EXPERIMENT 6

Implement and demonstrate the ID3 algorithm. Read the training data from a .CSV file.

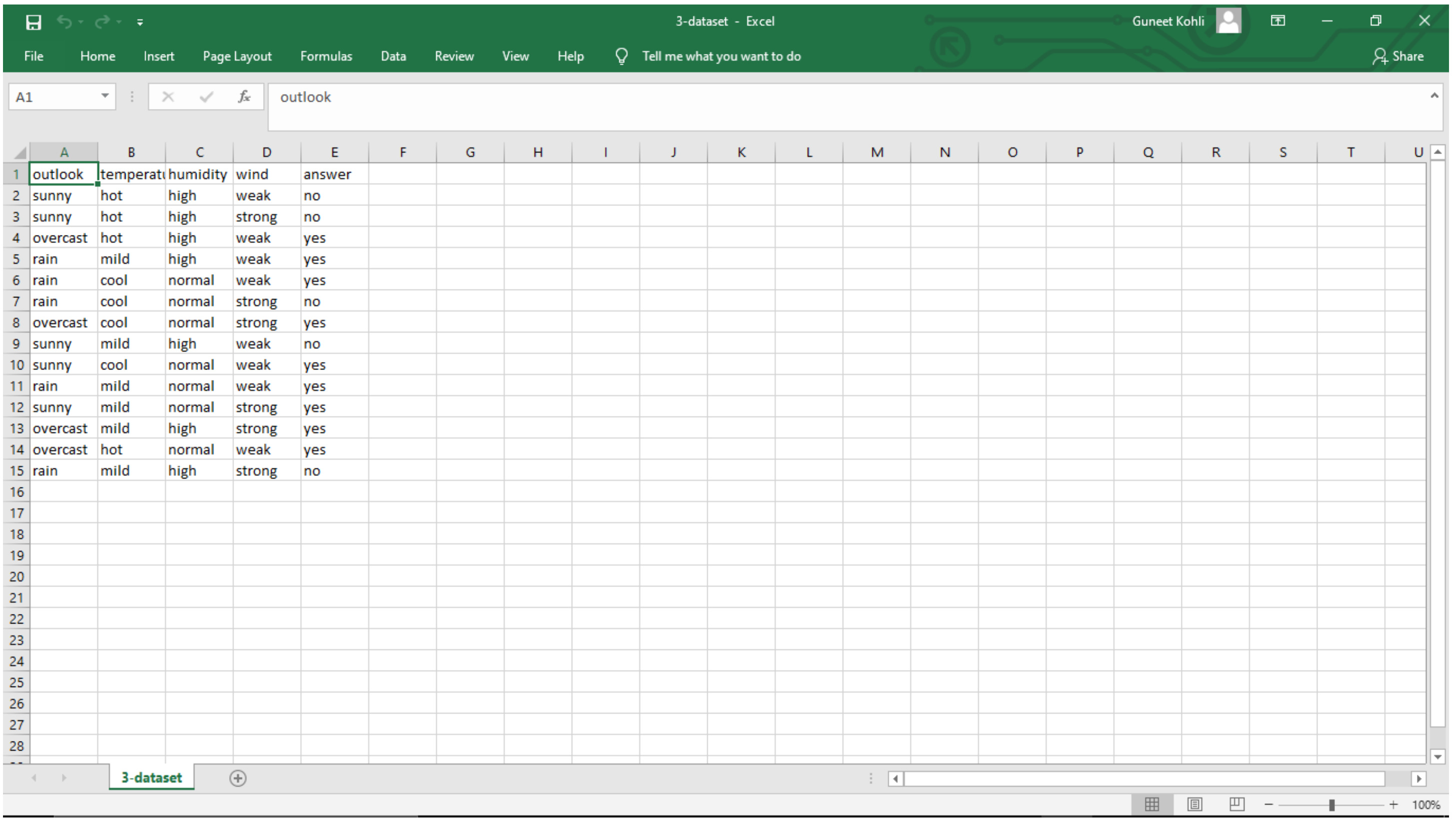
Code:

```
import pandas as pd
import math
import numpy as np

data = pd.read_csv("3-dataset.csv")
features = [feat for feat in data]
features.remove("answer")

class Node:
    def _init_(self):
        self.children = []
        self.value = ""
        self.isLeaf = False
        self.pred = ""

def entropy(examples):
    pos = 0.0
    neg = 0.0
    for _, row in examples.iterrows():
        if row["answer"] == "yes":
            pos += 1
        else:
            neg += 1
    if pos == 0.0 or neg == 0.0:
        return 0.0
    else:
        p = pos / (pos + neg)
        n = neg / (pos + neg)
        return -(p * math.log(p, 2) + n * math.log(n, 2))

def info_gain(examples, attr):
    uniq = np.unique(examples[attr])
    #print ("\n",uniq)
    gain = entropy(examples)
    #print ("\n",gain)
    for u in uniq:
        subdata = examples[examples[attr] == u]
        #print ("\n",subdata)
        sub_e = entropy(subdata)
        gain -= (float(len(subdata)) / float(len(examples))) * sub_e
        #print ("\n",gain)
    return gain

def ID3(examples, attrs):
    root = Node()
```

```
    max_gain = 0
    max_feat = ""
    for feature in attrs:
        #print ("\n",examples)
        gain = info_gain(examples, feature)
        if gain > max_gain:
            max_gain = gain
            max_feat = feature
    root.value = max_feat
    #print ("\nMax feature attr",max_feat)
    uniq = np.unique(examples[max_feat])
    #print ("\n",uniq)
    for u in uniq:
        #print ("\n",u)
        subdata = examples[examples[max_feat] == u]
        #print ("\n",subdata)
        if entropy(subdata) == 0.0:
            newNode = Node()
            newNode.isLeaf = True
            newNode.value = u
            newNode.pred = np.unique(subdata["answer"])
            root.children.append(newNode)
        else:
            dummyNode = Node()
            dummyNode.value = u
            new_attrs = attrs.copy()
            new_attrs.remove(max_feat)
            child = ID3(subdata, new_attrs)
            dummyNode.children.append(child)
            root.children.append(dummyNode)
    return root

def printTree(root: Node, depth=0):
    for i in range(depth):
        print("\t", end="")
    print(root.value, end="")
    if root.isLeaf:
        print(" -> ", root.pred)
    print()
    for child in root.children:
        printTree(child, depth + 1)

root = ID3(data, features)
printTree(root)
```

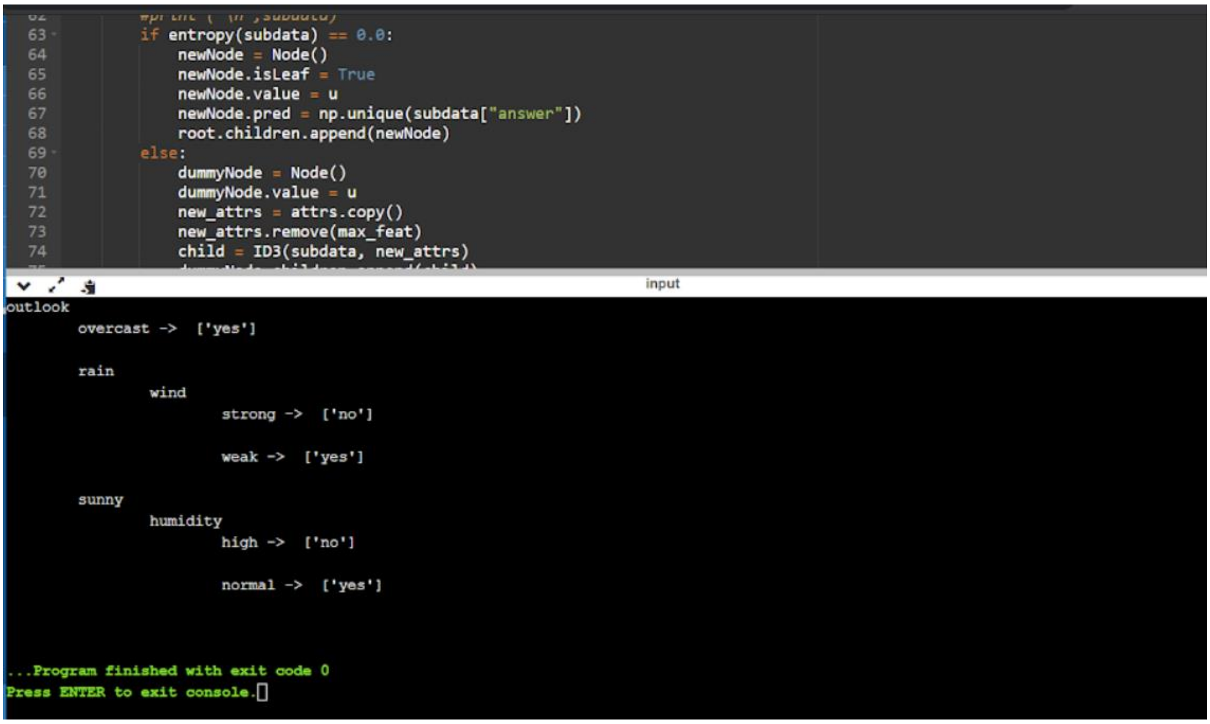
GUNEET KOHLI D3 CSE A1 1805172

Dataset:



outlook	temperatu	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

Output:



```
    if entropy(subdata) == 0.0:
        newNode = Node()
        newNode.isLeaf = True
        newNode.value = u
        newNode.pred = np.unique(subdata["answer"])
        root.children.append(newNode)
    else:
        dummyNode = Node()
        dummyNode.value = u
        new_attrs = attrs.copy()
        new_attrs.remove(max_feat)
        child = ID3(subdata, new_attrs)
```

```
outlook
        overcast -> ['yes']

        rain
                wind
                        strong -> ['no']

                        weak -> ['yes']

        sunny
                humidity
                        high -> ['no']

                        normal -> ['yes']



...Program finished with exit code 0
Press ENTER to exit console.
```

EXPERIMENT 7

Implement and demonstrate a perceptron model with two inputs for an OR and AND Gate.

Code:

i) OR Gate

```
import numpy as np
# define unit step function

def unitStep(v):
    if v >= 0:
        return 1
    else:
        return 0
# design Perceptron Model

def perceptronModel(x, w, b):
    v = np.dot(w, x) + b
    y = unitStep(v)
    return y
def OR_logicFunction(x):
    w = np.array([1, 1])
    b = -0.5
    return perceptronModel(x, w, b)
# testing the Perceptron Model

test1 = np.array([0, 0])
test2 = np.array([0, 1])
test3 = np.array([1, 0])
test4 = np.array([1, 1])
print("OR({}, {}) = {}".format(0, 0, OR_logicFunction(test1)))
print("OR({}, {}) = {}".format(0, 1, OR_logicFunction(test2)))
print("OR({}, {}) = {}".format(1, 0, OR_logicFunction(test3)))
print("OR({}, {}) = {}".format(1, 1, OR_logicFunction(test4)))
```

```
print("OR({}, {}) = {}".format(0, 0, OR_logicFunction(test1)))
print("OR({}, {}) = {}".format(0, 1, OR_logicFunction(test2)))
print("OR({}, {}) = {}".format(1, 0, OR_logicFunction(test3)))
print("OR({}, {}) = {}".format(1, 1, OR_logicFunction(test4)))
```

```
OR(0, 0) = 0
OR(0, 1) = 1
OR(1, 0) = 1
OR(1, 1) = 1
```

ii) AND Gate

```
def AND_logicFunction(x):
    w = np.array([1, 1])
    b = -1.5
    return perceptronModel(x, w, b)
# testing the Perceptron Model


test1 = np.array([0, 0])
test2 = np.array([0, 1])
test3 = np.array([1, 0])
test4 = np.array([1, 1])
print("AND({}, {}) = {}".format(0, 0, AND_logicFunction(test1)))
print("AND({}, {}) = {}".format(0, 1, AND_logicFunction(test2)))
print("AND({}, {}) = {}".format(1, 0, AND_logicFunction(test3)))
print("AND({}, {}) = {}".format(1, 1, AND_logicFunction(test4)))
```

```
In [9]: print("AND({}, {}) = {}".format(0, 0, AND_logicFunction(test1)))
        print("AND({}, {}) = {}".format(0, 1, AND_logicFunction(test2)))
        print("AND({}, {}) = {}".format(1, 0, AND_logicFunction(test3)))
        print("AND({}, {}) = {}".format(1, 1, AND_logicFunction(test4)))
```

```
AND(0, 0) = 0
AND(0, 1) = 0
AND(1, 0) = 0
AND(1, 1) = 1
```

EXPERIMENT 8

Implement and demonstrate Backpropagation algorithm for ANN.

Code:

```
#Backpropagation algorithm
import string
import math
import random
class Neural:
	def __init__(self, pattern):
		#
		# Lets take 2 input nodes, 3 hidden nodes and 1 output node.
		# Hence, Number of nodes in input(ni)=2, hidden(nh)=3, output(no)=1.
		#
		self.ni=3
		self.nh=3
		self.no=1

		#
		# Now we need node weights. We'll make a two dimensional array that maps
node from one layer to the next.
		# i-th node of one layer to j-th node of the next.
		#
		self.wih = []
		for i in range(self.ni):
			self.wih.append([0.0]*self.nh)

		self.who = []
		for j in range(self.nh):
			self.who.append([0.0]*self.no)

		#
		# Now that weight matrices are created, make the activation matrices.
		#
		self.ai, self.ah, self.ao = [],[],[]
		self.ai=[1.0]*self.ni
		self.ah=[1.0]*self.nh
```

```
            self.ao=[1.0]*self.no

            # To ensure node weights are randomly assigned, with some bounds on
#values, we pass it through ranomizeMatrix()

            randomizeMatrix(self.wih,-0.2,0.2)
            randomizeMatrix(self.who,-2.0,2.0)

            # To incorporate momentum factor, introduce another array for the
#'previous change'.

            self.cih = []
            self.cho = []
            for i in range(self.ni):
                  self.cih.append([0.0]*self.nh)
            for j in range(self.nh):
                  self.cho.append([0.0]*self.no)

      # backpropagate() takes as input, the patterns entered, the target values and
#the obtained values.
      # Based on these values, it adjusts the weights so as to balance out the error.
      # Also, now we have M, N for momentum and learning factors respectively.
      def backpropagate(self, inputs, expected, output, N=0.5, M=0.1):
            # We introduce a new matrix called the deltas (error) for the two layers
#output and hidden layer respectively.
            output_deltas = [0.0]*self.no
            for k in range(self.no):
                  # Error is equal to (Target value - Output value)
                  error = expected[k] - output[k]
                  output_deltas[k]=error*dsigmoid(self.ao[k])

            # Change weights of hidden to output layer accordingly.
            for j in range(self.nh):
                  for k in range(self.no):
                        delta_weight = self.ah[j] * output_deltas[k]
                        self.who[j][k]+= M*self.cho[j][k] + N*delta_weight
                        self.cho[j][k]=delta_weight

            # Now for the hidden layer.
```

```
        hidden_deltas = [0.0]*self.nh
        for j in range(self.nh):
            # Error as given by formule is equal to the sum of (Weight from each
#node in hidden layer times output delta of output node)
            # Hence delta for hidden layer = sum
(self.who[j][k]*output_deltas[k])
            error=0.0
            for k in range(self.no):
                error+=self.who[j][k] * output_deltas[k]
            # now, change in node weight is given by dsigmoid() of activation of
#each hidden node times the error.
            hidden_deltas[j]= error * dsigmoid(self.ah[j])


        for i in range(self.ni):
            for j in range(self.nh):
                delta_weight = hidden_deltas[j] * self.ai[i]
                self.wih[i][j]+= M*self.cih[i][j] + N*delta_weight
                self.cih[i][j]=delta_weight


    # Main testing function. Used after all the training and Backpropagation is
#completed.
    def test(self, patterns):
        for p in patterns:
            inputs = p[0]
            print ('For input:', p[0], ' Output -->', self.runNetwork(inputs),
'\tTarget: ', p[1])


    # So, runNetwork was needed because, for every iteration over a pattern []
#array, we need to feed the values.
    def runNetwork(self, feed):
        if(len(feed)!=self.ni-1):
            print ('Error in number of input values.')


        # First activate the ni-1 input nodes.
        for i in range(self.ni-1):
            self.ai[i]=feed[i]


        #
        # Calculate the activations of each successive layer's nodes.
        #
```

```
            for j in range(self.nh):
                  sum=0.0
                  for i in range(self.ni):
                        sum+=self.ai[i]*self.wih[i][j]
                  # self.ah[j] will be the sigmoid of sum. # sigmoid(sum)
                  self.ah[j]=sigmoid(sum)

            for k in range(self.no):
                  sum=0.0
                  for j in range(self.nh):
                        sum+=self.ah[j]*self.wih[j][k]
                  # self.ah[k] will be the sigmoid of sum. # sigmoid(sum)
                  self.ao[k]=sigmoid(sum)

            return self.ao



      def trainNetwork(self, pattern):
            for i in range(500):
                  # Run the network for every set of input values, get the output
values and Backpropagate them.
                  for p in pattern:
                        # Run the network for every tuple in p.
                        inputs = p[0]
                        out = self.runNetwork(inputs)
                        expected = p[1]
                        self.backpropagate(inputs,expected,out)
            self.test(pattern)


def randomizeMatrix ( matrix, a, b):
      for i in range ( len (matrix) ):
            for j in range ( len (matrix[0]) ):
                  # For each of the weight matrix elements, assign a random weight
uniformly between the two bounds.
                  matrix[i][j] = random.uniform(a,b)


def sigmoid(x):
      return 1 / (1 + math.exp(-x))
```

```
# Sigmoid function derivative.
def dsigmoid(y):
	return y * (1 - y)
def main():
	# take the input pattern as a map. Suppose we are working for AND gate.
	pat = [
		[[0,0], [0]],
		[[0,1], [0]],
		[[1,0], [0]],
		[[1,1], [1]]
	]
	newNeural = Neural(pat)
	newNeural.trainNetwork(pat)

if __name__ == "__main__":
	main()
```

Output:

```
	newNeural = Neural(pat)
	newNeural.trainNetwork(pat)


if __name__ == "__main__":
	main()
```

```
For input: [0, 0]  Output --> [0.9962562697369688]      Target:  [0]
For input: [0, 1]  Output --> [0.9968120675219813]      Target:  [0]
For input: [1, 0]  Output --> [0.9967963375215078]      Target:  [0]
For input: [1, 1]  Output --> [0.9970271700338775]      Target:  [1]
```

```
In [7]: def AND_logicFunction(x):
```

```
For input: [0, 0]  Output --> [0.707159457566464]       Target:  [0]
For input: [0, 1]  Output --> [0.9485971956200733]      Target:  [1]
For input: [1, 0]  Output --> [0.9481348576876182]      Target:  [1]
For input: [1, 1]  Output --> [0.9934296547591035]      Target:  [1]
```

EXPERIMENT 9

Implement and demonstrate Naïve Bayes algorithm.

Code:

```
def probAttr(data,attr,val):
    Total=data.shape[0]    #Get column length
    cnt = len(data[data[attr] == val]) #Count of Attribute [attr] equal to val
    return cnt,cnt/Total
def train(data,Attr,conceptVals,concept):
    conceptProbs = {} #P(A)
    countConcept={}
    for cVal in conceptVals: #Get probablity and count of Yes and No
        countConcept[cVal],conceptProbs[cVal] = probAttr(data,concept,cVal)

    AttrConcept = {} #P(X/A)
    probability_list = {} #P(X)
    for att in Attr: #Create a tree for attribute
        probability_list[att] = {}
        AttrConcept[att] = {}
        for val in Attr[att]: #Create Tree for Attribute value
            AttrConcept[att][val] = {}
            a,probability_list[att][val] = probAttr(data,att,val) #Get Probablity for 
att equal to val
            for cVal in conceptVals: #Create Tree to hold yes and no values
                dataTemp = data[data[att]==val] #Calculate att equal to val and 
concept equal to cVal
                AttrConcept[att][val][cVal] = len(dataTemp[dataTemp[concept] == 
cVal])/countConcept[cVal]

    print("P(A) : ",conceptProbs,"\n")
    print("P(X/A) : ",AttrConcept,"\n")
    print("P(X) : ",probability_list,"\n")
    return conceptProbs,AttrConcept,probability_list


def test(examples,Attr,concept_list,conceptProbs,AttrConcept,probability_list):
    misclassification_count=0
    Total = len(examples)    #Get Number of testing set
    for ex in examples:
```

```
        px={}  #Dict to hold final value
        for a in Attr:    #Iterrate thorugh the Tree with Attributes (Refer problem to find the tree)
            for x in ex:  #Iterrate thorugh the Tree for given example
                for c in concept_list:   #Iterrate thorugh the Tree using concepts
                    if x in AttrConcept[a]:  #Check if the value of x refering in same sub-tree of P(X/A)
                        if c not in px: #If c not in px multiply P(A) with 1st Itteration (for 1st value of x)
                            px[c] = conceptProbs[c]*AttrConcept[a][x][c]/probability_list[a][x]
                        else:  #multiply px in next Itterations (for next values of x)
                            px[c] = px[c]*AttrConcept[a][x][c]/probability_list[a][x]
        print(px)
        classification = max(px,key=px.get)  #Key of Maximum of px is required Classification
        print("Classification :",classification,"Expected :",ex[-1])
        if(classification!=ex[-1]):
            misclassification_count+=1
    misclassification_rate=misclassification_count*100/Total
    accuracy=100-misclassification_rate
    print("Misclassification Count={}".format(misclassification_count))
    print("Misclassification Rate={}%".format(misclassification_rate))
    print("Accuracy={}%".format(accuracy))


def main():
    import pandas as pd
    from pandas import DataFrame
    data = pd.read_csv(r"C:\Users\GuneetKohli\Desktop\PlayTennis_train1.csv")
    #data = DataFrame.from_csv('PlayTennis_train1.csv')
    #print(data)
    concept=str(list(data)[-1])
    concept_list = set(data[concept])
    Attr={}
    for a in list(data)[:-1]:    #Get attribute values
        Attr[a] = set(data[a])
    conceptProbs,AttrConcept,probability_list = train(data,Attr,concept_list,concept)

    #examples = DataFrame.from_csv('PlayTennis_test1.csv')
```

```
    #print(examples)

#test(examples.values,Attr,concept_list,conceptProbs,AttrConcept,probability_list)


main()
```

Output:

```
In [23]: main()

P(A) :  {'No': 0.35714285714285715, 'Yes': 0.6428571428571429}

P(X/A) :  {'Sno': {0: {'No': 0.2, 'Yes': 0.0}, 1: {'No': 0.2, 'Yes': 0.0}, 2: {'No': 0.0, 'Yes': 0.1111111111111111}, 3: {'No':
0.0, 'Yes': 0.1111111111111111}, 4: {'No': 0.0, 'Yes': 0.1111111111111111}, 5: {'No': 0.2, 'Yes': 0.0}, 6: {'No': 0.0, 'Yes':
0.1111111111111111}, 7: {'No': 0.2, 'Yes': 0.0}, 8: {'No': 0.0, 'Yes': 0.1111111111111111}, 9: {'No': 0.0, 'Yes': 0.11111111111
11111}, 10: {'No': 0.0, 'Yes': 0.1111111111111111}, 11: {'No': 0.0, 'Yes': 0.1111111111111111}, 12: {'No': 0.0, 'Yes': 0.111111
1111111111}, 13: {'No': 0.2, 'Yes': 0.0}}, 'Outlook': {'Rain': {'No': 0.4, 'Yes': 0.3333333333333333}, 'Overcast': {'No': 0.0,
'Yes': 0.2222222222222222}, 'Sunny ': {'No': 0.6, 'Yes': 0.2222222222222222}, ' Overcast': {'No': 0.0, 'Yes': 0.222222222222222
2}}, 'Temperature': {'Mild': {'No': 0.4, 'Yes': 0.4444444444444444}, 'Cool': {'No': 0.2, 'Yes': 0.3333333333333333}, 'Hot': {'N
o': 0.4, 'Yes': 0.2222222222222222}}, 'Humidity': {'High': {'No': 0.8, 'Yes': 0.3333333333333333}, 'Normal': {'No': 0.2, 'Yes':
0.6666666666666666}}, 'Wind': {'Weak': {'No': 0.4, 'Yes': 0.6666666666666666}, 'Strong': {'No': 0.6, 'Yes': 0.333333333333333
3}}}

P(X) :  {'Sno': {0: 0.07142857142857142, 1: 0.07142857142857142, 2: 0.07142857142857142, 3: 0.07142857142857142, 4: 0.071428571
42857142, 5: 0.07142857142857142, 6: 0.07142857142857142, 7: 0.07142857142857142, 8: 0.07142857142857142, 9: 0.0714285714285714
2, 10: 0.07142857142857142, 11: 0.07142857142857142, 12: 0.07142857142857142, 13: 0.07142857142857142}, 'Outlook': {'Rain': 0.3
5714285714285715, 'Overcast': 0.14285714285714285, 'Sunny ': 0.35714285714285715, ' Overcast': 0.14285714285714285}, 'Temperatu
re': {'Mild': 0.42857142857142855, 'Cool': 0.2857142857142857, 'Hot': 0.2857142857142857}, 'Humidity': {'High': 0.5, 'Normal':
0.5}, 'Wind': {'Weak': 0.5714285714285714, 'Strong': 0.42857142857142855}}
```

Dataset Used:

PlayTennis_train1

File Home Insert Page Layout Formulas Data Review View Help Tell me what

A1 Sno

	A	B	C	D	E	F	G	H	I	J
1	Sno	Outlook	Temperat	Humidity	Wind	PlayTennis				
2	0	Sunny	Hot	High	Weak	No				
3	1	Sunny	Hot	High	Strong	No				
4	2	Overcast	Hot	High	Weak	Yes				
5	3	Rain	Mild	High	Weak	Yes				
6	4	Rain	Cool	Normal	Weak	Yes				
7	5	Rain	Cool	Normal	Strong	No				
8	6	Overcast	Cool	Normal	Strong	Yes				
9	7	Sunny	Mild	High	Weak	No				
10	8	Sunny	Cool	Normal	Weak	Yes				
11	9	Rain	Mild	Normal	Weak	Yes				
12	10	Sunny	Mild	Normal	Strong	Yes				
13	11	Overcast	Mild	High	Strong	Yes				
14	12	Overcast	Hot	Normal	Weak	Yes				
15	13	Rain	Mild	High	Strong	No				

EXPERIMENT 10

Implement and demonstrate Bayesian Networks.

Code:

```
#Import required packages
import math
from pomegranate import *

# Initially the door selected by the guest is completely random
guest =DiscreteDistribution( { 'A': 1./3, 'B': 1./3, 'C': 1./3 } )

# The door containing the prize is also a random process
prize =DiscreteDistribution( { 'A': 1./3, 'B': 1./3, 'C': 1./3 } )

# The door Monty picks, depends on the choice of the guest and the prize door
monty =ConditionalProbabilityTable(
[[ 'A', 'A', 'A', 0.0 ],
[ 'A', 'A', 'B', 0.5 ],
[ 'A', 'A', 'C', 0.5 ],
[ 'A', 'B', 'A', 0.0 ],
[ 'A', 'B', 'B', 0.0 ],
[ 'A', 'B', 'C', 1.0 ],
[ 'A', 'C', 'A', 0.0 ],
[ 'A', 'C', 'B', 1.0 ],
[ 'A', 'C', 'C', 0.0 ],
[ 'B', 'A', 'A', 0.0 ],
[ 'B', 'A', 'B', 0.0 ],
[ 'B', 'A', 'C', 1.0 ],
[ 'B', 'B', 'A', 0.5 ],
[ 'B', 'B', 'B', 0.0 ],
[ 'B', 'B', 'C', 0.5 ],
[ 'B', 'C', 'A', 1.0 ],
[ 'B', 'C', 'B', 0.0 ],
[ 'B', 'C', 'C', 0.0 ],
[ 'C', 'A', 'A', 0.0 ],
```

```
[ 'C', 'A', 'B', 1.0 ],
[ 'C', 'A', 'C', 0.0 ],
[ 'C', 'B', 'A', 1.0 ],
[ 'C', 'B', 'B', 0.0 ],
[ 'C', 'B', 'C', 0.0 ],
[ 'C', 'C', 'A', 0.5 ],
[ 'C', 'C', 'B', 0.5 ],
[ 'C', 'C', 'C', 0.0 ]], [guest, prize] )

d1 = State( guest, name="guest" )
d2 = State( prize, name="prize" )
d3 = State( monty, name="monty" )

#Building the Bayesian Network
network = BayesianNetwork( "Solving the Monty Hall Problem With Bayesian Networks" )
network.add_states(d1, d2, d3)
network.add_edge(d1, d3)
network.add_edge(d2, d3)
network.bake()


beliefs = network.predict_proba({'guest' : 'A', 'monty' : 'B'})
print("n".join( "{}t{}".format( state.name, str(belief) ) for state, belief in zip( network.states, beliefs )))
```

Output:

```
In [11]: beliefs = network.predict_proba({'guest' : 'A', 'monty' : 'B'})
         print("n".join( "{}t{}".format( state.name, str(belief) ) for state, belief in zip( network.states, beliefs )))

guesttAnprizet{
    "class" : "Distribution",
    "dtype" : "str",
    "name" : "DiscreteDistribution",
    "parameters" : [
        {
            "A" : 0.3333333333333334,
            "B" : 0.0,
            "C" : 0.6666666666666664
        }
    ],
    "frozen" : false
}nmontytB
```

EXPERIMENT 11

Implement and demonstrate Genetic algorithm.

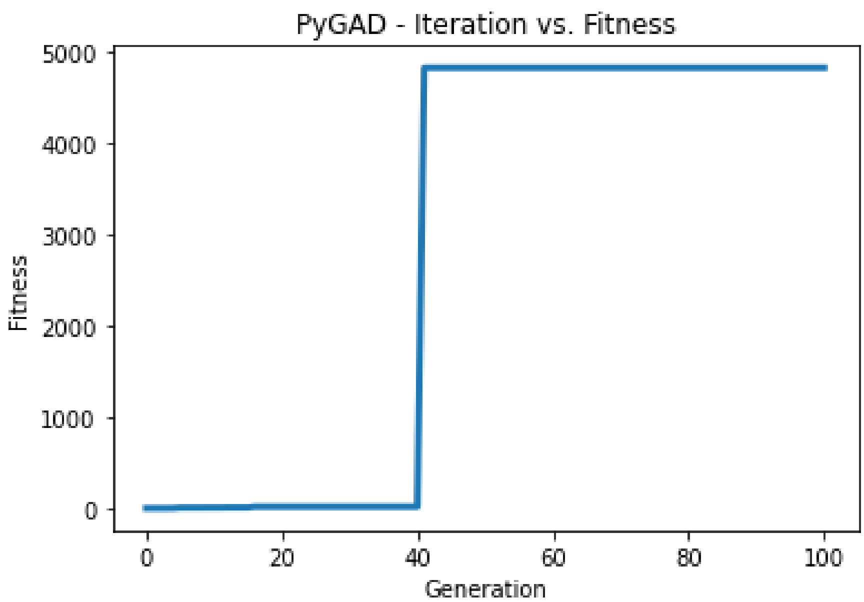
Code:

```
import pygad

import numpy

function_inputs = [4,-2,3.5,5,-11,-4.7] # Function inputs.

desired_output = 44 # Function output.


def fitness_func(solution, solution_idx):

    # Calculating the fitness value of each solution in the current population.

    # The fitness function calulates the sum of products between each input and its corresponding weight.

    output = numpy.sum(solution*function_inputs)

    # The value 0.000001 is used to avoid the Inf value when the denominator numpy.abs(output - desired_output) is 0.0.

    fitness = 1.0 / (numpy.abs(output - desired_output) + 0.000001)

    return fitness


      fitness_function = fitness_func

num_generations = 100 # Number of generations.

num_parents_mating = 10 # Number of solutions to be selected as parents in the mating pool.


# To prepare the initial population, there are 2 ways:

# 1) Prepare it yourself and pass it to the initial_population parameter. This way is useful when the user wants to start the genetic algorithm with a custom initial population.

# 2) Assign valid integer values to the sol_per_pop and num_genes parameters. If the initial_population parameter exists, then the sol_per_pop and num_genes parameters are useless.

sol_per_pop = 20 # Number of solutions in the population.

num_genes = len(function_inputs)


parent_selection_type = "sss" # Type of parent selection.

keep_parents = -1 # Number of parents to keep in the next population. -1 means keep all parents and 0 means keep nothing.


crossover_type = "single_point" # Type of the crossover operator.
```

```
# Parameters of the mutation operation.

mutation_type = "random" # Type of the mutation operator.

mutation_percent_genes = 10 # Percentage of genes to mutate. This parameter has no action if the parameter mutation_num_genes exists or when mutation_type is None.

last_fitness = 0


def callback_generation(ga_instance):

    global last_fitness

    print("Generation = {generation}".format(generation=ga_instance.generations_completed))

    print("Fitness    = {fitness}".format(fitness=ga_instance.best_solution(pop_fitness=ga_instance.last_generation_fitness)[1]))

    print("Change     = {change}".format(change=ga_instance.best_solution(pop_fitness=ga_instance.last_generation_fitness)[1] - last_fitness))

    last_fitness = ga_instance.best_solution(pop_fitness=ga_instance.last_generation_fitness)[1]


# Creating an instance of the GA class inside the ga module. Some parameters are initialized within the constructor.

ga_instance = pygad.GA(num_generations=num_generations,

                       num_parents_mating=num_parents_mating,

                       fitness_func=fitness_function,

                       sol_per_pop=sol_per_pop,

                       num_genes=num_genes,

                       parent_selection_type=parent_selection_type,

                       keep_parents=keep_parents,

                       crossover_type=crossover_type,

                       mutation_type=mutation_type,

                       mutation_percent_genes=mutation_percent_genes,

                       on_generation=callback_generation)


ga_instance.run()


# After the generations complete, some plots are showed that summarize the how the outputs/fitenss values evolve over generations.

ga_instance.plot_result()


# Returning the details of the best solution.

solution, solution_fitness, solution_idx = ga_instance.best_solution()
```

```
print("Parameters of the best solution : {solution}".format(solution=solution))

print("Fitness value of the best solution = {solution_fitness}".format(solution_fitness=solution_fitness))

print("Index of the best solution : {solution_idx}".format(solution_idx=solution_idx))


prediction = numpy.sum(numpy.array(function_inputs)*solution)

print("Predicted output based on the best solution : {prediction}".format(prediction=prediction))


if ga_instance.best_solution_generation != -1:

    print("Best fitness value reached after {best_solution_generation} generations.".format(best_solution_generation=ga_instance.best_solution_generation))


# Saving the GA instance.

filename = 'genetic' # The filename to which the instance is saved. The name is without extension.

ga_instance.save(filename=filename)


# Loading the saved GA instance.

loaded_ga_instance = pygad.load(filename=filename)

loaded_ga_instance.plot_result()
```

Output:



```
Generation = 1
Fitness    = 1.0063465291938989
Change     = 1.0063465291938989
Generation = 2
Fitness    = 1.0063465291938989
Change     = 0.0
Generation = 3
Fitness    = 1.0063465291938989
Change     = 0.0
Generation = 4
Fitness    = 1.8394365812114146
Change     = 0.8330900520175157
Generation = 5
Fitness    = 7.484543212299999
Change     = 5.645106631088584
Generation = 6
Fitness    = 7.484543212299999
Change     = 0.0
Generation = 7
```

jupyter Genetic Algorithm Last Checkpoint: 5 minutes ago (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Code

```
if ga_instance.best_solution_generation != -1:
    print("Best fitness value reached after {best_solution_generation} generations.".format(best_solution_generation=ga_instance.

# Saving the GA instance.
filename = 'genetic' # The filename to which the instance is saved. The name is without extension.
ga_instance.save(filename=filename)

# Loading the saved GA instance.
loaded_ga_instance = pygad.load(filename=filename)
loaded_ga_instance.plot_result()
```

```
Parameters of the best solution : [ 1.40047901 -0.2222016  -1.35255005  1.5242523  -3.63935663  1.0566749 ]
Fitness value of the best solution = 4819.633237995131
Index of the best solution : 0
Predicted output based on the best solution : 44.00020648466753
Best fitness value reached after 41 generations.
```