

In []:

In [1]:

```
import math
import numpy as np
import pandas as pd
# import seaborn as sns
# import matplotlib.pyplot as plt

# from sklearn.preprocessing import LabelEncoder
# from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import LabelBinarizer
from sklearn import preprocessing
from scipy.special import expit
from sklearn.model_selection import train_test_split
from numpy.random import default_rng

from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score

import time

from wine_accffaAnn import *

class MultiLayerPerceptron():
    # ===== Activation Functions ===== #
    # accepts a vector or list and returns a list after performing corresponding function

    @staticmethod
    def sigmoid(vectorSig):
        """returns 1/(1+exp(-x)), where the output values lies between zero and one"""
        sig = expit(vectorSig)
        return sig

    @staticmethod
    def binaryStep(x):
        """ It returns '0' if the input is less than zero otherwise it returns one """
        return np.heaviside(x, 1)

    @staticmethod
    def linear(x):
        """ y = f(x) It returns the input as it is"""
        return x

    @staticmethod
    def tanh(x):
        """ It returns the value (1-exp(-2x))/(1+exp(-2x)) and the value returned will be between -1 and 1 """
        return np.tanh(x)

    @staticmethod
    def relu(x): # Rectified Linear Unit
        """ It returns zero if the input is less than zero otherwise it returns the given input """
        x1 = []
        for i in x:
            if i < 0:
                x1.append(0)
            else:
                x1.append(i)

        return x1
```

```

@staticmethod
def leakyRelu(x):
    """ It returns zero if the input is less than zero otherwise it returns the give
x1 = []
for i in x:
    if i < 0:
        x1.append((0.01 * i))
    else:
        x1.append(i)

return x1

@staticmethod
def parametricRelu(self, a, x):
    """ It returns zero if the input is less than zero otherwise it returns the give
x1 = []
for i in x:
    if i < 0:
        x1.append((a * i))
    else:
        x1.append(i)

return x1

@staticmethod
def softmax(self, x):
    """ Compute softmax values for each sets of scores in x"""
    return np.exp(x) / np.sum(np.exp(x), axis=0)

# ===== Activation Functions Part Ends ===== #

# ===== Distance Calculation ===== #

@staticmethod
def chebishev(self, cord1, cord2, exponent_h):
    dist = 0.0
    if ((type(cord1) == int and type(cord2) == int) or ((type(cord1) == float and ty
        dist = math.pow((cord1 - cord2), exponent_h)
    else:
        for i, j in zip(cord1, cord2):
            dist += math.pow((i - j), exponent_h)
    dist = math.pow(dist, (1.0 / exponent_h))
    return dist

@staticmethod
def minimum_distance(self, cord1, cord2):
    # min(|x1-y1|, |x2-y2|, |x3-y3|, ...)
    dist = float('inf')
    if ((type(cord1) == int and type(cord2) == int) or ((type(cord1) == float and ty
        dist = math.fabs(cord1 - cord2)
    else:
        for i, j in zip(cord1, cord2):
            temp_dist = math.fabs(i - j)
            if (temp_dist < dist):
                dist = temp_dist
    return dist

@staticmethod
def maximum_distance(self, cord1, cord2):
    # max(|x1-y1|, |x2-y2|, |x3-y3|, ...)
    dist = float('-inf')
    if ((type(cord1) == int and type(cord2) == int) or ((type(cord1) == float and ty
        dist = math.fabs(cord1 - cord2)
    else:
        for i, j in zip(cord1, cord2):

```

```

        temp_dist = math.fabs(i - j)
        if (temp_dist > dist):
            dist = temp_dist
    return dist

@staticmethod
def manhattan(self, cord1, cord2):
    # |x1-y1| + |x2-y2| + |x3-y3| + ...
    dist = 0.0
    if ((type(cord1) == int and type(cord2) == int) or ((type(cord1) == float and ty
        dist = math.fabs(cord1 - cord2)
    else:
        for i, j in zip(cord1, cord2):
            dist += math.fabs(i - j)
    return dist

@staticmethod
def euclidian(self, cord1, cord2):
    dist = 0.0
    if ((type(cord1) == int and type(cord2) == int) or ((type(cord1) == float and ty
        dist = math.pow((cord1 - cord2), 2)
    else:
        for i, j in zip(cord1, cord2):
            dist += math.pow((i - j), 2)
    return math.pow(dist, 0.5)

# ===== Distance Calculation Ends ===== #

def __init__(self, dimensions=(8, 5), all_weights=(0.1, 0.2), fileName="iris", test : str=""):

    """
    Args:
        dimensions : dimension of the neural network
        all_weights : the optimal weights we get from the bio-algoANN models
    """
    self.allPop_Weights = []
    self.allPopl_Chromosomes = []
    self.allPop_ReceivedOut = []
    self.allPop_ErrorVal = []

    self.all_weights = all_weights
    self.fitness = []

    # ===== Input dataset and corresponding output ===== #

    self.fileName = fileName
    self.fileName += ".csv"
    data = pd.read_csv(self.fileName, sep=';')
    data = data.infer_objects()

    output_values_expected = []
    input_values = []

    # ~~~~ encoding ~~~~

    # labelencoder = LabelEncoder()
    # data[data.columns[-1]] = labelencoder.fit_transform(data[data.columns[-1]])

    # one hot encoding - for multi-column
    # enc = OneHotEncoder(handle_unknown='ignore')
    # combinedData = np.vstack((data[data.columns[-2]], data[data.columns[-1]])) .T
    # print(combinedData)
    # y = enc.fit_transform(combinedData).toarray()
    # y = OneHotEncoder().fit_transform(combinedData).toarray()

```

```

#
y = LabelBinarizer().fit_transform(data[data.columns[-1]])
# print(y)

# ~~~~ encoding ends~~~~

for j in range(len(data)):
    output_values_expected.append(y[j])

# print(output_values_expected)

input_values = []
for j in range(len(data)):
    b = []
    for i in range(len(data.columns) - 1):
        b.append(data[data.columns[i]][j])
    input_values.append(b)

self.X = input_values[:]
self.Y = output_values_expected[:]

# input and output
self.X = input_values[:]
self.Y = output_values_expected[:]

self.test = test
X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size=0.33)
if(self.test == True):
    self.X = X_test
    self.Y = Y_test
else:
    self.X = X_train
    self.Y = Y_train

self.dimension = dimensions
# print(self.dimension)

# ===== Finding Initial Weights ===== #

self.pop = [] # weights
reshaped_all_weights = []
start = 0
for i in range(len(self.dimension) - 1):
    end = start + self.dimension[i + 1] * self.dimension[i]
    temp_arr = self.all_weights[start:end]
    w = np.reshape(temp_arr[:,], (self.dimension[i + 1], self.dimension[i]))
    reshaped_all_weights.append(w)
    start = end
self.pop.append(reshaped_all_weights)

self.init_pop = self.all_weights

# ===== Initial Weights Part Ends ===== #

def Predict(self, chromo):
    # X, Y and pop are used
    self.fitness = []
    total_error = 0
    m_arr = []
    k1 = 0
    for i in range(len(self.dimension) - 1):
        p = self.dimension[i]
        q = self.dimension[i + 1]
        k2 = k1 + p * q

```

```

m_temp = chromo[k1:k2]
m_arr.append(np.reshape(m_temp, (p, q)))
k1 = k2

y_predicted = []
for x, y in zip(self.X, self.Y):

    yo = x

    for mCount in range(len(m_arr)):
        yo = np.dot(yo, m_arr[mCount])
        yo = self.sigmoid(yo)

    # converting to sklearn acceptable form
    max_yo = max(yo)
    for y_vals in range(len(yo)):
        if(yo[y_vals] == max_yo):
            yo[y_vals] = 1
        else:
            yo[y_vals] = 0
    y_predicted.append(yo)
return (y_predicted, self.Y)

def main(self):
    Y_PREDICT, Y_ACTUAL = self.Predict(self.init_pop)
    Y_PREDICT = np.array(Y_PREDICT)
    Y_ACTUAL = np.array(Y_ACTUAL)

    n_classes = 7

    label_binarizer = LabelBinarizer()
    label_binarizer.fit(range(n_classes))
    Y_PREDICT = label_binarizer.inverse_transform(np.array(Y_PREDICT))
    Y_ACTUAL = label_binarizer.inverse_transform(np.array(Y_ACTUAL))

    # find error
    if(self.test == True):
        print("\n Actual / Expected", Y_ACTUAL)
        print("\n Predictions", Y_PREDICT)
        print("\n\nConfusion Matrix")
        print(confusion_matrix(Y_ACTUAL, Y_PREDICT))

        print("\n\nClassification Report")
        target_names = []
        for i in range(7):
            k='class '+str(i)
            target_names.append(k)
        print(classification_report(Y_ACTUAL, Y_PREDICT, target_names=target_names))
        print("\n\n\n")
    return accuracy_score(Y_ACTUAL, Y_PREDICT)

```

In [2]:

```

start_time = time.time()
i = InputData(fileName="../ANN/winequality-white")
input_val, output_val = i.main()
end_time = time.time()
print("Time for inputting data : ", end_time - start_time)

print("===== Calling FFA to get best weights =====")

start_time = time.time()
a = ffaAnn(initialPopSize=100, m=10, dimensions = [100,10], input_values=input_val, outp
fit, b, weights, dim, all_gen_best_weight = a.main()

end_time = time.time()

```

```

print("Time taken : ", end_time - start_time)

print("\n Fitness : ", fit, "\n Best Weights : ", weights, "\n Dimensions : ", dim)

import matplotlib.pyplot as plt
x=b[:,]
z=[i for i in range(0,10)]
plt.plot(z,x)

plt.title("Firefly Algorithm")
plt.ylabel("Fitness")
plt.xlabel("Iterations")
end_time = time.time()
print("Time Taken : ", end_time - start_time)

```

```

Time for inputting data : 0.2642948627471924
===== Calling FFA to get best weights =====
-----GENERATION 0-----
Initial worst fitness = 173025.89881924863

Initial best fitness = 34972.049987817154
-----GENERATION 1-----
-----GENERATION 2-----
-----GENERATION 3-----
-----GENERATION 4-----
-----GENERATION 5-----
-----GENERATION 6-----
-----GENERATION 7-----
-----GENERATION 8-----
-----GENERATION 9-----
Fitness : 34972.049987817154
Time taken : 797.5820610523224

```

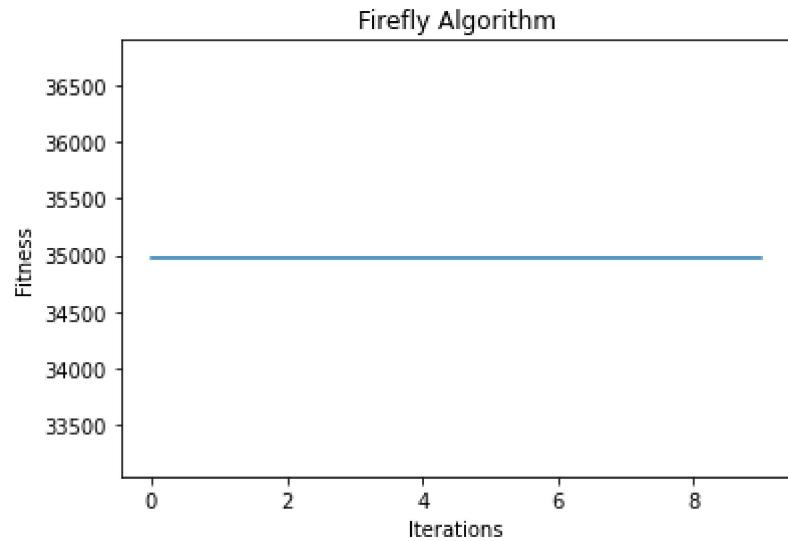
```

Fitness : 34972.049987817154
Best Weights : [-24, -12, -40, -50, 39, -12, 67, 36, 32, 64, 52, -11, -55, -70, 0, 80,
-27, -9, 76, -51, -82, 22, 84, -83, -21, -55, -57, -46, 36, 84, -61, 63, 33, -71, -73,
-78, 44, 60, -40, -13, 41, -13, 22, 33, 48, -73, -79, -39, 6, -35, 76, 74, -74, -6, 3, -4
9, 10, 12, -64, 56, 58, 2, -5, -17, -33, 24, -75, -22, 37, 58, -12, 61, 64, -82, 81, 21,
69, -84, 5, 19, -41, 29, 54, -88, -25, 8, 60, 32, 89, 86, 46, 48, -66, 46, -50, 16, 22,
89, 14, -90, -22, 46, 47, -35, 77, -90, -23, -89, -79, 35, 75, -77, 44, 73, -14, -10, 6
2, 38, 43, 79, -86, -84, 72, -14, 54, 52, 79, 16, -88, 68, 68, 0, 1, -58, -13, 29, 31, 6
1, -7, 86, -7, 18, -33, -55, -33, 15, 39, -71, 57, -5, -34, -85, -4, 67, -20, 74, 0, 9,
6, -16, -65, 44, -1, -53, 74, -89, -58, -35, 4, -19, 63, -9, -5, -84, -36, 67, 55, -29,
-5, -88, -16, 60, 52, 20, 38, 29, -71, -15, -84, 58, -56, -7, 25, -11, -34, -68, -29, 2
1, -85, 71, 73, 11, 46, -79, 16, 52, -25, 13, 36, 70, 45, -5, -9, -4, 23, 38, 38, 46, 4
6, 70, 51, 70, -23, -10, 31, 83, -49, 58, 80, 12, -54, 32, -89, -54, 6, -22, 14, -52, -6
9, 51, 36, -9, 62, 88, -84, -44, 47, 68, 9, 69, 50, -8, -3, 20, -72, -79, 55, 45, -57, 6
1, 88, -59, 51, -14, 30, -63, 62, -7, 45, 45, -80, 60, 86, -11, -3, -71, -84, -42, -54,
14, 51, -3, 40, 6, 10, 28, -45, 75, -61, 18, -44, -47, -82, -80, 83, -8, 13, 1, -83, 9,
-84, -37, 15, 17, -20, 7, 88, -85, -83, -6, 0, 0, 49, -44, -65, 35, 41, 80, -62, 28, 10,
66, -57, -66, -41, -88, 76, 50, -27, 85, -33, 32, -78, 74, -67, -26, 1, 13, -70, 14, -8
4, 37, -36, -59, 59, 80, 50, 45, 37, 21, -45, 34, -40, -52, -30, -8, -3, 82, -22, 50, -8
3, 75, -80, -43, 8, -68, 40, -83, 70, 17, -29, 55, 75, 53, 3, 61, -44, 77, -68, -18, 25,
-69, -66, 28, -55, -15, 14, -83, 81, -49, -71, 4, 47, -43, 43, -4, 17, 26, 36, 17, 28, 1
5, -26, -6, -30, -70, 73, 36, 4, -15, 59, 78, -65, 20, -11, -27, -31, -51, -64, -57, 71,
-70, 3, -76, 23, -43, 25, -33, 76, 8, -37, -31, 52, -50, -16, -65, 12, -15, -82, 17, 56,
-10, -87, -13, -88, 38, 77, -32, 77, 80, 55, 19, -88, 11, 63, -90, -78, -4, -5, 37, -10,
76, 5, -37, 81, 80, -56, -45, -69, -49, 22, 58, -15, 12, -14, -20, -42, 46, 82, -65, -7
3, -80, -4, 82, -22, 67, -64, -35, -24, 33, -18, -75, 34, 52, 78, 82, 41, 54, -4, 4, 33,
-48, 77, -19, 11, -36, -54, -86, -88, 38, -36, 13, -90, -45, -57, 33, -79, -86, -18, -1
5, -21, -38, -83, -62, 41, 20, -54, 88, -16, -52, -75, 46, -72, 77, 46, -19, -1, 31, 76,
-1, 72, 42, -69, 21, 89, -5, 1, 39, -78, -79, 44, 14, 39, -70, 9, -69, -7, 80, 17, -87,
-18, -49, -16, -25, -4, -1, -56, -10, 29, -49, -22, 16, -18, 32, 27, 50, -77, -83, -87,
51, -13, 45, -51, -84, 28, 86, -19, -16, 4, -6, -84, 29, -83, -25, -53, 43, -41, 68, 48,

```

-4, -64, 76, -50, -21, -89, -5, 46, -1, 89, 7, -35, 26, -32, 46, 70, 50, 7, -9, -86, 70, -16, -64, -87, -58, -66, -76, -57, -77, -16, 58, -84, -52, -56, 77, -49, -32, -3, -60, -89, -20, -68, 3, -56, -68, 80, 73, -16, 10, -18, 0, -51, 32, -89, -30, -80, -22, 4, 49, -79, 59, 16, -18, 36, 39, 87, 21, 36, 55, 73, 14, -70, -32, -22, -83, -43, -5, -49, 0, 4 0, 65, 41, -43, 68, -21, 2, 9, -11, -18, 80, -45, 3, -38, -43, 47, 42, 85, 68, -40, -13, -11, 84, 41, 73, -44, 87, -67, 18, -26, 8, 67, 1, 10, -46, -89, 85, -88, -72, -55, 39, -56, 89, 0, 65, 14, 4, 46, 72, 2, -41, -53, 24, -83, -73, 33, -13, 52, -33, -64, -42, -2 8, 80, -65, -54, -14, -29, 59, -57, -83, -9, -29, 61, 19, 31, 40, -7, -54, -9, -33, -76, 35, 34, 58, 33, 81, 32, 15, -35, 25, 43, -90, 7, -49, 75, 26, -3, 13, -14, 0, -70, -17, 47, 60, -57, 84, 21, -35, 40, -81, 32, 23, -66, 19, -25, 83, 35, -1, 69, 46, -62, -78, -24, 0, -19, 16, 65, 40, -6, -46, 49, 39, -50, 13, -46, 21, -18, -81, 3, 80, 67, -5, 68, 70, 15, 87, 69, 78, 12, 84, -25, -39, 7, -17, -26, 65, -41, 27, 56, 9, 65, -19, 40, -23, 37, 24, -76, -41, 34, -53, 51, 3, -88, 31, 59, 21, -58, 40, 73, -30, 83, -90, 46, 15, -4 4, -31, 17, -13, 75, -87, -90, -48, -8, -40, -34, 55, 72, -21, 82, -49, 72, -42, 14, 33, 44, 16, 38, 56, -12, -12, -46, 22, -20, -66, 88, -30, 7, 69, -69, -90, 18, 12, 34, -80, -83, -35, 89, -17, 21, 65, -22, -63, -77, -2, 22, -67, -36, 35, -51, -52, 76, 42, -40, 9, -68, 23, -44, -51, 21, 11, -11, 71, 54, 71, -4, 7, -54, 10, 61, 77, 54, -12, -46, 10, 64, -85, -73, 69, -40, -30, -20, 55, -2, 66, 41, -81, 5, -4, 59, 50, 37, 10, 8, 34, -13, -84, 42, 26, -73, 64, 54, -44, -80, 51, -27, 46, 36, -14, -63, -34, 13, 70, -81, -65, -4 8, -14, -41, -21, -60, 39, 46, 19, -54, -79, 48, 84, 67, -28, 89, 51, -56, 82, 23, -1, -58, 64, -34, 43, 53, 41, -13, 33, -12, -70, 37, -84, 52, 81, -68, 15, -89, -70, -38, -3 1, -72, 19, 36, 23, 42, -30, -80, -32, 14, 15, 73, 55, -38, 65, 83, 47, 2, 71, 51, -13, 79, -22, -71, 38, -45, 38, 0, 39, -13, -23, -49, -4, -83, -62, 2, 16, 62, 42, 44, 24, 4 4, 60, -65, 62, 2, -2, 24, 77, 64, -12, 15, -46, 17, 30, -19, -86, 2, 68, -90, -38, -29, 61, -58, -84, 57, 86, 74, -67, 73, -1, 56, 44, -36, -42, 23, -78, -74, 25, 14, 69, -78, -54, 34, 38, -81, 36, -39, 87, 43, -10, -20, -7, 30, 22, 75, 75, -72, -42, -22, 65, -31, 63, 8, 58, 59, 8, -34, -19, -16, -68, 5, -1, 12, 43, -2, 78, -57, 56, 36, -44, -45, -40, 19, 84, -16, 18, 26, 82, 59, 37, -90, -78, 76, -82, 29, 33, 83, 85, 85, -78, 84, 26, 14, 57, 31, -10, 40, -45, -69, 71, 16, 83, 23, 78, 44, 8, 28, -64, 7, -90, -88, -24, -14, -6 7, -3, -77, 3, 4, -6, 66, -44, -47, -16, 52, -48, 12, -32, 43, 81, 73, -2, -51, 16, 11, -8, 89, -57, -30, 80, -50, -18, 56, -13, 82, -85, -23, 49, 39, 28, -51, -55, -52, -11, -57, -28, -65, -45, -11, -44, -81, 46, 84, 22, 32, -18, 26, -76, 64, -67, 47, -31, 9, -4 7, -56, -5, 81, -47, -72, 59, 72, 30, -28, 3, 63, 61, -7, 13, 58, 65, -3, 42, -39, -45, 22, 80, 72, -27, -22, 40, 10, -9, -14, -33, 76, -40, -88, -63, 24, 50, 89, 41, -34, 50, -90, -39, 37, -87, 58, -45, -22, 61, 34, -75, -36, 51, -69, -17, -50, -69, -84, -24, -3 4, -1, -88, 30, 36, 29, -47, 6, 27, -42, -68, -41, -81, 23, -80, -41, -68, -56, 68, -83, -89, -62, 10, -9, 22, -51, -36, -67, 33, 30, 18, 68, 82, -85, 28, -81, -90, -11, 32, -4 7, -29, -6, -4, -29, 3, 23, 75, 20, 5, -26, -1, 34, -10, -13, -83, -46, 0, 64, -56, -75, -62, 61, -50, 82, -58, -18, 20, 67, -71, 54, 60, -51, -2, 39, -10, -40, -45, -85, 6, -3 7, -17, 69, -42, 3, -61, 40, -71, 79, -43, -87, -49, -5, 45, 76, 4, -29, 2, -17, 15, 1, 27, -21, -89, 73, -60, 75, 39, 10, -38, 70, 55, -6, 26, -50, 32, 69, 64, -11, -6, 74, -3 2, -52, 27, 28, -52, -48, 81, -90, -87, 48, -25, -60, -68, -85, -31, 14, -1, 48, -75, 3 0, 12, -81, 37, 78, 61, 68, -13, -22, 79, -5, 59, 60, -50, 53, 20, -46, 59, -29, -81, -2 8, 86, -52, -33, 89, -54, 51, -22, -12, -36, -59, 18, 45, 35, 69, 57, 38, -64, -20, 80, -58, 34, -84, 25, -43, 86, 30, 4, 59, 20, -89, -9, -90, -60, 85, 78, 58, -68, -81, -70, -45, 44, -89, -37, -48, 79, -61, 7, 48, 43, 0, 1, -50, -72, 20, -16, 47, -60, 60, 26, -4, -73, 3, 87, -86, 73, -37, 4, 4, 47, 55, 45, 14, -7, -36, 89, -6, 8, -78, -10, -18, -6 3, -63, 61, 39, -6, 33, -67, 4, 65, 8, -80, -68, 43, -10, -60, 13, -23, 81, 86, -2, -24, 18, 51, 3, -88, -67, -25, 17, 80, 16, -86, -6, 41, 38, 20, -28, 51, 15, 85, 45, -40, 4, 35, 66, 57, 31, 36, 78, 8, 79, 19, -78, -32, -63, 14, 1, -16, 9, -21, -11, -89, -43, 63, -30, 80, 89, -27, 78, -18, 9, 28, -56, -16, -56, -60, -36, 63, 66, 24, -67, -24, -17, -6 0, -50, 56, 2, 57, 16, 61, -30, 46, 12, 15, 67, 75, 84, -32, -64, 40, 48, -38, 22, -68, 81, 76, -53, 72, -57, 11, 19, 33, 47, -23, -18, -47, -56, 20, -31, -52, -27, -1, 16, 27, 3, 8, -10, 12, -75, 0, -18, 1, 56, 54, 48, 60, -59, -58, 6, 25, 4, -16, 62, -65, -30, -2 2, -35, -31, -83, -41, -88, -8, 39, 39, 5, 81, 25, -45, -7, 77, -34, 59, 24, 19, 67, 62, -71, 70, 6, 21, -53, 80, 12, 65, -90, 3, -61, 85, -31, 7, 55, -76, -69, 89, 37, 8, 33, 1 4, -32, 19, -7, 80, -26, 7, -31, -79, 19, 24, -50, -65, 30, 75, -4, -13, 3, 30, -66, -1 6, -83, -45, 21, 52, 75, 71, 61, -14, 76, 87, -22, 87, 33, -17, -49, 47, -77, 82, 57, -1 2, 86, 47, 75, 65, -72, 51, -76, 46, -47, 14, -59, 59, -10, 4, -49, 76, 62, 77, -36, -8 7, -38, 74, 51, 83, 70, 67, 39, -66, 6, -81, -49, -54, -25, 31, -12, 86, 61, 80, -61, -5 9, -90, 48, 37, 56, -26, 53, 41, -40, 18, 51, 20, -81, -40, 53, -49, 66, 83, -65, -70, 3 5, 85, -41, 38, -15, 29, 59, 60, 25, 86, 55, -15, 45, 80, -9, -56, -51, -2, 23, 55, 52, 25, -90, 35, 81, 22, -20, 68, -21, 59, -30, 9, 56, -74, -63, -48, 31, 34, 54, -11, 35, 1 8, 40, -17, 67, 16, 15, -10, 79, -28, -4, -15, 2, 22, -72, -89, -60, 60, -66, -27, 75, -2 2, -52, -50, 52, 35, 53, 13, 52, -31, 70, -5, 55, 43, -26, -75, -50, -87, 23, -69, 39, -34, -50, 37, -62, 68, 0, -24, 79, -19, 13, 86, -9, 33, 69, -86, -33, 12, -52, -76, -4, -

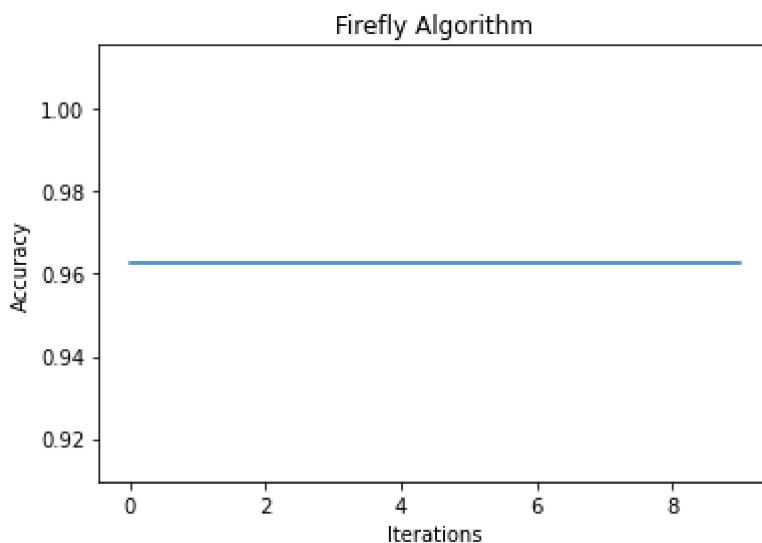
```
87, 40, -6, -48, 85, 50, -73, 42, -33, 65, 43, -51, 9, 51, 20, -12, -84, -31, 18, 33, -4  
7, -59, 88, 70, 20, 39, 81, -16, 20, -78, -39, 29, 27, -85, 39, 87, 81, -20, 73, -53, 3  
7, 83, 38, -11, 58, 30, -74, 19, -26, 85, -70, 51, -35, -1, 78, -7, 75, 80, -70, -50, -5  
3, -48, -71, 22, 15, 26, 63, 72, 66, 21, 23, -38, -64, -72, -69, 54, -60, 16, -76, 35, 8  
6, 86, -19, 36, 24, 63, 45, -72, -13, 86, -78, 82, 8, -56, -71, 28, -26, -61, -72, 67, -  
69, 74, 28, 5, -41, -80, 1, 77, 38, 2, 3, -70, -89, 83, -78, 47, 76, 82, 42, -90, -3, 3  
5, 68, -33, 46, 12, 22, -47, -1, -33, -69, 75, 11, 22, -35, -49, 65, -70, 4, 1, 17, -76,  
15, -83, -22, -32, 65, 12, -36, -17, 71, -80, -49, -46, -46, -23, -31, -72, -56, 81, 28,  
13, 43, 2, -13, -50, -75, -46, -12, -48, -42, -86, 10, -30, -26, -5, 68, -12, 5, 42, -6  
6, 52, 7, 5, -66, 16, -17, 39, -23, -32, -60, -54, -52, -57, -38, 57, 3, -66, -68, -69,  
46, -57, 15, 51, -38, -42, 6, 23, -26, 88, -68, 36, 49, 10, -89, -11, -90, -75, 77, -2,  
-71, 58, -74, -81, -37, -27, 73, 89, 87, -4, -76, -23, -26, -56, 65, -2, 71, -82, 61, -  
4]  
Dimensions : [11, 100, 10, 7]  
Time Taken : 797.6638431549072
```



```
In [3]: print("\n\n===== MLP Program Begins =====")  
  
start_time = time.time()  
print("Training")  
m = MultiLayerPerceptron(fileName="../ANN/winequality-white", dimensions=dim, all_weight  
m.main()  
end_time = time.time()  
print("Time taken = ", end_time - start_time)  
  
===== MLP Program Begins =====  
Training  
Time taken = 0.49268388748168945
```

```
In [4]: start_time = time.time()  
print("Testing")  
m = MultiLayerPerceptron(fileName="../ANN/winequality-white", dimensions=dim, all_weight  
m.main()  
  
end_time = time.time()  
print("Time taken = ", end_time - start_time)  
  
Testing  
Time taken = 0.5116314888000488
```

```
In [5]: all_accuracy = []  
for weights in all_gen_best_weight:  
    m = MultiLayerPerceptron(fileName="../ANN/winequality-white", dimensions=dim, all_we  
accuracy_val = m.main()  
print(accuracy_val)  
all_accuracy.append(accuracy_val)
```



In [1]:

In []:

In []:

In [6]:

```
start_time = time.time()
i = InputData(fileName="../ANN/winequality-white")
input_val, output_val = i.main()
end_time = time.time()
print("Time for inputting data : ", end_time - start_time)

print("===== Calling FFA to get best weights =====")

start_time = time.time()
a = ffaAnn(initialPopSize=100, m=10, dimensions = [100,10], input_values=input_val, outp

fit, b, weights, dim, all_gen_best_weight = a.main()

end_time = time.time()
print("Time taken : ", end_time - start_time)

print("\n Fitness : ", fit, "\n Best Weights : ", weights, "\n Dimensions : ", dim)
```

```
import matplotlib.pyplot as plt
x=b[:,]
z=[i for i in range(0,100)]
plt.plot(z,x)

plt.title("Firefly Algorithm")
plt.ylabel("Fitness")
plt.xlabel("Iterations")
end_time = time.time()
print("Time Taken : ", end_time - start_time)
```

```
Time for inputting data :  0.35205578804016113
===== Calling FFA to get best weights =====
-----GENERATION 0-----
Initial worst fitness =  200644.6410310447

Initial best fitness =  34307.82335153011
-----GENERATION 1-----
-----GENERATION 2-----
-----GENERATION 3-----
-----GENERATION 4-----
-----GENERATION 5-----
-----GENERATION 6-----
-----GENERATION 7-----
-----GENERATION 8-----
-----GENERATION 9-----
-----GENERATION 10-----
-----GENERATION 11-----
-----GENERATION 12-----
-----GENERATION 13-----
-----GENERATION 14-----
-----GENERATION 15-----
-----GENERATION 16-----
-----GENERATION 17-----
-----GENERATION 18-----
-----GENERATION 19-----
-----GENERATION 20-----
-----GENERATION 21-----
-----GENERATION 22-----
-----GENERATION 23-----
-----GENERATION 24-----
-----GENERATION 25-----
-----GENERATION 26-----
-----GENERATION 27-----
-----GENERATION 28-----
-----GENERATION 29-----
-----GENERATION 30-----
-----GENERATION 31-----
-----GENERATION 32-----
-----GENERATION 33-----
-----GENERATION 34-----
-----GENERATION 35-----
-----GENERATION 36-----
-----GENERATION 37-----
-----GENERATION 38-----
-----GENERATION 39-----
-----GENERATION 40-----
-----GENERATION 41-----
-----GENERATION 42-----
-----GENERATION 43-----
-----GENERATION 44-----
-----GENERATION 45-----
-----GENERATION 46-----
-----GENERATION 47-----
-----GENERATION 48-----
```

----- GENERATION 49 -----
----- GENERATION 50 -----
----- GENERATION 51 -----
----- GENERATION 52 -----
----- GENERATION 53 -----
----- GENERATION 54 -----
----- GENERATION 55 -----
----- GENERATION 56 -----
----- GENERATION 57 -----
----- GENERATION 58 -----
----- GENERATION 59 -----
----- GENERATION 60 -----
----- GENERATION 61 -----
----- GENERATION 62 -----
----- GENERATION 63 -----
----- GENERATION 64 -----
----- GENERATION 65 -----
----- GENERATION 66 -----
----- GENERATION 67 -----
----- GENERATION 68 -----
----- GENERATION 69 -----
----- GENERATION 70 -----
----- GENERATION 71 -----
----- GENERATION 72 -----
----- GENERATION 73 -----
----- GENERATION 74 -----
----- GENERATION 75 -----
----- GENERATION 76 -----
----- GENERATION 77 -----
----- GENERATION 78 -----
----- GENERATION 79 -----
----- GENERATION 80 -----
----- GENERATION 81 -----
----- GENERATION 82 -----
----- GENERATION 83 -----
----- GENERATION 84 -----
----- GENERATION 85 -----
----- GENERATION 86 -----
----- GENERATION 87 -----
----- GENERATION 88 -----
----- GENERATION 89 -----
----- GENERATION 90 -----
----- GENERATION 91 -----
----- GENERATION 92 -----
----- GENERATION 93 -----
----- GENERATION 94 -----
----- GENERATION 95 -----
----- GENERATION 96 -----
----- GENERATION 97 -----
----- GENERATION 98 -----
----- GENERATION 99 -----

Fitness : 23512.49667324166

Time taken : 8389.365004777908

Fitness : 23512.49667324166

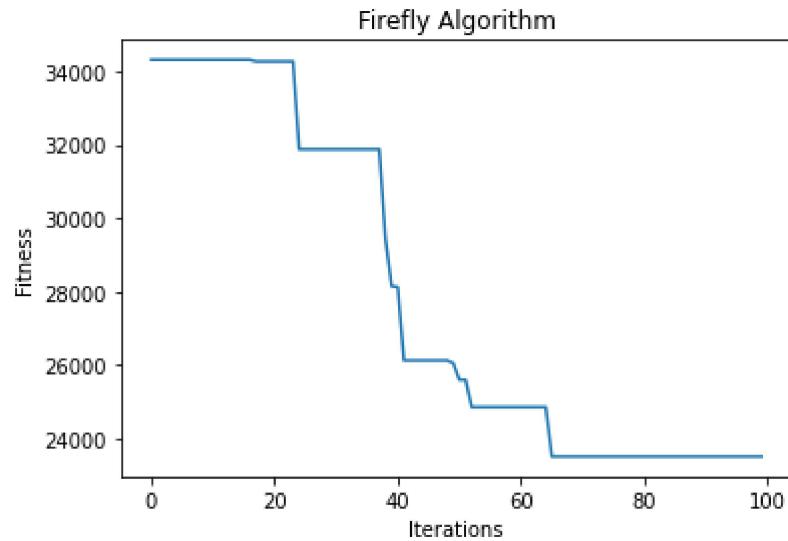
Best Weights : [0, 0, 0, 0, 8, -12, -1, 0, 0, -2, -1, -2, 0, 8, -15, -6, 0, 0, 1, -2,
0, 0, 0, -7, 0, 0, 4, 0, 0, -2, -3, -1, -14, -2, 5, 0, 1, 8, -1, 0, 5, -4, 0, -2, -1,
0, -2, 0, 0, -5, 0, 0, -1, 2, 0, -2, 0, 0, 1, 2, 0, 0, 0, 7, 0, 0, 0, -2, 11, 0, 6, 0,
0, 0, 0, -1, -1, 0, 3, -1, -2, 4, 4, 0, 0, 0, -5, 0, -6, -3, 5, 0, 0, 6, 0, 0, -7, -4,
0, 0, -1, 7, 0, 0, 0, -1, -4, -1, 2, -1, 0, -4, 0, 0, -2, 0, 2, 0, 0, -2, 0, 0, 0,
0, 0, 6, -8, 0, -1, 0, 1, 0, 0, 0, 3, -2, 0, -8, 0, -3, 0, -3, 0, 0, 0, 8, 0, 0, 0,
0, -2, 0, 0, 0, -11, 0, -1, -8, -4, 4, 0, -7, -10, 0, 0, 0, -1, 0, -3, 0, 0, 0, 0, 0, 0,
1, -3, -16, 3, 0, 4, -3, 0, 6, 0, 0, -8, 0, -2, -1, 0, 0, -6, -1, 0, -3, 0, -1, 0, 0, 0,
0, 0, -2, 0, 0, 0, -2, -2, 0, 0, 0, 0, 0, 0, 7, 1, 0, 7, -4, -1, 0, -6, 0, -5, 0,
-2, -8, -5, 15, -9, 0, 2, 11, -6, 0, 0, 0, 0, 5, 10, 0, -2, -2, 0, 0, -2, 0, -3, 0, 0,
0, 0, 0, 0, 0, -4, 1, -1, 8, 7, 0, -2, 0, -1, 0, 0, 0, 0, 0, -7, 0, -3, 0, 0, 0,

4, -1, -2, 0, 0, -1, 0, 1, 0, 0, 4, 0, 0, -4, 0, -1, -5, -1, -1, 0, 7, 4, -1, -1, -12,
 0, -3, 0, 0, -7, 0, 0, -2, 0, 0, 0, 0, 2, 0, -4, 0, 4, -2, 0, 0, -1, 0, 0, 13, 0,
 0, 0, -2, 6, 0, -4, -1, 0, 9, -4, 0, -2, 0, 0, 6, 0, -12, -11, -10, -11, 0, 0, 0, 0,
 0, 0, -1, -1, -1, -7, -6, -3, 5, -1, 0, 0, 0, 0, -14, -11, 0, 0, 0, 3, -1, 8, -10, 0, 0,
 0, 0, -1, 0, 0, -2, -15, 0, 2, -1, -1, 0, -1, 9, 0, 0, -3, 0, 0, 0, -5, 0, 0, 0, 0, -5,
 0, 0, 0, 0, 1, -5, 0, 0, -2, 0, -2, 0, -3, 0, 0, 0, 0, -7, -1, -2, 0, -5, -2, -4, 0,
 0, 3, -6, 0, -8, 0, -4, -2, -2, -3, 12, 2, 0, 0, -5, -8, 0, 0, 0, -1, 0, 0, 3, 0, 0,
 0, 7, 0, 11, -14, 0, 0, 0, 0, 0, 0, 3, -2, 0, -1, 0, 6, -1, 3, 0, 4, 0, -8, 0, -8, -1,
 0, 0, 0, -4, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, -6, -1, 0, 1, 0, -2, 9, -11, -5, -2, -
 8, 3, -7, 0, 0, 0, 0, -1, 0, 13, 0, -1, -5, -4, -8, 0, 0, -2, 0, 0, 0, 0, 0, 0, 0, 0, -
 2, -1, 0, 0, 0, 3, 0, 0, 0, -1, 0, 4, 0, -10, -2, 0, 0, 0, -2, -2, 0, 0, -1, 4, -2
 0, -5, 0, 1, 0, 0, 7, 0, 0, 0, -1, 1, -1, -2, 0, 0, -5, 0, -2, -1, 0, 0, -2, -3, 6, -1,
 0, 2, 3, 0, 0, -12, -1, -2, 0, -9, 0, 1, -6, -2, 0, 0, 0, 6, 0, 0, -6, 0, 0, 0, -7,
 0, 0, 0, 0, -1, 0, -2, 0, 8, -3, -1, 0, -7, 0, 0, 0, -2, 0, 2, -2, 1, 0, -2, 7, 0, -
 8, 0, 11, -1, -1, 0, 0, 0, 0, -5, -8, -2, 4, 0, -3, 9, 0, -2, 0, 0, 0, 0, 0, 0, -
 3, -7, 0, 5, 0, 2, 0, 0, 0, 0, 0, -1, -17, 0, 0, -1, -7, -2, 0, 2, 0, 9, 0, 0, -1, -
 1, -2, 0, 6, 0, -19, 0, 12, -14, -2, -8, -1, -7, 4, 0, 0, 0, 0, 2, 0, 8, -1, 0, 0, 0, -
 3, 0, -2, 0, 0, 7, 3, 0, 6, -7, 0, -2, 0, -3, -10, 4, -2, 0, 2, -1, 0, -2, 6, 0, 0, -
 6, -2, 2, -1, -3, 0, -6, 0, 0, 0, 0, -11, 0, 1, 0, -1, 0, -5, 0, 0, -8, 0, 0, -1, 0, -
 -1, 0, -1, 0, -2, -1, 0, -4, 3, -1, 0, 0, 0, 0, 0, 0, -1, -4, 2, 0, -5, 8, 6, 0, 0,
 0, 0, 0, -1, -1, 4, -5, 1, 0, 0, 8, 5, -1, -9, -1, -1, 0, 0, 0, -1, 0, 4, 3, 10, 3, 6,
 0, 8, -1, 0, -8, -12, 0, 13, 0, -6, -2, -1, 0, 0, 0, -1, 1, 0, 0, -2, -7, 1, 0, 0, -1, -
 4, 0, -1, 0, 0, -1, 10, 0, 0, 0, 0, 0, 3, 0, 0, 1, 0, 0, -1, 0, -9, 0, 0, -3, 0, 0, -
 12, 0, 0, 0, 4, 4, 0, -1, 0, 0, -1, 0, 0, 0, 0, 4, -1, -1, 0, 0, 0, 0, 11, -1, -
 4, -4, 0, 0, 0, 1, 5, 0, 0, -6, 0, 0, 0, 0, 0, 0, 0, -1, -7, -2, 0, 0, 0, -16, -
 5, -2, -2, -1, 11, 0, 0, -1, -1, -7, 0, 4, -1, -3, -1, -5, 0, 7, 0, 0, -2, 6, -2, 4, 0,
 0, 1, -2, -3, 0, -7, -1, 0, 0, 0, 0, -13, 5, 0, 12, 1, 0, 9, 0, -18, 0, -3, -4, 0, -
 4, 0, 0, 4, 0, -1, 0, 0, -1, -2, -2, 0, 0, 0, 0, 3, -2, -10, 0, -8, -3, 0, 0, -7, 3, 5,
 0, 1, -2, -2, 0, 0, -1, 1, -1, -3, 0, 0, 2, -4, -10, -4, -8, 0, 0, -1, -6, 0, 0, 0,
 7, -9, 0, 0, 0, 5, 0, 0, 0, 0, 4, 0, 0, 0, 0, 6, 1, 0, -3, 0, 0, 0, 0, -8, 0, -4,
 0, -12, 0, 11, 0, 0, -1, 1, 0, -2, -15, -1, -2, -1, 0, 0, 0, 0, -5, 0, 0, 0, 0, -1
 0, 6, -6, 0, 8, 0, -7, 0, -1, 0, 0, -6, 0, 0, 0, -3, -2, 0, 0, -2, 0, 0, 3, -7, -2, 0,
 0, 0, 0, 0, 7, 0, 0, 0, 0, -10, 0, 0, 3, 0, -2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, -4, 0, 0, 0, -1, 0, 0, 0, 0, 11, 0, 0, 0, 0, -8, 0, 0, 9, -1, 0, 0, -2, 6,
 0, 1, 0, -7, -9, 0, 0, 0, 0, -2, 0, 0, 0, 4, -1, 0, 0, 0, 0, 4, 7, 0, 12, -1, 0, 0,
 0, -6, -1, -1, -2, -3, -7, 0, 0, 1, 2, -2, -5, -7, 0, 0, 0, -2, -3, 0, -1, -2, 0, -2, 0,
 0, -10, 0, -12, 0, -2, 0, -1, 0, 1, 0, 0, -2, 0, -1, 0, 0, 17, 4, -3, 0, 0, 4, -9, -2,
 0, -3, 0, -8, 0, 0, 0, -3, -5, 0, 0, 0, 0, -1, 0, 0, 0, 0, -1, 0, 2, -1, 0, -2, 9, -
 5, 0, -1, -1, 0, 0, -5, 0, 0, -6, -3, 0, 0, -3, 0, 0, -6, -1, 0, 0, 0, -1, -1, 0, 0,
 0, 2, 0, 0, 7, -1, 9, 4, 0, 4, 0, -5, 13, 0, 0, 0, 0, -2, 0, -2, 0, -2, 0, 3, 0, 0,
 1, -7, -5, 4, 0, -5, 0, 0, 0, 0, 0, -5, -1, 0, 0, 0, 0, -1, 8, -5, 0, -5, 1, 0, 0, 0,
 11, 0, -4, 0, -2, 0, 0, 0, -4, 1, -1, 0, 1, 0, 0, -6, 0, -10, 0, 0, -3, -5, 1, -8, 4,
 -2, -2, 0, 0, 0, -1, -1, 0, -6, -4, 0, 0, 0, 0, 0, -1, 4, 0, 1, 0, 0, 0, 0, 7, -9, 1,
 0, 0, 0, -4, -6, 0, 0, -1, 0, 1, 0, 0, 1, 0, 13, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 -2, 0, 0, 0, 6, -1, 7, 0, 0, 0, -9, -7, 5, -7, -2, -2, 0, 0, 0, 4, 0, 11, 0, 4, -1, 0, -
 1, -2, -1, -1, 0, 0, 0, 8, -2, 0, -4, 0, -10, 0, 0, 0, 0, 9, 0, 9, -8, 7, 0, 6, 0, -1,
 0, 0, 0, -1, 0, 0, 0, -2, -7, 14, -1, 0, 0, 10, -2, 0, -1, -2, 4, -1, 0, 0, 0, 0, -1,
 0, 6, 12, 0, 0, 7, -2, -3, 3, 5, 0, 0, -5, 0, -8, -6, -18, -7, -5, 0, 2, 0, -7, -2, -5,
 0, 0, 5, 0, -1, -13, 0, 0, -9, 0, 0, 0, -3, 2, 0, 0, 0, -1, -3, 0, 0, 0, 3, 0, 0, 0, -1
 5, 3, 0, 0, -10, -6, 0, -4, -7, -3, 0, 1, -6, -1, 0, 0, 0, 0, -2, -21, 0, 0, 0, 0, -
 6, -2, 0, -7, 0, -9, 0, -9, 0, -5, 0, -2, -2, 0, -2, -18, 0, -1, -10, 0, -2, -9, -5, -1, 0,
 7, 0, 0, -1, -9, 0, 0, -2, 0, 13, 0, 0, -14, -7, 0, 0, -17, 0, 3, 0, -1, 0, -1, 1, -3, -
 2, -9, 0, 0, -1, -9, 0, -2, 0, -9, 0, 0, 1, 0, 0, 0, 0, 0, -2, -6, 0, -1, 0, -2, 7, 1
 3, -2, 0, 0, 0, -4, 0, -2, -6, 0, -12, 0, 0, 0, 0, -1, 0, 1, 0, -2, -6, 0, -1, 0, -2, 2, 0,
 -2, -2, 0, -2, 0, 0, -1, 0, 0, -2, 9, 0, 0, 0, 0, -2, 0, -1, 0, -3, 0, 0, 1, -1,
 1, 0, -1, 0, 0, -7, 0, 0, -5, 0, 0, -5, -2, -2, 0, 0, -11, -2, 0, 0, 0, 0, -2, 8, -1,
 0, 0, 0, 15, 2, -1, -2, 0, 1, 0, 0, 0, 0, 5, 7, -3, -5, 0, -1, 0, -7, 0, -6, 0, 0, -2, -
 4, -14, -1, 0, 0, 0, 0, -2, 0, 0, 0, 4, 11, 0, 3, 1, -5, -1, 0, -1, -2, 0, -1, 5, 0,
 8, 0, 0, 0, -5, 0, 0, 0, 0, 0, -3, -8, 0, -5, 0, 0, -2, 0, 0, 0, -10, 0, 0, -3, -1,
 0, 0, 0, 0, -3, 0, -1, 0, -7, -6, 0, -6, 0, -3, -3, -2, -7, 0, -2, 0, -3, 0, -14, 0, 0,
 7, -13, -2, -2, 0, 3, -5, -2, 3, 0, 0, 1, 7, 0, 0, 3, 0, -2, 0, 0, -9, -1, -15, 7, 0, -
 1, 0, 0, -5, -2, 0, -3, -3, 0, -2, 0, 0, -1, -1, -2, 0, -4, 0, 0, 0, 0, 0, -3, -1, 0, 0,
 -3, 0, 0, 0, -6, -1, 11, 0, 0, 0, 0, 0, 0, -3, -2, 0, -15, -6, -8, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 2, 0, 0, -1, -6, 0, 0, 0, -2, 0, 4, 0, 0, 0, 0, 4, -1, 3, 6, 0, -11, -2, 0, 2, 0,
 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, -2, 0, 0, -7, -2, -1, -1, 0, 1, -3, 0, 0, 0, 2, 0, 0, 0,
 0, 0, 7, -1, -2, -1, -2, -7, 0, 0, -2, -7, 5, -1, 0, -1, 0, 0, 0, 0, 0, 0, 0, -11, 0,

```

0, 0, 0, 0, 0, -1, 0, 6, -10, 0, 9, 0, 0, 0, 0, 0, 0, -13, 0, 7, 5, 3, 0, -1,
-7, 0, -4, 0, 0, -1, -2, 0, 0, 8, -19, 1, 0, 0, -3, 0, 0, -15, 0, 0, 0, 0, 3, 3, 0, 0,
1, -1, 0, -6, -4, -9, 0, 0, -1, 0, -11, 6, -1, -6, 0, -2, 0, -2, -2, 0, 0, 4, 0, -2, 0,
0, 0, 0, 10, 0, 0, -10, 0, 0, 0, 7, 0, -2, 0, 0, -6, 0, 0, -5, 0, -4, 0, -6, -13, -1
2, 0, 0, -2, -6, -6, -2, 0, 0, 0, -11, 0, -2, 0, 0, -3, 0, -3, 2, 0, 0, 0, 7, -3, 10, 0,
-17, -10, -4, -1, -19, 0, 0, -2, 0, 0, 0, 5, 2, 0, 0, 0, -10, 0, 0, -5, 0, -2, 0, -2, -
1, 0, 0, 0, 0, 0, 0, -1, -3, -1, -13, -2, 6, 0, 0, 0, 0, 7]
Dimensions : [11, 100, 10, 7]
Time Taken : 8389.378967523575

```



```
In [7]: print("\n\n===== MLP Program Begins =====")

start_time = time.time()
print("Training")
m = MultiLayerPerceptron(fileName="../ANN/winequality-white", dimensions=dim, all_weight=m.main())
end_time = time.time()
print("Time taken = ", end_time - start_time)
```

```
===== MLP Program Begins =====
Training
Time taken =  0.4777228832244873
```

```
In [8]: start_time = time.time()
print("Testing")
m = MultiLayerPerceptron(fileName="../ANN/winequality-white", dimensions=dim, all_weight=m.main())

end_time = time.time()
print("Time taken = ", end_time - start_time)
```

```
Testing
Time taken =  0.43782925605773926
```

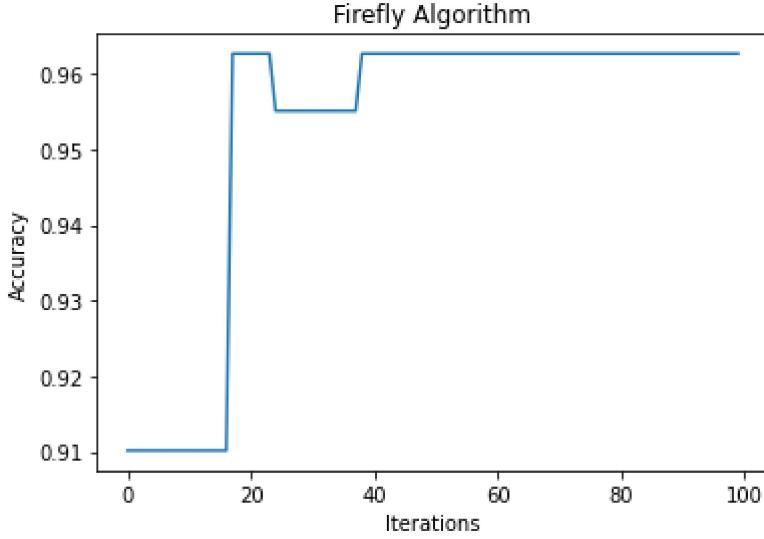
```
In [9]: all_accuracy = []
for weights in all_gen_best_weight:
    m = MultiLayerPerceptron(fileName="../ANN/winequality-white", dimensions=dim, all_weight=accuracy_val = m.main()
    print(accuracy_val)
    all_accuracy.append(accuracy_val)

import matplotlib.pyplot as plt
x=all_accuracy[:]
z=[i for i in range(len(x))]
plt.plot(z,x)

plt.title("Firefly Algorithm")
```

```
plt.ylabel("Accuracy")  
plt.xlabel("Iterations")
```

```
Out[9]: text(0.5, 0, iterations )
```



In []: