PySpark_GA_ANN

May 20, 2022

0.1 Spark Session Creation

[ ]:

```

```

[ ]:

```

```

[1]:

```
from pyspark.sql import SparkSession

spark = SparkSession.builder\
        .master("local")\
        .appName("Colab")\
        .config('spark.ui.port', '4050')\
        .getOrCreate()
```

[2]:

```
spark
```

[2]: <pyspark.sql.session.SparkSession at 0x7f60c0099fd0>

[3]:

```
sc = spark.sparkContext
```

1 Main Code Begins

[4]:

```
#@title
import math
import random
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 scipy.special import expit
```

```
from numpy.random import default_rng

# noinspection SpellCheckingInspection
import time

class InputData():
    # input and output
    # ================== Input dataset and corresponding output␣
 ↪========================= #
    def __init__(self):
        data = pd.read_csv("iris.csv")

        output_values_expected = []
        input_values = []

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

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

        for j in range(len(data)):
            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[:]

    def main(self):
        return (self.X, self.Y)
```

```
[5]: def mean_square_error(expected, predicted):
            total_error = 0.0
            for i in range(len(predicted)):
                total_error += ((predicted[i] - expected[i]) ** 2)
            return (-total_error)

     i = InputData()
     input_val, output_val = i.main()


     def Fitness(x,X=input_val,Y=output_val):
       fitness = []

       """ This code returns [array([-194]), array([-114]) ... ]    ==  The last␣
      ↪value in each chromosome *look above *
```

```python
    for a in x:
      k=np.array([2])*a
    """
    total_error = 0
    m = []
    k1 = 0
    mtemp=x[:]

    for i in range(len(dimension) - 1):
      p = dimension[i]
      q = dimension[i + 1]
      k2 = k1 + p * q
      mt = mtemp[k1:k2]
      m.append(np.reshape(mt, (p, q)))

    for x, y in zip(X, Y):

      yo = x

      for mCount in range(len(m)):
          yo = np.dot(yo, m[mCount])
          yo = 1/(1 + np.exp(-yo))

      for i in range(len(yo)):

          total_error += mean_square_error(yo, y)

    return total_error
```

[6]:
```python
class gaAnn():

    def __init__(self, dimensions=(8, 5), initialPopSize=100, iterations=10,␣
↪elicitation_rate=0.01,
                 mutation_rate=0.001, m = 10, bestCount = 50,
                input_values=[] , output_values_expected=[]):
        self.initialPopSize = initialPopSize
        self.allPop_Weights = []
        self.allPopl_Chromosomes = []
        self.allPop_ReceivedOut = []
        self.allPop_ErrorVal = []
        self.n_iterations = iterations
        self.elicitation_rate = elicitation_rate
        self.mutation_rate = mutation_rate
        self.fitness = []

        self.distribution_factor = m
```

```
        self.selection_var = bestCount

        # input and output

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

        self.dimensions = dimensions
        self.dimension = [len(self.X[0])]

        for i in self.dimensions:
            self.dimension.append(i)
        self.dimension.append(len(self.Y[0]))

        # print("Dimension of each layer : ", self.dimension)

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

        self.pop = []  # weights
        for g in range(self.initialPopSize):
            W = []
            for i in range(len(self.dimension) - 1):
                w = np.random.randint(-100, 100, (self.dimension[i + 1], self.dimension[i]))
                W.append(w)
            self.pop.append(W)

        self.init_pop = []  # chromosomes
        for W in self.pop:
            chromosome = []
            for w in W:
                chromosome.extend(w.ravel().tolist())
            self.init_pop.append(chromosome)

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

    def mean_square_error(self, expected, predicted):
        total_error = 0.0
        for i in range(len(predicted)):
            total_error += ((predicted[i] - expected[i]) ** 2)
        return (-total_error)

    def Fitness2(self, chromosome):
      fitness = []

      total_error = 0
      m = []
```

```
    k1 = 0
    mtemp=chromosome[:]

    for i in range(len(self.dimension) - 1):
      p = self.dimension[i]
      q = self.dimension[i + 1]
      k2 = k1 + p * q
      mt = mtemp[k1:k2]
      m.append(np.reshape(mt, (p, q)))

    for x, y in zip(self.X, self.Y):

      yo = x

      for mCount in range(len(m)):
          yo = np.dot(yo, m[mCount])
          yo = 1/(1 + np.exp(-yo))

      for i in range(len(yo)):
          total_error += self.mean_square_error(yo, y)

    return total_error

  def Selection(self, population):
    return population[:self.selection_var]

  def Crossover(self, bestPop):
    children = []
    for i in range(len(bestPop)):
      for j in range(len(bestPop)):
        if (i != j):
          child1 = bestPop[i][:]
          child2 = bestPop[j][:]

          # Performing Crossover
          k = random.randint(0, len(bestPop[i]))
          child1 = child1[:k] + child2[k:]
          child2 = child1[k:] + child2[:k]
          children.append(child1)
          children.append(child2)
    return children


  def mutation(self, bestPop):
      mutRate = 1
      temp = self.mutation_rate
      while (int(temp) != 1):
```

```
            temp *= 10
            mutRate *= 10
        chance = random.randint(1, int(mutRate) + 1)
        if (chance == mutRate):
            i = random.randint(0, len(bestPop) - 1)    # Selecting a random chromosome from the best population
            # print(i, len(bestPop))
            k = random.randint(0, len(bestPop[i]) - 1) # Selecting a random gene position on the selected chromosome
            t = random.randint(-100, 100)              # Selecting a random weight value - (as per line 178)
            sol = bestPop[i]
            sol[k] = t
        return bestPop

    def main(self):
      population = self.init_pop
      iterations = 0
      fitness = []
      best_per_gen = []
      while (iterations < self.n_iterations):  # Maximum Iteration Count = 100
        print("--------------GENERATION " + str(iterations) + "-----------")
        iterations += 1

        # Step 2: Calculate Fitness
        population_data = population[:]

        rdd = sc.parallelize(population_data)
        rdd2 = rdd.map(Fitness).collect()

        # cannot use this
        # rdd2 = rdd.map(self.Fitness2).collect()

        # print(rdd2)
        fitness = rdd2

        sorted_population = [x for y, x in sorted(zip(fitness, population))]
        fitness = [x for x, y in sorted(zip(fitness, population))]

        print(-fitness[-1])

        best_per_gen.append(-fitness[-1])


        # Step 3: Selection
        bestPop = self.Selection(population)[:]
```

```
        # Step 4: Crossover
        children = self.Crossover(bestPop)


        # Step 5: Mutation
        children = self.mutation(children)

        # Elitism
        elitRate = self.elicitation_rate
        temp1 = elitRate * len(children)
        temp1 = int(temp1)
        next_gen = sorted_population+children[:temp1]
        population = next_gen[:]

    print("Fitness : ", -fitness[-1])
    return (-fitness[-1], best_per_gen, sorted_population[-1], self.dimension)
```

[7]:

```
start_time = time.time()
n_iterations = 10
e_rate = 0.1
i = InputData()
input_val, output_val = i.main()
dimension = [100,10]
a = gaAnn(initialPopSize=100, m = 10, dimensions = dimension, bestCount = 30,␣
 ↪input_values=input_val , output_values_expected=output_val, iterations =␣
 ↪n_iterations, elicitation_rate = e_rate)
dimension = [4,100,10,3]
fit, b, weights, dim = a.main()
end_time = time.time()
print(end_time - start_time)
```

```
--------------GENERATION 0-----------
294.0067479215612
--------------GENERATION 1-----------
237.87149441494398
--------------GENERATION 2-----------
237.87149441494398
--------------GENERATION 3-----------
237.87149441494398
--------------GENERATION 4-----------
237.87149441494398
--------------GENERATION 5-----------
237.87149441494398
--------------GENERATION 6-----------
237.87149441494398
--------------GENERATION 7-----------
237.87149441494398
```

```
---------------GENERATION 8-----------
237.87149441494398
---------------GENERATION 9-----------
237.87149441494398
Fitness :  237.87149441494398
21.618762969970703
```

[8]:

```python
start_time = time.time()
n_iterations = 100
e_rate = 0.1
i = InputData()
input_val, output_val = i.main()
dimension = [100,10]
a = gaAnn(initialPopSize=100, m = 10, dimensions = dimension, bestCount = 30,␣
 ↪input_values=input_val , output_values_expected=output_val, iterations =␣
 ↪n_iterations, elicitation_rate = e_rate)
dimension = [4,100,10,3]
fit, b, weights, dim = a.main()
end_time = time.time()
print(end_time - start_time)
```

```
---------------GENERATION 0-----------
347.1376601783355
---------------GENERATION 1-----------
204.03270943838572
---------------GENERATION 2-----------
204.03270943838572
---------------GENERATION 3-----------
204.03270943838572
---------------GENERATION 4-----------
204.03270943838572
---------------GENERATION 5-----------
204.03270943838572
---------------GENERATION 6-----------
204.03270943838572
---------------GENERATION 7-----------
204.03270943838572
---------------GENERATION 8-----------
204.03270943838572
---------------GENERATION 9-----------
204.03270943838572
---------------GENERATION 10-----------
204.03270943838572
---------------GENERATION 11-----------
204.03270943838572
---------------GENERATION 12-----------
204.03270943838572
---------------GENERATION 13-----------
```

```
204.03270943838572
--------------GENERATION 14-----------
204.03270943838572
--------------GENERATION 15-----------
204.03270943838572
--------------GENERATION 16-----------
204.03270943838572
--------------GENERATION 17-----------
204.03270943838572
--------------GENERATION 18-----------
204.03270943838572
--------------GENERATION 19-----------
204.03270943838572
--------------GENERATION 20-----------
204.03270943838572
--------------GENERATION 21-----------
204.03270943838572
--------------GENERATION 22-----------
204.03270943838572
--------------GENERATION 23-----------
204.03270943838572
--------------GENERATION 24-----------
204.03270943838572
--------------GENERATION 25-----------
204.03270943838572
--------------GENERATION 26-----------
204.03270943838572
--------------GENERATION 27-----------
204.03270943838572
--------------GENERATION 28-----------
204.03270943838572
--------------GENERATION 29-----------
204.03270943838572
--------------GENERATION 30-----------
204.03270943838572
--------------GENERATION 31-----------
204.03270943838572
--------------GENERATION 32-----------
204.03270943838572
--------------GENERATION 33-----------
204.03270943838572
--------------GENERATION 34-----------
204.03270943838572
--------------GENERATION 35-----------
204.03270943838572
--------------GENERATION 36-----------
204.03270943838572
--------------GENERATION 37-----------
```

```
204.03270943838572
--------------GENERATION 38-----------
204.03270943838572
--------------GENERATION 39-----------
204.03270943838572
--------------GENERATION 40-----------
204.03270943838572
--------------GENERATION 41-----------
204.03270943838572
--------------GENERATION 42-----------
204.03270943838572
--------------GENERATION 43-----------
204.03270943838572
--------------GENERATION 44-----------
204.03270943838572
--------------GENERATION 45-----------
204.03270943838572
--------------GENERATION 46-----------
204.03270943838572
--------------GENERATION 47-----------
204.03270943838572
--------------GENERATION 48-----------
204.03270943838572
--------------GENERATION 49-----------
204.03270943838572
--------------GENERATION 50-----------
204.03270943838572
--------------GENERATION 51-----------
204.03270943838572
--------------GENERATION 52-----------
204.03270943838572
--------------GENERATION 53-----------
204.03270943838572
--------------GENERATION 54-----------
204.03270943838572
--------------GENERATION 55-----------
204.03270943838572
--------------GENERATION 56-----------
204.03270943838572
--------------GENERATION 57-----------
204.03270943838572
--------------GENERATION 58-----------
204.03270943838572
--------------GENERATION 59-----------
204.03270943838572
--------------GENERATION 60-----------
204.03270943838572
--------------GENERATION 61-----------
```

```
204.03270943838572
--------------GENERATION 62-----------
204.03270943838572
--------------GENERATION 63-----------
204.03270943838572
--------------GENERATION 64-----------
204.03270943838572
--------------GENERATION 65-----------
204.03270943838572
--------------GENERATION 66-----------
204.03270943838572
--------------GENERATION 67-----------
204.03270943838572
--------------GENERATION 68-----------
204.03270943838572
--------------GENERATION 69-----------
204.03270943838572
--------------GENERATION 70-----------
204.03270943838572
--------------GENERATION 71-----------
204.03270943838572
--------------GENERATION 72-----------
204.03270943838572
--------------GENERATION 73-----------
204.03270943838572
--------------GENERATION 74-----------
204.03270943838572
--------------GENERATION 75-----------
204.03270943838572
--------------GENERATION 76-----------
204.03270943838572
--------------GENERATION 77-----------
204.03270943838572
--------------GENERATION 78-----------
204.03270943838572
--------------GENERATION 79-----------
204.03270943838572
--------------GENERATION 80-----------
204.03270943838572
--------------GENERATION 81-----------
204.03270943838572
--------------GENERATION 82-----------
204.03270943838572
--------------GENERATION 83-----------
204.03270943838572
--------------GENERATION 84-----------
204.03270943838572
--------------GENERATION 85-----------
```

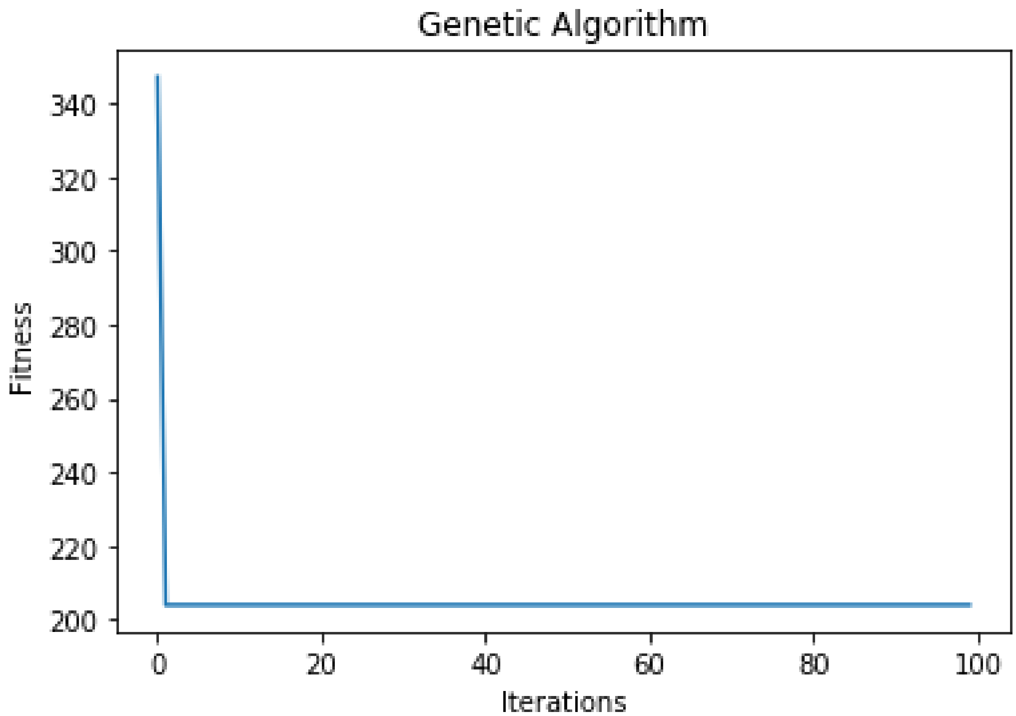
```
204.03270943838572
--------------GENERATION 86-----------
204.03270943838572
--------------GENERATION 87-----------
204.03270943838572
--------------GENERATION 88-----------
204.03270943838572
--------------GENERATION 89-----------
204.03270943838572
--------------GENERATION 90-----------
204.03270943838572
--------------GENERATION 91-----------
204.03270943838572
--------------GENERATION 92-----------
204.03270943838572
--------------GENERATION 93-----------
204.03270943838572
--------------GENERATION 94-----------
204.03270943838572
--------------GENERATION 95-----------
204.03270943838572
--------------GENERATION 96-----------
204.03270943838572
--------------GENERATION 97-----------
204.03270943838572
--------------GENERATION 98-----------
204.03270943838572
--------------GENERATION 99-----------
204.03270943838572
Fitness :  204.03270943838572
2197.1945190429688
```

[9]:

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

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

```
Time Taken :  2197.229176044464
```


Genetic Algorithm
Fitness
Iterations
340
320
300
280
260
240
220
200
0
20
40
60
80
100

```
[ ]:
```