|  |  |  |
| --- | --- | --- |
| **Ex.No.** | **Title of the Experiments** | **Page No** |
|  | Implement A\* Search algorithm. |  |
|  | Implement AO\* Search algorithm. |  |
|  | For a given set of training data examples stored in a .CSV file, implement and demonstrate the **Candidate-Elimination algorithm** to output a description of the set of all hypotheses consistent with the training examples. |  |
|  | Write a program to demonstrate the working of the decision tree based **ID3 algorithm.** Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample. |  |
|  | Build an Artificial Neural Network by implementing the **Backpropagation algorithm** and test the same using appropriate data sets. |  |
|  | Write a program to implement the **naïve Bayesian classifier** for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets. |  |
|  | Apply **EM algorithm** to cluster a set of data stored in a .CSV file. Use the same data set for clustering using **k-Means algorithm**. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program. |  |
|  | Write a program to implement **k-Nearest Neighbour algorithm** to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem. |  |
|  | Implement the non-parametric **Locally Weighted Regression algorithm** in order to fit data points. Select appropriate data set for your experiment and draw graphs. |  |

**INDEX**

1. Implement A\* Search algorithm.

**THEORY:**

**A\* Search Algorithm**

To choose an optimal path for destination, A\* is a part of informed search techniques in Artificial Intelligence. Informed search algorithm contains an array of knowledge such as how far we are from the destination, path cost, how to reach to destination node, etc.

      F (n) = G (n) + H (n) -------------------------------------------------------------- (1)

A\* Algorithm uses formula(1) to provide a optimal path.

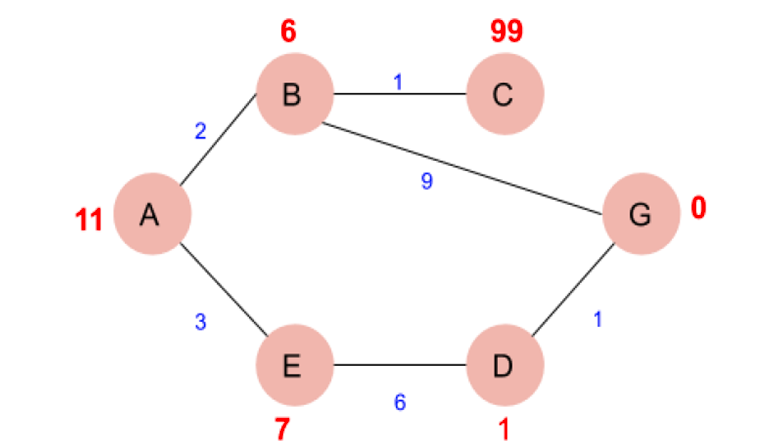
Here,

n-node (in current state)

F- Least cost from source (start node) to the destination (goal node)

G-Actual cost from start node to n node.

H-Estimated cost from n node to the Goal node (heuristic value of node).

****

Numbers written on edges represent the distance between nodes. Numbers written on nodes represent the heuristic value.

Given the graph, find the cost-effective path from A to G. That is A is the source node and G is the goal node.

Now from A, we can go to point B or E, so we compute f(x) for each of them,

A → B = g(B) + h(B) = 2 + 6 = 8

A → E = g(E) + h(E) = 3 + 7 = 10

Since the cost for A → B is less, we move forward with this path and compute the f(x) for the children nodes of B.

Now from B, we can go to point C or G, so we compute f(x) for each of them,

A → B → C = (2 + 1) + 99= 102

A → B → G = (2 + 9 ) + 0 = 11

Here the path A → B → G has the least cost but it is still more than the cost of A → E, thus we explore this path further.

Now from E, we can go to point D, so we compute f(x),

A → E → D = (3 + 6) + 1 = 10

Comparing the cost of A → E → D with all the paths we got so far and as this cost is least of all we move forward with this path.

Now compute the f(x) for the children of D

A → E → D → G = (3 + 6 + 1) +0 = 10

Now comparing all the paths that lead us to the goal, we conclude that **A → E → D → G** is the most cost-effective path to get from A to G.

**PROCEDURE / PROGRAMME**:

import os

clear=lambda : os.system('cls')

clear()

def aStarAlgo(start\_node, stop\_node):

open\_set = set(start\_node)

closed\_set = set()

g = {}

parents = {}

g[start\_node] = 0

parents[start\_node] = start\_node

while len(open\_set) > 0:

n = None

for v in open\_set:

if n == None or g[v] + heuristic(v) < g[n] + heuristic(n):

n = v

if n == stop\_node or Graph\_nodes[n] == None:

pass

else:

for (m, weight) in get\_neighbors(n):

if m not in open\_set and m not in closed\_set:

open\_set.add(m)

parents[m] = n

g[m] = g[n] + weight

else:

if g[m] > g[n] + weight:

g[m] = g[n] + weight

parents[m] = n

if m in closed\_set:

closed\_set.remove(m)

open\_set.add(m)

if n == None:

print('Path does not exist!')

return None

if n == stop\_node:

path = []

while parents[n] != n:

path.append(n)

n = parents[n]

path.append(start\_node)

path.reverse()

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

return path

open\_set.remove(n)

closed\_set.add(n)

print('Path does not exist!')

return None

def get\_neighbors(v):

if v in Graph\_nodes:

return Graph\_nodes[v]

else:

return None

def heuristic(n):

H\_dist = {

'A': 11,

'B': 6,

'C': 99,

'D': 1,

'E': 7,

'G': 0

}

return H\_dist[n]

Graph\_nodes = {

'A': [('E', 3), ('B',2)],

'B': [('C', 1),('G',9)] ,

'E': [('D', 6)],

'D': [('G', 1)],

}

aStarAlgo('A', 'G')

**OUTPUT:**

Path found: ['A', 'E', 'D', 'G']

OUTPUT: ['A', 'E', 'D', 'G']

**2. Implement AO\* Search algorithm.**

**THEORY:**

AO\* Algorithm basically based on problem decomposition. When a problem can be divided into a set of sub problems, where each sub problem can be solved separately and a combination of these will be a solution. AND-OR graphs are used for representing the solution.  The decomposition of the problem or problem reduction generates AND arcs.

**How AO\* works:**

The algorithm always moves towards a lower cost value.

Basically, We will calculate the cost function here (F(n)= G (n) + H (n))

H:  heuristic/ estimated value of the nodes and G: actual cost or edge value.

Here we have taken the edges value 1, meaning we have to focus solely on the heuristic value.

1. The Purple color values are edge values (here all are same that is one).

2. The Red color values are Heuristic values for nodes.

3. The Green color values are New Heuristic values for nodes.

**PROCEDURE / PROGRAM :**

**class**Graph:

**def**\_\_init\_\_(self, graph, heuristicNodeList, startNode):  *#instantiate graph object with graph topology, heuristic values, start node*

        self.graph= graph

        self.H=heuristicNodeList

        self.start=startNode

        self.parent={}

        self.status={}

        self.solutionGraph={}

**def** applyAOStar(self):

        self.aoStar(self.start, False)

**def** getNeighbors(self, v):

        return self.graph.get(v,'')

**def** getStatus(self,v):

        return self.status.get(v,0)

**def**setStatus(self,v, val):

        self.status[v]=val

**def** getHeuristicNodeValue(self, n):

        return self.H.get(n,0)

**def** setHeuristicNodeValue(self, n, value):

        self.H[n]=value

**def** printSolution(self):

        print("FOR GRAPH SOLUTION, TRAVERSE THE GRAPH FROM THE START

NODE:",self.start)

        print("------------------------------------------------------------")

        print(self.solutionGraph)

        print("------------------------------------------------------------")

**def** computeMinimumCostChildNodes(self, v):

        minimumCost=0

        costToChildNodeListDict={}

        costToChildNodeListDict[minimumCost]=[]

        flag=True

        for nodeInfoTupleListinself.getNeighbors(v):

            cost=0

            nodeList=[]

            for c, weight innodeInfoTupleList:

                cost=cost+self.getHeuristicNodeValue(c)+weight

                nodeList.append(c)

            if flag==True:

                costToChildNodeListDict[minimumCost]=nodeList

                flag=False

            else:

                if minimumCost>cost:

                    minimumCost=cost

                    costToChildNodeListDict[minimumCost]=nodeList

        return minimumCost, costToChildNodeListDict[minimumCost]

**def** aoStar(self, v, backTracking):

        print("HEURISTIC VALUES  :", self.H)

        print("SOLUTION GRAPH    :", self.solutionGraph)

        print("PROCESSING NODE   :", v)

        print("-----------------------------------------------------------------------------------------")

        if self.getStatus(v) >=0:

            minimumCost, childNodeList=self.computeMinimumCostChildNodes(v)

            self.setHeuristicNodeValue(v, minimumCost)

            self.setStatus(v,len(childNodeList))

            solved=True

            forchildNodeinchildNodeList:

                self.parent[childNode]=v

                ifself.getStatus(childNode)!=-1:

                    solved=solved &False

            if solved==True:

                self.setStatus(v,-1)

                self.solutionGraph[v]=childNodeList

            if v!=self.start:

                self.aoStar(self.parent[v], True)

            if backTracking==False:

                forchildNodeinchildNodeList:

                    self.setStatus(childNode,0)

                    self.aoStar(childNode, False)

h1 = {'A': 1, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1, 'T': 3}

graph1 = {

    'A': [[('B', 1), ('C', 1)], [('D', 1)]],

    'B': [[('G', 1)], [('H', 1)]],

    'C': [[('J', 1)]],

    'D': [[('E', 1), ('F', 1)]],

    'G': [[('I', 1)]]

}

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

G1.applyAOStar()

G1.printSolution()

h2 = {'A': 1, 'B': 6, 'C': 12, 'D': 10, 'E': 4, 'F': 4, 'G': 5, 'H': 7}

graph2 = {

    'A': [[('B', 1), ('C', 1)], [('D', 1)]],

    'B': [[('G', 1)], [('H', 1)]],

    'D': [[('E', 1), ('F', 1)]]

}

G2 =Graph(graph2, h2, 'A')

G2.applyAOStar()

G2.printSolution()

**OUTPUT:**

HEURISTIC VALUES : {'A': 1, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1, 'T': 3}

SOLUTION GRAPH : {}

PROCESSING NODE : A

-----------------------------------------------------------------------------------------

HEURISTIC VALUES : {'A': 0, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1, 'T': 3}

SOLUTION GRAPH : {}

PROCESSING NODE : B

-----------------------------------------------------------------------------------------

HEURISTIC VALUES : {'A': 0, 'B': 0, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1, 'T': 3}

SOLUTION GRAPH : {}

PROCESSING NODE : A

-----------------------------------------------------------------------------------------

HEURISTIC VALUES : {'A': 0, 'B': 0, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1, 'T': 3}

SOLUTION GRAPH : {}

PROCESSING NODE : G

-----------------------------------------------------------------------------------------

HEURISTIC VALUES : {'A': 0, 'B': 0, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 0, 'H': 7, 'I': 7, 'J': 1, 'T': 3}

SOLUTION GRAPH : {}

PROCESSING NODE : B

-----------------------------------------------------------------------------------------

HEURISTIC VALUES : {'A': 0, 'B': 0, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 0, 'H': 7, 'I': 7, 'J': 1, 'T': 3}

SOLUTION GRAPH : {}

PROCESSING NODE : A

-----------------------------------------------------------------------------------------

HEURISTIC VALUES : {'A': 0, 'B': 0, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 0, 'H': 7, 'I': 7, 'J': 1, 'T': 3}

SOLUTION GRAPH : {}

PROCESSING NODE : I

-----------------------------------------------------------------------------------------

HEURISTIC VALUES : {'A': 0, 'B': 0, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 0, 'H': 7, 'I': 0, 'J': 1, 'T': 3}

SOLUTION GRAPH : {'I': []}

PROCESSING NODE : G

-----------------------------------------------------------------------------------------

HEURISTIC VALUES : {'A': 0, 'B': 0, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 0, 'H': 7, 'I': 0, 'J': 1, 'T': 3}

SOLUTION GRAPH : {'I': [], 'G': ['I']}

PROCESSING NODE : B

-----------------------------------------------------------------------------------------

HEURISTIC VALUES : {'A': 0, 'B': 0, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 0, 'H': 7, 'I': 0, 'J': 1, 'T': 3}

SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G']}

PROCESSING NODE : A

-----------------------------------------------------------------------------------------

HEURISTIC VALUES : {'A': 0, 'B': 0, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 0, 'H': 7, 'I': 0, 'J': 1, 'T': 3}

SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G']}

PROCESSING NODE : C

-----------------------------------------------------------------------------------------

HEURISTIC VALUES : {'A': 0, 'B': 0, 'C': 0, 'D': 12, 'E': 2, 'F': 1, 'G': 0, 'H': 7, 'I': 0, 'J': 1, 'T': 3}

SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G']}

PROCESSING NODE : A

-----------------------------------------------------------------------------------------

HEURISTIC VALUES : {'A': 0, 'B': 0, 'C': 0, 'D': 12, 'E': 2, 'F': 1, 'G': 0, 'H': 7, 'I': 0, 'J': 1, 'T': 3}

SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G']}

PROCESSING NODE : J

-----------------------------------------------------------------------------------------

HEURISTIC VALUES : {'A': 0, 'B': 0, 'C': 0, 'D': 12, 'E': 2, 'F': 1, 'G': 0, 'H': 7, 'I': 0, 'J': 0, 'T': 3}

SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G'], 'J': []}

PROCESSING NODE : C

-----------------------------------------------------------------------------------------

HEURISTIC VALUES : {'A': 0, 'B': 0, 'C': 0, 'D': 12, 'E': 2, 'F': 1, 'G': 0, 'H': 7, 'I': 0, 'J': 0, 'T': 3}

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

PROCESSING NODE : A

-----------------------------------------------------------------------------------------

FOR GRAPH SOLUTION, TRAVERSE THE GRAPH FROM THE START NODE: A

------------------------------------------------------------

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

------------------------------------------------------------

HEURISTIC VALUES : {'A': 1, 'B': 6, 'C': 12, 'D': 10, 'E': 4, 'F': 4, 'G': 5, 'H': 7}

SOLUTION GRAPH : {}

PROCESSING NODE : A

-----------------------------------------------------------------------------------------

HEURISTIC VALUES : {'A': 0, 'B': 6, 'C': 12, 'D': 10, 'E': 4, 'F': 4, 'G': 5, 'H': 7}

SOLUTION GRAPH : {}

PROCESSING NODE : B

-----------------------------------------------------------------------------------------

HEURISTIC VALUES : {'A': 0, 'B': 0, 'C': 12, 'D': 10, 'E': 4, 'F': 4, 'G': 5, 'H': 7}

SOLUTION GRAPH : {}

PROCESSING NODE : A

-----------------------------------------------------------------------------------------

HEURISTIC VALUES : {'A': 0, 'B': 0, 'C': 12, 'D': 10, 'E': 4, 'F': 4, 'G': 5, 'H': 7}

SOLUTION GRAPH : {}

PROCESSING NODE : G

-----------------------------------------------------------------------------------------

HEURISTIC VALUES : {'A': 0, 'B': 0, 'C': 12, 'D': 10, 'E': 4, 'F': 4, 'G': 0, 'H': 7}

SOLUTION GRAPH : {'G': []}

PROCESSING NODE : B

-----------------------------------------------------------------------------------------

HEURISTIC VALUES : {'A': 0, 'B': 0, 'C': 12, 'D': 10, 'E': 4, 'F': 4, 'G': 0, 'H': 7}

SOLUTION GRAPH : {'G': [], 'B': ['G']}

PROCESSING NODE : A

-----------------------------------------------------------------------------------------

HEURISTIC VALUES : {'A': 0, 'B': 0, 'C': 12, 'D': 10, 'E': 4, 'F': 4, 'G': 0, 'H': 7}

SOLUTION GRAPH : {'G': [], 'B': ['G']}

PROCESSING NODE : C

-----------------------------------------------------------------------------------------

HEURISTIC VALUES : {'A': 0, 'B': 0, 'C': 0, 'D': 10, 'E': 4, 'F': 4, 'G': 0, 'H': 7}

SOLUTION GRAPH : {'G': [], 'B': ['G'], 'C': []}

PROCESSING NODE : A

-----------------------------------------------------------------------------------------

FOR GRAPH SOLUTION, TRAVERSE THE GRAPH FROM THE START NODE: A

------------------------------------------------------------

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

------------------------------------------------------------

3. For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.

import random

import csv

def g\_0(n):

return ("?",)\*n

def s\_0(n):

return ('0',)\*n

defmore\_general(h1, h2):

more\_general\_parts = []

for x, y in zip(h1, h2):

mg = x == "?" or (x != "0" and (x == y or y == "0"))

more\_general\_parts.append(mg)

return all(more\_general\_parts)

# min\_generalizations

deffulfills(example, hypothesis):

### the implementation is the same as for hypotheses:

returnmore\_general(hypothesis, example)

defmin\_generalizations(h, x):

h\_new = list(h)

for i in range(len(h)):

if not fulfills(x[i:i+1], h[i:i+1]):

h\_new[i] = '?' if h[i] != '0' else x[i]

return [tuple(h\_new)]

defmin\_specializations(h, domains, x):

results = []

for i in range(len(h)):

if h[i] == "?":

forval in domains[i]:

if x[i] != val:

h\_new = h[:i] + (val,) + h[i+1:]

results.append(h\_new)

elif h[i] != "0":

h\_new = h[:i] + ('0',) + h[i+1:]

results.append(h\_new)

return results

with open('data2.csv') as csvFile:

examples = [tuple(line) for line in csv.reader(csvFile)]

defget\_domains(examples):

d = [set() for i in examples[0]]

for x in examples:

for i, xi in enumerate(x):

d[i].add(xi)

return [list(sorted(x)) for x in d]

defcandidate\_elimination(examples):

domains = get\_domains(examples)[:-1]

G = set([g\_0(len(domains))])

S = set([s\_0(len(domains))])

i=0

print("\n G[{0}]:".format(i),G)

print("\n S[{0}]:".format(i),S)

forxcx in examples:

i=i+1

x, cx = xcx[:-1], xcx[-1] # Splitting data into attributes and decisions

if cx=='Y': # x is positive example

G = {g for g in G if fulfills(x, g)}

S = generalize\_S(x, G, S)

else: # x is negative example

S = {s for s in S if not fulfills(x, s)}

G = specialize\_G(x, domains, G, S)

print("\n G[{0}]:".format(i),G)

print("\n S[{0}]:".format(i),S)

print()

return

defgeneralize\_S(x, G, S):

S\_prev = list(S)

for s in S\_prev:

if s not in S:

continue

if not fulfills(x, s):

S.remove(s)

Splus = min\_generalizations(s, x)

## keep only generalizations that have a counterpart in G

S.update([h for h in Splus if any([more\_general(g,h)

for g in G])])

## remove hypotheses less specific than any other in S

S.difference\_update([h for h in S if

any([more\_general(h, h1)

for h1 in S if h != h1])])

return S

defspecialize\_G(x, domains, G, S):

G\_prev = list(G)

for g in G\_prev:

if g not in G:

continue

iffulfills(x, g):

G.remove(g)

Gminus = min\_specializations(g, domains, x)

## keep only specializations that have a conuterpart in S

G.update([h for h in Gminus if any([more\_general(h, s)

for s in S])])

## remove hypotheses less general than any other in G

G.difference\_update([h for h in G if

any([more\_general(g1, h)

for g1 in G if h != g1])])

return G

candidate\_elimination(examples)

data2.csv

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Sunny | warm | Normal | strong | warm | same | Y |
| Sunny | warm | High | strong | warm | same | Y |
| Rainy | cold | High | strong | warm | change | N |
| Sunny | warm | High | strong | cool | change | Y |

OUTPUT

G[0]: {('?', '?', '?', '?', '?', '?')}

S[0]: {('0', '0', '0', '0', '0', '0')}

G[1]: {('?', '?', '?', '?', '?', '?')}

S[1]: {('sunny', 'warm', 'normal', 'strong', 'warm', 'same')}

G[2]: {('?', '?', '?', '?', '?', '?')}

S[2]: {('sunny', 'warm', '?', 'strong', 'warm', 'same')}

G[3]: {('sunny', '?', '?', '?', '?', '?'), ('?', 'warm', '?', '?', '?', '?'), ('?', '?', '?', '?', '?', 'same')}

S[3]: {('sunny', 'warm', '?', 'strong', 'warm', 'same')}

G[4]: {('sunny', '?', '?', '?', '?', '?'), ('?', 'warm', '?', '?', '?', '?')}

S[4]: {('sunny', 'warm', '?', 'strong', '?', '?')}

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

THEORY:The ID3 algorithm begins with the original set as the root node. On each iteration of the algorithm, it iterates through every unused attribute of the set and calculates the entropy or the information gain of that attribute. It then selects the attribute which has the smallest entropy (or largest information gain) value. The set is then split or partitioned by the selected attribute to produce subsets of the data. (For example, a node can be split into child nodes based upon the subsets of the population whose ages are less than 50, between 50 and 100, and greater than 100.) The algorithm continues to recurse on each subset, considering only attributes never selected before.

* Calculate the entropy of every attribute {\displaystylea}a of the data set {\displaystyle S}S.
* Partition ("split") the set {\displaystyleS}S into subsets using the attribute for which the resulting entropy after splitting is minimized; or, equivalently, information gain is maximum.
* Make a decision tree node containing that attribute.
* Recurse on subsets using the remaining attributes.

The ID3 algorithm is used by training on a data set {\displaystyleS}S to produce a decision tree which is stored in memory. At runtime, this decision tree is used to classify new test cases (feature vectors) by traversing the decision tree using the features of the datum to arrive at a leaf node.

**PROCEDURE/PROGRAM:**

import pandas as pd

df =pd.read\_csv('PlayTennis.csv')

print("\n Input Data Set is:\n", df)

Input Data Set is:

outlook tempearture humidity wind playtennis

0 sunny hot high weak no

1 sunny hot high strong no

2 overcast hot high weak yes

3 rain mild high weak yes

4 rain cool normal weak yes

5 rain cool normal strong no

6 overcast cool normal strong yes

7 sunny mild high weak no

8 sunny cool normal weak yes

9 rain mild normal weak yes

10 sunny mild normal strong yes

11 overcast mild high strong yes

12 overcast hot normal weak yes

13 rain mild high strong no

t=df.keys()[-1]

print('Target Attribute is: ', t)

attribute\_names=list(df.keys())

attribute\_names.remove(t)

print('Predicting Attributes: ', attribute\_names)

Target Attribute is: playtennis

Predicting Attributes: ['outlook', 'tempearture', 'humidity', 'wind']

import math

**def**entropy(probs):

    returnsum( [-prob\*math.log(prob, 2) for prob in probs])

**def**entropy\_of\_list(ls,value):

    from collections import Counter

    cnt=Counter(x for x in ls)

    print('Target attribute class count(Yes/No)=',dict(cnt))

    total\_instances=len(ls)

    print("Total no of instances/records associated with {0} is: {1}".format(value,total\_instances ))

    probs = [x /total\_instancesfor x incnt.values()]

    print("Probability of Class {0} is: {1**:.4f**}".format(min(cnt),min(probs)))

    print("Probability of Class {0} is: {1**:.4f**}".format(max(cnt),max(probs)))

    return entropy(probs)

**def**information\_gain(df, split\_attribute, target\_attribute,battr):

    print("\n\n-----Information Gain Calculation of ",split\_attribute, " --------")

    df\_split=df.groupby(split\_attribute)

    glist=[]

    forgname,groupindf\_split:

        print('Grouped Attribute Values \n',group)

        glist.append(gname)

    glist.reverse()

    nobs =len(df.index) \*1.0

    df\_agg1=df\_split.agg({target\_attribute:**lambda** x:entropy\_of\_list(x, glist.pop())})

    df\_agg2=df\_split.agg({target\_attribute :**lambda** x:len(x)/nobs})

    df\_agg1.columns=['Entropy']

    df\_agg2.columns=['Proportion']

    new\_entropy=sum( df\_agg1['Entropy'] \* df\_agg2['Proportion'])

    ifbattr!='S':

        old\_entropy= entropy\_of\_list(df[target\_attribute],'S-'+df.iloc[0][df.columns.get\_loc(battr)])

    else:

        old\_entropy=entropy\_of\_list(df[target\_attribute],battr)

    returnold\_entropy-new\_entropy

**def**id3(df,target\_attribute,attribute\_names,default\_class=None,default\_attr='S'):

    from collections import Counter

    cnt=Counter(x for x in df[target\_attribute])

    iflen(cnt)==1:

        returnnext(iter(cnt))

    elifdf.emptyor (notattribute\_names):

        returndefault\_class

    else:

        default\_class=max(cnt.keys())

        gainz=[]

        forattrinattribute\_names:

            ig=information\_gain(df,attr,target\_attribute,default\_attr)

            gainz.append(ig)

            print('Information gain of',attr,'is:',ig)

        index\_of\_max=gainz.index(max(gainz))

        best\_attr=attribute\_names[index\_of\_max]

        print("\nAttribute with the maximum gain is:",best\_attr)

        tree={best\_attr:{}}

        remaining\_attribute\_names=[iforiinattribute\_namesifi!=best\_attr]

        forattr\_val,data\_subsetindf.groupby (best\_attr):

            subtree=id3(data\_subset,target\_attribute,remaining\_attribute\_names,default\_class,best\_attr)

            tree[best\_attr][attr\_val]=subtree

        return tree

frompprintimportpprint

tree = id3(df,t,attribute\_names)

print("\nthe Resultant Decision Tree is:")

pprint(tree)

-----Information Gain Calculation of outlook --------

Grouped Attribute Values

outlook tempearture humidity wind playtennis

2 overcast hot high weak yes

6 overcast cool normal strong yes

11 overcast mild high strong yes

12 overcast hot normal weak yes

Grouped Attribute Values

outlook tempearture humidity wind playtennis

3 rain mild high weak yes

4 rain cool normal weak yes

5 rain cool normal strong no

9 rain mild normal weak yes

13 rain mild high strong no

Grouped Attribute Values

outlook tempearture humidity wind playtennis

0 sunny hot high weak no

1 sunny hot high strong no

7 sunny mild high weak no

8 sunny cool normal weak yes

10 sunny mild normal strong yes

Target attribute class count(Yes/No)= {'yes': 4}

Total no of instances/records associated with overcast is: 4

Probability of Class yes is: 1.0000

Probability of Class yes is: 1.0000

Target attribute class count(Yes/No)= {'yes': 3, 'no': 2}

Total no of instances/records associated with rain is: 5

Probability of Class no is: 0.4000

Probability of Class yes is: 0.6000

Target attribute class count(Yes/No)= {'no': 3, 'yes': 2}

Total no of instances/records associated with sunny is: 5

Probability of Class no is: 0.4000

Probability of Class yes is: 0.6000

Target attribute class count(Yes/No)= {'no': 5, 'yes': 9}

Total no of instances/records associated with S is: 14

Probability of Class no is: 0.3571

Probability of Class yes is: 0.6429

Information gain of outlook is: 0.2467498197744391

-----Information Gain Calculation of tempearture --------

Grouped Attribute Values

outlook tempearture humidity wind playtennis

4 rain cool normal weak yes

5 rain cool normal strong no

6 overcast cool normal strong yes

8 sunny cool normal weak yes

Grouped Attribute Values

outlook tempearture humidity wind playtennis

0 sunny hot high weak no

1 sunny hot high strong no

2 overcast hot high weak yes

12 overcast hot normal weak yes

Grouped Attribute Values

outlook tempearture humidity wind playtennis

3 rain mild high weak yes

7 sunny mild high weak no

9 rain mild normal weak yes

10 sunny mild normal strong yes

11 overcast mild high strong yes

13 rain mild high strong no

Target attribute class count(Yes/No)= {'yes': 3, 'no': 1}

Total no of instances/records associated with cool is: 4

Probability of Class no is: 0.2500

Probability of Class yes is: 0.7500

Target attribute class count(Yes/No)= {'no': 2, 'yes': 2}

Total no of instances/records associated with hot is: 4

Probability of Class no is: 0.5000

Probability of Class yes is: 0.5000

Target attribute class count(Yes/No)= {'yes': 4, 'no': 2}

Total no of instances/records associated with mild is: 6

Probability of Class no is: 0.3333

Probability of Class yes is: 0.6667

Target attribute class count(Yes/No)= {'no': 5, 'yes': 9}

Total no of instances/records associated with S is: 14

Probability of Class no is: 0.3571

Probability of Class yes is: 0.6429

Information gain of tempearture is: 0.029222565658954647

-----Information Gain Calculation of humidity --------

Grouped Attribute Values

outlook tempearture humidity wind playtennis

0 sunny hot high weak no

1 sunny hot high strong no

2 overcast hot high weak yes

3 rain mild high weak yes

7 sunny mild high weak no

11 overcast mild high strong yes

13 rain mild high strong no

Grouped Attribute Values

outlook tempearture humidity wind playtennis

4 rain cool normal weak yes

5 rain cool normal strong no

6 overcast cool normal strong yes

8 sunny cool normal weak yes

9 rain mild normal weak yes

10 sunny mild normal strong yes

12 overcast hot normal weak yes

Target attribute class count(Yes/No)= {'no': 4, 'yes': 3}

Total no of instances/records associated with high is: 7

Probability of Class no is: 0.4286

Probability of Class yes is: 0.5714

Target attribute class count(Yes/No)= {'yes': 6, 'no': 1}

Total no of instances/records associated with normal is: 7

Probability of Class no is: 0.1429

Probability of Class yes is: 0.8571

Target attribute class count(Yes/No)= {'no': 5, 'yes': 9}

Total no of instances/records associated with S is: 14

Probability of Class no is: 0.3571

Probability of Class yes is: 0.6429

Information gain of humidity is: 0.15183550136234136

-----Information Gain Calculation of wind --------

Grouped Attribute Values

outlook tempearture humidity wind playtennis

1 sunny hot high strong no

5 rain cool normal strong no

6 overcast cool normal strong yes

10 sunny mild normal strong yes

11 overcast mild high strong yes

13 rain mild high strong no

Grouped Attribute Values

outlook tempearturehumidity windplaytennis

0 sunny hot high weak no

2 overcast hot high weak yes

3 rain mild high weak yes

4 rain cool normal weak yes

7 sunny mild high weak no

8 sunny cool normal weak yes

9 rain mild normal weak yes

12 overcast hot normal weak yes

Target attribute class count(Yes/No)= {'no': 3, 'yes': 3}

Total no of instances/records associated with strong is: 6

Probability of Class no is: 0.5000

Probability of Class yes is: 0.5000

Target attribute class count(Yes/No)= {'no': 2, 'yes': 6}

Total no of instances/records associated with weak is: 8

Probability of Class no is: 0.2500

Probability of Class yes is: 0.7500

Target attribute class count(Yes/No)= {'no': 5, 'yes': 9}

Total no of instances/records associated with S is: 14

Probability of Class no is: 0.3571

Probability of Class yes is: 0.6429

Information gain of wind is: 0.04812703040826927

Attribute with the maximum gain is: outlook

-----Information Gain Calculation of tempearture --------

Grouped Attribute Values

outlook tempearture humidity wind playtennis

4 rain cool normal weak yes

5 rain cool normal strong no

Grouped Attribute Values

outlook tempearture humidity wind playtennis

3 rain mild high weak yes

9 rain mild normal weak yes

13 rain mild high strong no

Target attribute class count(Yes/No)= {'yes': 1, 'no': 1}

Total no of instances/records associated with cool is: 2

Probability of Class no is: 0.5000

Probability of Class yes is: 0.5000

Target attribute class count(Yes/No)= {'yes': 2, 'no': 1}

Total no of instances/records associated with mild is: 3

Probability of Class no is: 0.3333

Probability of Class yes is: 0.6667

Target attribute class count(Yes/No)= {'yes': 3, 'no': 2}

Total no of instances/records associated with S-rain is: 5

Probability of Class no is: 0.4000

Probability of Class yes is: 0.6000

Information gain of tempearture is: 0.01997309402197489

-----Information Gain Calculation of humidity --------

Grouped Attribute Values

outlook tempearture humidity wind playtennis

3 rain mild high weak yes

13 rain mild high strong no

Grouped Attribute Values

outlook tempearture humidity wind playtennis

4 rain cool normal weak yes

5 rain cool normal strong no

9 rain mild normal weak yes

Target attribute class count(Yes/No)= {'yes': 1, 'no': 1}

Total no of instances/records associated with high is: 2

Probability of Class no is: 0.5000

Probability of Class yes is: 0.5000

Target attribute class count(Yes/No)= {'yes': 2, 'no': 1}

Total no of instances/records associated with normal is: 3

Probability of Class no is: 0.3333

Probability of Class yes is: 0.6667

Target attribute class count(Yes/No)= {'yes': 3, 'no': 2}

Total no of instances/records associated with S-rain is: 5

Probability of Class no is: 0.4000

Probability of Class yes is: 0.6000

Information gain of humidity is: 0.01997309402197489

-----Information Gain Calculation of wind --------

Grouped Attribute Values

outlook tempearture humidity wind playtennis

5 rain cool normal strong no

13 rain mild high strong no

Grouped Attribute Values

outlook tempearturehumidity windplaytennis

3 rain mild high weak yes

4 rain cool normal weak yes

9 rain mild normal weak yes

Target attribute class count(Yes/No)= {'no': 2}

Total no of instances/records associated with strong is: 2

Probability of Class no is: 1.0000

Probability of Class no is: 1.0000

Target attribute class count(Yes/No)= {'yes': 3}

Total no of instances/records associated with weak is: 3

Probability of Class yes is: 1.0000

Probability of Class yes is: 1.0000

Target attribute class count(Yes/No)= {'yes': 3, 'no': 2}

Total no of instances/records associated with S-rain is: 5

Probability of Class no is: 0.4000

Probability of Class yes is: 0.6000

Information gain of wind is: 0.9709505944546686

Attribute with the maximum gain is: wind

-----Information Gain Calculation of tempearture --------

Grouped Attribute Values

outlook tempearturehumidity windplaytennis

8 sunny cool normal weak yes

Grouped Attribute Values

outlook tempearture humidity wind playtennis

0 sunny hot high weak no

1 sunny hot high strong no

Grouped Attribute Values

outlook tempearture humidity wind playtennis

7 sunny mild high weak no

10 sunny mild normal strong yes

Target attribute class count(Yes/No)= {'yes': 1}

Total no of instances/records associated with cool is: 1

Probability of Class yes is: 1.0000

Probability of Class yes is: 1.0000

Target attribute class count(Yes/No)= {'no': 2}

Total no of instances/records associated with hot is: 2

Probability of Class no is: 1.0000

Probability of Class no is: 1.0000

Target attribute class count(Yes/No)= {'no': 1, 'yes': 1}

Total no of instances/records associated with mild is: 2

Probability of Class no is: 0.5000

Probability of Class yes is: 0.5000

Target attribute class count(Yes/No)= {'no': 3, 'yes': 2}

Total no of instances/records associated with S-sunny is: 5

Probability of Class no is: 0.4000

Probability of Class yes is: 0.6000

Information gain of tempearture is: 0.5709505944546686

-----Information Gain Calculation of humidity --------

Grouped Attribute Values

outlook tempearture humidity wind playtennis

0 sunny hot high weak no

1 sunny hot high strong no

7 sunny mild high weak no

Grouped Attribute Values

outlook tempearture humidity wind playtennis

8 sunny cool normal weak yes

10 sunny mild normal strong yes

Target attribute class count(Yes/No)= {'no': 3}

Total no of instances/records associated with high is: 3

Probability of Class no is: 1.0000

Probability of Class no is: 1.0000

Target attribute class count(Yes/No)= {'yes': 2}

Total no of instances/records associated with normal is: 2

Probability of Class yes is: 1.0000

Probability of Class yes is: 1.0000

Target attribute class count(Yes/No)= {'no': 3, 'yes': 2}

Total no of instances/records associated with S-sunny is: 5

Probability of Class no is: 0.4000

Probability of Class yes is: 0.6000

Information gain of humidity is: 0.9709505944546686

-----Information Gain Calculation of wind --------

Grouped Attribute Values

outlook tempearture humidity wind playtennis

1 sunny hot high strong no

10 sunny mild normal strong yes

Grouped Attribute Values

outlook tempearturehumidity windplaytennis

0 sunny hot high weak no

7 sunny mild high weak no

8 sunny cool normal weak yes

Target attribute class count(Yes/No)= {'no': 1, 'yes': 1}

Total no of instances/records associated with strong is: 2

Probability of Class no is: 0.5000

Probability of Class yes is: 0.5000

Target attribute class count(Yes/No)= {'no': 2, 'yes': 1}

Total no of instances/records associated with weak is: 3

Probability of Class no is: 0.3333

Probability of Class yes is: 0.6667

Target attribute class count(Yes/No)= {'no': 3, 'yes': 2}

Total no of instances/records associated with S-sunny is: 5

Probability of Class no is: 0.4000

Probability of Class yes is: 0.6000

Information gain of wind is: 0.01997309402197489

Attribute with the maximum gain is: humidity

the Resultant Decision Tree is:

{'outlook': {'overcast': 'yes',

'rain': {'wind': {'strong': 'no', 'weak': 'yes'}},

'sunny': {'humidity': {'high': 'no', 'normal': 'yes'}}}}

**def**classify(instance,tree,default=None):

    attribute=next(iter(tree))

    if instance[attribute] in tree[attribute].keys():

        result = tree[attribute][instance[attribute]]

        ifisinstance (result,dict):

            return classify(instance,result)

        else:

            return result

    else:

        return default

df\_new=pd.read\_csv('PlayTennisTest.csv')

df\_new['predicted'] =df\_new.apply(classify,axis=1,args=(tree,'?'))

print(df\_new)

outlook temperature humidity wind playtennis predicted

0 sunny hot normal weak ? yes

1 rain mild high strong ? no

OR

import math

importcsv

defload\_csv(filename):

lines=csv.reader(open(filename,"r"))

dataset=list(lines)

headers=dataset.pop(0)

return dataset, headers

class Node:

def \_init\_(self,attribute):

self.attribute=attribute

self.children=[]

self.answer=""

defsubtables(data,col,delete):

dic={}

coldata=[row[col]for row in data]

attr=list(set(coldata))

counts=[0]\*len(attr)

r=len(data)

c=len(data[0])

for x in range(len(attr)):

for y in range(r):

if data[y][col]==attr[x]:

counts[x]+=1

for x in range(len(attr)):

dic[attr[x]]=[[0 for i in range(c)] for j in range(counts[x])]

pos=0

for y in range(r):

if data[y][col]==attr[x]:

if delete:

del data[y][col]

dic[attr[x]][pos]=data[y]

pos +=1

returnattr, dic

def entropy(S):

attr=list(set(S))

iflen(attr)==1:

return 0

counts=[0,0]

for i in range(2):

counts[i]=sum([1 for x in S if attr[i]==x])/(len(S)\*1.0)

sums=0

forcnt in counts:

sums+=-1\*cnt\*math.log(cnt,2)

return sums

defcompute\_gain(data,col):

attr,dic = subtables(data,col,delete=False)

total\_size=len(data)

entropies=[0]\*len(attr)

ratio=[0]\*len(attr)

total\_entropy=entropy([row[-1] for row in data])

for x in range(len(attr)):

ratio[x]=len(dic[attr[x]]) /(total\_size\*1.0)

entropies[x]=entropy([row[-1] for row in dic[attr[x]]])

for x in range(len(entropies)):

total\_entropy-=ratio[x]\*entropies[x]

returntotal\_entropy

defbuil\_tree(data,features):

lastcol=[row[-1] for row in data]

if(len(set(lastcol)))==1:

node=Node("")

node.answer=lastcol[0]

return node

n=len(data[0])-1

gains=[0]\*n

for col in range(n):

gains[col]=compute\_gain(data,col)

split = gains.index(max(gains))

node=Node(features[split])

del(features[split])

attr,dic=subtables(data,split,delete=True)

for x in range(len(attr)):

child=build\_tree(dic[attr[x]],features)

node.children.append((attr[x],child))

return node

defprint\_tree(node,level):

if node.answer != "":

print(" "\*level,node.answer)

return

print(" "\*level,node.attribute)

for value, n in node.children:

print(" "\*(level+1),value)

print\_tree(n,level+2)

def classify(node,x\_test,features):

if node.answer !="":

print(node.answer)

return

pos=features.index(node.attribute)

for value, n in node.children:

ifx\_test[pos]==value:

classify(n,x\_test,features)

dataset,features=load\_csv('data3.csv')

node=build\_tree(dataset,features)

print("The decision tree for the dataset using ID3 algorithm is")

print\_tree(node,0)

testdata,features=load\_csv("data3\_test.csv")

forxtest in testdata:

print("The test instance ",xtest)

print("The label for test instance: ",end="")

classify(node,xtest,features)

Training instances: data3.csv

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Outlook | Temperature | Humidity | Wind | Answer |
| Sunny | hot | high | weak | no |
| Sunny | hot | high | strong | no |
| overcast | hot | high | weak | yes |
| Rain | mild | high | weak | yes |
| Rain | cool | normal | weak | yes |
| Rain | cool | normal | strong | no |
| overcast | cool | normal | strong | yes |
| Sunny | mild | high | weak | no |
| Sunny | cool | normal | weak | yes |
| Rain | mild | normal | weak | yes |
| Sunny | mild | normal | strong | yes |
| overcast | mild | high | strong | yes |
| overcast | hot | normal | weak | yes |
| Rain | mild | high | strong | no |

Testing instances: data3\_test.csv

|  |  |  |  |
| --- | --- | --- | --- |
| Outlook | Temperature | Humidity | Wind |
| Rain | cool | normal | strong |
| Sunny | mild | normal | strong |

OUTPUT

The decision tree for the dataset using ID3 algorithm is

Outlook

overcast

yes

rain

Wind

weak

yes

strong

no

sunny

Humidity

normal

yes

high

no

The test instance : ['rain', 'cool', 'normal', 'strong']

The predicted label : no

The test instance : ['sunny', 'mild', 'normal', 'strong']

The predicted label : yes

5. Build an Artificial Neural Network by implementing the Back propagation algorithm and test the same

using appropriate data sets.

importnumpy as np

X=np.array(([2,9],[1, 5], [3, 6]), dtype=float)#Features(Hrs Slept, Hrs Studied)

y= np.array(([92], [86],[89]), dtype=float)#Labels(Marks obtained)

X =X/np.amax(X,axis=0) #Normalize

y=y/100

def sigmoid (x):

return 1/(1 + np.exp(-x))

defsigmoid\_grad(x):

return x\* (1-x)

#variable initialization

epoch=1000 #Setting training iterations

eta=0.2 #Setting leaning rate(eta)

input\_neurons = 2 #number of features in data set

hidden\_neurons =3 #number of hidden layers neurons

output\_neurons =1 #number of neurons at output layer

#Weight and bias - Random initialization

wh=np.random.uniform(size=(input\_neurons,hidden\_neurons))

bh=np.random.uniform(size=(1,hidden\_neurons))

wout=np.random.uniform(size=(hidden\_neurons,output\_neurons))

bout=np.random.uniform(size=(1,output\_neurons))

for i in range(epoch):

#forward propogation

h\_ip=np.dot(X,wh)+bh

h\_act=sigmoid(h\_ip)

o\_ip=np.dot(h\_act,wout)

output =sigmoid(o\_ip)

#Back propogation

#Error at Outuput layer

Eo = y-output

outgrad=sigmoid\_grad(output)

d\_output=Eo\*outgrad

#Error at Hidden layer

Eh=d\_output.dot(wout.T)

hiddengrad=sigmoid\_grad(h\_act)

d\_hidden=Eh\*hiddengrad

wout += h\_act.T.dot(d\_output)\*eta

wh += X.T.dot(d\_hidden)\*eta

print("Normalized Input: \n"+ str(X))

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

print("Predicted Output : \n", output)

OUTPUT:

Normalized Input:

[[0.66666667 1. ]

[0.33333333 0.55555556]

[1. 0.66666667]]

Actual Output:

[[0.92]

[0.86]

[0.89]]

Predicted Output :

[[0.89648872]

[0.87592696]

[0.89713304]]

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

importcsv, random, math

import statistics as st

defloadCsv(filename):

lines = csv.reader(open(filename, "r"));

dataset = list(lines)

for i in range(len(dataset)):

dataset[i] = [float(x) for x in dataset[i]]

return dataset

defsplitDataset(dataset, splitRatio):

testSize = int(len(dataset) \* splitRatio);

trainSet = list(dataset);

testSet = []

whilelen(testSet) <testSize:

#randomly pick an instance from training data

index = random.randrange(len(trainSet));

testSet.append(trainSet.pop(index))

return [trainSet, testSet]

#Create a dictionary of classes 1 and 0 where the values are the

#instances belonging to each class

defseparateByClass(dataset):

separated = {}

for i in range(len(dataset)):

x = dataset[i]

if (x[-1] not in separated):

separated[x[-1]] = []

separated[x[-1]].append(x)

return separated

defcompute\_mean\_std(dataset):

mean\_std = [ (st.mean(attribute), st.stdev(attribute))

for attribute in zip(\*dataset)]; #zip(\*res) transposes a matrix (2-d array/list)

delmean\_std[-1] # Exclude label

returnmean\_std

defsummarizeByClass(dataset):

separated = separateByClass(dataset);

summary = {} # to store mean and std of +ve and -ve instances

forclassValue, instances in separated.items():

#summaries is a dictionary of tuples(mean,std) for each class value

summary[classValue] = compute\_mean\_std(instances)

return summary

#For continuous attributes p is estimated using Gaussion distribution

defestimateProbability(x, mean, stdev):

exponent = math.exp(-(math.pow(x-mean,2)/(2\*math.pow(stdev,2))))

return (1 / (math.sqrt(2\*math.pi) \* stdev)) \* exponent

defcalculateClassProbabilities(summaries, testVector):

p = {}

#class and attribute information as mean and sd

forclassValue, classSummaries in summaries.items():

p[classValue] = 1

for i in range(len(classSummaries)):

mean, stdev = classSummaries[i]

x = testVector[i] #testvector's first attribute

#use normal distribution

p[classValue] \*= estimateProbability(x, mean, stdev);

return p

def predict(summaries, testVector):

all\_p = calculateClassProbabilities(summaries, testVector)

bestLabel, bestProb = None, -1

forlbl, p in all\_p.items():#assigns that class which has he highest prob

ifbestLabel is None or p >bestProb:

bestProb = p

bestLabel = lbl

returnbestLabel

defperform\_classification(summaries, testSet):

predictions = []

for i in range(len(testSet)):

result = predict(summaries, testSet[i])

predictions.append(result)

return predictions

defgetAccuracy(testSet, predictions):

correct = 0

for i in range(len(testSet)):

iftestSet[i][-1] == predictions[i]:

correct += 1

return (correct/float(len(testSet))) \* 100.0

dataset = loadCsv('data51.csv');

print('Pima Indian Diabetes Dataset loaded...')

print('Total instances available :',len(dataset))

print('Total attributes present :',len(dataset[0])-1)

print("First Five instances of dataset:")

for i in range(5):

print(i+1 , ':' , dataset[i])

splitRatio = 0.2

trainingSet, testSet = splitDataset(dataset, splitRatio)

print('\nDataset is split into training and testing set.')

print('Training examples = {0} \nTesting examples = {1}'.format(len(trainingSet),

len(testSet)))

summaries = summarizeByClass(trainingSet);

predictions = perform\_classification(summaries, testSet)

accuracy = getAccuracy(testSet, predictions)

print('\nAccuracy of the Naive Baysian Classifier is :', accuracy)

OUTPUT

Pima Indian Diabetes Dataset loaded...

Total instances available : 768

Total attributes present : 8

First Five instances of dataset:

1 : [6.0, 148.0, 72.0, 35.0, 0.0, 33.6, 0.627, 50.0, 1.0]

2 : [1.0, 85.0, 66.0, 29.0, 0.0, 26.6, 0.351, 31.0, 0.0]

3 : [8.0, 183.0, 64.0, 0.0, 0.0, 23.3, 0.672, 32.0, 1.0]

4 : [1.0, 89.0, 66.0, 23.0, 94.0, 28.1, 0.167, 21.0, 0.0]

5 : [0.0, 137.0, 40.0, 35.0, 168.0, 43.1, 2.288, 33.0, 1.0]

Dataset is split into training and testing set.

Training examples = 615

Testing examples = 153

Accuracy of the Naive Baysian Classifier is : 73.85

Or

import pandas as pd

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

from sklearn.naive\_bayes import GaussianNB

from sklearn import metrics

df = pd.read\_csv('pima\_indian.csv')

feature\_col\_names = ['num\_preg','glucose\_conc','diastolic\_bp','thickness','insulin','bmi','diab\_pred','age']

predicted\_class\_names = ['diabetes']

x = df[feature\_col\_names].values

y = df[predicted\_class\_names].values

xtrain,xtest,ytrain,ytest = train\_test\_split(x,y,test\_size=0.33)

print('\n Total number of Training\_data :', ytrain.shape)

print('\n Total number of Test\_data :', ytest.shape)

clf = GaussianNB().fit(xtrain,ytrain.ravel())

predicted = clf.predict(xtest)

PredictTestData = clf.predict([[6,148,72,35,0,33.6,0.627,50]])

print("\n Confusion matrix")

print(metrics.confusion\_matrix(ytest,predicted))

print("\n Accuracy of the classifier is:",metrics.accuracy\_score(ytest,predicted))

print("\n The value of Precision:",metrics.precision\_score(ytest,predicted))

print("\n The value of Recall:",metrics.recall\_score(ytest,predicted))

print("Predicted value for individual Test Data:",PredictTestData)

OUTPUT:

Total number of Training\_data : (514, 1)

Total number of Test\_data : (254, 1)

Confusion matrix

[[137 25]

[ 36 56]]

Accuracy of the classifier is: 0.7598425196850394

The value of Precision: 0.691358024691358

The value of Recall: 0.6086956521739131

Predicted value for individual Test Data: [1]

7. Apply **EM algorithm** to cluster a set of data stored in a .CSV file. Use the same data set for clustering using **k-Means algorithm**. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program.

importmatplotlib.pyplot as plt

fromsklearn import datasets

fromsklearn.cluster import KMeans

importsklearn.metrics as sm

import pandas as pd

importnumpy as np

%matplotlib inline

# import some data to play with

iris = datasets.load\_iris()

# Store the inputs as a Pandas Dataframe and set the column names

X = pd.DataFrame(iris.data)

X.columns = ['Sepal\_Length','Sepal\_Width','Petal\_Length','Petal\_Width']

y = pd.DataFrame(iris.target)

y.columns = ['Targets']

# K Means Cluster

model = KMeans(n\_clusters=3)

model.fit(X) # This is what KMeans thought

#model.labels\_

# Set the size of the plot

plt.figure(figsize=(14,14))

# Create a colormap

colormap = np.array(['red', 'lime', 'black'])

# Plot the Original Classifications using Petal features

plt.subplot(2, 2, 1)

plt.scatter(X.Petal\_Length, X.Petal\_Width, c=colormap[y.Targets], s=40)

plt.title('Real Classification')

plt.xlabel("Petal Length")

plt.ylabel('Petal Width')

# Plot the Models Classifications

plt.subplot(2, 2, 2)

plt.scatter(X.Petal\_Length, X.Petal\_Width, c=colormap[model.labels\_], s=40)

plt.title('K Mean Classification')

plt.xlabel("Petal Length")

plt.ylabel('Petal Width')

sm.accuracy\_score(y, model.labels\_)

sm.confusion\_matrix(y,model.labels\_)

fromsklearn import preprocessing

scaler = preprocessing.StandardScaler()

scaler.fit(X)

xsa = scaler.transform(X)

xs = pd.DataFrame(xsa, columns = X.columns)

#xs.sample(5)

fromsklearn.mixture import GaussianMixture

gmm = GaussianMixture(n\_components=3)

gmm.fit(xs)

y\_cluster\_gmm = gmm.predict(xs)

plt.subplot(2, 2, 3)

plt.scatter(X.Petal\_Length, X.Petal\_Width, c=colormap[y\_cluster\_gmm], s=40)

plt.title('GMM Classification')

plt.xlabel("Petal Length")

plt.ylabel('Petal Width')

print("Observation: Comparision of GMM and K-Means Observed ")

OUTPUT

Observation: Comparison of GMM and K-Means Observed





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

fromsklearn.datasets import load\_iris

fromsklearn.neighbors import KNeighborsClassifier

importnumpy as np

fromsklearn.model\_selection import train\_test\_split

iris\_dataset=load\_iris()

print("\n IRIS FEATURES \ TARGET NAMES: \n ", iris\_dataset.target\_names)

for i in range(len(iris\_dataset.target\_names)):

print("\n[{0}]:[{1}]".format(i,iris\_dataset.target\_names[i]))

print("\n IRIS DATA :\n",iris\_dataset["data"])

X\_train, X\_test, y\_train, y\_test = train\_test\_split(iris\_dataset["data"], iris\_dataset["target"], random\_state=0)

print("\n Target :\n",iris\_dataset["target"])

print("\n X TRAINING DATA SET \n", X\_train)

print("\n Y TRAINING DATA SET \n", y\_train)

print("\n X TESTING DATA SET \n", X\_test)

print("\n Y TESTING DATA SET \n", y\_test)

kn = KNeighborsClassifier(n\_neighbors=1)

kn.fit(X\_train, y\_train)

print("Prediction Test / Validation \n ")

for i in range(len(X\_test)):

x = X\_test[i]

x\_new = np.array([x])

prediction = kn.predict(x\_new)

print("\n Actual : {0} {1}, Predicted :{2}{3}".format(y\_test[i],

iris\_dataset["target\_names"][y\_test[i]],prediction,iris\_dataset["target\_names"][prediction]))

print("\n TEST SCORE[ACCURACY]: {:.2f}\n".format(kn.score(X\_test, y\_test)))

OUTPUT

IRIS FEATURES \ TARGET NAMES:

['setosa' 'versicolor' 'virginica']

[0]:[setosa]

[1]:[versicolor]

[2]:[virginica]

IRIS DATA :

[[5.1 3.5 1.4 0.2]

[4.9 3. 1.4 0.2]

[4.7 3.2 1.3 0.2]

[4.6 3.1 1.5 0.2]

[5. 3.6 1.4 0.2]

[5.4 3.9 1.7 0.4]

[4.6 3.4 1.4 0.3]

[5. 3.4 1.5 0.2]

[4.4 2.9 1.4 0.2]

[4.9 3.1 1.5 0.1]

[5.4 3.7 1.5 0.2]…..CONTINUES

Target :

[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2

2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2

2 2]

X TRAINING DATA SET

[[5.9 3. 4.2 1.5]

[5.8 2.6 4. 1.2]

[6.8 3. 5.5 2.1]

[4.7 3.2 1.3 0.2]

[6.9 3.1 5.1 2.3]

[5. 3.5 1.6 0.6]…..CONTINUES

Y TRAINING DATA SET

[1 1 2 0 2 0 0 1 2 2 2 2 1 2 1 1 2 2 2 2 1 2 1 0 2 1 1 1 1 2 0 0 2 1 0 0 1

0 2 1 0 1 2 1 0 2 2 2 2 0 0 2 2 0 2 0 2 2 0 0 2 0 0 0 1 2 2 0 0 0 1 1 0 0

1 0 2 1 2 1 0 2 0 2 0 0 2 0 2 1 1 1 2 2 1 1 0 1 2 2 0 1 1 1 1 0 0 0 2 1 2

0]

X TESTING DATA SET

[[5.8 2.8 5.1 2.4]

[6. 2.2 4. 1. ]

[5.5 4.2 1.4 0.2]

[7.3 2.9 6.3 1.8]…..CONTINUES

Y TESTING DATA SET

[2 1 0 2 0 2 0 1 1 1 2 1 1 1 1 0 1 1 0 0 2 1 0 0 2 0 0 1 1 0 2 1 0 2 2 1 0

1]

Prediction Test / Validation

Actual : 2 virginica, Predicted :[2]['virginica']

Actual : 1 versicolor, Predicted :[1]['versicolor']

Actual : 0 setosa, Predicted :[0]['setosa']

Actual : 2 virginica, Predicted :[2]['virginica']

Actual : 0 setosa, Predicted :[0]['setosa']

Actual : 2 virginica, Predicted :[2]['virginica']

Actual : 0 setosa, Predicted :[0]['setosa']

Actual : 1 versicolor, Predicted :[1]['versicolor']

Actual : 1 versicolor, Predicted :[1]['versicolor']

Actual : 1 versicolor, Predicted :[1]['versicolor']

Actual : 2 virginica, Predicted :[2]['virginica']

Actual : 1 versicolor, Predicted :[1]['versicolor']

Actual : 1 versicolor, Predicted :[1]['versicolor']

Actual : 1 versicolor, Predicted :[1]['versicolor']

Actual : 1 versicolor, Predicted :[1]['versicolor']

Actual : 0 setosa, Predicted :[0]['setosa']

Actual : 1 versicolor, Predicted :[1]['versicolor']

Actual : 1 versicolor, Predicted :[1]['versicolor']

Actual : 0 setosa, Predicted :[0]['setosa']

Actual : 0 setosa, Predicted :[0]['setosa']

Actual : 2 virginica, Predicted :[2]['virginica']

Actual : 1 versicolor, Predicted :[1]['versicolor']

Actual : 0 setosa, Predicted :[0]['setosa']

Actual : 0 setosa, Predicted :[0]['setosa']

Actual : 2 virginica, Predicted :[2]['virginica']…CONTINUES

TEST SCORE[ACCURACY]: 0.97

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

THEORY: Linear regression is a supervised learning algorithm used for computing linear relationships between input (X) and output (Y).

Locally weighted linear regression is a non-parametric algorithm, that is, the model does not learn a fixed set of parameters as is done in ordinary linear regression. Rather parameters \theta are computed individually for each query point x. While computing \theta, a higher “preference” is given to the points in the training set lying in the vicinity of x than the points lying far away from x.

* This algorithm is used for making predictions when there exists a non-linear relationship between the features.
* Locally weighted linear regression is a supervised learning algorithm.
* It a non-parametric algorithm.
* doneThere exists No training phase. All the work is done during the testing phase/while making predictions.

The modified cost function is: J(\theta) = $\sum\_{i=1}^{m} w^{(i)}(\theta^Tx^{(i)} - y^{(i)})^2

where, w^{(i)} is a non-negative “weight” associated with training point x^{(i)}.

For x^{(i)}s lying closer to the query point x, the value of w^{(i)} is large, while for x^{(i)}s lying far away from x the value of w^{(i)} is small.

A typical choice of w^{(i)} is: w^{(i)} = exp(\frac{-(x^{(i)} - x)^2}{2\tau^2})

where, \tau is called the bandwidth parameter and controls the rate at which w^{(i)} falls with distance from x

Clearly, if |x^{(i)} - x| is small w^{(i)} is close to 1 and if |x^{(i)} - x| is large w^{(i)} is close to 0.

Thus, the training-set-points lying closer to the query point x contribute more to the cost J(\theta) than the points lying far away from x.

**PROCEDURE/PROGRAM:**

import pandas as pd

import numpyas np

import matplotlib.pyplotasplt

**def** local\_regression(x0,X,y,tau):

    x0=np.r\_[1,x0]

    X=np.c\_[np.ones(len(X)),X]

    xw=X.T\*radial\_kernel(x0,X,tau)

    beta=np.linalg.pinv(xw@ X) @xw@ y

    return x0 @ beta

print(np.r\_[np.array([1,2,3]),0,0,0,np.array([4,5,6])])

print(np.c\_[np.array([1,2,3]),np.array([4,5,6])])

[1 2 3 0 0 0 4 5 6]

[[1 4]

[2 5]

[3 6]]

**def** radial\_kernel(x0,X,tau):

    returnnp.exp(np.sum((X-x0)\*\*2,axis=1)/(-2\*tau\*tau))

data=pd.read\_csv("tips.csv")

print(data)

total\_bill tip sex smoker day time size

0 16.99 1.01 Female No Sun Dinner 2

1 10.34 1.66 Male No Sun Dinner 3

2 21.01 3.50 Male No Sun Dinner 3

3 23.68 3.31 Male No Sun Dinner 2

.. ... ... ... ... ... ... ...

232 11.61 3.39 Male No Sat Dinner 2

233 10.77 1.47 Male No Sat Dinner 2

234 15.53 3.00 Male Yes Sat Dinner 2

235 10.07 1.25 Male No Sat Dinner 2

236 12.60 1.00 Male Yes Sat Dinner 2

237 32.83 1.17 Male Yes Sat Dinner 2

238 35.83 4.67 Female No Sat Dinner 3

239 29.03 5.92 Male No Sat Dinner 3

240 27.18 2.00 Female Yes Sat Dinner 2

241 22.67 2.00 Male Yes Sat Dinner 2

242 17.82 1.75 Male No Sat Dinner 2

243 18.78 3.00 Female No Thur Dinner 2

[244 rows x 7 columns]

bill=data.total\_bill.values

print(bill)

[16.99 10.34 21.01 23.68 24.59 25.29 8.77 26.88 15.04 14.78 10.27 35.26

15.42 18.43 14.83 21.58 10.33 16.29 16.97 20.65 17.92 20.29 15.77 39.42

19.82 17.81 13.37 12.69 21.7 19.65 9.55 18.35 15.06 20.69 17.78 24.06

16.31 16.93 18.69 31.27 16.04 17.46 13.94 9.68 30.4 18.29 22.23 32.4

28.55 18.04 12.54 10.29 34.81 9.94 25.56 19.49 38.01 26.41 11.24 48.27

20.29 13.81 11.02 18.29 17.59 20.08 16.45 3.07 20.23 15.01 12.02 17.07

26.86 25.28 14.73 10.51 17.92 27.2 22.76 17.29 19.44 16.66 10.07 32.68

15.98 34.83 13.03 18.28 24.71 21.16 28.97 22.49 5.75 16.32 22.75 40.17

27.28 12.03 21.01 12.46 11.35 15.38 44.3 22.42 20.92 15.36 20.49 25.21

18.24 14.31 14. 7.25 38.07 23.95 25.71 17.31 29.93 10.65 12.43 24.08

11.69 13.42 14.26 15.95 12.48 29.8 8.52 14.52 11.38 22.82 19.08 20.27

11.17 12.26 18.26 8.51 10.33 14.15 16. 13.16 17.47 34.3 41.19 27.05

16.43 8.35 18.64 11.87 9.78 7.51 14.07 13.13 17.26 24.55 19.77 29.85

48.17 25. 13.39 16.49 21.5 12.66 16.21 13.81 17.51 24.52 20.76 31.71

10.59 10.63 50.81 15.81 7.25 31.85 16.82 32.9 17.89 14.48 9.6 34.63

34.65 23.33 45.35 23.17 40.55 20.69 20.9 30.46 18.15 23.1 15.69 19.81

28.44 15.48 16.58 7.56 10.34 43.11 13. 13.51 18.71 12.74 13. 16.4

20.53 16.47 26.59 38.73 24.27 12.76 30.06 25.89 48.33 13.27 28.17 12.9

28.15 11.59 7.74 30.14 12.16 13.42 8.58 15.98 13.42 16.27 10.09 20.45

13.28 22.12 24.01 15.69 11.61 10.77 15.53 10.07 12.6 32.83 35.83 29.03

27.18 22.67 17.82 18.78]

tip=data.tip.values

print(tip)

[ 1.01 1.66 3.5 3.31 3.61 4.71 2. 3.12 1.96 3.23 1.71 5.

1.57 3. 3.02 3.92 1.67 3.71 3.5 3.35 4.08 2.75 2.23 7.58

3.18 2.34 2. 2. 4.3 3. 1.45 2.5 3. 2.45 3.27 3.6

2. 3.07 2.31 5. 2.24 2.54 3.06 1.32 5.6 3. 5. 6.

2.05 3. 2.5 2.6 5.2 1.56 4.34 3.51 3. 1.5 1.76 6.73

3.21 2. 1.98 3.76 2.64 3.15 2.47 1. 2.01 2.09 1.97 3.

3.14 5. 2.2 1.25 3.08 4. 3. 2.71 3. 3.4 1.83 5.

2.03 5.17 2. 4. 5.85 3. 3. 3.5 1. 4.3 3.25 4.73

4. 1.5 3. 1.5 2.5 3. 2.5 3.48 4.08 1.64 4.06 4.29

3.76 4. 3. 1. 4. 2.55 4. 3.5 5.07 1.5 1.8 2.92

2.31 1.68 2.5 2. 2.52 4.2 1.48 2. 2. 2.18 1.5 2.83

1.5 2. 3.25 1.25 2. 2. 2. 2.75 3.5 6.7 5. 5.

2.3 1.5 1.36 1.63 1.73 2. 2.5 2. 2.74 2. 2. 5.14

5. 3.75 2.61 2. 3.5 2.5 2. 2. 3. 3.48 2.24 4.5

1.61 2. 10. 3.16 5.15 3.18 4. 3.11 2. 2. 4. 3.55

3.68 5.65 3.5 6.5 3. 5. 3.5 2. 3.5 4. 1.5 4.19

2.56 2.02 4. 1.44 2. 5. 2. 2. 4. 2.01 2. 2.5

4. 3.23 3.41 3. 2.03 2.23 2. 5.16 9. 2.5 6.5 1.1

3. 1.5 1.44 3.09 2.2 3.48 1.92 3. 1.58 2.5 2. 3.

2.72 2.88 2. 3. 3.39 1.47 3. 1.25 1. 1.17 4.67 5.92

2. 2. 1.75 3. ]

tau=10

ypred=np.array([local\_regression(x0,bill,tip,tau) for x0 in bill])

print("YPRED",ypred)

**YPRED [2.7268086 1.9373207 3.16357671 3.42997873 3.51633213 3.58128485**

**1.74575924 3.72438753 2.50176793 2.47124009 1.92878673 4.43083574**

**2.54618056 2.88785842 2.4771194 3.2220979 1.93610165 2.64687501**

**2.72453903 3.12615891 2.83136711 3.08839096 2.58686061 4.82568169**

**3.03856248 2.81910225 2.30391956 2.22231784 3.2343044 3.02039613**

**1.8409488 2.87903818 2.50411159 3.1303338 2.81575242 3.46630673**

**2.64917273 2.71999731 2.91641655 4.09613433 2.61808699 2.77989264**

**2.37189436 1.85681398 4.02415732 2.87241286 3.28774109 4.18945095**

**3.86895131 2.84471472 2.20425145 1.93122515 4.39190117 1.88853866**

**3.60600781 3.00322967 4.68304419 3.68269381 2.0468769 6.10051242**

**3.08839096 2.35642868 2.02013067 2.87241286 2.79448878 3.06619949**

**2.66523434 1.06133091 3.08206252 2.49825118 2.14145993 2.73587829**

**3.72262312 3.58036571 2.46535676 1.9580398 2.83136711 3.7525037**

**3.34040171 2.76074818 2.99785156 2.68925208 1.90439686 4.21263647**

**2.61115978 4.39362136 2.26318279 2.8713078 3.52755625 3.17906333**

**3.9045567 3.31367154 1.37936878 2.6503213 3.33941529 4.90638073**

**3.75949977 2.14266965 3.16357671 2.19460712 2.06023987 2.54151737**

**5.42942362 3.30670864 3.15425528 2.53918471 3.10941617 3.57392472**

**2.86688515 2.41578298 2.37902458 1.56067946 4.68889724 3.45583063**

**3.61966545 2.76300381 3.98507723 1.97509508 2.19098893 3.468208**

**2.10149563 2.30989875 2.40986358 2.60769359 2.19701877 3.97423349**

**1.71526491 2.44060371 2.06388307 3.34631445 2.95894005 3.08628255**

**2.03836958 2.17047031 2.86909695 1.71404542 1.93610165 2.39682809**

**2.61346962 2.27877438 2.78101677 4.34832366 5.02235154 3.73935081**

**2.66294224 1.69453712 2.91093747 2.12330458 1.86901685 1.59226714**

**2.38733683 2.27517802 2.75736308 3.51258242 3.03322728 3.97840611**

**6.08109332 3.55452765 2.3063116 2.6698161 3.2139382 2.21870635**

**2.63767619 2.35642868 2.78551106 3.5097674 3.13762944 4.13245304**

**1.96778656 1.97265906 6.64501673 2.59149542 1.56067946 4.14400888**

**2.70749001 4.2308908 2.82802494 2.43588109 1.84705095 4.37645923**

**4.37817153 3.39617323 5.5884145 3.38060805 4.94870736 3.1303338**

**3.15218085 4.02913406 2.85692022 3.37377624 2.57758204 3.03749596**

**3.85958186 2.55317002 2.68011314 1.59834591 1.9373207 5.26315703**

**2.25958205 2.32065357 2.91860646 2.22833498 2.25958205 2.65950258**

**3.11360832 2.66752563 3.69871905 4.75450408 3.48621694 2.23074113**

**3.99590512 3.63598288 6.11223285 2.29195215 3.83650081 2.24757246**

**3.83478624 2.08936953 1.62023977 4.00256103 2.15838857 2.30989875**

**1.72258234 2.61115978 2.30989875 2.6445765 1.90683627 3.10521971**

**2.29314942 3.27671427 3.46154888 2.57758204 2.0917953 1.98970783**

**2.55898966 1.90439686 2.21148067 4.22507855 4.4809218 3.90962252**

**3.75075265 3.331514 2.8202184 2.92626333]**

SortIndex=bill.argsort(0)

xsort=bill[SortIndex]

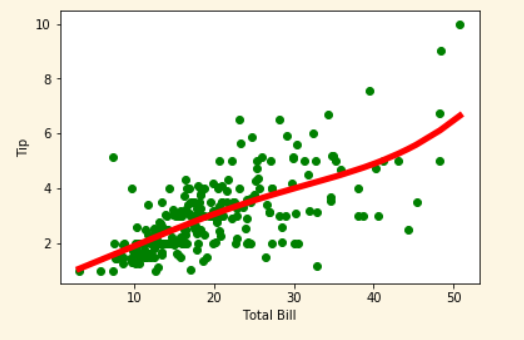
plt.scatter(bill,tip,color='green')

plt.plot(xsort,ypred[SortIndex],color='red',linewidth=5)

plt.xlabel("Total Bill")

plt.ylabel("Tip")

plt.show()



or

importnumpy as np

frombokeh.plotting import figure, show, output\_notebook

frombokeh.layouts import gridplot

from bokeh.io import push\_notebook

output\_notebook()

deflocal\_regression(x0, X, Y, tau):

# add bias term

x0 = np.r\_[1, x0] # Add one to avoid the loss in information

X = np.c\_[np.ones(len(X)), X]

# fit model: normal equations with kernel

xw = X.T \* radial\_kernel(x0, X, tau) # XTranspose \* W

beta = np.linalg.pinv(xw @ X) @ xw @ Y # @ Matrix Multiplication or Dot Product # predict value

return x0 @ beta

# @ Matrix Multiplication or Dot Product for prediction

defradial\_kernel(x0, X, tau):

returnnp.exp(np.sum((X - x0) \*\* 2, axis=1) / (-2 \* tau \* tau))

# Weight or Radial Kernal Bias Function

n = 1000 # generate dataset

X = np.linspace(-3, 3, num=n)

print("The Data Set ( 10 Samples) X :\n",X[1:10])

Y = np.log(np.abs(X \*\* 2 - 1) + .5)

print("The Fitting Curve Data Set (10 Samples) Y :\n",Y[1:10]) # jitter X

X += np.random.normal(scale=.1, size=n)

print("Normalised (10 Samples) X :\n",X[1:10])

domain = np.linspace(-3, 3, num=300)

print(" Xo Domain Space(10 Samples) :\n",domain[1:10])

defplot\_lwr(tau): # prediction through regression

prediction = [local\_regression(x0, X, Y, tau) for x0 in domain]

plot = figure(plot\_width=400, plot\_height=400)

plot.title.text='tau=%g' % tau

plot.scatter(X, Y, alpha=.3)

plot.line(domain, prediction, line\_width=2, color='red')

return plot

show(gridplot([ [plot\_lwr(10.), plot\_lwr(1.)],

[plot\_lwr(0.1), plot\_lwr(0.01)] ]))

OUTPUT

BokehJS 0.13.0 successfully loaded.

The Data Set ( 10 Samples) X :

[-2.99399399 -2.98798799 -2.98198198 -2.97597598 -2.96996997 -2.96396396

-2.95795796 -2.95195195 -2.94594595]

The Fitting Curve Data Set (10 Samples) Y :

[2.13582188 2.13156806 2.12730467 2.12303166 2.11874898 2.11445659

2.11015444 2.10584249 2.10152068]

Normalised (10 Samples) X :

[-2.90641945 -3.01116206 -2.95953729 -2.95577671 -2.74074931 -2.65843895

-2.8770543 -2.86064353 -2.91268905]

Xo Domain Space(10 Samples) :

[-2.97993311 -2.95986622 -2.93979933 -2.91973244 -2.89966555 -2.87959866

-2.85953177 -2.83946488 -2.81939799]

