University of Mumbai

PRACTICAL JOURNAL – ELECTIVE I

PSIT3P2a

Applied Artificial Intelligence

SUBMITTED BY

Karkera Prateek Ramesh

SEAT NO 30102

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR QUALIFYING
M.Sc. (I.T.) PART-II (SEMESTER – III) EXAMINATION

2022-2023

Department of Information Technology

3RD FLOOR, DR. SHANKAR DAYAL SHARMA BHAVAN, IDOL
BUILDING, VIDYANAGRI,
SANTACRUZ (E), MUMBAI – 400098.

University of Mumbai

Department of Information Technology

CERTIFICATE

This is to certify that Mr. **Karkera Prateek Ramesh Seat No. 30102** studying in **Master of Science** in **Information Technology Part II Semester III** has satisfactorily completed the Practical of **PSIT3P2a Applied Artificial Intelligence** as prescribed by University of Mumbai, during the academic year **2022-23**.

Signature	Signature	Signature
Guide	External Examiner Examined By	Head of the Department Certified by

College Seal

Date:

INDEX

PRACTICAL 1

AIM: Design a bot using AIML.

CODE:

Step 1: Create the XML file.

Open the notepad, write the following code, and save it as 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>
```

Step 2: Create the aiml file.

Open the notepad, write the following code, and save it as 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>
```

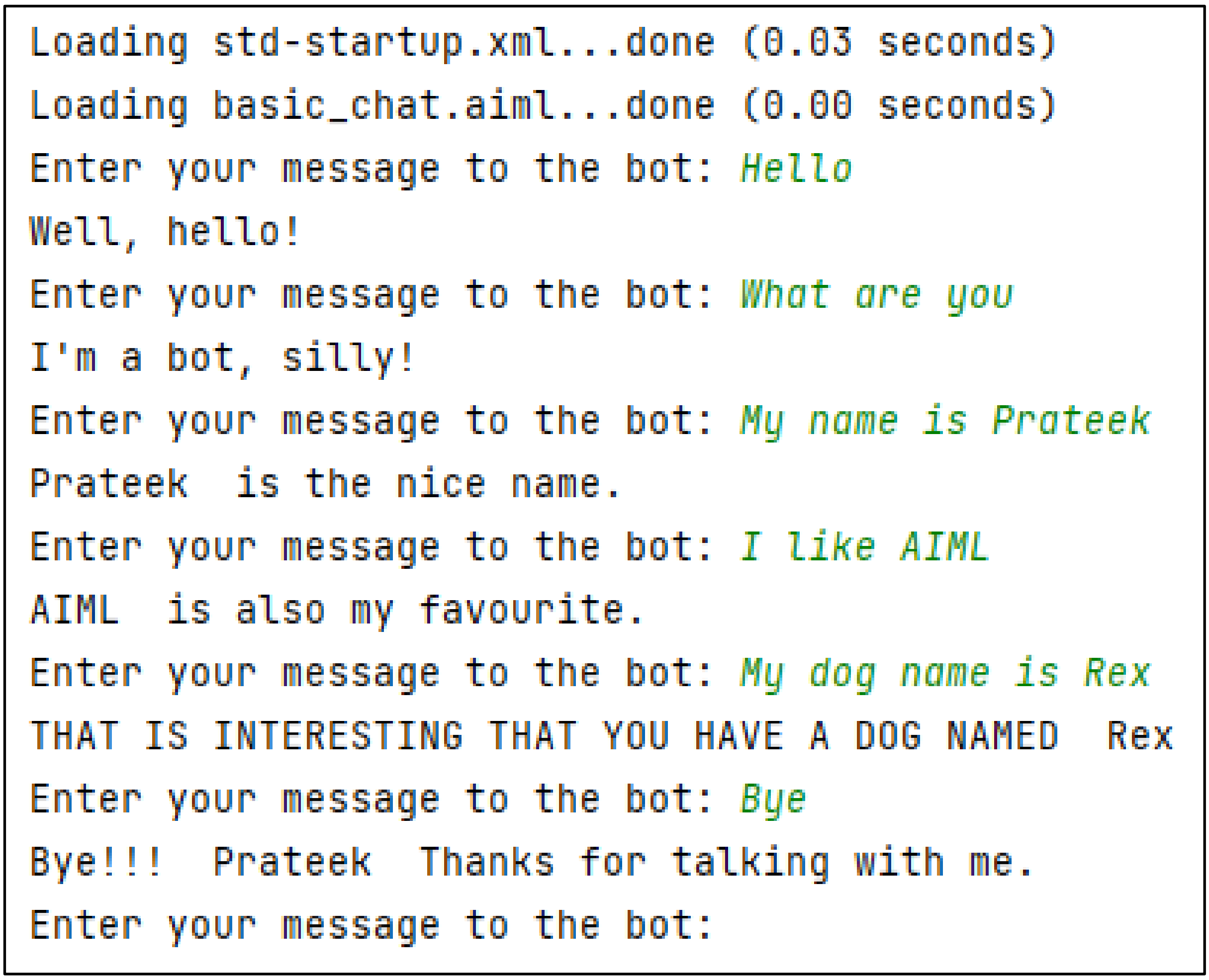
Step 3: Install aiml packages

```
pip install aiml
pip install aimlbotkernel
or
pip3 install aiml
pip3 install aimlbotkernel
```

Step 4: Create chatbot.py file

```
import aiml  # 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:



```
Loading std-startup.xml...done (0.03 seconds)
Loading basic_chat.aiml...done (0.00 seconds)
Enter your message to the bot: Hello
Well, hello!
Enter your message to the bot: What are you
I'm a bot, silly!
Enter your message to the bot: My name is Prateek
Prateek  is the nice name.
Enter your message to the bot: I like AIML
AIML  is also my favourite.
Enter your message to the bot: My dog name is Rex
THAT IS INTERESTING THAT YOU HAVE A DOG NAMED  Rex
Enter your message to the bot: Bye
Bye!!!  Prateek  Thanks for talking with me.
Enter your message to the bot:
```

PRACTICAL 2

AIM: Design an Expert system using AIML.

CODE:

Step 1: Create the XML file

Open the notepad, write the following code, and save it as 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>
```

Step 2: Create the aiml file

Open the notepad, write the following code, and save it as basic_chat.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>
```

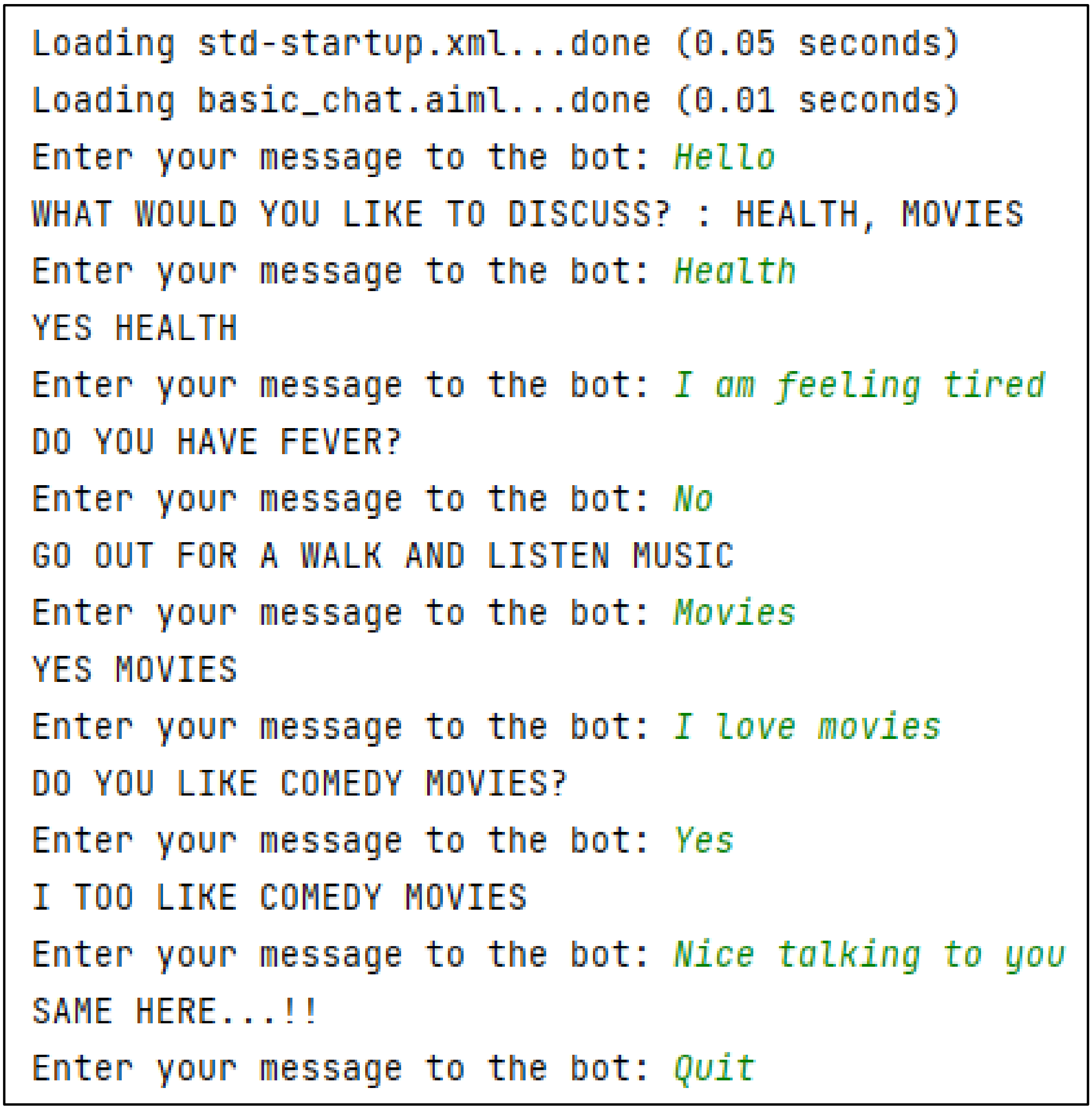
Step 3: Install aiml packages

```
pip install aiml
pip install aimlbotkernel
or
pip3 install aiml
pip3 install aimlbotkernel
```

Step 4: Create chatbot.py file and run chatbot.py

```
import aiml
# 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:



```
Loading std-startup.xml...done (0.05 seconds)
Loading basic_chat.aiml...done (0.01 seconds)
Enter your message to the bot: Hello
WHAT WOULD YOU LIKE TO DISCUSS? : HEALTH, MOVIES
Enter your message to the bot: Health
YES HEALTH
Enter your message to the bot: I am feeling tired
DO YOU HAVE FEVER?
Enter your message to the bot: No
GO OUT FOR A WALK AND LISTEN MUSIC
Enter your message to the bot: Movies
YES MOVIES
Enter your message to the bot: I love movies
DO YOU LIKE COMEDY MOVIES?
Enter your message to the bot: Yes
I TOO LIKE COMEDY MOVIES
Enter your message to the bot: Nice talking to you
SAME HERE...!!
Enter your message to the bot: Quit
```

PRACTICAL 3

AIM: Implement Bayes Theorem using Python.

CODE:

```
# calculate the probability of cancer patient and diagnostic test
# 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:

```
P(A|B) = 0.339%
```

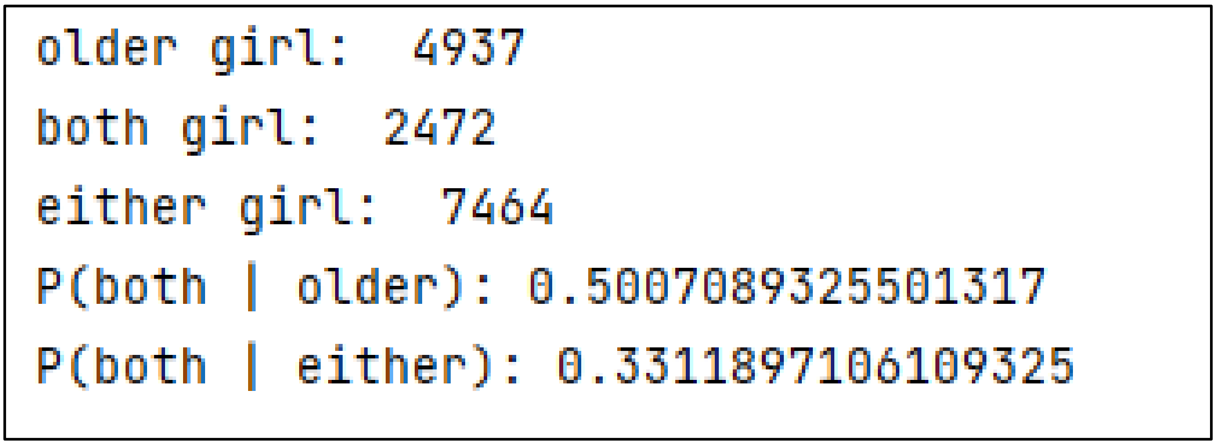
PRACTICAL 4

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("older girl: ", older_girl)
print("both girl: ", both_girls)
print("either girl: ", either_girl)
print("P(both | older):", both_girls / older_girl)

print("P(both | either):", both_girls / either_girl)
```

OUTPUT:



```
older girl:  4937
both girl:  2472
either girl:  7464
P(both | older): 0.5007089325501317
P(both | either): 0.3311897106109325
```

PRACTICAL 5

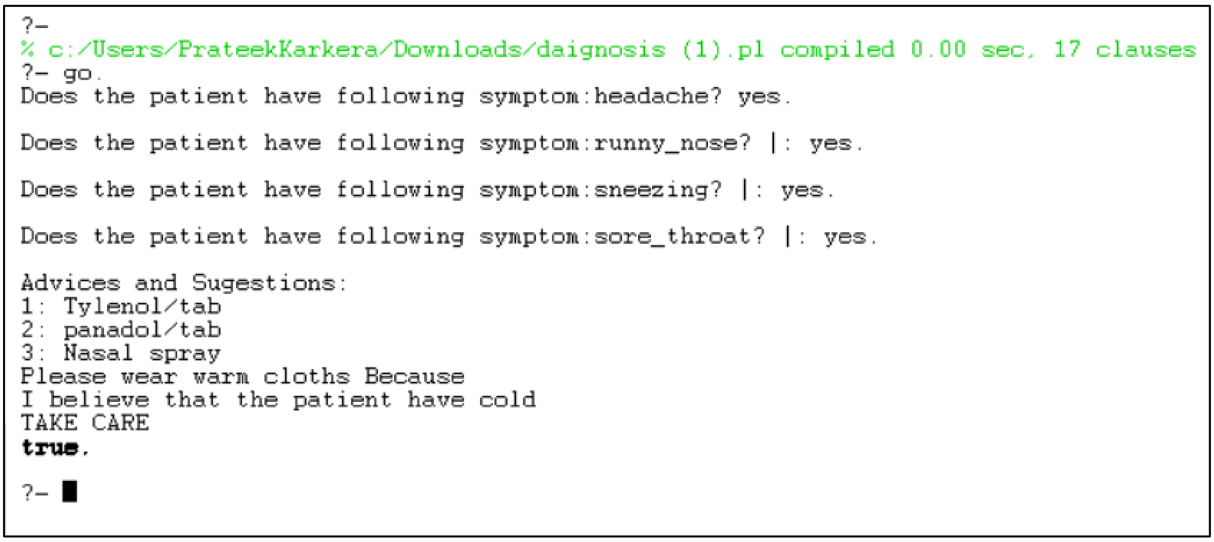
AIM: Write a program for to implement Rule based system. (Prolog).

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:



```
?-
% c:/Users/PrateekKarkera/Downloads/daignosis (1).pl compiled 0.00 sec, 17 clauses
?- go.
Does the patient have following symptom:headache? yes.

Does the patient have following symptom:runny_nose? |: yes.

Does the patient have following symptom:sneezing? |: yes.

Does the patient have following symptom:sore_throat? |: yes.

Advices and Sugestions:
1: Tylenol/tab
2: panadol/tab
3: Nasal spray
Please wear warm cloths Because
I believe that the patient have cold
TAKE CARE
true.

?- █
```

PRACTICAL 6

AIM: Design a Fuzzy based application using Python/ R.

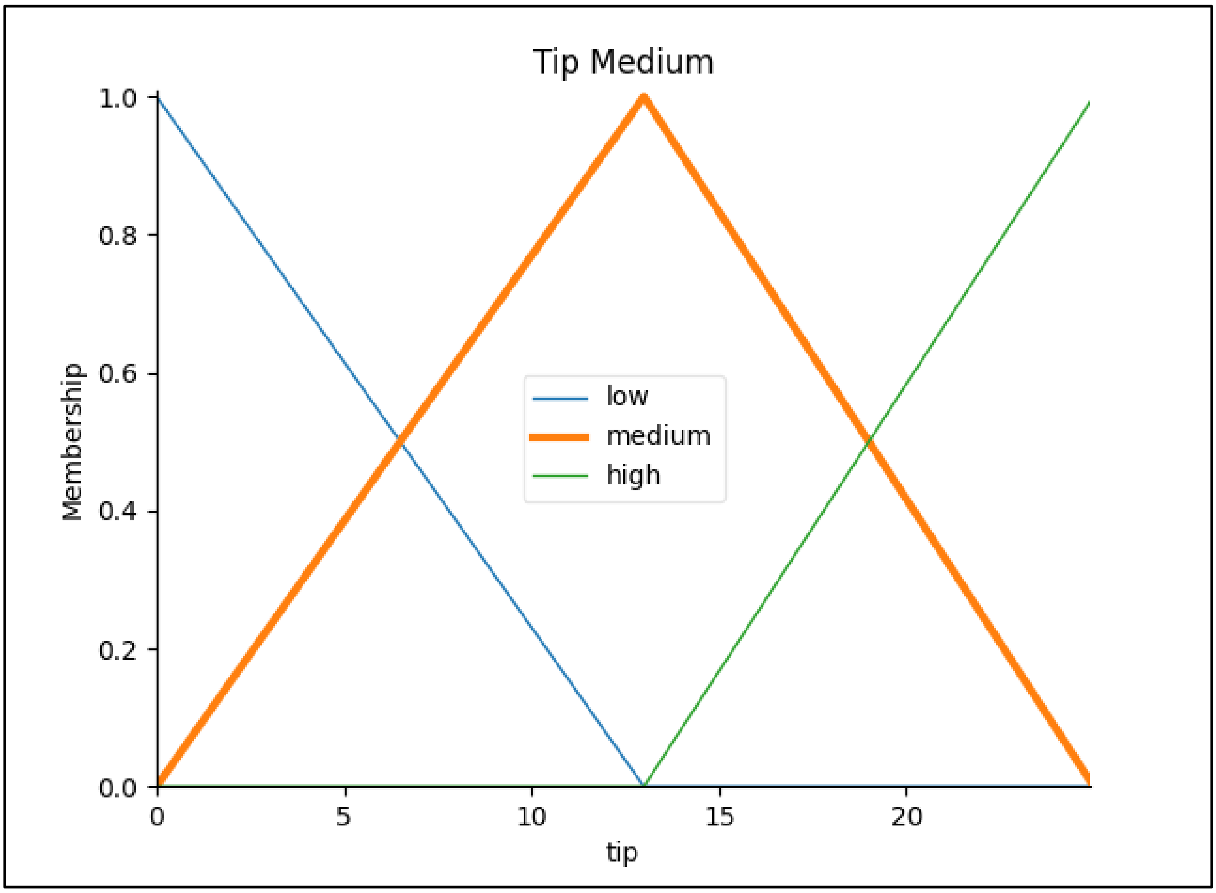
CODE:

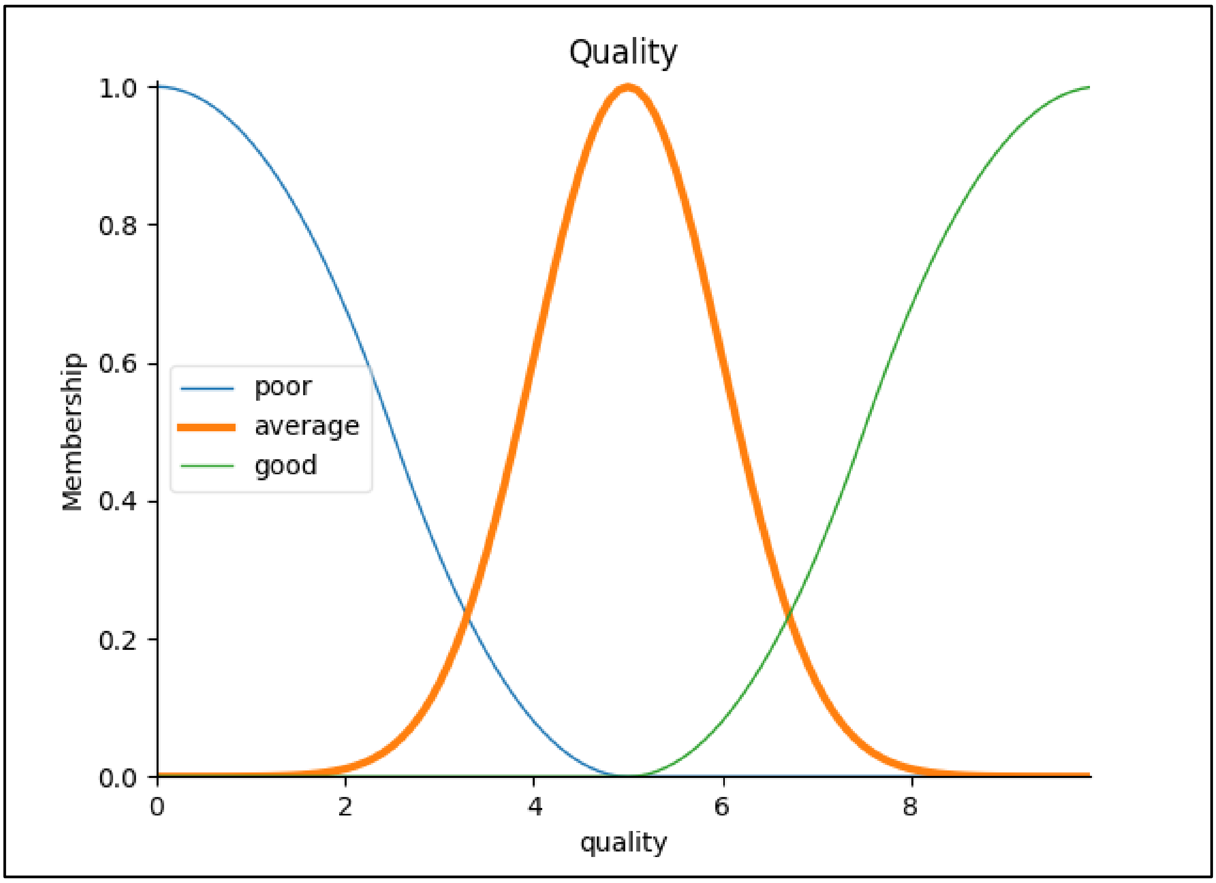
```
import numpy as np
import skfuzzy as fuzz
import matplotlib.pyplot as plt
from skfuzzy import control as ctrl
from mpl_toolkits.mplot3d import Axes3D  # Required for 3D plotting

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

quality = ctrl.Antecedent(np.arange(0, 10, 0.1), 'quality')
service = ctrl.Antecedent(np.arange(0, 10, 0.1), 'service')
tip = ctrl.Consequent(np.arange(0, 25, 0.1), 'tip')

quality['poor'] = fuzz.zmf(quality.universe, 0,5)
quality['average'] = fuzz.gaussmf(quality.universe,5,1)
quality['good'] = fuzz.smf(quality.universe,5,10)

service['poor'] = fuzz.zmf(service.universe, 0,5)
service['average'] = fuzz.gaussmf(service.universe,5,1)
service['good'] = fuzz.smf(service.universe,5,10)

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

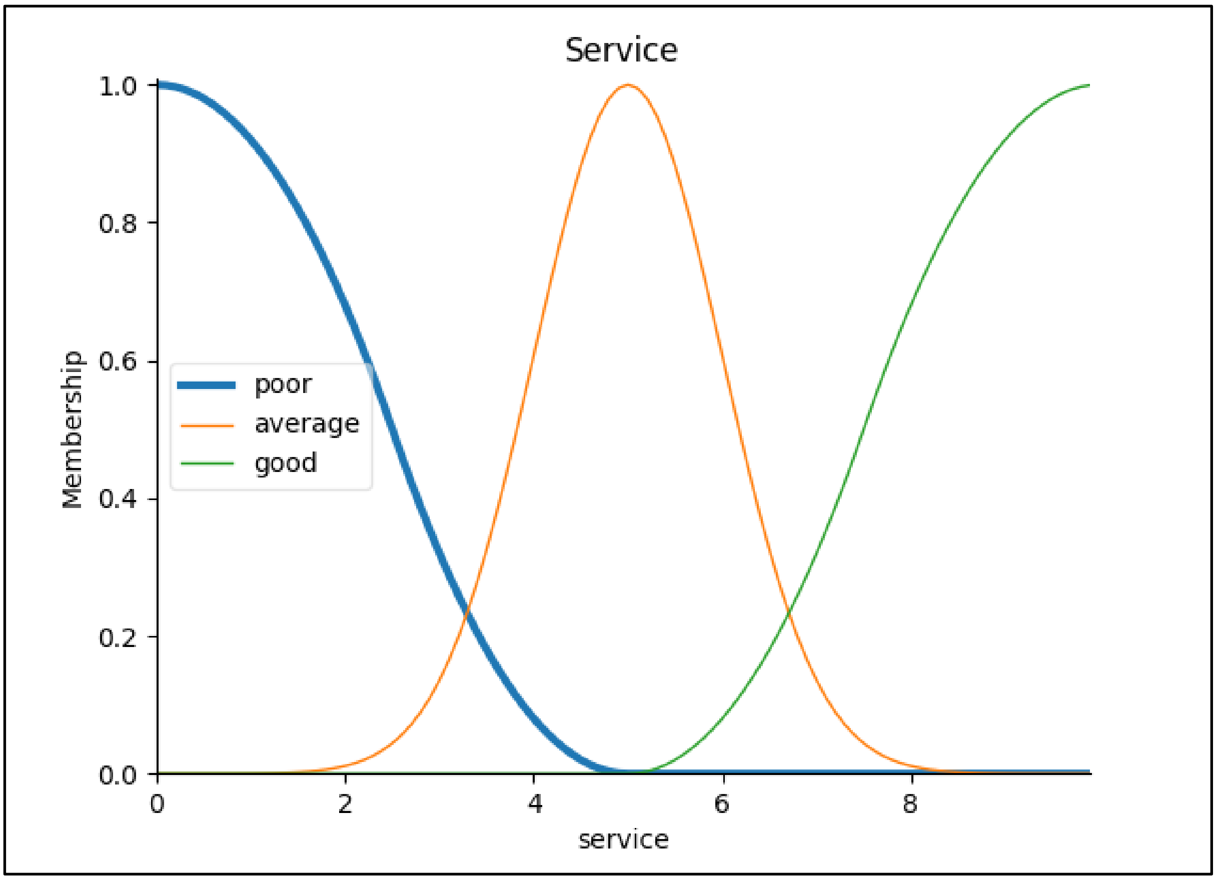
```
tip['medium'] = fuzz.trimf(tip.universe, [0, 13, 25])
tip['high'] = fuzz.trimf(tip.universe, [13, 25, 25])

quality['average'].view()
plt.title('Quality')

service['poor'].view()
plt.title('Service')

tip['medium'].view()
plt.title('Tip Medium')

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()
plt.title('Rule 1')
rule2.view()
plt.title('Rule 2')
rule3.view()
plt.title('Rule 3')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)
```

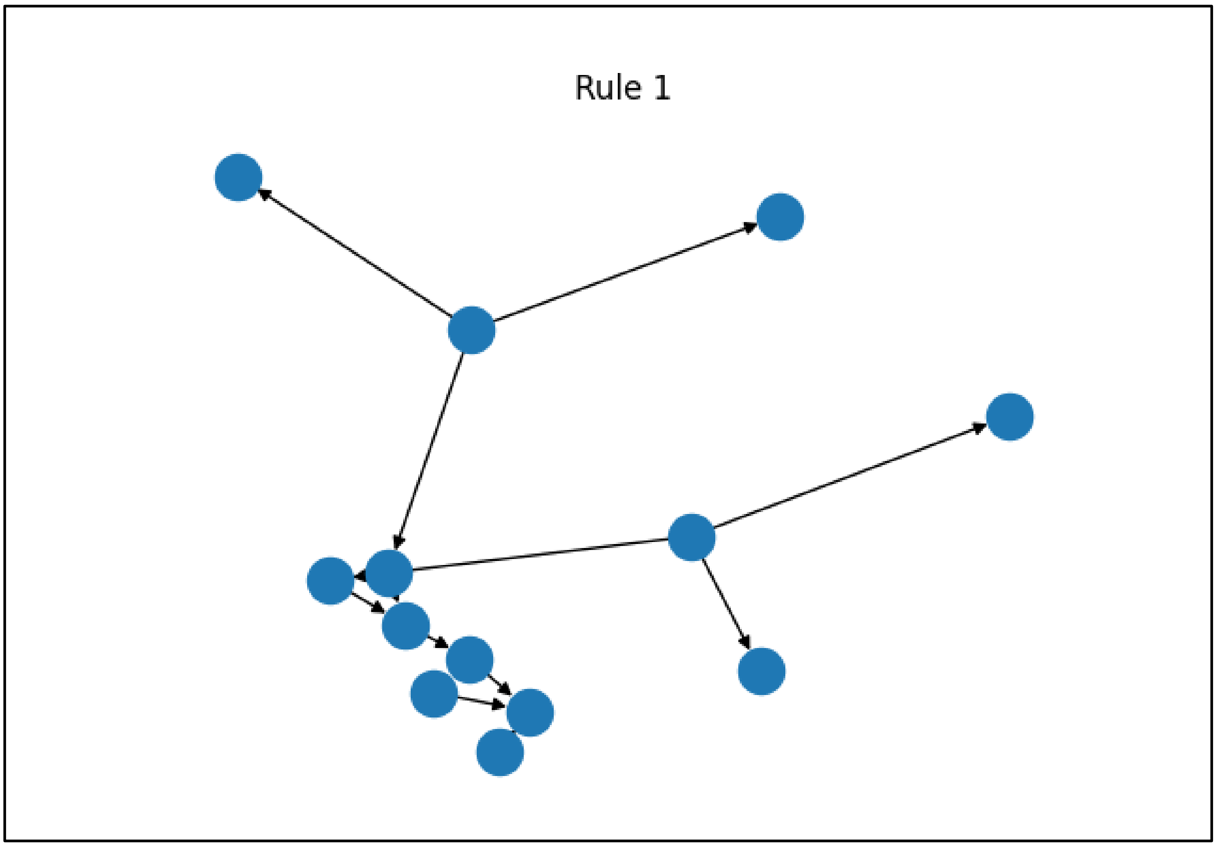
```
plt.title('Result')

plt.show(block=True)
```

OUTPUT:

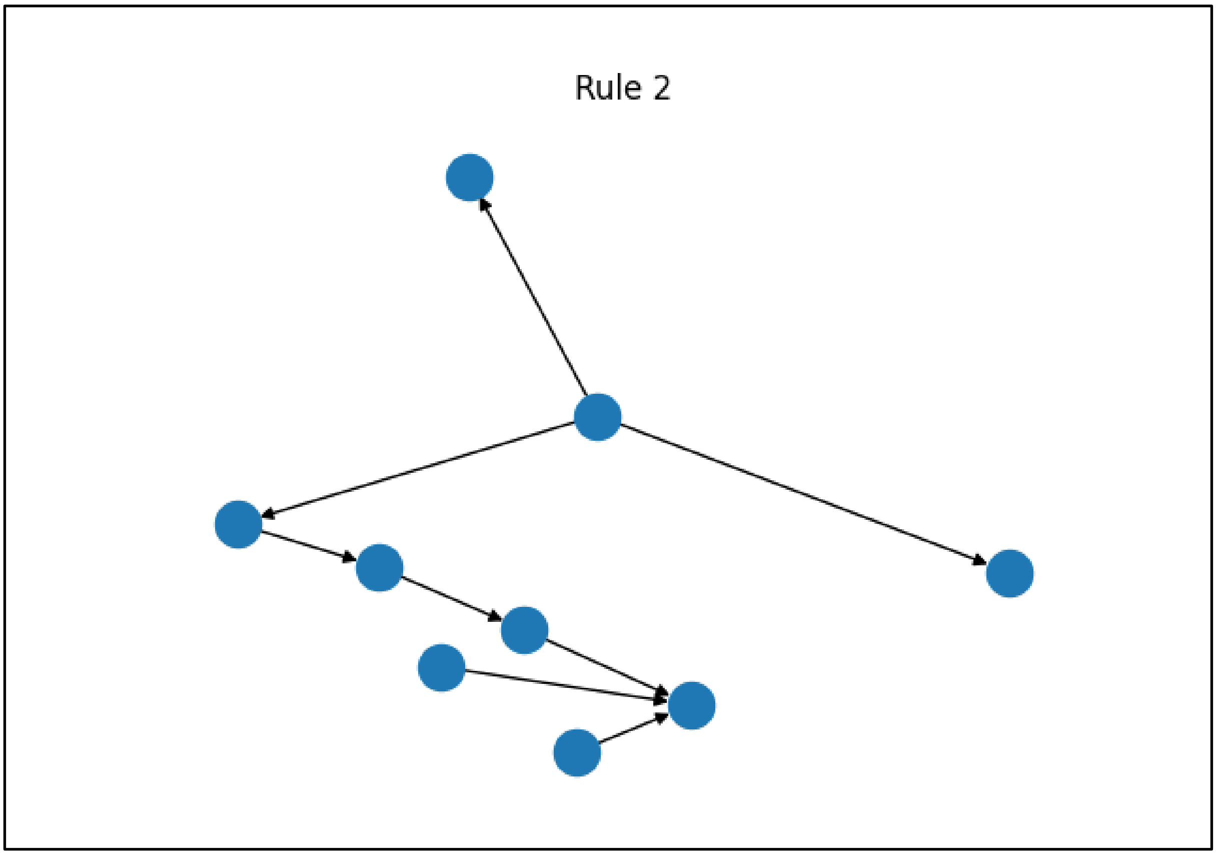



Quality
Membership
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8
quality
poor
average
good

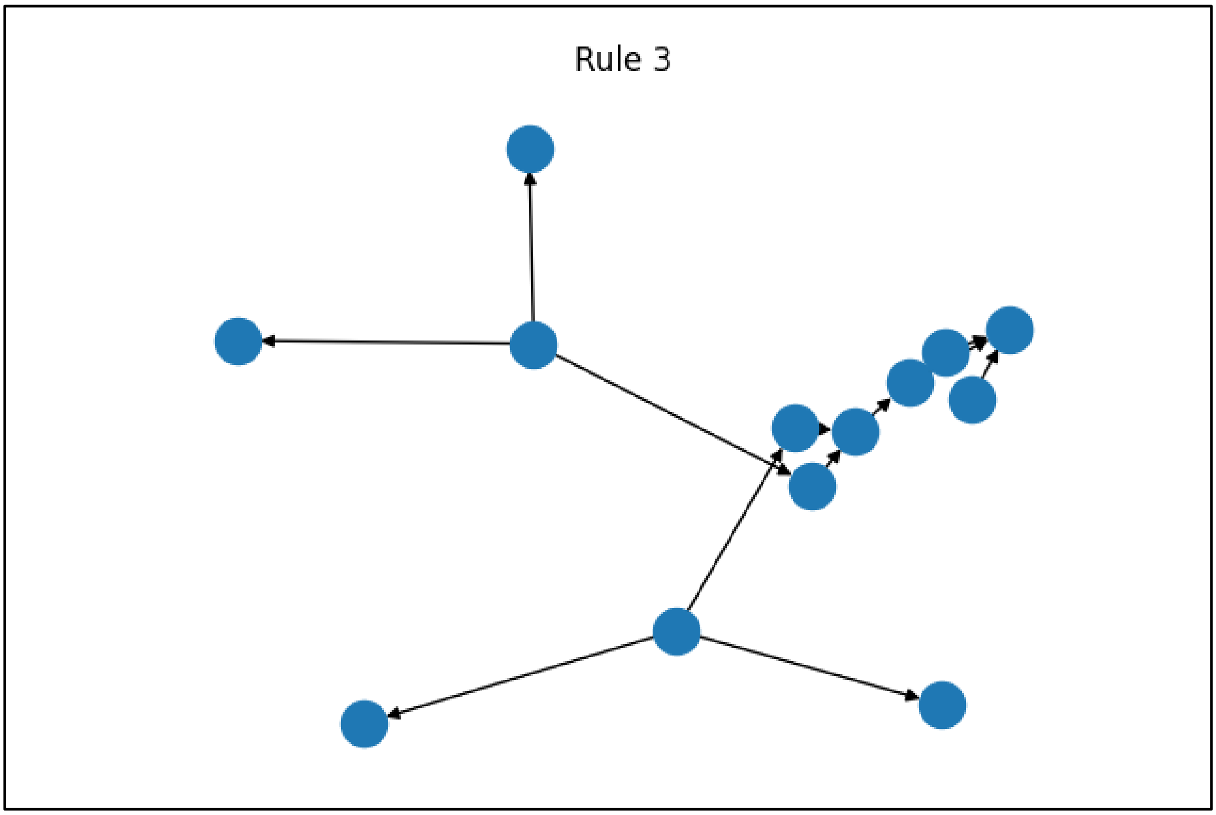

Service
Membership
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8
service
poor
average
good

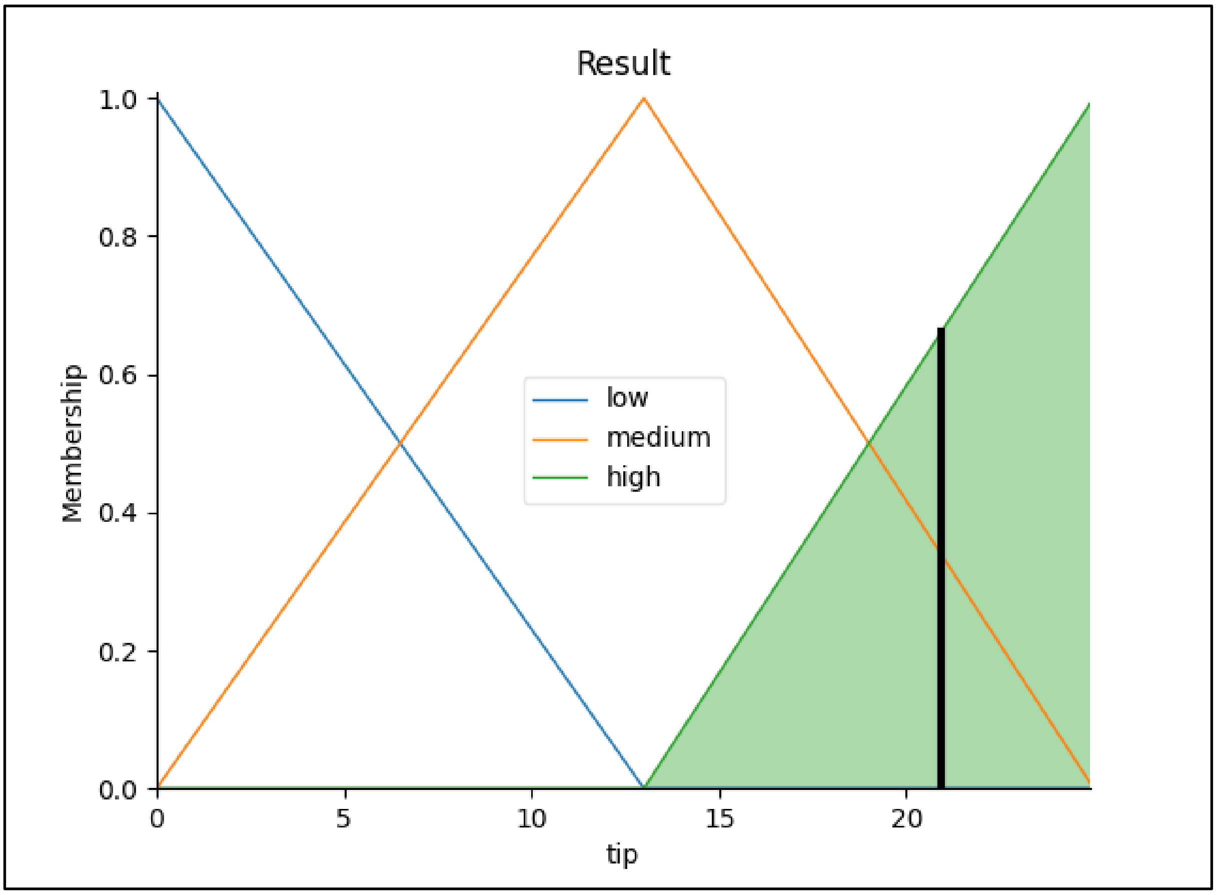


Rule 1



Rule 2


Rule 3


Result
1.0
0.8
0.6
0.4
0.2
0.0
Membership
0
5
10
15
20
tip
low
medium
high

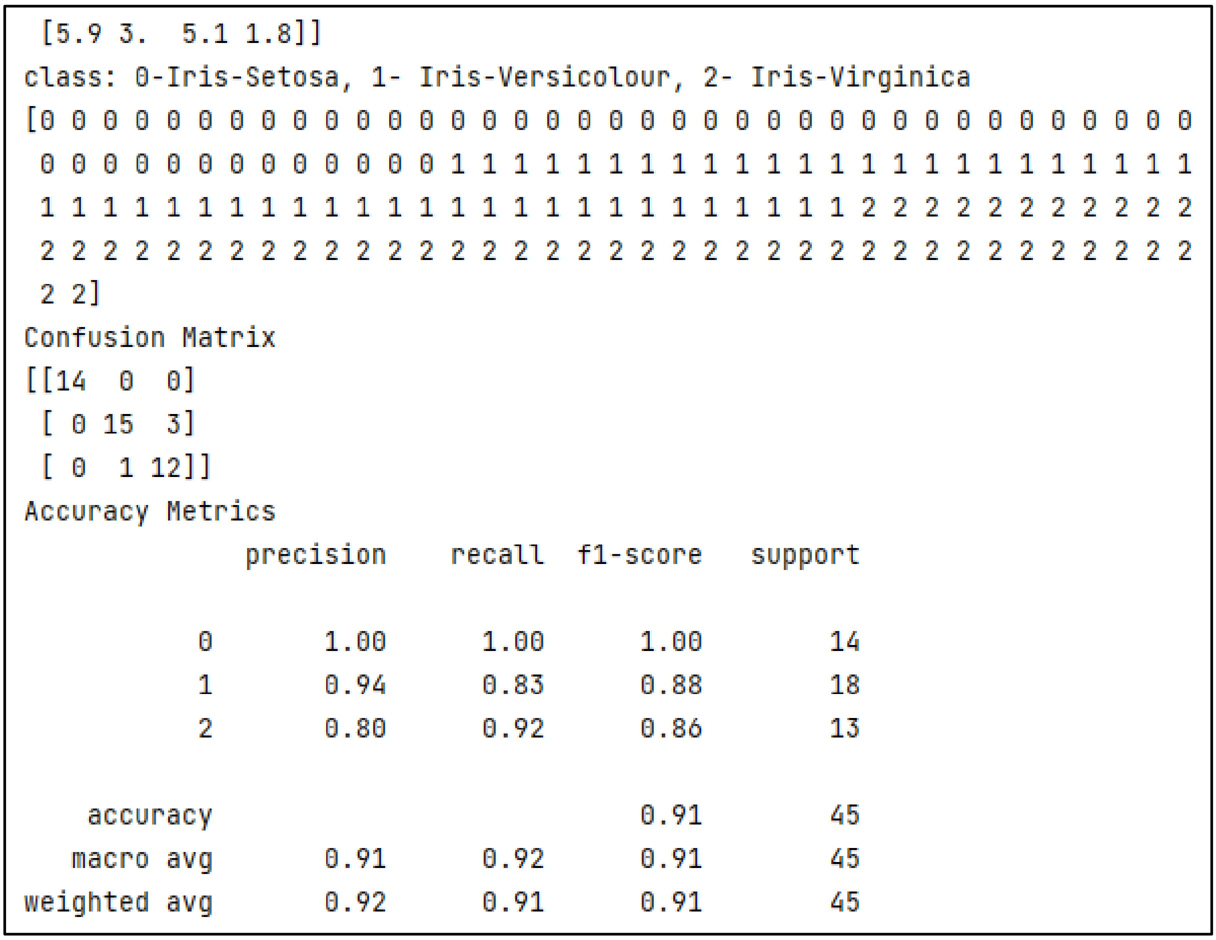
PRACTICAL 7

[A] AIM: Write an application to stimulate supervised learning model.

CODE:

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn import datasets
iris=datasets.load_iris()
x = iris.data
y = iris.target
print ('sepal-length', 'sepal-width', 'petal-length', 'petal-width')
print(x)
print('class: 0-Iris-Setosa, 1- Iris-Versicolour, 2- Iris-Virginica')
print(y)
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.3)
#To Training the model and Nearest nighbors K=5
classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(x_train, y_train)
#To make predictions on our test data
y_pred=classifier.predict(x_test)
print('Confusion Matrix')
print(confusion_matrix(y_test,y_pred))
print('Accuracy Metrics')
print(classification_report(y_test,y_pred))
```

OUTPUT:



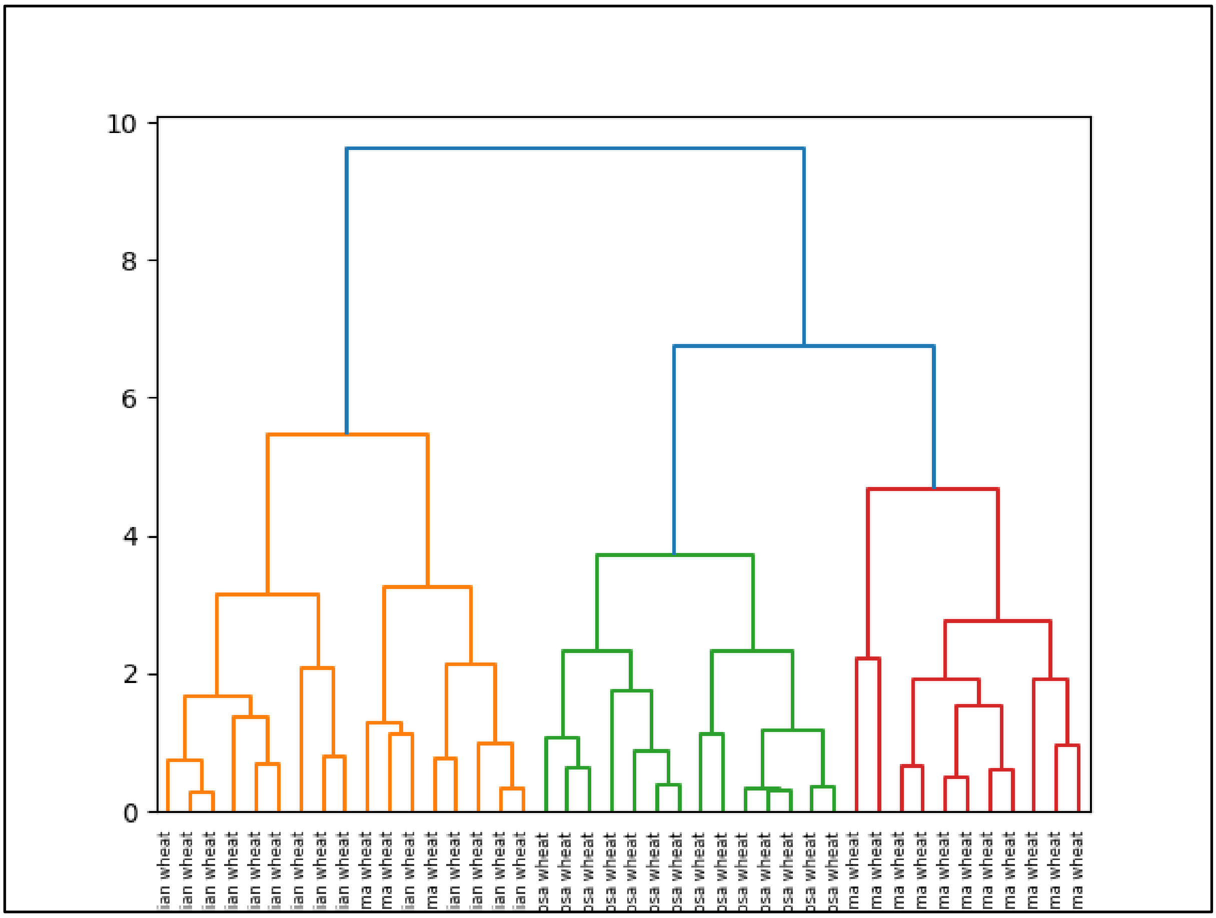
```
 [5.9 3.  5.1 1.8]]
class: 0-Iris-Setosa, 1- Iris-Versicolour, 2- Iris-Virginica
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 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]
Confusion Matrix
[[14  0  0]
 [ 0 15  3]
 [ 0  1 12]]
Accuracy Metrics
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        14
           1       0.94      0.83      0.88        18
           2       0.80      0.92      0.86        13

    accuracy                           0.91        45
   macro avg       0.91      0.92      0.91        45
weighted avg       0.92      0.91      0.91        45
```

[B] AIM: Write an application to stimulate unsupervised learning model.

CODE:

```
# Importing Modules
from scipy.cluster.hierarchy import linkage, dendrogram
import matplotlib.pyplot as plt
import pandas as pd

# Reading the DataFrame
seeds_df = pd.read_csv("seeds-less-rows.csv")

# Remove the grain species from the DataFrame, save for later
varieties = list(seeds_df.pop('grain_variety'))

# Extract the measurements as a NumPy array
samples = seeds_df.values

"""
Perform hierarchical clustering on samples using the
linkage() function with the method='complete' keyword argument.
Assign the result to mergings.
"""
mergings = linkage(samples, method='complete')

"""
Plot a dendrogram using the dendrogram() function on mergings,
specifying the keyword arguments labels=varieties, leaf_rotation=90,
and leaf_font_size=6.
"""
dendrogram(mergings,
           labels=varieties,
           leaf_rotation=90,
           leaf_font_size=6,
            )


plt.show()
```

OUTPUT:



PRACTICAL 8

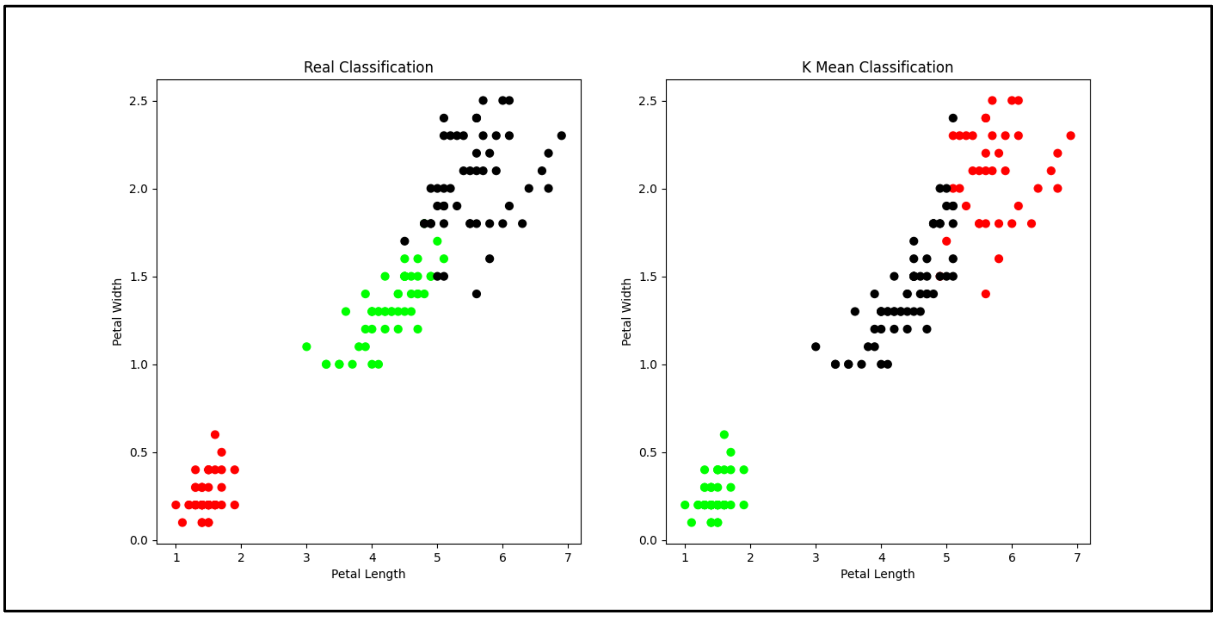
AIM: Write an application to implement clustering algorithm.

CODE:

```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans
import sklearn.metrics as sm
import pandas as pd
import numpy as np
iris = datasets.load_iris()
X = pd.DataFrame(iris.data)
X.columns = ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']
y = pd.DataFrame(iris.target)
y.columns = ['Targets']
model = KMeans(n_clusters=3)
model.fit(X)
plt.figure(figsize=(14,7))
colormap = np.array(['red', 'lime', 'black'])
# Plot the Original Classifications
plt.subplot(1, 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(1, 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')

plt.show()

print('The accuracy score of K-Mean: ',sm.accuracy_score(y, model.labels_))

print('The    Confusion    matrix    of    K-Mean:    ',sm.confusion_matrix(y,
model.labels_))
```

OUTPUT:



PRACTICAL 9

AIM: Write an application to implement support vector machine algorithm.

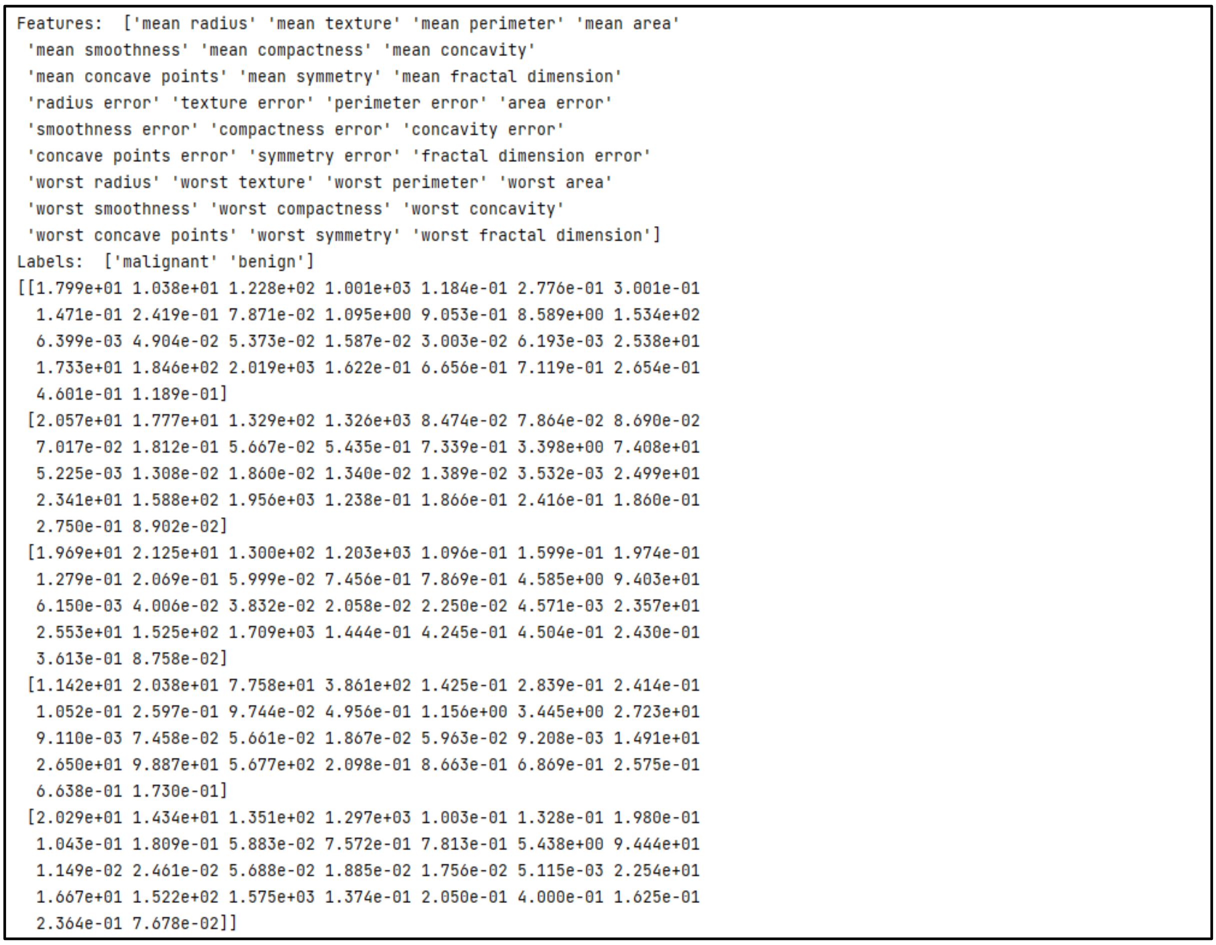
CODE:

```
#Import scikit-learn dataset library
from sklearn import datasets

#Import svm model
from sklearn import svm

# Import train_test_split function
from sklearn.model_selection import train_test_split

#Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics

#Load dataset
cancer = datasets.load_breast_cancer()

# print the names of the 13 features
print("Features: ", cancer.feature_names)

# print the label type of cancer('malignant' 'benign')
print("Labels: ", cancer.target_names)

# print data(feature)shape
```

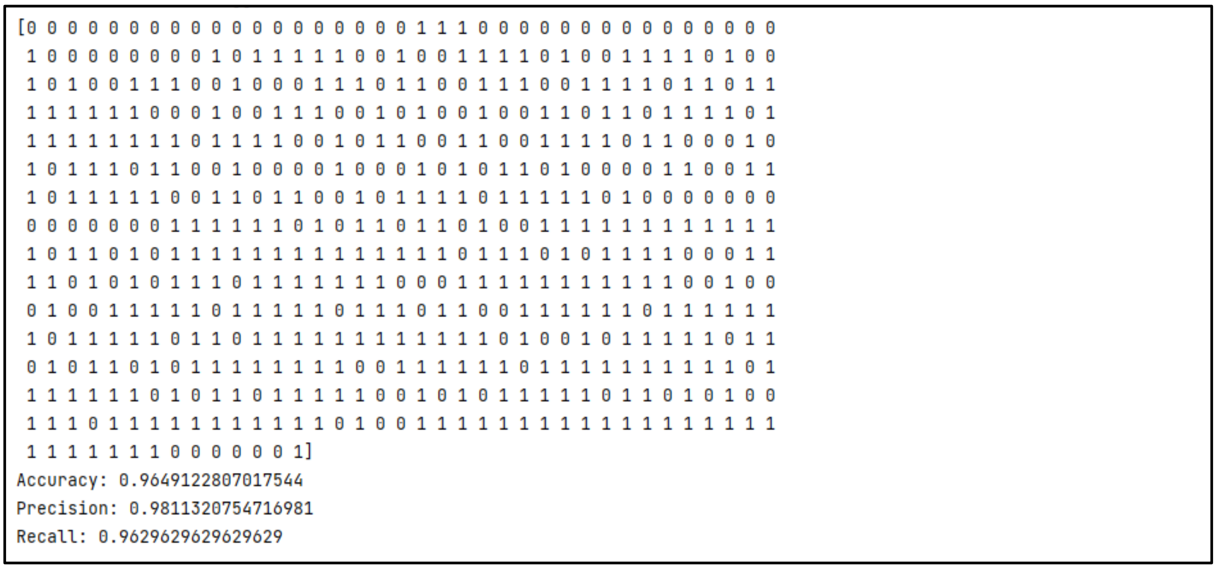
```
cancer.data.shape


# print the cancer data features (top 5 records)

print(cancer.data[0:5])


# print the cancer labels (0:malignant, 1:benign)

print(cancer.target)


# Split dataset into training set and test set

X_train, X_test, y_train, y_test = train_test_split(cancer.data,
cancer.target, test_size=0.3,random_state=109) # 70% training and 30% test


#Create a svm Classifier

clf = svm.SVC(kernel='linear') # Linear Kernel


#Train the model using the training sets

clf.fit(X_train, y_train)


#Predict the response for test dataset

y_pred = clf.predict(X_test)




# Model Accuracy: how often is the classifier correct?

print("Accuracy:",metrics.accuracy_score(y_test, y_pred))


# Model Precision: what percentage of positive tuples are labeled as such?

print("Precision:",metrics.precision_score(y_test, y_pred))
```

```
# Model Recall: what percentage of positive tuples are labelled as such?

print("Recall:",metrics.recall_score(y_test, y_pred))
```

OUTPUT:



```
Features:  ['mean radius' 'mean texture' 'mean perimeter' 'mean area'
 'mean smoothness' 'mean compactness' 'mean concavity'
 'mean concave points' 'mean symmetry' 'mean fractal dimension'
 'radius error' 'texture error' 'perimeter error' 'area error'
 'smoothness error' 'compactness error' 'concavity error'
 'concave points error' 'symmetry error' 'fractal dimension error'
 'worst radius' 'worst texture' 'worst perimeter' 'worst area'
 'worst smoothness' 'worst compactness' 'worst concavity'
 'worst concave points' 'worst symmetry' 'worst fractal dimension']
Labels:  ['malignant' 'benign']
[[1.799e+01 1.038e+01 1.228e+02 1.001e+03 1.184e-01 2.776e-01 3.001e-01
  1.471e-01 2.419e-01 7.871e-02 1.095e+00 9.053e-01 8.589e+00 1.534e+02
  6.399e-03 4.904e-02 5.373e-02 1.587e-02 3.003e-02 6.193e-03 2.538e+01
  1.733e+01 1.846e+02 2.019e+03 1.622e-01 6.656e-01 7.119e-01 2.654e-01
  4.601e-01 1.189e-01]
 [2.057e+01 1.777e+01 1.329e+02 1.326e+03 8.474e-02 7.864e-02 8.690e-02
  7.017e-02 1.812e-01 5.667e-02 5.435e-01 7.339e-01 3.398e+00 7.408e+01
  5.225e-03 1.308e-02 1.860e-02 1.340e-02 1.389e-02 3.532e-03 2.499e+01
  2.341e+01 1.588e+02 1.956e+03 1.238e-01 1.866e-01 2.416e-01 1.860e-01
  2.750e-01 8.902e-02]
 [1.969e+01 2.125e+01 1.300e+02 1.203e+03 1.096e-01 1.599e-01 1.974e-01
  1.279e-01 2.069e-01 5.999e-02 7.456e-01 7.869e-01 4.585e+00 9.403e+01
  6.150e-03 4.006e-02 3.832e-02 2.058e-02 2.250e-02 4.571e-03 2.357e+01
  2.553e+01 1.525e+02 1.709e+03 1.444e-01 4.245e-01 4.504e-01 2.430e-01
  3.613e-01 8.758e-02]
 [1.142e+01 2.038e+01 7.758e+01 3.861e+02 1.425e-01 2.839e-01 2.414e-01
  1.052e-01 2.597e-01 9.744e-02 4.956e-01 1.156e+00 3.445e+00 2.723e+01
  9.110e-03 7.458e-02 5.661e-02 1.867e-02 5.963e-02 9.208e-03 1.491e+01
  2.650e+01 9.887e+01 5.677e+02 2.098e-01 8.663e-01 6.869e-01 2.575e-01
  6.638e-01 1.730e-01]
 [2.029e+01 1.434e+01 1.351e+02 1.297e+03 1.003e-01 1.328e-01 1.980e-01
  1.043e-01 1.809e-01 5.883e-02 7.572e-01 7.813e-01 5.438e+00 9.444e+01
  1.149e-02 2.461e-02 5.688e-02 1.885e-02 1.756e-02 5.115e-03 2.254e+01
  1.667e+01 1.522e+02 1.575e+03 1.374e-01 2.050e-01 4.000e-01 1.625e-01
  2.364e-01 7.678e-02]]
```



```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 1 0 0 0 0 0 0 0 0 1 0 1 1 1 1 1 0 0 1 0 0 1 1 1 1 0 1 0 0 1 1 1 1 0 1 0 0
 1 0 1 0 0 1 1 1 0 0 1 0 0 0 1 1 1 0 1 1 0 0 1 1 1 0 0 1 1 1 1 0 1 1 0 1 1
 1 1 1 1 1 1 0 0 0 1 0 0 1 1 1 0 0 1 0 1 0 0 1 0 0 1 1 0 1 1 0 1 1 1 1 0 1
 1 1 1 1 1 1 1 1 0 1 1 1 1 0 0 1 0 1 1 0 0 1 1 0 0 1 1 1 1 0 1 1 0 0 0 1 0
 1 0 1 1 1 0 1 1 0 0 1 0 0 0 0 1 0 0 0 1 0 1 0 1 1 0 1 0 0 0 0 1 1 0 0 1 1
 1 0 1 1 1 1 1 0 0 1 1 0 1 1 0 0 1 0 1 1 1 1 0 1 1 1 1 1 0 1 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 1 1 1 1 1 1 0 1 0 1 1 0 1 1 0 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1
 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 0 1 1 1 1 0 0 0 1 1
 1 1 0 1 0 1 0 1 1 1 0 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0 0
 0 1 0 0 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 0 1 1 0 0 1 1 1 1 1 1 0 1 1 1 1 1 1
 1 0 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0 1 0 1 1 1 1 1 0 1 1
 0 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 0 1
 1 1 1 1 1 1 0 1 0 1 1 0 1 1 1 1 1 0 0 1 0 1 0 1 1 1 1 1 0 1 1 0 1 0 1 0 0
 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 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 0 0 0 0 0 0 1]
Accuracy: 0.9649122807017544
Precision: 0.9811320754716981
Recall: 0.9629629629629629
```

PRACTICAL 10

AIM: Simulate artificial neural network model with both feedforward and backpropagation approach.

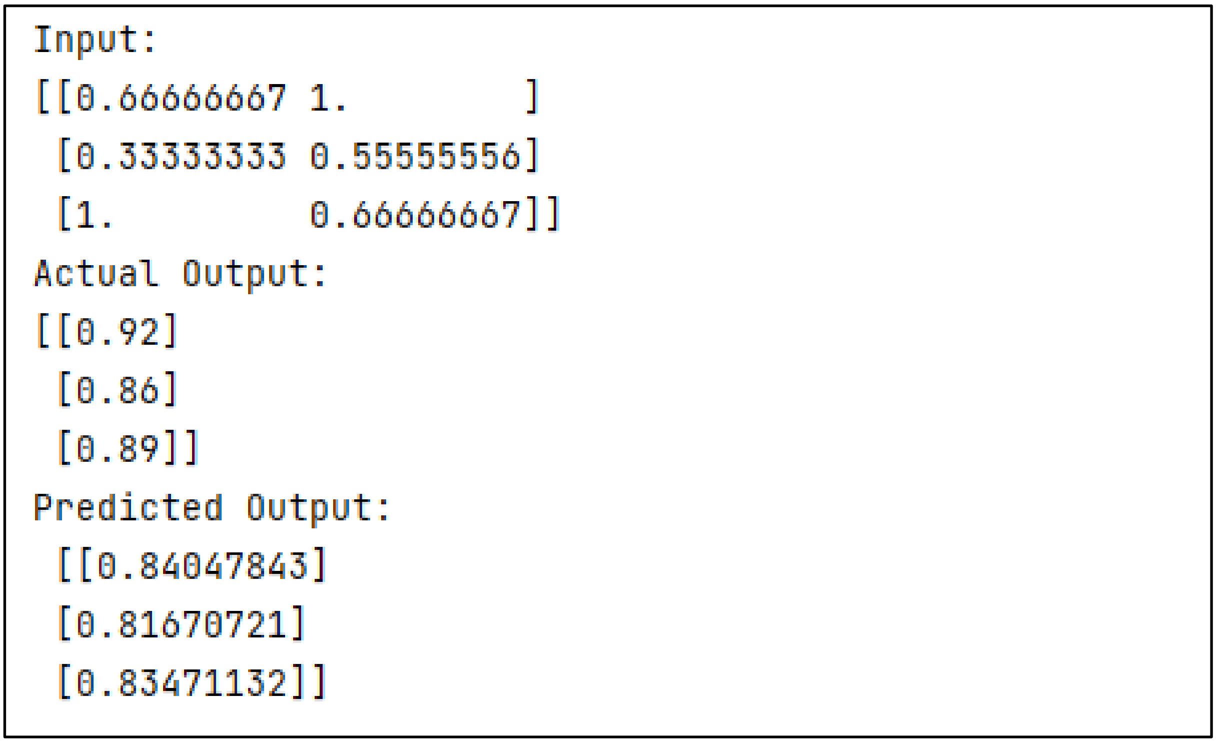
CODE:

```
import numpy as np
X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)  # two inputs [sleep,study]
y = np.array(([92], [86], [89]), dtype=float)  # one output [Expected % in Exams]
X = X / np.amax(X, axis=0)  # maximum of X array longitudinally
y = y / 100


# Sigmoid Function
def sigmoid(x):
    return 1 / (1 + np.exp(-x))


# Derivative of Sigmoid Function
def derivatives_sigmoid(x):
    return x * (1 - x)


# Variable initialization
epoch = 5000  # Setting training iterations
lr = 0.1  # Setting learning rate
inputlayer_neurons = 2  # number of features in data set
hiddenlayer_neurons = 3  # number of hidden layers neurons
output_neurons = 1  # number of neurons at output layer
```

```
# weight and bias initialization
wh = np.random.uniform(size=(inputlayer_neurons, hiddenlayer_neurons))   # weight of the link from input node to hidden node
bh = np.random.uniform(size=(1, hiddenlayer_neurons))   # bias of the link from input node to hidden node
wout = np.random.uniform(size=(hiddenlayer_neurons, output_neurons))   # weight of the link from hidden node to output node
bout = np.random.uniform(size=(1, output_neurons))  # bias of the link from hidden node to output node
# draws a random range of numbers uniformly of dim x*y
for i in range(epoch):
    # Forward Propogation
    hinp1 = np.dot(X, wh)
    hinp = hinp1 + bh
    hlayer_act = sigmoid(hinp)
    outinp1 = np.dot(hlayer_act, wout)
    outinp = outinp1 + bout
    output = sigmoid(outinp)
    # Backpropagation
    EO = y - output
    outgrad = derivatives_sigmoid(output)
    d_output = EO * outgrad
    EH = d_output.dot(wout.T)
    # how much hidden layer weights contributed to error
    hiddengrad = derivatives_sigmoid(hlayer_act)
    d_hiddenlayer = EH * hiddengrad
# dotproduct of nextlayererror and currentlayerop
wout += hlayer_act.T.dot(d_output) * lr
```

```
wh += X.T.dot(d_hiddenlayer) * lr

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

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

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

OUTPUT:



```
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.84047843]
 [0.81670721]
 [0.83471132]]
```