

INDEX

Practical no	Practical Aim	Page no
1	Design an Expert system using AIML.	01
2	Design a bot using AIML.	04
3	Implement Bayes Theorem using Python.	08
4	Implement Conditional Probability and JointProbability using Python.	11
5	Write a program to implement a Rule Based System.	13
6	Design a Fuzzy based application using Python.	14
7	Write an application to simulate supervised and unsupervised learning models.	18
8	Write an application to implement a clustering algorithm.	23
9	Write an application to implement the BFS algorithm.	24
10	Write an application to implement the DFS algorithm.	26

Practical: 1

Design an Expert system using AIML.

Aim : Design an Expert system using AIML.

Description :

What is an Expert System?

An expert system is a computer program that is designed to solve complex problems and to provide decision-making ability like a human expert. It performs this by extracting knowledge from its knowledge base using the reasoning and inference rules according to the user queries.

Code :

Simpley.py:

```
import aiml

kernel = aiml.Kernel()
kernel.learn("std-startup.xml")
kernel.respond("load aiml b")

while True:
    input_text = input(">Human: ")
    response = kernel.respond(input_text)
    print(">Bot: "+response)
```

basic_chat.aiml:

```
<aiml version="1.0.1" encoding="UTF-8">
<!-- basic_chat.aiml -->
    <category>
```

```
<pattern>HELLO *</pattern>
<template>
    Well, hello Misbah!
</template>
</category>

<category>
    <pattern>WHAT ARE YOU</pattern>
    <template>
        I'm a bot, and I'm silly!
    </template>
</category>

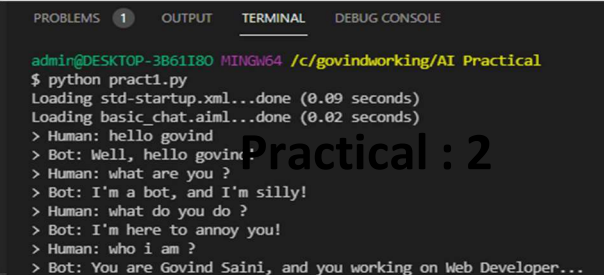
<category>
    <pattern>WHAT DO YOU DO</pattern>
    <template>
        I'm here to annoy you!
    </template>
</category>

<category>
    <pattern>Who am i</pattern>
    <template>
        You run a crappy YouTube Channel, get a
life...
    </template>
</category>
</aiml>
```

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>
```

Output:



```
PROBLEMS 1 OUTPUT TERMINAL DEBUG CONSOLE
admin@DESKTOP-3B61I80 MINGW64 /c/govindworking/AI Practical
$ python pract1.py
Loading std-startup.xml...done (0.09 seconds)
Loading basic_chat.aiml...done (0.02 seconds)
> Human: hello govind
> Bot: Well, hello govind!
> Human: what are you ?
> Bot: I'm a bot, and I'm silly!
> Human: what do you do ?
> Bot: I'm here to annoy you!
> Human: who i am ?
> Bot: You are Govind Saini, and you working on Web Developer...
```

Practical: 2

Design a bot using AIML.

Aim : Design a bot using AIML.

Description :

What is AIML?

AIML stands for Artificial Intelligence Modelling Language. AIML is an XML based markup language meant to create artificial intelligent applications. AIML makes it possible to create human interfaces while keeping the implementation simple to program, easy to understand and highly maintainable. This tutorial will teach you the basics of AIML. All the basic components of AIML with suitable examples have been discussed in this tutorial.

AIML Tags/Description

- `<aiml>` – defines the beginning and end of a AIML document.
- `<category>` – defines the unit of knowledge in bot's knowledge base.
- `<pattern>` – defines the pattern to match what a user may input to an bot.
- `<template>` – defines the response of a bot to user's input.

Code:

Simple.py:

```
import aiml

kernel = aiml.Kernel()

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

kernel.respond("load aiml b")

while True:

    input_text = input(">Human: ")

    response = kernel.respond(input_text)
```

```
print(">Bot: "+response)
```

basic_chat.aiml:

```
<aiml version="1.0.1" encoding="UTF-8">
<!-- basic_chat.aiml -->
    <category>
<pattern>HELLO *</pattern>
        <template>
            Well, hello Misbah!
        </template>
    </category>
    <category>
        <pattern>WHAT ARE YOU</pattern>
        <template>
            I'm a bot, and I'm silly!
        </template>
    </category>
    <category>
        <pattern>WHAT DO YOU DO</pattern>
        <template>
            I'm here to annoy you!
        </template>
    </category>
    <category>
        <pattern>Who am i</pattern>
        <template>
            You run a crappy YouTube Channel, get a
life...
```

```

        </template>
    </category>
</aiml>

```

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>

```

Output:

```
PROBLEMS 1 OUTPUT TERMINAL DEBUG CONSOLE

admin@DESKTOP-3B61I80 MINGW64 /c/govindworking/AI Practical
$ python pract1.py
Loading std-startup.xml...done (0.09 seconds)
Loading basic_chat.aiml...done (0.02 seconds)
> Human: hello govind
> Bot: Well, hello govind!
> Human: what are you ?
> Bot: I'm a bot, and I'm silly!
> Human: what do you do ?
> Bot: I'm here to annoy you!
> Human: who i am ?
> Bot: You are Govind Saini, and you working on Web Developer...
```


Practical: 3

Implement Bayes Theorem using Python.

Description :

Bayes' Theorem provides a way that we can calculate the probability of a piece of data belonging to a given class, given our prior knowledge. Bayes' Theorem is stated as:

$$P(\text{class} | \text{data}) = (P(\text{data} | \text{class}) * P(\text{class})) / P(\text{data})$$

Where $P(\text{class} | \text{data})$ is the probability of class given the provided data.

Naive Bayes is a classification algorithm for binary (two-class) and multiclass classification problems. It is called Naive Bayes or idiot Bayes because the calculations of the probabilities for each class are simplified to make their calculations tractable.

Code :

```
from sklearn.model_selection

import train_test_split from sklearn.datasets
import load_breast_cancer from sklearn.naive_bayes
import GaussianNB

from sklearn.metrics import accuracy_score

print("Saurabh Yadav 60")
data =
load_breast_cancer()

label_names = data['target_names']
labels = data['target']
features_names

=data['feature_names'] features =
data['data'] print(label_names)
print(labels[0])
print(features_names[0

]) print(features[0])
```

```
train, test, train_labels, test_labels =  
train_test_split(features, labels, test_size = 0.40, random_state = 3)  
gnb = GaussianNB()  
  
model =
```

```

gnb.fit(train,train_labels) pred
= model.predict(test) print(pred)

print("Accuracy:" ,accuracy_score(test_labels,pred)*100,"%")

```

Output:

```

['malignant' 'benign']
0
mean radius
[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]

```

```

[1 1 1 1 0 1 1 1 1 1 1 0 1 0 1 1 1 1 0 1 1 1 1 0 0 1 0 1 1 1 1 1
 1 0 0 0 1 1 0 1 1 1 1 0 1 0 1 1 0 0 1 1 0 1 1 1 0 1 1 0 0 1 1 0 0
 1 0 1 0 0 0 0 1 1 1 1 0 1 1 1 1 0 0 0 1 1 0 1 0 1 1 1 1 1 1 0 0 1 0 1 1
 0 1 1 0 0 1 0 0 0 1 1 0 1 0 1 0 1 0 1 1 0 0 1 1 1 1 0 1 1 1 1 0 1 1 1
 0 1 1 0 1 1 1 0 0 1 0 1 1 1 0 1 1 0 1 0 1 1 1 1 1 0 0 1 0 1 1 1 1 0 0 1 1
 0 1 1 1 0 0 1 1 1 1 1 1 0 1 1 1 1 0 0 1 1 0 0 0 1 0 1 1 0 1 1 1 1 0 1 1 1
 1 1 1 1 0 1]
Accuracy: 96.49122807017544 %

```

Practical: 4

Implement Conditional Probability and joint probability using Python.

Description :

What is Conditional Probability?

The probability of one event given the occurrence of another event is called the conditional probability. The conditional probability of one to one or more random variables is referred to as the conditional probability distribution.

For example, the conditional probability of event A given event B is written formally as:

• $P(A \text{ given } B)$

The “given” is denoted using the pipe “|” operator; for example:

• $P(A | B)$

The conditional probability for events A given event B is calculated as follows:

• $P(A \text{ given } B) = P(A \text{ and } B) / P(B)$

Code :

```
import enum, random
print("saurabh Yadav
60") 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)
print("P(both | either):", both_girls /
either_girl) print("P(either_girls):",
(either_girl/10000)*100) print("P(both_girls):",
(both_girls/10000)*100)

print("P(older_girl):", (older_girl/10000)*100)

```

Ouput :

```

P(both | older): 0.5007089325501317
P(both | either): 0.3311897106109325
P(either_girls): 74.64
P(both_girls): 24.72
P(older_girl): 49.3700000000000005

```

Practical: 5

A program to implement a Rule Based System.

Description :

What is a Rule Based System?

A rule-based system is a system that applies human-made rules to store, sort and manipulate data. In doing so, it mimics human intelligence.

To work, rule-based systems require a set of facts or source of data, and a set of rules for manipulating that data. These rules are sometimes referred to as 'If statements' as they tend to follow the line of 'IF X happens THEN do Y'.

Automation software like Think Automation is a good example. It automates processes by breaking them down into steps.

- First comes the data or new business event
- Then comes the analysis: the part where the system conditionally processes the data against its rules
- Then comes any subsequent automated follow-up action

Code :

```
practs5.pl
File Edit Browse Compile Prolog Pce Help

pracs5.pl | practs5.pl
man(lou).
man(pete).
man(ian).
man(peter).

woman(pauline).
woman(cathy).
woman(lucy).

parent(ian,lucy).
parent(ian,peter).
parent(cathy,ian).
parent(pete,ian).
parent(lou,pete).
parent(lou,pauline).

mother(X,Y):- woman(X),parent(X,Y),(X\=Y).
father(X,Y):- man(X),parent(X,Y),(X\=Y).

sibling(X,Y):- parent(Z,X),parent(Z,Y),(X\=Y).
brother(X,Y):- man(X), sibling(X,Y),(X\=Y).
sister(X,Y):- woman(X),sibling(X,Y),(X\=Y).

grandfather(X,Y):- father(X,Z),parent(Z,Y),(X\=Y).
grandmother(X,Y):- mother(X,Z),parent(Z,Y),(X\=Y).

ancestor(X,Y):- parent(X,Y),(X\=Y).
ancestor(X,Y):- parent(X,Z),ancestor(Z,Y),(X\=Y).

c:/govindworking/ai practical/practs5.pl compiled
Line: 11
```

Practical: 6

Design a Fuzzy based application using Python

Description :

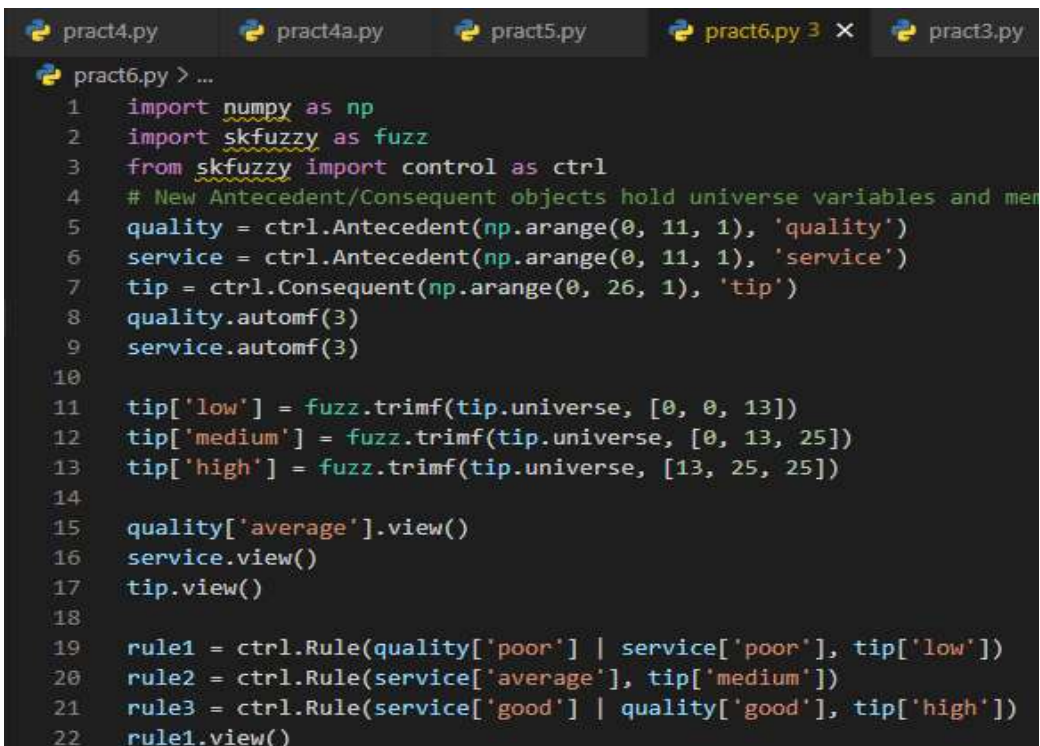
What is a Fuzzy based application?

Fuzzy sets were introduced by Lotfi Zadeh (1921–2017) in 1965.

Unlike crisp sets, a fuzzy set allows partial belonging to a set, that is defined by a degree of membership, denoted by μ , that can take any value from 0 (element does not belong at all in the set) to 1 (element belongs fully to the set).

It is evident that if we remove all the values of belonging except from 0 and 1, the fuzzy set will collapse to a crisp set that was described in the previous section.

Code :



```
pract6.py > ...
1  import numpy as np
2  import skfuzzy as fuzz
3  from skfuzzy import control as ctrl
4  # New Antecedent/Consequent objects hold universe variables and membership functions
5  quality = ctrl.Antecedent(np.arange(0, 11, 1), 'quality')
6  service = ctrl.Antecedent(np.arange(0, 11, 1), 'service')
7  tip = ctrl.Consequent(np.arange(0, 26, 1), 'tip')
8  quality.automf(3)
9  service.automf(3)
10
11  tip['low'] = fuzz.trimf(tip.universe, [0, 0, 13])
12  tip['medium'] = fuzz.trimf(tip.universe, [0, 13, 25])
13  tip['high'] = fuzz.trimf(tip.universe, [13, 25, 25])
14
15  quality['average'].view()
16  service.view()
17  tip.view()
18
19  rule1 = ctrl.Rule(quality['poor'] | service['poor'], tip['low'])
20  rule2 = ctrl.Rule(service['average'], tip['medium'])
21  rule3 = ctrl.Rule(service['good'] | quality['good'], tip['high'])
22  rule1.view()
```

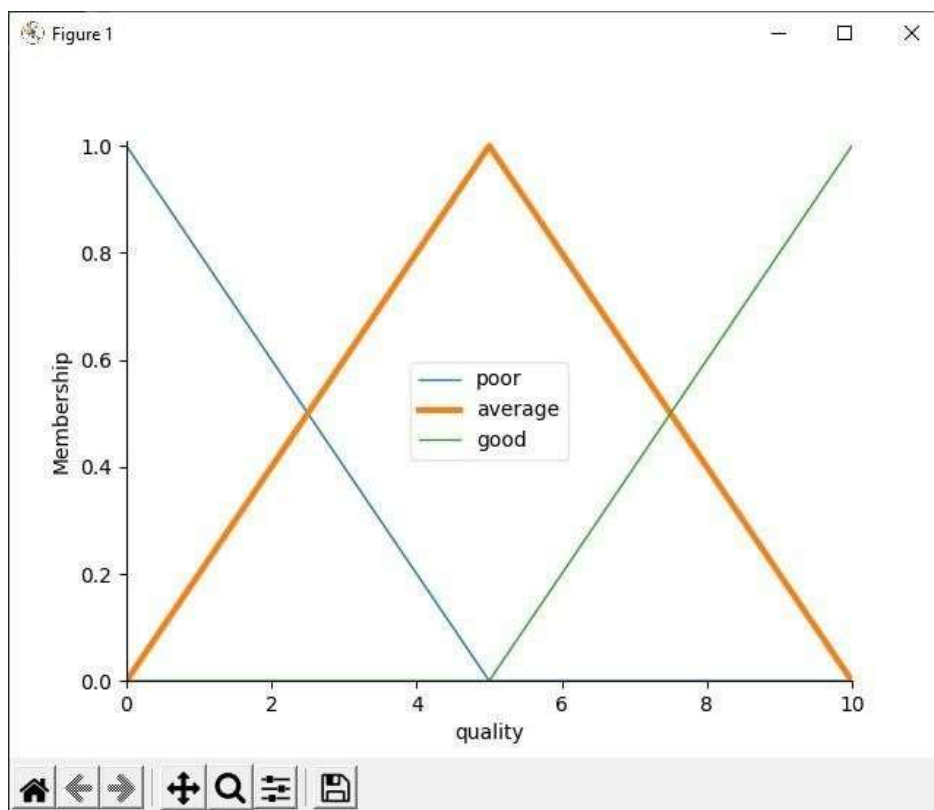


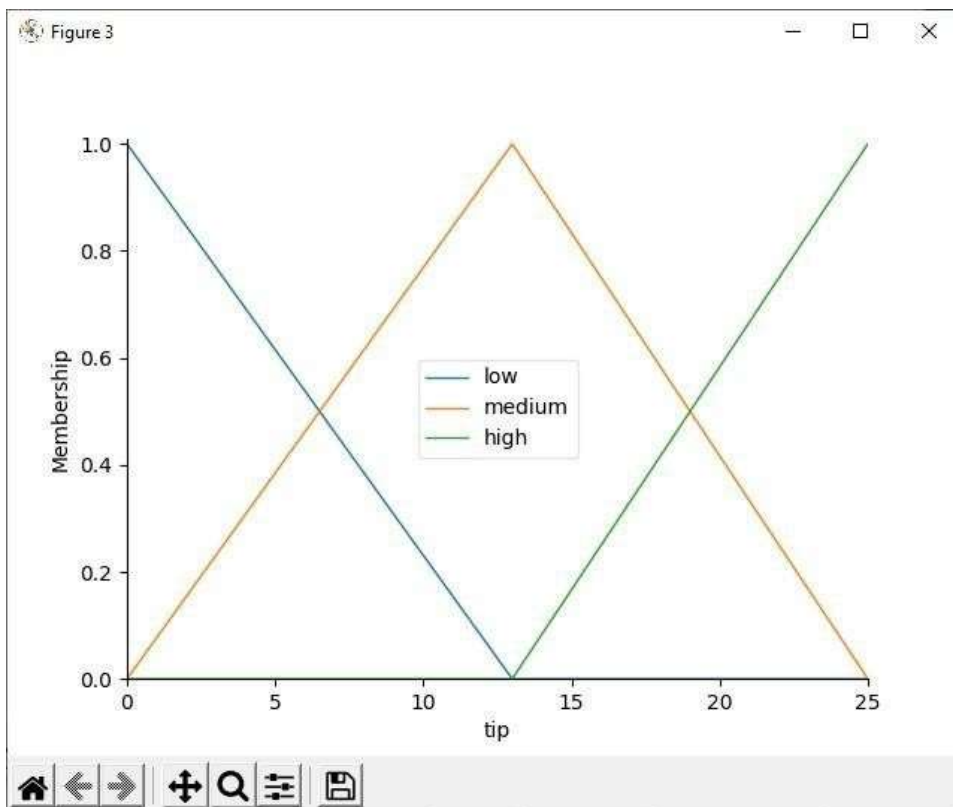
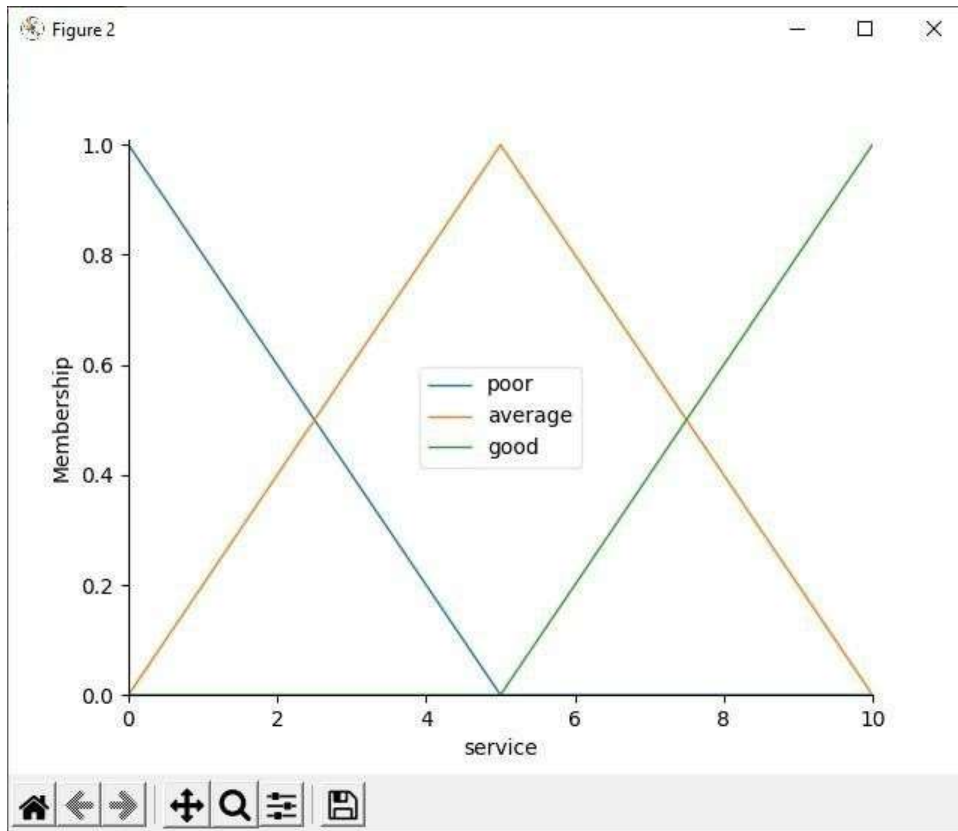
```

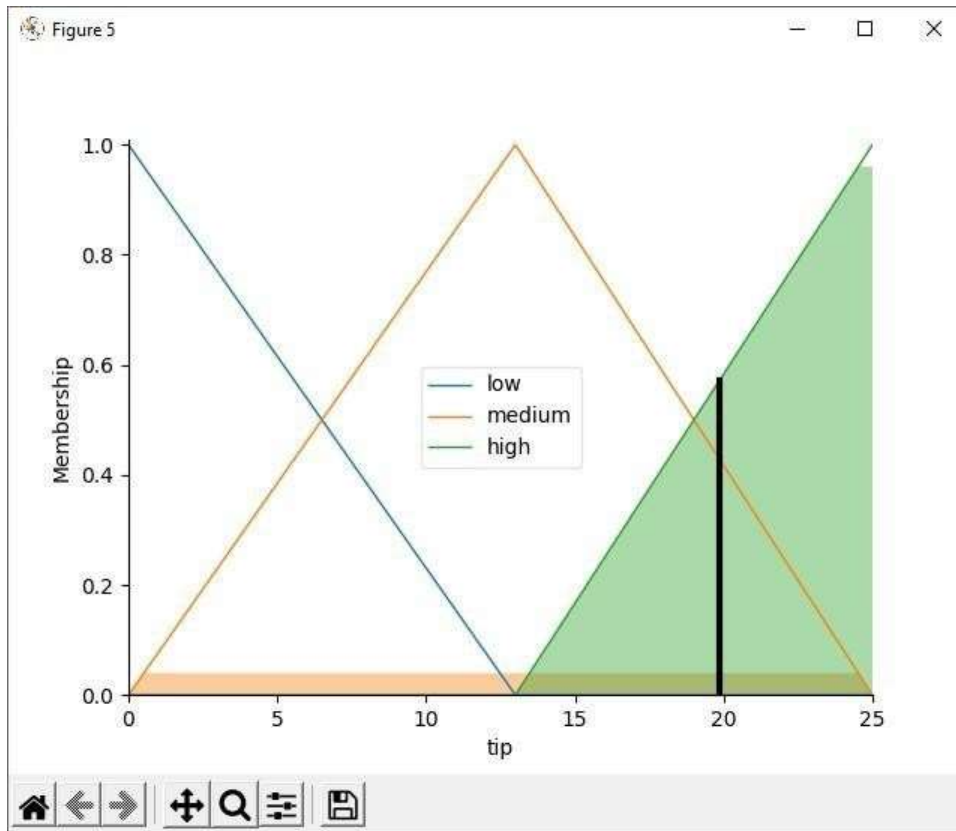
18
19 rule1 = ctrl.Rule(quality['poor'] | service['poor'], tip['low'])
20 rule2 = ctrl.Rule(service['average'], tip['medium'])
21 rule3 = ctrl.Rule(service['good'] | quality['good'], tip['high'])
22 rule1.view()
23
24 tipping_ctrl = ctrl.ControlSystem([rule1, rule2, rule3])
25 tipping = ctrl.ControlSystemSimulation(tipping_ctrl)
26 tipping.input['quality'] = 6.5
27 tipping.input['service'] = 9.8
28 tipping.compute()
29 print(tipping.output['tip'])
30 tip.view(sim=tipping)
31

```

Output :







PROBLEMS 3 OUTPUT TERMINAL DEBUG CONSOLE

```
admin@DESKTOP-3B61I80 MINGW64 /c/govindworking/AI Practical
$ python pract6.py
19.847607361963192

admin@DESKTOP-3B61I80 MINGW64 /c/govindworking/AI Practical
$
```

Practical: 7

Write an application to simulate supervised and unsupervised learning models.

Description :

What is supervised learning?

Supervised learning as the name indicates the presence of a supervisor as a teacher. Basically, supervised learning is a learning in which we teach or train the machine using data which is well labelled that means some data is already tagged with the correct answer.

Supervised learning classified into two categories of algorithms:

- **Classification:** A classification problem is when the output variable is a category, such as “Red” or “blue” or “disease” and “no disease”.
- **Regression:** A regression problem is when the output variable is a real value, such as “dollars” or “weight”.

Supervised learning deals with or learns with “labelled” data. Which implies that

some data is already tagged with the correct

answer. Types: -

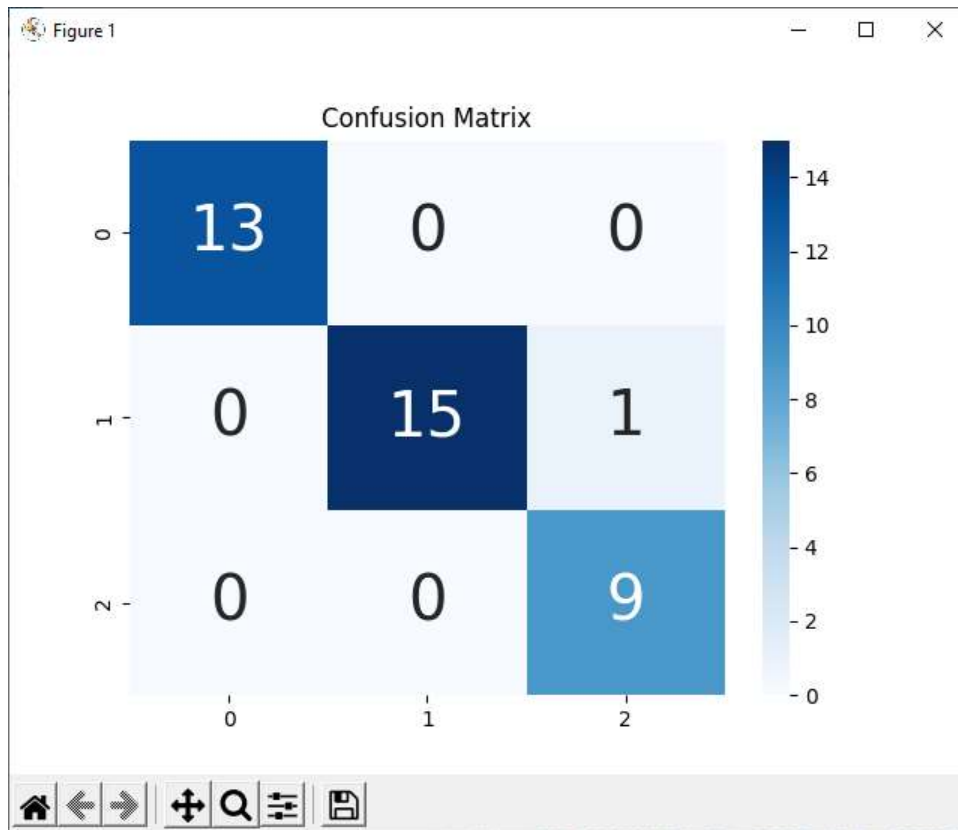
- **Regression**
- **Logistic Regression**
- **Classification**
- **Naive Bayes Classifiers**
- **K-NN (k nearest neighbours)**
- **Decision Trees**
- **Support Vector Machine**

Code :

```
pract7.py > ...
1  import numpy as np
2  import matplotlib.pyplot as plt
3  import pandas as pd
4  from sklearn.linear_model import LogisticRegression
5  from sklearn import datasets
6
7  # Importing the dataset
8  dataset = pd.read_csv("iris.csv")
9  dataset.describe()
10
11 # Splitting the dataset into the Training set and Test set
12 X = dataset.iloc[:, [0,1,2, 3]].values
13 y = dataset.iloc[:, 4].values
14 from sklearn.model_selection import train_test_split
15 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
16 from sklearn.preprocessing import StandardScaler
17 sc = StandardScaler()
18 X_train = sc.fit_transform(X_train)
19 X_test = sc.transform(X_test)
20
21 # Fitting Logistic Regression to the Training set
22 classifier = LogisticRegression(random_state = 0, solver='lbfgs', multi_class='auto')
23 classifier.fit(X_train, y_train)
24
25 # Predicting the Test set results
26 y_pred = classifier.predict(X_test)
27 # Predict probabilities
28 probs_y=classifier.predict_proba(X_test)
29 from sklearn.metrics import confusion_matrix
30 cm = confusion_matrix(y_test, y_pred)
31 print(cm)
32
33 # Plot confusion matrix
34 import seaborn as sns
35 import pandas as pd
36
37 # confusion matrix sns heatmap
38 ax = plt.axes()
39 df_cm = cm
40 sns.heatmap(df_cm, annot=True, annot_kws={"size": 30}, fmt='d', cmap="Blues", ax = ax )
41 ax.set_title('Confusion Matrix')
42 plt.show()
```

PROBLEMS 18 OUTPUT TERMINAL DEBUG CONSOLE

```
admin@DESKTOP-3B61I80 MINGW64 /c/govindworking/AI Practical
$ python pract7.py
[[13  0  0]
 [ 0 15  1]
 [ 0  0  9]]
```



Description :

What is Unsupervised Learning?

Unsupervised learning is the training of machine using information that is neither classified nor labelled and allowing the algorithm to act on that information without guidance. Here the task of machine is to group unsorted information according to similarities, patterns and differences without any prior training of data. Unsupervised learning classified into two categories of algorithms:

- Clustering: A clustering problem is where you want to discover the inherent groupings in the data, such as grouping customers by purchasing behaviour.
- Association: An association rule learning problem is where you want to discover rules that describe large portions of your data, such as people that buy X also tend to buy Y.

Types of Unsupervised Learning: -

Clustering

- Exclusive (partitioning)
- Agglomerative
- Overlapping
- Probabilistic

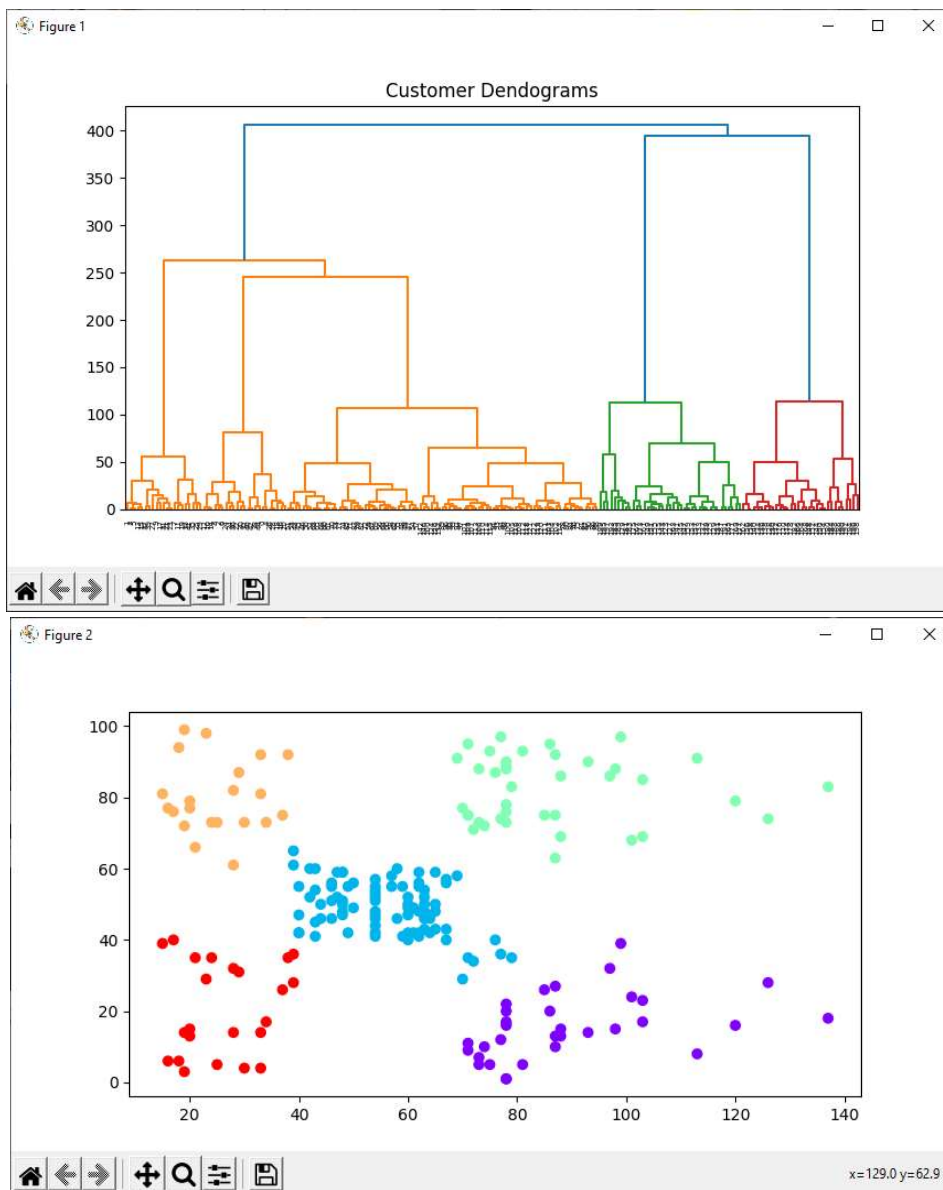
Clustering Types: -

- Hierarchical clustering
- K-means clustering
- Principal Component Analysis
- Singular Value Decomposition
- Independent Component Analysis

Code :

```
pract7b.py > ...
1  import matplotlib.pyplot as plt
2  import pandas as pd
3  import numpy as np
4  customer_data = pd.read_csv('Mall_Customers.csv')
5  customer_data.shape
6  customer_data.head()
7  data = customer_data.iloc[:, 3:5].values
8  import scipy.cluster.hierarchy as shc
9  plt.figure(figsize=(10, 7))
10 plt.title("Customer Dendograms")
11 dend = shc.dendrogram(shc.linkage(data, method='ward'))
12 from sklearn.cluster import AgglomerativeClustering
13 cluster = AgglomerativeClustering(n_clusters=5, affinity='euclidean', linkage='ward')
14 cluster.fit_predict(data)
15 plt.figure(figsize=(10, 7))
16 plt.scatter(data[:,0], data[:,1], c=cluster.labels_, cmap='rainbow')
17 plt.show()
```


Output :



Practical: 8

Write an application to implement Clustering algorithm.

Description :

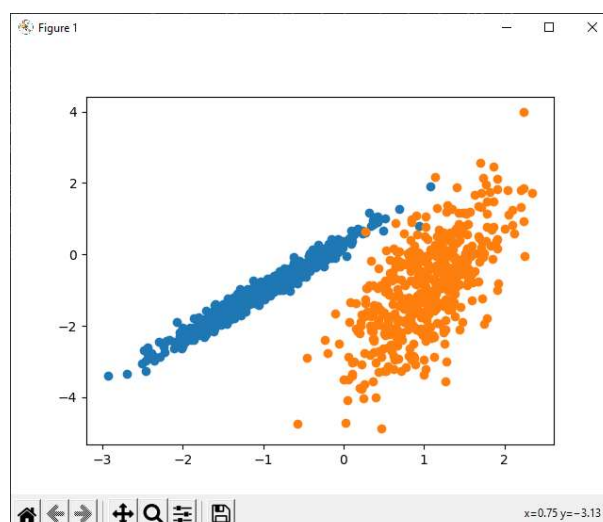
What is Clustering?

Clustering is the task of dividing the population or data points into a number of groups such that data points in the same groups are more similar to other data points in the same group than those in other groups.

Code :

```
pract4.py  pract4a.py  pract5.py  pract6.py  pract7a.py  pract7b.py 5
pract8.py > ...
1  # synthetic classification dataset
2  from numpy import where
3  from sklearn.datasets import make_classification
4  from matplotlib import pyplot
5  # define dataset
6  X, y = make_classification(n_samples=1000, n_features=2, n_informative=2, n_redundant=0,
7  n_clusters_per_class=1, random_state=4)
8  # create scatter plot for samples from each class
9  for class_value in range(2):
10     # get row indexes for samples with this class
11     row_ix = where(y == class_value)
12     # create scatter of these samples
13     pyplot.scatter(X[row_ix, 0], X[row_ix, 1])
14 # show the plot
15 pyplot.show()
```

Output :



Practical: 9

Write a Program to implement BFS algorithm.

Description :

What is Breadth-First Search?

Breadth-First Search (BFS) is an algorithm used for traversing graphs or trees. Traversing means visiting each node of the graph. Breadth-First Search is a recursive algorithm to search all the vertices of a graph or a tree. BFS in python can be implemented by using data structures like a dictionary and lists. As breadth-first search is the process of traversing each node of the graph, a standard BFS algorithm traverses each vertex of the graph into two parts:

1) Visited

2) Not Visited. So, the purpose of the algorithm is to visit all the vertex while avoiding cycles.

The steps of the algorithm work as follow:

1. Start by putting any one of the graph's vertices at the back of the queue.
2. Now take the front item of the queue and add it to the visited list.
3. Create a list of that vertex's adjacent nodes. Add those which are not within the visited list to the rear of the queue.
4. Keep continuing steps two and three till the queue is empty.

Code:

```
pract9.py > ...
1  import collections
2  # BFS algorithm
3  def bfs(graph, root):
4
5      visited, queue = set(), collections.deque([root])
6      visited.add(root)
7
8      while queue:
9          vertex = queue.popleft()
10         print(str(vertex) + " ", end="")
11
12         for neighbour in graph[vertex]:
13             if neighbour not in visited:
14                 visited.add(neighbour)
15                 queue.append(neighbour)
16
17 if __name__ == '__main__':
18     graph = {0: [1, 2], 1: [2], 2: [3], 3: [1, 2]}
19     print("Following is Breadth First Traversal: ")
20     bfs(graph, 0)
```

Output:

```
PROBLEMS 1 OUTPUT TERMINAL DEBUG CONSOLE
admin@DESKTOP-3B61I80 MINGW64 /c/govindworking/AI Practical
$ python pract9.py
Following is Breadth First Traversal:
0 1 2 3
admin@DESKTOP-3B61I80 MINGW64 /c/govindworking/AI Practical
```

Practical : 10

Aim : Write a Program to implement DFS algorithm.

Description :

What is Depth-First Search ?

The Depth-First Search is a recursive algorithm that uses the concept of backtracking. It involves thorough searches of all the nodes by going ahead if potential, else by backtracking. Here, the word backtrack means once you are moving forward and there are not any more nodes along the present path, you progress backward on an equivalent path to seek out nodes to traverse.

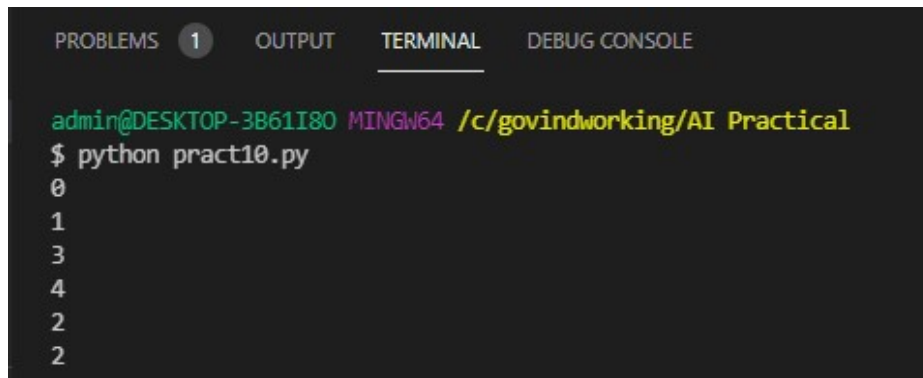
Algorithm:

- Create a recursive function that takes the index of the node and a visited array.
- Mark the current node as visited and print the node.
- Traverse all the adjacent and unmarked nodes and call the recursive function with the index of the adjacent node.

Code :

```
pract10.py > ...
1  # DFS algorithm
2  def dfs(graph, start, visited=None):
3      if visited is None:
4          visited = set()
5          visited.add(start)
6
7          print(start)
8
9          for next in graph[start] - visited:
10             dfs(graph, next, visited)
11     return visited
12
13
14     graph = {'0': set(['1', '2']),
15             '1': set(['0', '3', '4']),
16             '2': set(['0']),
17             '3': set(['1']),
18             '4': set(['2', '3'])}
19
20     dfs(graph, '0')
```

Output :



The screenshot shows a code editor interface with a dark theme. At the top, there are four tabs: 'PROBLEMS', '1', 'OUTPUT', 'TERMINAL', and 'DEBUG CONSOLE'. The 'TERMINAL' tab is active and underlined. The terminal content shows a command prompt session where the user runs a Python script. The output of the script is displayed on the following lines:

```
admin@DESKTOP-3B61I80 MINGW64 /c/govindworking/AI Practical
$ python pract10.py
0
1
3
4
2
2
```