DA5400 - Foundation of Machine learning
Assignment 1

Gokulakrishnan Balaji
DA24M007

September 21, 2024

Instructions to run the code

All the code is organized within a "code" folder in the submission file. It consists of multiple files, with each model implemented as a separate class. These classes are imported into the 'main.py' file, where they are utilized to address the given questions. To verify the answers, simply run the 'main.py' file.

1. Write a piece of code to obtain the least squares solution to the regression problem using the analytical solution.

The closed solution (Analytical solution) of linear regression is given as

$$w_{ml} = (X^T X)^{-1} X^T y$$

Derivation:

$$\begin{aligned}
\text{loss}(w) &= |Xw - y|^2 \\
&= (Xw - y)^T (Xw - y) \\
&= (w^T X^T - y^T)(Xw - y) \\
&= w^T X^T X w - w^T X^T y - y^T X w + y^T y
\end{aligned}$$

$$\frac{\partial \text{loss}}{\partial w} = 2X^T X w - 2X^T y$$

Set $\frac{\partial \text{loss}}{\partial w} = 0$ to find w_{ml}:

$$\begin{aligned}
2X^T X w_{ml} - 2X^T y &= 0 \\
X^T X w_{ml} &= X^T y \\
\boxed{w_{ml} = (X^T X)^{-1} X^T y}
\end{aligned}$$

Predicting y_{pred} and calculating MSE

$$y_{pred} = X_{test} w_{ml}$$

$$MSE = \frac{\sum (y_{pred} - y)^2}{Number\ of\ data\ points}$$

After using X as X_{train} and y as y_{train} on the given dataset, we get the w_{ml} as [1.76570568, 3.5215898, 9.89400832], with MSE of 66.01 on the test set.

2. Code the gradient descent algorithm with suitable step size to solve the least squares algorithms and plot $|w_{ml} - w_{gd}|^2$ as a function of t. What do you observe?

The gradient descent solution is implemented with learning rate 0.001 , maximum number of iterations as 1000. The update rule of gradient is derived by following set of equations.

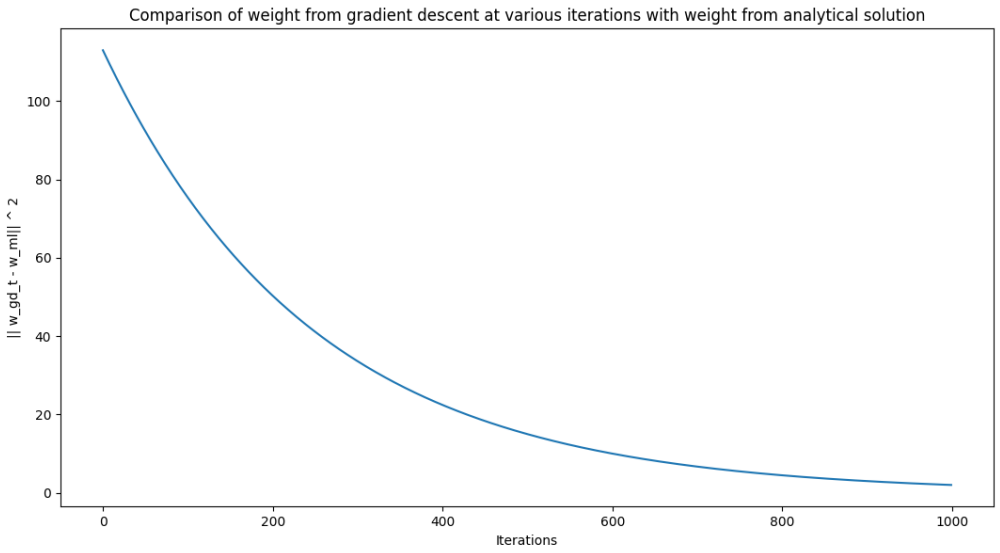
$$MSE = \frac{\sum(x_i w - y_i)^2}{n}$$
$$\frac{\partial(MSE)}{\partial w} = dw = (\frac{2}{n})(Xw - y)X$$

Gradient update Rule:

$$w_{t+1} = w_t - \alpha \, dw_t$$

Running the above algorithm for 1000 iterations, we get the final value of w_{gd} as [1.43016467, 3.13607743, 8.57072706], which gave a test error of 65.67.

Plotting $|w_{ml} - w_{gd}|^2$ as a function of t

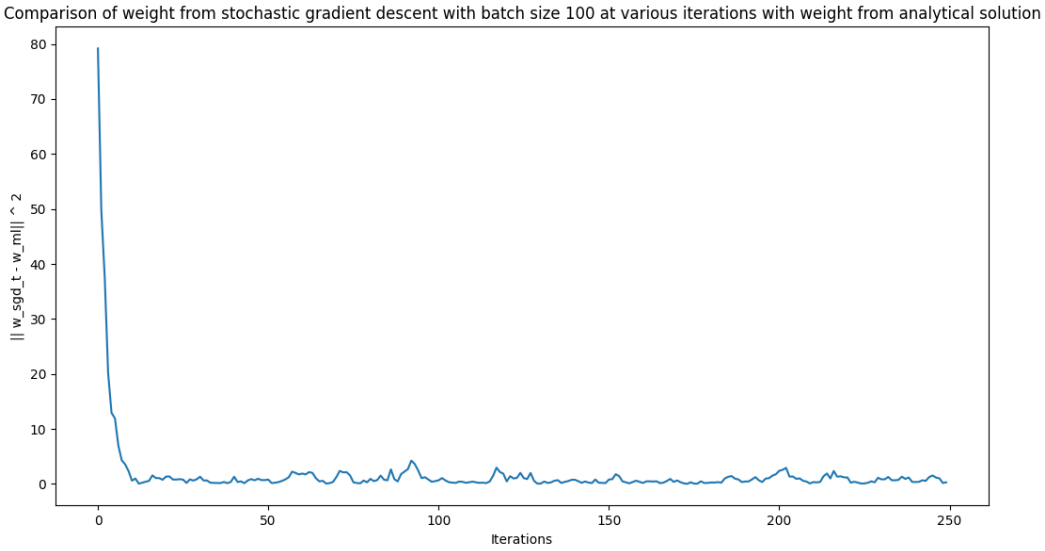


In the given plot, we observe that the difference between w_{ml} and w_{gd} steadily decreases as the number of iterations increases. This indicates that w_{gd} is making progress towards the optimal solution with each iteration. Initially, w_{gd} converges rapidly. However, as it approaches the optimal point, the rate of convergence slows down, as evident in the plot.

3. Code the stochastic gradient descent algorithm using batch size of 100 and plot $|w_{ml} - w_{sgd}|^2$ as a function of t. What are your observations?

A limitation of gradient descent is the computational cost associated with matrix operations when dealing with large datasets. To address this, a common approach is stochastic gradient descent. In stochastic gradient descent, only a random subset of data points is used in each iteration, significantly reducing computational overhead. Over many iterations, this approximation can converge to the optimal weight vector.

Plotting $|w_{ml} - w_{sgd}|^2$ as a function of t



Here, we can see that the plot is not as smooth as gradient descent algorithm. The difference oscillates when w_{sgd} reaches near w_{ml}. We store all these w_{sgd-t} in a list and we finally take average of it as the final w_{sgd}. The reason for this disruption is we are choosing some points randomly. If we choose "bad" points, then we move in wrong direction, increasing the difference. But on average in a long run, we get w_{sgd} close to w_{gd} with lesser computation.

The w_{sgd} obtained with the batch size of 100 is [1.8126199 3.35274312 9.68029142] with the error on test set of 66.05, which is pretty close to gradient descent solution.

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?

The Ridge regression includes a regularisation term in the error function. This helps to prevent overfitting by placing some constraints on the norm of w (penalising w), so that it won't learn the noises in the data.

$$loss = \frac{\sum(Xw - y)^2}{n} + \lambda|w|^2$$

Here λ is the amount of penalty that we assign to w. This λ is a hyperparameter. We need to perform cross validation to find the best value of λ.

Gradient update rule :

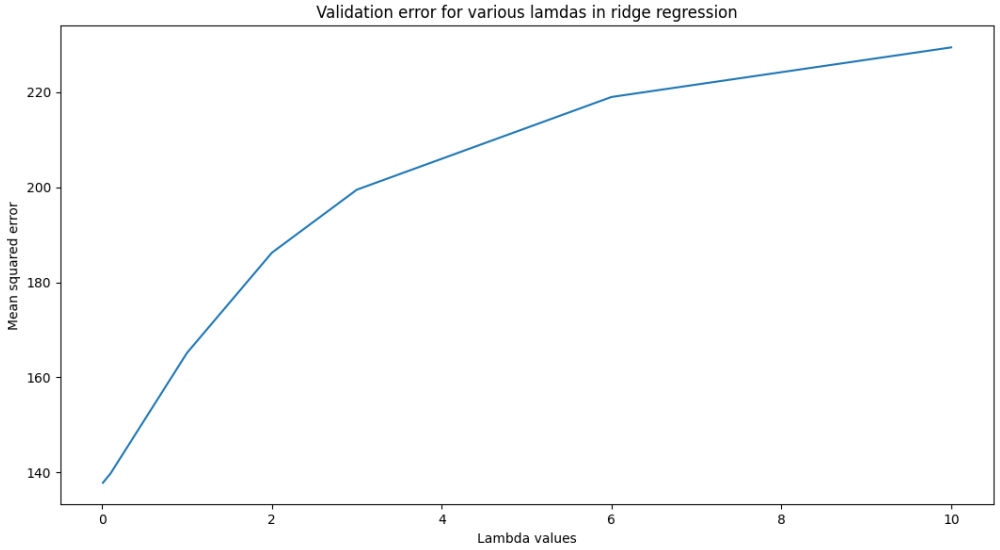
$$Error = \frac{\sum (X_i w - y_i)^2}{n} + \lambda |w|^2$$
$$\frac{\partial (Error)}{\partial w} = (\frac{2}{n})(Xw - y)X + 2\lambda w$$

Thus,

$$dw = (\frac{2}{n})(Xw - y)X + 2\lambda w$$
$$w_{t+1} = w_t - \alpha \, dw$$

Thus we run the above updation algorithm for some number of iteration, we get w_{ridge} .

Plotting validation error as a function of λ



Here we have chosen values of λ from 0.01 to 10. We can observe the best value of λ is 0.01 . In the cross validation, we split our dataset into training and validation set. Let's assume there are n candidates for the hyperparameter. So we create n models with every candidate as a hyperparameter and train it with the train dataset. Then the error is calculated with the validation set by the learnt parameter w. The best hyperparameter is chosen which gives least validation error.

In our case we got the best λ to be 0.01. The corresponding w_{ridge} is [0.77166884, 1.81724788, 4.87300514] that gave the test error of 86.16.

Mean Squared Error with $w_{ml} = 66.01$
Mean Squared Error with $w_{ridge} = 86.16$

We can observe that error of w_{ridge} is higher than error of w_{ml}. This is expected and the reason for this comes from the regularisation term that we added. Even tough we have w that fits the data better than w_{ridge}, we didn't choose that w, as we want to minimise $\lambda.|w|^2$ term as well. This ensures the reduction in the variance between train and test error.

Which is better and why?

The answer to the question which is better w depends on the data. w_{ml} will perform badly compared to w_{ridge} when there are more outliers in the dataset. When the dataset is clean (no outliers), then former is good compared to latter. Since in the given dataset, the noise is less, w_{ml} performs better than w_{ridge}, so w_{ml} is a better parameter.

But when dealing with data from real world scenarios, there will be more noise, thus it is good to go with w_{ridge} on real world datasets.

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.

The primary purpose of kernel regression is to capture non-linear patterns in the data. It achieves this by mapping the non-linear data into a higher-dimensional space, where the relationships become linear.

Which Kernel would you choose and why?

I would choose the Gaussian kernel (also known as the RBF kernel) due to its ability to effectively capture non-linear patterns in the data. Its flexible and smooth structure allows it to efficiently model complex relationships within the data.

The formula for Gaussian kernel is given as

$$KernelDistance(u, v) = e^{-\frac{||u-v||^2}{2\sigma^2}}$$

Here σ is a hyperparameter, we can find the best value of it using cross validation.

Implementation

Step 1

Compute the K matrix on the train dataset. It is calculated by computing $KernelDistance(x_i, x_j)$ for all x_i and x_j in the training dataset.

$$K(i, j) = KernelDistance(x_i, x_j) \quad \forall i, j \in n$$

Step 2

Compute α by solving the equation

$$K\alpha = y$$

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

Step 3

Predict for points from test dataset (X_{test}) .For every point in X_{test} compute y_{pred} by the following formula:

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

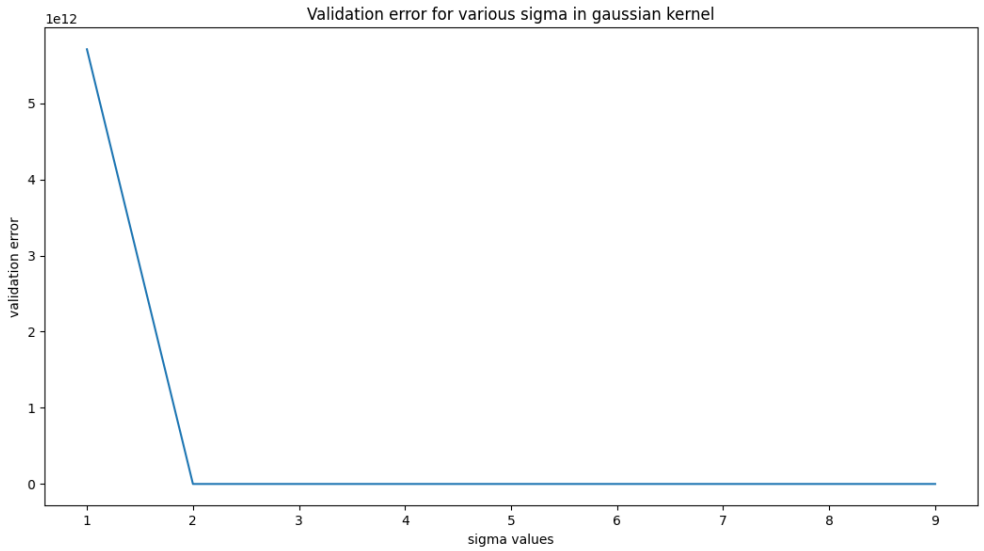
where,

$$K_{test} = [\ KernelDistance(x_{test}, x_i)\ ]\ \ \forall x_i \in X_{train}$$

Cross Validation

We know that σ is a hyper parameter here, thus we perform cross validation here to find the value of σ with least error on validation data.

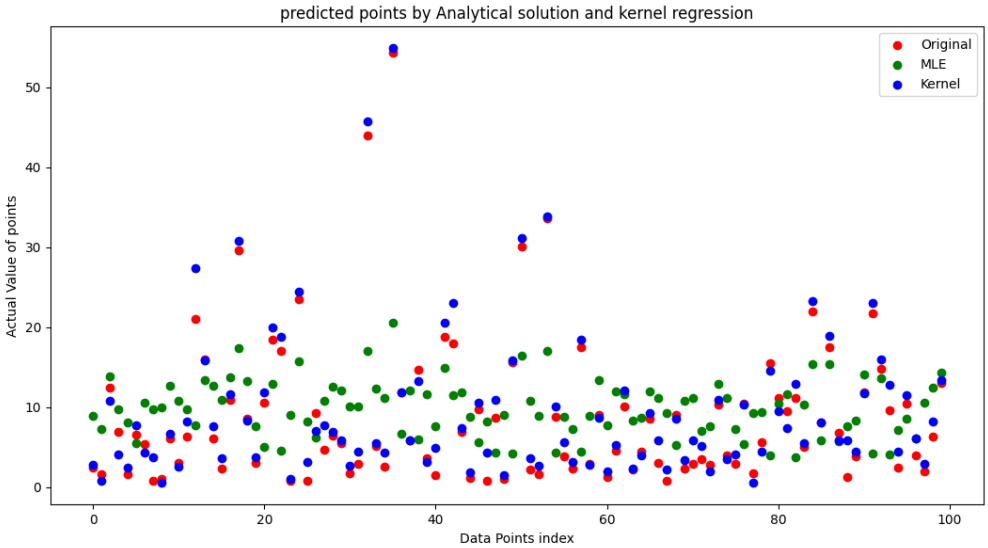
Below is the graph of various values of σ and its validation error.



σ that gave least validation error is 6. Although it is not clearly visible in image, CrossValidation function that we developed returned 6.

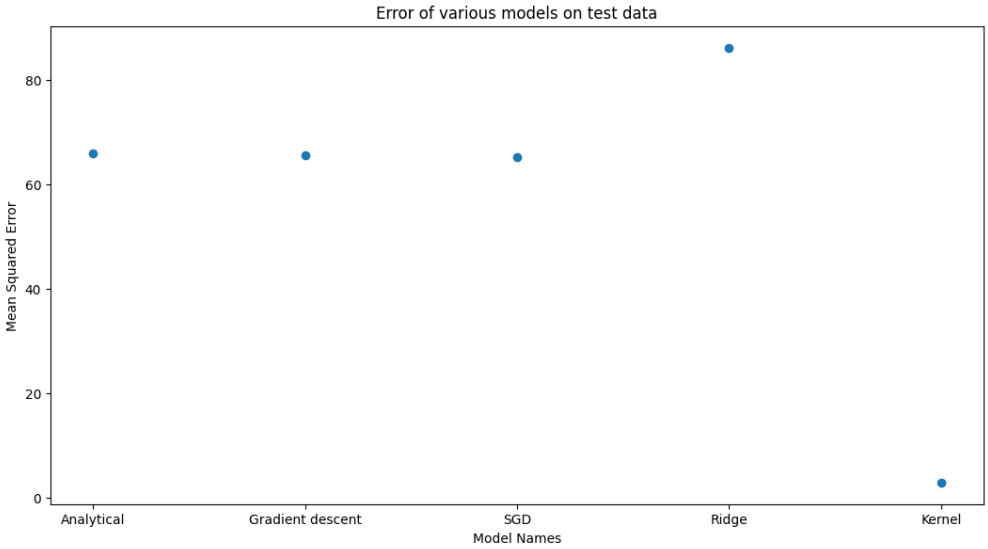
Comparison of kernel regression and least squares regression

With the value of σ as 6, we get MSE of 2.86 compared to 66.01 of least squares regression. Thus we can say that kernel regression performs better than the least squared regression.



In the image above, the blue points, representing the predictions from kernel regression, are closer to the red points (original data points) compared to the green points, which represent the predictions from the least squares solution. The green points exhibit a linear pattern, making it unsuitable for modeling the non-linear original data. In contrast, the predictions from kernel regression closely follow the non-linear trend of the original data points. Therefore, we can conclude that kernel regression performs better than least squares regression.

Comparison of all the models



Model Name	MSE on test data
Closed form solution	66.01
Gradient Descent	65.67
Stochastic Gradient Descent	66.05
Ridge Regression	86.16
Kernel Regression	2.86

Table 1: Model and their MSE

We are plotting the mean squared errors of various models on the test data. It is evident that kernel regression achieves the lowest error among all models due to its capacity to capture non-linear patterns. The analytical (least squares regression), gradient descent model, and stochastic gradient descent model all yield similar errors, as they have comparable weights (though not identical). In contrast, ridge regression exhibits the highest error, likely because it incorporates a regularization term in the loss function to mitigate overfitting.