DA5400 Foundations of Machine Learning

Assignment 1

Roll no. - DA24C026

Name – Rishabh Garg

Problem 1

Write a piece of code to obtain the least squares solution W_{ML} to the regression problem using the analytical solution.

Solution

The Analytical solution a.k.a. least squares solution for the regression problem is found using the normal equation.

$$\mathbf{W_{ML} = (XX^T)^{-1} Xy}$$

Where X is the d x n dimensional feature matrix with n data points and d features, *y* is the n x 1 dimensional target vector with output values corresponding to each datapoint.

The prediction on new data is made using the formula:

$$y_{pred} = X_{new}^T . W_{ML}$$

Where, W_{ML} is our d x 1 dimensional weight vector.

Approach

- Data Loading and Exploration: Initially, loaded the dataset into a Pandas DataFrame for manipulation and analysis.

- Using methods like head () and describe (), inspected the first few rows of the data and generated summary statistics such as the mean, standard deviation, minimum, and maximum values.

- Data Splitting: The dataset consists of two feature columns and one target column. I separated the features and target variables into two NumPy arrays:

    - Feature Array (X): Consisting of two independent variables (features) and a bias term (added as a column of ones to account for the intercept).
    - Target Array (y): Consisting of the dependent variable (target values).

- Analytical Solution (W_{ML}): To compute the Least Squares Solution W_{ML}, applied the closed-form formula:

    - $W_{ML} = (X^T X)^{-1} X^T y$

Where, X is n x d dimensional array and y is n x 1 vector.

- Here, I utilized NumPy's linear algebra functions, such as np.linalg.inv (for matrix inversion), X.T (for matrix transpose), and np.dot (for matrix multiplication). These operations allowed me to efficiently calculate the weight vector W_{ML}, which minimizes the squared error between the predicted and actual target values.

- Prediction Using W_{ML}: Once I obtained W_{ML}, I used it to make predictions on the training data. The prediction for each data point was computed by projecting the feature matrix onto the learned weight vector:

  - $y_{pred} = X^T . W_{ML}$

  This gave me the predicted target values (y_{pred}) for all the data points in the training set.

- Training Error Calculation (MSE): To evaluate the performance of the model on the training data, I computed the Mean Squared Error (MSE). The MSE measures the average squared difference between the actual target values and the predicted values:

  - $MSE = (1/n) * \Sigma (y_{pred} - y)^2$

- This provided a numerical estimate of the model's fit to the training data. By computing the MSE, I were able to quantify the error and assess the accuracy of our analytical solution.

Results / Observations

The weights obtained through the analytical solution are:

$$W_{ML} = [\ 9.89400832,\ 1.76570568,\ 3.5215898\ ]$$

Mean Squared Error: 123.365

Problem 2

Code the gradient descent algorithm with suitable step size to solve the least squares algorithms and plot $\| W_t - W_{ML} \|_2$ as a function of *t*. What do you observe?

Solution

Gradient descent is an iterative algorithm used to solve the least squares regression problem. It aims to minimize the mean squared error (MSE) by iteratively updating the weights based on the gradient of the loss function.

Gradient descent update rule is given by: -

$$w_{t+1} = w_t - \eta \nabla L(w_t)$$

Where, $\nabla L(w_t) = -\frac{2}{n} X^T (y - X w_t)$ is the gradient of the loss function with respect to the weights at iteration *t*.

The convergence of the gradient descent weights (w_t) with the analytical least squares solution (w_{ml}) can be tracked by plotting the Euclidean norm $\| w_t - w_{ML} \|_2$ as a function of the iteration number *t*.

Approach

- Initialization: The feature matrix X (with a bias term) and the target values y were extracted from the dataset.

- The weights *w* was initialized to zero, learning rate (η) was chosen to ensure convergence without oscillations or divergence.

    - The learning rate of $^1/_{(t+10)}$ as a function of *t*, was taken to ensure diverging sum of 1/ η and converging sum of 1/ η^2.

- Gradient Descent Update Rule: The weights were updated iteratively using the gradient of the mean squared error (MSE) with respect to the weights. The update rule is given by:

    $$w_{t+1} = w_t - \eta \nabla L(w_t)$$

    Where, $\nabla L(w_t) = -\frac{2}{n} X^T (y - X w_t)$ is the gradient of the loss function with respect to the weights at iteration *t*.

    The negative of gradient term guides algorithm to decide the direction of descent.

- Convergence Criteria: The algorithm was run for 10,000 iterations or until the change in weights between consecutive iterations was Zero (rounded off to 3 decimals).

- Distance from Analytical Solution: At each iteration, I computed the Euclidean norm of the difference between the current weights w_t and the analytically computed least squares solution w_{ML}.

- Plotting Results: After running the gradient descent algorithm, I plotted $\|w_t - w_{ml}\|_2$ versus the iteration number *t* to visually assess the pattern of gradient descent convergence.

Results / Observations

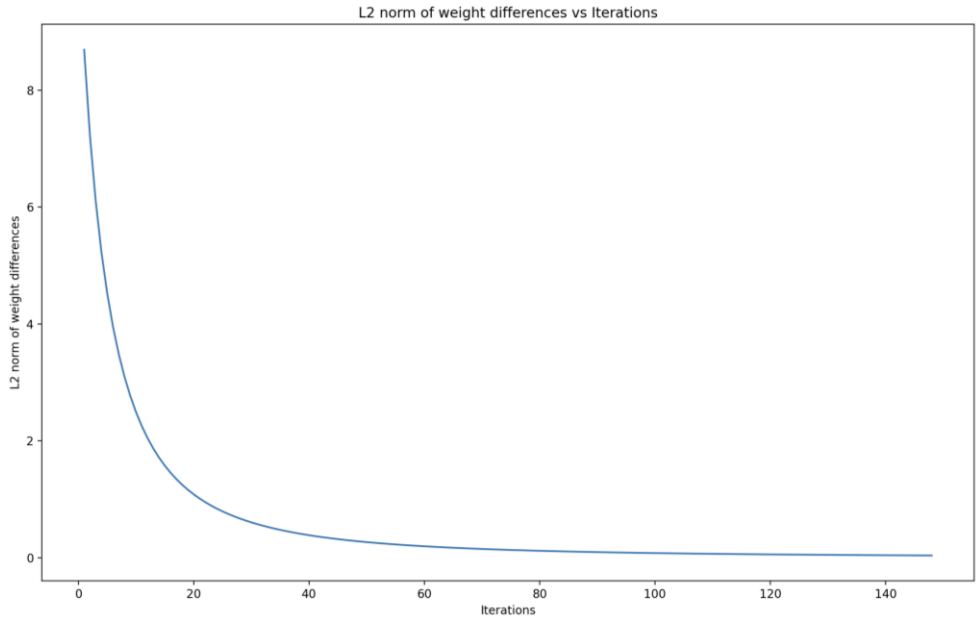
The weights obtained through the Gradient solution are:

$$W_{GD} = [\ 9.858, 1.752, 3.516\ ]$$

Mean Squared Error: 123.366

The Gradient Descent algorithm converged after **148** iterations.

Plot - L2 Norm of weight difference vs Iterations



Key Observation:

The Euclidean norm **decreased sharply within the first 50 iterations**, indicating that the gradient descent algorithm was quickly closing in on the analytical least squares solution.

Problem 3

Code the stochastic gradient descent algorithm using batch size of 100 and plot $||w_t - w_{ML}||_2$ as a function of t. What are your observations?

Solution

Stochastic Gradient Descent (SGD) algorithm with a batch size of 100 is implemented to minimize the least squares loss function. The goal was to track the convergence of the SGD weights (w_t) compared to the analytical least squares solution (w_{ML}) by plotting the Euclidean norm $||w_t - w_{ML}||_2$ as a function of the iteration number t.

Unlike Gradient Descent, in stochastic instead of all 1000 data points only 100 points will be used for every update iteration.

In Stochastic GD the gradient of loss is given by:

$$\nabla L(w_t) = -\frac{2}{b} X_{\text{batch}}^T (y_{\text{batch}} - X_{\text{batch}} w_t)$$

Approach

- After weights initialization, for batch creation, I performed random shuffling of all 1000 data points to create batches. For each iteration, I selected the top 100 points after shuffling. Once all 1000 points were processed, I repeated the random shuffling.

- The weights were updated iteratively using the mini batch of size 100. The update rule was similar to that used in gradient descent, with the arguments changed to X_batch and y_batch :

$$w_{t+1} = w_t - \eta \nabla L(w_t)$$

,

$$\nabla L(w_t) = -\frac{2}{b} X_{\text{batch}}^T (y_{\text{batch}} - X_{\text{batch}} w_t)$$

- For each iteration until convergence, Euclidean norm was calculated to monitor the pattern of convergence towards the analytical solution.

- After convergence, I plotted the norm $|| w_t - w_{ML} ||_2$ against the iteration number *t* to visually assess convergence.

Results / Observations

The weights obtained through the Gradient solution are:

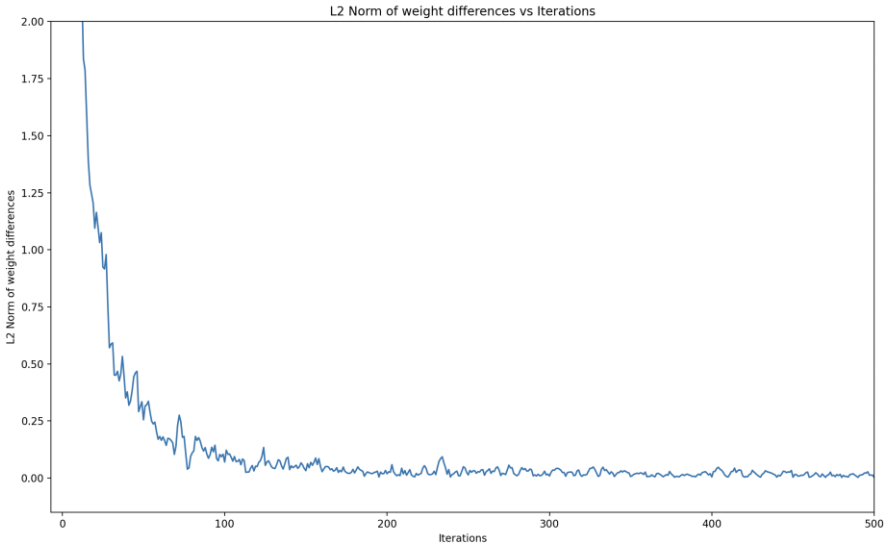
$$W_{SGD} = [\ 9.890,\ 1.766,\ 3.526]$$

Mean Squared Error: 123.365

The Stochastic GD algorithm converged after **1224** iterations.

Note: - (It varies every time I run SGD, but it is always more than 500)

Plot - L2 Norm of weight difference vs Iterations



Key Observations:

- The plot reflects the inherent randomness in SGD, leading to **more variability in the convergence path** compared to batch gradient descent.

- The norm decreased initially but exhibited fluctuations due to the stochastic nature of the updates. As the result of higher variability in SGD algorithm, it converged after significantly higher number of iterations than that of the gradient descent method, indicating that **SGD required more steps than GD** to reach a comparable solution.

Problem 4

Code the gradient descent algorithm for ridge regression. Cross-validate for various choices of λ and plot the error in the validation set as a function of λ. For the best λ chosen, obtain w_R. Compare the test error (for the test data in the file FMLA1Q1Data test.csv) of w_R with w_{ML}. Which is better and why?

Solution

Ridge Regression algorithm introduces a regularization term to the least squares problem to prevent overfitting. Ridge regression aims to find the optimal regularization parameter λ through cross-validation and further I can compare the performance of the ridge regression solution (w_R) with the analytical least squares solution (w_{ML}) on the test data.

Ridge regression modifies the least squares cost function by adding an L2 - norm regularization term, where λ controls the strength of the regularization.

The cost function for Ridge regression is given by:

$$L(w) = \|Xw - y\|_2^2 + \lambda \|w\|_2^2$$

The λ that minimized the validation error during cross validation is selected as the optimal regularization parameter. Using this optimal λ, I can compute the corresponding weights w_R for Ridge Regression.

Finally**,** I can compare the test error (MSE_{Test}) for the ridge regression solution w_R and the analytical least squares solution w_{ML}.

Approach

- Modified Weight Update Rule: The gradient descent weight update rule for ridge regression is modified to include the regularization term:

$$w^{(t+1)} = w^{(t)} - \eta \left(X^T (X w^{(t)} - y) + \lambda w^{(t)} \right)$$

  Here, η is the step size, and λ controls the regularization strength. The additional $\lambda w^{(t)}$ term penalizes larger weights, helping to prevent overfitting.

- Cross-Validation for λ: I performed 5-fold cross-validation. In this approach, the dataset was split into 5 subsets. After shuffling whole dataset, each subset of 200 data points served as a validation set once, while the remaining 80% of the data (800 datapoints) was used for training. This split ensures a good balance between training and validation, and it helps ensure that the results are not overly dependent on any single part of the data.

- Validation Error Calculation: For each value of λ (tested between 1 and 10), I performed gradient descent to minimize the ridge regression loss function on the training data. The mean squared error (MSE) on the validation set was calculated for each fold as:

$$\mathbf{MSE_{val}} = \frac{1}{n_{\mathbf{val}}} \sum_{i=1}^{n_{\mathrm{val}}} (y_i - X_i^T w)^2$$

  The validation errors for all 5 folds were averaged for each λ, and the results were stored in cv_errors.

- Selection of Best λ: Selected the λ that minimized the average validation error across the 5 folds. Using this optimal λ, I ran gradient descent again to obtain the corresponding ridge regression weights w_R .

- Test Error Calculation: Finally, I computed the test error for both the ridge regression solution w_R and the least squares solution w_{ML} as:

$$\text{MSE}_{\text{test}} = \frac{1}{n_{\text{test}}} \sum_{i=1}^{n_{\text{test}}} (y_i - X_i^T w)^2$$

  Then, I compared the test errors for w_R and w_{ML}.

- Plotting Cross-Validation Errors: Plotted the averaged cross-validation errors for different values of λ, identifying the minimum point corresponding to the best λ.

Results / Observations

The optimal λ was found to be **3.5**

The weights obtained through Ridge Regression with λ=3.5 are:

$$W_R = [9.859, 1.758, 3.511]$$

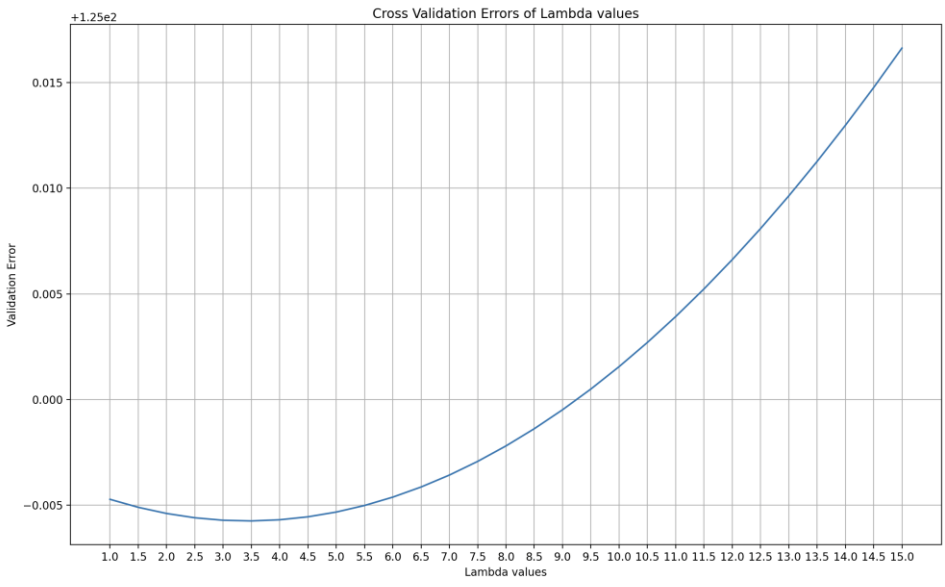
Ridge Regression MSE_{val} (λ = 3.5): 124.99

Test Error:

Ridge Regression MSE_{Test} : 65.95

Analytical Solution MSE_{Test} : 66.01

Plot – Validations errors corresponding to different λ values



Key Observations and Conclusion:

- Smaller Weights in Ridge Regression: The weights from ridge regression were slightly smaller than those from the least squares solution due to the regularization term. This is expected because the L2 -penalty in ridge regression helps **control the size of the weights**, reducing model complexity.

- Validation Error Comparison: The validation error for the ridge regression model was marginally higher than that of the least squares solution. This indicates that ridge regression sacrificed some accuracy on the training data to **avoid overfitting, prioritizing generalization** over fitting the training set too closely.

- Slightly Lower Test Error: The test error for ridge regression ($\boldsymbol{W_R}$) was slightly lower than that of the least squares solution ($\boldsymbol{W_{ML}}$), suggesting that the **regularization improved the model's performance on new data**. This improvement is a direct result of ridge regression's ability to shrink the weights, reducing variance and leading to better generalization.

Problem 5

Assume that you would like to perform kernel regression on this dataset. Which Kernel would you choose and why? Code the Kernel regression algorithm and predict for the test data. Argue why/why not the kernel you have chosen is a better kernel than the standard least squares regression.

Solution:

Kernel Regression is a technique that is used to map the original data into a higher-dimensional feature space such that they can model more complex, non-linear relationships.

Kernel regression replaces the dot product in linear regression with a kernel function, which computes the similarity between two data points in the transformed space.

The regression model is developed using a kernel trick, where instead of mapping data points to a high-dimensional space explicitly, the kernel function computes the inner product between the transformed data points directly in the high-dimensional space:

$$K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$$

where $\phi(x)\phi(x)$ is the transformation function. This avoids the computational burden of calculating the high-dimensional transformation.

In Kernel regression, the prediction is based on a combination of kernel similarities between the test and training points. The training procedure involves solving for the dual coefficients α by minimizing a least squares objective, typically written as:

$$\alpha = K^{-1}y$$

Here, K is the kernel matrix and y is the target vector. Predictions are then made using the kernel matrix between the test points and the training data:

$$y_{pred} = K_{test}\,\alpha$$

Kernel regression allows us to use powerful non-linear transformations efficiently, that helps us to generate models that can capture non – linear relationships more effectively.

Approach

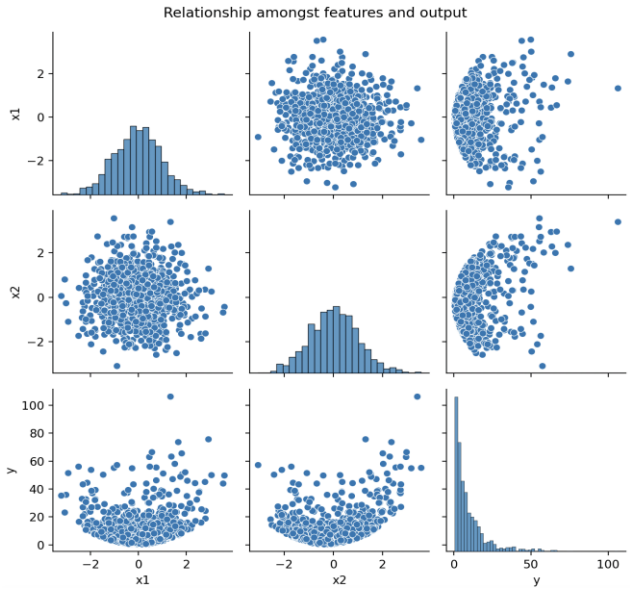
- Selecting Kernel: In order to select right kernel I performed following steps:

    - Feature Interaction Study: To understand how the features relate to each other and to the output, I first printed the correlation matrix:

$$\begin{bmatrix} 1 & -0.0093 & 0.1469 \\ -0.0093 & 1 & 0.2970 \\ 0.1469 & 0.2970 & 1 \end{bmatrix}$$

      This matrix revealed that although x_1 and x_2 were not strongly related, they both slight correlation with the output y, especially x_2

    - Pair Plot Observation: To further study the relationships, I plotted a pair plot of the features and output. This plot visually confirmed the presence of non-linear relationships between the features and the output, suggesting that a linear model would not capture the data's complexity adequately.



Based on the non-linear behavior observed in the pair plot, polynomial kernel was implemented to capture the non-linearity.

- The polynomial kernel of degree d is given by:

$$K(x_i, x_j) = (x_i^T x_j + 1)^d$$

- Cross-Validation for Kernel Degree: To find the optimal degree for the polynomial kernel, we performed cross-validation over degrees ranging from 1 to 10. We used a 5-fold cross-validation method to split the training data into training and validation sets (80-20 ratio).

- Kernel Matrix Calculation and Alpha Estimation: For each degree d, I computed the kernel matrix:

$$K = (X_{\text{train}} X_{\text{train}}^T + 1)^d$$

- Using the kernel matrix, I solved for the coefficients α.

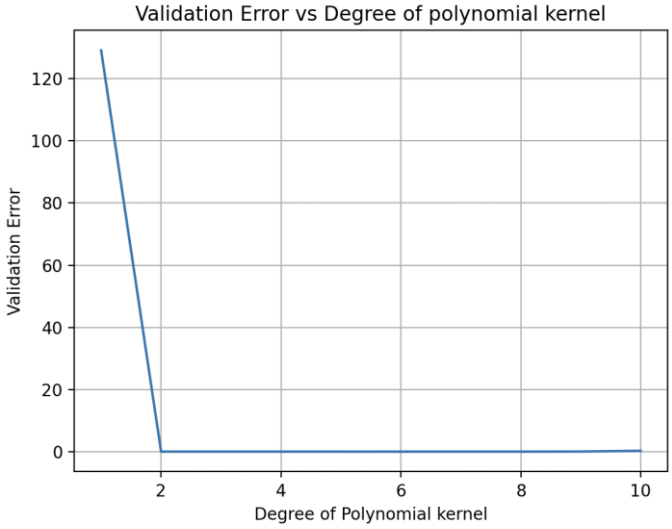
$$\alpha = K_{pinv}\, y$$

- Validation Error Calculation: For each fold, I used the computed α to make predictions on the validation set and calculated the mean squared error (MSE). I averaged the validation errors for all folds and stored the results for each degree.

$$K_{val} = (X_{train} X_{val}^T + 1)^d$$

$$y_{pred} = K_{val}^T \alpha$$

- Degree Selection: After plotting the validation errors as a function of kernel degree, I observed that the MSE_{val} dropped rply at degree 2 and remained very low from degrees 2 to 10. This suggests that a polynomial kernel of degree 2 is sufficient to model the data effectively, with higher degrees not providing any significant improvement.



- Final Model and Test Error: Using the optimal degree d=2, I re-calculated the kernel matrix for the training data and obtained the final α. I then computed the kernel matrix for the test data and made predictions. The test error was then reported to compare with the standard least squares regression model.

Final kernel Model

(i) $K = (X_{train}X_{train}^{T} + 1)^{2}$

(ii) $\alpha = K_{pinv}\, y_{train}$

(iii) $K_{test} = (X_{train}X_{test}^{T} + 1)^{2}$

(iv) $y_{pred} = K_{test}^{T}\, \alpha$

(v) $MSE_{Test} : (1/n) * \Sigma\, (y_{pred} - y_{test})^2)$

Results / Observations:

The **polynomial kernel** of **degree 2** reported least MSE_{val} of 0.0106.

Test Error:

MSE_{Test} Polynomial Kernel (degree 2): 0.0091

MSE_{Test} Least Squares Regression: 66.01

Key Observations:

- The validation MSE dropped sharply at **degree 2** and remained near zero from degrees 2 to 10.

- The test error for the polynomial kernel was **significantly lower** than that of the least squares solution, highlighting the superiority of the kernel in modeling the **non-linear structure** of the dataset.

- Generalization: The polynomial kernel generalized better to the test data compared to the least squares solution. The non-linear transformation provided by the kernel captured the data's underlying structure more effectively, resulting in better predictive performance.

- **Conclusion:** The polynomial kernel of degree 2 outperformed the least squares regression by capturing non-linear interactions in the data, demonstrating that kernel methods can be highly effective for datasets with complex, non-linear relationships.