Q1 - Linear Regression

Implementation

The question asked is to perform Linear Regression using Least Squares, Least Median of Squares and Least Trimmed Squares. All the above algorithms are implemented in MATLAB. The `matlab\Q1` folder has all the MATLAB files related to this question.

Least Squares

```
%% Ordinary Least Squares using Gradient Descent
function [beta, cost_history] = ordinaryLeastSquares(X, y, alpha, epochs)
    %% Input Arguments
    %   X: Input features (n x m matrix, where n is the number of data points
    %      and m is the number of features)
    %   y: Target variable (n x 1 vector)
    %   alpha: Learning rate (scalar)
    %   epochs: Number of epochs (scalar)

    %% Output Arguments
    %   beta: Learned or Estimated parameters of the model(m+1 x 1 vector)
    %   cost_history: Cost history for each iteration (epochs x 1 vector)

    %% Initialize parameters
    n = length(y);                   % Number of data points
    m = size(X, 2);                  % Number of features
    beta = zeros(m + 1, 1);         % Initialize parameters
    cost_history = zeros(epochs, 1); % Store cost for each iteration
    X = [ones(n, 1) X];              % Add bias term to X

    %% Gradient Descent
    for epoch = 1:epochs
        % Compute y_hat and cost
        y_hat = X * beta;                % Hypothesis: y_hat = X * theta
        cost = sum((y_hat - y).^2) / (2 * n); % Ordinary Least Squares
        cost_history(epoch) = cost; % Store cost for plotting

        % Compute gradients
        grad = (X' * (y_hat - y)) / n;

        % Update parameters
        beta = beta - alpha * grad;
    end
end
```

Least Median of Squares

```
%% Least Median Squares using Gradient Descent
function [beta, cost_history] = leastMedianSquares(X, y, alpha, epochs)
    %% Input Arguments
    %   X: Input features (n x m matrix, where n is the number of data points
    %      and m is the number of features)
    %   y: Target variable (n x 1 vector)
    %   alpha: Learning rate (scalar)
    %   epochs: Number of epochs (scalar)

    %% Output Arguments
    %   beta: Learned or Estimated parameters of the model(m+1 x 1 vector)
    %   cost_history: Cost history for each iteration (epochs x 1 vector)

    %% Initialize parameters
    n = length(y);                  % Number of data points
    m = size(X, 2);                 % Number of features
    beta = zeros(m + 1, 1);         % Initialize parameters
    cost_history = zeros(epochs, 1); % Store cost for each iteration
    X = [ones(n, 1) X];             % Add bias term to X

    %% Gradient Descent
    for epoch = 1:epochs
        % Compute y_hat and cost
        y_hat = X * beta;             % Hypothesis: y_hat = X * theta
        e = y_hat - y;                % Residual
        squared_e = e.^2;             % Squared Residual
        cost = median(squared_e)/ (2 * n);      % Least Median Squares
        cost_history(epoch) = cost; % Store cost for plotting

        % Compute gradients (Technically Subgradient)
        grad = (2/n) * (X' * (e.* xsign(squared_e, cost)));

        % Update parameters
        beta = beta - alpha * grad;
    end
end
```

Least Trimmed Squares

```
%% Least Trimmed Squares using Gradient Descent
function [beta, cost_history] = leastTrimmedSquares(X, y, alpha, epochs)
    %% Input Arguments
    %   X: Input features (n x m matrix, where n is the number of data points
    %      and m is the number of features)
    %   y: Target variable (n x 1 vector)
    %   alpha: Learning rate (scalar)
    %   epochs: Number of epochs (scalar)

    %% Output Arguments
    %   beta: Learned or Estimated parameters of the model(m+1 x 1 vector)
    %   cost_history: Cost history for each iteration (epochs x 1 vector)

    %% Initialize parameters
    n = length(y);                     % Number of data points
    q = ceil((n/2)+1);                % Number of samples to consider for computing loss
    m = size(X, 2);                    % Number of features
    beta = zeros(m + 1, 1);           % Initialize parameters
    cost_history = zeros(epochs, 1); % Store cost for each iteration
    X = [ones(n, 1) X];                % Add bias term to X
    index_column = (1:n)';             % Indexing Column

    %% Gradient Descent
    for epoch = 1:epochs
        % Compute y_hat and cost
        y_hat = X * beta;              % Hypothesis: y_hat = X * theta
        e = y_hat - y;                 % Residual

        e_indexed = [e.^2, index_column]; % Add index column (Useful
                                while calculating the gradients)
        e_sorted = sortrows(e_indexed, 1); % Ordered squared residuals
        e_sampled = e_sorted(1:q, :); % Consider the first q samples
        cost = sum(e_sampled(:, 1))/ (2 * q); % Least Trimmed Squares
        cost_history(epoch) = cost; % Store cost for plotting

        % Compute gradients
        Xs = X(e_sampled(:, 2),:); % Sampled Input
        es = e(e_sampled(:, 2)); % Sampled Error
        grad = (Xs' * (es)) / q;

        % Update parameters
        beta = beta - alpha * grad;
    end
end
```

A few helper functions are also defined and used in the above functions which are in the `matlab\Q1` folder.

Synthetic Data

As per the question, synthetic data is generated and the models are evaluated on this data.

The following is the result of that analysis,

Parameters	OLS	LMS	LTS
MSE	0.034016	0.01881	0.26578
RB	6.7453	8.1251	3.9865
MAD	18.367	22.303	10.385

From the above, we can observe the following:

- LMS excels in terms of the **lowest MSE**, which implies better accuracy in terms of squared error minimization, but it has the highest RB and MAD, indicating **higher bias** and **less consistency**.
- LTS shows the **least bias** (lowest RB) and **highest consistency** (lowest MAD), making it robust in terms of bias and deviation, but it suffers from a high MSE.
- OLS stands in the middle ground with moderate values for MSE, RB, and MAD, indicating a **balanced performance** but **not excelling in any specific metric** of comparison.

These inferences suggest that while **LMS minimizes squared errors effectively, LTS is better for minimizing bias and maintaining consistency**. The choice between these methods depends on the specific requirements of the problem, whether minimizing errors, reducing bias, or ensuring consistent performance is the priority.

Real Dataset - Medical Insurance

The medical insurances dataset in the `dataset` folder is used in this question. Linear regression is carried out to predict the medical charges for a person based on the regressors.

All the above three methods are used on the data with MSE as the comparison metric. The given data also has categorical variables like gender, religion, smoker or not which are encoded into numerical values. The following is the result of the analysis,

Parameters	OLS	LMS	LTS
MSE	3.9839e+07	5.0851e+07	2.664e+08

Based on the given table, we can observe the following:

- OLS demonstrates the **best performance** with the lowest MSE, suggesting that it is the most accurate method in terms of minimizing squared errors.
- LMS has a higher MSE than OLS, indicating that it is less accurate than OLS but still performs better than LTS.

- LTS has the highest MSE, suggesting that it is the least accurate method among the three in terms of minimizing squared errors.

Given these results, OLS is the preferred method in this task when the goal is to achieve the lowest mean square error. LMS, while not as effective as OLS, still provides better performance than LTS. LTS, with the highest MSE, is the least desirable method in this context for minimizing squared errors.