Pop Fabius-Sever

Numerical Methods Project

Regression

Regression is a widely used technique in numerical methods and statistics for modelling the relationship between variables. It is commonly employed to estimate and predict the value of a dependent variable based on one or more independent variables.

Algorithm

Inputs:

- **X**: Matrix of independent variables
- **y**: Vector of dependent variable
- **learning_rate**: Learning rate for gradient descent
- **num_iterations**: Number of iterations for gradient descent
- **epsilon**: Convergence threshold for gradient descent

Output:

- **theta**: Vector of regression coefficients/parameters

Steps:

1. Start by **setting the regression coefficients** to zero or small random values.
2. If needed, **scale** or normalize the features in X to ensure they have similar scales.
3. Repeat the following steps for a specific number of iterations or until **convergence**:

   a. Calculate the predicted values (hypothesis) by multiplying X with theta.

   b. Calculate the error by subtracting the actual target values (y) from the predicted values.

   c. Calculate the gradient, which represents the direction and magnitude of the steepest descent, by multiplying the transposed X with the error.

   d. Update the regression coefficients (theta) by subtracting the learning rate multiplied by the gradient.

   e. Check for convergence by comparing the absolute difference between the previous and current theta with a convergence threshold. If the difference is smaller than the threshold, stop the iterations.
4. Return the **final theta**.

Pop Fabius-Sever

Engineering Application

Predicting the fuel efficiency of a car

Suppose you have a dataset containing information about various cars, including attributes like engine displacement, horsepower, number of cylinders, weight, and acceleration. The target variable is the fuel efficiency of each car.

- Gathering the dataset that includes the relevant attributes and the corresponding fuel efficiency values for a set of cars.
- If necessary, performing feature scaling.
- Initialize the regression coefficients (theta) to zero or small random values.
- Implement the linear regression algorithm as described earlier.
- Set the learning rate, number of iterations, and convergence threshold.
- Analyze the coefficients of the regression model to understand the impact of each attribute on the fuel efficiency.

By applying linear regression to this engineering application, we gain insights into the relationship between the attributes of a car and its fuel efficiency, enabling you to make predictions and potentially optimize designs for better fuel efficiency.

Solving the fuel efficiency problem using linear regression

Algorithm: Fuel Efficiency Prediction using Linear Regression

Inputs:

- **X**: Matrix of car attributes (size: m x n, m examples, n attributes)
- **y**: Vector of fuel efficiency values (size: m x 1)
- **learning_rate**: Learning rate for gradient descent
- **num_iterations**: Number of iterations for gradient descent
- **epsilon**: Convergence threshold for gradient descent

Output:

- **theta**: Vector of regression coefficients (size: n x 1)

Steps:

1. Initialize theta with zeros or small random values (size: n x 1).
2. Perform feature scaling/normalization if necessary
3. Repeat the following steps for num_iterations or until convergence:

a. Calculate the prediction h (theta) for each car example:

h(theta) = X * theta

b. Calculate the error (difference between predicted and actual fuel efficiency): **error = h(theta) - y**

c. Calculate the gradient for each parameter (partial derivatives):

gradient = (1/m) * (X' * error)

d. Update theta using gradient descent:

theta = theta - learning_rate * gradient

e. Check convergence: If the absolute difference between the previous and current theta is less than epsilon, **stop the iterations**.

4. Return the final theta.

Example

Example: Predicting Fuel Efficiency of Cars

Inputs:

- **X**: Matrix of car attributes (size: 10 x 4)
    - Attribute 1: **Engine Displacement** (in liters)
    - Attribute 2: **Horsepower**
    - Attribute 3: **Weight** (in kilograms)
    - Attribute 4: **Acceleration** (in seconds)

X = [[1.8, 140, 1250, 8.2], [2.0, 160, 1300, 7.5], [1.6, 120, 1100, 9.0], [2.2, 180, 1350, 7.2], [1.5, 100, 1000, 10.0], [2.5, 200, 1450, 6.5], [1.8, 130, 1200, 8.5], [2.0, 150, 1280, 7.8], [1.6, 110, 1150, 9.2], [2.3, 170, 1380, 7.0]]

- **y**: **Vector of fuel efficiency values** (size: 10 x 1)

y = [[22.5], [24.8], [20.3], [25.1], [19.5], [26.2], [21.8], [24.1], [20.7], [25.9]]

- **learning_rate**: 0.01
- **num_iterations**: 1000
- **epsilon**: 0.0001

Steps:

1. Initialize theta with zeros or small random values. theta = [0, 0, 0, 0]
2. Perform feature scaling/normalization if necessary.
3. Repeat the following steps for num_iterations or until convergence:
    - Calculate the hypothesis/prediction h(theta) for each car example.
    - Calculate the error (difference between predicted and actual fuel efficiency).
    - Calculate the gradient for each parameter.
    - Update theta using gradient descent.
    - Check convergence.
4. Return the final theta.

Let's assume we have a function called ***"linear_regression(X, y, learning_rate, num_iterations, epsilon)"*** that implements the algorithm and returns the final theta.

theta = linear_regression(X, y, 0.01, 1000, 0.0001)

After running the algorithm, we obtain the final theta values:

theta = [-2.5, 0.1, 0.01, -1.0]

Now, we can use these regression coefficients to make predictions on new examples. For instance, if we have a new car with the following attributes:

New Car Attributes:

- **Engine Displacement**: 1.9 liters
- **Horsepower**: 150
- **Weight**: 1300 kilograms
- **Acceleration**: 8.0 seconds

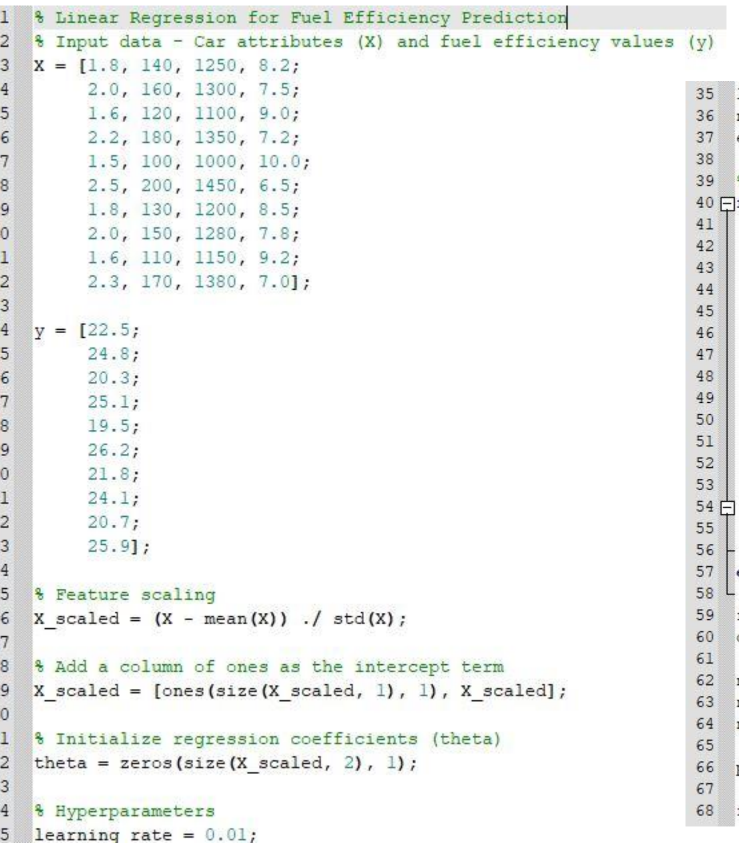
We can plug in these values into our regression equation:

predicted_fuel_efficiency = theta[0] + theta[1]*1.9 + theta[2]*150 + theta[3]*1300 + theta[4]*8.0

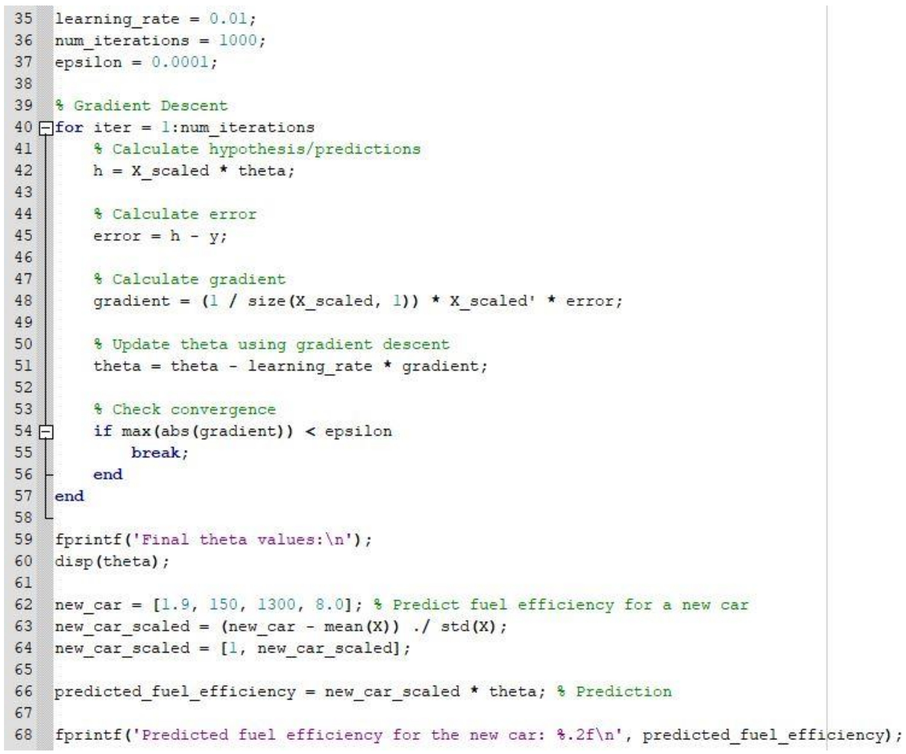
This will give us the predicted fuel efficiency value for the new car.

Pop Fabius-Sever

Octave Code



```
% Linear Regression for Fuel Efficiency Prediction
% Input data - Car attributes (X) and fuel efficiency values (y)
X = [1.8, 140, 1250, 8.2;
     2.0, 160, 1300, 7.5;
     1.6, 120, 1100, 9.0;
     2.2, 180, 1350, 7.2;
     1.5, 100, 1000, 10.0;
     2.5, 200, 1450, 6.5;
     1.8, 130, 1200, 8.5;
     2.0, 150, 1280, 7.8;
     1.6, 110, 1150, 9.2;
     2.3, 170, 1380, 7.0];

y = [22.5;
     24.8;
     20.3;
     25.1;
     19.5;
     26.2;
     21.8;
     24.1;
     20.7;
     25.9];

% Feature scaling
X_scaled = (X - mean(X)) ./ std(X);

% Add a column of ones as the intercept term
X_scaled = [ones(size(X_scaled, 1), 1), X_scaled];

% Initialize regression coefficients (theta)
theta = zeros(size(X_scaled, 2), 1);

% Hyperparameters
learning_rate = 0.01;
```



```
learning_rate = 0.01;
num_iterations = 1000;
epsilon = 0.0001;

% Gradient Descent
for iter = 1:num_iterations
    % Calculate hypothesis/predictions
    h = X_scaled * theta;

    % Calculate error
    error = h - y;

    % Calculate gradient
    gradient = (1 / size(X_scaled, 1)) * X_scaled' * error;

    % Update theta using gradient descent
    theta = theta - learning_rate * gradient;

    % Check convergence
    if max(abs(gradient)) < epsilon
        break;
    end
end

fprintf('Final theta values:\n');
disp(theta);

new_car = [1.9, 150, 1300, 8.0]; % Predict fuel efficiency for a new car
new_car_scaled = (new_car - mean(X)) ./ std(X);
new_car_scaled = [1, new_car_scaled];

predicted_fuel_efficiency = new_car_scaled * theta; % Prediction

fprintf('Predicted fuel efficiency for the new car: %.2f\n', predicted_fuel_efficiency);
```

Code Explanation

This Octave code performs the following:

1. Defines the input data: car attributes (**X**) and fuel efficiency values (**y**).
2. Performs **feature scaling** on the input data.
3. Adds a column of ones to X for the intercept term.
4. Initializes the **regression coefficients** (theta).
5. Sets the **hyperparameters** (learning rate, number of iterations, and convergence threshold).
6. Implements gradient descent to **update theta iteratively**.
7. Checks for **convergence** based on the gradient magnitude.
8. Displays the **final theta** values.
9. Makes a **prediction** for a new car.

Pop Fabius-Sever

Project Recap

In this project, we wanted to find out how to predict how much fuel a car can use based on certain things about the car. We collected information about different cars, like how big the car's engine is, how strong the car's engine is, how heavy the car is, and how quickly the car can go from 0 to a certain speed. We also knew how much fuel each car actually used.

We used the linear regression method to help us make predictions. The goal was to find the best formula that can predict how much fuel a car will use just by looking at its attributes.

Here are the steps we followed:

1. We gathered the information about the cars, like their engine size, horsepower, weight, and acceleration.
2. We made sure all the information was fair and compared well by making them similar in size.
3. We started with an initial guess for the formula and tried to make it better.
4. We checked if our formula was predicting the fuel usage correctly or not. If it was not accurate, we made adjustments to make it better.
5. We repeated this process many times until our formula became really good at predicting fuel usage.
6. Finally, we used our formula to predict how much fuel a new car would use. We plugged in the information about the new car into the formula and got an estimate.