

AI VIET NAM – COURSE 2024

M04 - Exercises (Genetic Algorithm and Its Applications)

Ngày 14 tháng 10 năm 2024

Giới thiệu về bài tập :

Bài toán dự đoán Doanh thu bán hàng (sale) dựa vào các kênh Marketing khác nhau là một dạng bài toán kinh điển thường được dùng trong giảng dạy Machine Learning. Bộ dữ liệu có thể download tại [LINK](#)

Bộ dữ liệu bao gồm 200 mẫu, mỗi mẫu có 3 đặc trưng, là số tiền quảng cáo trên TV, Radio và Newspaper. Giá trị sale (doanh thu) được cung cấp tương ứng với chi phí của 3 kênh marketing được minh họa qua hình 1.

TV	Radio	Newspaper	Sales
230.1	37.8	69.2	22.1
44.5	39.3	45.1	10.4
17.2	45.9	69.3	12
151.5	41.3	58.5	16.5
180.8	10.8	58.4	17.9
8.7	48.9	75	7.2
57.5	32.8	23.5	11.8
120.2	19.6	11.6	13.2
8.6	2.1	1	4.8
199.8	2.6	21.2	15.6

Hình 1: Một vài sample data từ dữ liệu quảng cáo advertising.csv

Để đơn giản vấn đề, chúng ta giả sử chi phí bỏ ra cho các kênh Marketing tuyến tính với doanh thu (mặc dù thực tế còn chịu ảnh hưởng bởi một số yếu tố khác nữa), vì vậy bộ dữ liệu này được mô tả lại theo công thức:

$$sale = \theta_1 * TV + \theta_2 * Radio + \theta_3 * Newspaper + \theta_4 \quad (1)$$

Chúng ta sẽ dùng giải thuật di truyền (GA) để tìm ra giá trị các tham số $\theta_1, \theta_2, \theta_3, \theta_4$. Từ yêu cầu bài toán, chúng ta đã xác định được một số thông tin cho GA như chiều dài mỗi chromosome.

Question 1: Hãy cho biết chiều dài của chromosome trong trường hợp này là bao nhiêu:

- (A) 4
- (B) 5
- (C) 6
- (D) 7

Để thuận tiện cho việc cài đặt, phương trình (1) có thể viết lại như sau:

$$sale = \theta_1 * TV + \theta_2 * Radio + \theta_3 * Newspaper + \theta_4 * 1.0 \quad (2)$$

Triển khai thuật toán

1. Khai báo các tham số cần thiết:

- **n**: size of individual
- **m**: size of population
- **n_generations**: number of generations
- **losses**: lưu giá trị loss để vẽ biểu đồ

2. Khởi tạo các hàm tính toán cần thiết: các bạn có thể đặt tên khác, nhưng phải rõ nghĩa, và tuân theo quy tắc đặt tên hàm

- **compute_loss(individual)**: hàm tính loss theo individual (trọng số)
- **compute_fitness(individual)**:
- **create_individual()**: tạo một cá thể (tham số theta)
- **crossover(individual1, individual2, crossover_rate = 0.9)**: hàm crossover giữa 2 individual
- **mutate(individual, mutation_rate = 0.05)**: hàm mutation
- **selection(sorted_old_population)**: hàm chọn lọc
- **create_new_population(old_population, elitism=2, gen=1)**: hàm tạo population mới

3. Thực hiện huấn luyện:

- (a) Initial Population: Khởi tạo quần thể (population) một cách ngẫu nhiên
- (b) Fitness function: Hàm đánh giá độ tốt của một cá thể. Giá trị càng lớn thì cá thể càng tốt.
- (c) Selection: Bước chọn lọc những cá thể tốt trong quần thể
- (d) Cross-over: Bước lai tạo (trao đổi gen) giữa các cá thể.
- (e) Mutation: Bước đột biến cho một cá thể (thay đổi gen một cách ngẫu nhiên)

Giải thuật GAs thực hiện bước (a) một lần ban đầu, và các bước từ (b) đến (e) được thực hiện cho mỗi generation.

Để hiện thực giải thuật GA trên, các bạn cần hoàn thành các functions sau đây:

Bài tập 1 (kỹ thuật đọc và xử lý dữ liệu từ file .csv): Cho trước file dữ liệu advertising.csv, hãy hoàn thành function **load_data_from_file()** trả về dữ liệu đã được tổ chức (features_X cho input và sales_Y cho output).

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 import random
4 random.seed(0) # please do not remove this line
5 %matplotlib inline
6
7 def load_data_from_file(fileName = "advertising.csv"):
8     data = np.genfromtxt(fileName, dtype=None, delimiter=',', skip_header=1)
9     features_X = data[:, :3]
10    sales_Y = data[:, 3]
11

```

```

12
13 # ***** your code here *****
14
15
16 return features_X, sales_Y


```

Question 2: Hãy cho biết kết quả của đoạn chương trình sau:

```

features_X, _ = load_data_from_file()
print(features_X[:5,:])

```

a)  `[[1. 230.1 37.8 69.2]`
`[1. 44.5 39.3 45.1]`
`[1. 17.2 45.9 69.3]`
`[1. 151.5 41.3 58.5]`
`[1. 180.8 10.8 58.4]]`

b) `[[0. 230.1 37.8 69.2]`
`[0. 44.5 39.3 45.1]`
`[0. 17.2 45.9 69.3]`
`[0. 151.5 41.3 58.5]`
`[0. 180.8 10.8 58.4]]`

c) `[[-1. 230.1 37.8 69.2]`
`[-1. 44.5 39.3 45.1]`
`[-1. 17.2 45.9 69.3]`
`[-1. 151.5 41.3 58.5]`
`[-1. 180.8 10.8 58.4]]`


d) `[[2. 230.1 37.8 69.2]`
`[2. 44.5 39.3 45.1]`
`[2. 17.2 45.9 69.3]`
`[2. 151.5 41.3 58.5]`
`[2. 180.8 10.8 58.4]]`

Question 3: Hãy cho biết kết quả của đoạn chương trình sau:

```

_, sales_Y = load_data_from_file()
print(sales_Y.shape)

```

a) (200,1)
 b) (200,)
c) (1,200)
d) (0,200)

Bài tập 2 (kỹ thuật khởi tạo giá trị cho từng gene trong chromosome (individual), tạo ra các giá trị ngẫu nhiên cho một cá thể (tham số θ của mô hình)): hãy hoàn thiện function **create_individual(n,bound)** để khởi tạo ngẫu nhiên giá trị của n genes (tham số θ_i) trong chromosome với giá trị từng gene được khởi tạo **[-bound/2, bound/2]**

```

def create_individual(n=4, bound=10):
    individual = []

    # ***** Your code here *****

    return individual

individual = create_individual()
print(individual)

```

```
#sample result: [[4.097462559682401, 4.827854760376531, 3.1021723599658957,
4.021659504395827]
]
```

Bài tập 3 (kỹ thuật xây dựng fitness function để đánh giá fitness score cho từng chromosome): hãy hoàn thiện function **compute_fitness(individual)** để tính fitness values của một chromosome bằng cách nghịch đảo giá trị của hàm loss theo công thức $fitness = 1 / (loss + 1)$. Hàm **compute_fitness(individual)** đánh giá độ tốt của một cá thể. Giá trị càng lớn thì cá thể càng tốt.

```
features_X, sales_Y = load_data_from_file()

def compute_loss(individual):
    theta = np.array(individual)
    y_hat = features_X.dot(theta)
    loss = np.multiply((y_hat-sales_Y), (y_hat-sales_Y)).mean()
    return loss


def compute_fitness(individual):
    loss = compute_loss(individual)
    fitness_value = 0

    # ***** your code here *****

    return fitness_value
```

Question 4: Hãy cho biết kết quả của đoạn chương trình sau:

```
features_X, sales_Y = load_data_from_file()
individual = [4.09, 4.82, 3.10, 4.02]
fitness_score = compute_fitness(individual)
print(fitness_score)
```

- (A) 2.018e-06
- (B) 3.018e-06
-  (C) 1.018e-06
- (D) 4.018e-06

Bài tập 4 (kỹ thuật thực hiện crossover giữa 2 chromosomes): hãy hoàn thiện function **crossover(individual1, individual2, crossover_rate = 0.9)** để thực hiện crossover giữa 2 individual (chromosome). Bước này, thực hiện việc lai tạo (trao đổi gen) giữa 2 individual với tỉ lệ *crossover_rate*

```
def crossover(individual1, individual2, crossover_rate = 0.9):
    individual1_new = individual1.copy()
    individual2_new = individual2.copy()

    #***** Your code here *****

    return individual1_new, individual2_new
```

Question 5: Hãy cho biết kết quả của đoạn chương trình sau:

```
#question 5
individual1 = [4.09, 4.82, 3.10, 4.02]
individual2 = [3.44, 2.57, -0.79, -2.41]

individual1, individual2 = crossover(individual1, individual2, 2.0)
print("individual1: ", individual1)
print("individual2: ", individual2)

(A)
individual1 = [4.09, 4.82, 3.10, 4.02]
individual2 = [3.44, 2.57, -0.79, -2.41]

(B)
individual1: [7.44, 2.57, -0.79, -2.41]
individual2: [4.09, 4.82, 3.1, 4.02]

(C)
individual1: [5.44, 2.57, -0.79, -2.41]
individual2: [4.09, 4.82, 3.1, 4.02]

(D)
individual1: [3.44, 2.57, -0.79, -2.41]
individual2: [4.09, 4.82, 3.1, 4.02]
```

Bài tập 5 (kỹ thuật mutation với chromosome): hãy hoàn thiện function **mutate(individual, mutation_rate = 0.05)** để thực hiện việc đột biến cho một cá thể với tỉ lệ đột biến là `mutation_rate`.

```
def mutate(individual, mutation_rate = 0.05):
    individual_m = individual.copy()

    # ***** Your code here *****

    return individual_m
```

Question 6: Hãy cho biết kết quả của đoạn chương trình sau:

```
before_individual = [4.09, 4.82, 3.10, 4.02]
after_individual = mutate(individual, mutation_rate = 2.0)
print(before_individual == after_individual)
```

(A) False
(B) True

Bài tập 6 (khởi tạo population đầu tiên): hãy hoàn thiện function **initializePopulation(m)** để thực hiện việc khởi tạo m cá thể (individual, chromosome) cho population.

```
def initializePopulation(m):
    population = [create_individual() for _ in range(m)]
    return population
```

Bài tập 7 (Kỹ thuật selection để thực hiện việc chọn lọc tự nhiên, đào thải những cá thể yếu trong population): hãy hoàn thiện function **selection()** để thực hiện việc chọn lọc tự nhiên cá thể tốt vào thế hệ kế tiếp.

```
def selection(sorted_old_population, m = 100):
    index1 = random.randint(0, m-1)
    while True:
        index2 = random.randint(0, m-1)
        if (index2 != index1):
            break

    individual_s = sorted_old_population[index1]
    if index2 > index1:
        individual_s = sorted_old_population[index2]

    return individual_s
```

Bài tập 8 (Kỹ thuật tạo ra quần thể (population) mới: Hãy hoàn thiện function **create_new_population()** để tạo ra quần thể mới dựa trên các bước selection, crossover và mutation. Lưu ý rằng chúng ta sẽ sử dụng giải thuật Elitist algorithms để đảm bảo rằng **elitism** cá thể tốt nhất không bị loại bỏ, bằng cách chuyển chúng trực tiếp sang thế hệ tiếp theo.

```
def create_new_population(old_population, elitism=2, gen=1):
    m = len(old_population)
    sorted_population = sorted(old_population, key=compute_fitness)

    if gen%1 == 0:
        print("Best loss:", compute_loss(sorted_population[m-1]), "with chromosome: ",
              sorted_population[m-1])

    new_population = []
    while len(new_population) < m-elitism:
        # selection
        # ***** your code here *****

        # crossover
        # ***** your code here *****

        # mutation
        # ***** your code here *****

        # copy elitism chromosomes that have best fitness score to the next generation
    for ind in sorted_population[m-elitism:]:
        # ***** your code here *****

    return new_population, compute_loss(sorted_population[m-1])
```

Question 7: Hãy cho biết kết quả của đoạn chương trình sau:

```
individual1 = [4.09, 4.82, 3.10, 4.02]
individual2 = [3.44, 2.57, -0.79, -2.41]
old_population = [individual1, individual2]
new_population, _ = create_new_population(old_population, elitism=2, gen=1)
```

- (A) Best loss: 123415.051528805 with chromosome: [3.44, 2.57, -0.79, -2.41]
 (B) Best loss: 123415.051528805 with chromosome: [4.09, 4.82, 3.10, 4.02]

Bài tập 9 (All-in-one) Hãy hoàn thiện function **run_GA()** để tìm tham số $\theta_1, \theta_2, \theta_3, \theta_4$ tối ưu sử dụng giải thuật di truyền.

```
def run_GA():
    n_generations = 100
    m = 600
    features_X, sales_Y = load_data_from_file()
    population = initializePopulation(m)
    losses_list = []
    for i in range(n_generations):

        # ***** your code here *****

    return losses_list
```

Bài tập 10 (Trực quan hoá loss values) Hoàn thiện function **visualize_loss()** để thể hiện giá trị loss tốt nhất của cá thể (chromosome) ứng với từng thế hệ (generations) (hình 2)

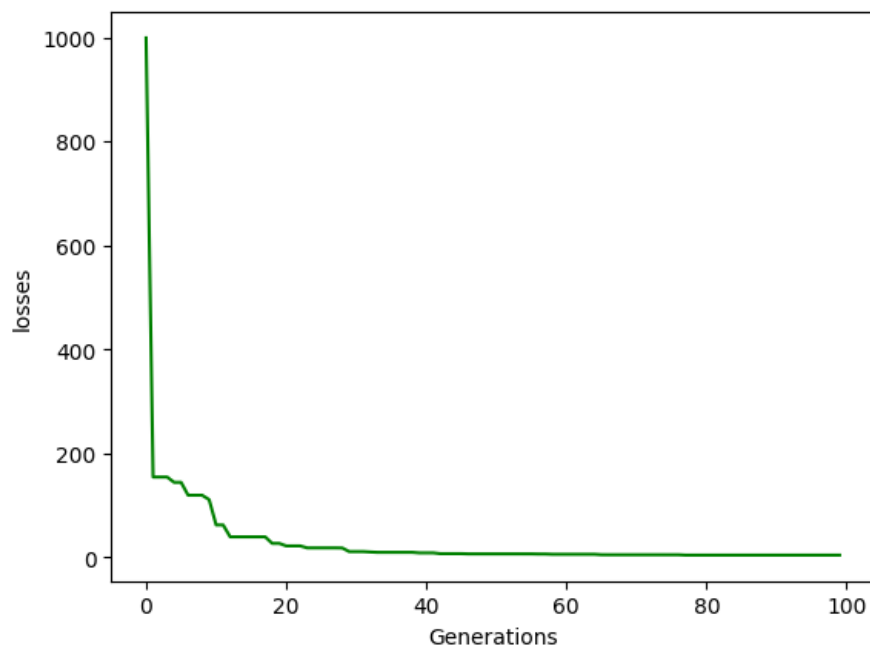
```
import matplotlib.pyplot as plt

import matplotlib.pyplot as plt

def visualize_loss(losses_list):

    # ***** your code here *****

losses_list = run_GA()
visualize_loss(losses_list)
```



Hình 2: Giá trị loss tốt nhất ứng với từng generation

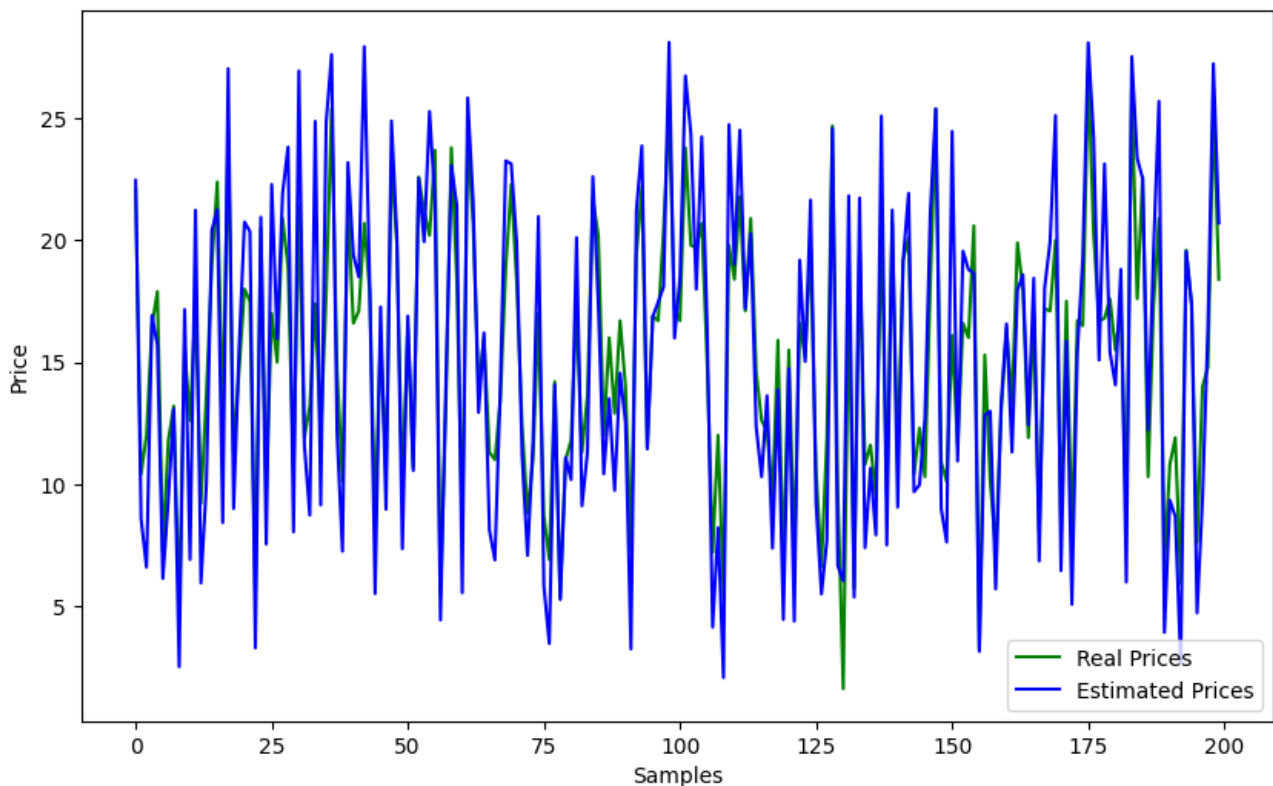
Bài tập 11 (Trực quan hoá kết quả dự đoán sales và giá trị thực tế) Hoàn thiện function **visualize_predict_gt** để hiển thị kết quả dự đoán sales với giá trị thực tế từ ground truth (hình 3)

```
def visualize_predict_gt():
    # visualization of ground truth and predict value
    sorted_population = sorted(population, key=compute_fitness)
    print(sorted_population[-1])
    theta = np.array(sorted_population[-1])

    estimated_prices = []
    for feature in features_X:
        # ***** your code here *****

    fig, ax = plt.subplots(figsize=(10, 6))
    plt.xlabel('Samples')
    plt.ylabel('Price')
    plt.plot(sales_Y, c='green', label='Real Prices')
    plt.plot(estimated_prices, c='blue', label='Estimated Prices')
    plt.legend()
    plt.show()

visualize_predict_gt()
```



Hình 3: kết quả dự đoán sales và giá trị thực tế.