**Practical No: 1**

**Aim: Design an Expert system using AIML.**

1. **An expert system for responding the patient query for identifying the flu.**

**Code**:

* std-startup1.xml

<aiml>

<category>

<pattern>LOAD AIML B</pattern>

<template>

<learn>basic\_chat1.aiml</learn>

</template>

</category>

</aiml>

* basic\_chat1.aiml

<aiml version="1.0.1" encoding="UTF-8">

<!-- basic\_chat.aiml -->

<category>

<pattern>HELLO</pattern>

<template>

WHAT WOULD YOU LIKE TO DISCUSS? : HEALTH, MOVIES

</template>

</category>

<category>

<pattern>MOVIES</pattern>

<template>

YES <set name = "topic">MOVIES</set>

</template>

</category>

<category>

<pattern>HEALTH</pattern>

<template>

YES <set name = "topic">HEALTH</set>

</template>

</category>

<topic name ="MOVIES">

<category>

<pattern>\*</pattern>

<template>

DO YOU LIKE COMEDY MOVIES?

</template>

</category>

<category>

<pattern>YES</pattern>

<template>

I TOO LIKE COMEDY MOVIES

</template>

</category>

<category>

<pattern>NO</pattern>

<template>

BUT I LIKE COMEDY MOVIES

</template>

</category>

</topic>

<topic name ="HEALTH">

<category>

<pattern>\*</pattern>

<template>

DO YOU HAVE FEVER?

</template>

</category>

<category>

<pattern>YES</pattern>

<template>

PLEASE TAKE MEDICINES AND PROPER REST

</template>

</category>

<category>

<pattern>NO</pattern>

<template>

GO OUT FOR A WALK AND LISTEN MUSIC

</template>

</category>

</topic>

<category>

<pattern>NICE TALKING TO YOU</pattern>

<template>

SAME HERE...!!

</template>

</category>

</aiml>

* chat\_bot.py

import aiml

# Create the kernel and learn AIML files

kernel = aiml.Kernel()

kernel.learn("std-startup1.xml")

kernel.respond("load aiml b")

# Press CTRL-C to break this loop

while True:

message = input("Enter your message to the bot: ")

if message == "quit":

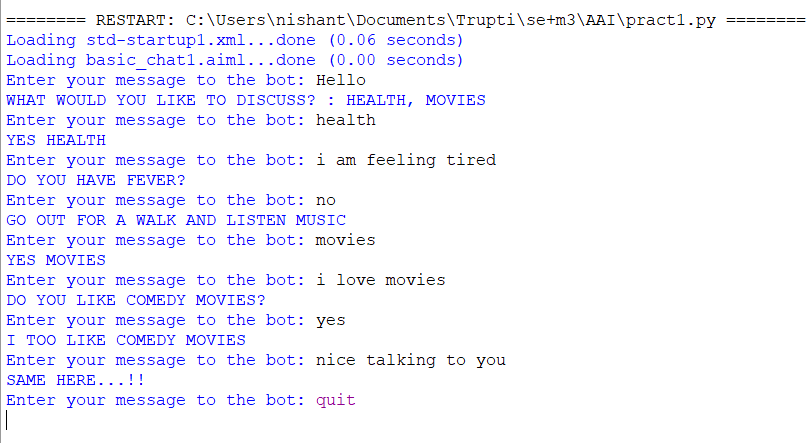
break

else:

bot\_response = kernel.respond(message)

print(bot\_response)

**Output:**



**Practical No: 2**

**Aim: Design a bot using AIML.**

**Code**:

* std-startup.xml

<aiml version="1.0.1" encoding="UTF-8">

<!-- std-startup.xml -->

<!-- Category is an atomic AIML unit -->

<category>

<!-- Pattern to match in user input -->

<!-- If user enters "LOAD AIML B" -->

<pattern>LOAD AIML B</pattern>

<!-- Template is the response to the pattern -->

<!-- This learn an aiml file -->

<template>

<learn>basic\_chat.aiml</learn>

<!-- You can add more aiml files here -->

<!--<learn>more\_aiml.aiml</learn>-->

</template>

</category>

</aiml>

* basic\_chat.aiml

<aiml version="1.0.1" encoding="UTF-8">

<!-- basic\_chat.aiml -->

<category>

<pattern>HELLO</pattern>

<template>

Well, hello!

</template>

</category>

<category>

<pattern>WHAT ARE YOU</pattern>

<template>

I'm a bot, silly!

</template>

</category>

<category>

<pattern>MY NAME IS \*</pattern>

<template>

<set name = "username"><star/></set> is the nice name.

</template>

</category>

<category>

<pattern>I LIKE \*</pattern>

<template>

<set name = "liking"><star/></set> is also my favourite.

</template>

</category>

<category>

<pattern>MY DOG NAME IS \*</pattern>

<template>

THAT IS INTERESTING THAT YOU HAVE A DOG NAMED <set name ="dog"><star/></set>.

</template>

</category>

<category>

<pattern>Bye</pattern>

<template>

Bye!!! <get name = "username"/> Thanks for talking with me.

</template>

</category>

</aiml>

* chat\_bot.py

import aiml

import time

# Create the kernel and learn AIML files

kernel = aiml.Kernel()

kernel.learn("std-startup.xml")

kernel.respond("load aiml b")

# Press CTRL-C to break this loop

while True:

message = input("Enter your message to the bot: ")

if message == "quit":

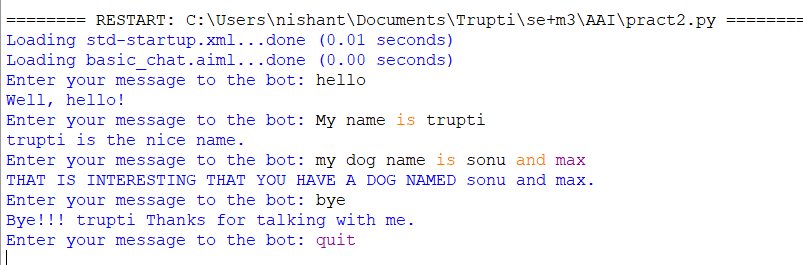
break

else:

bot\_response = kernel.respond(message)

print(bot\_response)

**Output:**



**Practical No: 3**

**Aim: Implement Bayes Theorem using Python.**

1. Consider a human population that may or may not have cancer (Cancer is True or False) and a medical test that returns positive or negative for detecting cancer (Test is Positive or Negative), e.g., like a mammogram for detecting breast cancer. If a randomly selected patient has the test and it comes back positive, what is the probability that the patient has cancer?

**Code**:

# calculate P(A|B) given P(A), P(B|A), P(B|not A)

def bayes\_theorem(p\_a, p\_b\_given\_a, p\_b\_given\_not\_a):

# calculate P(not A)

not\_a = 1 - p\_a

# calculate P(B)

p\_b = p\_b\_given\_a \* p\_a + p\_b\_given\_not\_a \* not\_a

# calculate P(A|B)

p\_a\_given\_b = (p\_b\_given\_a \* p\_a) / p\_b

return p\_a\_given\_b

# P(A)

p\_a = 0.0002

# P(B|A)

p\_b\_given\_a = 0.85

# P(B|not A)

p\_b\_given\_not\_a = 0.05

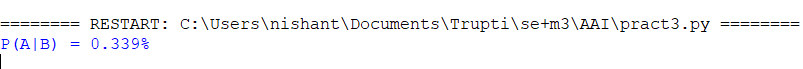
# calculate P(A|B)

result = bayes\_theorem(p\_a, p\_b\_given\_a, p\_b\_given\_not\_a)

# summarize

print('P(A|B) = %.3f%%' % (result \* 100))

**Output:**

****

1. A bag I contains 4 white and 6 black balls while another Bag II contains 4 white and 3 black balls. One ball is drawn at random from one of the bags, and it is found to be black. Find the probability that it was drawn from Bag I.

**Code**:

def bayes\_theorem(P\_E1, P\_E2, P\_A\_given\_E1, P\_A\_given\_E2):

P\_A= P\_A\_given\_E1\* P\_E1 + P\_A\_given\_E2 \* P\_E2

P\_E1\_given\_A = (P\_A\_given\_E1\* P\_E1) / P\_A

return P\_E1\_given\_A

P\_E1 = 0.5

P\_E2 = 0.5

P\_A\_given\_E1= 0.6

P\_A\_given\_E2 = 0.42

result = bayes\_theorem(P\_E1, P\_E2, P\_A\_given\_E1, P\_A\_given\_E2)

print('P(E1|A) = %.3f%%' % (result \* 100))

**Output:**

****

1. Imagine there is a drug test that is 98% accurate, meaning that 98% of the time, it shows a true positive result for someone using the drug, and 98% of the time, it shows a true negative result for nonusers of the drug.

**Code**:

def bayes\_theorem(p\_H, p\_E\_given\_H):

not\_H = 1 - p\_H

p\_E\_given\_not\_H= 1 - p\_E\_given\_H

p\_E = p\_E\_given\_H \* p\_H + p\_E\_given\_not\_H\* not\_H

p\_H\_given\_E = (p\_E\_given\_H \* p\_H) / p\_E

return p\_H\_given\_E

p\_H = 0.005

p\_E\_given\_H = 0.98

result = bayes\_theorem(p\_H, p\_E\_given\_H)

# summarize

print('P(H|E) = %.3f%%' % (result \* 100))

**Output:**



**Practical No: 4**

**Aim:** **Implement Conditional Probability and joint probability using Python.**

**Code**:

import enum, random

class Kid(enum.Enum):

BOY = 0

GIRL = 1

def random\_kid() -> Kid:

return random.choice([Kid.BOY, Kid.GIRL])

both\_girls = 0

older\_girl = 0

either\_girl = 0

random.seed(0)

for \_ in range(10000):

younger = random\_kid()

older = random\_kid()

if older == Kid.GIRL:

older\_girl += 1

if older == Kid.GIRL and younger == Kid.GIRL:

both\_girls += 1

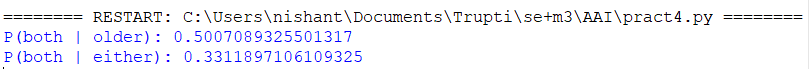
if older == Kid.GIRL or younger == Kid.GIRL:

either\_girl += 1

print("P(both | older):", both\_girls / older\_girl) # 0.5007089325501317

print("P(both | either):", both\_girls / either\_girl) # 0.3311897106109325

**Output:**

****

**Practical No: 5**

**Aim: Write a program for to implement Rule based system.**

**Code**:

go:-

hypothesis(Disease),

write('I believe that the patient have '),

write(Disease),

nl,

write('TAKE CARE '),

undo.

/\*Hypothesis that should be tested\*/

hypothesis(cold) :- cold, !.

hypothesis(flu) :- flu, !.

hypothesis(typhoid) :- typhoid, !.

hypothesis(measles) :- measles, !.

hypothesis(malaria) :- malaria, !.

hypothesis(unknown). /\* no diagnosis\*/

/\*Hypothesis Identification Rules\*/

cold :-

verify(headache),

verify(runny\_nose),

verify(sneezing),

verify(sore\_throat),

write('Advices and Sugestions:'),

nl,

write('1: Tylenol/tab'),

nl,

write('2: panadol/tab'),

nl,

write('3: Nasal spray'),

nl,

write('Please wear warm cloths Because'),

nl.

flu :-

verify(fever),

verify(headache),

verify(chills),

verify(body\_ache),

write('Advices and Sugestions:'),

nl,

write('1: Tamiflu/tab'),

nl,

write('2: panadol/tab'),

nl,

write('3: Zanamivir/tab'),

nl,

write('Please take a warm bath and do salt gargling Because'),

nl.

typhoid :-

verify(headache),

verify(abdominal\_pain),

verify(poor\_appetite),

verify(fever),

write('Advices and Sugestions:'),

nl,

write('1: Chloramphenicol/tab'),

nl,

write('2: Amoxicillin/tab'),

nl,

write('3: Ciprofloxacin/tab'),

nl,

write('4: Azithromycin/tab'),

nl,

write('Please do complete bed rest and take soft Diet Because'),

nl.

measles :-

verify(fever),

verify(runny\_nose),

verify(rash),

verify(conjunctivitis),

write('Advices and Sugestions:'),

nl,

write('1: Tylenol/tab'),

nl,

write('2: Aleve/tab'),

nl,

write('3: Advil/tab'),

nl,

write('4: Vitamin A'),

nl,

write('Please Get rest and use more liquid Because'),

nl.

malaria :-

verify(fever),

verify(sweating),

verify(headache),

verify(nausea),

verify(vomiting),

verify(diarrhea),

write('Advices and Sugestions:'),

nl,

write('1: Aralen/tab'),

nl,

write('2: Qualaquin/tab'),

nl,

write('3: Plaquenil/tab'),

nl,

write('4: Mefloquine'),

nl,

write('Please do not sleep in open air and cover your full skin Because'),

nl.

/\* how to ask questions \*/

ask(Question) :-

write('Does the patient have following symptom:'),

write(Question),

write('? '),

read(Response),

nl,

( (Response == yes ; Response == y)

->

assert(yes(Question)) ;

assert(no(Question)), fail).

:- dynamic yes/1,no/1.

/\*How to verify something \*/

verify(S) :-

(yes(S)

->

true ;

(no(S)

->

fail ;

ask(S))).

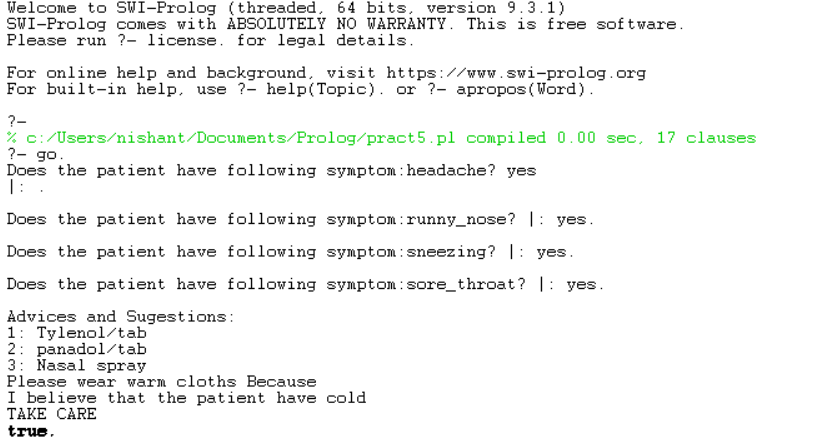
/\* undo all yes/no assertions\*/

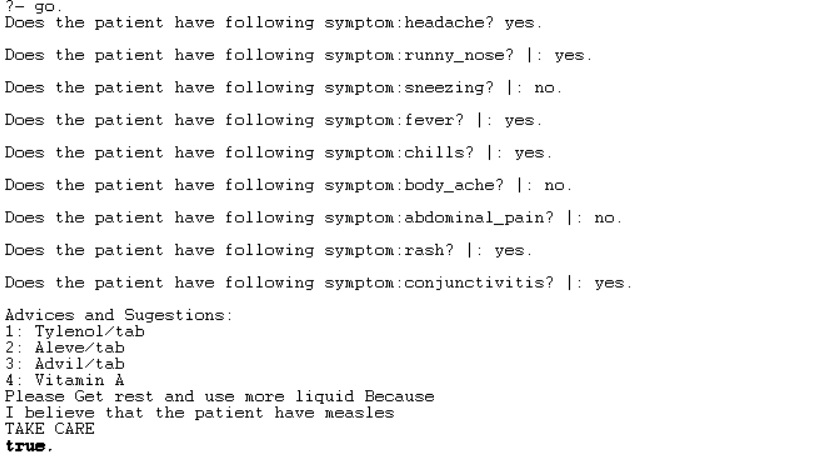
undo :- retract(yes(\_)),fail.

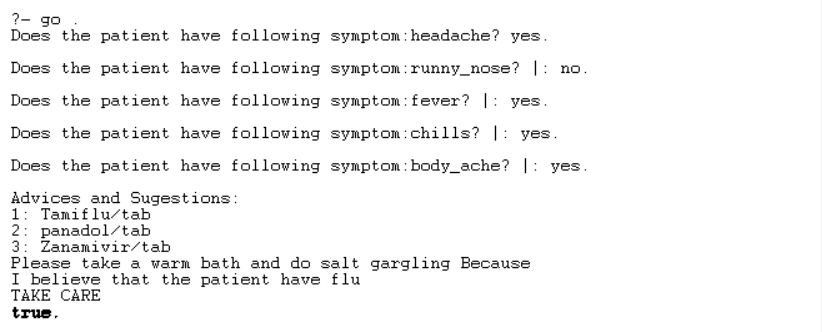
undo :- retract(no(\_)),fail.

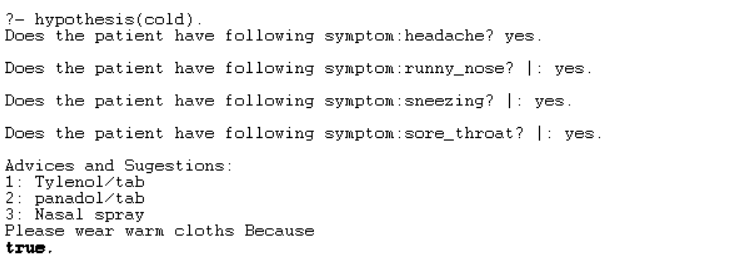
undo.

**Output:**









**Practical No: 6**

**Aim: Design a Fuzzy based application using Python / R.**

1. **Solve Tipping problem using fuzzy logic.**

**Theory:**

The ‘tipping problem’ is commonly used to illustrate the power of fuzzy logic principles to generate complex behaviour from a compact, intuitive set of expert rules. If you are new to the world of fuzzy control systems, you might want to check out the Fuzzy Control Primer before reading through this worked example.

The Tipping Problem Let us create a fuzzy control system which models how you might choose to tip at a restaurant. When tipping, you consider the service and food quality, rated between 0 and 10. You use this to leave a tip of between 0 and 25%.

We would formulate this problem as:

• Antecednets (Inputs)

– service

\* Universe (i.e., crisp value range): How good was the service of the wait staff, on a scale of 0 to 10?

\* Fuzzy set (i.e., fuzzy value range): poor, acceptable, amazing

– food quality

\* Universe: How tasty was the food, on a scale of 0 to 10?

\* Fuzzy set: bad, decent, great

• Consequents (Outputs)

– tip

\* Universe: How much should we tip, on a scale of 0% to 25%

\* Fuzzy set: low, medium, high

• Rules

– IF the service was good or the food quality was good, THEN the tip will be high.

– IF the service was average, THEN the tip will be medium.

– IF the service was poor and the food quality was poor THEN the tip will be low.

• Usage

– If I tell this controller that I rated:

\* The service as 9.8, and

\* The quality as 6.5,

– it would recommend I leave:

\* a 20.2% tip.

**Code**:

import numpy as np

import skfuzzy as fuzz

from skfuzzy import control as ctrl

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

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

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

quality.automf(3)

service.automf(3)

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

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

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

quality['average'].view()

service.view()

tip.view()

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

rule2 = ctrl.Rule(service['average'], tip['medium'])

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

rule1.view()

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

tipping = ctrl.ControlSystemSimulation(tipping\_ctrl)

tipping.input['quality'] = 6.5

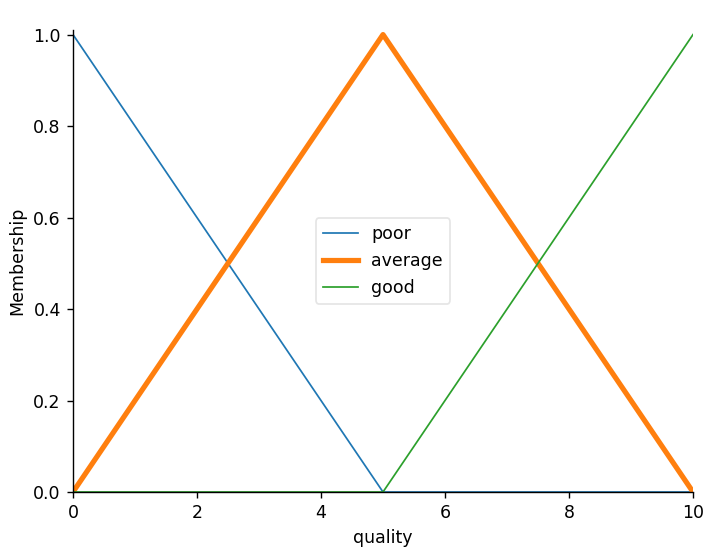
tipping.input['service'] = 9.8

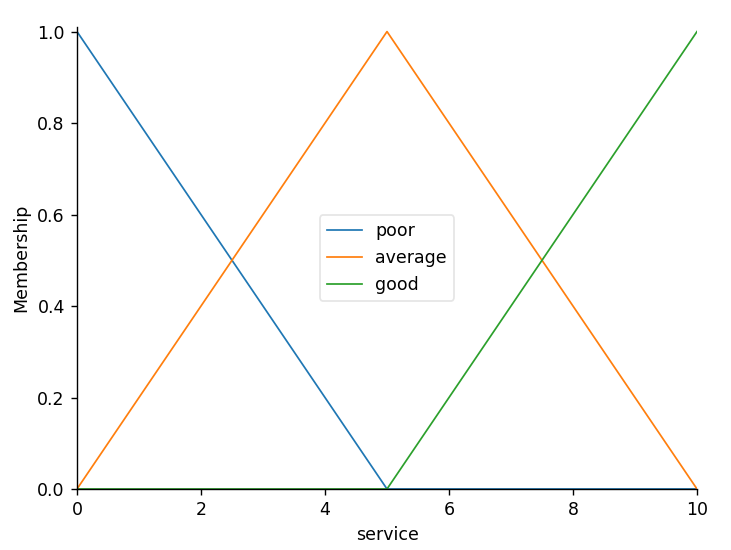
tipping.compute()

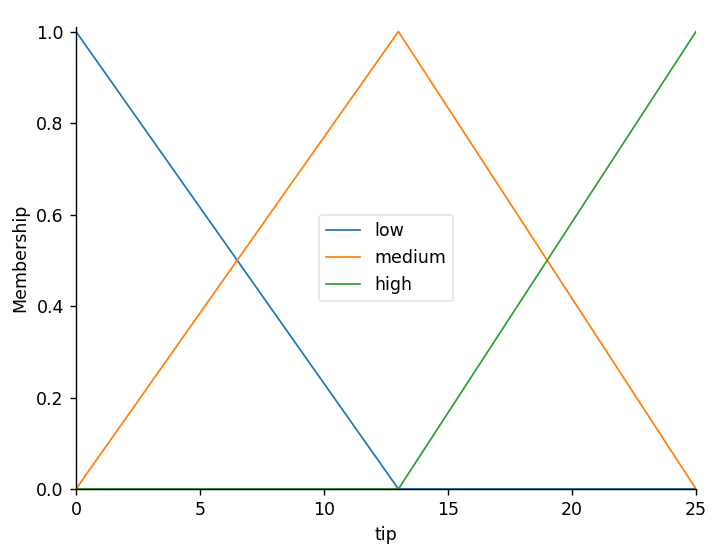
print(tipping.output['tip'])

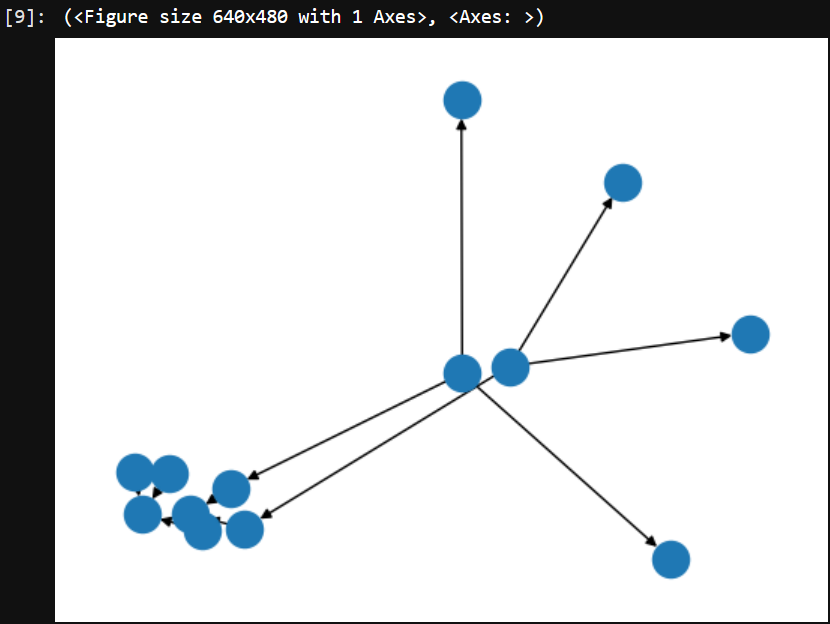
tip.view(sim=tipping)

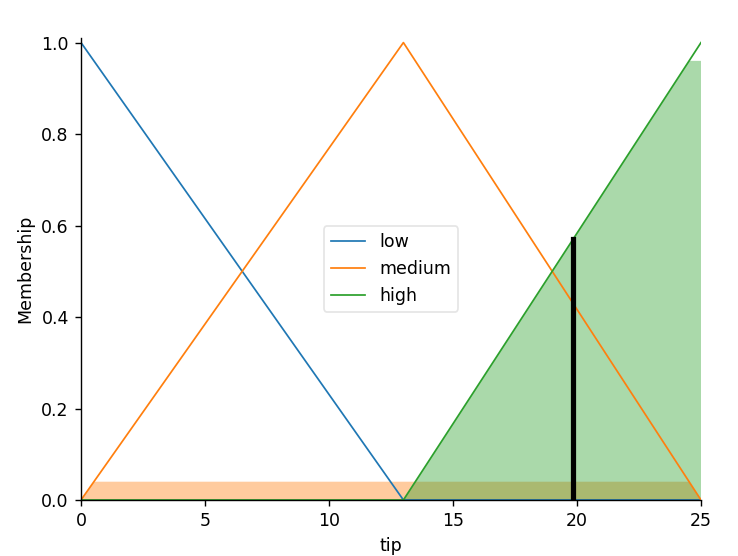
**Output:**

****









**Practical No: 7**

**Aim: Write an application to simulate supervised and un-supervised learning model.**

* **Supervised Learning.**

**Code**:

from sklearn import datasets

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.neighbors import KNeighborsClassifier

iris = datasets.load\_iris()

print(type(iris))

print(iris.keys())

type(iris.data), type(iris.target)

print(iris.data.shape)

print(iris.target\_names)

X = iris.data

# Load iris target set

Y = iris.target

# Convert datasets' type into dataframe

df = pd.DataFrame(X, columns=iris.feature\_names)

print(df)

knn = KNeighborsClassifier(n\_neighbors=6)

knn.fit(iris['data'], iris['target'])

X = [

[5.9, 1.0, 5.1, 1.8],

[3.4, 2.0, 1.1, 4.8],

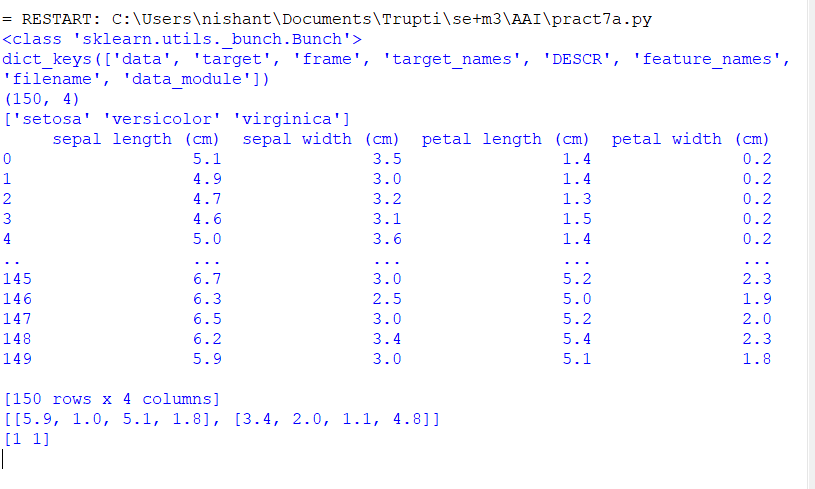
]

print(X)

prediction = knn.predict(X)

print(prediction)

**Output:**



**Code**:

from sklearn import datasets, linear\_model

import matplotlib.pyplot as plt

import numpy as np

diabetes = datasets.load\_diabetes()

diabetes

diabetes\_X = diabetes.data[:, np.newaxis, 2]

diabetes\_X\_train = diabetes\_X[:-20]

diabetes\_X\_test = diabetes\_X[-20:]

diabetes\_y\_train = diabetes.target[:-20]

diabetes\_y\_test = diabetes.target[-20:]

regr = linear\_model.LinearRegression()

regr.fit(diabetes\_X\_train, diabetes\_y\_train)

print('Input Values')

print(diabetes\_X\_test)

diabetes\_y\_pred = regr.predict(diabetes\_X\_test)

print("Predicted Output Values")

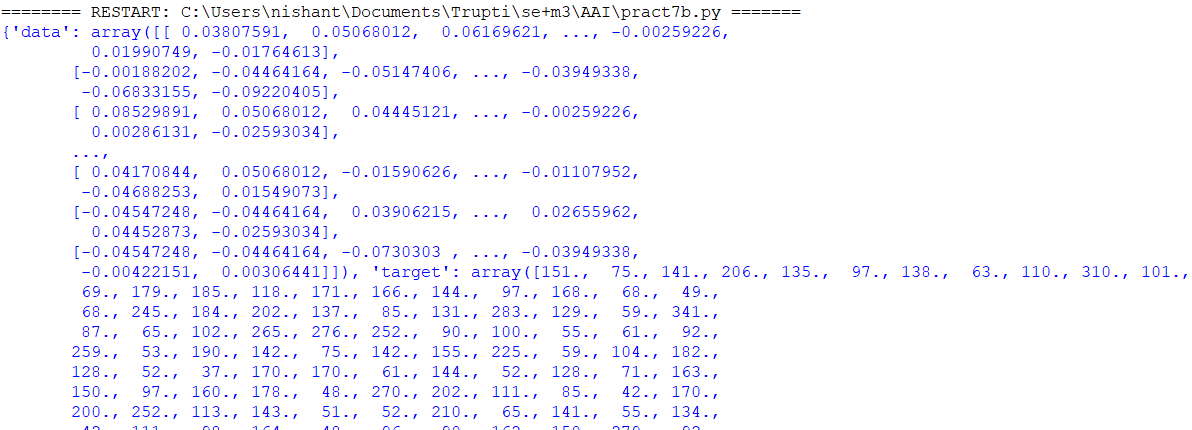
print(diabetes\_y\_pred)

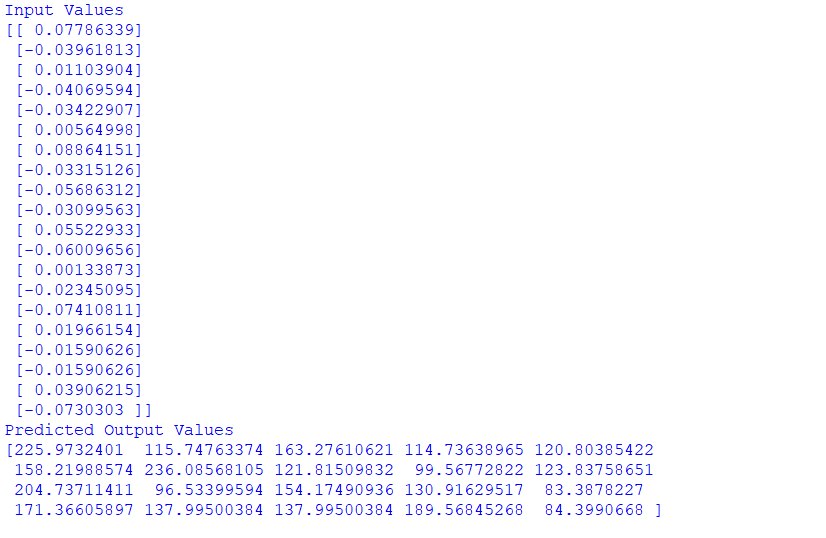
plt.scatter(diabetes\_X\_test, diabetes\_y\_test, color='black')

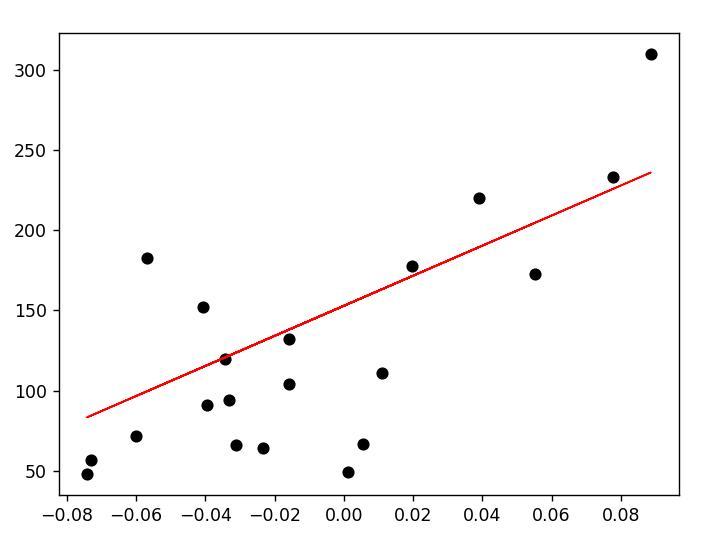
plt.plot(diabetes\_X\_test, diabetes\_y\_pred, color='red', linewidth=1)

plt.show()

**Output:**

****

****

****

* **Unsupervised Learning.**

**Code**:

from sklearn import datasets

from sklearn.cluster import KMeans

iris\_df = datasets.load\_iris()

model = KMeans(n\_clusters=3, n\_init = 10)

model.fit(iris\_df.data)

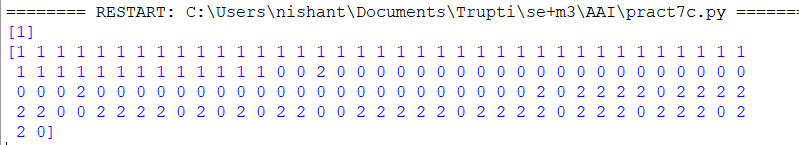
predicted\_label = model.predict([[7.2, 3.5, 0.8, 1.6]])

all\_predictions = model.predict(iris\_df.data)

print(predicted\_label)

print(all\_predictions)

**Output:**

****

**Practical No: 8**

**Aim: Write an application to implement clustering algorithm.**

**Code**:

from scipy.cluster.hierarchy import linkage, dendrogram

import matplotlib.pyplot as plt

import pandas as pd

seeds\_df = pd.read\_csv("seeds-less-rows.csv")

varieties = list(seeds\_df.pop('grain\_variety'))

samples = seeds\_df.values

mergings = linkage(samples, method='complete')

dendrogram(mergings,

labels=varieties,

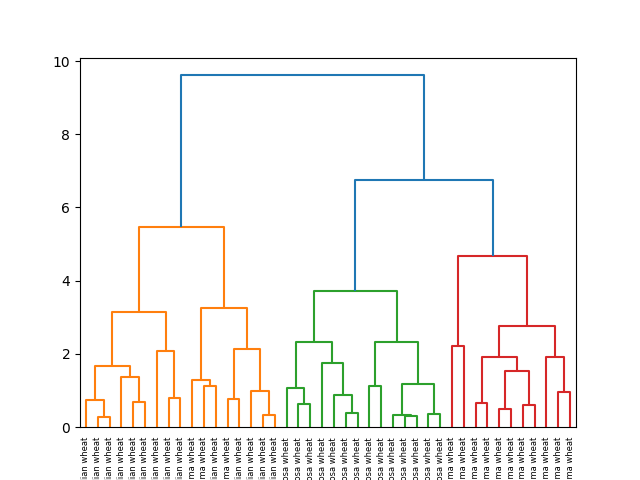
leaf\_rotation=90,

leaf\_font\_size=6,

)

plt.show()

**Output:**

****

**Practical No: 9**

**Aim: Write an application to implement support vector machine algorithm.**

**Code**:

import matplotlib.pyplot as plt

import numpy as np

from sklearn import svm

# linear data

X = np.array([1, 5, 1.5, 8, 1, 9, 7, 8.7, 2.3, 5.5, 7.7, 6.1])

y = np.array([2, 8, 1.8, 8, 0.6, 11, 10, 9.4, 4, 3, 8.8, 7.5])

# show unclassified data

plt.scatter(X, y)

plt.show()

# shaping data for training the model

training\_X = np.vstack((X, y)).T

print(np.vstack((X, y)).T)

training\_y = [0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1]

# define the model

clf = svm.SVC(kernel='linear', C=1.0)

# train the model

clf.fit(training\_X, training\_y)

# get the weight values for the linear equation from the trained SVM model

w = clf.coef\_[0]

# get the y-offset for the linear equation

a = -w[0] / w[1]

# make the x-axis space for the data points

XX = np.linspace(0, 13)

# get the y-values to plot the decision boundary

yy = a \* XX - clf.intercept\_[0] / w[1]

# plot the decision boundary

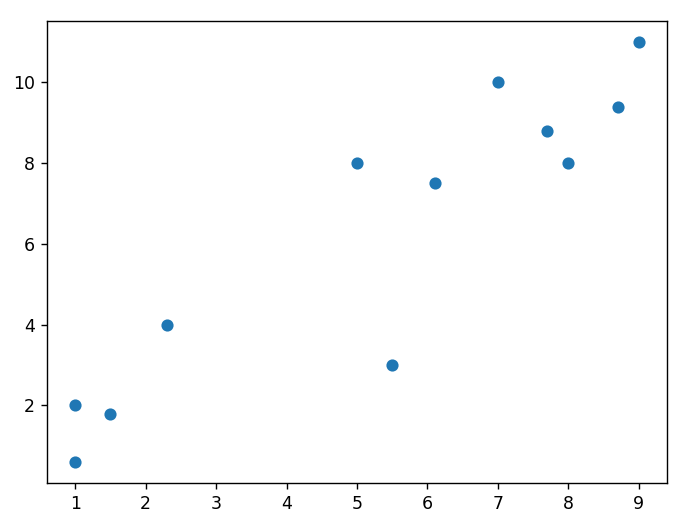
plt.plot(XX, yy, 'k-')

# show the plot visually

plt.scatter(training\_X[:, 0], training\_X[:, 1], c=training\_y)

plt.legend()

**Output:**



**Practical No: 10**

**Aim: Write a program to implement Backpropagation Algorithm.**

**Code**:

import numpy as np

import decimal

import math

print("Practical performed by 06 Sahil Dhawde")

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

np.set\_printoptions(precision=4)

v1=np.array([0.6,-0.3])

v2=np.array([-0.1,0.4])

w=np.array([-0.2,0.4,0.1])

b=np.array([0.3,0.5])

x=np.array([0,1])

alpha=0.25

zin=[]

print("Calculate net input to z layer")

for i in range(0,2):

zin.append(round(b[i]+x[0]\*v1[i]+x[1]\*v2[i],4))

print("zin= ",zin)

z=[]

print("Apply activation function to calculate output")

for i in range(0,2):

z.append(round(1/(1+math.exp(-zin[i])),4))

print("z= ",z)

print("Calculate net input to output layer")

yin=w[0]+z[0]\*w[1]+z[1]\*w[2]

print("yin=",yin)

print("Calculate net output")

y=round(1/(1+math.exp(-yin)),4)

print("y=",y)

fyin=y\*(1-y)

dk=round((1-y)\*fyin,4)

print("dk",dk)

dw1=alpha \* dk \* z[0]

dw2=alpha \* dk \* z[1]

dw0=alpha \* dk

print("Compute error portion in delta")

din=[]

for i in range(1,3):

din.append(dk\*w[i])

print("din",din)

fzin=[]

d=[]

print("Error in delta")

for i in range(0,2):

fzin.append(round(z[i]\*(1-z[i]),4))

d.append(round((din[i]\*fzin[i]),4))

print("fzin= ",fzin)

print("d= ",d)

dv=[[0,0],[0,0],[0,0]]

print("Changes in weights between input and hidden layer")

for i in range(0,3):

for j in range(0,2):

if i<=1:

dv[i][j]=alpha\*d[j]\*x[i]

else:

dv[i][j]=alpha\*d[j]

print(dv)

print("Final weights of networks")

for i in range(0,2):

v1[i]=v1[i]+dv[0][i]

print("v1= ",v1)

for i in range(0,2):

v2[i]=v2[i]+dv[1][i]

print("v2= ",v2)

for i in range(0,2):

b[i]=b[i]+dv[2][i]

print("b= ",b)

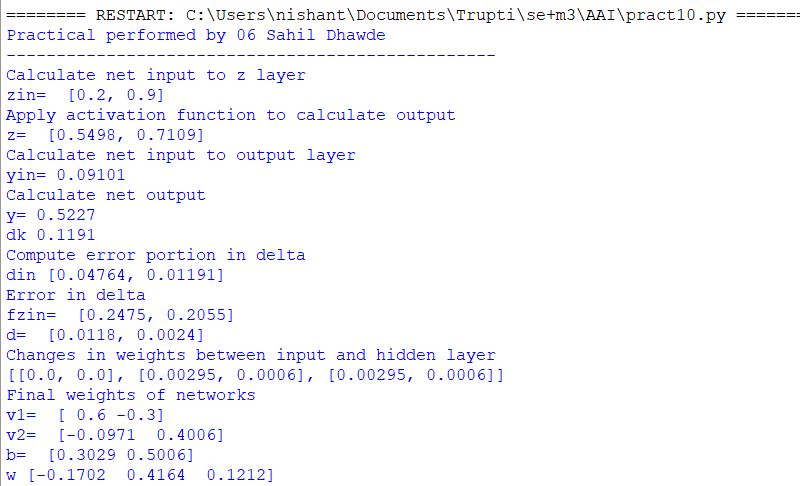
w[1]=w[1]+dw1

w[2]=w[2]+dw2

w[0]=w[0]+dw0

print("w",w)

**Output:**

****