

|  |  |
| --- | --- |
|  |  |
|  |  |
|  | **Artificial Intelligence** |
|  | **TPGCS P401** |



CERTIFICATE

**This is here to certify that Mr.**

**Seat Number of M.Sc. II Computer Science, has satisfactorily completed the required number of experiments prescribed by the **







 **during the academic**

**year 2022 – 2023.**



**Date:**

**Place: Mumbai**

**Teacher In-Charge Head of Department**

**External Examiner**



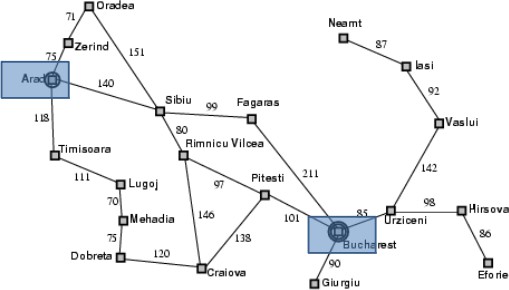
|  |  |  |
| --- | --- | --- |
|  |  |  |
|  | ***ImplementBreadthfirstsearchalgorithmforRomanian mapproblem.*** | **06-01-2023** |
|  | ***ImplementIterativedeepdepthfirstsearchforRomanian mapproblem.*** | **06-01-2023** |
|  | ***ImplementA\*searchalgorithmforRomanianmap problem.*** | **7-01-2023** |
|  | ***Implementrecursivebest-firstsearchalgorithmfor Romanianmapproblem.*** | **13-01-2023** |
|  | ***Implementdecisiontreelearningalgorithmforthe restaurantwaitingproblem.*** | **13-01-2023** |
|  | ***ImplementGeneticAlgorithmsforStaffPlanning*** | **20-01-2023** |
|  | ***ImplementANN*** | **20-01-2023** |
|  | ***Implementfeedforwardbackpropagationneuralnetwork learningalgorithmfortherestaurantwaitingproblem.*** | **27-01-2023** |
|  | ***ImplementthePerceptronAlgorithm*** | **27-01-2023** |
|  | ***ImplementFuzzyInferenceSystem*** | **03-02-2023** |
|  | ***SolveFuzzyControlSystems:TheTippingProblem*** | **10-02-2023** |
|  | ***ImplementNaiveBayes’learningalgorithmforthe restaurantwaitingproblem.*** | **17-02-2023** |

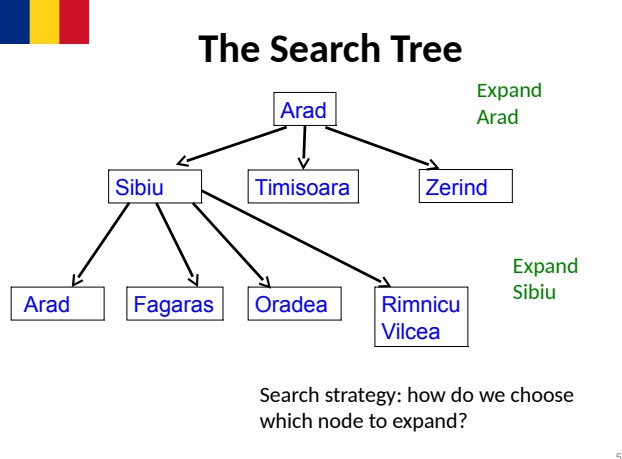


 ***ImplementBreadthfirstsearchalgorithmforRomanianmapproblem.***



* **In the state space view of the world, finding a solution is finding a path through the state space.**
* **When we (as humans) solve a problem like the 8-puzzle we have some idea of what constitutes the next best move.**
* **It is hard to program this kind of approach.**
* **Instead, we start by programming the kind of repetitive task that computers are good at.**
* **A brute force approach to problem solving involves exhaustively searching through the space of all possible action sequences to find one that achieves the goal.**



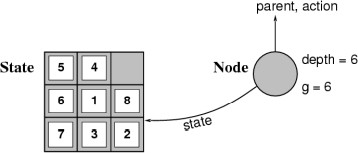




* **The tree is built by taking the initial state and identifying the states that can be obtained by a single application of the operators/actions available.**
* **These new states become the children of the initial state in the tree.**
* **These new states are then examined to see if they are the goal state.**
* **If not, the process is repeated on the new states.**
* **We can formalise this description by giving an algorithm for it.**
* **We have different algorithms for different choices of nodes to expand.**



* **A state is a (representation of) a physical configuration.**
* **A node is a data structure constituting part of a search tree that includes state, parent node, action, path cost g(x), depth.**



**Expanding the tree creates new nodes, filling in the various fields and creating the corresponding states.**



agenda = [initial state]; while agenda not empty do pick node from agenda;

new nodes = apply operations to state; if goal state in new nodes then

return solution;

else add new nodes to agenda;

* ***Question:Howtopickstatesforexpansion?***
* ***Twoobviousstrategies:***

***–depthfirstsearch;***

***–***



* **Start by expanding initial state - gives tree of depth 1.**
* **Then expand all nodes that resulted from previous step gives tree of depth 2.**
* **Then expand all nodes that resulted from previous step, and so on.**
* **Expand nodes all at depth n before going to level n + 1.**



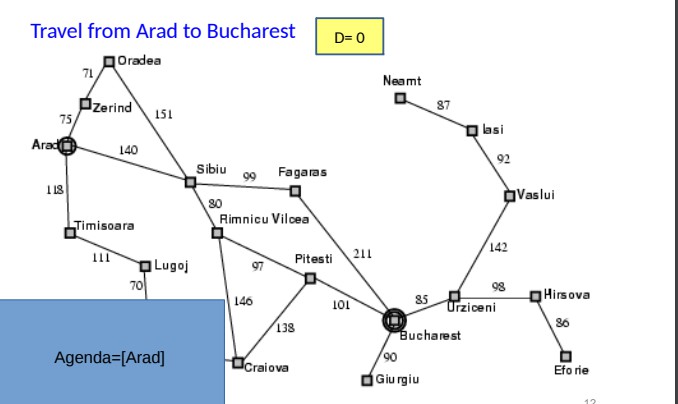
/\* Breadth first search \*/ agenda = [initial state]; while agenda not empty do

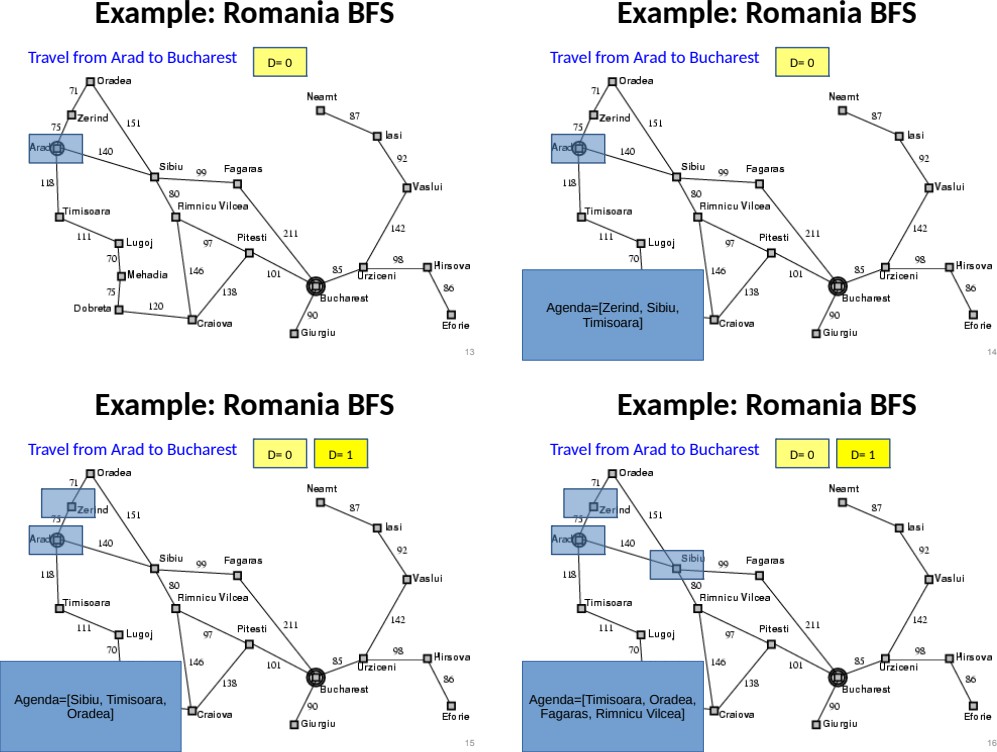
pick node from front of agenda;

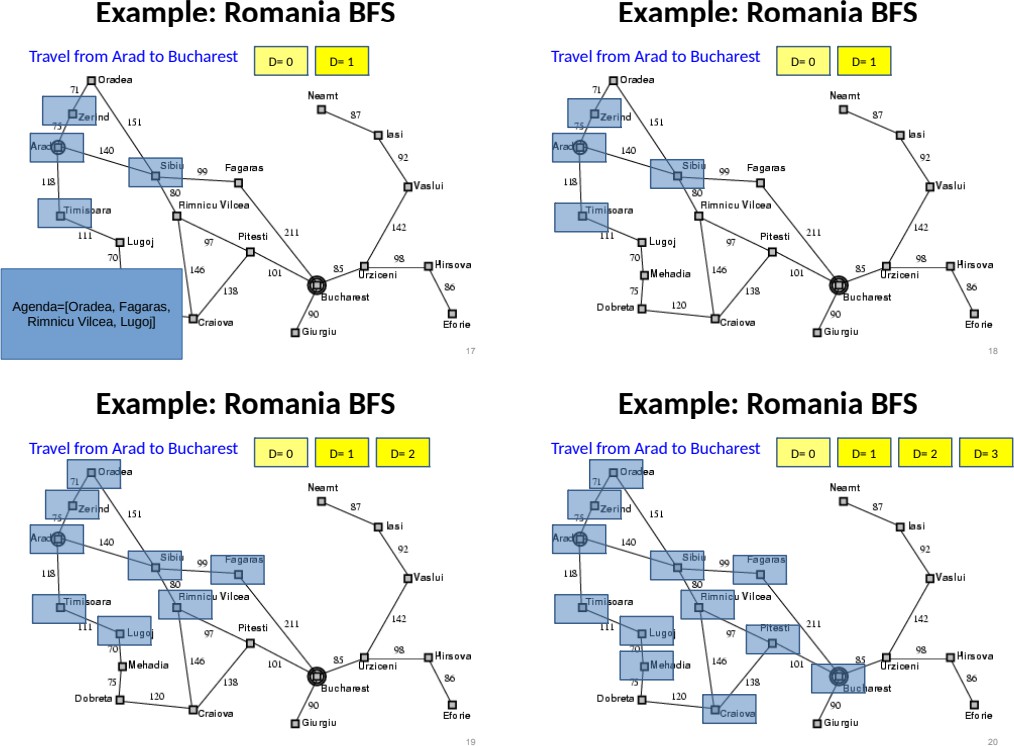
new nodes = apply operations to state; if goal state in new nodes then

return solution;

else APPEND new nodes to END of agenda









* ***Advantage:guaranteedtoreachasolutionifoneexists.***
* ***Findstheshortest(cheapest)solutionintermsofthenumberofoperationsapplied toreachasolution.***
* ***Disadvantage:timetakentoreachsolution.***

***–Letbbebranchingfactor-averagenumberofoperationsthatmaybeperformed fromanylevel.***

***–Ifsolutionoccursatdepthd,thenwewilllookat b+b2+b3+···+bd nodesbeforereachingsolutionexponential.***

***–Thememoryrequirementisbd***



from queue import Queue

romaniaMap = {

'Arad': ['Sibiu', 'Zerind', 'Timisoara'],

'Zerind': ['Arad', 'Oradea'],

'Oradea': ['Zerind', 'Sibiu'],

'Sibiu': ['Arad', 'Oradea', 'Fagaras', 'Rimnicu'], 'Timisoara': ['Arad', 'Lugoj'],

'Lugoj': ['Timisoara', 'Mehadia'],

'Mehadia': ['Lugoj', 'Drobeta'],

'Drobeta': ['Mehadia', 'Craiova'],

'Craiova': ['Drobeta', 'Rimnicu', 'Pitesti'],

'Rimnicu': ['Sibiu', 'Craiova', 'Pitesti'],

'Fagaras': ['Sibiu', 'Bucharest'],

'Pitesti': ['Rimnicu', 'Craiova', 'Bucharest'],

'Bucharest': ['Fagaras', 'Pitesti', 'Giurgiu', 'Urziceni'], 'Giurgiu': ['Bucharest'],

'Urziceni': ['Bucharest', 'Vaslui', 'Hirsova'],

'Hirsova': ['Urziceni', 'Eforie'],

'Eforie': ['Hirsova'],

'Vaslui': ['Iasi', 'Urziceni'],

'Iasi': ['Vaslui', 'Neamt'], 'Neamt': ['Iasi']

**}**

def bfs(startingNode, destinationNode):

# For keeping track of what we have visited visited = {}

# keep track of distance distance = {}

# parent node of specific graph parent = {}

bfs\_traversal\_output = []

# BFS is queue based so using 'Queue' from python built-in queue = Queue()

# travelling the cities in map for city in romaniaMap.keys():

# since intially no city is visited so there will be nothing in visited list visited[city] = False

parent[city] = None distance[city] = -1

# starting from 'Arad' startingCity = startingNode visited[startingCity] = True distance[startingCity] = 0 queue.put(startingCity)

while not queue.empty():

u = queue.get() # first element of the queue, here it will be 'arad' bfs\_traversal\_output.append(u)

# explore the adjust cities adj to 'arad' for v in romaniaMap[u]:

**if not visited[v]: visited[v] = True parent[v] = u**

**distance[v] = distance[u] + 1 queue.put(v)**

**# reaching our destination city i.e 'bucharest' g = destinationNode**

**path = []**

**while g is not None: path.append(g)**

**g = parent[g]**

**path.reverse()**

**# printing the path to our destination city print(path)**

**# Starting City & Destination City bfs('Arad', 'Bucharest')**



***AradtoBucharest***



***AradtoNeamt***



***ImplementedBreadthfirstsearchalgorithmforRomanianmapproblem.***



 ***ImplementIterativedeepdepthfirstsearchforRomanianmapproblem.***



**There are two common ways to traverse a graph, BFS and DFS. Considering a Tree (or Graph) of huge height and width, both BFS and DFS are not very efficient due to following reasons.**

1. **** **first traverses nodes going through one adjacent of root, then next adjacent. The problem with this approach is, if there is a node close to root, but not in first few subtrees explored by DFS, then DFS reaches that node very late. Also, DFS may not find shortest path to a node (in terms of number of edges).**
2. **** **goes level by level, but requires more space. The space required by DFS is O(d) where d is depth of tree, but space required by BFS is O(n) where n is number of nodes in tree (Why? Note that the last level of tree can have around n/2 nodes and second last level n/4 nodes and in BFS we need to have every level one by one in queue).**

 **combines depth-first search’s space-efficiency and breadth-first search’s fast search (for nodes closer to root).**



**IDDFS calls DFS for different depths starting from an initial value. In every call, DFS is restricted from going beyond given depth. So basically we do DFS in a BFS fashion.**



// Returns true if target is reachable from

// src within max\_depth

bool IDDFS(src, target, max\_depth) for limit from 0 to max\_depth

if DLS(src, target, limit) == true return true

return false

bool DLS(src, target, limit) if (src == target)

return true;

// If reached the maximum depth,

// stop recursing. if (limit <= 0)

return false;

foreach adjacent i of src if DLS(i, target, limit?1)

return true

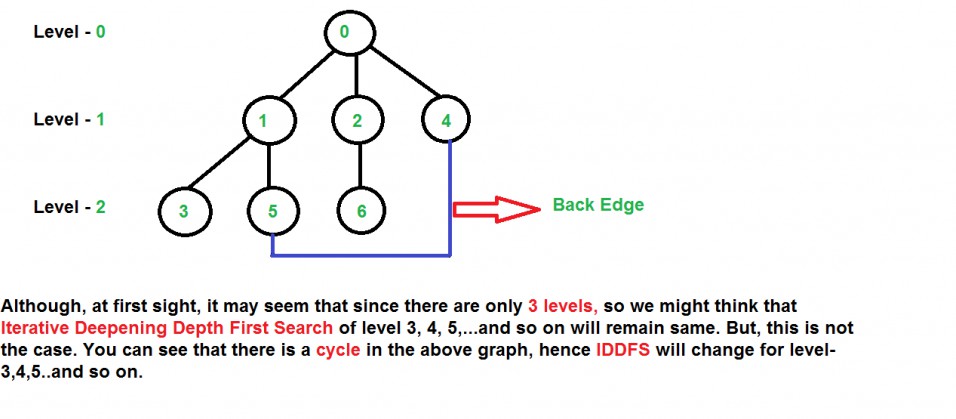
return false

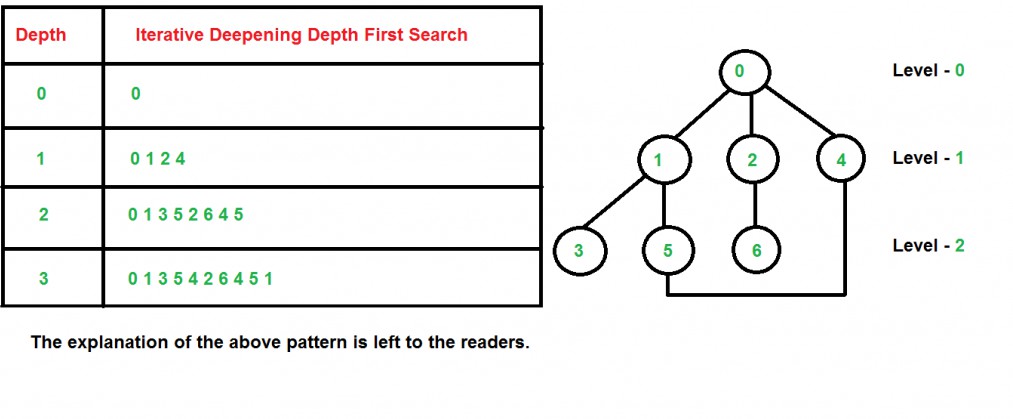
**An important thing to note is, we visit top level nodes multiple times. The last (or max depth) level is visited once, second last level is visited twice, and so**

**on. It may seem expensive, but it turns out to be not so costly, since in a tree most of the nodes are in the bottom level. So it does not matter much if the upper levels are visited multiple times.**

 **There can be two cases:**

1. **** **This case is simple. We can DFS multiple times with different height limits.**
2. **** **This is interesting as there is no visited flag in IDDFS.**





 **Suppose we have a tree having branching factor ‘b’ (number of children of each node), and its depth ‘d’, i.e., there are ** **nodes. In an iterative deepening search, the nodes on the bottom level are expanded once, those on the next to bottom level are expanded twice, and so on, up to the root of the search tree, which is expanded d+1 times. So the total number of expansions in an iterative deepening search is-**

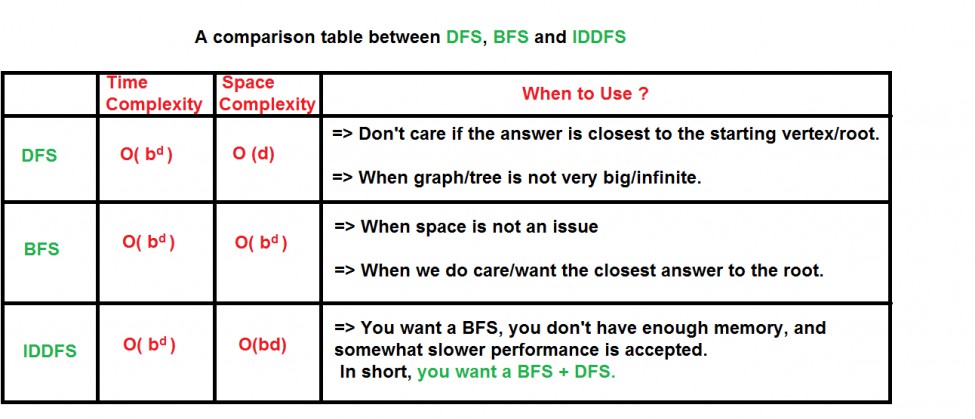
**(d)b + (d-1)b2 +** **+ 3bd-2 + 2bd-1 + bd**

**That is,**

**Summation[(d + 1 - i) bi], from i = 0 to i = d Which is same as O(bd)**

**After evaluating the above expression, we find that asymptotically IDDFS takes the same time as that of DFS and BFS, but it is indeed slower than both of them as it has a higher constant factor in its time complexity expression. IDDFS is best suited for a complete infinite tree**

**O(bd)**



**O(d)**

dict\_graph = {}

# Read the data.txt file with open('data.txt', 'r') as f:

for l in f:

city\_a, city\_b, p\_cost = l.split() if city\_a not in dict\_graph:

dict\_graph[city\_a] = {} dict\_graph[city\_a][city\_b] = int(p\_cost) if city\_b not in dict\_graph:

dict\_graph[city\_b] = {} dict\_graph[city\_b][city\_a] = int(p\_cost)

# Iterative Deepening Search Method def IterativeDeepening(graph, src, dst):

level = 0

count = 0

stack = [(src, [src], 0)] visited = {src}

while True: level += 1 while stack:

if count <= level: count = 0

(node, path, cost) = stack.pop() for temp in graph[node].keys():

if temp == dst:

return path + [temp], cost + graph[node][temp]

**else:**

**if temp not in visited: visited.add(temp) count += 1**

**stack.append((temp, path + [temp], cost + graph[node][temp]))**

**else:**

**q = stack visited\_bfs = {src} while q:**

**(node, path, cost) = q.pop(0) for temp in graph[node].keys():**

**if temp == dst:**

**return path + [temp], cost + graph[node][temp] else:**

**if temp not in visited\_bfs: visited\_bfs.add(temp)**

**q.append((temp, path + [temp], cost + graph[node][temp]))**

**break**

**print(dict\_graph) print("**

**")**

**#src = raw\_input("Enter the source:")**

**#dst = raw\_input("Enter the Destination: ") src = "Oradea"**

**dst = "Iasi" print("for ID")**

**print (IterativeDeepening(dict\_graph, src, dst))**

**data.txt**

Oradea Zerind 71

Oradea Sibiu 151

Zerind Arad 75

Arad Sibiu 140

Arad Timisoara 118

Timisoara Lugoj 111

Lugoj Mehadia 70

Mehadia Drobeta 75

Drobeta Craiova 120

Sibiu Rimnicu\_Vilcea 80

Sibiu Fagaras 99

Rimnicu\_Vilcea Piteshi 97

Rimnicu\_Vilcea Craiova 146

Craiova Piteshi 138

Piteshi Bucharest 101

Fagaras Bucharest 211

Bucharest Giurgiu 90

Bucharest Urziceni 85

Urziceni Hirsova 98

Urziceni Vaslui 142

Hirsova Eforie 86

Vaslui Iasi 92

Neamt Iasi 87



**Oradea to Iasi**



***ImplementedIterativedeepdepthfirstsearchforRomanianmapproblem.***



**: *ImplementA\*searchalgorithmforRomanianmapproblem.***



**What is A\* Search Algorithm?**

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



**Informally speaking, A\* Search algorithms, unlike other traversal techniques, it has “brains”. What it means is that it is really a smart algorithm which separates it from the other conventional algorithms. This fact is cleared in detail in below sections.**

**And it is also worth mentioning that many games and web-based maps use this algorithm to find the shortest path very efficiently (approximation).**

**Consider a square grid having many obstacles and we are given a starting cell and a target cell. We want to reach the target cell (if possible) from the starting cell as quickly as possible. Here A\* Search Algorithm comes to the rescue.**

**What A\* Search Algorithm does is that at each step it picks the node according to a value-‘f’ which is a parameter equal to the sum of two other parameters – ‘g’ and ‘h’. At each step it picks the node/cell having the lowest ‘f’, and process that node/cell.**

**We define ‘g’ and ‘h’ as simply as possible below**

**g = the movement cost to move from the starting point to a given square on the grid, following the path generated to get there.**

**h = the estimated movement cost to move from that given square on the grid to the final destination. This is often referred to as the heuristic, which is nothing but a kind of smart guess. We really don’t know the actual distance until we find the path, because all sorts of things can be in the way (walls, water, etc.). There can be many ways to calculate this ‘h’ which are discussed in the later sections.**



1. Initialize the open list
2. Initialize the closed list put the starting node on the open list (you can leave its f at zero)
3. while the open list is not empty
   1. find the node with the least f on the open list, call it "q"
   2. pop q off the open list
   3. generate q 8 successors and set their parents to q
   4. for each successor
      1. if successor is the goal, stop search
      2. else, compute both g and h for successor successor.g = q.g + distance between successor and q

successor.h = distance from goal to successor (This can be done using many ways, we will discuss three heuristics- Manhattan, Diagonal and Euclidean Heuristics)

successor.f = successor.g + successor.h

* + 1. if a node with the same position as successor is in the OPEN list which has a lower f than successor, skip this successor

iV) if a node with the same position as successor is in the CLOSED list which has a lower f than successor, skip this successor otherwise, add the node to the open list

end (for loop)

* 1. push q on the closed list end (while loop)

**So, suppose as in the below figure if we want to reach the target cell from the source cell, then the A\* Search algorithm would follow path as shown below. Note that the below figure is made by considering Euclidean Distance as a heuristic.**



**import heapq**

**class priorityQueue: def**  **init** **(self):**

**self.cities = []**

def push(self, city, cost): heapq.heappush(self.cities, (cost, city))

def pop(self):

return heapq.heappop(self.cities)[1]

def isEmpty(self):

if (self.cities == []): return True

else:

return False

def check(self): print(self.cities)

class ctNode:

def init (self, city, distance): self.city = str(city) self.distance = str(distance)

romania = {} def makedict():

file = open("romania.txt", 'r') for string in file:

line = string.split(',') ct1 = line[0]

ct2 = line[1]

dist = int(line[2])

romania.setdefault(ct1, []).append(ctNode(ct2, dist)) romania.setdefault(ct2, []).append(ctNode(ct1, dist))

def makehuristikdict(): h = {}

with open("romania\_sld.txt", 'r') as file: for line in file:

line = line.strip().split(",") node = line[0].strip()

sld = int(line[1].strip()) h[node] = sld

return h

def heuristic(node, values): return values[node]

def astar(start, end): path = {}

distance = {}

q = priorityQueue()

h = makehuristikdict()

q.push(start, 0)

distance[start] = 0 path[start] = None expandedList = []

while (q.isEmpty() == False): current = q.pop() expandedList.append(current)

if (current == end): break

for new in romania[current]:

g\_cost = distance[current] + int(new.distance)

# print(new.city, new.distance, "now : " + str(distance[current]), g\_cost) if (new.city not in distance or g\_cost < distance[new.city]):

distance[new.city] = g\_cost

f\_cost = g\_cost + heuristic(new.city, h) q.push(new.city, f\_cost)

path[new.city] = current

printoutput(start, end, path, distance, expandedList) def printoutput(start, end, path, distance, expandedlist):

finalpath = [] i = end

while (path.get(i) != None): finalpath.append(i)

i = path[i] finalpath.append(start) finalpath.reverse() print("Path From") print("\tArad => Bucharest")

print("=======================================================")

print("Exploreable cities : " + str(expandedlist))

print("The number of possible cities: " + str(len(expandedlist))) print("=======================================================")

print("The city that is passed the shortest distance: " + str(finalpath)) print("Number of cities passed: " + str(len(finalpath)))

print("Total Distance : " + str(distance[end]))

def main(): src = "Arad"

dst = "Bucharest" makedict() astar(src, dst)

if name == " main ": main()



**Arad, 366**

**Bucharest, 0**

**Craiova, 160**

**Dobreta, 242**

**Eforie, 161**

**Fagaras, 176**

**Giurgiu, 77**

**Hirsowa, 151**

**Lasi, 226**

**Lugoj, 244**

**Mehadia, 241**

**Neamt, 234**

**Oradea, 380**

**Pitesti, 100**

**Rimnicu Vilcea, 193**

**Sibiu, 253**

**Timisoara, 329**

**Urziceni, 80**

**Vaslui, 199**

**Zerind, 374**



Arad,Zerind, 75

Arad,Sibiu, 140

Arad,Timisoara, 118

Zerind,Oradea, 71

Oradea,Sibiu, 151

Timisoara,Lugoj, 111

Sibiu,Fagaras, 99

Sibiu,Rimnicu Vilcea, 80

Lugoj,Mehadia, 70

Fagaras,Bucharest, 211

Rimnicu Vilcea,Pitesti, 97

Rimnicu Vilcea,Craiova, 146

Mehadia,Dobreta, 75

Bucharest,Pitesti, 101

Bucharest,Urziceni, 85

Bucharest,Giurglu, 90

Pitesti,Craiova, 138

Craiova,Dobreta, 120

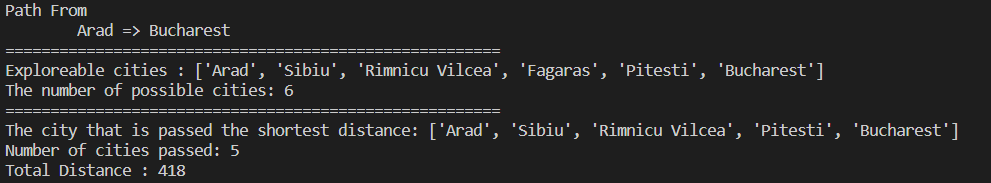
Urziceni,Hirsova, 98

Urziceni,Vaslui, 142

Hirsova,Eforie, 86

Vaslui,Lasi, 92

Lasi,Neamt, 87



***ImplementedA\*searchalgorithmforRomanianmapproblem.***



 ***Implementrecursivebest-firstsearchalgorithmforRomanianmapproblem.***



**It is simple recursive algorithm that resembles the operation of standard best first search but uses only linear space. It is similar to recursive DFS and differs from Recursive DFS as follows,**

**It keeps track of the f value of the best alternative path available from any ancestor of the current node. Instead of continuing indefinitely down the current path.**

**{**

function RECURSIVE\_BEST\_FIRST\_SEARCH (Problem) returns a solution, or failure

RBFS (problem, MAKE\_NODE (INITIAL\_STATE problem), Ұ)

function RBFS (Problem, Node, f-limit) returns a solution or failure and a new f-cost limit if GOAL\_TEST [Problem] [State] then return node

Successors ← EXPAND (node, problem)

if successors is empty then return failure, ¥

for each S in successors do

f(s) ← max (g(s) + h(s), f[node]) repeat

best ← the lowest f value node in successors if [best] > f-limit then return failure, f[best]

alternative ← the second lowest f-value among successors result, f[best] ← BFS (Problem, best, min (f-limit, alternative)

if result # failure then return result

**}**



* ***MoreefficientthanIDA\****
* ***Itisanoptimalalgorithmifh(n)isadmissible***
* ***SpacecomplexityisO(bd).***



* ***Itsuffersfromexcessivenoderegeneration.***
* ***Itstimecomplexityisdifficulttocharacterizebecauseitdependsonthe accuracyofh(n)andhowoftenthebestpathchangesasthenodesare expanded.***



dict\_hn={'Arad':336,'Bucharest':0,'Craiova':160,'Drobeta':242,'Eforie':161, 'Fagaras':176,'Giurgiu':77,'Hirsova':151,'Iasi':226,'Lugoj':244, 'Mehadia':241,'Neamt':234,'Oradea':380,'Pitesti':100,'Rimnicu':193, 'Sibiu':253,'Timisoara':329,'Urziceni':80,'Vaslui':199,'Zerind':374}

dict\_gn=dict( Arad=dict(Zerind=75,Timisoara=118,Sibiu=140),

Bucharest=dict(Urziceni=85,Giurgiu=90,Pitesti=101,Fagaras=211), Craiova=dict(Drobeta=120,Pitesti=138,Rimnicu=146), Drobeta=dict(Mehadia=75,Craiova=120),

Eforie=dict(Hirsova=86), Fagaras=dict(Sibiu=99,Bucharest=211), Giurgiu=dict(Bucharest=90), Hirsova=dict(Eforie=86,Urziceni=98), Iasi=dict(Neamt=87,Vaslui=92), Lugoj=dict(Mehadia=70,Timisoara=111), Mehadia=dict(Lugoj=70,Drobeta=75), Neamt=dict(Iasi=87), Oradea=dict(Zerind=71,Sibiu=151), Pitesti=dict(Rimnicu=97,Bucharest=101,Craiova=138), Rimnicu=dict(Sibiu=80,Pitesti=97,Craiova=146),

Sibiu=dict(Rimnicu=80,Fagaras=99,Arad=140,Oradea=151), Timisoara=dict(Lugoj=111,Arad=118), Urziceni=dict(Bucharest=85,Hirsova=98,Vaslui=142), Vaslui=dict(Iasi=92,Urziceni=142), Zerind=dict(Oradea=71,Arad=75)

**)**

import queue as Q

start='Arad' goal='Bucharest' result=''

def get\_fn(citystr): cities=citystr.split(',') hn=gn=0

for ctr in range(0,len(cities)-1): gn=gn+dict\_gn[cities[ctr]][cities[ctr+1]]

hn=dict\_hn[cities[len(cities)-1]] return(hn+gn)

def printout(cityq):

for i in range(0,cityq.qsize()): print(cityq.queue[i])

def expand(cityq): global result

tot,citystr,thiscity=cityq.get() nexttot=999

if not cityq.empty():

**nexttot,nextcitystr,nextthiscity=cityq.queue[0] if thiscity==goal and tot<nexttot:**

**result=citystr+'::'+str(tot) return**

**print("Expanded city** **",thiscity)**

**print("Second best f(n)** **",nexttot)**

**tempq=Q.PriorityQueue() for cty in dict\_gn[thiscity]:**

**tempq.put((get\_fn(citystr+','+cty),citystr+','+cty,cty)) for ctr in range(1,3):**

**ctrtot,ctrcitystr,ctrthiscity=tempq.get() if ctrtot<nexttot:**

**cityq.put((ctrtot,ctrcitystr,ctrthiscity)) else:**

**cityq.put((ctrtot,citystr,thiscity)) break**

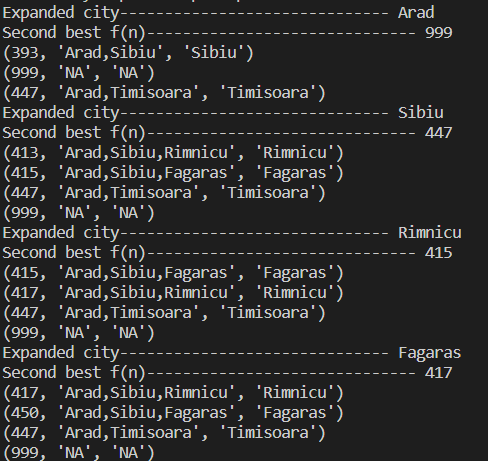
**printout(cityq) expand(cityq)**

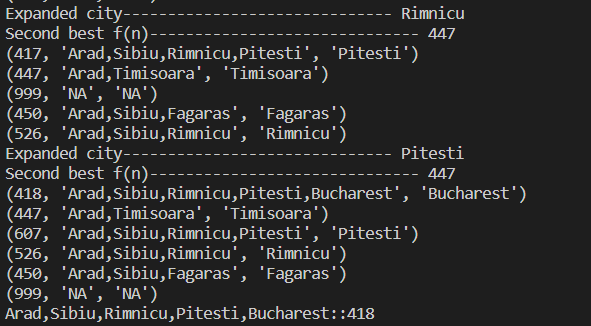
**def main(): cityq=Q.PriorityQueue() thiscity=start cityq.put((999,"NA","NA"))**

**cityq.put((get\_fn(start),start,thiscity)) expand(cityq)**

**print(result) main()**







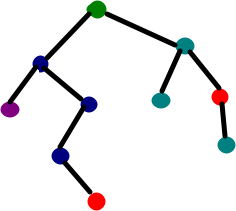
***Implementedrecursivebest-firstsearchalgorithmforRomanianmapproblem.***



 ***Implementdecisiontreelearningalgorithmfortherestaurantwaitingproblem.***



***DecisionTreeisoneofthemostpowerfulandpopularalgorithms.Decision- treealgorithmfallsunderthecategoryofsupervisedlearningalgorithms.Itworksfor bothcontinuousaswellascategoricaloutputvariables.***

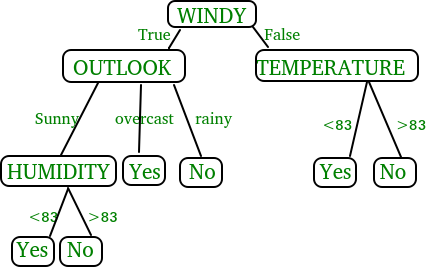




* ***Atthebeginning,weconsiderthewholetrainingsetastheroot.***
* ***Attributesareassumedtobecategoricalforinformationgainandforgini index,attributesareassumedtobecontinuous.***
* ***Onthebasisofattributevaluesrecordsaredistributedrecursively.***
* ***Weusestatisticalmethodsfororderingattributesasrootorinternalnode.***



1. ***Findthebestattributeandplaceitontherootnodeofthetree.***
2. ***Now,splitthetrainingsetofthedatasetintosubsets.Whilemakingthe subsetmakesurethateachsubsetoftrainingdatasetshouldhavethe samevalueforanattribute.***
3. ***Findleafnodesinallbranchesbyrepeating1and2oneachsubset.***



***Whileimplementingthedecisiontreewewillgothroughthefollowingtwophases:***

1. ***BuildingPhase***
   * ***Preprocessthedataset.***
   * ***SplitthedatasetfromtrainandtestusingPythonsklearn package.***
   * ***Traintheclassifier.***
2. ***OperationalPhase***
   * ***Makepredictions.***
   * ***Calculatetheaccuracy.***



* ***Toimportandmanipulatethedataweareusingthepandaspackageprovided inpython.***
* ***Here,weareusingaURLwhichisdirectlyfetchingthedatasetfromtheUCI sitenoneedtodownloadthedataset.Whenyoutrytorunthiscodeonyour systemmakesurethesystemshouldhaveanactiveInternetconnection.***
* ***Asthedatasetisseparatedby“,”sowehavetopassthesepparameter’s valueas“,”.***
* ***Anotherthingisnoticeisthatthedatasetdoesn’tcontaintheheadersowe willpasstheHeaderparameter’svalueasnone.Ifwewillnotpasstheheader parameterthenitwillconsiderthefirstlineofthedatasetastheheader.***



* ***Beforetrainingthemodelwehavetosplitthedatasetintothetrainingand testingdataset.***
* ***Tosplitthedatasetfortrainingandtestingweareusingthesklearn moduletrain\_test\_split***
* ***Firstofallwehavetoseparatethetargetvariablefromtheattributesinthe dataset.***

X = balance\_data.values[:, 1:5] Y = balance\_data.values[:,0]

* ***Abovearethelinesfromthecodewhichseparatethedataset.ThevariableX containstheattributeswhilethevariableYcontainsthetargetvariableofthe dataset.***
* ***Nextstepistosplitthedatasetfortrainingandtestingpurpose.***

X\_train, X\_test, y\_train, y\_test = train\_test\_split( X, Y, test\_size = 0.3, random\_state = 100)

* ***Abovelinesplitthedatasetfortrainingandtesting.Aswearesplittingthe datasetinaratioof70:30betweentrainingandtestingsoweare passtest\_sizeparameter’svalueas0.3.***
* ***random\_statevariableisapseudo-randomnumbergeneratorstateusedfor randomsampling.***



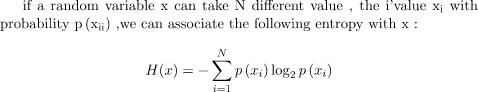
***Giniindexandinformationgainbothofthesemethodsareusedtoselectfrom thenattributesofthedatasetwhichattributewouldbeplacedattherootnodeorthe***

***internalnode.***





* ***GiniIndexisametrictomeasurehowoftenarandomlychosenelement wouldbeincorrectlyidentified.***
* ***Itmeansanattributewithlowerginiindexshouldbepreferred.***
* ***Sklearnsupports“gini”criteriaforGiniIndexandbydefault,ittakes“gini” value.***



* ***Entropyisthemeasureofuncertaintyofarandomvariable,itcharacterizes theimpurityofanarbitrarycollectionofexamples.Thehighertheentropythe moretheinformationcontent.***



* ***Theentropytypicallychangeswhenweuseanodeinadecisiontreeto partitionthetraininginstancesintosmallersubsets.Informationgainisa measureofthischangeinentropy.***
* ***Sklearnsupports“entropy”criteriaforInformationGainandifwewanttouse InformationGainmethodinsklearnthenwehavetomentionitexplicitly.***



* ***Accuracyscoreisusedtocalculatetheaccuracyofthetrainedclassifier.***



* ***ConfusionMatrixisusedtounderstandthetrainedclassifierbehaviorover thetestdatasetorvalidatedataset.***



**import numpy as np import pandas as pd import sklearn as sk**

**from sklearn.metrics import confusion\_matrix**

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

from sklearn.tree import DecisionTreeClassifier from sklearn.metrics import accuracy\_score from sklearn.metrics import classification\_report

#func importing dataset def importdata():

balance\_data=pd.read\_csv("balance-scale.data")

#print the dataset shape

print("Dataset Length : ",len(balance\_data))

#printing the dataset observations print("Dataset : ",balance\_data.head()) return balance\_data

#func to split the dataset

def splitdataset(balance\_data): #seperating the target variable X=balance\_data.values[:,1:5] Y=balance\_data.values[:,0]

#splitting the dataset into train and test X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,Y,test\_size=0.3,random\_state=100) return X,Y,X\_train,X\_test,y\_train,y\_test

#function to perform training with entropy

def train\_using\_entropy(X\_train,X\_test,y\_train,y\_test): #decision tree with entropy

clf\_entropy=DecisionTreeClassifier(criterion="entropy",random\_state=100,max\_depth=3,min\_sa mples\_leaf=5)

#performing training clf\_entropy.fit(X\_train,y\_train) return clf\_entropy

def prediction(X\_test,clf\_object): y\_pred=clf\_object.predict(X\_test) print("Predicted Values : ") print(y\_pred)

return y\_pred

def cal\_accuracy(y\_test,y\_pred):

print("Accuracy : ",accuracy\_score(y\_test,y\_pred)\*100)

def main():

data=importdata() X,Y,X\_train,X\_test,y\_train,y\_test=splitdataset(data)

clf\_entropy=train\_using\_entropy(X\_train,X\_test,y\_train,y\_test)

print("Results using entropy : ") y\_pred\_entropy=prediction(X\_test,clf\_entropy) cal\_accuracy(y\_test,y\_pred\_entropy)

main()



B,1,1,1,1

R,1,1,1,2

R,1,1,1,3

R,1,1,1,4

R,1,1,1,5

R,1,1,2,1

R,1,1,2,2

**.**

**.**

.L,5,5,4,5

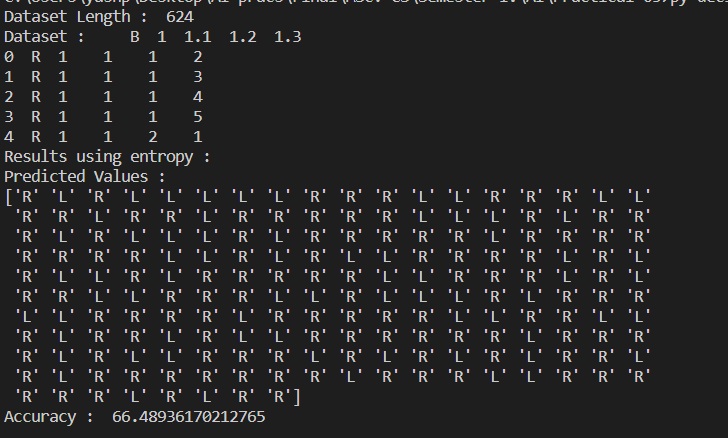
L,5,5,5,1

L,5,5,5,2

L,5,5,5,3

L,5,5,5,4

B,5,5,5,5



***Implementeddecisiontreelearningalgorithmfortherestaurantwaitingproblem.***



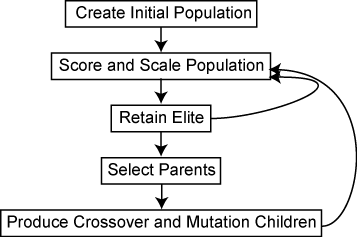
 ***ImplementGeneticAlgorithmsforStaffPlanning.***



**The genetic algorithm is a method for solving both constrained and unconstrained optimization problems that is based on natural selection, the process that drives biological evolution. The genetic algorithm repeatedly modifies a population of individual solutions. At each step, the genetic algorithm selects individuals from the current population to be parents and uses them to produce the children for the next generation. Over successive generations, the population "evolves" toward an optimal solution. You can apply the genetic algorithm to solve a variety of optimization problems that are not well suited for standard optimization algorithms, including problems in which the objective function is discontinuous, nondifferentiable, stochastic, or highly nonlinear. The genetic algorithm can address problems**

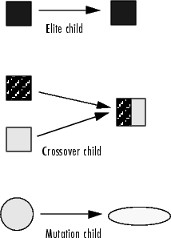
**of *mixedintegerprogramming*, where some components are restricted to be integer- valued.**

**This flow chart outlines the main algorithmic steps. For details, see How the Genetic Algorithm Works.**



**The genetic algorithm uses three main types of rules at each step to create the next generation from the current population:**

* ***Selectionrules*select the individuals, called *parents*, that contribute to the population at the next generation. The selection is generally stochastic, and can depend on the individuals' scores.**
* ***Crossoverrules*combine two parents to form children for the next generation.**
* ***Mutationrules*apply random changes to individual parents to form children.**





import numpy as np import pandas as pd staff\_planning = [

[[0, 0, 10],[1, 0, 10],[2, 0, 10],[3, 0, 10],[4, 0, 10],[5, 0, 10],[6, 0, 10],[7, 0, 10],[8, 0, 10],[9, 0, 10],[10, 0,

10]],

[[0, 0, 10],[1, 0, 10],[2, 0, 10],[3, 0, 10],[4, 0, 10],[5, 0, 10],[6, 0, 10],[7, 0, 10],[8, 0, 10],[9, 0, 10],[10, 0,

10]],

[[0, 0, 10],[1, 0, 10],[2, 0, 10],[3, 0, 10],[4, 0, 10],[5, 0, 10],[6, 0, 10],[7, 0, 10],[8, 0, 10],[9, 0, 10],[10, 0,

10]],

[[0, 0, 10],[1, 0, 10],[2, 0, 10],[3, 0, 10],[4, 0, 10],[5, 0, 10],[6, 0, 10],[7, 0, 10],[8, 0, 10],[9, 0, 10],[10, 0,

10]],

[[0, 0, 10],[1, 0, 10],[2, 0, 10],[3, 0, 10],[4, 0, 10],[5, 0, 10],[6, 0, 10],[7, 0, 10],[8, 0, 10],[9, 0, 10],[10, 0,

10]]

**]**

hourlystaff\_needed = np.array([

[0, 0, 0, 0, 0, 0, 4, 4, 4, 2, 2, 2, 6, 6, 2, 2, 2, 6, 6, 6, 2, 2, 2, 2],

[0, 0, 0, 0, 0, 0, 4, 4, 4, 2, 2, 2, 6, 6, 2, 2, 2, 6, 6, 6, 2, 2, 2, 2],

[0, 0, 0, 0, 0, 0, 4, 4, 4, 2, 2, 2, 6, 6, 2, 2, 2, 6, 6, 6, 2, 2, 2, 2],

[0, 0, 0, 0, 0, 0, 4, 4, 4, 2, 2, 2, 6, 6, 2, 2, 2, 6, 6, 6, 2, 2, 2, 2],

[0, 0, 0, 0, 0, 0, 4, 4, 4, 2, 2, 2, 6, 6, 2, 2, 2, 6, 6, 6, 2, 2, 2, 2]

**])**

**"""**

Employee Present: analyse whether the employee is present yes or no on a given time Based on the employee list of 3 (id, start time, duration)

**"""**

def employee\_present(employee, time):

**"""**

employee\_start\_time = employee[1] employee\_duration = employee[2]

employee\_end\_time = employee\_start\_time + employee\_duration if (time >= employee\_start\_time) and (time < employee\_end\_time):

return True return False

convert a staff planning to a staff-needed plannig

The employee planning is organised per employee, the staff-needed planning is the number of employees working per hour

The staff-needed planning is based on the employee planning and will allow to calculate the difference with the staff-needed

It doesnt work overnight, but our shop isnt open at night anyway """

def staffplanning\_to\_hourlyplanning(staff\_planning):

hourlystaff\_week = []

for day in staff\_planning:

hourlystaff\_day = [] for employee in day:

employee\_present\_hour = [] for time in range(0, 24):

employee\_present\_hour.append(employee\_present(employee, time)) hourlystaff\_day.append(employee\_present\_hour)

hourlystaff\_week.append(hourlystaff\_day)

hourlystaff\_week = np.array(hourlystaff\_week).sum(axis = 1) return hourlystaff\_week

**"""**

the cost is calculated as hours understaffed + hours overstaffed """

def cost(hourlystaff, hourlystaff\_needed): errors = hourlystaff - hourlystaff\_needed overstaff = abs(errors[errors > 0].sum()) understaff = abs(errors[errors < 0].sum())

overstaff\_cost = 1

understaff\_cost = 1

cost = overstaff\_cost \* overstaff + understaff\_cost \* understaff return cost

**"""**

generate an entirely random staff planning for a certain number of days

start time is random between 0 and 23; duration is random between 0 and 10 """

def generate\_random\_staff\_planning(n\_days, n\_staff): period\_planning = []

for day in range(n\_days): day\_planning = []

for employee\_id in range(n\_staff): start\_time = np.random.randint(0, 23) duration = np.random.randint(0, 10)

employee = [employee\_id, start\_time, duration] day\_planning.append(employee)

period\_planning.append(day\_planning) return period\_planning

# An example of the code until here

# show the random initialization of the week planning

random\_staff\_planning = generate\_random\_staff\_planning(n\_days = 5, n\_staff = 11) random\_staff\_planning

# show the cost of this random week planning cost(staffplanning\_to\_hourlyplanning(random\_staff\_planning), hourlystaff\_needed)

**"""**

create a parent generation of n parent plannings """

def create\_parent\_generation(n\_parents, n\_days = 7, n\_staff = 11): parents = []

for i in range(n\_parents):

parent = generate\_random\_staff\_planning(n\_days = n\_days, n\_staff = n\_staff) parents.append(parent)

return parents

**"""**

for each iteration, select randomly two parents and make a random combination of those two parents

by applying a randomly generated yes/no mask to the two selected parents """

def random\_combine(parents, n\_offspring): n\_parents = len(parents)

n\_periods = len(parents[0]) n\_employees = len(parents[0][0])

offspring = []

for i in range(n\_offspring):

random\_dad = parents[np.random.randint(low = 0, high = n\_parents - 1)] random\_mom = parents[np.random.randint(low = 0, high = n\_parents - 1)]

dad\_mask = np.random.randint(0, 2, size = np.array(random\_dad).shape) mom\_mask = np.logical\_not(dad\_mask)

child = np.add(np.multiply(random\_dad, dad\_mask), np.multiply(random\_mom, mom\_mask)) offspring.append(child)

return offspring

def mutate\_parent(parent, n\_mutations):

size1 = parent.shape[0] size2 = parent.shape[1]

for i in range(n\_mutations):

rand1 = np.random.randint(0, size1) rand2 = np.random.randint(0, size2) rand3 = np.random.randint(1, 2)

parent[rand1,rand2,rand3] = np.random.randint(0, 10) return parent

def mutate\_gen(parent\_gen, n\_mutations): mutated\_parent\_gen = []

for parent in parent\_gen: mutated\_parent\_gen.append(mutate\_parent(parent, n\_mutations))

return mutated\_parent\_gen

def is\_acceptable(parent):

return np.logical\_not((np.array(parent)[:,:,2:] > 10).any()) #work more than 10 hours is not ok

def select\_acceptable(parent\_gen):

parent\_gen = [parent for parent in parent\_gen if is\_acceptable(parent)] return parent\_gen

def select\_best(parent\_gen, hourlystaff\_needed, n\_best): costs = []

for idx, parent\_staff\_planning in enumerate(parent\_gen):

parent\_hourly\_planning = staffplanning\_to\_hourlyplanning(parent\_staff\_planning) parent\_cost = cost(parent\_hourly\_planning, hourlystaff\_needed) costs.append([idx, parent\_cost])

print('generations best is: {}, generations worst is: {}'.format(pd.DataFrame(costs)[1].min(), pd.DataFrame(costs)[1].max()))

costs\_tmp = pd.DataFrame(costs).sort\_values(by = 1, ascending = True).reset\_index(drop=True)

selected\_parents\_idx = list(costs\_tmp.iloc[:n\_best,0])

selected\_parents = [parent for idx, parent in enumerate(parent\_gen) if idx in selected\_parents\_idx]

return selected\_parents

**"""**

the overall function """

def gen\_algo(hourlystaff\_needed, n\_iterations): generation\_size = 500

parent\_gen = create\_parent\_generation(n\_parents = generation\_size, n\_days = 5, n\_staff = 11) for it in range(n\_iterations):

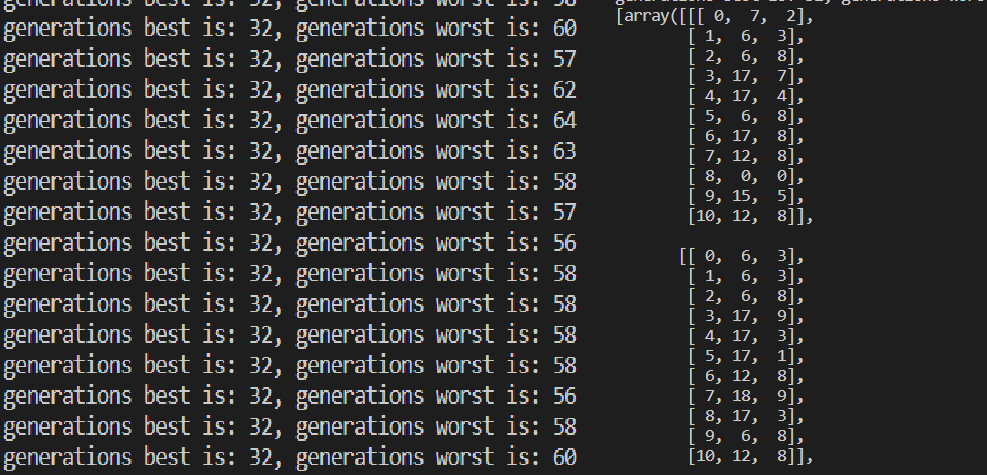
parent\_gen = select\_acceptable(parent\_gen)

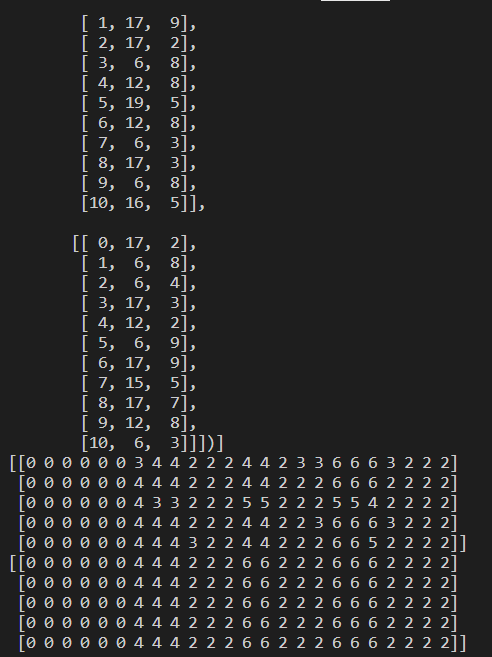
parent\_gen = select\_best(parent\_gen, hourlystaff\_needed, n\_best = 100)

**parent\_gen = random\_combine(parent\_gen, n\_offspring = generation\_size) parent\_gen = mutate\_gen(parent\_gen, n\_mutations = 1)**

**best\_child = select\_best(parent\_gen, hourlystaff\_needed, n\_best = 1) return best\_child**

**best\_planning = gen\_algo(hourlystaff\_needed, n\_iterations = 100) print(best\_planning) print(staffplanning\_to\_hourlyplanning(best\_planning[0])) print(hourlystaff\_needed)**





***ImplementedGeneticAlgorithmsforStaffPlanning.***



 ***ImplementANN.***



**Artificial Neural Network Tutorial provides basic and advanced concepts of ANNs. Our Artificial Neural Network tutorial is developed for beginners as well as professions.**

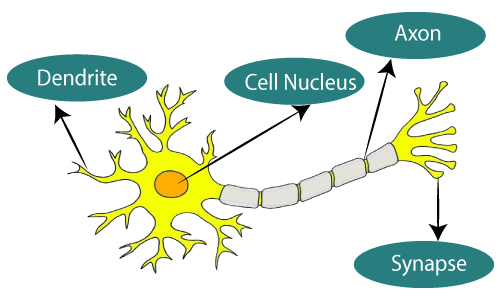
**The term "Artificial neural network" refers to a biologically inspired sub-field of artificial intelligence modeled after the brain. An Artificial neural network is usually a computational network based on biological neural networks that construct the structure of the human brain. Similar to a human brain has neurons interconnected to each other, artificial neural networks also have neurons that are linked to each other in various layers of the networks. These neurons are known as nodes.**

**Artificial neural network tutorial covers all the aspects related to the artificial neural network. In this tutorial, we will discuss ANNs, Adaptive resonance theory, Kohonen self-organizing map, Building blocks, unsupervised learning, Genetic algorithm, etc.**



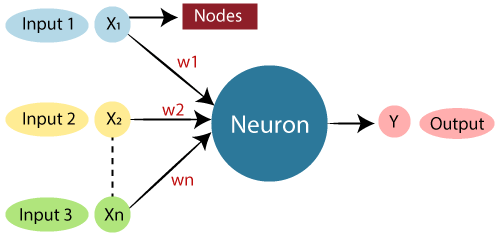


**The term "**  **" is derived from Biological neural networks that develop the structure of a human brain. Similar to the human brain that has neurons interconnected to one another, artificial neural networks also have neurons that are interconnected to one another in various layers of the networks. These neurons are known as nodes.**









**Dendrites from Biological Neural Network represent inputs in Artificial Neural Networks, cell nucleus represents Nodes, synapse represents Weights, and Axon represents Output.**

**Relationship between Biological neural network and artificial neural network:**



|  |  |  |  |
| --- | --- | --- | --- |
|  | | | |
| **Dendrites** | | **Inputs** | |
| **Cell nucleus** | | **Nodes** | |
| **Synapse** | | **Weights** | |
| **Axon** | | **Output** | |
| **An** | **in the field of** | |  |

  **where it attempts to mimic the network of neurons makes up a human brain so that computers will have an option to understand things and make decisions in a human-like manner. The artificial neural network is designed by programming computers to behave simply like interconnected brain cells.**

**There are around 1000 billion neurons in the human brain. Each neuron has an association point somewhere in the range of 1,000 and 100,000. In the human brain, data is stored in such a manner as to be distributed, and we can extract more than one piece of this data when necessary from our memory parallelly. We can say that the human brain is made up of incredibly amazing parallel processors.**

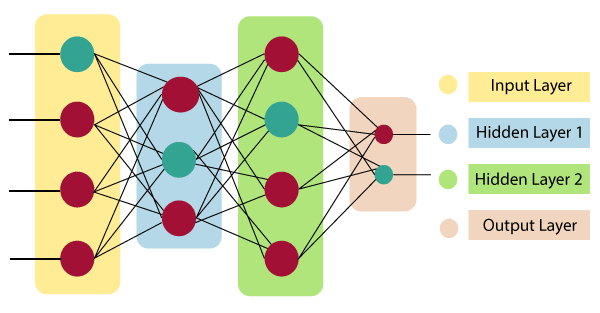
**We can understand the artificial neural network with an example, consider an example of a digital logic gate that takes an input and gives an output. "OR" gate,**

**which takes two inputs. If one or both the inputs are "On," then we get "On" in output. If both the inputs are "Off," then we get "Off" in output. Here the output depends upon input. Our brain does not perform the same task. The outputs to inputs relationship keep changing because of the neurons in our brain, which are "learning."**



**To understand the concept of the architecture of an artificial neural network, we have to understand what a neural network consists of. In order to define a neural network that consists of a large number of artificial neurons, which are termed units arranged in a sequence of layers. Lets us look at various types of layers available in an artificial neural network.**

**Artificial Neural Network primarily consists of three layers:**







**As the name suggests, it accepts inputs in several different formats provided by the programmer.**





**The hidden layer presents in-between input and output layers. It performs all the calculations to find hidden features and patterns.**



**The input goes through a series of transformations using the hidden layer, which finally results in output that is conveyed using this layer.**

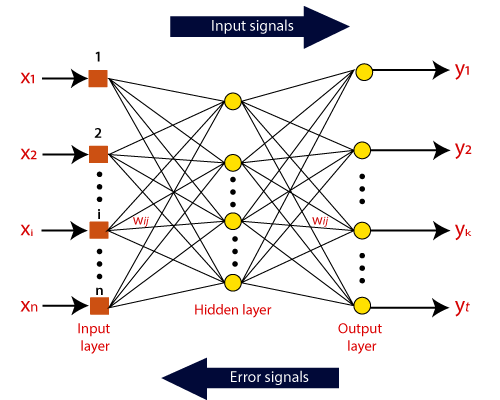
**The artificial neural network takes input and computes the weighted sum of the inputs and includes a bias. This computation is represented in the form of a transfer function.**



**It determines weighted total is passed as an input to an activation function to produce the output. Activation functions choose whether a node should fire or not. Only those who are fired make it to the output layer. There are distinctive activation functions available that can be applied upon the sort of task we are performing.**



**Artificial Neural Network can be best represented as a weighted directed graph, where the artificial neurons form the nodes. The association between the neurons outputs and neuron inputs can be viewed as the directed edges with weights. The Artificial Neural Network receives the input signal from the external source in the form of a pattern and image in the form of a vector. These inputs are then mathematically assigned by the notations x(n) for every n number of inputs.**



**Afterward, each of the input is multiplied by its corresponding weights ( these weights are the details utilized by the artificial neural networks to solve a specific problem ). In general terms, these weights normally represent the strength of the interconnection between neurons inside the artificial neural network. All the weighted inputs are summarized inside the computing unit.**

**If the weighted sum is equal to zero, then bias is added to make the output non-zero or something else to scale up to the system's response. Bias has the same input,**

**and weight equals to 1. Here the total of weighted inputs can be in the range of 0 to positive infinity. Here, to keep the response in the limits of the desired value, a certain maximum value is benchmarked, and the total of weighted inputs is passed through the activation function.**

**The activation function refers to the set of transfer functions used to achieve the desired output. There is a different kind of the activation function, but primarily either linear or non-linear sets of functions. Some of the commonly used sets of activation functions are the Binary, linear, and Tan hyperbolic sigmoidal activation functions.**

**Let us take a look at each of them in details:**



**In binary activation function, the output is either a one or a 0. Here, to accomplish this, there is a threshold value set up. If the net weighted input of neurons is more than 1, then the final output of the activation function is returned as one or else the output is returned as 0.**



**The Sigmoidal Hyperbola function is generally seen as an "" shaped curve. Here the tan hyperbolic function is used to approximate output from the actual net input. The function is defined as:**





**Where ???? is considered the Steepness parameters.**



**import numpy as np import torch**

**import torchvision**

**from torch.utils.data import DataLoader**

**import layers import loss import optimizers**

**from model import Model**

**def get\_dataset(batch\_size): train\_loader = DataLoader(**

**torchvision.datasets.MNIST('./data/', train=True, download=True, transform=torchvision.transforms.Compose([**

**torchvision.transforms.ToTensor(),**

**])), shuffle=True, batch\_size=batch\_size)**

**test\_loader = DataLoader(**

**torchvision.datasets.MNIST('./data/', train=False, download=True, transform=torchvision.transforms.Compose([**

**torchvision.transforms.ToTensor(),**

])), shuffle=True, batch\_size=batch\_size) return train\_loader, test\_loader

if name == ' main ': torch.random.manual\_seed(1234) np.random.seed(1234)

epochs = 10

lr = 0.01

batch\_size = 32

optimizer = optimizers.SGD(learning\_rate=lr) criterion = loss.CrossEntropy()

layers = [ layers.LinearLayer(784, 512), layers.ReLU(),

layers.Dropout(keep\_rate=0.8), layers.LinearLayer(512, 512), layers.ReLU(), layers.Dropout(keep\_rate=0.8), layers.LinearLayer(512, 10)

**]**

model = Model(layers, optimizer, criterion)

train\_loader, test\_loader = get\_dataset(batch\_size) for epoch\_id in range(epochs):

model.train() total = 0

correct = 0

for i, (x, y) in enumerate(train\_loader):

x = x.numpy().reshape(y.shape[0], -1, 1) y = y.numpy()

model.optimizer.zero\_grad()

loss, pred, logits = model.forward(x, y) model.backward(y, logits)

correct += np.sum(y == pred.flatten()) total += y.shape[0]

if i % 100 == 0:

print("Loss:", loss.mean())

print("Accuracy (train) epoch {}: {} %".format(epoch\_id + 1, correct / total \* 100.0))

model.eval() total = 0

correct = 0

for i, (x, y) in enumerate(test\_loader):

x = x.numpy().reshape(y.shape[0], -1, 1) y = y.numpy()

\_, pred, \_ = model.forward(x, y)

correct += np.sum(y == pred.flatten()) total += y.shape[0]

**print("Accuracy (test) epoch {}: {} %".format(epoch\_id + 1, correct / total \* 100.0))**

**total = 0**

**correct = 0**

**for i, (x, y) in enumerate(train\_loader):**

**x = x.numpy().reshape(y.shape[0], -1, 1) y = y.numpy()**

**\_, pred, \_ = model.forward(x, y)**

**correct += np.sum(y == pred.flatten()) total += y.shape[0]**

**print("Accuracy final (train) epoch {}: {} %".format(epochs, correct / total \* 100.0))**

**from typing import Tuple, List, Dict, Any import numpy as np**

**class Param:**

**def**  **init** **(self, data: np.ndarray): self.data = data**

**class Layer:**

**def**  **init** **(self, train: bool = True): """Creates layer.**

**:param train: bool deciding whether the layer is in train/eval mode """**

**self.train = train**

**def forward(self, x: np.ndarray) -> Tuple[np.ndarray, List[Any], Dict[str, Any]]: """Forward pass.**

**:param x: input of the layer**

**:return: output of the layer, \*args and \*\*kwargs as a tuple**

**Args and kwargs are passed as arguments for backward pass. """**

**pass**

**def backward(self, x: np.ndarray, dy: np.ndarray, \*args, \*\*kwargs) -> Tuple[np.ndarray, List[np.ndarray]]:**

**"""Backward pass.**

**:param x: the layer input**

**:param dy: upstream gradient**

**:param args: optional args**

**:param kwargs: optional kwargs**

**:return: tuple of downstream gradient, then list of gradients with respect to parameters (in order)**

defined in weights method """

pass

def weights(self) -> List[Param]:

"""Learnable parameters of the layer - order must be the same as in backward.""" return []

def xavier\_uniform\_init(input\_dim, output\_dim, gain: float = 1.0): r = gain \* np.sqrt(6.0 / (input\_dim + output\_dim))

return np.random.uniform(-r, r, (input\_dim, output\_dim)) class LinearLayer(Layer):

def init (self, input\_dim, output\_dim): super(). init ()

self.W = Param(xavier\_uniform\_init(input\_dim, output\_dim)) self.b = Param(np.zeros(shape=(1, output\_dim))) self.output\_dim = output\_dim

def forward(self, x: np.ndarray) -> Tuple[np.ndarray, List[Any], Dict[str, Any]]:

y = (np.matmul(x.transpose((0, 2, 1)), self.W.data) + self.b.data).transpose((0, 2, 1)) assert y.shape == (x.shape[0], self.output\_dim, 1)

return y, [], {}

def backward(self, x: np.ndarray, dy: np.ndarray, \*args, \*\*kwargs) -> Tuple[np.ndarray, List[np.ndarray]]:

batch\_size = x.shape[0]

dx = dy.transpose((0, 2, 1)).dot(self.W.data.T).transpose((0, 2, 1)) assert dx.shape == x.shape

dW = np.matmul(dy, x.transpose((0, 2, 1))).transpose((0, 2, 1)) assert dW.shape == (batch\_size, \*self.W.data.shape)

db = dy.transpose((0, 2, 1))

assert db.shape == (batch\_size, \*self.b.data.shape) return dx, [dW, db]

def weights(self) -> List[Param]: return [self.W, self.b]

class ReLU(Layer):

def forward(self, x: np.ndarray) -> Tuple[np.ndarray, List[Any], Dict[str, Any]]: return np.maximum(x, 0), [], {}

def backward(self, x: np.ndarray, dy: np.ndarray, \*args, \*\*kwargs) -> Tuple[np.ndarray, List[np.ndarray]]:

dx = np.maximum(x, 0) \* dy assert dy.shape == dx.shape return dx, []

class Dropout(Layer):

**def**  **init** **(self, keep\_rate: float): super().** **init** **() self.keep\_rate = keep\_rate**

**def forward(self, x: np.ndarray) -> Tuple[np.ndarray, List[Any], Dict[str, Any]]: mask = (np.random.binomial(1, self.keep\_rate, size=x.shape) / self.keep\_rate**

**if self.train else np.ones\_like(x)) return mask \* x, [], {"mask": mask}**

**def backward(self, x: np.ndarray, dy: np.ndarray, \*args, \*\*kwargs) -> Tuple[np.ndarray, List[np.ndarray]]:**

**assert "mask" in kwargs return kwargs["mask"] \* dy, []**



from typing import Tuple import numpy as np

def softmax(x):

e = np.exp(x - np.max(x, axis=1, keepdims=True)) return e / np.sum(e, axis=1, keepdims=True)

class Loss:

def forward(self, y\_true: np.ndarray, logits: np.ndarray) -> Tuple[np.ndarray, np.ndarray]: """Returns loss value and prediction."""

pass

def backward(self, y\_true: np.ndarray, logits: np.ndarray): pass

class CrossEntropy(Loss):

def forward(self, y\_true, logits): prob = softmax(logits)

return - np.log(prob[range(logits.shape[0]), y\_true]), np.argmax(prob, axis=1)

def backward(self, y\_true, logits): grad = softmax(logits)

grad[range(logits.shape[0]), y\_true] -= 1 return grad

**from typing import List**

**from layers import Layer from loss import Loss**

**from optimizers import Optimizer class Model:**

**def**  **init** **(self, layers: List[Layer], optimizer: Optimizer, criterion: Loss): self.layers = layers**

**self.optimizer = optimizer self.criterion = criterion**

**def train(self):**

**for l in self.layers: l.train = True**

**def eval(self):**

**for l in self.layers: l.train = False**

**def forward(self, x, y):**

**for idx, layer in enumerate(self.layers): new\_x, args, kwargs = layer.forward(x) self.optimizer.save(idx, (x, args, kwargs)) x = new\_x**

**logits = x**

**loss, pred = self.criterion.forward(y, logits) return loss, pred, logits**

**def backward(self, y, logits):**

**upstream\_grad = self.criterion.backward(y, logits) for idx, layer in reversed(list(enumerate(self.layers))):**

**x, args, kwargs = self.optimizer.load(idx)**

**upstream\_grad, grad = layer.backward(x, upstream\_grad, \*args, \*\*kwargs) self.optimizer.update\_layer(grad, layer)**



**import numpy as np**

**from layers import Layer, Param class Optimizer:**

**def**  **init** **(self): self.cache = {}**

**def load(self, idx): return self.cache[idx]**

**def save(self, idx, x): self.cache[idx] = x**

**def update\_layer(self, grad: np.ndarray, layer: Layer): weights = layer.weights()**

**assert len(weights) == len(grad)**

**for weight, grad in zip(weights, grad): self.step(grad, weight)**

**def zero\_grad(self):**

**self.cache = {}**

**def step(self, grad: np.ndarray, weight: Param): pass**

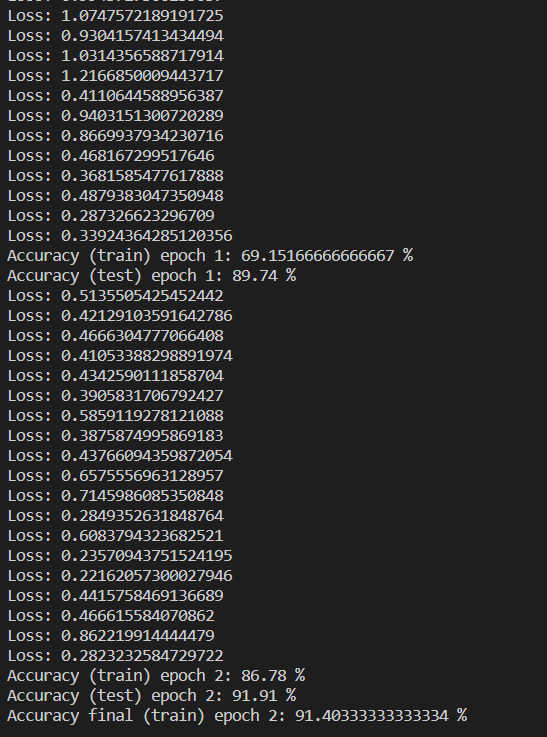
**class SGD(Optimizer):**

**def**  **init** **(self, learning\_rate: float = 0.01): super().** **init** **()**

**self.learning\_rate = learning\_rate**

**def step(self, grad: np.ndarray, weight: Param): weight.data -= self.learning\_rate \* grad.mean(axis=0)**





***ImplementedANN.***



 ***Implementfeedforwardbackpropagationneuralnetworklearningalgorithm fortherestaurantwaitingproblem.***



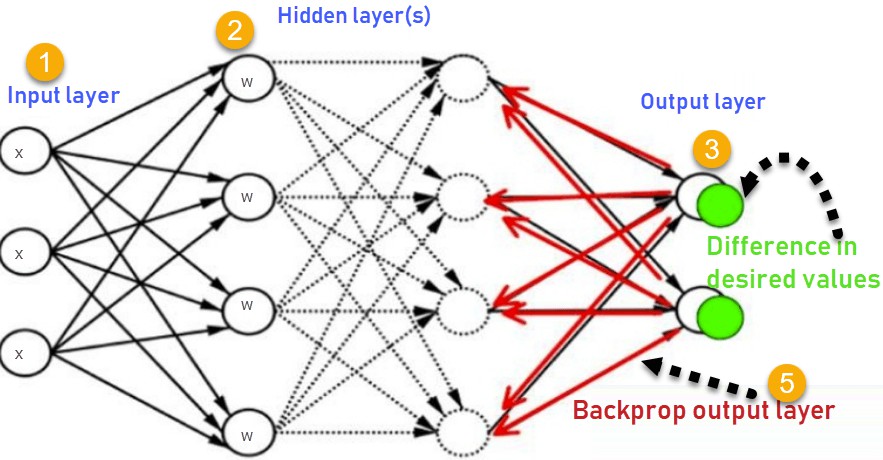
***Backpropagationistheessenceofneuralnetworktraining.Itisthemethodoffine- tuningtheweightsofaneuralnetworkbasedontheerrorrateobtainedinthe previousepoch(i.e.,iteration).Propertuningoftheweightsallowsyoutoreduce errorratesandmakethemodelreliablebyincreasingitsgeneralization.***

***Backpropagationinneuralnetworkisashortformfor“backwardpropagationof errors.”Itisastandardmethodoftrainingartificialneuralnetworks.Thismethod helpscalculatethegradientofalossfunctionwithrespecttoalltheweightsinthe network.***



***TheBackpropagationalgorithminneuralnetworkcomputesthegradientoftheloss functionforasingleweightbythechainrule.Itefficientlycomputesonelayerata time,unlikeanativedirectcomputation.Itcomputesthegradient,butitdoesnot definehowthegradientisused.Itgeneralizesthecomputationinthedeltarule.***

***ConsiderthefollowingBackpropagationneuralnetworkexamplediagramto understand:***



1. ***InputsX,arrivethroughthepreconnectedpath***
2. ***InputismodelledusingrealweightsW.Theweightsareusuallyrandomly selected.***
3. ***Calculatetheoutputforeveryneuronfromtheinputlayer,tothehiddenlayers, totheoutputlayer.***
4. ***Calculatetheerrorintheoutputs.***
5. *Travel back from the output layer to the hidden layer to adjust the weights such that the error is decreased.*

*Keep repeating the process until the desired output is achieved*

***Why We Need Backpropagation?***

*Most prominent advantages of Backpropagation are:*

* + *Backpropagation is fast, simple and easy to program*
  + *It has no parameters to tune apart from the numbers of input*
  + *It is a flexible method as it does not require prior knowledge about the network*
  + *It is a standard method that generally works well*
  + *It does not need any special mention of the features of the function to be learned.*

***What is a Feed Forward Network?***

*A feedforward neural network is an artificial neural network where the nodes never form a cycle. This kind of neural network has an input layer, hidden layers, and an output layer. It is the first and simplest type of artificial neural network.*

***Types of Backpropagation Networks***

*Two Types of Backpropagation Networks are:*

* + *Static Back-propagation*
  + *Recurrent Backpropagation*



**It is one kind of backpropagation network which produces a mapping of a static input for static output. It is useful to solve static classification issues like optical character recognition.**



**Recurrent Back propagation in data mining is fed forward until a fixed value is achieved. After that, the error is computed and propagated backward.**

**The main difference between both of these methods is: that the mapping is rapid in static back-propagation while it is non-static in recurrent backpropagation.**



* **Simplifies the network structure by elements weighted links that have the least effect on the trained network**
* **You need to study a group of input and activation values to develop the relationship between the input and hidden unit layers.**
* **It helps to assess the impact that a given input variable has on a network output. The knowledge gained from this analysis should be represented in rules.**
* **Backpropagation is especially useful for deep neural networks working on error-prone projects, such as image or speech recognition.**
* **Backpropagation takes advantage of the chain and power rules allows backpropagation to function with any number of outputs.**



**Backpropagation in neural network can be explained with the help of “Shoe Lace” analogy**





* **Not enough constraining and very loose**



* **Too much constraint (overtraining)**
* **Taking too much time (relatively slow process)**
* **Higher likelihood of breaking**





* **Discomfort (bias)**



* **The actual performance of backpropagation on a specific problem is dependent on the input data.**
* **Back propagation algorithm in data mining can be quite sensitive to noisy**

**data**

* **You need to use the matrix-based approach for backpropagation instead of mini-batch.**



import numpy as np class NeuralNetwork():

**def**  **init** **(self):**

#seeding for random number generation np.random.seed()

#converting weights to a 3 by 1 matrix self.synaptic\_weights=2\*np.random.random((3,1))-1

#x is output variable def sigmoid(self, x):

#applying the sigmoid function return 1/(1+np.exp(-x))

def sigmoid\_derivative(self,x):

#computing derivative to the sigmoid function return x\*(1-x)

def train(self,training\_inputs,training\_outputs,training\_iterations):

#training the model to make accurate predictions while adjusting for iteration in range(training\_iterations):

#siphon the training data via the neuron output=self.think(training\_inputs)

error=training\_outputs-output #performing weight adjustments

adjustments=np.dot(training\_inputs.T,error\*self.sigmoid\_derivative(output)) self.synaptic\_weights+=adjustments

def think(self,inputs):

#passing the inputs via the neuron to get output #converting values to floats

inputs=inputs.astype(float) output=self.sigmoid(np.dot(inputs,self.synaptic\_weights))

return output

**if**  **name** **=="** **main** **":**

#initializing the neuron class neural\_network=NeuralNetwork()

print("Beginning randomly generated weights: ") print(neural\_network.synaptic\_weights)

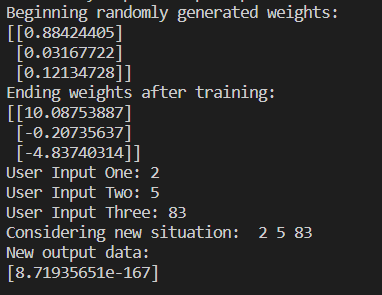
**#training data consisting of 4 examples--3 inputs & 1 output training\_inputs=np.array([[0,0,1],[1,1,1],[1,0,1],[0,1,1]]) training\_outputs=np.array([[0,1,1,0]]).T**

**#training taking place neural\_network.train(training\_inputs,training\_outputs,15000)**

**print("Ending weights after training: ") print(neural\_network.synaptic\_weights)**

**user\_input\_one=str(input("User Input One: ")) user\_input\_two=str(input("User Input Two: ")) user\_input\_three=str(input("User Input Three: "))**

**print("Considering new situation: ",user\_input\_one,user\_input\_two,user\_input\_three) print("New output data: ") print(neural\_network.think(np.array([user\_input\_one,user\_input\_two,user\_input\_three])))**



***Implementedfeedforwardbackpropagationneuralnetworklearningalgorithmfor therestaurantwaitingproblem.***

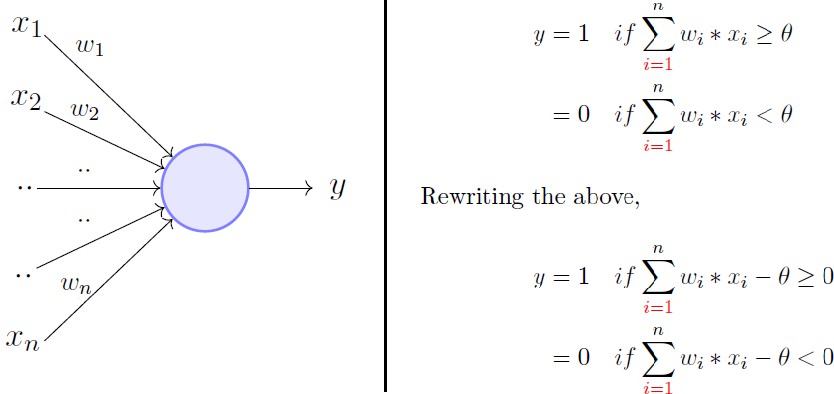


 ***ImplementthePerceptronAlgorithm.***

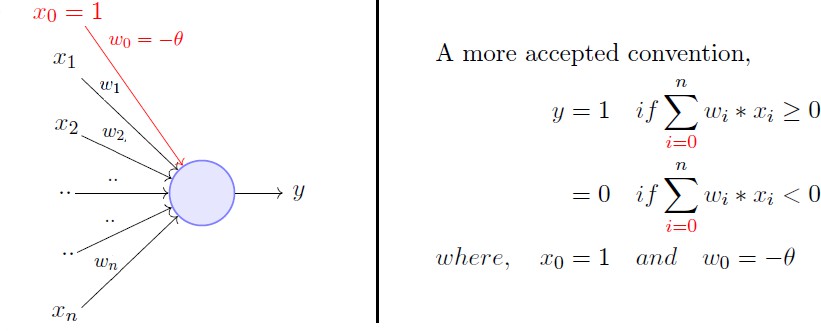


***AperceptronisnottheSigmoidneuronweuseinANNsoranydeeplearning***

***networkstoday.***

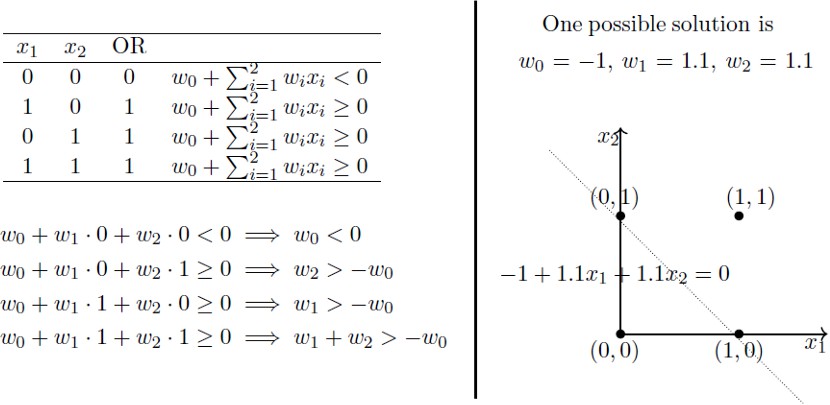


***TheperceptronmodelisamoregeneralcomputationalmodelthanMcCulloch-Pitts neuron.Ittakesaninput,aggregatesit(weightedsum)andreturns1onlyifthe aggregatedsumismorethansomethresholdelsereturns0.Rewritingthethreshold asshownaboveandmakingitaconstantinputwithavariableweight,wewouldend upwithsomethinglikethefollowing:***



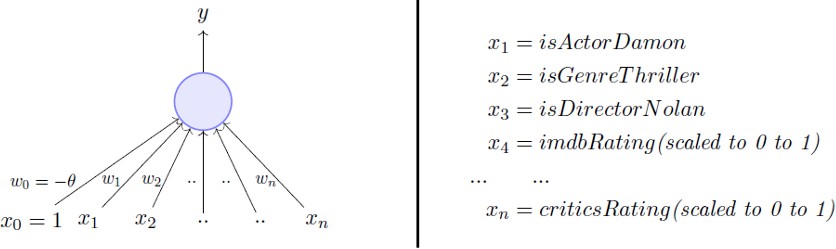
***Asingleperceptroncanonlybeusedtoimplement functions.It takesbothrealandbooleaninputsandassociatesasetof tothem,along witha(thethresholdthingImentionedabove).Welearntheweights,wegetthe function.Let'suseaperceptrontolearnanORfunction.***





***What’sgoingonaboveisthatwedefinedafewconditions(theweightedsumhasto bemorethanorequalto0whentheoutputis1)basedontheORfunctionoutputfor varioussetsofinputs,wesolvedforweightsbasedonthoseconditionsandwegota linethatperfectlyseparatespositiveinputsfromthoseofnegative.***

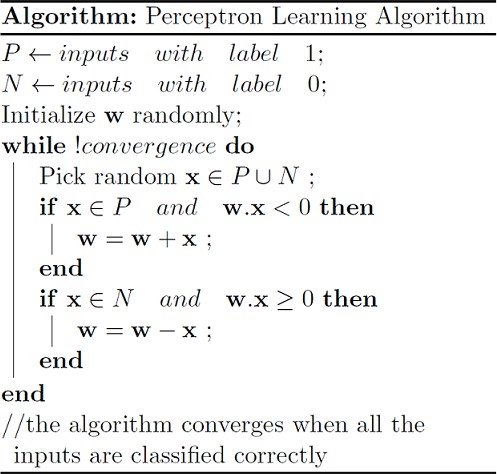
***Doesn’tmakeanysense?MaybenowisthetimeyougothroughthatpostIwas talkingabout.MinskyandPapertalsoproposedamoreprincipledwayoflearning theseweightsusingasetofexamples(data).MindyouthatthisisNOTaSigmoid neuronandwe’renotgoingtodoanyGradientDescent.***



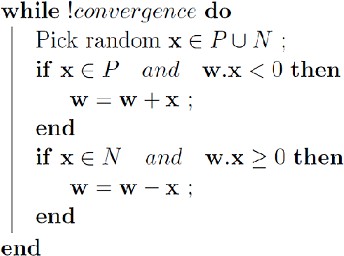
***WearegoingtouseaperceptrontoestimateifIwillbewatchingamoviebasedon historicaldatawiththeabove-mentionedinputs.Thedatahaspositiveandnegative examples,positivebeingthemoviesIwatchedi.e.,1.Basedonthedata,weare goingtolearntheweightsusingtheperceptronlearningalgorithm.Forvisual simplicity,wewillonlyassumetwo-dimensionalinput.***



***Ourgoalistofindthe vectorthatcanperfectlyclassifypositiveinputsand negativeinputsinourdata.Iwillgetstraighttothealgorithm.Heregoes:***



***Weinitialize withsomerandomvector.Wetheniterateoveralltheexamplesinthe data,(PUN)bothpositiveandnegativeexamples.NowifaninputbelongstoP, ideallywhatshouldthedotproduct be?I’dsaygreaterthanorequalto0 becausethat’stheonlythingwhatourperceptronwantsattheendofthedaysolet's giveitthat.AndifbelongstoN,thedotproductMUSTbelessthan0.Soifyoulook attheifconditionsinthewhileloop:***



  ***WhenbelongstoPanditsdotproduct <0 WhenbelongstoNanditsdotproduct ≥0***

***Onlyforthesecases,weareupdatingourrandomlyinitialized .Otherwise,wedon’t touch atallbecauseCase1andCase2areviolatingtheveryruleofaperceptron. Soweareaddingto (ahemvectoradditionahem)inCase1and subtractingfrom inCase2.***

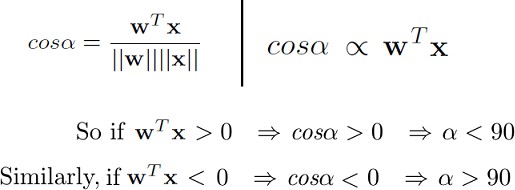


***Butwhywouldthiswork?Ifyougetitalreadywhythiswouldwork,you’vegotthe entiregistofmypostandyoucannowmoveonwithyourlife,thanksforreading, bye.Butifyouarenotsurewhytheseseeminglyarbitraryoperations***

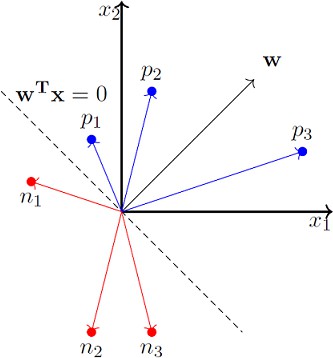
***ofand wouldhelpyoulearnthatperfect thatcanperfectlyclassifyPandN, stickwithme.***

***WehavealreadyestablishedthatwhenbelongstoP,wewant >0,basic perceptronrule.WhatwealsomeanbythatisthatwhenbelongstoP,theangle between andshouldbe******than90degrees.Fillintheblank.***

***Answer:Theanglebetween andshouldbelessthan90becausethecosineof theangleisproportionaltothedotproduct.***

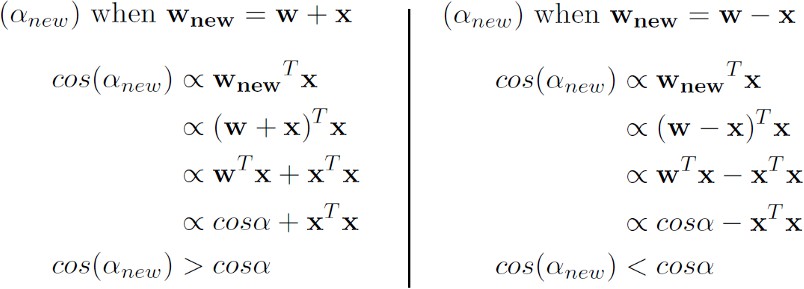


***Sowhateverthe vectormaybe,aslongasitmakesananglelessthan90degrees withthepositiveexampledatavectors( EP)andananglemorethan90degrees withthenegativeexampledatavectors( EN),wearecool.Soideally,itshouldlook somethinglikethis:***



***x\_0isalways1soweignoreitfornow.***

***Sowenowstronglybelievethattheanglebetween andshouldbelessthan90 whenbelongstoPclassandtheanglebetweenthemshouldbemorethan90 whenbelongstoNclass.Pauseandconvinceyourselfthattheabovestatements aretrueandyouindeedbelievethem.Here’swhytheupdateworks:***



***Sowhenweareaddingto ,whichwedowhenxbelongstoPand <0(Case1),***

***weareessentially value,whichmeans,weare ***



***,theanglebetween and , .Andthesimilar intuitionworksforthecasewhenbelongstoNand ≥0(Case2).***

***Here’satoysimulationofhowwemightupenduplearning thatmakesanangle lessthan90forpositiveexamplesandmorethan90fornegativeexamples.***



from sklearn import datasets import numpy as np

import matplotlib.pyplot as plt

X, y =

datasets.make\_blobs(n\_samples=1100,n\_features=2,centers=2,cluster\_std=1.05,random\_state= 2)

#Plotting

fig = plt.figure(figsize=(10,8)) plt.plot(X[:, 0][y == 0], X[:, 1][y == 0], 'r^')

plt.plot(X[:, 0][y == 1], X[:, 1][y == 1], 'bs')

plt.xlabel("feature 1")

plt.ylabel("feature 2")

plt.title('Random Classification Data with 2 classes')

def step\_func(z):

return 1.0 if (z > 0) else 0.0 def perceptron(X, y, lr, epochs):

# X --> Inputs.

# y --> labels/target. # lr --> learning rate.

# epochs --> Number of iterations.

# m-> number of training examples # n-> number of features

m, n = X.shape

# Initializing parapeters(theta) to zeros. # +1 in n+1 for the bias term.

theta = np.zeros((n+1,1))

# Empty list to store how many examples were # misclassified at every iteration.

n\_miss\_list = []

# Training.

for epoch in range(epochs):

# variable to store #misclassified. n\_miss = 0

# looping for every example. for idx, x\_i in enumerate(X):

# Insering 1 for bias, X0 = 1.

x\_i = np.insert(x\_i, 0, 1).reshape(-1,1)

# Calculating prediction/hypothesis. y\_hat = step\_func(np.dot(x\_i.T, theta))

# Updating if the example is misclassified. if (np.squeeze(y\_hat) - y[idx]) != 0:

theta += lr\*((y[idx] - y\_hat)\*x\_i)

# Incrementing by 1. n\_miss += 1

# Appending number of misclassified examples # at every iteration.

n\_miss\_list.append(n\_miss) return theta, n\_miss\_list

def plot\_decision\_boundary(X, theta):

# X --> Inputs

# theta --> parameters

# The Line is y=mx+c

# So, Equate mx+c = theta0.X0 + theta1.X1 + theta2.X2 # Solving we find m and c

x1 = [min(X[:,0]), max(X[:,0])]

m = -theta[1]/theta[2] c = -theta[0]/theta[2] x2 = m\*x1 + c

# Plotting

fig = plt.figure(figsize=(10,8)) plt.plot(X[:, 0][y==0], X[:, 1][y==0], "r^")

plt.plot(X[:, 0][y==1], X[:, 1][y==1], "bs")

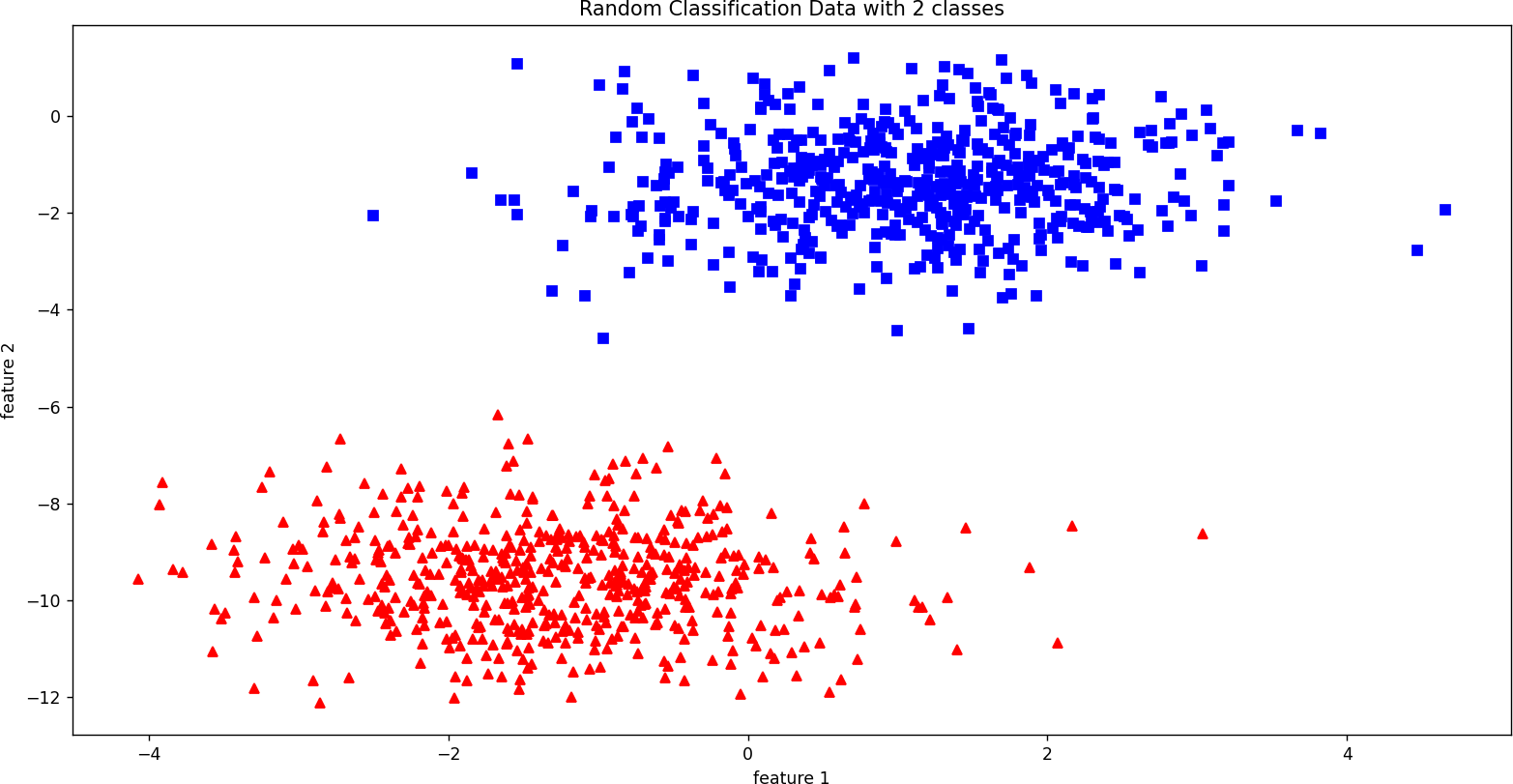
plt.xlabel("feature 1")

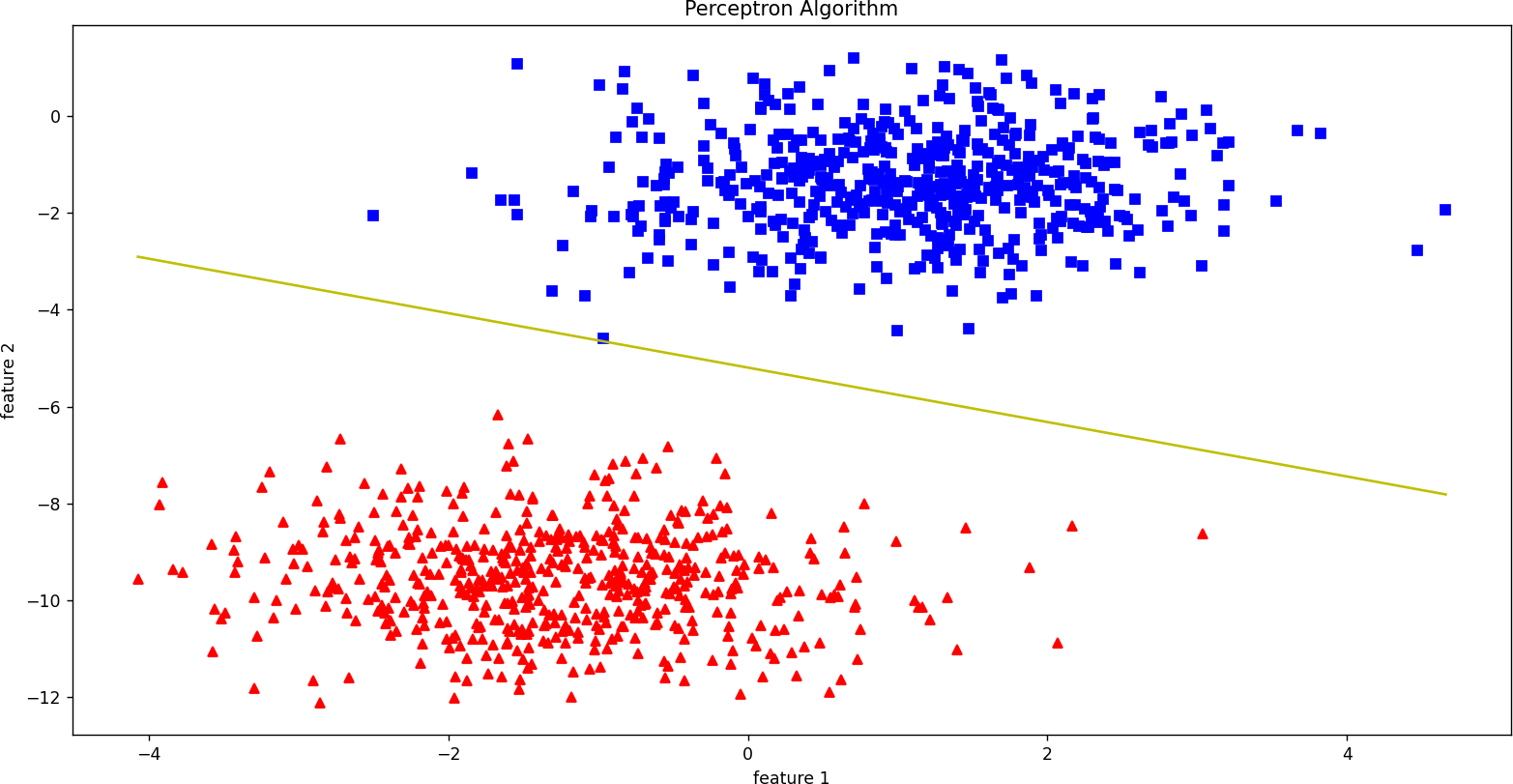
plt.ylabel("feature 2") plt.title("Perceptron Algorithm") plt.plot(x1, x2, 'y-')

plt.show()

theta, miss\_l = perceptron(X, y, 0.5, 100) plot\_decision\_boundary(X, theta)







***ImplementedthePerceptronAlgorithm.***



 ***ImplementFuzzyInferenceSystem.***



***FuzzyInferenceSystemisthekeyunitofafuzzylogicsystemhavingdecision makingasitsprimarywork.Itusesthe“IF…THEN”rulesalongwithconnectors“OR” or“AND”fordrawingessentialdecisionrules.***



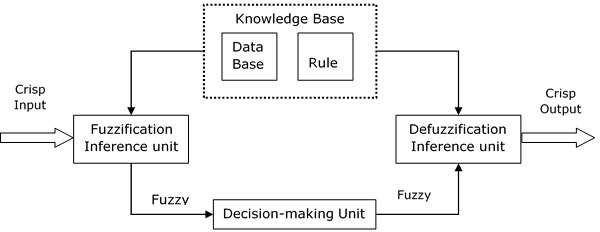
* ***TheoutputfromFISisalwaysafuzzysetirrespectiveofitsinputwhichcan befuzzyorcrisp.***
* ***Itisnecessarytohavefuzzyoutputwhenitisusedasacontroller.***
* ***AdefuzzificationunitwouldbetherewithFIStoconvertfuzzyvariablesinto crispvariables.***



***ThefollowingfivefunctionalblockswillhelpyouunderstandtheconstructionofFIS***

***-***

* ***RuleBase−ItcontainsfuzzyIF-THENrules.***
* ***Database−Itdefinesthemembershipfunctionsoffuzzysetsusedinfuzzy rules.***
* ***Decision-makingUnit−Itperformsoperationonrules.***
* ***FuzzificationInterfaceUnit−Itconvertsthecrispquantitiesintofuzzy quantities.***
* ***DefuzzificationInterfaceUnit−Itconvertsthefuzzyquantitiesintocrisp quantities.Followingisablockdiagramoffuzzyinterferencesystem.***



***WorkingofFIS TheworkingoftheFISconsistsofthefollowingsteps−***

* ***Afuzzificationunitsupportstheapplicationofnumerousfuzzification methods,andconvertsthecrispinputintofuzzyinput.***
* ***Aknowledgebase-collectionofrulebaseanddatabaseisformeduponthe conversionofcrispinputintofuzzyinput.***
* ***Thedefuzzificationunitfuzzyinputisfinallyconvertedintocrispoutput.***

***MethodsofFIS***

***LetusnowdiscussthedifferentmethodsofFIS.Followingarethetwoimportant methodsofFIS,havingdifferentconsequentoffuzzyrules−***

* ***MamdaniFuzzyInferenceSystem***
* ***Takagi-SugenoFuzzyModel(TSMethod) MamdaniFuzzyInferenceSystem***

***Thissystemwasproposedin1975byEbhasimMamdani.Basically,itwas anticipatedtocontrolasteamengineandboilercombinationbysynthesizingaset offuzzyrulesobtainedfrompeopleworkingonthesystem.***

***StepsforComputingtheOutput FollowingstepsneedtobefollowedtocomputetheoutputfromthisFIS−***

  ***−Setoffuzzyrulesneedtobedeterminedinthisstep.***

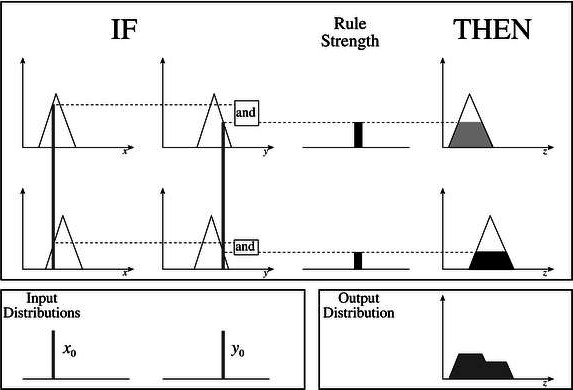
***−Inthisstep,byusinginputmembershipfunction,theinputwouldbe madefuzzy.***

 ***−Nowestablishtherulestrengthbycombiningthefuzzifiedinputs accordingtofuzzyrules.***

 ***−Inthisstep,determinetheconsequentofrulebycombiningtherule strengthandtheoutputmembershipfunction.***

 ***−Forgettingoutputdistributioncombinealltheconsequents. *** ***−Finally,adefuzzifiedoutputdistributionisobtained.***

***FollowingisablockdiagramofMamdaniFuzzyInterfaceSystem.***





**fuzzy\_clause.py '''**

**Fuzzy Clause class. Used in Fuzzy rule '''**

**class FuzzyClause(): '''**

**A fuzzy clause of the type 'variable is set' used in fuzzy IF ... THEN ... rules**

**clauses can be antecedent (if part) or consequent (then part)**

**'''**

**def**  **init** **(self, variable, f\_set, degree=1): '''**

initialization of the fuzzy clause Arguments:

variable -- the clause variable in 'variable is set' set -- the clause set in 'variable is set'

**'''**

if f\_set is None:

raise Exception('set none')

if f\_set.name == '':

raise Exception(str(f\_set), 'no set name')

self.\_variable = variable self.\_set = f\_set

def str (self): '''

string representation of the clause. Returns:

str: str, string representation of the clause in the form A is x

**'''**

return f'{self.\_variable.name} is {self.\_set.name}'

@property

def variable\_name(self): '''

returns the name of the clause variable Returns:

variable\_name: str, name of variable '''

return self.\_variable.name

@property

def set\_name(self): '''

returns the name of the clause variable Returns:

variable\_name: str, name of variable '''

return self.\_set.name

def evaluate\_antecedent(self): '''

Used when set is antecedent.

returns the set degree of membership. Returns:

dom -- number, the set degree of membership given a value for that variable. This value is determined at an earlier stage and stored in the set

**'''**

return self.\_set.last\_dom\_value

def evaluate\_consequent(self, dom): '''

Used when clause is consequent. Arguments:

dom -- number, scalar value from the antecedent clauses Returns:

set -- Type1FuzzySet, a set resulting from min operation with the scalar value

**'''**

self.\_variable.add\_rule\_contribution(self.\_set.min\_scalar(dom))

fuzzy\_rule.py

from fuzzy\_logic.fuzzy\_clause import FuzzyClause

class FuzzyRule(): '''

A fuzzy rule of the type

IF [antecedent clauses] THEN [consequent clauses] '''

def init (self): '''

initializes the rule. Two data structures are necessary:

Antecedent clauses list consequent clauses list

**'''**

self.\_antecedent = [] self.\_consequent = []

def str (self): '''

string representation of the rule. Returns:

str: str, string representation of the rule in the form

IF [antecedent clauses] THEN [consequent clauses]

**'''**

ante = ' and '.join(map(str, self.\_antecedent)) cons = ' and '.join(map(str, self.\_consequent)) return f'If {ante} then {cons}'

def add\_antecedent\_clause(self, var, f\_set): '''

adds an antecedent clause to the rule

Arguments:

clause -- FuzzyClause, the antecedent clause

**'''**

self.\_antecedent.append(FuzzyClause(var, f\_set))

def add\_consequent\_clause(self, var, f\_set): '''

adds an consequent clause to the rule Arguments:

clause -- FuzzyClause, the consequent clause

**'''**

self.\_consequent.append(FuzzyClause(var, f\_set))

def evaluate(self): '''

evaluation of the rule.

the antecedent clauses are executed and the minimum degree of membership is retained.

This is used in teh consequent clauses to min with the consequent set

The values are returned in a dict of the form {variable\_name: scalar min set, ...} Returns:

rule\_consequence -- dict, the resulting sets in the form

{variable\_name: scalar min set, ...}

**'''**

# rule dom initialize to 1 as min operator will be performed rule\_strength = 1

# execute all antecedent clauses, keeping the minimum of the # returned doms to determine the rule strength

for ante\_clause in self.\_antecedent:

rule\_strength = min(ante\_clause.evaluate\_antecedent(), rule\_strength)

# execute consequent clauses, each output variable will update its output\_distribution set for consequent\_clause in self.\_consequent:

consequent\_clause.evaluate\_consequent(rule\_strength)

def evaluate\_info(self): '''

evaluation of the rule.

the antecedent clauses are executed and the minimum degree of membership is retained.

This is used in teh consequent clauses to min with the consequent set

The values are returned in a dict of the form {variable\_name: scalar min set, ...} Returns:

rule\_consequence -- dict, the resulting sets in the form

{variable\_name: scalar min set, ...}

**'''**

# rule dom initialize to 1 as min operator will be performed rule\_strength = 1

# execute all antecedent clauses, keeping the minimum of the # returned doms to determine the rule strength

for ante\_clause in self.\_antecedent:

rule\_strength = min(ante\_clause.evaluate\_antecedent(), rule\_strength)

# execute consequent clauses, each output variable will update its output\_distribution set for consequent\_clause in self.\_consequent:

consequent\_clause.evaluate\_consequent(rule\_strength)

return f'{rule\_strength} : {self}' fuzzy\_set.py

import numpy as np import copy

import matplotlib.pyplot as plt class FuzzySet:

\_precision: int = 3

def init (self, name, domain\_min, domain\_max, res): self.\_domain\_min = domain\_min

self.\_domain\_max = domain\_max self.\_res = res

self.\_domain = np.linspace(domain\_min, domain\_max, res) self.\_dom = np.zeros(self.\_domain.shape)

self.\_name = name self.\_last\_dom\_value = 0

def getitem (self, x\_val):

return self.\_dom[np.abs(self.\_domain-x\_val).argmin()]

def setitem (self, x\_val, dom):

self.\_dom[np.abs(self.\_domain-x\_val).argmin()] = round(dom, self.\_precision)

**def**  **str** **(self):**

return ' + '.join([str(a) + '/' + str(b) for a,b in zip(self.\_dom, self.\_domain)])

def get\_last\_dom\_value(self): return self.\_last\_dom\_value

def set\_last\_dom\_value(self, d): self.\_last\_dom\_value = d

last\_dom\_value = property( get\_last\_dom\_value, set\_last\_dom\_value) @property

def name(self): return self.\_name

@property

def empty(self):

return np.all(self.\_dom == 0)

@property

def name(self): return self.\_name

@classmethod

def create\_trapezoidal(cls, name, domain\_min, domain\_max, res, a, b, c, d): t1fs = cls(name, domain\_min, domain\_max, res)

a = t1fs.\_adjust\_domain\_val(a)

b = t1fs.\_adjust\_domain\_val(b)

c = t1fs.\_adjust\_domain\_val(c)

d = t1fs.\_adjust\_domain\_val(d)

t1fs.\_dom = np.round(np.minimum(np.maximum(np.minimum((t1fs.\_domain-a)/(b-a), (d- t1fs.\_domain)/(d-c)), 0), 1), t1fs.\_precision)

return t1fs

@classmethod

def create\_triangular(cls, name, domain\_min, domain\_max, res, a, b, c): t1fs = cls(name, domain\_min, domain\_max, res)

a = t1fs.\_adjust\_domain\_val(a)

b = t1fs.\_adjust\_domain\_val(b)

c = t1fs.\_adjust\_domain\_val(c)

if b == a:

t1fs.\_dom = np.round(np.maximum((c-t1fs.\_domain)/(c-b), 0), t1fs.\_precision) elif b == c:

t1fs.\_dom = np.round(np.maximum((t1fs.\_domain-a)/(b-a), 0), t1fs.\_precision) else:

t1fs.\_dom = np.round(np.maximum(np.minimum((t1fs.\_domain-a)/(b-a), (c- t1fs.\_domain)/(c-b)), 0), t1fs.\_precision)

return t1fs

def \_adjust\_domain\_val(self, x\_val):

return self.\_domain[np.abs(self.\_domain-x\_val).argmin()]

def clear\_set(self): self.\_dom.fill(0)

def min\_scalar(self, val):

result = FuzzySet(f'({self.\_name}) min ({val})', self.\_domain\_min, self.\_domain\_max, self.\_res) result.\_dom = np.minimum(self.\_dom, val)

return result

def union(self, f\_set):

result = FuzzySet(f'({self.\_name}) union ({f\_set.\_name})', self.\_domain\_min, self.\_domain\_max, self.\_res)

result.\_dom = np.maximum(self.\_dom, f\_set.\_dom) return result

def intersection(self, f\_set):

result = FuzzySet(f'({self.\_name}) intersection ({f\_set.\_name})', self.\_domain\_min, self.\_domain\_max, self.\_res)

result.\_dom = np.minimum(self.\_dom, f\_set.\_dom) return result

def complement(self):

result = FuzzySet(f'not ({self.\_name})', self.\_domain\_min, self.\_domain\_max, self.\_res) result.\_dom = 1 - self.\_dom

return result

def cog\_defuzzify(self):

num = np.sum(np.multiply(self.\_dom, self.\_domain)) den = np.sum(self.\_dom)

return num/den

def domain\_elements(self): return self.\_domain

def dom\_elements(self): return self.\_dom

def plot\_set(self, ax, col=''): ax.plot(self.\_domain, self.\_dom, col) ax.set\_ylim([-0.1,1.1]) ax.set\_title(self.\_name)

ax.grid(True, which='both', alpha=0.4) ax.set(xlabel='x', ylabel='$\mu(x)$')

**if**  **name** **== "** **main** **":**

s = FuzzySet.create\_trapezoidal('test', 1, 100, 100, 20, 30, 50, 80) print(s.empty)

u = FuzzySet('u', 1, 100, 100) print(u.empty)

t = FuzzySet.create\_trapezoidal('test', 1, 100, 100, 30, 50, 90, 100) fig, axs = plt.subplots(1, 1)

s.union(t).complement().intersection(s).min\_scalar(0.2).plot\_set(axs)

plt.show() print(s.cog\_defuzzify())

#fuzzy\_system.py

from fuzzy\_logic.fuzzy\_rule import FuzzyRule

from fuzzy\_logic.fuzzy\_variable\_output import FuzzyOutputVariable from fuzzy\_logic.fuzzy\_variable\_input import FuzzyInputVariable

import matplotlib.pyplot as plt from matplotlib import rc import numpy as np

class FuzzySystem: '''

A type-1 fuzzy system based on Mamdani inference system Reference:

Mamdani, Ebrahim H., and Sedrak Assilian. "An experiment in linguistic synthesis with a fuzzy logic controller." Readings in Fuzzy Sets

for Intelligent Systems. Morgan Kaufmann, 1993. 283-289. '''

def init (self): '''

initializes fuzzy system. data structures required:

input variables -- dict, having format {variable\_name: FuzzyVariable, ...} output variables -- dict, having format {variable\_name: FuzzyVariable, ...} rules -- list of FuzzyRule

output\_distribution -- dict holding fuzzy output for each variable having format

{variable\_name: FuzzySet, ...}

**'''**

self.\_input\_variables = {} self.\_output\_variables = {} self.\_rules = []

def str (self): '''

string representation of the system. Returns:

str: str, string representation of the system in the form Input:

input\_variable\_name(set\_names)... Output: output\_variable\_name(set\_names)... Rules:

IF [antecedent clauses] THEN [consequent clauses]

**'''**

ret\_str = 'Input: \n'

for n, s in self.\_input\_variables.items(): ret\_str = ret\_str + f'{n}: ({s})\n'

ret\_str = ret\_str + 'Output: \n'

for n, s in self.\_output\_variables.items():

ret\_str = ret\_str + f'{n}: ({s})\n'

ret\_str = ret\_str + 'Rules: \n' for rule in self.\_rules:

ret\_str = ret\_str + f'{rule}\n' return ret\_str

def add\_input\_variable(self, variable): '''

adds an input variable to the system Arguments:

variable -- FuzzyVariable, the input variable '''

self.\_input\_variables[variable.name] = variable

def add\_output\_variable(self, variable): self.\_output\_variables[variable.name] = variable

def get\_input\_variable(self, name): '''

get an input variable given the name Arguments:

name -- str, name of variable Returns:

variable -- FuzzyVariable, the input variable '''

return self.\_input\_variables[name]

def get\_output\_variable(self, name): '''

get an output variable given the name Arguments:

name -- str, name of variable Returns:

variable -- FuzzyVariable, the output variable '''

return self.\_output\_variables[name] def \_clear\_output\_distributions(self):

**'''**

used for each iteration. The fuzzy result is cleared '''

map(lambda output\_var: output\_var.clear\_output\_distribution(), self.\_output\_variables.values())

def add\_rule(self, antecedent\_clauses, consequent\_clauses): '''

adds a new rule to the system. TODO: add checks

Arguments:

antecedent\_clauses -- dict, having the form {variable\_name:set\_name, ...} consequent\_clauses -- dict, having the form {variable\_name:set\_name, ...} '''

# create a new rule

# new\_rule = FuzzyRule(antecedent\_clauses, consequent\_clauses) new\_rule = FuzzyRule()

for var\_name, set\_name in antecedent\_clauses.items(): # get variable by name

var = self.get\_input\_variable(var\_name) # get set by name

f\_set = var.get\_set(set\_name) # add clause

new\_rule.add\_antecedent\_clause(var, f\_set)

for var\_name, set\_name in consequent\_clauses.items(): var = self.get\_output\_variable(var\_name)

f\_set = var.get\_set(set\_name) new\_rule.add\_consequent\_clause(var, f\_set)

# add the new rule self.\_rules.append(new\_rule)

def evaluate\_output(self, input\_values): '''

Executes the fuzzy inference system for a set of inputs Arguments:

input\_values -- dict, containing the inputs to the systems in the form

{input\_variable\_name: value, ...}

Returns:

output -- dict, containing the outputs from the systems in the form

{output\_variable\_name: value, ...}

**'''**

# clear the fuzzy consequences as we are evaluating a new set of inputs.

# can be optimized by comparing if the inputs have changes from the previous # iteration.

self.\_clear\_output\_distributions()

# Fuzzify the inputs. The degree of membership will be stored in # each set

for input\_name, input\_value in input\_values.items(): self.\_input\_variables[input\_name].fuzzify(input\_value)

# evaluate rules

for rule in self.\_rules: rule.evaluate()

# finally, defuzzify all output distributions to get the crisp outputs output = {}

for output\_var\_name, output\_var in self.\_output\_variables.items(): output[output\_var\_name] = output\_var.get\_crisp\_output()

return output

def evaluate\_output\_info(self, input\_values): '''

Executes the fuzzy inference system for a set of inputs Arguments:

input\_values -- dict, containing the inputs to the systems in the form

{input\_variable\_name: value, ...}

Returns:

output -- dict, containing the outputs from the systems in the form

{output\_variable\_name: value, ...}

**'''**

info = {}

# clear the fuzzy consequences as we are evaluating a new set of inputs.

# can be optimized by comparing if the inputs have changes from the previous # iteration.

self.\_clear\_output\_distributions()

# Fuzzify the inputs. The degree of membership will be stored in # each set

fuzzification\_info = []

for input\_name, input\_value in input\_values.items(): fuzzification\_info.append(self.\_input\_variables[input\_name].fuzzify\_info(input\_value))

info['fuzzification'] = '\n'.join(fuzzification\_info) # evaluate rules

rule\_info = []

for rule in self.\_rules: rule\_info.append(rule.evaluate\_info())

info['rules'] = '\n'.join(rule\_info)

# finally, defuzzify all output distributions to get the crisp outputs output = {}

for output\_var\_name, output\_var in self.\_output\_variables.items(): output[output\_var\_name], info = output\_var.get\_crisp\_output\_info() # info[output\_var\_name] = info

return output, info def plot\_system(self):

total\_var\_count = len(self.\_input\_variables) + len(self.\_output\_variables)

if total\_var\_count <2:

total\_var\_count = 2

fig, axs = plt.subplots(total\_var\_count, 1) fig.tight\_layout(pad=1.0)

for idx, var\_name in enumerate(self.\_input\_variables): self.\_input\_variables[var\_name].plot\_variable(ax=axs[idx], show=False)

for idx, var\_name in enumerate(self.\_output\_variables): self.\_output\_variables[var\_name].plot\_variable(ax=axs[len(self.\_input\_variables)+idx],

show=False)

plt.show()

if name == " main ": pass

fuzzy\_variable\_input.py

from fuzzy\_logic.fuzzy\_variable import FuzzyVariable class FuzzyInputVariable(FuzzyVariable):

def init (self, name, min\_val, max\_val, res): super(). init (name, min\_val, max\_val, res)

def fuzzify(self, value): '''

performs fuzzification of the variable. used when the variable is an input one

Arguments:

value -- number, input value for the variable '''

# get dom for each set and store it - it will be required for each rule for set\_name, f\_set in self.\_sets.items():

f\_set.last\_dom\_value = f\_set[value]

def fuzzify\_info(self, value): '''

performs fuzzification of the variable. used when the variable is an input one

Arguments:

value -- number, input value for the variable '''

# get dom for each set and store it - it will be required for each rule for set\_name, f\_set in self.\_sets.items():

f\_set.last\_dom\_value = f\_set[value] res = []

res.append(self.\_name) res.append('\n')

for \_, f\_set in self.\_sets.items(): res.append(f\_set.name) res.append(str(f\_set.last\_dom\_value)) res.append('\n')

return ' '.join(res)

if name == " main ": pass

#fuzzy\_variable\_output.py

from fuzzy\_logic.fuzzy\_variable import FuzzyVariable from fuzzy\_logic.fuzzy\_set import FuzzySet

class FuzzyOutputVariable(FuzzyVariable):

def init (self, name, min\_val, max\_val, res): super(). init (name, min\_val, max\_val, res)

self.\_output\_distribution = FuzzySet(name, min\_val, max\_val, res)

def clear\_output\_distribution(self): self.\_output\_distribution.clear\_set()

def add\_rule\_contribution(self, rule\_consequence):

self.\_output\_distribution = self.\_output\_distribution.union(rule\_consequence)

def get\_crisp\_output(self):

return self.\_output\_distribution.cog\_defuzzify()

def get\_crisp\_output\_info(self):

return self.\_output\_distribution.cog\_defuzzify(), self.\_output\_distribution

if name == " main ": pass

#fuzzy\_variable.py

from fuzzy\_logic.fuzzy\_set import FuzzySet import matplotlib.pyplot as plt

import numpy as np

class FuzzyVariable(): '''

A type-1 fuzzy variable that is mage up of a number of type-1 fuzzy sets '''

def init (self, name, min\_val, max\_val, res): '''

creates a new type-1 fuzzy variable (universe) Arguments:

min\_val -- number, minimum value of variable max\_val -- number, maximum value of variable res -- int, resolution of variable

**'''**

self.\_sets={}

self.\_max\_val = max\_val self.\_min\_val = min\_val self.\_res = res self.\_name = name

**def**  **str** **(self):**

return ', '.join(self.\_sets.keys())

@property

def name(self): return self.\_name

def \_add\_set(self, name, f\_set): '''

adds a fuzzy set to the variable Arguments:

name -- string, name of the set f\_set -- FuzzySet, The set

**'''**

self.\_sets[name] = f\_set

def get\_set(self, name): '''

returns a set given the name Arguments:

name -- str, set name Returns:

set -- FuzzySet, the set '''

return self.\_sets[name]

def add\_triangular(self, name, low, mid, high):

new\_set = FuzzySet.create\_triangular(name, self.\_min\_val, self.\_max\_val, self.\_res, low, mid, high)

self.\_add\_set(name, new\_set) return new\_set

def add\_trapezoidal(self, name, a, b, c, d):

new\_set = FuzzySet. create\_trapezoidal(name, self.\_min\_val, self.\_max\_val, self.\_res, a, b, c,

d)

self.\_add\_set(name, new\_set) return new\_set

def plot\_variable(self, ax=None, show=True): '''

plots a graphical representation of the fuzzy variable Reference:

https://stackoverflow.com/questions/4700614/how-to-put-the-legend-out-of-the-plot

**'''**

**if ax == None:**

**ax = plt.subplot(111)**

**for n ,s in self.\_sets.items():**

**ax.plot(s.domain\_elements(), s.dom\_elements(), label=n)**

**# Shrink current axis by 20% pos = ax.get\_position()**

**ax.set\_position([pos.x0, pos.y0, pos.width \* 0.8, pos.height]) ax.grid(True, which='both', alpha=0.4) ax.set\_title(self.\_name)**

**ax.set(xlabel='x', ylabel='$\mu (x)$')**

**# Put a legend to the right of the current axis ax.legend(loc='center left', bbox\_to\_anchor=(1, 0.5))**

**if show: plt.show()**



from fuzzy\_logic.fuzzy\_variable\_output import FuzzyOutputVariable from fuzzy\_logic.fuzzy\_variable\_input import FuzzyInputVariable

# from fuzzy\_system.fuzzy\_variable import FuzzyVariable from fuzzy\_logic.fuzzy\_system import FuzzySystem

temp = FuzzyInputVariable('Temperature', 10, 40, 100)

temp.add\_triangular('Cold', 10, 10, 25)

temp.add\_triangular('Medium', 15, 25, 35)

temp.add\_triangular('Hot', 25, 40, 40)

humidity = FuzzyInputVariable('Humidity', 20, 100, 100)

humidity.add\_triangular('Wet', 20, 20, 60)

humidity.add\_trapezoidal('Normal', 30, 50, 70, 90)

humidity.add\_triangular('Dry', 60, 100, 100)

motor\_speed = FuzzyOutputVariable('Speed', 0, 100, 100)

motor\_speed.add\_triangular('Slow', 0, 0, 50)

motor\_speed.add\_triangular('Moderate', 10, 50, 90)

motor\_speed.add\_triangular('Fast', 50, 100, 100)

system = FuzzySystem() system.add\_input\_variable(temp) system.add\_input\_variable(humidity) system.add\_output\_variable(motor\_speed)

system.add\_rule(

{ 'Temperature':'Cold',

'Humidity':'Wet' },

{ 'Speed':'Slow'})

system.add\_rule(

{ 'Temperature':'Cold',

'Humidity':'Normal' },

{ 'Speed':'Slow'})

system.add\_rule(

{ 'Temperature':'Medium', 'Humidity':'Wet' },

{ 'Speed':'Slow'})

system.add\_rule(

{ 'Temperature':'Medium', 'Humidity':'Normal' },

{ 'Speed':'Moderate'})

system.add\_rule(

{ 'Temperature':'Cold',

'Humidity':'Dry' },

{ 'Speed':'Moderate'})

system.add\_rule(

{ 'Temperature':'Hot',

'Humidity':'Wet' },

{ 'Speed':'Moderate'})

system.add\_rule(

{ 'Temperature':'Hot',

'Humidity':'Normal' },

{ 'Speed':'Fast'})

system.add\_rule(

{ 'Temperature':'Hot',

'Humidity':'Dry' },

{ 'Speed':'Fast'})

system.add\_rule(

{ 'Temperature':'Medium', 'Humidity':'Dry' },

{ 'Speed':'Fast'})

output = system.evaluate\_output({ 'Temperature':18, 'Humidity':60

**})**

print(output)

# print('fuzzification\n \n', info['fuzzification'])

# print('rules\n \n', info['rules'])

system.plot\_system()



**from fuzzy\_logic.fuzzy\_variable\_output import FuzzyOutputVariable from fuzzy\_logic.fuzzy\_variable\_input import FuzzyInputVariable**

**from fuzzy\_logic.fuzzy\_system import FuzzySystem**

x1 = FuzzyInputVariable('x1', 0, 100, 100)

x1.add\_triangular('S', 0, 25, 50)

x1.add\_triangular('M', 25, 50, 75)

x1.add\_triangular('L', 50, 75, 100)

x2 = FuzzyInputVariable('x2', 0, 100, 100)

x2.add\_triangular('S', 0, 25, 50)

x2.add\_triangular('M', 25, 50, 75)

x2.add\_triangular('L', 50, 75, 100)

y = FuzzyOutputVariable('y', 0, 100, 100)

y.add\_triangular('S', 0, 25, 50)

y.add\_triangular('M', 25, 50, 75)

y.add\_triangular('L', 50, 75, 100)

z = FuzzyOutputVariable('z', 0, 100, 100)

z.add\_triangular('S', 0, 25, 50)

z.add\_triangular('M', 25, 50, 75)

z.add\_triangular('L', 50, 75, 100)

system = FuzzySystem() system.add\_input\_variable(x1) system.add\_input\_variable(x2) system.add\_output\_variable(y) system.add\_output\_variable(z)

system.add\_rule(

{ 'x1':'S',

'x2':'S' },

{ 'y':'S',

'z':'L' })

system.add\_rule(

{ 'x1':'M',

'x2':'M' },

{ 'y':'M',

'z':'M' })

system.add\_rule(

{ 'x1':'L',

'x2':'L' },

{ 'y':'L',

'z':'S' })

system.add\_rule(

{ 'x1':'S',

'x2':'M' },

{ 'y':'S',

'z':'L' })

system.add\_rule(

{ 'x1':'M',

'x2':'S' },

{ 'y':'S',

'z':'L' })

**system.add\_rule(**

**{ 'x1':'L',**

**'x2':'M' },**

**{ 'y':'L',**

**'z':'S' })**

**system.add\_rule(**

**{ 'x1':'M',**

**'x2':'L' },**

**{ 'y':'L',**

**'z':'S' })**

**system.add\_rule(**

**{ 'x1':'L',**

**'x2':'S' },**

**{ 'y':'M',**

**'z':'M' })**

**system.add\_rule(**

**{ 'x1':'S',**

**'x2':'L' },**

**{ 'y':'M',**

**'z':'M' })**

**output = system.evaluate\_output({ 'x1':35,**

**'x2':75**

**})**

**print(output)**







***ImplementedFuzzyInferenceSystem.***



 ***SolveFuzzyControlSystems:TheTippingProblem.***



***The‘tippingproblem’iscommonlyusedtoillustratethepoweroffuzzylogic principlestogeneratecomplexbehaviorfromacompact,intuitivesetofexpertrules.***

***Ifyou’renewtotheworldoffuzzycontrolsystems,youmightwanttocheckout***

***theFuzzyControlPrimerbeforereadingthroughthisworkedexample.***



***Let’screateafuzzycontrolsystemwhichmodelshowyoumightchoosetotipata restaurant.Whentipping,youconsidertheserviceandfoodquality,ratedbetween0 and10.Youusethistoleaveatipofbetween0and25%.***

***Wewouldformulatethisproblemas:***



* ***Universe(ie,crispvaluerange):Howgoodwastheserviceofthe waitstaff,onascaleof0to10?***
* ***Fuzzyset(ie,fuzzyvaluerange):poor,acceptable,amazing***





* ***Universe:Howtastywasthefood,onascaleof0to10?***
* ***Fuzzyset:bad,decent,great***



* ***Universe:Howmuchshouldwetip,onascaleof0%to25%***
* ***Fuzzyset:low,medium,high***



* ***IFtheservicewasgoodorthefoodqualitywasgood,THENthetipwill behigh.***
* ***IFtheservicewasaverage,THENthetipwillbemedium.***
* ***IFtheservicewaspoorandthefoodqualitywaspoorTHENthetipwill below.***



* + ***theserviceas9.8,and***
  + ***thequalityas6.5,***



* + ***a20.2%tip.***



***WecanusetheskfuzzycontrolsystemAPItomodelthis.First,let’sdefinefuzzy variables***

import numpy as np import skfuzzy as fuzz

from skfuzzy import control as ctrl

# New Antecedent/Consequent objects hold universe variables and membership # functions

quality = ctrl.Antecedent(np.arange(0, 11, 1), 'quality')

service = ctrl.Antecedent(np.arange(0, 11, 1), 'service')

tip = ctrl.Consequent(np.arange(0, 26, 1), 'tip')

# Auto-membership function population is possible with .automf(3, 5, or 7) quality.automf(3)

service.automf(3)

# Custom membership functions can be built interactively with a familiar, # Pythonic API

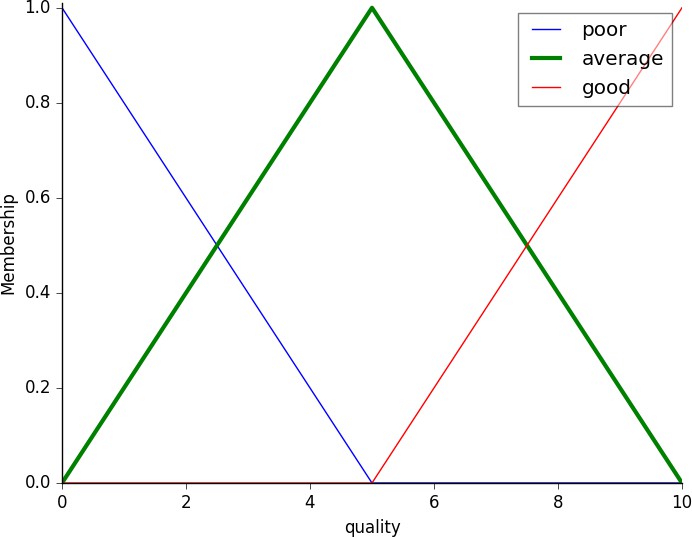
tip['low'] = fuzz.trimf(tip.universe, [0, 0, 13])

tip['medium'] = fuzz.trimf(tip.universe, [0, 13, 25])

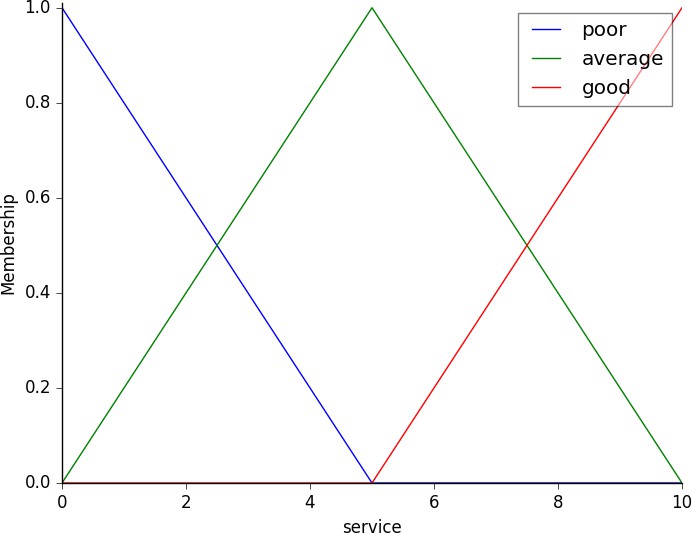
tip['high'] = fuzz.trimf(tip.universe, [13, 25, 25])

***#Youcanseehowtheselookwith.view()***

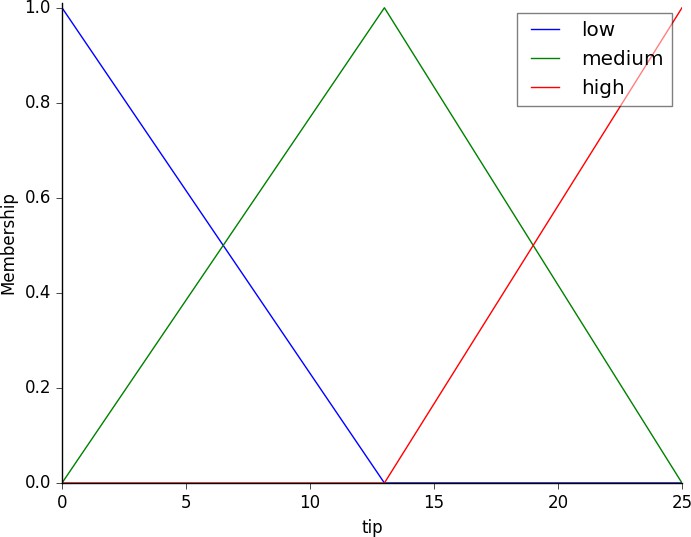
***quality['average'].view()***



**service.view()**



**tip.view()**

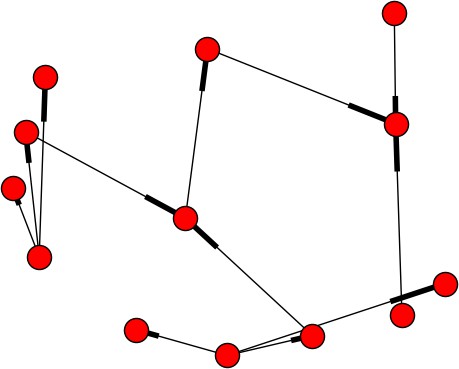




***Now,tomakethesetrianglesuseful,wedefinethefuzzyrelationshipbetweeninput andoutputvariables.Forthepurposesofourexample,considerthreesimplerules:***

1. ***IfthefoodispoorORtheserviceispoor,thenthetipwillbelow***
2. ***Iftheserviceisaverage,thenthetipwillbemedium***
3. ***IfthefoodisgoodORtheserviceisgood,thenthetipwillbehigh.***

***Mostpeoplewouldagreeontheserules,buttherulesarefuzzy.Mappingthe impreciserulesintoadefined,actionabletipisachallenge.Thisisthekindoftaskat whichfuzzylogicexcels.***



rule1 = ctrl.Rule(quality['poor'] | service['poor'], tip['low']) rule2 = ctrl.Rule(service['average'], tip['medium'])

rule3 = ctrl.Rule(service['good'] | quality['good'], tip['high'])

rule1.view()



***Nowthatwehaveourrulesdefined, wecansimplycreateacontrol systemvia:***

***tipping\_ctrl= ctrl.ControlSystem([rule1,rule2, rule3])***

***Inordertosimulatethiscontrol system,wewillcreate aControlSystemSimulation.Think ofthisobjectrepresentingour***

***controllerappliedtoaspecificsetofcirucmstances.Fortipping,thismightbe tippingSharonatthelocalbrew-pub.Wewouldcreate anotherControlSystemSimulationwhenwe’retryingtoapplyourtipping\_ctrlfor Travisatthecafebecausetheinputswouldbedifferent.***

***tipping=ctrl.ControlSystemSimulation(tipping\_ctrl)***

***Wecannowsimulateourcontrolsystembysimplyspecifyingtheinputsandcalling thecomputemethod.Supposeweratedthequality6.5outof10andtheservice9.8 of10.***

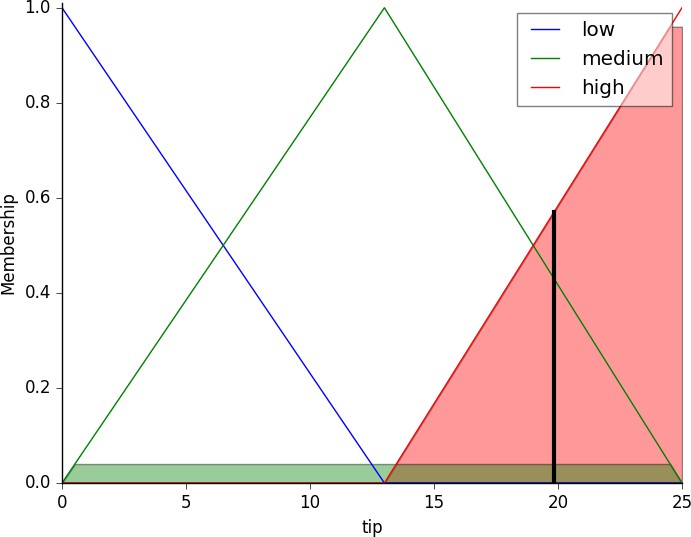
# Pass inputs to the ControlSystem using Antecedent labels with Pythonic API # Note: if you like passing many inputs all at once, use .inputs(dict\_of\_data) tipping.input['quality'] = 6.5

tipping.input['service'] = 9.8

# Crunch the numbers tipping.compute()

***Oncecomputed,wecanviewtheresultaswellasvisualizeit.***

print tipping.output['tip'] tip.view(sim=tipping)



***Theresultingsuggestedtipis.***

***Thepoweroffuzzysystemsisallowingcomplicated,intuitivebehaviorbasedona sparsesystemofruleswithminimaloverhead.Noteourmembershipfunction universeswerecoarse,onlydefinedattheintegers, butfuzz.interp\_membershipallowedtheeffectiveresolutiontoincreaseondemand. Thissystemcanrespondtoarbitrarilysmallchangesininputs,andtheprocessing burdenisminimal.***



**import numpy as np import skfuzzy as fuzz**

**from skfuzzy import control as ctrl import matplotlib.pyplot as plt**

**# New Antecedent/Consequent objects hold universe variables and membership # functions**

**quality = ctrl.Antecedent(np.arange(0, 11, 1), 'quality')**

**service = ctrl.Antecedent(np.arange(0, 11, 1), 'service')**

**tip = ctrl.Consequent(np.arange(0, 26, 1), 'tip')**

**# Auto-membership function population is possible with .automf(3, 5, or 7) quality.automf(3)**

**service.automf(3)**

**# Custom membership functions can be built interactively with a familiar, # Pythonic API**

**tip['low'] = fuzz.trimf(tip.universe, [0, 0, 13])**

**tip['medium'] = fuzz.trimf(tip.universe, [0, 13, 25])**

**tip['high'] = fuzz.trimf(tip.universe, [13, 25, 25])**

**# You can see how these look with .view() quality['average'].view()**

**service.view() tip.view()**

**rule1 = ctrl.Rule(quality['poor'] | service['poor'], tip['low']) rule2 = ctrl.Rule(service['average'], tip['medium'])**

**rule3 = ctrl.Rule(service['good'] | quality['good'], tip['high'])**

**rule1.view()**

**tipping\_ctrl = ctrl.ControlSystem([rule1, rule2, rule3]) tipping = ctrl.ControlSystemSimulation(tipping\_ctrl)**

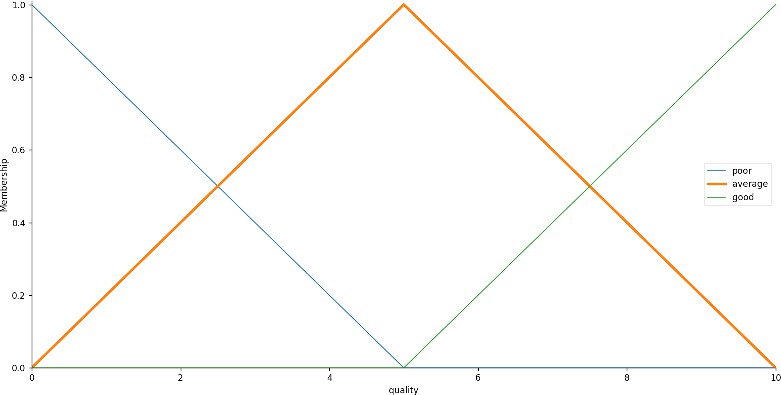
**# Pass inputs to the ControlSystem using Antecedent labels with Pythonic API # Note: if you like passing many inputs all at once, use .inputs(dict\_of\_data) tipping.input['quality'] = 6.5**

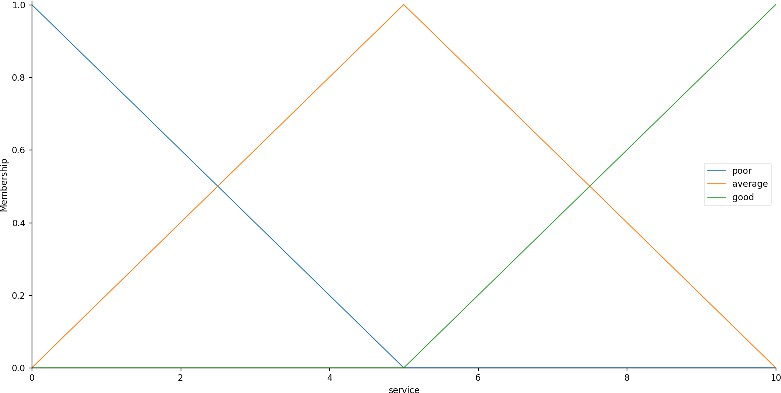
**tipping.input['service'] = 9.8**

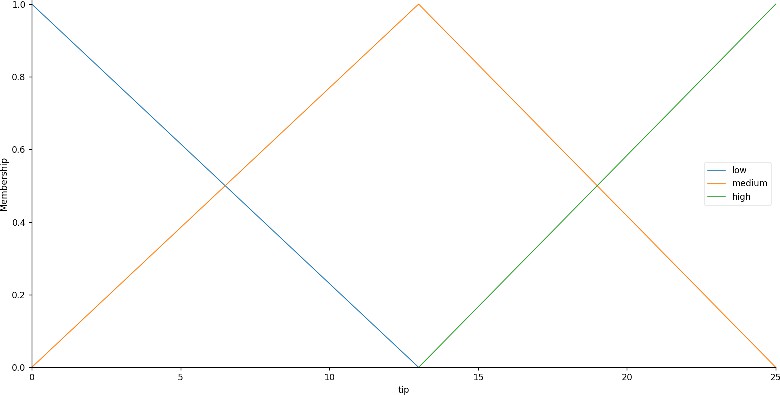
**# Crunch the numbers tipping.compute()**

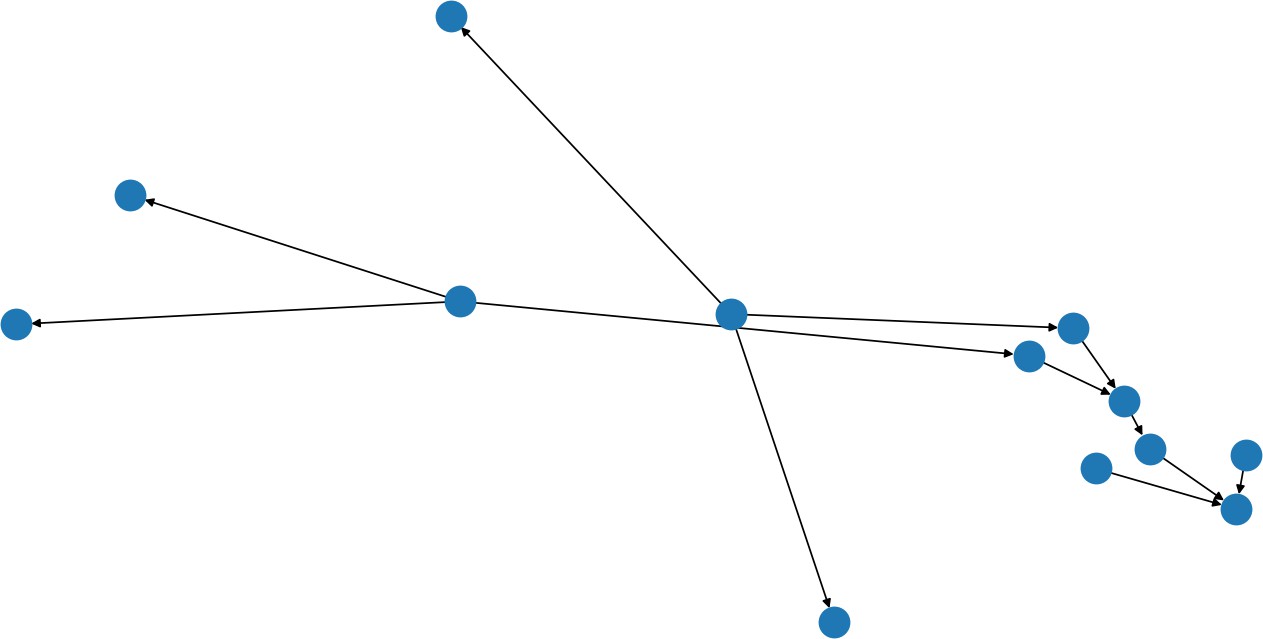
**print(tipping.output['tip']) tip.view(sim=tipping)**

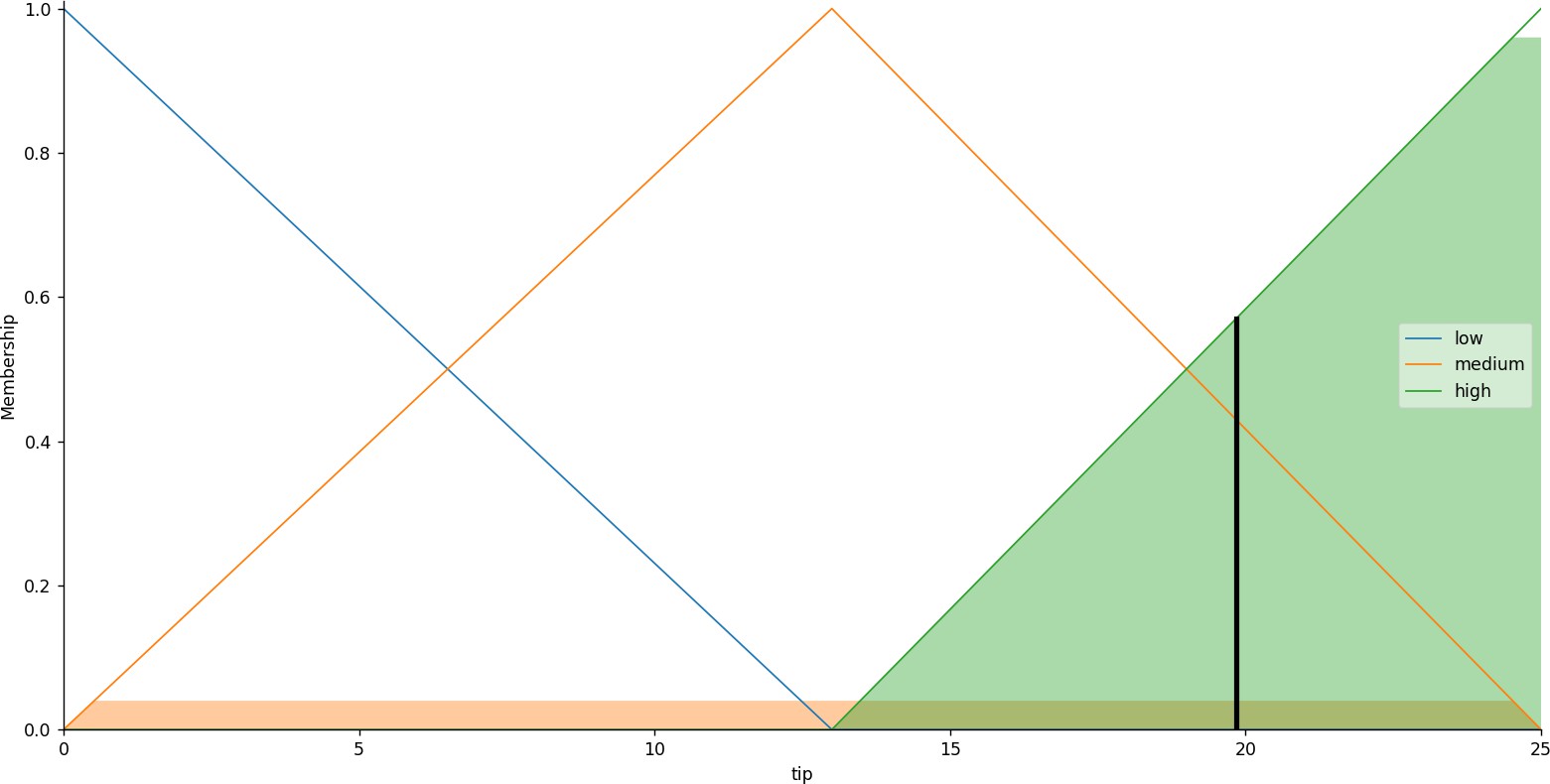
**plt.show()**











***SolvedFuzzyControlSystems:TheTippingProblem.***



 ***ImplementNaiveBayes’learningalgorithmfortherestaurantwaitingproblem.***



***NaïveBayesClassifierAlgorithm***

* ***NaïveBayesalgorithmisasupervisedlearningalgorithm,whichisbased on *** ***andusedforsolvingclassificationproblems.***
* ***Itismainlyusedintextclassificationthatincludesahigh-dimensionaltraining dataset.***
* ***NaïveBayesClassifierisoneofthesimpleandmosteffectiveClassification algorithmswhichhelpsinbuildingthefastmachinelearningmodelsthatcan makequickpredictions.***
* ***Itisaprobabilisticclassifier,whichmeansitpredictsonthebasisofthe probabilityofanobject.***
* ***SomepopularexamplesofNaïveBayesAlgorithmarespamfiltration, Sentimentalanalysis,andclassifyingarticles.***





***TheNaïveBayesalgorithmiscomprisedoftwowordsNaïveandBayes,Whichcan bedescribedas:***

* ***:ItiscalledNaïvebecauseitassumesthattheoccurrenceofacertain featureisindependentoftheoccurrenceofotherfeatures.Suchasifthefruit isidentifiedonthebasesofcolor,shape,andtaste,thenred,spherical,and sweetfruitisrecognizedasanapple.Henceeachfeatureindividually contributestoidentifythatitisanapplewithoutdependingoneachother.***
* ***:ItiscalledBayesbecauseitdependsontheprincipleofBayes' Theorem.***



* ***Bayes'theoremisalsoknownas *** ***or ,whichisusedto determinetheprobabilityofahypothesiswithpriorknowledge.Itdependson theconditionalprobability.***
* ***TheformulaforBayes'theoremisgivenas:***





***ofahypothesisistrue.***

***:ProbabilityofhypothesisAontheobservedeventB.***

***:Probabilityoftheevidencegiventhattheprobability***

***:Probabilityofhypothesisbeforeobservingtheevidence.***

***:ProbabilityofEvidence.***

***WorkingofNaïveBayes'Classifier:***

***Working of Naïve Bayes' Classifier can be understood with the help of the below example:***

***Supposewehaveadatasetof *** ***andcorrespondingtargetvariable " ".Sousingthisdatasetweneedtodecidethatwhetherweshouldplayornoton aparticulardayaccordingtotheweatherconditions.Sotosolvethisproblem, we needtofollowthebelowsteps:***

1. ***Convertthegivendatasetintofrequencytables.***
2. ***GenerateLikelihoodtablebyfindingtheprobabilitiesofgivenfeatures.***
3. ***Now,useBayestheoremtocalculatetheposteriorprobability.***

***:Iftheweatherissunny,thenthePlayershouldplayornot?***

***:Tosolvethis,firstconsiderthebelowdataset:***

|  |  |  |
| --- | --- | --- |
|  | | |
|  | ***Rainy*** | ***Yes*** |
|  | ***Sunny*** | ***Yes*** |
|  | ***Overcast*** | ***Yes*** |
|  | ***Overcast*** | ***Yes*** |
|  | ***Sunny*** | ***No*** |
|  | ***Rainy*** | ***Yes*** |
|  | ***Sunny*** | ***Yes*** |
|  | ***Overcast*** | ***Yes*** |
|  | ***Rainy*** | ***No*** |
|  | ***Sunny*** | ***No*** |
|  | ***Sunny*** | ***Yes*** |
|  | ***Rainy*** | ***No*** |
|  | ***Overcast*** | ***Yes*** |
|  | ***Overcast*** | ***Yes*** |



|  |  |  |
| --- | --- | --- |
| ***Weather*** | ***Yes*** | ***No*** |
| ***Overcast*** | ***5*** | ***0*** |
| ***Rainy*** | ***2*** | ***2*** |
| ***Sunny*** | ***3*** | ***2*** |



***Total***

***10***

***5***

|  |  |  |  |
| --- | --- | --- | --- |
| ***Weather*** | ***No*** | ***Yes*** |  |
| ***Overcast*** | ***0*** | ***5*** | ***5/14=0.35*** |
| ***Rainy*** | ***2*** | ***2*** | ***4/14=0.29*** |
| ***Sunny*** | ***2*** | ***3*** | ***5/14=0.35*** |
| ***All*** | ***4/14=0.29*** | ***10/14=0.71*** |  |



***P(Sunny|Yes)=3/10=0.3 P(Sunny)=0.35 P(Yes)=0.71***

***SoP(Yes|Sunny)=0.3\*0.71/0.35= ***



***P(Sunny|NO)=2/4=0.5 P(No)=0.29***

***P(Sunny)=0.35 SoP(No|Sunny)=0.5\*0.29/0.35= ***

***Soaswecanseefromtheabovecalculationthat ***



***AdvantagesofNaïveBayesClassifier:***

* ***NaïveBayesisoneofthefastandeasyMLalgorithmstopredictaclassof datasets.***
* ***ItcanbeusedforBinaryaswellasMulti-classClassifications.***
* ***It performs well in Multi-class predictions as compared to the other Algorithms.***
* ***Itisthemostpopularchoicefor .***

***DisadvantagesofNaïveBayesClassifier:***

* ***Naive Bayes assumes that all features are independent or unrelated, so it cannotlearntherelationshipbetweenfeatures.***



* ***Itisusedfor .***
* ***Itisusedin.***
* ***It canbeusedinbecause Naïve Bayes Classifieris an eagerlearner.***
* ***ItisusedinTextclassificationsuchas *** ***and .***



***TherearethreetypesofNaiveBayesModel,whicharegivenbelow:***

* ***: The Gaussian model assumes that features follow a normal distribution. This means if predictors take continuous values instead of discrete, then the model assumes that these values are sampled from the Gaussiandistribution.***
* ***:TheMultinomialNaïveBayesclassifierisusedwhenthedatais multinomialdistributed.Itisprimarilyusedfordocumentclassification problems,itmeansaparticulardocumentbelongstowhichcategorysuchas Sports,Politics,education,etc.***

***Theclassifierusesthefrequencyofwordsforthepredictors.***

* ***:TheBernoulliclassifierworkssimilartotheMultinomialclassifier, butthepredictorvariablesaretheindependentBooleansvariables.Suchasif aparticularwordispresentornotinadocument.Thismodelisalsofamous fordocumentclassificationtasks.***



**import numpy as nm**

**import matplotlib.pyplot as mtp import pandas as pd**

# Importing the dataset

dataset = pd.read\_csv('user\_data.csv') x = dataset.iloc[:, [2, 3]].values

y = dataset.iloc[:, 4].values

# Splitting the dataset into the Training set and Test set from sklearn.model\_selection import train\_test\_split

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size = 0.25, random\_state = 0)

# Feature Scaling

from sklearn.preprocessing import StandardScaler sc = StandardScaler()

x\_train = sc.fit\_transform(x\_train) x\_test = sc.transform(x\_test)

from sklearn.naive\_bayes import GaussianNB classifier = GaussianNB() classifier.fit(x\_train, y\_train)

# Predicting the Test set results y\_pred = classifier.predict(x\_test)

# Making the Confusion Matrix

from sklearn.metrics import confusion\_matrix cm = confusion\_matrix(y\_test, y\_pred)

# Visualising the Training set results

from matplotlib.colors import ListedColormap x\_set, y\_set = x\_train, y\_train

X1, X2 = nm.meshgrid(nm.arange(start = x\_set[:, 0].min() - 1, stop = x\_set[:, 0].max() + 1, step = 0.01),

nm.arange(start = x\_set[:, 1].min() - 1, stop = x\_set[:, 1].max() + 1, step = 0.01)) mtp.contourf(X1, X2, classifier.predict(nm.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),

alpha = 0.75, cmap = ListedColormap(('purple', 'green'))) mtp.xlim(X1.min(), X1.max())

mtp.ylim(X2.min(), X2.max())

for i, j in enumerate(nm.unique(y\_set)):

mtp.scatter(x\_set[y\_set == j, 0], x\_set[y\_set == j, 1], c = ListedColormap(('purple', 'green'))(i), label = j)

mtp.title('Naive Bayes (Training set)') mtp.xlabel('Age') mtp.ylabel('Estimated Salary') mtp.legend()

mtp.show()

# Visualising the Test set results

from matplotlib.colors import ListedColormap x\_set, y\_set = x\_test, y\_test

X1, X2 = nm.meshgrid(nm.arange(start = x\_set[:, 0].min() - 1, stop = x\_set[:, 0].max() + 1, step = 0.01),

nm.arange(start = x\_set[:, 1].min() - 1, stop = x\_set[:, 1].max() + 1, step = 0.01)) mtp.contourf(X1, X2, classifier.predict(nm.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),

alpha = 0.75, cmap = ListedColormap(('purple', 'green'))) mtp.xlim(X1.min(), X1.max())

mtp.ylim(X2.min(), X2.max())

for i, j in enumerate(nm.unique(y\_set)):

**mtp.scatter(x\_set[y\_set == j, 0], x\_set[y\_set == j, 1],**

**c = ListedColormap(('purple', 'green'))(i), label = j) mtp.title('Naive Bayes (test set)')**

**mtp.xlabel('Age') mtp.ylabel('Estimated Salary') mtp.legend()**

**mtp.show()**



User ID,Gender,Age,Estimated Salary,Purchased 15622,Male,36,15000,1

15669,Female,39,21000,1

15678,Male,31,15000,1

15622,Female,28,46000,0

15641,Female,32,42000,1

15647,Male,35,38000,0

15658,Female,32,28000,1

15650,Female,38,8000,1

15652,Male,21,12000,1

15618,Female,45,17000,0

15682,Female,34,13000,1

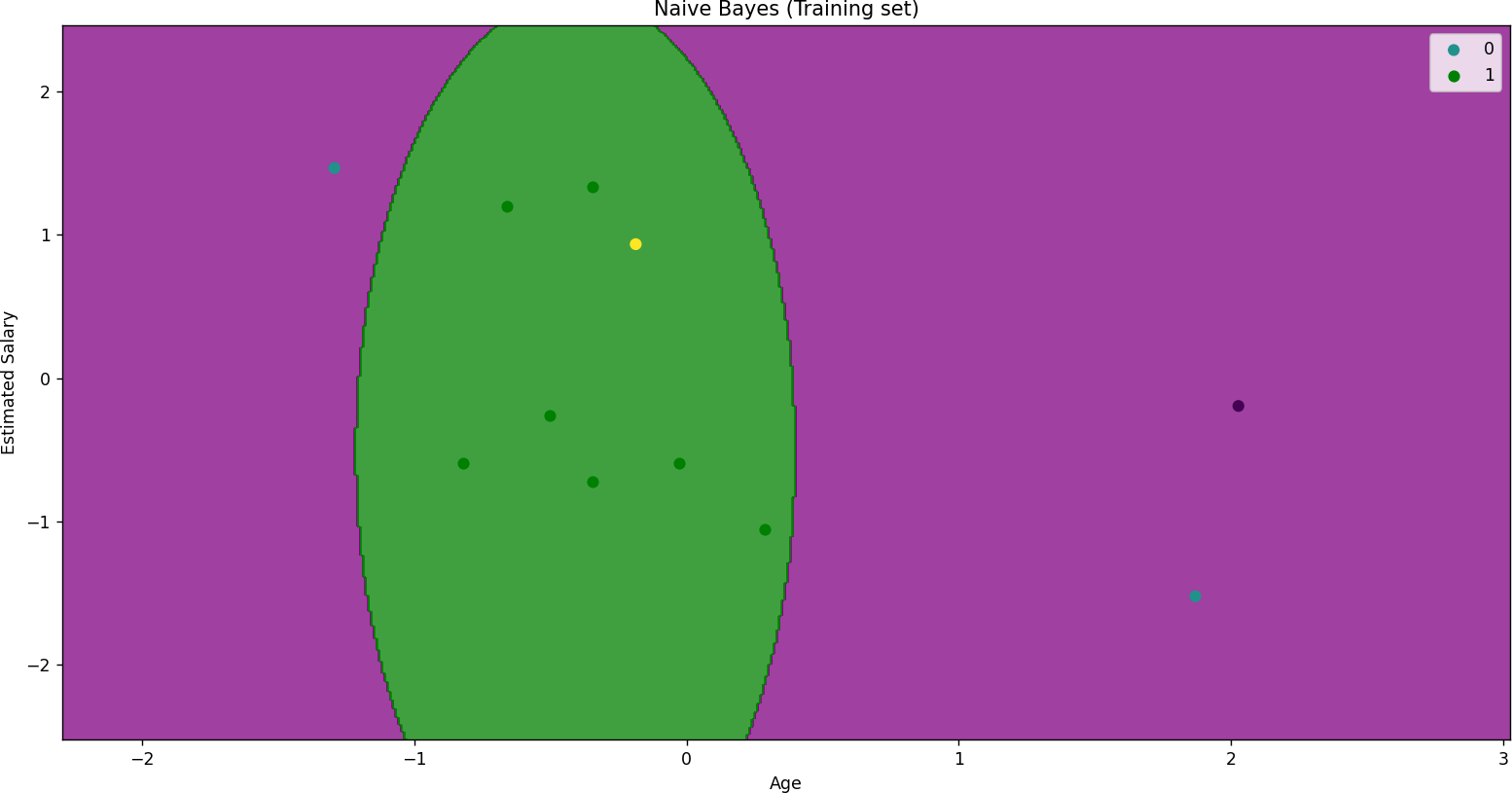
15696,Male,34,44000,1

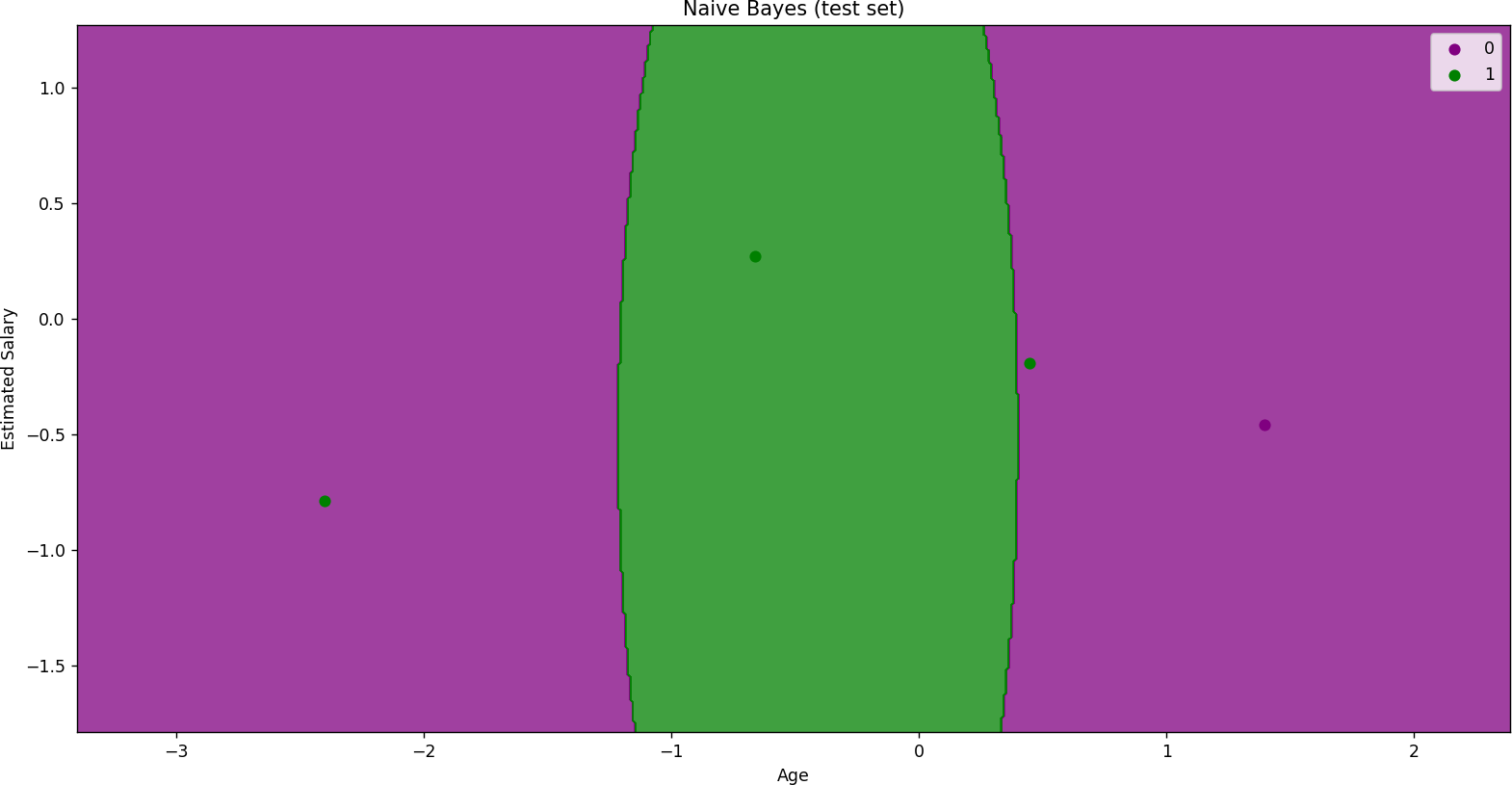
15682,Male,33,20000,1

15650,Female,49,21000,0

15609,Male,48,1000,0







***ImplementedNaiveBayes’learningalgorithmfortherestaurantwaitingproblem.***