National University of Computer and Emerging Sciences, Lahore Campus

Course:	**AI Lab**	**Course Code:**	**CL1002**
Program:	**BS (CS, DS)**	**Semester:**	**Spring 2023**
Duration:	**2 Hours 30 Minutes**	**Total Marks:**	**40**
Paper Date:	**6th June, 2023**	**Weight:**	**40%**
Section:	**All Sections**	**Page(s):**	**3**
Exam:	**Lab Final**	**Reg. No.**	

Read below Instructions Carefully:

- Understanding the question statement is also part of the exam, so do not ask for any clarification. In case of any ambiguity, make suitable assumptions.
- You have to complete exam in specified time. No extra time will be given for submission.
- Submit a single. ipynb **file** for all questions.
- Submit folder on **cactus** by following path: \\cactus1\Xeon\Spring 2023\Junaid Hussain\AI Lab Final Slot 1
- Your code should be **intended** and **commented** properly. Use **meaningful variable names**.
- It is your responsibility to save your code from being copied. All matching codes will be considered cheating cases. **PLAGIARISM** will result in forwarding of **case to Disciplinary Committee**. You know what happened last time for cheating!

Question No. 1: "Implementing an Online, Median-based K-Means Algorithm from Scratch in Python."

The traditional K-Means algorithm uses a batch processing approach, where all points are assigned to clusters, then all centroids are recalculated using the mean of the assigned points. Your task is to implement a variant of K-Means in Python that uses a different approach: update the centroid after each new point is assigned (an "online" approach), and recalculate the centroid using the median of the assigned points instead of the mean. **Data set driverdata.csv**

Requirements:

1. Implement the traditional K-Means algorithm from scratch in Python. (5 Marks)
2. Implement the variant of K-Means described above from scratch in Python. (5 Marks)
3. Run both algorithms on the given dataset, and visualize the clustering results from both (5 + 5Marks)

Note: The 'median' version of K-means is sometimes referred to as K-Medians. In higher dimensions, calculating the median is not as straightforward as in one dimension, so you'll need to choose an appropriate method for multidimensional median calculation.

Please remember not to use pre-built K-Means or K-Medians libraries; the aim is to understand the underlying algorithms. However, you can use standard Python libraries for handling data and visualizations (like NumPy, pandas, matplotlib, etc.).

Q No. 2. Linear regression (10 Marks)

You are provided with a dataset that contains hourly measurements of weather-related variables, including temperature, apparent temperature, humidity, wind speed, wind bearing, visibility, cloud cover, and pressure. Your task is to explore the relationships between humidity and temperature, as well as between humidity and apparent temperature. Additionally, you need to develop a model to predict the apparent temperature based on the given humidity.

1. Read the weatherHistory.csv dataset from a file, where each line represents an hourly measurement with the variables mentioned above.
2. Conduct a statistical analysis to determine the correlation between humidity and temperature, as well as between humidity and apparent temperature. Calculate the correlation coefficient and interpret its significance.
3. Visualize the relationships between humidity and temperature, and between humidity and apparent temperature, using scatter plots.
4. Split the dataset into a training set and a testing set, with a specified ratio (e.g., 80% for training and 20% for testing).
5. Implement a linear regression model using the training data to predict the apparent temperature based on the given humidity.
6. Evaluate the performance of the model by calculating the mean squared error (MSE) on the testing data.
7. Use the trained model to predict the apparent temperature for a new observation with a given humidity.
8. Visualize the predicted apparent temperature against the actual apparent temperature from the testing data using a scatter plot.

Note: Built in functions can be used.

Q No. 3: Creating and Tuning a Genetic Algorithm for Numerical Function Optimization in Python from Scratch (10 Marks)

Genetic algorithms are highly effective for optimization problems due to their bio-inspired operations: selection, mutation, and crossover. Your task is to implement a genetic algorithm to optimize a numerical function. Specifically, let's consider a well-known benchmark function in the field of optimization, the Sphere Function, which is defined as f(x) = Σ xi^2 for i = 1 to n, where x is a vector in n-dimensional space.

Requirements:

1. Implement a genetic algorithm from scratch in Python. The basic steps are:
2. Initialize a population with random potential solutions.
3. Define a fitness function to evaluate the quality of each solution (in this case, the Sphere function).
4. Run the genetic algorithm for a set number of generations or until an optimal solution is found, doing the following in each generation:
5. Select the best solutions based on their fitness scores.
6. Perform crossover and mutation to create a new population.
7. Evaluate the fitness of the new population.
8. Test your genetic algorithm to find the global minimum of the Sphere Function in the search space [-5.12, 5.12]^n for different dimensions n = 2, 5, and 10.

Tune the parameters of your genetic algorithm (like population size, mutation rate, and crossover method) to find

Remember to implement the genetic algorithm from scratch, without using specialized libraries. However, you can use standard Python libraries for handling data and calculations, like NumPy.

Q No. 4: Backpropagation and Feed Froward (20 marks) ONLY BDS

Implement a feedforward neural network using Python. Your network should have one input layer with 3 nodes, one hidden layer with 4 nodes, and one output layer with 2 nodes. The activation function for all layers is the sigmoid function. Write a function called feedforward that takes an input vector and returns the output vector of the neural network.

Your function should follow the steps of a feedforward process:

1. Compute the weighted sum of inputs for each node in the hidden layer.
2. Apply the sigmoid activation function to the hidden layer outputs.
3. Compute the weighted sum of inputs for each node in the output layer.
4. Apply the sigmoid activation function to the output layer outputs.
5. Return the output vector.
6. Implement both feedforward and backpropagation functions.
7. Run the input vector = [0.5, 0.1, 0.9] and display for both feed forward and back propagation and then again feed forward in the main. Learning Rate = 0.04, predicted output = 2

You can assume that the weights and biases for the network are pre-defined as the following lists:

```
input_weights = [[0.1, 0.2, 0.3, 0.4],
                 [0.5, 0.6, 0.7, 0.8],
                 [0.9, 1.0, 1.1, 1.2]]
hidden_weights = [[1.3, 1.4],
                  [1.5, 1.6],
                  [1.7, 1.8],
                  [1.9, 2.0]]
output_weights = [[2.1, 2.2],
                  [2.3, 2.4],
                  [2.5, 2.6],
                  [2.7, 2.8]]
```

The biases for the hidden and output layers are defined as follows:

```
hidden_biases = [0.9, 1.0, 1.1, 1.2]
output_biases = [1.3, 1.4]
```

You can use the sigmoid function sigmoid(x) defined below for the activation function:

```
import math
def sigmoid(x):
  return 1 / (1 + math.exp(-x))

          #Sample output: [0.7874101769784412, 0.8946330293610033]
```