Program_1

```
import numpy as np
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics

names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'Class']

# Read dataset to pandas dataframe
dataset = pd.read_csv("/content/8-dataset.csv", names=names)
X = dataset.iloc[:, :-1]
y = dataset.iloc[:, -1]
print(X.head())
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, test_size=0.10)

classifier = KNeighborsClassifier(n_neighbors=5).fit(Xtrain, ytrain)

ypred = classifier.predict(Xtest)

i = 0
print ("\n-----------------------------------------------------------------------")
print ('%-25s %-25s %-25s' % ('Original Label', 'Predicted Label', 'Correct/Wrong'))
print ("-----------------------------------------------------------------------")
for label in ytest:
    print ('%-25s %-25s' % (label, ypred[i]), end="")
    if (label == ypred[i]):
        print (' %-25s' % ('Correct'))
    else:
        print (' %-25s' % ('Wrong'))
    i = i + 1
print ("-----------------------------------------------------------------------")
print("\nConfusion Matrix:\n",metrics.confusion_matrix(ytest, ypred))
print ("-----------------------------------------------------------------------")
print("\nClassification Report:\n",metrics.classification_report(ytest, ypred))
print ("-----------------------------------------------------------------------")
print('Accuracy of the classifer is %0.2f' % metrics.accuracy_score(ytest,ypred))
print ("-----------------------------------------------------------------------")
```

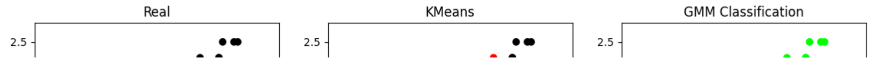
```
   sepal-length  sepal-width  petal-length  petal-width
0           5.1          3.5           1.4          0.2
1           4.9          3.0           1.4          0.2
2           4.7          3.2           1.3          0.2
3           4.6          3.1           1.5          0.2
4           5.0          3.6           1.4          0.2

-----------------------------------------------------------------------
Original Label            Predicted Label           Correct/Wrong
-----------------------------------------------------------------------
Iris-virginica            Iris-virginica            Correct
Iris-setosa               Iris-setosa               Correct
Iris-versicolor           Iris-versicolor           Correct
Iris-setosa               Iris-setosa               Correct
Iris-virginica            Iris-virginica            Correct
Iris-versicolor           Iris-versicolor           Correct
Iris-versicolor           Iris-versicolor           Correct
Iris-virginica            Iris-virginica            Correct
Iris-setosa               Iris-setosa               Correct
Iris-setosa               Iris-setosa               Correct
Iris-setosa               Iris-setosa               Correct
Iris-setosa               Iris-setosa               Correct
Iris-setosa               Iris-setosa               Correct
Iris-virginica            Iris-virginica            Correct
Iris-setosa               Iris-setosa               Correct
-----------------------------------------------------------------------

Confusion Matrix:
 [[8 0 0]
 [0 3 0]
 [0 0 4]]
-----------------------------------------------------------------------

Classification Report:
                  precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00         8
Iris-versicolor       1.00      1.00      1.00         3
 Iris-virginica       1.00      1.00      1.00         4

       accuracy                           1.00        15
      macro avg       1.00      1.00      1.00        15
   weighted avg       1.00      1.00      1.00        15
```

```
-------------------------------------------------------------------------
Accuracy of the classifer is 1.00
-------------------------------------------------------------------------
```

Program_2

```
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture
import sklearn.metrics as metrics
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

names = ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width', 'Class']

dataset = pd.read_csv("/content/8-dataset.csv", names=names)

X = dataset.iloc[:, :-1]

label = {'Iris-setosa': 0,'Iris-versicolor': 1, 'Iris-virginica': 2}

y = [label[c] for c in dataset.iloc[:, -1]]

plt.figure(figsize=(14,7))
colormap=np.array(['red','lime','black'])

# REAL PLOT
plt.subplot(1,3,1)
plt.title('Real')
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y])

# K-PLOT
model=KMeans(n_clusters=3, random_state=0).fit(X)
plt.subplot(1,3,2)
plt.title('KMeans')
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[model.labels_])

print('The accuracy score of K-Mean: ',metrics.accuracy_score(y, model.labels_))
print('The Confusion matrixof K-Mean:\n',metrics.confusion_matrix(y, model.labels_))

# GMM PLOT
gmm=GaussianMixture(n_components=3, random_state=0).fit(X)
y_cluster_gmm=gmm.predict(X)
plt.subplot(1,3,3)
plt.title('GMM Classification')
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y_cluster_gmm])

print('The accuracy score of EM: ',metrics.accuracy_score(y, y_cluster_gmm))
print('The Confusion matrix of EM:\n ',metrics.confusion_matrix(y, y_cluster_gmm))
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning
  warnings.warn(
The accuracy score of K-Mean:  0.24
The Confusion matrixof K-Mean:
 [[ 0 50  0]
 [48  0  2]
 [14  0 36]]
The accuracy score of EM:  0.36666666666666664
The Confusion matrix of EM:
  [[50  0  0]
 [ 0  5 45]
 [ 0 50  0]]
```



Program_3


```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

def kernel(point, xmat, k):
    m,n = np.shape(xmat)
    weights = np.mat(np.eye((m)))
    for j in range(m):
        diff = point - X[j]
        weights[j,j] = np.exp(diff*diff.T/(-2.0*k**2))
    return weights

def localWeight(point, xmat, ymat, k):
    wei = kernel(point,xmat,k)
    W = (X.T*(wei*X)).I*(X.T*(wei*ymat.T))
    return W

def localWeightRegression(xmat, ymat, k):
    m,n = np.shape(xmat)
    ypred = np.zeros(m)
    for i in range(m):
        ypred[i] = xmat[i]*localWeight(xmat[i],xmat,ymat,k)
    return ypred

# load data points
data = pd.read_csv('/content/10-dataset.csv')
bill = np.array(data.total_bill)
tip = np.array(data.tip)

#preparing and add 1 in bill
mbill = np.mat(bill)
mtip = np.mat(tip)

m= np.shape(mbill)[1]
one = np.mat(np.ones(m))
X = np.hstack((one.T,mbill.T))

#set k here
ypred = localWeightRegression(X,mtip,0.5)
SortIndex = X[:,1].argsort(0)
xsort = X[SortIndex][:,0]

fig = plt.figure()
ax = fig.add_subplot(1,1,1)
ax.scatter(bill,tip, color='green')
ax.plot(xsort[:,1],ypred[SortIndex], color = 'red', linewidth=5)
plt.xlabel('Total bill')
plt.ylabel('Tip')
plt.show();
```

Program_4

```
import numpy as np
inputNeurons=2
hiddenlayerNeurons=4
outputNeurons=2
iteration=6000
input = np.random.randint(1,5,inputNeurons)
output = np.array([1.0,0.0])
hidden_layer=np.random.rand(1,hiddenlayerNeurons)
hidden_biass=np.random.rand(1,hiddenlayerNeurons)
output_bias=np.random.rand(1,outputNeurons)
hidden_weights=np.random.rand(inputNeurons,hiddenlayerNeurons)
output_weights=np.random.rand(hiddenlayerNeurons,outputNeurons)
def sigmoid (layer):
  return 1/(1 + np.exp(-layer))
def gradient(layer):
  return layer*(1-layer)
for i in range(iteration):

  hidden_layer=np.dot(input,hidden_weights)
  hidden_layer=sigmoid(hidden_layer+hidden_biass)
  output_layer=np.dot(hidden_layer,output_weights)
  output_layer=sigmoid(output_layer+output_bias)
  error = (output-output_layer)
  gradient_outputLayer=gradient(output_layer)
  error_terms_output=gradient_outputLayer * error
  error_terms_hidden=gradient(hidden_layer)*np.dot(error_terms_output,output_weights.T)
  gradient_hidden_weights =np.dot(input.reshape(inputNeurons,1),error_terms_hidden.reshape(1,hiddenlayerNeurons))
  gradient_ouput_weights =np.dot(hidden_layer.reshape(hiddenlayerNeurons,1),error_terms_output.reshape(1,outputNeurons))
  hidden_weights = hidden_weights + 0.05*gradient_hidden_weights
  output_weights = output_weights + 0.05*gradient_ouput_weights
  if i<50 or i>iteration-50:
    print("**********************")
    print("iteration:",i,"::::",error)
    print("###output########",output_layer)
```

```
**********************
iteration: 0 :::: [[ 0.12594904 -0.95089727]]
###output######## [[0.87405096 0.95089727]]
**********************
iteration: 1 :::: [[ 0.12572517 -0.9505321 ]]
###output######## [[0.87427483 0.9505321 ]]
**********************
iteration: 2 :::: [[ 0.1255029  -0.95016202]]
###output######## [[0.8744971  0.95016202]]
**********************
iteration: 3 :::: [[ 0.1252822  -0.94978692]]
###output######## [[0.8747178  0.94978692]]
**********************
iteration: 4 :::: [[ 0.12506308 -0.94940672]]
###output######## [[0.87493692 0.94940672]]
**********************
iteration: 5 :::: [[ 0.12484552 -0.94902132]]
###output######## [[0.87515448 0.94902132]]
**********************
iteration: 6 :::: [[ 0.12462951 -0.94863062]]
###output######## [[0.87537049 0.94863062]]
**********************
iteration: 7 :::: [[ 0.12441505 -0.94823452]]
###output######## [[0.87558495 0.94823452]]
**********************
iteration: 8 :::: [[ 0.12420211 -0.94783292]]
###output######## [[0.87579789 0.94783292]]
**********************
iteration: 9 :::: [[ 0.1239907  -0.94742571]]
###output######## [[0.8760093  0.94742571]]
**********************
iteration: 10 :::: [[ 0.12378081 -0.94701279]]
###output######## [[0.87621919 0.94701279]]
**********************
iteration: 11 :::: [[ 0.12357241 -0.94659404]]
###output######## [[0.87642759 0.94659404]]
**********************
iteration: 12 :::: [[ 0.12336552 -0.94616935]]
###output######## [[0.87663448 0.94616935]]
```

```
**********************
iteration: 13 :::: [[ 0.1231601  -0.94573861]]
###output######## [[0.8768399  0.94573861]]
**********************
iteration: 14 :::: [[ 0.12295617 -0.9453017 ]]
###output######## [[0.87704383 0.9453017 ]]
**********************
iteration: 15 :::: [[ 0.12275371 -0.94485849]]
###output######## [[0.87724629 0.94485849]]
**********************
iteration: 16 :::: [[ 0.1225527  -0.94440886]]
###output######## [[0.8774473  0.94440886]]
**********************
iteration: 17 :::: [[ 0.12235315 -0.94395269]]
###output######## [[0.87764685 0.94395269]]
**********************
iteration: 18 :::: [[ 0.12215504 -0.94348984]]
###output######## [[0.87784496 0.94348984]]
**********************
```

Program_5

```
# Python3 program to create target string, starting from
# random string using 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 fitness 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()
```

```
Generation: 1   String: R Yo"U3ry$-5{d$VAZdC    Fitness: 18
Generation: 2   String: R Yo"U3ry$-5{d$VAZdC    Fitness: 18
Generation: 3   String: ac U$)-GfBky"J}GN)_X    Fitness: 17
Generation: 4   String: m qonvU3aBk9j1;-lt6s    Fitness: 16
Generation: 5   String: m lo]sUBaBk9F7V-c,gs    Fitness: 15
Generation: 6   String: m bo)e [47k dq@&ln]s    Fitness: 14
Generation: 7   String: I 2one/GjnkZm?;reB1K    Fitness: 13
Generation: 8   String: I 2one/GjnkZm?;reB1K    Fitness: 13
Generation: 9   String: I 2one/GjnkZm?;reB1K    Fitness: 13
Generation: 10  String: I lo]e BfB7s2}VGljGs    Fitness: 11
Generation: 11  String: I lo]e BfB7s2}VGljGs    Fitness: 11
Generation: 12  String: I lo]e BfB7s2}VGljGs    Fitness: 11
Generation: 13  String: mRlov7 GaNkZ9}rGejks    Fitness: 9
Generation: 14  String: mRlov7 GaNkZ9}rGejks    Fitness: 9
Generation: 15  String: J@lovegGaWkscoMGe3ks    Fitness: 8
Generation: 16  String: J@lovegGaWkscoMGe3ks    Fitness: 8
Generation: 17  String: I lov gG!WkscxrGefks    Fitness: 7
Generation: 18  String: I lov gG!WkscxrGefks    Fitness: 7
Generation: 19  String: J love Gfnks yrGe ks    Fitness: 6
Generation: 20  String: J love Gfnks yrGe ks    Fitness: 6
Generation: 21  String: J love Gfnks yrGe ks    Fitness: 6
Generation: 22  String: J love Gfnks yrGe ks    Fitness: 6
Generation: 23  String: I love GaBksco7Gesks    Fitness: 5
Generation: 24  String: I love GaBksco7Gesks    Fitness: 5
Generation: 25  String: I love GaBksco7Gesks    Fitness: 5
```

```
Generation: 26  String: I love GaBksco7Gesks    Fitness: 5
Generation: 27  String: I[love Geeka5orGejks    Fitness: 4
Generation: 28  String: I[love Geeka5orGejks    Fitness: 4
Generation: 29  String: I[love Geeka5orGejks    Fitness: 4
Generation: 30  String: I love GseksforGexks    Fitness: 2
Generation: 31  String: I love GseksforGexks    Fitness: 2
Generation: 32  String: I love GseksforGexks    Fitness: 2
Generation: 33  String: I love GseksforGexks    Fitness: 2
Generation: 34  String: I love GeeksforGejks    Fitness: 1
Generation: 35  String: I love GeeksforGejks    Fitness: 1
Generation: 36  String: I love GeeksforGejks    Fitness: 1
Generation: 37  String: I love GeeksforGejks    Fitness: 1
Generation: 38  String: I love GeeksforGejks    Fitness: 1
Generation: 39  String: I love GeeksforGejks    Fitness: 1
Generation: 40  String: I love GeeksforGejks    Fitness: 1
Generation: 41  String: I love GeeksforGejks    Fitness: 1
Generation: 42  String: I love GeeksforGejks    Fitness: 1
Generation: 43  String: I love GeeksforGejks    Fitness: 1
Generation: 44  String: I love GeeksforGejks    Fitness: 1
Generation: 45  String: I love GeeksforGejks    Fitness: 1
Generation: 46  String: I love GeeksforGejks    Fitness: 1
Generation: 47  String: I love GeeksforGejks    Fitness: 1
Generation: 48  String: I love GeeksforGejks    Fitness: 1
Generation: 49  String: I love GeeksforGejks    Fitness: 1
Generation: 50  String: I love GeeksforGejks    Fitness: 1
Generation: 51  String: I love GeeksforGejks    Fitness: 1
Generation: 52  String: I love GeeksforGejks    Fitness: 1
Generation: 53  String: I love GeeksforGejks    Fitness: 1
Generation: 54  String: I love GeeksforGejks    Fitness: 1
Generation: 55  String: I love GeeksforGejks    Fitness: 1
Generation: 56  String: I love GeeksforGejks    Fitness: 1
Generation: 57  String: I love GeeksforGejks    Fitness: 1
Generation: 58  String: I love GeeksforGejks    Fitness: 1
```

Program_6

```
import numpy as np
# Estado terminal
terminal = 5
# Possiveis acoes
actions = ['UP','DW','LF','RG']
# Recompensas
rws = np.array([-1]*6)
rws[5] = 10
# Duas trajetorias
paths = [(0, ['UP','UP','UP','RG']), (4, ['RG','RG','LF','UP'])]
# Constantes
alpha = 0.5
gamma = 0.8

def print_value(value):
  print('[' + str(value[2]) + ' ' + str(value[5]))
  print(str(value[1]) + ' ' + str(value[4]))
  print(str(value[0]) + ' ' + str(value[3]) + ']\n')

def update_value(value, state, action):
  index = actions.index(action)
  next_state = state
  rw = 0
  if action == 'UP':
      if state == 2 or state == 5:
        rw = -10
      else:
        next_state = state + 1

  elif action == 'DW':
    if state == 0 or state == 3:
      rw = -10
    else:
      next_state = state - 1

  elif action == 'LF':
    if state == 0 or state == 1 or state == 2:
      rw = -10
    else:
      next_state = state - 3

  elif action == 'RG':
    if state == 3 or state == 4 or state == 5:
      rw = -10
    else:
      next_state = state + 3

  if rw == 0:
    rw = rws[next_state]
```

```
    rw = rws[next_state]

    value[index][state] = value[index][state] + alpha * (rw + gamma * max(value[i][next_state] for i in range(4)) - value[index][state])
    return value, next_state

def return_policy(value):
  policy = np.array([' ']*6)
  policy[5] = '+10'

  for state in range(5):
    policy[state] = actions[np.argmax([value[action][state] for action in range(4)])]

  print(policy[2] + ' ' + policy[5])
  print(policy[1] + ' ' + policy[4])
  print(policy[0] + ' ' + policy[3]+ '\n')

def main():
  # Inicializar matriz Q com valores 0, considerando as quatro acoes
  value = [np.zeros(6),np.zeros(6),np.zeros(6),np.zeros(6)]
  for i in range(len(paths)):
    state = paths[i][0]
    actions = paths[i][1]
    for action in actions:
      value, state = update_value(value, state, action)
      if state == terminal:
        break
    # Acao UP
    print_value(value[0])
    # Acao DW
    print_value(value[1])
    # Acao LF
    print_value(value[2])
    # Acao RG
    print_value(value[3])
    # Politica
    return_policy(value)

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

```
[-5.0 0.0
-0.5 0.0
-0.5 0.0]

[0.0 0.0
0.0 0.0
0.0 0.0]

[0.0 0.0
0.0 0.0
0.0 0.0]

[5.0 0.0
0.0 0.0
0.0 0.0]

R +
D U
D U

[-5.0 0.0
1.25 0.0
-0.5 0.0]

[0.0 0.0
0.0 0.0
0.0 0.0]

[0.0 0.0
0.0 -0.5
0.0 0.0]

[5.0 0.0
0.0 -7.5
0.0 0.0]

R +
U U
D U
```

Start coding or generate with AI.