accuracy - ffaANN MLP V0.1

May 20, 2022

[ ]:

```

```

[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 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 accffaAnn_thread_V_I import *



class MultiLayerPerceptron():
    # ================== Activation Functions ================ #

    # accepts a vector or list and returns a list after performing␣
↪corresponding function on all elements

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

```python
        return sig

    @staticmethod
    def binaryStep(x):
        """ It returns '0' is the input is less then 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 lies in between -1 to 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 given input"""
        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):
```

```python
        """ 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((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 type(cord2) == float))):
            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 type(cord2) == float))):
            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
```

```python
    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 type(cord2) == float))):
            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 type(cord2) == float))):
            dist = math.fabs(cord1 - cord2)
        else:
            for i, j in zip(cord1, cord2):
                dist += math.fabs(i - j)
        return dist

    @staticmethod
    def eucledian(self, cord1, cord2):
        dist = 0.0
        if ((type(cord1) == int and type(cord2) == int) or ((type(cord1) ==␣
↪float and type(cord2) == float))):
            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"):

        """
        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)

        classes = []
        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)):
```

```python
            b = []
            for i in range(1, 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.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
```

```python
        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 = 3

        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

        #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 = ['class 0', 'class 1', 'class 2']
        #print(classification_report(Y_ACTUAL, Y_PREDICT,␣
↪target_names=target_names))
        #print("\n\n\n")
        return accuracy_score(Y_ACTUAL, Y_PREDICT)
```

[6]:

```python
start_time = time.time()
i = InputData(fileName="iris")
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, output_values_expected=output_val, iterations = 100)

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 :  0.028992414474487305
============ Calling FFA to get best weights ===============
--------------GENERATION 0-----------
Initial worst fitness =  939.1697997791503

 Initial best fitness =  368.3201652603337
--------------GENERATION 1-----------
--------------GENERATION 2-----------
--------------GENERATION 3-----------
hi
hi
hi
hi
hi
```

```
hi
hi
hi
hi
hi
--------------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-----------
hi
hi
hi
hi
hihi

hi
hi
hi
hi
--------------GENERATION 19-----------
hi
hi
hi
hi
hi
hi
hihi
hi

hi
--------------GENERATION 20-----------
hi
hihi
hi

hi
hihi
```

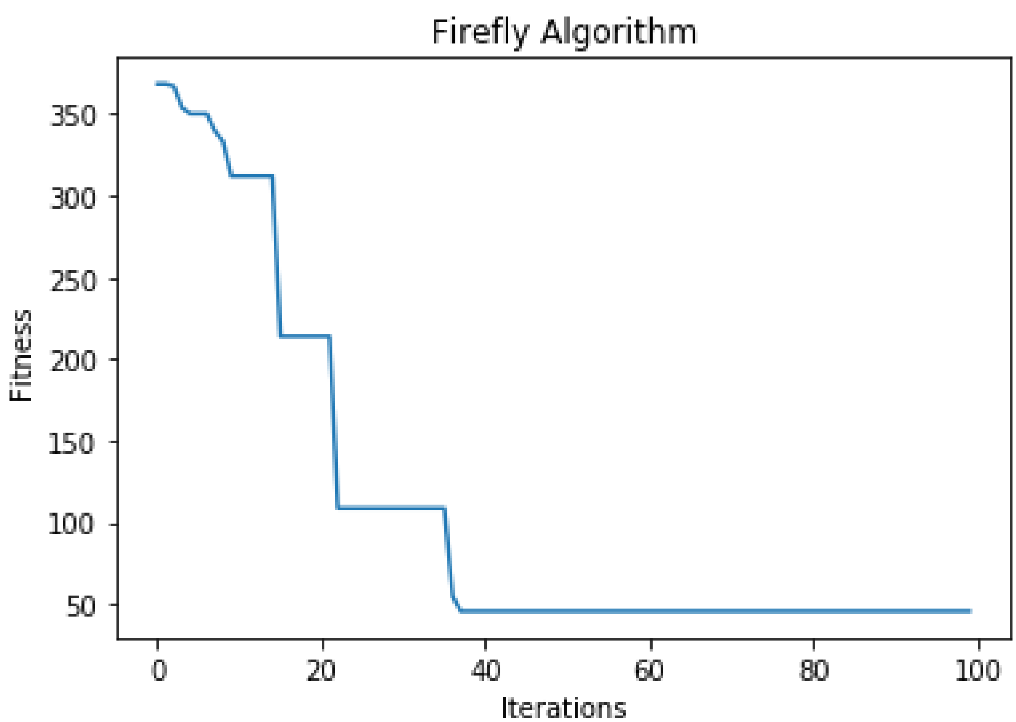
```
hi
hi
hi
--------------GENERATION 21-----------
--------------GENERATION 22-----------
hi
hi
hi
hi
hi
hi
hihi

hi
hi
--------------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 :  45.52211760841647
Time taken :  168.88895511627197
```

Fitness : 45.52211760841647
Best Weights : [12, 1, 19, 12, 0, 18, 25, -3, 2, -7, -13, 0, 0, -5, -18, 0,
15, 2, 4, -17, -11, -13, -10, -6, 3, 10, 2, 6, 7, -1, 17, 9, -3, -7, -20, 14,
-18, 0, -20, 0, 11, 15, -14, -5, 13, 0, 6, -2, 2, -23, 0, 3, -21, 7, 0, 3, 1, 1,
-9, 19, -10, 0, 0, 15, 1, 0, -1, 11, -6, -5, 3, 13, 0, 7, -1, -9, 3, -15, -17,
13, -24, 9, 0, 0, 13, -18, -18, -10, -32, -4, -2, 2, -1, 24, 10, -3, -17, 1, 10,
4, 27, 10, 10, 1, 8, 13, 9, -14, 0, -4, -1, 7, 5, 15, -14, 1, 16, 1, 16, -4, 14,
8, 0, 16, -4, 24, 16, 0, 2, -10, 26, 1, 21, 0, -5, 3, 16, 14, 5, 0, -16, -25,
-9, -4, 0, 12, 17, -15, -1, -12, 0, 0, 6, -7, 4, -4, 0, -13, -5, 14, -6, -9, 0,
-22, 0, -11, 10, 0, -13, -3, -13, -6, 0, -25, 16, -24, 15, 3, -4, 9, 0, 1, 0, 0,
16, -2, 0, 6, -7, -11, -4, -5, 18, -5, -6, -16, -2, -3, -8, -1, 12, 6, -4, 23,
5, -12, -14, -12, 0, -5, 16, -4, -3, -17, -2, -1, 0, -23, -1, 0, 12, 0, -6, -1,
20, 18, -7, -20, 14, -15, -26, 18, -13, -1, 12, -1, 22, 4, 17, 9, 5, -3, 3, 21,
13, 22, -2, -19, 3, 1, -10, 3, 0, -24, 0, -25, 0, 0, 15, 13, 12, -3, -11, 4,
-12, 7, -1, 2, 0, 0, 13, 35, 6, 4, -4, -7, 6, -1, 4, -3, 0, -2, -24, -1, -15,
-1, -11, 12, 12, -1, 29, 13, -5, -15, -10, -5, -15, 21, -4, 0, 2, -8, -11, 3,
-6, -2, 0, -10, -20, 10, 1, -7, 0, 13, 2, -21, -24, 9, -5, 2, 9, -4, 14, 8, 0,
-3, 28, -21, 8, -11, 0, -8, -2, -7, -5, 8, -17, -1, -1, -30, 4, -11, 3, 0, 1,
-13, 4, 10, 16, 6, -19, 24, 8, 22, 5, 1, 0, 0, 0, -20, -16, 25, -15, -2, -2, -9,
3, 8, 0, -2, -10, -18, -8, -14, 4, 0, -5, 8, 13, -1, -17, 7, 15, 7, 21, 4, -22,
-15, -18, -23, 1, 10, 24, 12, -5, 3, 1, -8, 4, 16, 14, 3, -6, 29, -12, -32, 5,
0, -13, -2, -21, -15, 26, 11, 7, 2, 7, -7, -1, -17, -18, 1, -21, -19, 3, -26,
10, -1, 4, -20, 5, 11, 12, -1, -11, 20, 0, 0, 0, -11, 13, -22, 7, -15, -8, 0, 0,
12, 9, 15, -13, -11, 3, -5, -6, 22, 19, 17, 1, 5, -1, -1, -8, -13, 1, 31, 5, -7,
-6, 8, 10, 5, 21, -14, -15, 10, -2, -16, 8, -27, -21, 22, 23, 3, 4, 0, 14, -4,
11, 9, 8, -9, 7, 4, 0, 6, -16, 20, -13, 0, 17, -15, 0, 0, 9, 22, 0, 20, -1, -7,
-6, -4, -11, -17, 0, -19, 1, 8, -3, 0, -3, 8, 0, 9, -4, -2, 14, 7, 14, -6, 22,
15, -4, -4, 0, -7, 13, -1, 11, -11, -16, -34, 2, -16, 5, 1, 24, -15, -14, 14,
-4, -4, 7, 8, 6, 14, 2, -4, -19, 0, -18, -9, -14, 0, 0, -2, 5, 0, 8, 6, 9, 2, 6,
-19, -10, -6, 0, 15, 14, -12, -15, 5, 9, -9, -6, -1, 30, 1, -14, -3, -17, -6,
21, -3, -26, -1, 11, 11, 0, 14, 16, 17, -12, -26, 6, 0, 0, 13, 16, -16, 11, -14,
0, 3, 4, 17, -4, 0, 24, 8, -4, -9, -20, 15, 15, 14, 0, 3, 0, -10, -10, -19, -19,
-18, -1, -6, 14, -36, -5, 0, -20, -20, -13, -11, -1, -19, -4, -2, -16, -13, 14,
-1, 3, 0, -10, 0, 9, -18, -5, -11, -7, 6, 12, -1, -4, 16, -21, 11, -4, 11, -5,
7, -1, 15, -2, 2, -14, 5, 4, -6, -12, 9, 3, -1, 13, -11, -12, -23, -3, 10, -18,
-9, 0, 0, 16, 0, -12, -17, -12, 22, -3, -7, 0, -6, -19, 1, -2, -6, -15, -9, 9,
3, -2, 4, -7, 1, 11, 4, -24, 26, 17, 15, 6, 12, -10, 30, 1, -18, 10, 0, 4, 8, 6,
0, -19, -7, -1, 0, 0, -1, 0, -10, 0, -10, -10, 4, 21, 18, 25, 0, 1, -16, -4, 5,
11, -26, 0, 1, 7, -1, -5, 8, -8, -3, 19, -1, 12, 0, -5, -2, 3, 2, -16, -15, 0,
2, 2, 2, 22, 8, -8, -21, -5, 16, 17, 0, 29, 9, -11, -13, -2, 4, -3, 7, -8, 4,
-8, 1, 0, -14, 15, 16, 2, -6, 2, 0, -8, -1, 3, -24, -18, -3, 0, 8, -3, -16, 21,
3, 0, -10, 14, 10, -13, 1, 18, -4, 8, -18, 19, -1, 8, -19, -2, 0, -16, 31, -17,
31, 0, -5, -5, -1, -2, 15, -18, -9, 3, 16, 0, -3, -6, 1, -2, 12, 13, 4, 19, -1,
2, -7, 11, 4, -4, -16, 0, 2, -7, 3, -1, 0, -11, 18, 11, 8, 6, 4, 2, 0, 0, 9,
-15, -10, -9, -8, 9, -2, 2, -11, -14, 18, 11, 8, -7, -4, 6, 9, 0, 10, -13, 0, 0,
-3, 21, 24, 5, 2, 13, 2, 15, 28, -17, 9, 10, 17, 1, 0, 0, 10, 10, -14, -2, -20,
-14, 4, 24, 0, 0, -17, 0, -5, -13, 7, 16, 0, -2, 17, 17, -7, 23, 12, 0, -12,
-20, -1, 8, 3, 1, 6, 3, 7, -23, 17, -5, 14, -5, -7, 11, 21, 9, -15, -17, 4, 7,

```
13, 1, 0, -3, -10, -7, 21, 23, -2, -9, 12, 4, -1, -17, 0, 20, 3, 2, -24, 29, -4,
0, -1, -19, -13, 22, 0, 0, 1, 18, 0, -22, 3, -8, -18, 3, 0, 2, -6, -7, 18, 0, 0,
17, -13, -4, -15, 2, 10, -3, 7, 2, 18, -2, 0, 2, 8, 1, -10, 15, 16, -11, 6, 5,
16, -3, 16, 0, 0, -5, 19, 1, -6, 11, -4, 7, -2, -20, -6, -9, 0, 9, -11, 0, 15,
5, 9, -6, 20, -20, -25, 0, 4, -8, -7, 15, 9, -12, -19, -1, -12, -24, 9, 6, 1,
-26, -7, -25, 9, 5, 13, 5, -12, 18, 2, -1, -9, 0, 9, 4, -3, 0, 6, 5, 8, 4, -19,
4, -5, 17, -18, -33, -1, -17, 0, 28, -15, -3, -10, -34, 0, 6, 3, -12, 7, -14, 0,
1, 16, -21, -5, -1, 18, 12, 0, -13, 0, 3, 9, 1, 0, 9, -17, 19, 2, 15, 6, 4, 4,
0, 1, -12, 10, 5, -3, 3, -15, 13, 4, 20, -19, 7, 3, 21, 0, 0, -5, -16, -4, 19,
0, 0, 10, 1, 2, -27, -14, -1, 16, 9, -4, -9, 4, 0, -1, 1, 5, 27, -2, 10, -12, 2,
0, -2, 19, 0, -1, -10, -8, -8, -15, 26, 4, -15, -4, -15, 2, 7, 1, 13, -8, -8,
-1, 9, -1, -14, 8, 3, -1, 8, 11, -13, 0, -14, -10, -20, 0, -11, 20, -17, 18, 16,
-5, -15, 20, 22, -10, 0, -9, 0, -12, -11, 0, -20, -1, 18, -7, 15, -5, 11, 2, 4,
-6, 13, 0, 8, 4, -1, -16, -2, 4, 25, 8, 11, -19, -12, -1, 9, 13, -11, 5, -28,
-16, 13, 6, 16, 4, 4, 4, 0, -14, 0, -6, -8, -2, 22, -4, 0, -3, -4, -1, -1, -8,
0, -12, -2, -3, 7, -8, -16, -2, 5, -14, 2, 4, -14, 19, 5, 2, 0, 0, -8, 0, -18,
0, -11, -10, 14, 0, 25, 8, -35, 4, 4, 0, 0, 0, 0, 18, 9, 18, 0, 4, -20, 1, 18,
-22, 5, 9, -2, 0, 0, 0, 6, 8, 9, 33, -5, -1, -6, 3, -2, -3, 8, 1, 27, 5, -18, 9,
-9, -1, -1, -18, 0, -17, 10, 6, -13, 0, 17, 0, 20, 6, 16, -7, 3, -18, 9, 20,
-15, -19, 5, 1, -2, 4, 1, -13, 5, 0, -4, -3, -10, -19, -1, -2, 1, 8, -2, 0, 0,
-9, 12, -8, -6, 1, -14, -1, 5, 7, -4, 11, -29, 9, 25, -1, -15, -6, -6, 5, 13, 8,
0, -8, -14, 4, -25, 8, 14, -15, -15, -3, 9, 5, -12, 19, -10, 0, -19, -9, 11, 5,
11, -20, 15, -10, -1, 7]
 Dimensions :  [4, 100, 10, 3]
Time Taken :  168.90278959274292
```



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

start_time = time.time()
print("Training")
m = MultiLayerPerceptron(fileName="iris_train", dimensions=dim,␣
 ↪all_weights=weights)
m.main()
end_time = time.time()
print("Time taken = ", end_time - start_time)
```

```


============= MLP Program Begins ============
Training
Time taken =  0.019589900970458984
```

[8]:
```
start_time = time.time()
print("Testing")
m = MultiLayerPerceptron(fileName="iris_test", dimensions=dim,␣
 ↪all_weights=weights)
m.main()

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

```
Testing
Time taken =  0.008346796035766602
```

[9]:
```
all_accuracy = []
for weights in all_gen_best_weight:
    m = MultiLayerPerceptron(fileName="iris_train", dimensions=[4,100,10,3],␣
 ↪all_weights=weights)
    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")
```

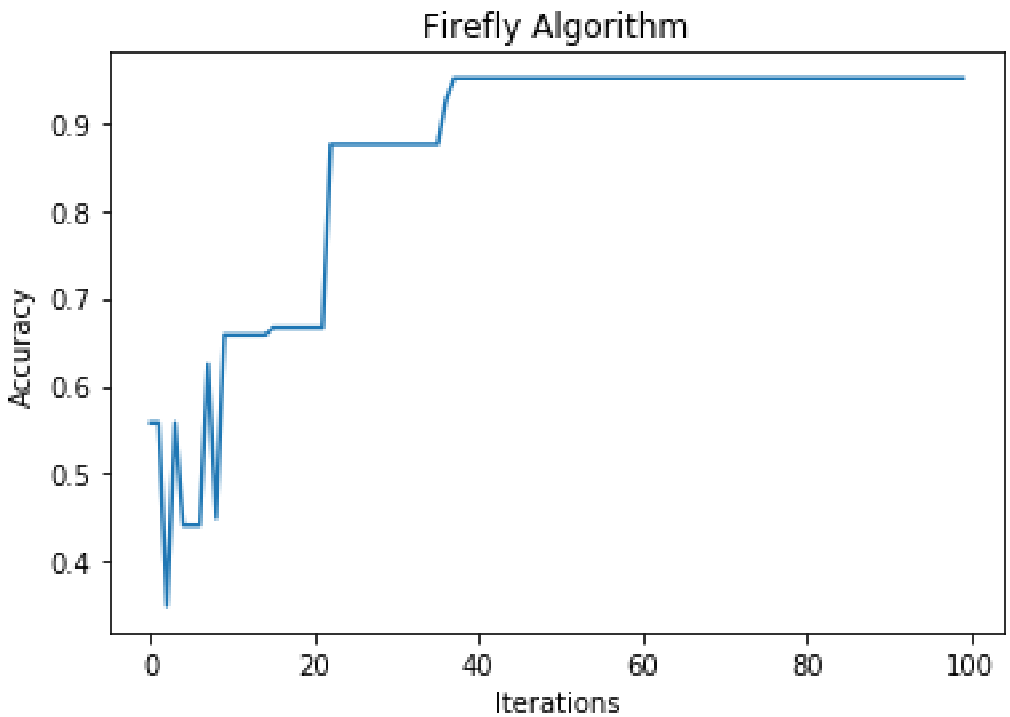
```
0.5583333333333333
```

```
0.5583333333333333
0.35
0.5583333333333333
0.44166666666666665
0.44166666666666665
0.44166666666666665
0.625
0.45
0.6583333333333333
0.6583333333333333
0.6583333333333333
0.6583333333333333
0.6583333333333333
0.6583333333333333
0.6666666666666666
0.6666666666666666
0.6666666666666666
0.6666666666666666
0.6666666666666666
0.6666666666666666
0.6666666666666666
0.875
0.875
0.875
0.875
0.875
0.875
0.875
0.875
0.875
0.875
0.875
0.875
0.875
0.875
0.925
0.95
0.95
0.95
0.95
0.95
0.95
0.95
0.95
0.95
0.95
0.95
0.95
```

```
0.95
0.95
0.95
0.95
0.95
0.95
0.95
0.95
0.95
0.95
0.95
0.95
0.95
0.95
0.95
0.95
0.95
0.95
0.95
0.95
0.95
0.95
0.95
0.95
0.95
0.95
0.95
0.95
0.95
0.95
0.95
0.95
0.95
0.95
0.95
0.95
0.95
0.95
0.95
0.95
0.95
0.95
0.95
0.95
0.95
0.95
0.95
0.95
```

```
0.95
0.95
0.95
```

[9]: Text(0.5,0,'Iterations')



[ ]: