Home Assignment – 4

Aarav Aryaman, 220012

Problem:

Samples of solid rocket propellants will be used if their shear strengths are adequate. Shear strength is found to be a function of propellant age and storage temperature. The propellants are accepted or rejected, based on shear strength measurements, as shown below.

Test	Propellant age (Weeks)	Storage temperature (°C)	Pass/fail for application
1	15.5	40	fail
2	23.75	23.25	fail
3	8	17	pass
4	17	21	fail
5	5.5	10	pass
6	19	12	pass
7	24	20	fail
8	2.5	12	pass
9	7.5	15	pass
10	11	26	fail

Write a computer program (preferably in python), from scratch, to compute the contour of passing (or failing) probabilities using logistic regression. The computer program must NOT use scikitlearn/scipy/statistics or similar packages/libraries. You can only use packages for vector/matrix/array operations and plotting (numpy, matplotlib etc.).

1. Define a cost function and deduce the gradient of the same.
2. Use gradient descent with an appropriate line search technique to minimize the above cost function.
3. Write the pseudocode of the above procedure.
4. Plot the scatter of data, and probability (of passing or failing) contour in one figure.

Pseudocode:

Input:

- x_in_age: Array of propellant ages (in weeks)
- x_in_temp: Array of storage temperatures (in °C).
- y_in: Array of pass/fail labels (0 for fail, 1 for pass)

Output:

- minimum_value: Minimum value of the objective function
- w: Optimal parameters (intercept, coefficient for age, coefficient for temperature)

Control Flow:

1. **Initialise Data**:

- Combine x_in_age and x_in_temp into a matrix X with an additional column of 1's for the intercept term:

  X = [[1, x1_age, x1_temp],

  [1, x2_age, x2_temp],

  ...,

  [1, xn_age, xn_temp]]

2. **Define Objective Function**:
   - For parameters w = [w0, w1, w2], compute:
     - Compute z for each data point:

       z_i = w0 + w1 * x_i_age + w2 * x_i_temp

       z1_i = log(1 + exp(-z_i))

       z2_i = log(1 + exp(z_i))
     - Compute the objective function (negative log-likelihood):

       objective_function(w) = sum(y_i * z1_i + (1 - y_i) * z2_i)

3. **Gradient of Objective Function**:
   - Compute the gradient of the objective function with respect to w:
     - Compute the predicted probabilities hi using the logistic function:

       z_i = w0 + w1 * x_i_age + w2 * x_i_temp

       hi_i = 1 / (1 + exp(-z_i))

       yh_i = hi_i - y_i
     - Compute the gradient for each parameter:

       Gradient_w0 = sum(yh_i)

       Gradient_w1 = sum((yh_i) * x_i_age)

       Gradient_w2 = sum((yh_i) * x_i_temp)

       Gradient = [Gradient_w0, Gradient_w1, Gradient_w2]

4. **Line Search**:
   - Set initial step size (stepsize) and parameters for line search (e.g., beta, tau).
   - For a predefined number of trials or until convergence:
     - Compute the objective function value with current w and stepsize:

       fx1 = objective_function(w)
     - Compute the objective function value with updated w (i.e., w - stepsize * Gradient):

w = w - stepsize * Gradient

fx2 = objective_function(w)

- If the reduction in the objective function is sufficient:

  fx2 - fx1 <= -beta * stepsize * sum(Gradient^2)

  return stepsize

- Otherwise:

  stepsize = tau * stepsize

5. **Gradient Descent Method**:
   - Set up the parameters: maximum number of iterations (maxit = 1000000), convergence threshold (epsilon = 1.e-3) and initialise parameter w with initial guesses (w = [-2, 1, 1]).
   - Repeat until convergence or reaching maximum iterations:
     - Compute the gradient of the objective function:

       Gradient = [Gradient_w0, Gradient_w1, Gradient_w2]

     - Check the norm of the gradient. If it is less than epsilon, stop the optimization:

       if norm(Gradient) < epsilon, then break

     - Perform line search to find the optimal step size:

       stepsize = line_search(objective_function, Gradient, w)

     - Update the parameter vector w and print iterations:

       w = w - stepsize * Gradient

       print(i, norm(gradient))

   - Evaluate the objective function with the final parameter values w to find minimum:

     minimum_value = objective_function(w)

6. **Output**:
   - Minimum value: 0.005260327749866586
   - Minimum location: [38.74045315 -0.68218905 -1.58718858]
   - Iteration: 37272

7. **Plot**:
   - Define range and meshgrid of values for propellant age and storage temperature:

     x1_plot = linspace(min(x_in_age) - 5, max(x_in_age) + 5, 100)

     x2_plot = linspace(min(x_in_temp) - 5, max(x_in_temp) + 5, 100)

     x1_grid, x2_grid = meshgrid(x1_plot, x2_plot)

- Compute Probability Contour:

  z = -w[0] - w[1] * x1_grid - w[2] * x2_grid

  p = 1 / (1 + exp(z))

- Compute Decision Boundary:

  decision_boundary = -w[0] / w[2] - (w[1] / w[2]) * x1_plot

- Plot both on the same graph and scatter the original data points with their labels.