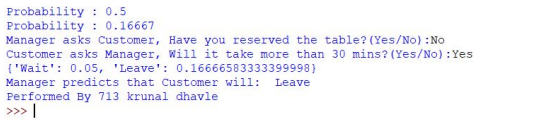
**Aim**: Implement Naive Bayes learning algorithm for the restaurant waiting problem.

**CODE:**

|  |
| --- |
| class NaiveBayes:  def \_\_init\_\_(self, f, r):  self.features = f  self.response = r  def predict(self,custcase):  anskeys = list(self.response.keys())  ansvalues = dict.fromkeys(anskeys,0)  for key in anskeys :  ansvalues[key] = self.response[key]  for ikey, ival in custcase.items() :  ansvalues[key] = ansvalues[key] \* self.features[ikey][ival][key]  print(ansvalues)  maxkey=""  maxans=-1  for ikey, ival in ansvalues.items():  if ival > maxans :  maxans= ival  maxkey = ikey  return maxkeyresponse = {"Wait":0.4, "Leave":0.6}  features = {  "Reservation":  {  "Yes" : {"Wait":0.5, "Leave":0.666667},  "No" : {"Wait":0.5, "Leave":0.333333}  } ,  "Time>30":  {  "Yes" : {"Wait":0.25, "Leave":0.83333},  "No" : {"Wait":0.75, "Leave":0.16667}  }  }  nb = NaiveBayes(features, response)  print("Probability :", nb.features["Reservation"]["Yes"]["Wait"])  print("Probability :", nb.features["Time>30"]["No"]["Leave"])  resstatus = input("Manager asks Customer, Have you reserved the table?(Yes/No):")  timestatus = input("Customer asks Manager, Will it take more than 30 mins?(Yes/No):")  custcase = {"Reservation":resstatus, "Time>30":timestatus}  print("Manager predicts that Customer will: " , nb.predict(custcase) )  print("Performed By 713 krunal dhavle") |

**OUTPUT:**

****

****