Lab File

On

Soft Computing and its applications

Submitted To

Dr Malay Ranjan Tripathy

Amity University Uttar Pradesh

In partial fulfilment of the requirements for the award degree of

Bachelor of Technology

In

Artificial Intelligence

by

Srishti Singh

A023119820035

DEPARTMENT OF ARTIFICIAL INTELLIGENCE

AMITY SCHOOL OF ENGINEERING AND TECHNOLOGY

AMITY UNIVERSITY UTTAR PRADESH

INDEX

S.NO	AIM	DATE	SIGNATURE
1.	Create a perceptron with appropriate no. of inputs and outputs. Train, it using fixed increment learning algorithm until no change in weights is required. Output the final weights.		
2.	Implement TSP using GA		
3.	Solve Greg Viot’s fuzzy cruise controller using MATLAB Fuzzy logic toolbox.		
4.	Implement Union, Intersection, Complement and Difference operations on fuzzy sets. Also create fuzzy relation by Cartesian product of any two fuzzy sets and perform max min composition on any two fuzzy relations.		
5.	Create a simple ADALINE network with appropriate no. of input and output nodes. Train it using delta learning rule until no change in weights is required. Output the final weights.		

EXPERIMENT 1

AIM: Create a perceptron with appropriate no. of inputs and outputs. Train, it using fixed increment learning algorithm until no change in weights is required. Output the final weights.

SOGTWARE USED: JUPYTER NOTEBOOK

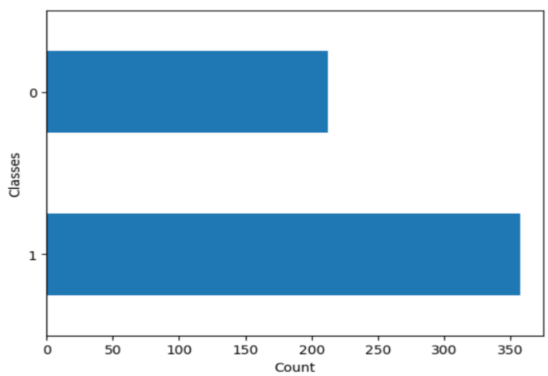
PROGRAM CODE:

```
#import packages
import sklearn.datasets
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
#load the breast cancer data
breast_cancer = sklearn.datasets.load_breast_cancer()
#convert the data to pandas dataframe.
data = pd.DataFrame(breast_cancer.data, columns = breast_cancer.feature_names)
data["class"] = breast_cancer.target
data.head()
data.describe()
#plotting a graph to see class imbalance
data['class'].value_counts().plot(kind = "barh")
plt.xlabel("Count")
plt.ylabel("Classes")
plt.show()
from sklearn.preprocessing import MinMaxScaler
#perform scaling on the data.
X = data.drop("class", axis = 1)
Y = data["class"]
mnscaler = MinMaxScaler()
X = mnscaler.fit_transform(X)
X = pd.DataFrame(X, columns=data.drop("class",axis = 1).columns)
#train test split.
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size = 0.1, stratify = Y, random_state = 1)
class Perceptron:
 #constructor
 def __init__ (self):
   self.w = None
   self.b = None

 #model
 def model(self, x):
   return 1 if (np.dot(self.w, x) >= self.b) else 0
```

```
  #predictor to predict on the data based on w
  def predict(self, X):
    Y = []
    for x in X:
      result = self.model(x)
      Y.append(result)
    return np.array(Y)

  def fit(self, X, Y, epochs = 1, lr = 1):
    self.w = np.ones(X.shape[1])
    self.b = 0
    accuracy = {}
    max_accuracy = 0
    wt_matrix = []
    #for all epochs
    for i in range(epochs):
      for x, y in zip(X, Y):
        y_pred = self.model(x)
        if y == 1 and y_pred == 0:
          self.w = self.w + lr * x
          self.b = self.b - lr * 1
        elif y == 0 and y_pred == 1:
          self.w = self.w - lr * x
          self.b = self.b + lr * 1

      wt_matrix.append(self.w)
      accuracy[i] = accuracy_score(self.predict(X), Y)
      if (accuracy[i] > max_accuracy):
        max_accuracy = accuracy[i]
        chkptw = self.w
        chkptb = self.b
    #checkpoint (Save the weights and b value)
    self.w = chkptw
    self.b = chkptb
    print(max_accuracy)
    #plot the accuracy values over epochs
    plt.plot(accuracy.values())
    plt.xlabel("Epoch #")
    plt.ylabel("Accuracy")
    plt.ylim([0, 1])
    plt.show()
    #return the weight matrix, that contains weights over all epochs
    return np.array(wt_matrix)
perceptron = Perceptron()
```

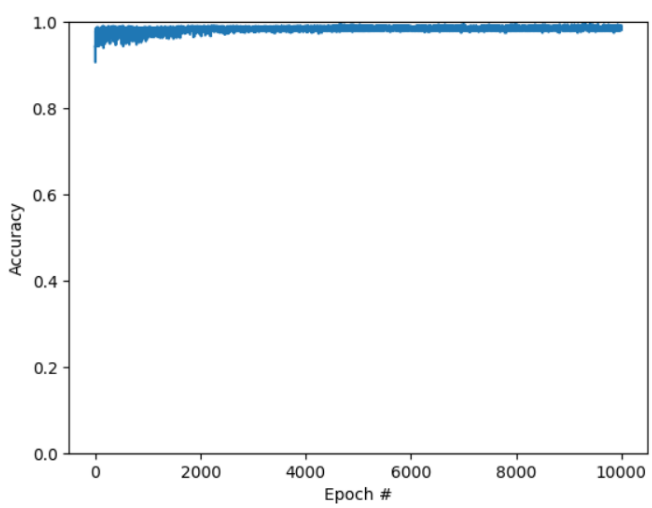
```
#epochs = 10000 and lr = 0.3
wt_matrix = perceptron.fit(X_train, Y_train, 10000, 0.3)
#making predictions on test data
Y_pred_test = perceptron.predict(X_test)

#checking the accuracy of the model
print(accuracy_score(Y_pred_test, Y_test))
```

RESULT:




```
<class 'pandas.core.series.Series'>
Int64Index: 512 entries, 430 to 161
Series name: class
Non-Null Count  Dtype
--------------  -----
512 non-null    float64
dtypes: float64(1)
memory usage: 8.0 KB
```

```
0.994140625
```




```
0.9824561403508771
```

CONCLUSION: The perceptron algorithm was successfully implemented.

CRITERIA	TOTAL MARKS	MARKS OBTAINED	COMMENTS
Concept (A)	2		
Implementation (B)	2		
Performance (C)	2		
Total	6		

EXPERIMENT 2

AIM: Implement TSP using GA

SOFTWARE USED: JUPYTER NOTEBOOK

PROGRAM CODE:

1. from random import randint
2. INT_MAX = 2147483647
3. # Number of cities in TSP
4. V = 5
5. # Names of the cities
6. GENES = "ABCDE"
7. # Starting Node Value
8. START = 0
9. # Initial population size for the algorithm
10. POP_SIZE = 10
11. # Structure of a GNOME
12. # defines the path traversed
13. # by the salesman while the fitness value
14. # of the path is stored in an integer
15. class individual:
    - a. def __init__(self) -> None:
        - i. self.gnome = ""
        - ii. self.fitness = 0
    - b. def __lt__(self, other):
        - i. return self.fitness < other.fitness
    - c. def __gt__(self, other):
        - i. return self.fitness > other.fitness
16. # Function to return a random number
17. # from start and end
18. def rand_num(start, end):
    - a. return randint(start, end-1)
19. # Function to check if the character
20. # has already occurred in the string
21. def repeat(s, ch):
    - a. for i in range(len(s)):
        - i. if s[i] == ch:
            - 1. return True
    - b. return False
22. # Function to return a mutated GNOME
23. # Mutated GNOME is a string
24. # with a random interchange
25. # of two genes to create variation in species

26. def mutatedGene(gnome):
    a. gnome = list(gnome)
    b. while True:
        i. r = rand_num(1, V)
        ii. r1 = rand_num(1, V)
        iii. if r1 != r:
            1. temp = gnome[r]
            2. gnome[r] = gnome[r1]
            3. gnome[r1] = temp
            4. break
    c. return ''.join(gnome)
27. # Function to return a valid GNOME string
28. # required to create the population
29. def create_gnome():
    a. gnome = "0"
    b. while True:
        i. if len(gnome) == V:
            1. gnome += gnome[0]
            2. break
        ii. temp = rand_num(1, V)
        iii. if not repeat(gnome, chr(temp + 48)):
            1. gnome += chr(temp + 48)
    c. return gnome
30. # Function to return the fitness value of a gnome.
31. # The fitness value is the path length
32. # of the path represented by the GNOME.
33. def cal_fitness(gnome):
    a. mp = [
        i. [0, 2, INT_MAX, 12, 5],
        ii. [2, 0, 4, 8, INT_MAX],
        iii. [INT_MAX, 4, 0, 3, 3],
        iv. [12, 8, 3, 0, 10],
        v. [5, INT_MAX, 3, 10, 0],
    b.]
    c. f = 0
    d. for i in range(len(gnome) - 1):
        i. if mp[ord(gnome[i]) - 48][ord(gnome[i + 1]) - 48] == INT_MAX:
            1. return INT_MAX
        ii. f += mp[ord(gnome[i]) - 48][ord(gnome[i + 1]) - 48]
    e. return f
34. # Function to return the updated value
35. # of the cooling element.
36. def cooldown(temp):
    a. return (90 * temp) / 100

37. # Comparator for GNOME struct.
38. # def lessthan(individual t1,
39. #   individual t2)
40. # :
41. #   return t1.fitness < t2.fitness
42. # Utility function for TSP problem.
43. def TSPUtil(mp):
    a. # Generation Number
    b. gen = 1
    c. # Number of Gene Iterations
    d. gen_thres = 5
    e. population = []
    f. temp = individual()
    g. # Populating the GNOME pool.
    h. for i in range(POP_SIZE):
        i. temp.gnome = create_gnome()
        ii. temp.fitness = cal_fitness(temp.gnome)
        iii. population.append(temp)
    i. print("\nInitial population: \nGNOME   FITNESS VALUE\n")
    j. for i in range(POP_SIZE):
        i. print(population[i].gnome, population[i].fitness)
    k. print()
    l. found = False
    m. temperature = 10000
    n. # Iteration to perform
    o. # population crossing and gene mutation.
    p. while temperature > 1000 and gen <= gen_thres:
        i. population.sort()
        ii. print("\nCurrent temp: ", temperature)
        iii. new_population = []
        iv. for i in range(POP_SIZE):
            1. p1 = population[i]
            2. while True:
                a. new_g = mutatedGene(p1.gnome)
                b. new_gnome = individual()
                c. new_gnome.gnome = new_g
                d. new_gnome.fitness = cal_fitness(new_gnome.gnome)
                e. if new_gnome.fitness <= population[i].fitness:
                    i. new_population.append(new_gnome)
                    ii. break
                f. else:
                    i. # Accepting the rejected children at
                    ii. # a possible probability above threshold.
                    iii. prob = pow(

            1. 2.7,
            2. -1((float)(new_gnome.fitness - population[i].fitness)/ temperature),
            3.)
         iv. if prob > 0.5:
            1. new_population.append(new_gnome)
            2. break
      v. temperature = cooldown(temperature)
      vi. population = new_population
      vii. print("Generation", gen)
      viii. print("GNOME   FITNESS VALUE")
      ix. for i in range(POP_SIZE):
         1. print(population[i].gnome, population[i].fitness)
      x. gen += 1

44. if __name__ == "__main__":
    a. mp = [
       i. [0, 2, INT_MAX, 12, 5],
       ii. [2, 0, 4, 8, INT_MAX],
       iii. [INT_MAX, 4, 0, 3, 3],
       iv. [12, 8, 3, 0, 10],
       v. [5, INT_MAX, 3, 10, 0],
    b.]
    c. TSPUtil(mp)

RESULT:

```
Initial population:
GNOME     FITNESS VALUE

023140 2147483647
023140 2147483647
023140 2147483647
023140 2147483647
023140 2147483647
023140 2147483647
023140 2147483647
023140 2147483647
023140 2147483647
023140 2147483647


Current temp:  10000
Generation 1
GNOME     FITNESS VALUE
021340 2147483647
023410 2147483647
021340 2147483647
032140 2147483647
021340 2147483647
021340 2147483647
013240 21
021340 2147483647
021340 2147483647
013240 21

Current temp:  9000.0
Generation 2
GNOME     FITNESS VALUE
012340 24
031240 32
```

```
Current temp:  7290.0
Generation 4
GNOME     FITNESS VALUE
031240 32
043210 24
012340 24
042130 32
042130 32
012340 24
031240 32
023140 2147483647
023140 2147483647
024310 2147483647

Current temp:  6561.0
Generation 5
GNOME     FITNESS VALUE
013240 21
012430 31
042310 21
013240 21
042310 21
012430 31
013240 21
021340 2147483647
032140 2147483647
014320 2147483647
```

CONCLUSION: Implementation of TSP by genetic algorithm was successfully done.

CRITERIA	TOTAL MARKS	MARKS OBTAINED	COMMENTS
Concept (A)	2		
Implementation (B)	2		
Performance (C)	2		
Total	6		

EXPERIMENT 3

AIM: Solve Greg Viot's fuzzy cruise controller using MATLAB Fuzzy logic toolbox.

SOFTWARE USED: JUPYTER NOTEBOOK

PROGRAM CODE:

1. import numpy as np
2. import skfuzzy.control as ctrl
3. universe = np.linspace(-2, 2, 5)
4. error = ctrl.Antecedent(universe, 'error')
5. delta = ctrl.Antecedent(universe, 'delta')
6. output = ctrl.Consequent(universe, 'output')
7. names = ['nb', 'ns', 'ze', 'ps', 'pb']
8. error.automf(names=names)
9. delta.automf(names=names)
10. output.automf(names=names)
11. rule0 = ctrl.Rule(antecedent=((error['nb'] & delta['nb']) |
        i. (error['ns'] & delta['nb']) |
        ii. (error['nb'] & delta['ns'])),
    b. consequent=output['nb'], label='rule nb')
12. rule1 = ctrl.Rule(antecedent=((error['nb'] & delta['ze']) |
        i. (error['nb'] & delta['ps']) |
        ii. (error['ns'] & delta['ns']) |
        iii. (error['ns'] & delta['ze']) |
        iv. (error['ze'] & delta['ns']) |
        v. (error['ze'] & delta['nb']) |
        vi. (error['ps'] & delta['nb'])),
    b. consequent=output['ns'], label='rule ns')
13. rule2 = ctrl.Rule(antecedent=((error['nb'] & delta['pb']) |
        i. (error['ns'] & delta['ps']) |
        ii. (error['ze'] & delta['ze']) |
        iii. (error['ps'] & delta['ns']) |
        iv. (error['pb'] & delta['nb'])),
    b. consequent=output['ze'], label='rule ze')
14. rule3 = ctrl.Rule(antecedent=((error['ns'] & delta['pb']) |
        i. (error['ze'] & delta['pb']) |
        ii. (error['ze'] & delta['ps']) |
        iii. (error['ps'] & delta['ps']) |
        iv. (error['ps'] & delta['ze']) |
        v. (error['pb'] & delta['ze']) |
        vi. (error['pb'] & delta['ns'])),
    b. consequent=output['ps'], label='rule ps')
15. rule4 = ctrl.Rule(antecedent=((error['ps'] & delta['pb']) |
        i. (error['pb'] & delta['pb']) |

ii. (error['pb'] & delta['ps'])),

b. consequent=output['pb'], label='rule pb')

16. system = ctrl.ControlSystem(rules=[rule0, rule1, rule2, rule3, rule4])
17. sim = ctrl.ControlSystemSimulation(system, flush_after_run=6 * 6 + 1)
18. upsampled = np.linspace(-2, 2, 6)
19. x, y = np.meshgrid(upsampled, upsampled)
20. z = np.zeros_like(x)
21. print("\tError\t\tDelta\t\tOutput\n")
22. for i in range(6):
23. for j in range(6):
24. sim.input['error'] = x[i, j]
25. sim.input['delta'] = y[i, j]
26. sim.compute()
27. z[i, j] = sim.output['output']
28. print("\t",'{0:.2f}'.format(x[i,j]),"\t\t",'{0:.2f}'.format(y[i,j]),"\t\t",'{0:.2f}'.format(z[i,j]),"\n")
29. import matplotlib.pyplot as plt
30. from mpl_toolkits.mplot3d import Axes3D
31. fig = plt.figure(figsize=(8, 8))
32. ax = fig.add_subplot(111, projection='3d')
33. surf = ax.plot_surface(x, y, z, rstride=1, cstride=1, cmap='viridis',
    i. linewidth=0.4, antialiased=True)
34. cset = ax.contourf(x, y, z, zdir='z', offset=-2.5, cmap='viridis', alpha=0.5)
35. cset = ax.contourf(x, y, z, zdir='x', offset=3, cmap='viridis', alpha=0.5)
36. cset = ax.contourf(x, y, z, zdir='y', offset=3, cmap='viridis', alpha=0.5)
37. ax.view_init(30, 200)
38. plt.show()

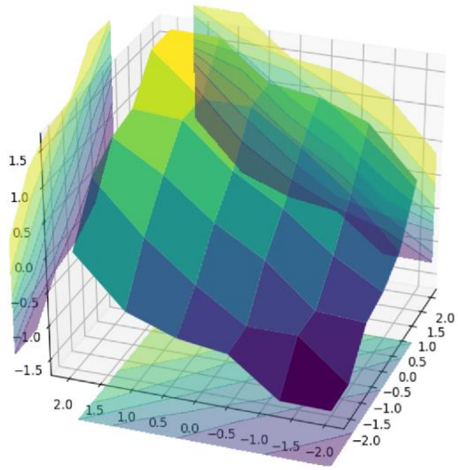
RESULT: define the complex set of rules in the fuzzy system

```
rule0 = ctrl.Rule(antecedent=((error['nb'] & delta['nb']) |
                              (error['ns'] & delta['nb']) |
                              (error['nb'] & delta['ns'])),
                  consequent=output['nb'], label='rule nb')

rule1 = ctrl.Rule(antecedent=((error['nb'] & delta['ze']) |
                              (error['nb'] & delta['ps']) |
                              (error['ns'] & delta['ns']) |
                              (error['ns'] & delta['ze']) |
                              (error['ze'] & delta['ns']) |
                              (error['ze'] & delta['nb']) |
                              (error['ps'] & delta['nb'])),
                  consequent=output['ns'], label='rule ns')

rule2 = ctrl.Rule(antecedent=((error['nb'] & delta['pb']) |
                              (error['ns'] & delta['ps']) |
                              (error['ze'] & delta['ze']) |
                              (error['ps'] & delta['ns']) |
                              (error['pb'] & delta['nb'])),
                  consequent=output['ze'], label='rule ze')

rule3 = ctrl.Rule(antecedent=((error['ns'] & delta['pb']) |
                              (error['ze'] & delta['pb']) |
                              (error['ze'] & delta['ps']) |
                              (error['ps'] & delta['ps']) |
                              (error['ps'] & delta['ze']) |
                              (error['pb'] & delta['ze']) |
                              (error['pb'] & delta['ns'])),
                  consequent=output['ps'], label='rule ps')

rule4 = ctrl.Rule(antecedent=((error['ps'] & delta['pb']) |
                              (error['pb'] & delta['pb']) |
                              (error['pb'] & delta['ps'])),
                  consequent=output['pb'], label='rule pb')
```

Based on these rules, the fuzzy controller system predicts the output values.

Error	Delta	Output
-2.00	-2.00	-1.67
-1.20	-2.00	-1.66
-0.40	-2.00	-1.10
0.40	-2.00	-1.00
1.20	-2.00	-0.74
2.00	-2.00	-0.00
-2.00	-1.20	-1.66
-1.20	-1.20	-1.06
-0.40	-1.20	-1.05
0.40	-1.20	-0.57
1.20	-1.20	0.00
2.00	-1.20	0.74
-2.00	-0.40	-1.10
-1.20	-0.40	-1.05

With helpful use of Matplotlib and repeated simulations, we can observe what the entire control system surface looks like in three dimensions.



CONCLUSION: Greg Viot's fuzzy cruise controller using MATLAB was solved succefully.

CRITERIA	TOTAL MARKS	MARKS OBTAINED	COMMENTS
Concept (A)	2		
Implementation (B)	2		
Performance (C)	2		
Total	6		

Experiment 4

AIM: Implement Union, Intersection, Complement and Difference operations on fuzzy sets. Also create fuzzy relation by Cartesian product of any two fuzzy sets and perform max min composition on any two fuzzy relations.

SOFTWARE USED: JUPYTER NOTEBOOK

PROGRAM CODE:

```
import numpy as np
def union(A,B):
result={}
for i in A:
if(A[i]>B[i]):
result[i]=A[i]
else:
result[i]=B[i]
print("Union of two sets is",result)
def intersection(A,B):
result={}
for i in A:
if(A[i]<B[i]):
result[i]=A[i]
else:
result[i]=B[i]
print("Intersection of two sets is",result)
def complement(A,B):
result={}
result1={}
for i in A:
result[i]=round(1-A[i],2)
for i in B:
result1[i]=round(1-B[i],2)
print("Complement of  1st set is",result)
print("Complement of  2nd set is",result1)
def difference(A,B):
result={}
for i in A:
result[i]=round(min(A[i],1-B[i]),2)
print("Difference of two sets is",result)
def cartprod(A,B):
R = [[] for i in range(len(A))]
i = 0
```

```
for x in A:
for y in B:
R[i].append(min(A[x], B[y]))
i += 1
print("Cartesian Product is",np.array(R),"\n")
def maxmin():
R = None
S = None
with open("./relations.json") as f:
relations = json.load(f)
R = relations["R"]
S = relations["S"]
print("\nR: " + str(R))
print("S: " + str(S))
m, n = len(R), len(R[0])
    •  = len(S[0])
composition = dict()
for i in range(m):
composition[i] = dict()
for k in range(o):
composition[i][k] = max([min(R[i][j], S[j][k]) for j in range(n)])
return composition
import json
def main():
while True:
print("Menu Driven Program")
print("1.Union")
print("2.Intersection")
print("3.Complement")
print("4.Difference")
print("5.Cartesian product")
print("6.MaxMin Composition")
print("7.Exit")
choice=int(input("Enter your choice:"))
if choice==1:
union(d,d1)
elif choice==2:
intersection(d,d1)
elif choice==3:
complement(d,d1)
elif choice==4:
difference(d,d1)
elif choice==5:
cartprod(d,d1)
```

```
elif choice==6:
composition=maxmin()
print("\nMax-min composition:", composition, sep="\n")
elif choice==7:
break
else:
print("Wrong choice")
if __name__ == "__main__":
print("---------------------"+
"FUZZY SET OPERATIONS"+
"---------------------")
n = int(input("enter  no.of elements of set 1:"))
d = {}
for i in range(n):
keys = input()
values = float(input())
d[keys] = values
n1 = int(input("enter  no.of elements of set 2:"))
d1 = {}
for i in range(n1):
keys1 = input()
values1 = float(input())
            d1[keys1] = values1
            main()
```

RESULT:

```
----------------------FUZZY SET OPERATIONS---------------------
enter  no.of elements of set 1:5
0.2
0.3
0.6
0.5
0.3
0.9
1.0
0.7
0.8
0.1
enter  no.of elements of set 2:2
0.1
0.4
0.6
0.8
Menu Driven Program
1.Union
2.Intersection
3.Complement
4.Difference
5.Cartesian product
6.MaxMin Composition
7.Exit
Enter your choice:1
```

```
Complement of  1st set is {'x1': 0.8, 'x2': 0.5, 'x3': 0.2}
Complement of  2nd set is {'x1': 0.6, 'x2': 0.8, 'x3': 0.9}
```

CONCLUSION: Implementation of Union, Intersection, Complement and Difference operations on fuzzy sets was successfully performed.

CRITERIA	TOTAL MARKS	MARKS OBTAINED	COMMENTS
Concept (A)	2		
Implementation (B)	2		
Performance (C)	2		
Total	6		

Experiment 5

AIM: Create a simple ADALINE network with appropriate no. of input and output nodes. Train it using delta learning rule until no change in weights is required. Output the final weights.

SOFTWARE USED: JUPYTER NOTEBOOK

PROGRAM CODE:

1. import numpy as np
2. from numpy.random import seed
3. class AdalineSGD(object):
    a. """ ADAptive LInear NEuron classifier.
    b. Parameters
    c. -----------
    d. eta : float
        i. Learning rate (between 0.0 and 1.0)
    e. n_iter : int
        i. Passes over the training dataset.
    f. Attributes
    g. -----------
    h. w_ : 1d-array
        i. Weights after fitting.
    i. errors_ : list
        i. Number of misclassifications in every epoch.
    j. shuffle : bool (deafult: True)
        i. Shuffles training data every epoch if True
        ii. to prevent cycles.
    k. random_state : int (default: None)
        i. Set random state for shuffling and
        ii. initializing the weights.
    l. """
    m. def __init__(self, eta = 0.01, n_iter = 10, shuffle= True,
                a. random_state = None):
        ii. self.eta = eta
        iii. self.n_iter = n_iter
        iv. self.w_initialization = False
        v. self.shuffle = shuffle
        vi. if random_state:
            1. seed(random_state)
    n. def fit(self, X, y):
        i. """ Fit training data.
        ii. Parameters
        iii. ------------
        iv. X : {array-like}, shape = [n_samples, n_features]

            1. Training vectors, where n_samples is the
            2. number of samples and n_features is the number
            3. of features.
        v. y : array-like, shape = [n_samples]
            1. Target values.
        vi. Return
        vii. -------
        viii. self : object
        ix. """
        x. self._initialize_weights(X.shape[1])
        xi. self.cost_ = []
        xii. for i in range(self.n_iter):
            1. if self.shuffle:
                a. X, y = self._shuffle(X, y)
            2. cost = []
            3. for xi, target in zip(X, y):
                a. cost.append(self._update_weights(xi, target))
            4. avg_cost = sum(cost) / len(y)
            5. self.cost_.append(avg_cost)
        xiii. return self
    o. def partial_fit(self, X, y):
        i. """ Fit training data without reinitializing the weights """
        ii. if not self.w_initialized:
            1. self._initialize_weights(X.shape[1])
        iii. if y.ravel().shape[0] > 1:
            1. for xi, target in zip(X, y):
                a. self._update_weights(xi, target)
        iv. else:
            1. self._update_weights(X, y)
        v. return self
    p. def _shuffle(self, X, y):
        i. """ Shuffle training data """
        ii. r = np.random.permutation(len(y))
        iii. return X[r], y[r]
    q. def _initialize_weights(self, m):
        i. """ Initialize weights to zeros """
        ii. self.w_ = np.zeros(1 + m)
        iii. self.w_initialized = True
    r. def _update_weights(self, xi, target):
        i. """ Apply Adaline learning rule to update the weights """
        ii. output = self.net_input(xi)
        iii. error = (target - output)
        iv. self.w_[1:] += self.eta * xi.dot(error)

v. self.w_[0] += self.eta * error
vi. cost = 0.5 * (error ** 2)
vii. return cost

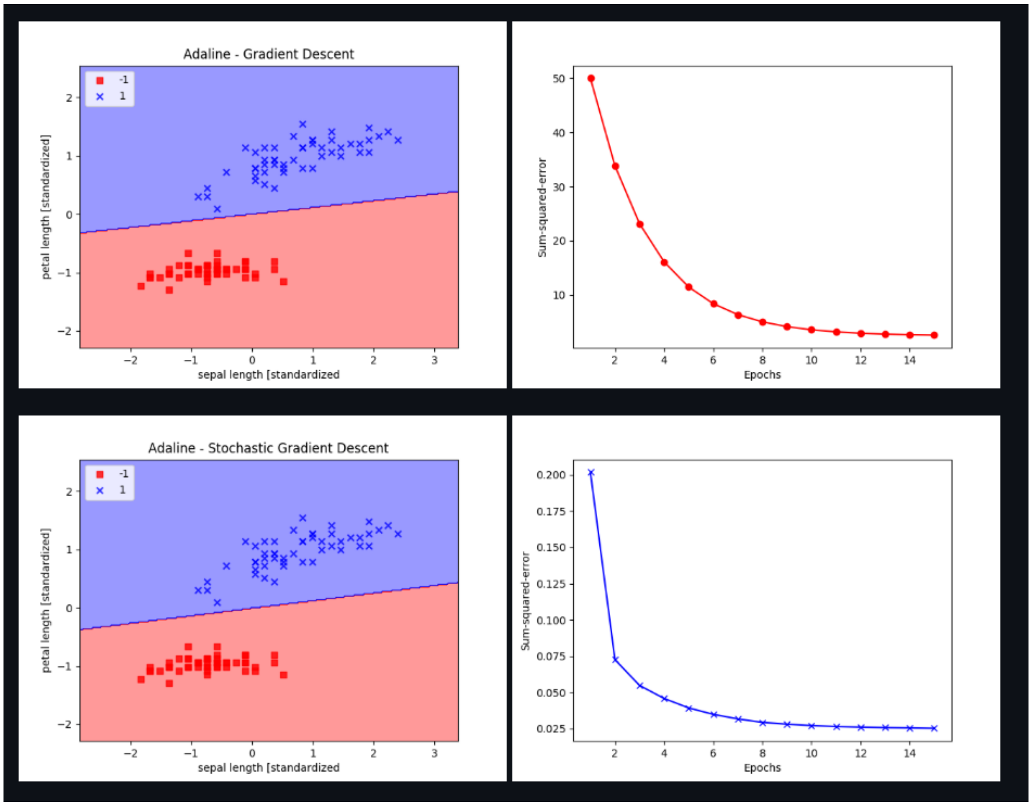
s. def net_input(self, X):
    i. """ Calculate net input """
    ii. return np.dot(X, self.w_[1:]) + self.w_[0]
t. def activation(self, X):
    i. """ Compute linear activation """
    ii. return self.net_input(X)
u. def predict(self, X):
    i. """ Return class label after the unit step """
    ii. return np.where(self.activation(X) >= 0.0, 1, -1)

main.py

1. import numpy as np
2. import pandas as pd
3. import matplotlib.pyplot as plt
4. from adalinegd import AdalineGD
5. from adalinesgd import AdalineSGD
6. import pdr
7. # get the iris data
8. df = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data',
    a. header = None)
9. # Plot 100 samples of the data
10. y = df.iloc[0:100, 4].values
11. y = np.where(y == 'Iris-setosa', -1, 1)
12. X = df.iloc[0:100, [0, 2]].values
13. plt.scatter(X[:50, 0], X[:50, 1], color = 'red', marker = 'o', label = 'setosa')
14. plt.scatter(X[50:100, 0], X[50:100, 1], color = 'blue', marker = 'x', label = 'versicolor')
15. plt.xlabel('petal length')
16. plt.ylabel('sepal length')
17. plt.legend(loc = 'upper left')
18. plt.show()
19. # Standardize the data
20. X_std = np.copy(X)
21. X_std[:, 0] = (X[:, 0] - X[:, 0].mean()) / X[:, 0].std()
22. X_std[:, 1] = (X[:, 1] - X[:, 1].mean()) / X[:, 1].std()
23. # Create the AdalineGD model
24. model1 = AdalineGD(n_iter = 15, eta = 0.01)
25. # Train the model
26. model1.fit(X_std, y)

```
# Plot the training error
plt.plot(range(1, len(model1.cost_) + 1), model1.cost_, marker = 'o', color = 'red')
plt.xlabel('Epochs')
plt.ylabel('Sum-squared-error')
plt.show()
# Plot the decision boundary
pdr.plot_decision_regions(X_std, y, classifier = model1)
plt.title('Adaline - Gradient Descent')
plt.xlabel('sepal length [standardized')
plt.ylabel('petal length [standardized]')
plt.legend(loc = 'upper left')
plt.show()


# Create the AdalineSGD model
model2 = AdalineSGD(n_iter = 15, eta = 0.01, random_state = 1)


# Train the model
model2.fit(X_std, y)
# Plot the training errors of both of the models
plt.plot(range(1, len(model2.cost_) + 1), model2.cost_, marker = 'x', color = 'blue')
plt.xlabel('Epochs')
plt.ylabel('Sum-squared-error')
plt.show()
# Plot the decision boundary
pdr.plot_decision_regions(X_std, y, classifier = model2)
plt.title('Adaline - Stochastic Gradient Descent')
plt.xlabel('sepal length [standardized')
plt.ylabel('petal length [standardized]')
plt.legend(loc = 'upper left')
plt.show()
```

RESULT:



CONLUSION: implementation was successful.

CRITERIA	TOTAL MARKS	MARKS OBTAINED	COMMENTS
Concept (A)	2		
Implementation (B)	2		
Performance (C)	2		
Total	6		