Practical 1

Aim: Design an Expert system using AIML.

Code:

Python file

```
import aiml
def initialize_aiml_kernel():
kernel = aiml.Kernel()
try:
kernel.learn("animal_expert.aiml")
except Exception as e:
print("Error loading AIML file:", e)
return kernel
def expert_system():
print("Welcome to the Animal Expert System")
print("You can ask questions about animals. Type 'EXIT' to quit.")
kernel = initialize_aiml_kernel()
if kernel is None:
print("Exiting due to AIML initialization error.")
return
while True:
user_input = input("You: ").strip().upper()
if user_input == 'EXIT':
print("Goodbye!")
break
try:
response = kernel.respond(user_input)
print("Expert System: " + response)
except Exception as e:
print("Error processing input:", e)
if __name__ == "__main__":
expert_system()
```

animal_expert.aiml file :

```
<aiml>
  <category>
    <pattern>WHAT IS A DOG</pattern>
    <template>A dog is a domesticated carnivorous mammal.</template>
  </category>
  <category>
    <pattern>WHAT DOES A CAT EAT</pattern>
    <template>Cats are carnivores and usually eat meat.</template>
  </category>
  <category>
    <pattern>WHERE DOES A LION LIVE</pattern>
    <template>Lions are typically found in grasslands and savannas.</template>
  </category>
  <category>
    <pattern>EXIT</pattern>
    <template>Goodbye! If you have more questions, feel free to ask.</template>
  </category>
  <category>
    <pattern>*</pattern>
    <template>I'm sorry, I don't have information on that topic.</template>
  </category>
</aiml>
```

Output:

```
Welcome to the Animal Expert System
You can ask questions about animals. Type 'EXIT' to quit.
Loading C:\Users\apurv\Downloads\animal_expert.aiml...done (0.19 seconds)
You: what is a dog
Expert System: A dog is a domesticated carnivorous mammal.
You: what does a cat eat
Expert System: Cats are carnivores and usually eat meat.
You: where does a lion live
Expert System: Lions are typically found in grasslands and savannas.
You: *
WARNING: No match found for input: *
Expert System:
You: cat
Expert System: I'm sorry, I don't have information on that topic.
You: exit
Goodbye!
```

Practical 2

Aim: Design a bot using AIML.

Code:

```
#!/usr/bin/python3

import os

import aiml

BRAIN_FILE="brain.dump"

k = aiml.Kernel()

# To increase the startup speed of the bot it is possible to save the parsed aiml files as a dump.
#This code checks if a dump exists and otherwise loads the aiml from the xml files and saves the
#brain dump.

if os.path.exists(BRAIN_FILE):

   print("Loading from brain file: " + BRAIN_FILE)

   k.loadBrain(BRAIN_FILE)

else:

   print("Parsing aiml files")

   k.bootstrap(learnFiles="std-startup.aiml", commands="load aiml b")

   print("Saving brain file: " + BRAIN_FILE)

   k.saveBrain(BRAIN_FILE)

# Endless loop which passes the input to the bot and prints its response

while True:

   input_text = input("Enter the query> ")

   response = k.respond(input_text)

   print(response)
```

hi.aiml

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<aiml version="1.0">

<meta name="language" content="en"/>

<category>

     <pattern>HI</pattern>

     <template>

            <random>

                        <li>Hello there!</li>
```

```
                <li>Hey</li>
            </random>
             </template>
   </category>
<category>
    <pattern>HELLO</pattern>
    <template>
                <srai>HI</srai>
             </template>
   </category>
<category>
    <pattern>WHAT IS YOUR NAME</pattern>
    <template>
                You suggest something!
             </template>
   </category>
<category>
    <pattern>WHAT IS YOUR NAME?</pattern>
    <template>
                You suggest something!
             </template>
   </category>
<category>
    <pattern>LET YOUR NAME BE *</pattern>
    <template>
                Okay, <set name = "username"> <star/></set> is a good name!
             </template>
   </category>
<category>
    <pattern>HOW ABOUT *</pattern>
    <template>
                Okay, <set name = "botname"> <star/></set> is a good name!
```

```
        </template>
  </category>
<category>
   <pattern>MY NAME IS *</pattern>
   <template>
               Oh! Nice to meet you<set name = "username"> <star/></set>
        </template>
  </category>
<category>
   <pattern>THANK YOU *</pattern>
   <template>
               Your most welcome!
        </template>
  </category>
<category>
   <pattern>THANK YOU</pattern>
   <template>
               Your most welcome!
        </template>
  </category>
<category>
   <pattern>BYE *</pattern>
   <template>
               Goodbye!
        </template>
  </category>
<category>
   <pattern>BYE</pattern>
   <template>
               Goodbye!
        </template>
  </category>
```

</aiml>

Output:

```
Enter the query> Hello
Hey
Enter the query> How are you
I'm a bot, silly!
Enter the query> what are you doing
WARNING: No match found for input: what are you doing
```

Practical 3

Aim: Implement Bayes Theorem using Python

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

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

Aim: Implement Conditional Probability and joint probability using Python.

Code:

```
import enum
import random


class Kid(enum.Enum):
    BOY = 0
    GIRL = 1


def random_kid() -> Kid:
    return random.choice([Kid.BOY, Kid.GIRL])


def probability_example(iterations):
    both_girls = 0
    older_girl = 0
    either_girl = 0

    random.seed(0)

    for _ in range(iterations):
        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
```

```
    # Conditional Probability: P(both | older)
    conditional_prob_both_given_older = both_girls / older_girl

    # Conditional Probability: P(both | either)
    conditional_prob_both_given_either = both_girls / either_girl

    # Joint Probability: P(either_girls)
    joint_prob_either_girls = (either_girl / iterations) * 100

    # Joint Probability: P(both_girls)
    joint_prob_both_girls = (both_girls / iterations) * 100

    # Joint Probability: P(older_girl)
    joint_prob_older_girl = (older_girl / iterations) * 100

    print("Conditional Probability - P(both | older):", conditional_prob_both_given_older)
    print("Conditional Probability - P(both | either):",
conditional_prob_both_given_either)
    print("Joint Probability - P(either_girls):", joint_prob_either_girls)
    print("Joint Probability - P(both_girls):", joint_prob_both_girls)
    print("Joint Probability - P(older_girl):", joint_prob_older_girl)

if __name__ == "__main__":
    # Get user input for the number of iterations
    iterations = int(input("Enter the number of iterations: "))

    # Run the probability example with user-specified iterations
    probability_example(iterations)
```

Output:

```
Enter the number of iterations: 10
Conditional Probability - P(both | older): 0.5
Conditional Probability - P(both | either): 0.3333333333333333
Joint Probability - P(either_girls): 90.0
Joint Probability - P(both_girls): 30.0
Joint Probability - P(older_girl): 60.0
PS C:\Users\schau\Videos\Shivam-All-Updated-Practicals(AAI)>
```

Practical 5

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

Code:

```
def check_eligibility(age, income):
  rules = {
    "Rule1": age >= 18 and income > 30000,
    "Rule2": age >= 25 and income > 20000,
    "Rule3": age >= 30 and income > 15000
  }
  # Applying rules
  if rules["Rule1"]:
    return "Eligible for 10% discount"
  elif rules["Rule2"]:
    return "Eligible for 15% discount"
  elif rules["Rule3"]:
    return "Eligible for 20% discount"
  else:
    return "Not eligible for any discount"
# Test the rule-based system
person_age = 28
person_income = 25000
result = check_eligibility(person_age, person_income)
print(result)
```

Output:

```
Eligible for 15% discount
```

Practical 6

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

Code:

```
from fuzzywuzzy import fuzz

from fuzzywuzzy import process

s1 = "I love fuzzysforfuzzys"

s2 = "I am loving fuzzysforfuzzys"

print ("FuzzyWuzzy Ratio:", fuzz.ratio(s1, s2))

print ("FuzzyWuzzy PartialRatio: ", fuzz.partial_ratio(s1, s2))

print ("FuzzyWuzzy TokenSortRatio: ", fuzz.token_sort_ratio(s1, s2))

print ("FuzzyWuzzy TokenSetRatio: ", fuzz.token_set_ratio(s1, s2))

print ("FuzzyWuzzy WRatio: ", fuzz.WRatio(s1, s2),'\n\n')

# for process library,

query = 'fuzzys for fuzzys'

choices = ['fuzzy for fuzzy', 'fuzzy fuzzy', 'g. for fuzzys']

print ("List of ratios: ")

print (process.extract(query, choices), '\n')

print ("Best among the above list:",process.extractOne(query, choices))
```

Output:

```
FuzzyWuzzy Ratio: 86
FuzzyWuzzy PartialRatio:  86
FuzzyWuzzy TokenSortRatio:  86
FuzzyWuzzy TokenSetRatio:  87
FuzzyWuzzy WRatio:  86


List of ratios:
[('g. for fuzzys', 95), ('fuzzy for fuzzy', 94), ('fuzzy fuzzy', 86)]

Best among the above list: ('g. for fuzzys', 95)
```

Practical 7

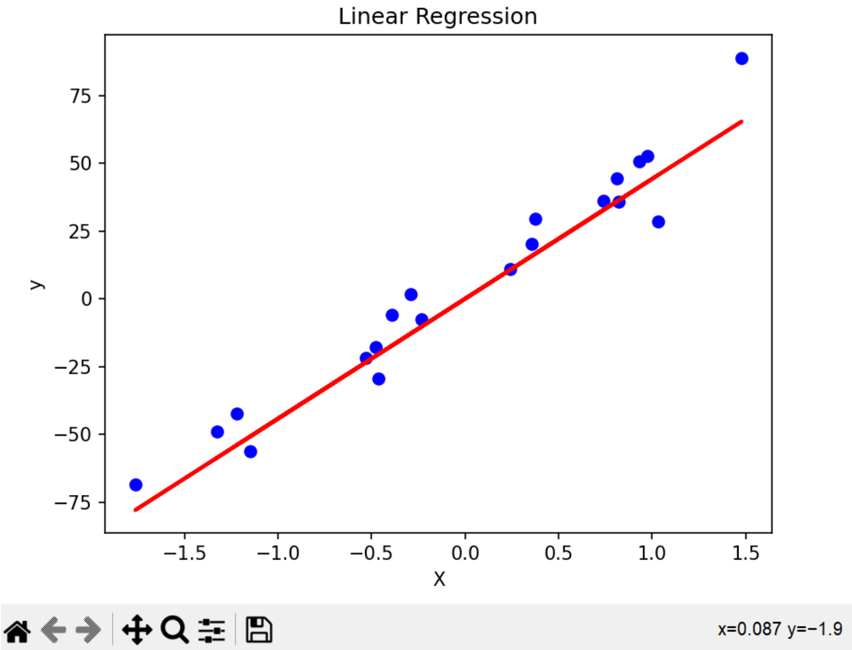
Aim: Write an application to simulate supervised and un-supervised learning model.

Code:

Supervised Learning (Linear Regression)

```
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
# Generate synthetic data for regression
X, y = make_regression(n_samples=100, n_features=1, noise=10, random_state=42)
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Create and train a linear regression model
model = LinearRegression()
model.fit(X_train, y_train)
# Make predictions
y_pred = model.predict(X_test)
# Calculate Mean Squared Error (MSE)
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")
# Plotting the results
plt.scatter(X_test, y_test, color='blue')
plt.plot(X_test, y_pred, color='red', linewidth=2)
plt.xlabel('X')
plt.ylabel('y')
plt.title('Linear Regression')
plt.show()
```

Output:

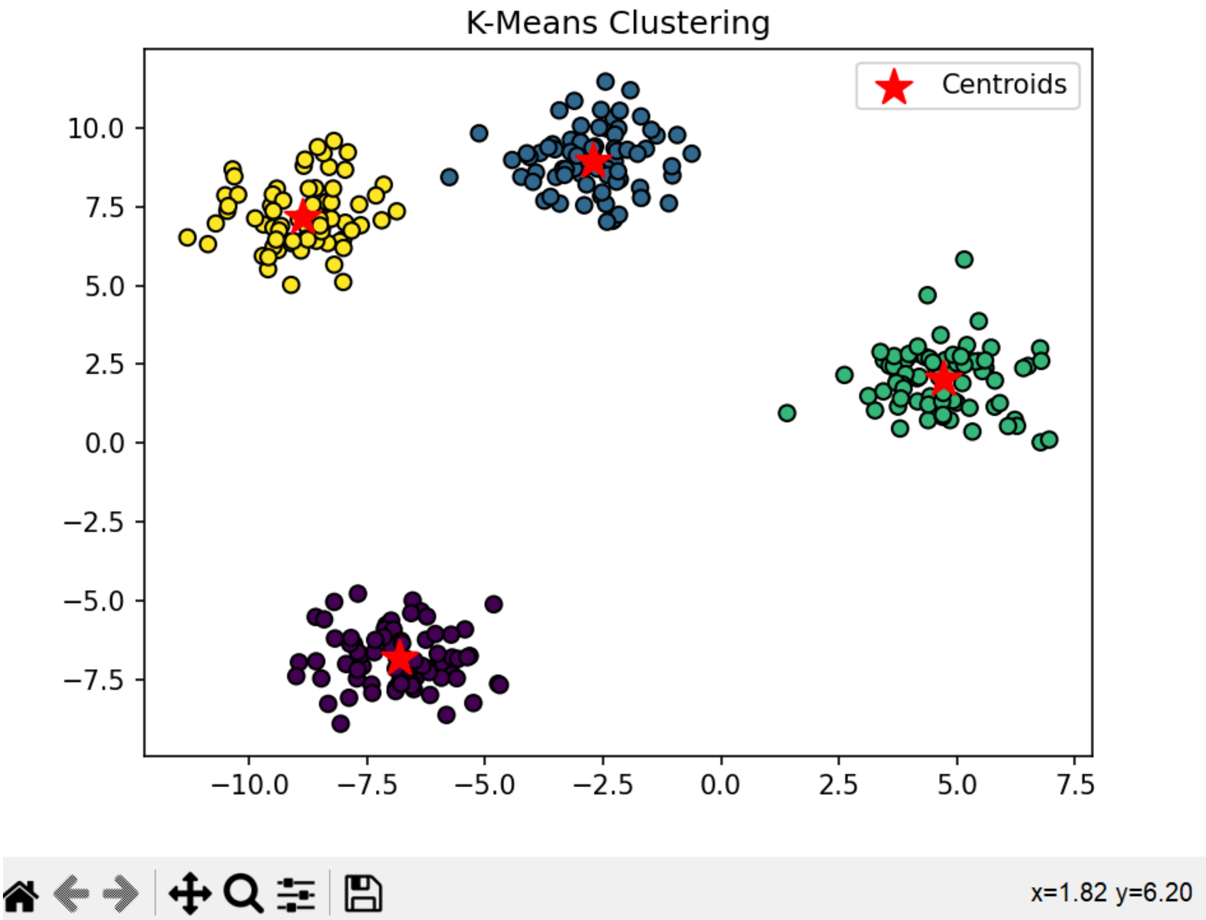


Supervised Learning (Linear Regression)

```
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
# Generate synthetic data for clustering
X, _ = make_blobs(n_samples=300, centers=4, cluster_std=1.0, random_state=42)
# Apply K-Means clustering
kmeans = KMeans(n_clusters=4)
kmeans.fit(X)
# Get cluster labels and centroids
labels = kmeans.labels_
centers = kmeans.cluster_centers_
# Plotting the clusters and centroids
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', edgecolor='k')
plt.scatter(centers[:, 0], centers[:, 1], c='red', marker='*', s=200, label='Centroids')
plt.title('K-Means Clustering')
plt.legend()
```

```
plt.show()
```

Output:



Practical 8

Aim: Write an application to implement clustering algorithm.

Code:

```
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.mixture import GaussianMixture

# Function to generate synthetic data
def generate_data():
    n_samples = int(input("Enter the number of data points to generate: "))
    n_centers = int(input("Enter the number of clusters to generate: "))

    A, B = make_blobs(
        n_samples=n_samples,
        centers=n_centers,
        cluster_std=0.60,
        random_state=0
    )

    return A, B

# Function to plot generated data
def plot_data(A):
    plt.scatter(A[:, 0], A[:, 1])
    plt.title('Generated Data')
    plt.show()

# Function to perform Gaussian Mixture Model (GMM) clustering
def apply_gmm(A, num_clusters):
    gmm = GaussianMixture(n_components=num_clusters, random_state=0)
    pred_B = gmm.fit_predict(A)
```

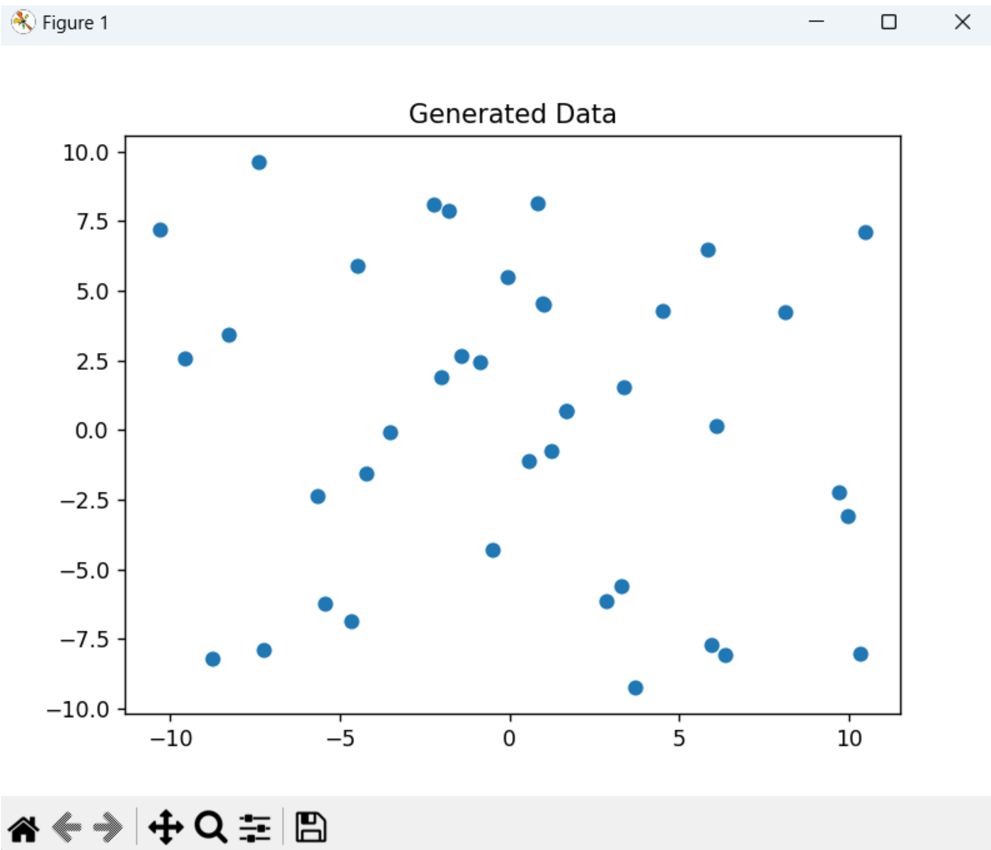
```
    # Plot the clustered data
    plt.scatter(A[:, 0], A[:, 1], c=pred_B, cmap='viridis')
    plt.title('Gaussian Mixture Model (GMM) Clustering')
    plt.show()

if __name__ == "__main__":
    # Generate and plot synthetic data
    A, B = generate_data()
    plot_data(A)

    # Get user input for the number of clusters for GMM
    num_clusters_gmm = int(input("Enter the number of clusters for GMM: "))

    # Apply GMM clustering with the chosen number of clusters
    apply_gmm(A, num_clusters_gmm)
```

Output:



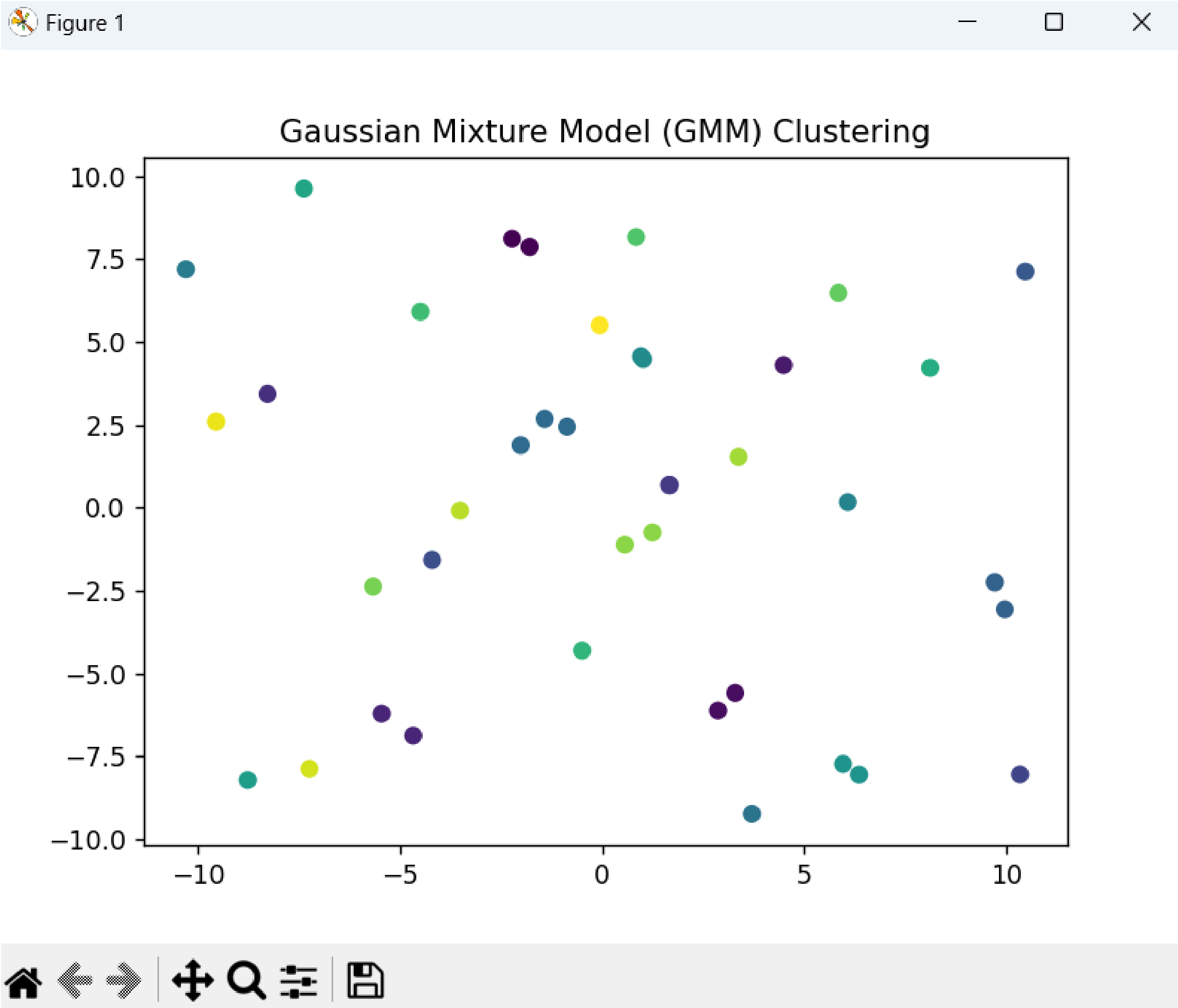


Figure 1
Gaussian Mixture Model (GMM) Clustering
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−10
−5
0
5
10

Practical 9

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

Code:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix
# Data Source
datasrc = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
colnames = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'Class']
DS = pd.read_csv(datasrc, names=colnames)
# Data Splitting
P = DS.drop('Class', axis=1)
Q = DS['Class']
P_train, P_test, Q_train, Q_test = train_test_split(P, Q, test_size=0.20)
# Polynomial Kernel
print("\nPolynomial Kernel")
degree = 8
classifierp = SVC(kernel='poly', degree=degree)
classifierp.fit(P_train, Q_train)
Q_pred = classifierp.predict(P_test)
print("Confusion Matrix:")
print(confusion_matrix(Q_test, Q_pred))
print("\nClassification Report:")
print(classification_report(Q_test, Q_pred))
# Gaussian Kernel
print("\nGaussian Kernel")
classifierg = SVC(kernel='rbf')
classifierg.fit(P_train, Q_train)
Q_pred = classifierg.predict(P_test)
print("Confusion Matrix:")
print(confusion_matrix(Q_test, Q_pred))
```

```
print("\nClassification Report:")
print(classification_report(Q_test, Q_pred))
# Sigmoid Kernel
print("\nSigmoid Kernel")
classifiers = SVC(kernel='sigmoid')
classifiers.fit(P_train, Q_train)
Q_pred = classifiers.predict(P_test)
print("Confusion Matrix:")
print(confusion_matrix(Q_test, Q_pred))
print("\nClassification Report:")
print(classification_report(Q_test, Q_pred))
```

Output:

```
Polynomial Kernel
Confusion Matrix:
[[10  0  0]
 [ 0 10  0]
 [ 0  0 10]]

Classification Report:
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        10
Iris-versicolor       1.00      1.00      1.00        10
 Iris-virginica       1.00      1.00      1.00        10

       accuracy                           1.00        30
      macro avg       1.00      1.00      1.00        30
   weighted avg       1.00      1.00      1.00        30


Gaussian Kernel
Confusion Matrix:
[[10  0  0]
 [ 0 10  0]
 [ 0  0 10]]

Classification Report:
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        10
Iris-versicolor       1.00      1.00      1.00        10
 Iris-virginica       1.00      1.00      1.00        10

       accuracy                           1.00        30
      macro avg       1.00      1.00      1.00        30
   weighted avg       1.00      1.00      1.00        30


Sigmoid Kernel
Confusion Matrix:
[[ 2  0  8]
 [ 6  0  4]
 [10  0  0]]


Classification Report:
```

```
                 precision    recall  f1-score   support

    Iris-setosa       0.11      0.20      0.14        10
Iris-versicolor       0.00      0.00      0.00        10
 Iris-virginica       0.00      0.00      0.00        10

       accuracy                           0.07        30
      macro avg       0.04      0.07      0.05        30
   weighted avg       0.04      0.07      0.05        30
```

Practical 10

Aim: Simulate artificial neural network model with both feedforward and backpropagation approach. [You can add some functionalities to enhance the model].

Code:

```
from math import exp

from random import seed, random


def initialize_network(n_inputs, n_hidden, n_outputs):

    network = list()

    hidden_layer = [{'weights': [random() for _ in range(n_inputs + 1)]} for _ in range(n_hidden)]

    network.append(hidden_layer)

    output_layer = [{'weights': [random() for _ in range(n_hidden + 1)]} for _ in range(n_outputs)]

    network.append(output_layer)

    return network


def activate(weights, inputs):

    activation = weights[-1]

    for i in range(len(weights) - 1):

        activation += weights[i] * inputs[i]

    return activation


def transfer(activation):

    return 1.0 / (1.0 + exp(-activation))


def forward_propagate(network, row):

    inputs = row

    for layer in network:

        new_inputs = []

        for neuron in layer:
```

```
            activation = activate(neuron['weights'], inputs)
            neuron['output'] = transfer(activation)
            new_inputs.append(neuron['output'])
        inputs = new_inputs
    return inputs

def predict(network, row):
    outputs = forward_propagate(network, row)
    return outputs.index(max(outputs))

def transfer_derivative(output):
    return output * (1.0 - output)

def backward_propagate_error(network, expected):
    for i in reversed(range(len(network))):
        layer = network[i]
        errors = list()
        if i != len(network) - 1:
            for j in range(len(layer)):
                error = 0.0
                for neuron in network[i + 1]:
                    error += (neuron['weights'][j] * neuron['delta'])
                errors.append(error)
        else:
            for j in range(len(layer)):
                neuron = layer[j]
                errors.append(expected[j] - neuron['output'])
        for j in range(len(layer)):
            neuron = layer[j]
            neuron['delta'] = errors[j] * transfer_derivative(neuron['output'])
```

```
def update_weights(network, row, learn_rate):
    for i in range(len(network)):
        inputs = row[:-1]
        if i != 0:
            inputs = [neuron['output'] for neuron in network[i - 1]]
        for neuron in network[i]:
            for j in range(len(inputs)):
                neuron['weights'][j] += learn_rate * neuron['delta'] * inputs[j]
            neuron['weights'][-1] += learn_rate * neuron['delta']

def train_network(network, train, learn_rate, n_epoch, n_outputs):
    for epoch in range(n_epoch):
        sum_error = 0
        for row in train:
            outputs = forward_propagate(network, row)
            expected = [0 for _ in range(n_outputs)]
            expected[int(row[-1])] = 1
            sum_error += sum([(expected[i] - outputs[i]) ** 2 for i in range(len(expected))])
            backward_propagate_error(network, expected)
            update_weights(network, row, learn_rate)
        print('>epoch=%d, lrate=%.3f, error=%.3f' % (epoch, learn_rate, sum_error))

def back_propagation(train, test, l_rate, n_epoch, n_hidden):
    n_inputs = len(train[0]) - 1
    n_outputs = len(set([row[-1] for row in train]))
    network = initialize_network(n_inputs, n_hidden, n_outputs)
    train_network(network, train, l_rate, n_epoch, n_outputs)
    predictions = [predict(network, row) for row in test]
    return predictions
```

```
# User Input
seed(1)
hidden_neurons = int(input("Enter the number of hidden neurons: "))
learning_rate = float(input("Enter the learning rate: "))
epochs = int(input("Enter the number of training epochs: "))

# Example Dataset
dataset = [
    [2.7810836, 2.550537003, 0],
    [1.465489372, 2.362125076, 0],
    [3.396561688, 4.400293529, 0],
    [1.38807019, 1.850220317, 0],
    [3.06407232, 3.005305973, 0],
    [7.627531214, 2.759262235, 1],
    [5.332441248, 2.088626775, 1],
    [6.922596716, 1.77106367, 1],
    [8.675418651, -0.242068655, 1],
    [7.673756466, 3.508563011, 1]
]

n_inputs = len(dataset[0]) - 1
n_outputs = len(set([row[-1] for row in dataset]))
network = initialize_network(n_inputs, hidden_neurons, n_outputs)

# Feedforward
print("Feedforward:")
for row in dataset:
    output = forward_propagate(network, row)
    print(f"Input: {row[:-1]}, Predicted Output: {output}")
```

```
# Backpropagation
print("\nBackpropagation:")
for epoch in range(epochs):
    sum_error = 0
    for row in dataset:
        outputs = forward_propagate(network, row)
        expected = [0 for _ in range(n_outputs)]
        expected[int(row[-1])] = 1
        sum_error += sum([(expected[i] - outputs[i]) ** 2 for i in range(len(expected))])
        backward_propagate_error(network, expected)
        update_weights(network, row, learning_rate)
    print('>epoch=%d, lrate=%.3f, error=%.3f' % (epoch, learning_rate, sum_error))

# Print final weights
print("\nFinal Weights:")
for i, layer in enumerate(network):
    print(f"\nLayer {i + 1} Weights:")
    for j, neuron in enumerate(layer):
        print(f"Neuron {j + 1} Weights: {neuron['weights']}")
```

Output:

```
Enter the number of hidden neurons: 4
Enter the learning rate: 0.1
Enter the number of training epochs: 10
Feedforward:
Input: [2.7810836, 2.550537003], Predicted Output: [0.8885643239851334, 0.911714898040579]
Input: [1.465489372, 2.362125076], Predicted Output: [0.8851355393872001, 0.907712559089926]
Input: [3.396561688, 4.400293529], Predicted Output: [0.8949829858660413, 0.9176241863566853]
Input: [1.38807019, 1.850220317], Predicted Output: [0.8790158007982337, 0.9030308873736764]
Input: [3.06407232, 3.005305973], Predicted Output: [0.8912574671817232, 0.9140875129576786]
Input: [7.627531214, 2.759262235], Predicted Output: [0.8920537118875939, 0.9173889653558378]
Input: [5.332441248, 2.088626775], Predicted Output: [0.8874647088546003, 0.9134807793586764]
Input: [6.922596716, 1.77106367], Predicted Output: [0.8860288601159657, 0.9141174259908511]
Input: [8.675418651, -0.242068655], Predicted Output: [0.8540135905842575, 0.9025156879101746]
Input: [7.673756466, 3.508563011], Predicted Output: [0.8941576422423522, 0.9184465505672885]

Backpropagation:
>epoch=0, lrate=0.100, error=8.044
>epoch=1, lrate=0.100, error=7.735
>epoch=2, lrate=0.100, error=7.369
>epoch=3, lrate=0.100, error=6.954
>epoch=4, lrate=0.100, error=6.515
>epoch=5, lrate=0.100, error=6.085
>epoch=6, lrate=0.100, error=5.693
>epoch=7, lrate=0.100, error=5.356
>epoch=8, lrate=0.100, error=5.071
>epoch=9, lrate=0.100, error=4.834
```

```
Final Weights:

Layer 1 Weights:
Neuron 1 Weights: [-0.010745630573869708, 0.800117219495545, 0.7347906622056257]
Neuron 2 Weights: [0.2573034055891132, 0.40578941320665657, 0.4187859712901473]
Neuron 3 Weights: [0.6565048730774249, 0.7987689667175697, 0.09820888102589154]
Neuron 4 Weights: [-0.4214759837467835, 0.7670124853740808, 0.38802578262318876]

Layer 2 Weights:
Neuron 1 Weights: [0.40279983856312623, -0.3992466053389011, 0.030585485646470203, 0.5117672326574959, -0.17969881864106055]
Neuron 2 Weights: [0.5542012192882254, 0.5553726981500557, -0.3475078697375515, -0.39937645531782623, 0.1439052592032093]
```

Practical 11

Aim: Simulate genetic algorithm with suitable example using Python / R or any other platform.

Code:

```
import random


def generate_random_gene(genes):
    return random.choice(genes)


def generate_random_individual(target_string, genes):
    return ''.join(generate_random_gene(genes) for _ in range(len(target_string)))


def calculate_fitness(individual, target_string):
    return sum(1 for a, b in zip(individual, target_string) if a == b)


def crossover(parent1, parent2):
    crossover_point = random.randint(0, len(parent1) - 1)
    child = parent1[:crossover_point] + parent2[crossover_point:]
    return child


def mutate(individual, mutation_rate, genes):
    mutated_individual = list(individual)
    for i in range(len(mutated_individual)):
        if random.random() < mutation_rate:
            mutated_individual[i] = generate_random_gene(genes)
    return ''.join(mutated_individual)


def genetic_algorithm(target_string, genes, population_size, mutation_rate):
    # Initialize population
    population = [generate_random_individual(target_string, genes) for _ in
range(population_size)]
```

```
    generation = 1

    while True:
        # Evaluate fitness of each individual in the population
        fitness_scores = [calculate_fitness(individual, target_string) for individual in
population]

        # Check for a perfect match
        if max(fitness_scores) == len(target_string):
            print("Target string reached!")
            break

        # Select the top individuals for reproduction
        selected_indices = sorted(range(len(fitness_scores)), key=lambda k:
fitness_scores[k],
                                  reverse=True)[:10]
        selected_parents = [population[i] for i in selected_indices]

        # Create a new population through crossover and mutation
        new_population = []
        while len(new_population) < population_size:
            parent1, parent2 = random.choices(selected_parents, k=2)
            child = crossover(parent1, parent2)
            child = mutate(child, mutation_rate, genes)
            new_population.append(child)

        population = new_population

        # Print progress
        print(f"Generation {generation}: {max(fitness_scores)} / {len(target_string)}")
        generation += 1
```

```
if __name__ == "__main__":
    # Get user input
    target_string = input("Enter the target string: ")
    genes = input("Enter the possible genes (characters): ")
    population_size = int(input("Enter the population size: "))
    mutation_rate = float(input("Enter the mutation rate: "))

    genetic_algorithm(target_string, genes, population_size, mutation_rate)
```

Output:

```
schau/Downloads/simulate_genetic_algorithm_with_suitable_example_usin
Enter the target string: shivam
Enter the possible genes (characters): abcdefghijklmnopqrstuvwxyz
Enter the population size: 15
Enter the mutation rate: 0.01
Generation 1: 1 / 6
Generation 2: 1 / 6
Generation 3: 1 / 6
Generation 4: 1 / 6
Generation 5: 1 / 6
Generation 6: 2 / 6
Generation 7: 2 / 6
Generation 8: 2 / 6
Generation 9: 2 / 6
Generation 10: 2 / 6
Generation 11: 2 / 6
Generation 12: 2 / 6
Generation 13: 2 / 6
Generation 14: 2 / 6
Generation 15: 2 / 6
Generation 16: 2 / 6
Generation 17: 2 / 6
Generation 18: 2 / 6
Generation 19: 2 / 6
Generation 20: 2 / 6
Generation 21: 2 / 6
Generation 22: 2 / 6
Generation 23: 2 / 6
Generation 24: 2 / 6
Generation 25: 2 / 6
Generation 26: 2 / 6
Generation 27: 2 / 6
Generation 28: 2 / 6
Generation 29: 2 / 6
Generation 30: 2 / 6
```

```
Generation 388: 5 / 6
Generation 389: 5 / 6
Generation 390: 5 / 6
Generation 391: 5 / 6
Generation 392: 5 / 6
Generation 393: 5 / 6
Generation 394: 5 / 6
Generation 395: 5 / 6
Generation 396: 5 / 6
Generation 397: 5 / 6
Generation 398: 5 / 6
Generation 399: 5 / 6
Generation 400: 5 / 6
Generation 401: 5 / 6
Generation 402: 5 / 6
Generation 403: 5 / 6
Generation 404: 5 / 6
Generation 405: 5 / 6
Generation 406: 5 / 6
Generation 407: 5 / 6
Generation 408: 5 / 6
Generation 409: 5 / 6
Generation 410: 5 / 6
Generation 411: 5 / 6
Generation 412: 5 / 6
Generation 413: 5 / 6
Generation 414: 5 / 6
Generation 415: 5 / 6
Generation 416: 5 / 6
Generation 417: 5 / 6
Generation 418: 5 / 6
Generation 419: 5 / 6
Target string reached!
```

Practical 12

Aim: Design an Artificial Intelligence application to implement intelligent agents.

Code:

```
import random

class Agent:
    def __init__(self, name, position):
        self.name = name
        self.position = position

    def move(self, direction):
        x, y = self.position
        if direction == 'up':
            self.position = (x, y + 1)
        elif direction == 'down':
            self.position = (x, y - 1)
        elif direction == 'left':
            self.position = (x - 1, y)
        elif direction == 'right':
            self.position = (x + 1, y)

class Environment:
    def __init__(self, width, height):
        self.width = width
        self.height = height
        self.agents = []

    def add_agent(self, agent):
        self.agents.append(agent)
```

```
    def display(self):
        matrix = [['.' for _ in range(self.width)] for _ in range(self.height)]
        for agent in self.agents:
            x, y = agent.position
            matrix[y][x] = 'A'  # A for Agent

        for row in matrix:
            print(' '.join(row))
        print()

def main():
    width = int(input("Enter the width of the environment: "))
    height = int(input("Enter the height of the environment: "))
    env = Environment(width=width, height=height)

    num_agents = int(input("Enter the number of agents: "))
    for i in range(1, num_agents + 1):
        agent_name = f"Agent{i}"
        initial_x = int(input(f"Enter the initial x-coordinate for {agent_name}: "))
        initial_y = int(input(f"Enter the initial y-coordinate for {agent_name}: "))
        agent = Agent(name=agent_name, position=(initial_x, initial_y))
        env.add_agent(agent)

    num_iterations = int(input("Enter the number of iterations: "))
    for _ in range(num_iterations):
        env.display()
        print("Agents' positions:")
        for agent in env.agents:
            print(f"{agent.name}: {agent.position}")
```

```
        print()

        for agent in env.agents:
            direction = random.choice(['up', 'down', 'left', 'right'])
            agent.move(direction)

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

output:

```
Enter the width of the environment: 5
Enter the height of the environment: 5
Enter the number of agents: 2
Enter the initial x-coordinate for Agent1: 1
Enter the initial y-coordinate for Agent1: 2
Enter the initial x-coordinate for Agent2: 3
Enter the initial y-coordinate for Agent2: 4
Enter the number of iterations: 3
```

```
. . . . .
. . . . .
. A . . .
. . . . .
. . . A .

Agents' positions:
Agent1: (1, 2)
Agent2: (3, 4)

. . . . .
. . . . .
. . . . .
. A . . .
. . A . .

Agents' positions:
Agent1: (1, 3)
Agent2: (2, 4)

. . . . .
. . . . .
. . . . .
. . . . .
. A . . .

Agents' positions:
Agent1: (1, 4)
Agent2: (1, 4)
```

Practical 13

Aim: Design an application to simulate language parser.

Code:

```
#!pip install spacy
#!python -m spacy download en_core_web_sm

import spacy

class LanguageParser:
    def __init__(self):
        # Load the spaCy English language model
        self.nlp = spacy.load("en_core_web_sm")

    def parse_query(self, query):
        # Process the input query using spaCy
        doc = self.nlp(query)

        # Example: Extracting verbs from the parsed query
        verbs = [token.text for token in doc if token.pos_ == "VERB"]

        return verbs

# Example Usage
if __name__ == "__main__":
    # Create an instance of the LanguageParser
    parser = LanguageParser()

    # User enters a query
    user_query = input("Enter your query: ")
```

```
    # Parse the query
    parsed_result = parser.parse_query(user_query)

    # Display the parsed result
    print("\nParsed Result:")
    if parsed_result:
        for i, verb in enumerate(parsed_result, 1):
            print(f"{i}. {verb}")
    else:
        print("No verbs found in the query.")
```

Output:

```
Enter your query: Artificial intelligence, or AI, refers to the simulation of human intelligence by software-coded heuristics.
 code is prevalent in everything from cloud-based enterprise applications to consumer apps and even embedded firmware.

Parsed Result:
1. refers
2. coded
3. based
4. embedded
```