Assignment-2: Genetic Algorithm and Data Science

Template version 1.4 – March 12, 2020

DV2618: Applied Artificial Intelligence

October 14, 2023

Title	Travelling Salesman problem
Supervisor(s)	Huseyin Kusetogullari

Student 1	Name	Venkata Surya Akash Mernedi
	E-Mail	veme22@student.bth.se
	Program	Masters Qualification Plan in Computer Science

Submission/change history (latest entry on top)	
Date	**Changes made**
2023-10-16	Initial submission.

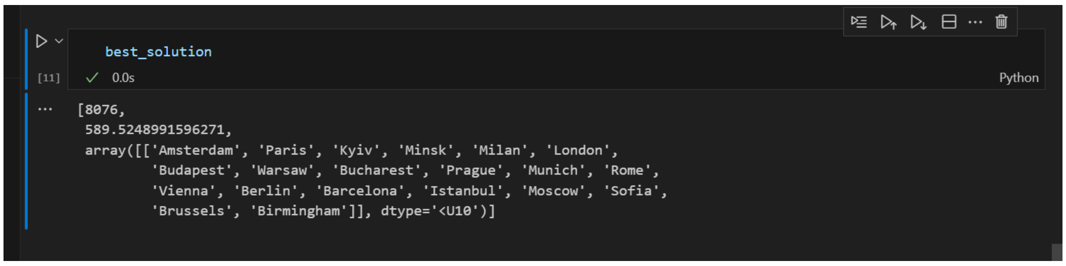
1 Task-1: Analysing the Genetic algorithm

The goal of task 1 is to understand the genetic algorithm and solve the traveling salesman problem. In the traveling salesman problem, our goal is to find the shortest path between different pairs of cities, visiting each city exactly once and returning to the origin city.

While experimenting with the given genetic algorithm code we can observe the best solution route, distance to the solution, and the number of iterations it took to find the solution. After testing various combinations of population size and mutation rate using the given genetic algorithm code, we obtained the following results:

Cities	Population	Mutation rate	Iterations	Shortest distance
20	100	0.3	6330	529.69
20	10	0.9	6100	575.18
20	20	0.6	8081	399.49
20	50	0.3	9810	555.98
20	100	0.1	8926	628.41

In case 1 with population 100 using the genetic algorithm, it achieved a shortest distance of 529.69, while a much lower mutation rate achieved a shortest distance of 628.41. In case3 with a population of 20 using the genetic algorithm with a mutation rate of 0.6 it traveled the shortest distance of 399.49 which suggests that a moderate mutation rate is effective in balancing exploration and exploitation, allowing the algorithm to escape local optima whereas the mutation rate of 0.9 and 0.3 it achieved the shortest distance of 575.18 and 555.98. The number of iterations is higher in the case of population 50 with population 50. However, it resulted in an optimal shortest distance of 555.98, which takes a long time to predict. In this case, 8081 predictions were made with a best travel distance of 399.49. **Note:** The running output values of the corresponding genetic algorithms are stored in the Task1 folder.


```
best_solution
```

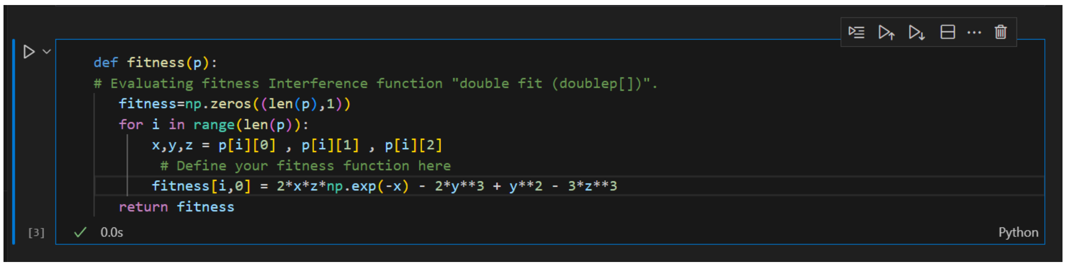
```
[8076,
 589.5248991596271,
 array([['Amsterdam', 'Paris', 'Kyiv', 'Minsk', 'Milan', 'London',
         'Budapest', 'Warsaw', 'Bucharest', 'Prague', 'Munich', 'Rome',
         'Vienna', 'Berlin', 'Barcelona', 'Istanbul', 'Moscow', 'Sofia',
         'Brussels', 'Birmingham']], dtype='<U10')]
```

Figure 1: Best solution route to the cities

Finally, after tuning the parameters of the genetic algorithms for optimization problems, we can conclude that case 3 is the best way to find the shortest route among the scenarios tested with moderate population size and mutation rate.

2 Task-2: Maximization problem

The implementation of the fitness function is as follows.



```python
def fitness(p):
# Evaluating fitness Interference function "double fit (doublep[])".
    fitness=np.zeros((len(p),1))
    for i in range(len(p)):
        x,y,z = p[i][0] , p[i][1] , p[i][2]
         # Define your fitness function here
        fitness[i,0] = 2*x*z*np.exp(-x) - 2*y**3 + y**2 - 3*z**3
    return fitness
```

Figure 2: fitness function implementation

3 Task-3: Maze problem using genetic algorithms

Below is the generalized working of the genetic algorithm to solve the problem of the mouse finding a route to reach the destination to eat the food with the shortest and acceptable path.

Algorithm:

1. Create an initial population and assign them a different gene(possible set-off actions) randomly.
2. Define a fitness function by counting the number of steps taken from the initial position.
3. Choose the fittest members of the population.
4. Take the parent data, perform cross-over, and create a new generation of possible paths.
   Note: Here, we should also mutate the genes, otherwise it leads to the problem of local minima. So mutation is done which gives the opportunity to try new things.
5. Perform the mutation on the data.
   Note: If the mutation is too high then it doesn't listen to parent data and might take a very long time. If the mutation is too small then it will listen to parent data and learn faster but it may take a long time to change the way if it is near local minima. The good mutation rate is generally small.
6. Repeat steps 3-5 until we get the solution to the given problem (rat finds food).

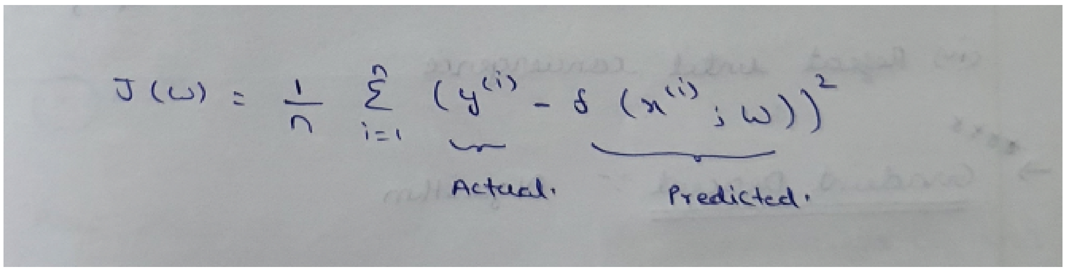
4 Task-4: Regression for the data

In this section, we will perform machine learning and neural net algorithms on the given data, calculate the mean square error, and plot the data. The code follows the below steps,

1. Download the required packages.
2. Import the required packages.
3. Generate the data as per the description module.
4. Using `train_test_split` module from scikit learn split the data into train and test set of ratio 80 and 20.
5. **Linear Regression:** In linear regression we will find the relationship between a dependent variable and one or more independent variables by fitting a straight line to the data. We have implemented this by using the `linear_model` from the

`scikit-learn` package. After we calculated the coefficient, intercept, calculated the MSE, and finally plotted the data using Seaborn and Matplotlib.

6. **Polynomial Regression:** In polynomial regression we will find the relationship between a dependent variable and one or more independent variables as a polynomial. It can capture non-linear relationships between variables by fitting a non-linear regression line. It is not a straight line compared to linear regression. We have implemented this by using the `polynomial_features` from the `scikit-learn` package. First, we transform the model with polynomial features and then fit it into the linear regression model then we will predict the model, calculate the MSE, and finally plot the model using Seaborn and Matplotlib.
7. **Neural Net:** The below is the explanation of each process done in the neural network.
    - **Perceptron:** A perception is a neural network unit(An artificial neuron) that does some computations to detect features or business intelligence in the input data.
    - **Activation function:** The activation function is a non-linear transformation that we do over the input before sending it to the next layer of neurons. Here we have used `relu` activation.
    - **Optimizer:** In code `adam` optimizer is used. The generalized algorithm of optimizer in neural net is as follows:
        - Randomly pick an initial weight (w0, w1).
        - Compute gradient.
        - Take a small step in the opposite direction of the gradient.
        - Repeat until convergence.
    - **Mean Squared error:**



$$J(\omega) = \frac{1}{n} \sum_{i=1}^{n} (\underbrace{y^{(i)}}_{\text{Actual.}} - \underbrace{f(x^{(i)}; \omega)}_{\text{Predicted.}})^2$$

Figure 3: Mean square error for the neural net

- **Epoches:** The model is trained with 100 epochs (iterated over the entire training set during the training process.)
- Finally, the mean square error is calculated.

Algorithms	Mean Square Error
Linear Regression	0.006989
Polynomial Regression	0.000333
Neural Network	0.000450

The mean square error of polynomial regression and neural networks is similar. However, polynomial regression has a slightly lower MSE.