INDEX

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr.No** | **Name** | **Date** | **Remark** |
| **1** | **Design an Embedded system using AIML** |  |  |
| **2** | **Design a bot using AIML** |  |  |
| **3** | **Implement Bayes Theorem using Python. Description** |  |  |
| **4** | **(a)Implement Conditional Probability (b)write an program to implement Joint Probability** |  |  |
| **5** | **Implement the back propogation algorithm**  **for the above neural network** |  |  |
| **6** | **Design fuzzy based application using python** |  |  |
| **7** | **Write an application to simulate supervised and unsupervised learning model** |  |  |
| **8** | **Write an application to implement clustering**  **Algorithm** |  |  |
| **9** | **Write an application to implement support**  **vector machine algorithm.** |  |  |
| **10** | **write an program to implement Genetic**  **Algorithm** |  |  |

# PRACTICAL -1

### AIM: Design an Embedded system using AIML

#### Bot.py

import os 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)

#### std-startup.xml

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

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

<category>

<pattern>LOAD AIML B</pattern>

<template>

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

</template>

</category>

</aiml>

#### Basic\_chat.aiml

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

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

<category>

<pattern>HELLO \*</pattern>

<template>

Well, hello Nasreen!

</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 I AM</pattern>

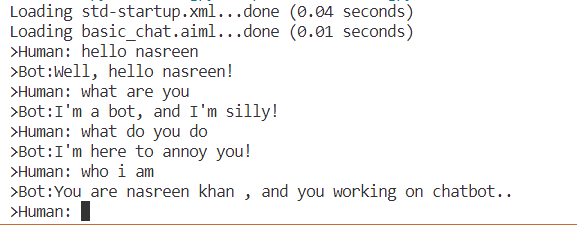
<template>

You are nasreen khan , and you working on chatbot.

</template>

</category>

</aiml>



# PRACTICAL - 2

### Aim : Design a bot using AIML.

**std-startup.xml**

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

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

<category>

<pattern>LOAD</pattern>

<template>

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

</template>

</category>

</aiml>

**basic\_chat.aiml**

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

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

<category>

<pattern>HELLO I AM \*</pattern>

<template> HELLO <set name="username"> <star/> </set> </template>

</category>

<category>

<pattern>I LIKE \* COLOR</pattern>

<template><star index="1"/> is a nice color.</template>

</category>

<category>

<pattern>BYE</pattern>

<template> BYE <get name="username"/> THANKS FOR THE CONVERSATION.

</template>

</category>

</aiml>

#### testbot.py

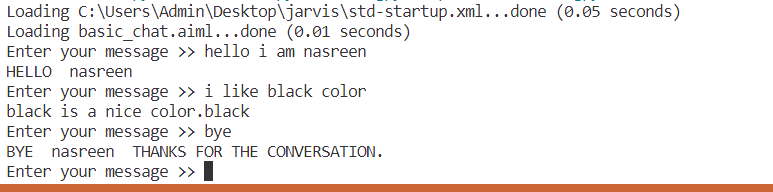
import aiml

kernel = aiml.Kernel() kernel.learn("E:/ai pracs/std- startup.xml")kernel.respond("LOAD")

# Press CTRL-C to break this loop

while True:

print( kernel.respond(input("Enter your message >> ")))



## Practical-3

### AIM: Implement Bayes Theorem using Python.Description

def drug\_user(prob\_th=0.8, sensitivity=0.79, specificity=0.79, prevelance=0.02, verbose=True):

p\_user = prevelance p\_non\_user = 1-prevelance p\_pos\_user = sensitivity p\_neg\_user = 1-specificity num = p\_pos\_user\*p\_user

den = p\_pos\_user\*p\_user+p\_non\_user\*p\_non\_user prob = num/den

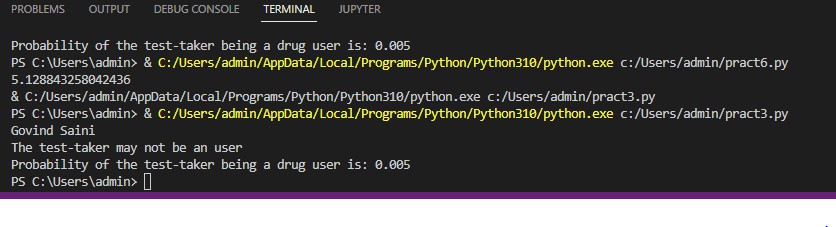
if verbose:

if prob > prob\_th:

print("The test-taker could be an user") else:

print("The test-taker may not be an user") return prob

print("Govind Saini") p=drug\_user(prob\_th=0.5,sensitivity=0.97,specificity=0.95,prevelance=0.005) print("Probability of the test-taker being a drug user is:", round(p,3))



## Practical- 4

### AIM: (A) Implement Conditional Probability using python.

def conditional():

pass\_stats = 0.15

pass\_codingWStats = 0.60

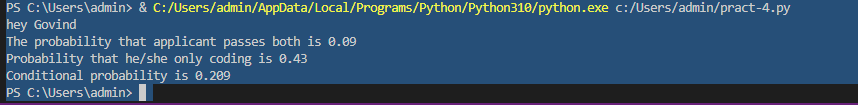
pass\_codingWOStats = 0.40

prob\_both = pass\_stats \* pass\_codingWStats

print("The probability that applicant passes both is", round(prob\_both, 3)) prob\_coding = (prob\_both) +((1-pass\_stats)\*pass\_codingWOStats) print("Probability that he/she only coding is", round(prob\_coding, 3)) stats\_given\_coding = prob\_both/prob\_coding

print("Conditional probability is", round(stats\_given\_coding, 3)) print("hey Govind")

conditional()



### 4 (b) AIM: write an program to implement Joint Probability

import random import pandas as pd

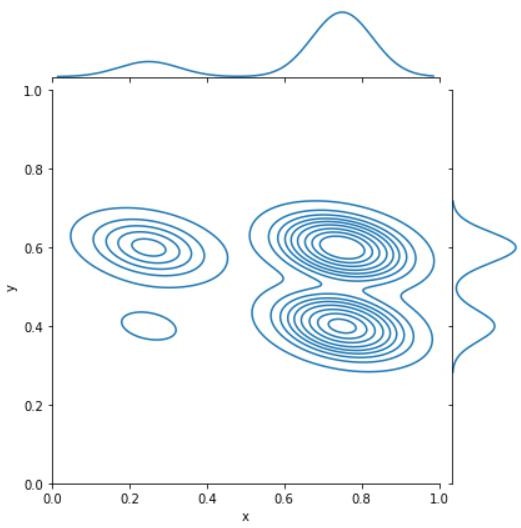
import matplotlib.pyplot as plt import seaborn as sns

samples = 100

x = random.choices(population = [0.25, 0.75], weights = [25, 75], k = samples)

y = random.choices(population = [0.4, 0.6], weights = [4, 6], k = samples) df = pd.DataFrame({'x': x, 'y': y})

sns.jointplot(data = df, x = 'x', y = 'y', kind = 'kde', xlim = (0, 1), ylim = (0, 1)) plt.show()



## Practical - 5

### AIM : Implement the back propogation algorithm for the above neural network

import numpy as np import decimal import math

np.set\_printoptions(precision=2) v1=np.array([0.6, 0.3])

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

w=np.array([-0.2,0.4,0.1]) b1=0.3

b2=0.5 x1=-1 x2=1

alpha=0.25

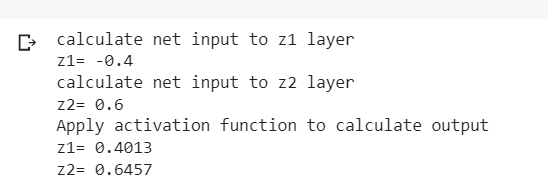
print("calculate net input to z1 layer") zin1=round(b1+ x1\*v1[0]+x2\*v2[0],4) print("z1=",round(zin1,4))

print("calculate net input to z2 layer") zin2=round(b2+ x1\*v1[1]+x2\*v2[1],4) print("z2=",round(zin2,4))

print("Apply activation function to calculate output") z1=1/(1+math.exp(-zin1))

z1=round(z1,4) z2=1/(1+math.exp(-zin2)) z2=round(z2,4) print("z1=",z1)

print("z2=",z2)



print("calculate net input to output layer") yin=w[0]+z1\*w[1]+z2\*w[2] print("yin=",yin)



print("calculate net output") y=1/(1+math.exp(-yin)) print("y=",y)



fyin=y \*(1- y) dk=(1-y)\*fyin print("dk",dk)



dw1= alpha \* dk \* z1 dw2= alpha \* dk \* z2 dw0= alpha \* dk

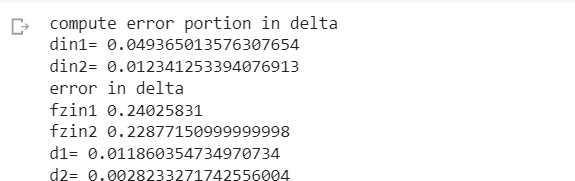
print("compute error portion in delta") din1=dk\* w[1]

din2=dk\* w[2] print("din1=",din1) print("din2=",din2) print("error in delta") fzin1= z1 \*(1-z1) print("fzin1",fzin1) d1=din1\* fzin1

fzin2= z2 \*(1-z2) print("fzin2",fzin2) d2=din2\* fzin2

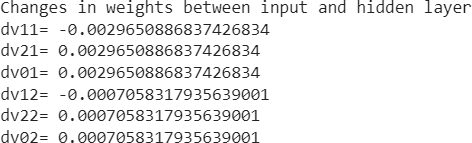
print("d1=",d1)

print("d2=",d2)



print("Changes in weights between input and hidden layer") dv11=alpha \* d1 \* x1

print("dv11=",dv11) dv21=alpha \* d1 \* x2 print("dv21=",dv21) dv01=alpha \* d1 print("dv01=",dv01) dv12=alpha \* d2 \* x1 print("dv12=",dv12) dv22=alpha \* d2 \* x2 print("dv22=",dv22) dv02=alpha \* d2 print("dv02=",dv02)



print("Final weights of network") v1[0]=v1[0]+dv11 v1[1]=v1[1]+dv12

print("v=",v1) v2[0]=v2[0]+dv21 v2[1]=v2[1]+dv22

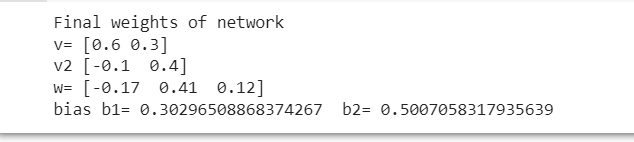
print("v2",v2)

w[1]=w[1]+dw1

w[2]=w[2]+dw2 b1=b1+dv01 b2=b2+dv02 w[0]=w[0]+dw0

print("w=",w)

print("bias b1=",b1, " b2=",b2)



## Practical -6

### AIM : Design fuzzy based application using python

import spacy import numpy as np

import skfuzzy as fuzz

from skfuzzy import control as ctrl

quality = ctrl.Antecedent(np.arange(0,11,1),'quality') service= ctrl.Antecedent(np.arange(0,11,1),'service') tip=ctrl.Consequent(np.arange(0,11,1),'tip') quality.automf(3)

service.automf(3)

tip['low']= fuzz.trimf(tip.universe,[0,0,13]) tip['medium']= fuzz.trimf(tip.universe,[0,13,13]) tip['high']= fuzz.trimf(tip.universe,[13,25,25]) quality['average'].view()

service.view() tip.view()

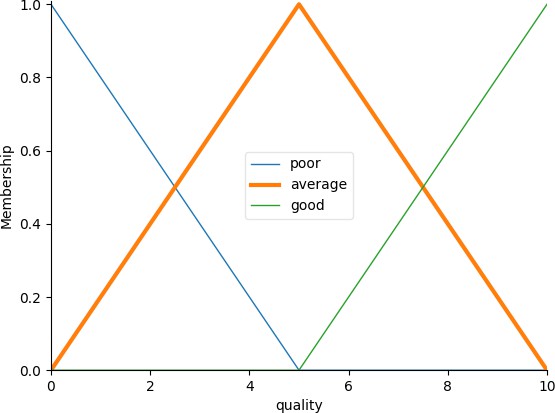
rule1 = ctrl.Rule(quality['poor'] & service['poor'], tip['low']) rule1.view

rule2 = ctrl.Rule(quality['average'] & service['average'], tip['medium']) rule3 = ctrl.Rule(quality['good'] & service['good'], tip['high']) tripping\_ctrl= ctrl.ControlSystem([rule1,rule2,rule3]) tripping=ctrl.ControlSystemSimulation(tripping\_ctrl) tripping.input['quality']=6.5

tripping.input['service']=9.8 tripping.compute()

print(tripping.output['tip']) tip.view(sim=tripping) x=input()

Output:

Figure:1

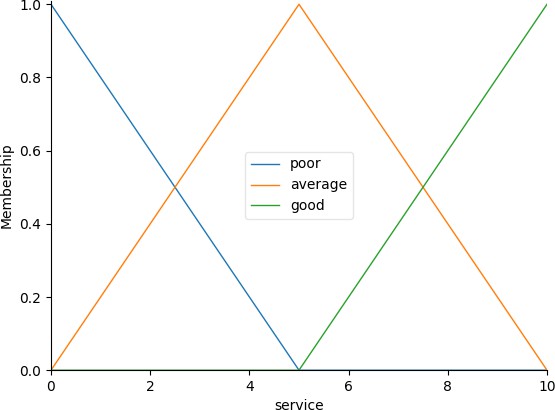
Figure:2

Figure:3

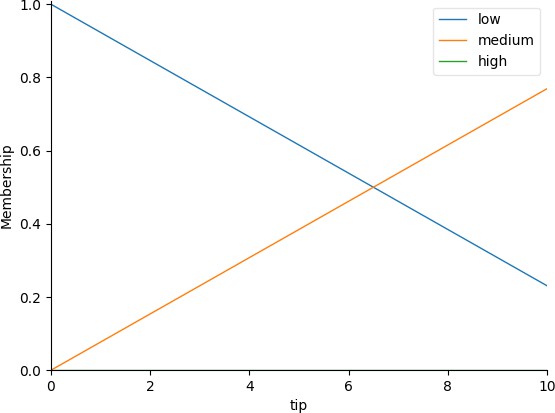
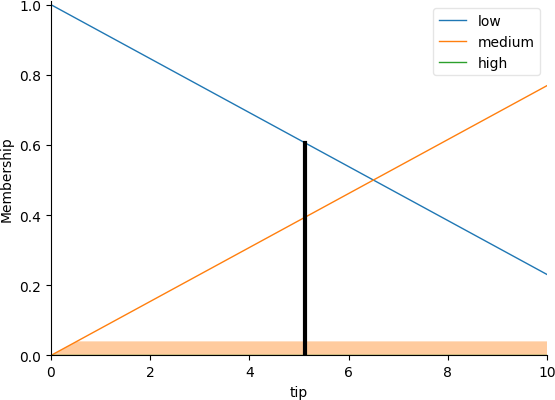


Figure:4



## Practical-7

### AIM: Write an application to simulate supervised and unsupervised learning model

from scipy.cluster.hierarchy import linkage, dendrogram import matplotlib.pyplot as plt

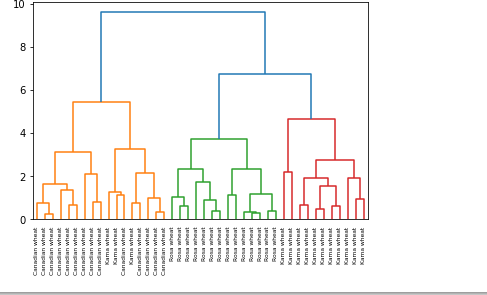
import pandas as pd

seeds\_df = pd.read\_csv("https://raw.githubusercontent.com/vihar/unsupervis ed-learning-with-python/master/seeds-less-rows.csv")

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

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

labels=varieties, leaf\_rotation=90, leaf\_font\_size=6,) plt.show()



## Practical-8

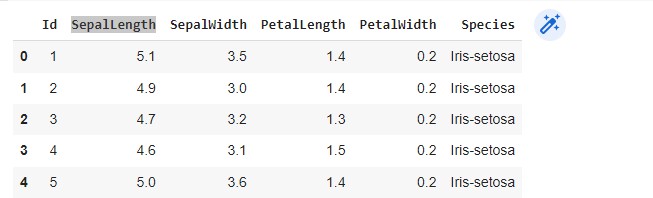
### AIM: Write an application to implement clustering algorithm

import pandas as pd import numpy as np

import matplotlib.pyplot as plt import seaborn as sns

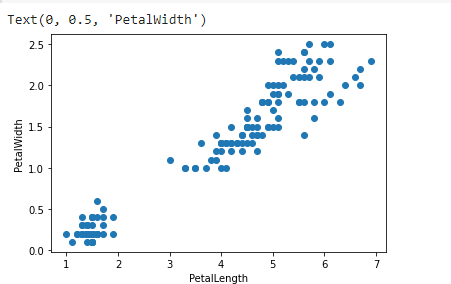
from sklearn import preprocessing from sklearn.cluster import KMeans # importing \*\*cleaned\*\* data

df = pd.read\_csv("/content/sample\_data/Iris.csv") df.head()



# scatterplot

plt.scatter(df["PetalLength"], df["PetalWidth"]) plt.xlabel("PetalLength") plt.ylabel("PetalWidth")

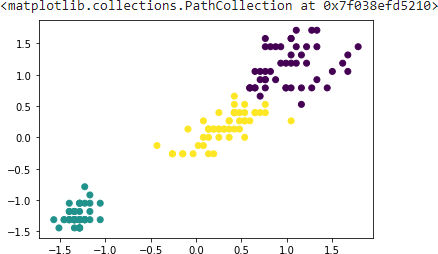


df = df[["PetalLength", "PetalWidth"]]# normalizing inputs

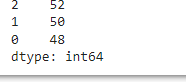
X = preprocessing.scale(df)

model = KMeans(n\_clusters = 3)# fit predict y\_model = model.fit\_predict(X)

plt.scatter(X[:,0], X[:,1], c=model.labels\_, cmap='viridis')



# value counts in each cluster pd.value\_counts(y\_model)

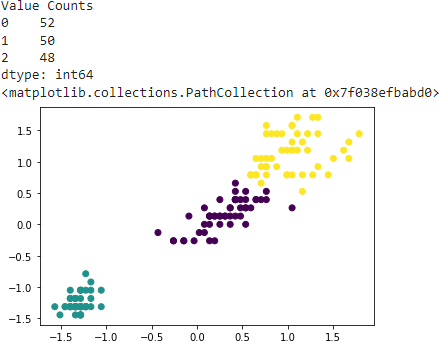


df = df[["PetalLength", "PetalWidth"]] # normalizing inputs

X = preprocessing.scale(df)# instantiate model model = KMeans(n\_clusters = 3)# fit predict y\_model = model.fit\_predict(X)

print("Value Counts") print(pd.value\_counts(y\_model)) # visualize clusters

plt.scatter(X[:,0], X[:,1], c=model.labels\_, cmap='viridis')



# feature selection

df = df[["PetalLength", "PetalWidth"]] # inputs (NOT normalized)

X = df.values

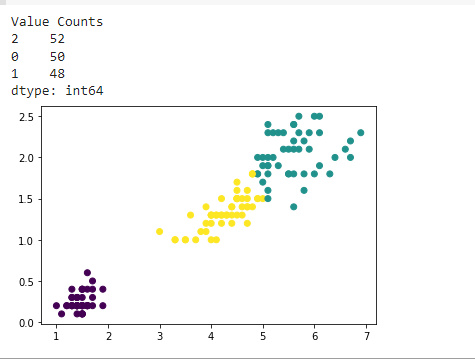
# instantiate model

model = KMeans(n\_clusters = 3) # fit predict

y\_model = model.fit\_predict(X) # visualizing clusters

plt.scatter(X[:,0], X[:,1], c=model.labels\_, cmap='viridis') # counts per cluster

print("Value Counts") print(pd.value\_counts(y\_model))



# import model

from sklearn.cluster import AgglomerativeClustering # feature selection

df = df[["PetalLength", "PetalWidth"]] # normalizing inputs

X = preprocessing.scale(df) # instantiate model

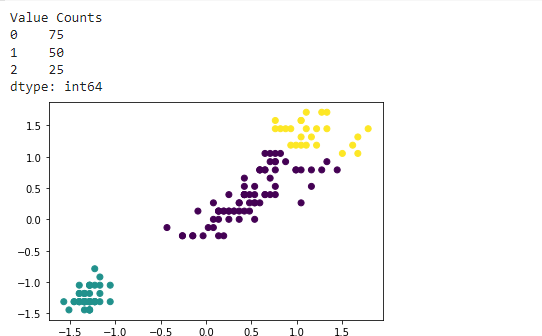
model = AgglomerativeClustering(n\_clusters=3, affinity='euclidean', linkage='w ard')

# fit/predict model

y\_model= model.fit\_predict(X) # ploting clusters

plt.scatter(X[:,0], X[:,1], c=model.labels\_, cmap='viridis') # counts per cluster

print("Value Counts") print(pd.value\_counts(y\_model))



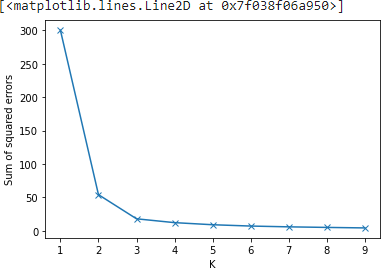
# determine number of clusters using "elbow method" k\_range = range(1,10)

sse = [] # we want to minimize SSE for k in k\_range:

m = KMeans(n\_clusters=k, random\_state=0) m.fit(X)

sse.append(m.inertia\_) plt.xlabel("K")

plt.ylabel("Sum of squared errors") plt.plot(k\_range, sse, marker='x')



# # Working code for clustering

# from numpy import where

# from sklearn.datasets import make\_classification

# from matplotlib import pyplot

# # define dataset

# X, y = make\_classification(n\_samples=1000, n\_features=2, n\_informative=2, n\_redundant=0,n\_clusters\_per\_class=1, random\_state=4)

# # create scatter plot for samples from each class

# for class\_value in range(2):

# # get row indexes for samples with this class

# row\_ix = where(y == class\_value)

# # create scatter of these samples

# pyplot.scatter(X[row\_ix, 0], X[row\_ix, 1])

# # show the plot

# pyplot. show()

## Practical-9

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

# Basic packages import numpy as np import pandas as pd

import matplotlib.pyplot as plt # Sklearn modules & classes

from sklearn.linear\_model import Perceptron, LogisticRegression from sklearn.svm import SVC

from sklearn.model\_selection import train\_test\_split from sklearn.preprocessing import StandardScaler from sklearn import datasets

from sklearn import metrics

# Load the data set; In this example, the breast cancer dataset is loaded. bc = datasets.load\_breast\_cancer()

X = bc.data y = bc.target

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_st ate=1, stratify=y)

sc = StandardScaler() sc.fit(X\_train)

X\_train\_std = sc.transform(X\_train) X\_test\_std = sc.transform(X\_test)

# Instantiate the Support Vector Classifier (SVC) svc = SVC(C=1.0, random\_state=1, kernel='linear') # Fit the model

svc.fit(X\_train\_std, y\_train) # Make the predictions

y\_predict = svc.predict(X\_test\_std)

#Measure the performance

print("Accuracy score %.3f" %metrics.accuracy\_score(y\_test, y\_predict))



## Practical-10

### AIM: write an program to implement Genetic Algorithm

import random

*# Number of individuals in each generation*

POPULATION\_SIZE = 100

*# Valid genes*

GENES = '''abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOP QRSTUVWXYZ 1234567890, .-;:\_!"#%&/()=?@${[]}'''

*# Target string to be generated* TARGET = "I love GeeksforGeeks" class Individual( object ):

*'''*

*Class representing individual in population '''*

def init (self, chromosome): self.chromosome = chromosome self.fitness = self.cal\_fitness()

@classmethod

def mutated\_genes(self):

*'''*

*create random genes for mutation '''*

global GENES

gene = random.choice( GENES ) return gene

@classmethod

def create\_gnome(self):

*'''*

*create chromosome or string of genes '''*

global TARGET

gnome\_len = len( TARGET )

return [self.mutated\_genes() for \_ in range( gnome\_len )]

def mate(self, par2):

*'''*

*Perform mating and produce new offspring '''*

*# chromosome for offspring*

child\_chromosome = []

for gp1, gp2 in zip( self.chromosome, par2.chromosome ):

*# random probability*

prob = random.random()

*# if prob is less than 0.45, insert gene # from parent 1*

if prob < 0.45: child\_chromosome.append( gp1 )

*# if prob is between 0.45 and 0.90, insert # gene from parent 2*

elif prob < 0.90: child\_chromosome.append( gp2 )

*# otherwise insert random gene(mutate), # for maintaining diversity*

else:

child\_chromosome.append( self.mutated\_genes() )

*# create new Individual(offspring) using # generated chromosome for offspring* return Individual( child\_chromosome )

def cal\_fitness(self):

*'''*

*Calculate fittness score, it is the number of characters in string which differ from target string.*

*'''*

global TARGET fitness = 0

for gs, gt in zip( self.chromosome, TARGET ): if gs != gt: fitness += 1

return fitness

*# Driver code*

def main():

global POPULATION\_SIZE

*# current generation*

generation = 1

found = False population = []

*# create initial population*

for \_ in range( POPULATION\_SIZE ): gnome = Individual.create\_gnome()

population.append( Individual( gnome ) ) while not found:

*# sort the population in increasing order of fitness score*

population = sorted( population, key=lambda x: x.fitness )

*# if the individual having lowest fitness score ie.*

*# 0 then we know that we have reached to the target # and break the loop*

if population[0].fitness <= 0: found = True

break

*# Otherwise generate new offsprings for new generation*

new\_generation = []

*# Perform Elitism, that mean 10% of fittest population # goes to the next generation*

s = int( (10 \* POPULATION\_SIZE) / 100 )

new\_generation.extend( population[:s] )

*# From 50% of fittest population, Individuals # will mate to produce offspring*

s = int( (90 \* POPULATION\_SIZE) / 100 )

for \_ in range( s ):

parent1 = random.choice( population[:50] ) parent2 = random.choice( population[:50] ) child = parent1.mate( parent2 ) new\_generation.append( child )

population = new\_generation

print( "Generation: {}\tString: {}\tFitness: {}". \ format( generation,

"".join( population[0].chromosome ), population[0].fitness ) )

generation += 1

print( "Generation: {}\tString: {}\tFitness: {}". \ format( generation,

"".join( population[0].chromosome ), population[0].fitness ) )

if name == ' main ': main()

