Step 1.**Design a bot using AIML.**

<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)

**2.Design an Expert system using AIML.**

**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

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)

**3.Implement Bayes Theorem using Python.**

# 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))

**4. Implement Conditional Probability and joint probability using Python.**

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)

**5.Write a program for to implement Rule based system. (Prolog).**

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.

**6.Design a Fuzzy based application using Python/ R**.

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)

**7A.Write an application to stimulate supervised learning model.**

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))

**7B.Write an application to implement UNSUPERISED algoritham..**

# 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()

**8.Write an application to implement clustering algorithm.**

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\_))

**9.Write an application to implement support vector machine algorithm.**

#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))

**10.Simulate artificial neural network model with both feedforward and backpropagation approach**

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)