Module Big data & Small data
Data Collection and Machine Learning
HTE1

Submitted by AD Shinde (2132290)

Part 1 Linear Regression with Regularization

Question 1: Data preparation

The data was imported and normalized using the following code:

```
%% loading the dataset
data = readtable('energy_efficiency_data_heating_load.csv');

%% extracting features and target variable
X = table2array(data(:, 1:end-1));
y = table2array(data(:, end));
n = size(data, 2) - 1; % Number of features
m = length(y); % Number of observations

%% normalization
normalized_X = (X - mean(X)) ./ std(X);
normalized_data = [normalized_X, y];
```

The data was then split into train, validation and test sets with the ratio of 0.6, 0.2 and 0.2 respectively. The validation scheme is to use the values of theta from the training process of the model to calculate the hypothesis function for both the validation and testing process and calculate the MSE (Mean Squared Error) to analyze the prediction performance of the model.

Normalization of the features is necessary to bring the features in the dataset to a common scale to improve modelling accuracy.

There was no problem of outliers observed in the dataset.

With quadratic features the cost becomes slightly worse.

Below is the visualization of the training set with feature 1:

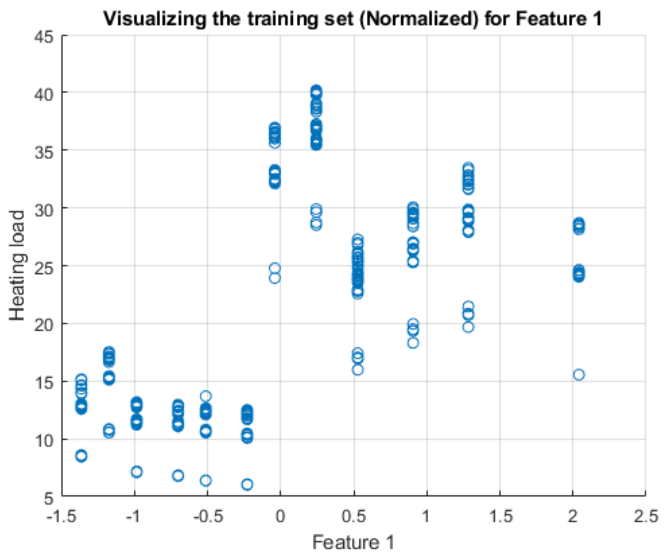

Figure 1: Visualization of training set feature 1

Question 2: Implementation of linear regression with regularization

a) Implement a gradient descent algorithm for linear regression with regularization parameter and create a model for predicting the heating load. Make decision on the selection of the learning rate and regularization parameter, the features. Motivate your choices.

Solution:

The below code was written to train the model using gradient descent with regularization:

```
%% preparing the data for linear regression
m = size(X_train, 1); % no. of observations in the training set
X_train = [ones(m,1), X_train]; % intercept term

% initial theta
theta = zeros(n+1, 1);

% learning parameters
alpha = 0.01; % learning rate
lambda = 0.1; % regularization parameter
iterations = 1000;

% placeholder for the cost function history
J_history = zeros(iterations, 1);

%% training the model with regularized gradient descent
for iter = 1:iterations
    h = X_train * theta;
    error = h - y_train;
    % regularized gradient descent update formula
    theta = theta - (alpha/m) * (X_train' * error + lambda * theta);
    % regularized cost function
    J_history(iter) = (1/(2*m)) * sum(error .^ 2) + (lambda/(2*m)) * sum(theta(2:end) .^ 2);
end
```

And from this a plot of the cost function was generated and the training was visualized as well:

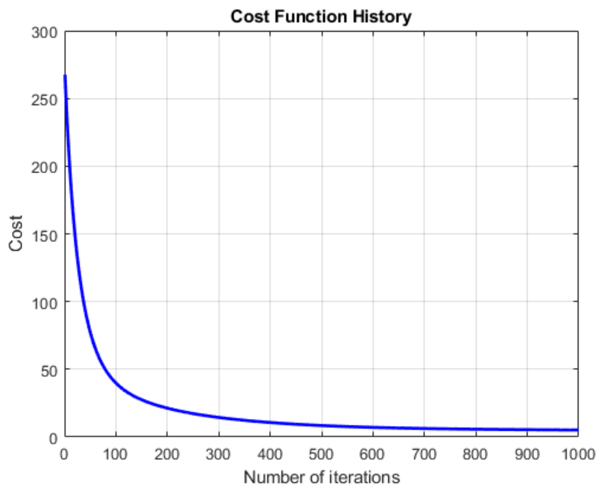

Figure 2: Cost function vs Iterations

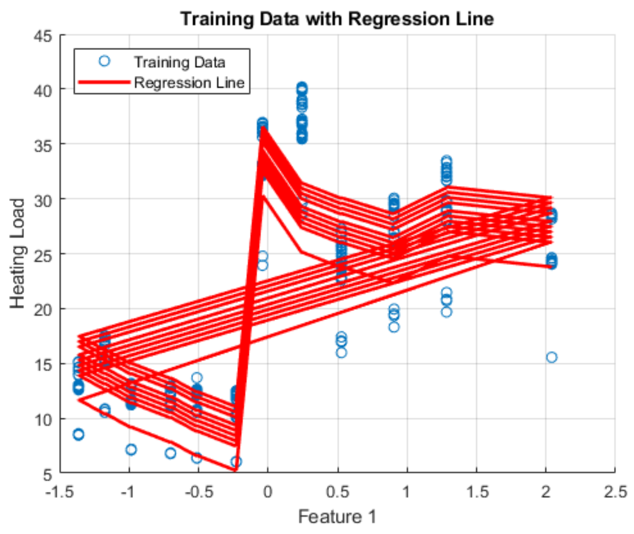

Figure 3: Visualizing model training process

The first feature from the dataset was chosen because the relative compactness can be directly related to the heating load of a house and is a crucial parameter in sustainable building design.

The values for alpha and lambda were chosen to be 0.01 and 0.1 respectively as a starting point and resulted in a good output. Any larger values of the regularization parameter did not result in any significant improvements, there was a slight increase in the MSE value on the validation set and a decrease in MSE value on the test set, and any values of alpha larger than 0.01 resulted in higher MSE on the test set, therefore the initial values for both of these parameters were considered in the end.

b. Why is regularization used? Is it really required to be implemented for this group of features for this project? Why or why not?

Solution: Regularization is used to prevent overfitting by penalizing large parameter values, the necessity of regularization depends on the complexity of the model and the size of the dataset relative to the number of features. If the dataset is small or the number of features is large, regularization can help prevent overfitting. In this assignment the dataset provided had a large number of features and therefore regularization could be helpful to prevent overfitting.

c. Validate the model using the validation set and evaluate the performance of the model. How well does the model predict? Can you make some comments on the usability of the model based on the observed performance?

Solution: After the training, the validation was carried out with the below given code:

```
%% validation
X_val = [ones(size(validation_data, 1), 1), validation_data(:, 1:end-1)]; % also adding
intercept term to the validation set features
y_val = validation_data(:, end);
h_val = X_val * theta; % predictions on the validation set

% mean squared error (MSE) for the validation set
mse_val = mean((h_val - y_val).^2);
fprintf('Mean Squared Error on Validation Set: %f\n', mse_val);
```

Here the intercept term is added to the feature matrix of the validation set and then the hypothesis function is calculated using the theta values from the training process.

After this the Mean Squared Error is calculated for the validation set, which is as given below:

Mean Squared Error on Validation Set: 10.670110

Below is the plot for visualizing the validation performance.

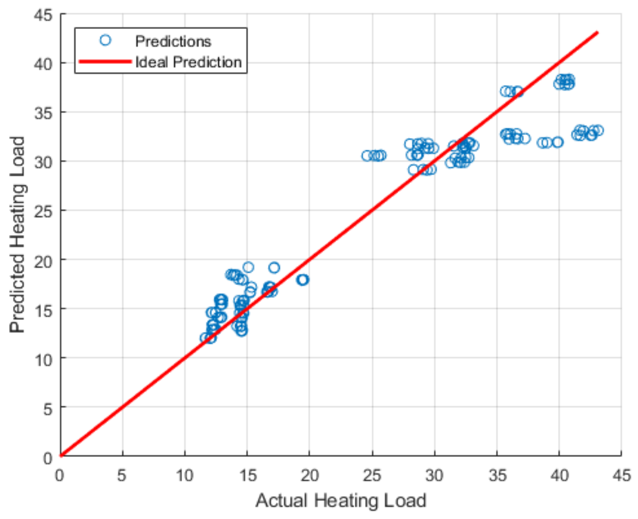

Figure 4: Performance on validation set

The testing process was carried out with the below given code:

```
%% testing

X_test = [ones(size(test_data, 1), 1), test_data(:, 1:end-1)]; % also adding intercept term to
the test set features
y_test = test_data(:, end);

h_test = X_test * theta; % predictions on the test set

% mean squared error (MSE) for the test set
mse_test = mean((h_test - y_test).^2);
fprintf('Mean Squared Error on Test Set: %f\n', mse_test);
```

Here first the intercept term is added to the feature matrix of the test set and then the hypothesis function is calculated using the values of theta from the training process.

After this the Mean Squared Error is calculated for the test set, which is as given below:

Mean Squared Error on Test Set: 9.236193

Below is the plot for visualizing the testing performance.

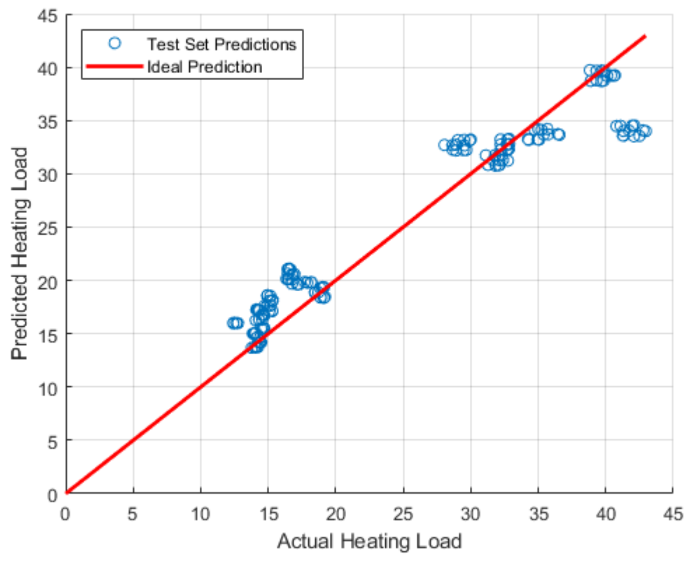

Figure 5: Performance on test set

The performance of the model was evaluated based on the MSE values. The lower MSE values of 10.670110 on the validation set and 9.236193 on the testing set indicate better predictive performance of the model. Additionally, the visual inspection of the plots of the actual vs predicted values with the zero-error line also show that the model has a good predictive accuracy.

Part 2 Artificial Neural Network: Prediction of Drive Failures

Question 3 Data Preparation

The dataset was loaded, set up and normalized with the following code:

```
%% loading and setting up the data
data = readtable('drive_diagnosis_NN.csv');
X = data{3:end, 1:end-1}; % the first two rows were excluded and the last column since it has
the classes
y = data{3:end, end}; % first two rows of the classes column were also excluded because of non
numerical values

%% to ensure class labels are continuous and start from 1
[unique_classes, ~, y] = unique(y);  % this gives class vector in a non repetitive manner
num_classes = length(unique_classes);

%% feature normalization
X = (X - mean(X)) ./ std(X);
```

Explaining the comments in the code again, the first two rows of the loaded dataset contain non numerical values which are not part of the features and hence were excluded while setting up the data. And the 'unique' function was used to make a vector of the classes 1-7 from 'y' in a non-repetitive manner which is important for multi-class classification. After this the feature normalization was also carried out which helps in faster convergence during training.

After this, 'one hot encoding' was carried out for the target variable Y where each class label is represented by a binary vector of length equal to the number of classes, with a 1 in the position corresponding to the class label, and 0s elsewhere.

```
%% one-hot encoding for y
Y = zeros(length(y), num_classes);
for i = 1:length(y)
    Y(i, y(i)) = 1;
end
```

And then the data was split into training and test sets with 70% and 30% split ratio:

```
%% splitting the data into training and test sets
cv = cvpartition(size(X, 1), 'HoldOut', 0.3); % 70% and 30%
idx = cv.test;
X_train = X(~idx,:);
Y_train = Y(~idx,:);
X_test = X(idx,:);
Y_test = Y(idx,:);
```

Question 4: neural network

Based on the provided example code, the following neural network learning algorithm was implemented.

As can be seen in the code below, the input layer size is equal to the number of features which is 48, the hidden layer size is 2 and the output layer size is 7.

```
%% initializing network parameters
input_layer_size = size(X_train, 2);
hidden_layer_size = 2;
output_layer_size = size(Y_train, 2);
```

After some experimentation with the values of alpha, lambda and the number of iterations the values in the following code were kept:

```
% training parameters
alpha = 1; % learning rate
num_iters = 10000; % number of iterations
lambda = 1; % regularization parameter
```

After this the code for training the neural network was written, mostly based on the functions from the example code.

Inside the for loop for the training process, first the forward propagation was implemented as given below:

```
%% trainging the neural network
for iter = 1:num_iters

    % forward propagation
    z2 = X_train * initial_Theta1';
    a2 = [ones(size(z2, 1), 1) sigmoid(z2)];
    z3 = a2 * initial_Theta2';
    a3 = sigmoid(z3);
```

The forward propagation process involves computing the activations of each layer till the output layer. This includes applying weights to inputs, adding biases, and using a sigmoid activation function.

Below you can see the regularized cost function which measures the performance of the model on the training data, this also has regularization added to reduce overfitting:

```
% regularized cost function
    regularization_term = (lambda / (2 * size(X_train, 1))) * (sum(sum(Theta1_reg .^ 2)) +
sum(sum(Theta2_reg .^ 2)));
    J = (1/size(X_train, 1)) * sum(sum(-Y_train .* log(a3) - (1 - Y_train) .* log(1 - a3))) +
regularization_term;
```

After this the backpropagation was implemented which is a process which computes gradients of the cost function with respect to the weights, allowing for the adjustment of weights to minimize the cost. It involves propagating errors backward through the network, and was carried out with the following code:

```
% backpropagation
    delta3 = a3 - Y_train;
    delta2 = (delta3 * initial_Theta2) .* [ones(size(z2, 1), 1) sigmoidGradient(z2)];
    delta2 = delta2(:,2:end); % Taking off the bias row
```

After this the gradient descent was carried out where the weights are updated using gradient descent, which uses the gradients calculated during backpropagation to adjust the weights in the direction that minimally increases the cost function:

```
% accumulate gradients
    Theta1_grad = (delta2' * X_train) / size(X_train, 1);
    Theta2_grad = (delta3' * a2) / size(X_train, 1);
% regularized gradients
    Theta1_grad(:, 2:end) = Theta1_grad(:, 2:end) + (lambda / size(X_train, 1)) * Theta1_reg;
    Theta2_grad(:, 2:end) = Theta2_grad(:, 2:end) + (lambda / size(X_train, 1)) * Theta2_reg;
% gradient descent parameter update
    initial_Theta1 = initial_Theta1 - alpha * Theta1_grad;
    initial_Theta2 = initial_Theta2 - alpha * Theta2_grad;
```

And then, the cost was printed after every thousandth iteration, which gave basically the following output:

```
 Iteration: 1000 | Cost: 1.197374
 Iteration: 2000 | Cost: 1.122759
 Iteration: 3000 | Cost: 1.094881
 Iteration: 4000 | Cost: 1.097748
 Iteration: 5000 | Cost: 1.065027
 Iteration: 6000 | Cost: 1.056024
 Iteration: 7000 | Cost: 1.053214
 Iteration: 8000 | Cost: 1.062033
 Iteration: 9000 | Cost: 1.041430
Iteration: 10000 | Cost: 1.043162
```

After training, the model's performance was evaluated on the test set by computing the accuracy with the following code. First forward propagation was carried out on the test set and then prediction was carried out to test the accuracy:

```
%% calculating prediction and accuracy
% forward propagation on the test set
z2_test = X_test * initial_Theta1';
a2_test = [ones(size(z2_test, 1), 1) sigmoid(z2_test)];
z3_test = a2_test * initial_Theta2';
a3_test = sigmoid(z3_test);

[~, predictions] = max(a3_test, [], 2);
[~, actuals] = max(Y_test, [], 2);
accuracy = mean(double(predictions == actuals)) * 100;
fprintf('Test Set Accuracy: %f\n', accuracy);
```

The result was around 85-90% of accuracy like the output shown below:
Test Set Accuracy: 90.357143

Question 5: network optimization

To make the required analysis, the training cost was also calculated in the code. With a minimum training cost of 1.043162 and a test cost of 1.0722, the difference between these costs is relatively small, suggesting that the model is neither severely overfitting nor underfitting. This closeness in cost values indicates that the model has a good balance between bias and variance. The model seems to generalize well to unseen data, suggesting that its current complexity and the chosen regularization are appropriate for the task at hand.