DA5400: Foundation of Machine Learning

Assignment 1

Nandhakishore C S (DA24M011)

September 22, 2024

1 Details regarding the files in submitted *.zip* file

The given submission **Solutions_DA24M011.zip** file contains three main types of files:

- **Python script** files with ***.py*** extension, where all the **source code** is written. The classes for Linear Regression, Kernel Regression and it's corresponding helper functions are written in separate files.
- The file ***data_loader.py*** reads the dataset and checks for missing values. If the data is clean, it returns a *pandas* dataframe for using in computations.
- **Text file** with *.txt* extension, which contains my **details**.
- A **PDF file** named ***'Report.pdf'***(i.e.) this file, with the explanation for the solutions for the above question. The latex file for the report can be found here. There is also another file named ***rough_work.pdf*** which has the rough work and miscellaneous code which were tried during implementing the solutions for assignment.
- **To run** the python scripts, keep both the train and test datasets, ***linear_regression.py***, ***kernel_regression.py*** and ***data_loader.py*** files in the same directory and run *python3 main.py* in terminal to see the results.
- The **graphs** are displayed while executing the code and also saved in the same directory as the code files.

2 Questions

You are given a data-set in the file *FMLA1Q1Datatrain.csv* with 1000 points in $(\mathbb{R}^2, \mathbb{R})$ (Each row corresponds to a data-point where the first 2 components are features and the last component is the associated y value).

i. Write a piece of code to obtain the least squares solution $\mathbf{w}_{ML}$ to the regression problem using the analytical solution.

Solution

The given dataset contains 1000 rows and 3 columns where, each row is data-point. The first two elements of a row are features (named as x_1, x_2) and the third element is the label y. As a regression problem, we can see that, the labels are $\mathbb{R}$.

To find the analytical solution, we model the regression problem as follows:

$$X \in \mathbb{R}^{1000\times 2} \quad (i.e.)\ X = [\,x_1\ x_2\,] \Rightarrow x_1, x_2 \in \mathbb{R}^{1000\times 1}$$

$$y \in \mathbb{R}^{1000\times 1}$$

The hypothesis function is modeled as a linear combination of features and weights
Each target (label) variable is modeled as:

$$y_i = w_1x_1 + w_2x_2 = [w_1\ w_2]\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

For the optimal value of $\mathbf{w}_{ML}$, we formulate the objective function as a Sum of Squares function

$$\mathcal{L}(w, X, y) = \arg\min_{w} \sum_{i=1}^{i=1000} (\hat{y}_i - y_i)^2$$

where $\hat{y}$ is the prediction for a test data-point. The vector and matrix equation are as follows:

$$\mathcal{L}(w, X, Y) = \arg\min_{w} \sum_{i=1}^{i=1000} \left(w^T x_i - y_i\right)^2$$

The matrix form will be:

$$y = Xw \qquad (i.e) \quad X \in \mathbb{R}^{1000\times 2},\ w \in \mathbb{R}^{2\times 1}\ \&\ y \in \mathbb{R}^{1000\times 1}$$

The summation vanishes as we know that, for any vector

$$||x||_2^2 = x^T x = \sum_{i=1}^{i=d} x_i^2 \qquad x \in \mathbb{R}^d$$

Thus, finally we can express the loss function as:

$$\mathcal{L}(w, X, y) = \arg\min_{w} ||Xw - y||_2$$

To find the optimal value of **w**, we can find the critical point for the objective function by taking it's partial derivative with respect to **w**.

Note that, this process is same as finding the Maximum Likelihood Estimate (MLE) of **w**, such that the target is modeled with noise as standard Gaussian.

$$\frac{\partial \mathcal{L}}{\partial w} = \frac{\partial}{\partial w}((Xw - y)^T (Xw - y)) = 0$$

$$\frac{\partial}{\partial w}((w^T X^T - y^T)(Xw - y)) = 0$$

$$\frac{\partial}{\partial w}(w^T X^T X w - 2X^T y w + y^T y) = 0$$

$$2X^T X w - 2y^T X = 0$$

The analytic solution is as follows:

$$\mathbf{w}_{ML} = (X^T X)^{-1} X^T y = X^{\dagger} y$$

where $X^{\dagger}$ is the Pseudo Inverse matrix.
In the above equation, geometrically, the labels are projected into the subspace by spanned by the features. As the features are matrices, the projection acts as the pseudo inverse of the feature matrix.

The $\mathbf{w}_{ML}$ can also be done with a bias feature added to the given data matrix. For the ease of calculations, the bias term is engulfed by the data matrix and then multiplied with weight vector. Note that, as the number of columns in the data matrix increases, another element is added in the weight vector, as bias is a feature now.

For implementing the analytical solution in Python, the *NumPy* library is used. Refer code file: *linear_regression.py*

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

Solution

Gradient Descent algorithm is a sequential learning technique, where the model learns (or) understands the structure of the relationship between data features minimising the error in predictions and updating the weights with a suitable step-size and penalty. Data points are considered one at a time, and the model parameters are updated after each such presentation with a suitable step size / learning rate (represented as η). The general equation of gradient descent is given as follows:

$$\mathbf{w}^t = \mathbf{w}^{t-1} - \eta \nabla \mathcal{L} \qquad \textit{where } \eta > 0$$

where the gradient is calculated with respect to the parameter which is being learnt. The value of weight vector $\mathbf{w}$ is initially set to $\mathbf{w}^0$ with random values. For the given problem, the Loss function's differential would be as follows:

$$\frac{\partial \mathcal{L}}{\partial w} = 2X^T X w - 2X^T y$$

Rearranging the above equation,

$$\frac{\partial \mathcal{L}}{\partial w} = 2X^T (Xw - y) = 2X^T (\hat{y} - y)$$

The above equation is then incorporated into the gradient descent equation:

$$\mathbf{w}^t = \mathbf{w}^{t-1} - \eta(2X^T(\hat{y} - y))$$

The above equation is implemented in code using a for loop iterating over a fixed number of iterations. Refer code file: *linear_regression.py.*

Using *NumPy's linalg.norm* function, the euclidean distance $||\mathbf{w}^t - \mathbf{w}_{ML}||_2$ is calculated in stored in an array for plotting. Using *Matplotlib* library, the graph $||\mathbf{w}^t - \mathbf{w}_{ML}||_2$ vs t (*iterations*) is plotted.

Finding from the graph are as follows (Refer Figure 1 and 2):

- For a chosen learning rate, initially the weights are initialised randomly and then the weights are updated using the Gradient Descent Algorithm.
- After a number of iterations, the weights converge to a value very close to $\mathbf{w}_{MLE}$.
- In the initial iterations, the distance between the weights in gradient descent in $\mathbf{t}^{th}$ iteration and $\mathbf{w}_{MLE}$ is large.
- When the gradient descent algorithm converges, the distance between $\mathbf{w}_{MLE}$ and $\mathbf{w}^t$ is very small, very close to zero.
- Thus, it is evident that, for a chosen learning rate, with a fixed number of iterations, the gradient descent solution and the analytical solution are similar (or) same in same cases.
- The graph plotted for solving linear regression for the given train data, with bias and without bias (for both the cases the graph is same!)
- It is interesting to note that, for the case where the solution has bias term added to the weights, the distance between $\mathbf{w}_{MLE}$ and $\mathbf{w}^t$ is large at the initial rounds of iterations, but it converges as same as the solution for non - bias solution!

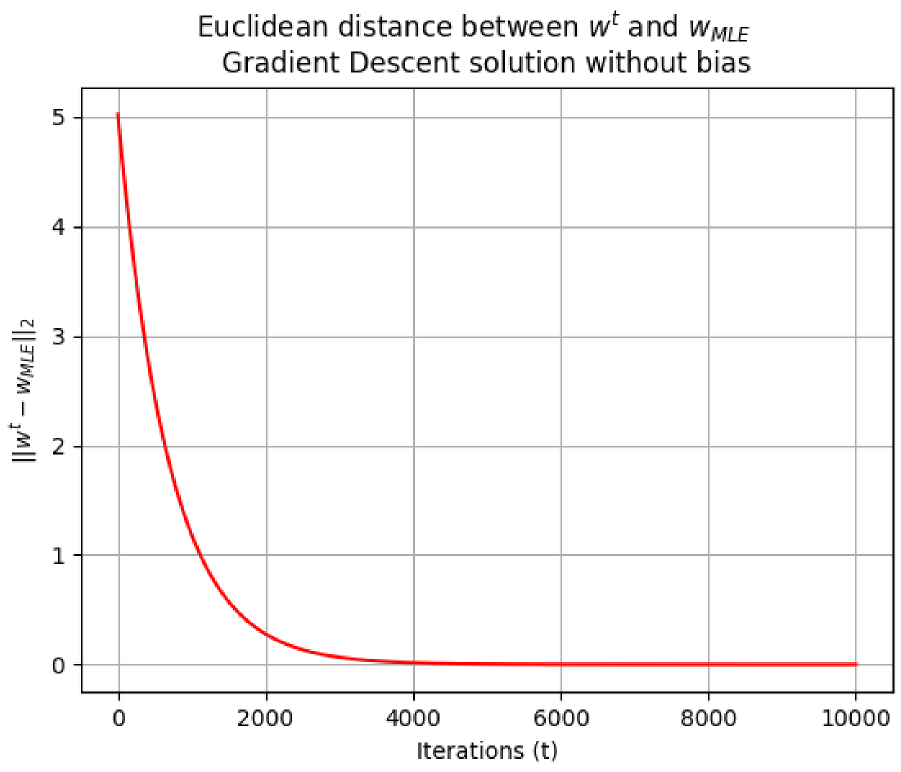

Figure 1: $||\mathbf{w}^t - \mathbf{w}_{MLE}||_2$ **for Gradient Descent solution without bias**

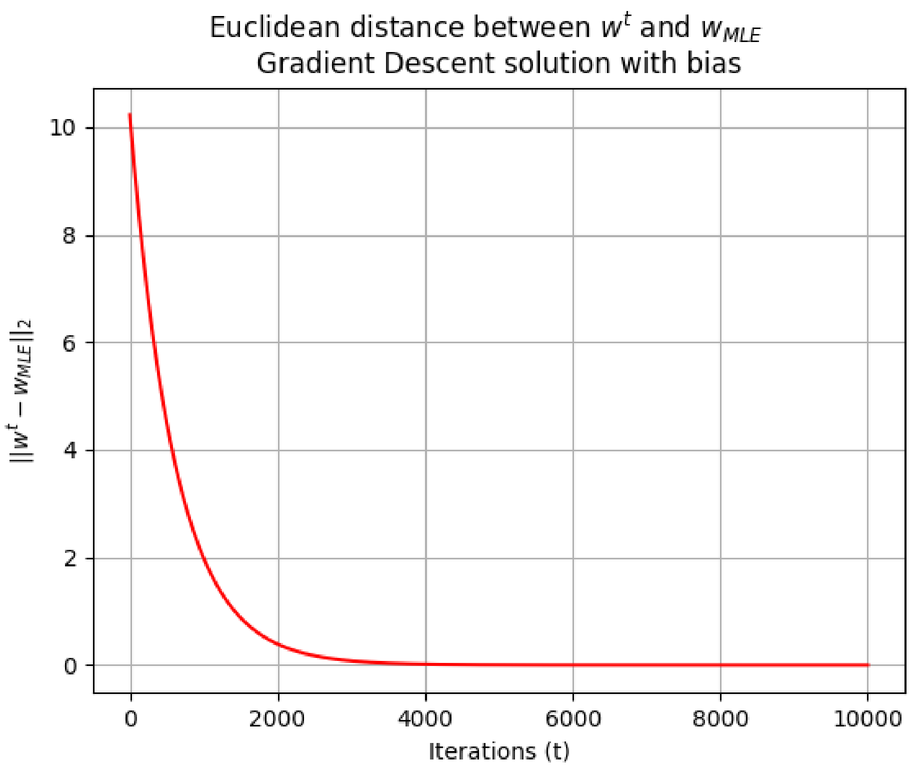

Figure 2: $||\mathbf{w}^t - \mathbf{w}_{MLE}||_2$ **for Gradient Descent solution with bias**

iii. Code the stochastic gradient descent algorithm using batch size of 100 and plot $||\mathbf{w}^t - \mathbf{w}_{ML}||_2$ as a function of t. What are your observations?

Solution

Stochastic Gradient Descent (SGD) algorithm is a sequential learning technique [3][Bishop Book], where the model learns (or) understands the structure of the relationship between data features minimising the error in predictions and updating the weights with a suitable step-size and penalty. In Gradient Descent this is done for each data point, but in SGD, this is done for batches of data. Data points are considered one at a time, and the model parameters are updated after each such presentation with a suitable step size / learning rate (represented as η). The general equation of gradient descent for t iterations is given as follows:

$$\mathbf{w}^t_{batch} = \mathbf{w}^{t-1}_{batch} - \eta\nabla\mathcal{L} \qquad where\ \eta > 0$$

where the gradient is calculated with respect to the parameter which is being learnt. The value of weight vector $\mathbf{w}$ is initially set to $\mathbf{w}^0$ with random values. For the given problem, the Loss function's differential would be as follows: (The equation for weight updation is same as gradient descent, but the data-points are taken in batches)

$$\mathbf{w}^t_{batch} = \mathbf{w}^{t-1}_{batch} - \eta(2X^T(\hat{y} - y))$$

Once the weights are updated, the average of the weights are taken as follows to do final predictions:

$$\mathbf{w}^t_{SGD} = \frac{1}{T}\sum_{t=1}^{t=T} w_t \qquad where\ T\ is\ batch\ size$$

The above equation is implemented in code using a for loop iterating over a fixed number of iterations. Refer code file: *linear_regression.py*. As mentioned in the question, the SGD is done for a batch size of 100.

Using NumPy's *linalg.norm* function, the euclidean distance $||\mathbf{w}^t - \mathbf{w}_{ML}||_2$ is calculated in stored in an array for plotting. Using *Matplotlib* library, the graph $||\mathbf{w}^t - \mathbf{w}_{ML}||_2$ vs t (*iterations*) is plotted.

Finding from the graph are as follows (Refer Figure 3 and 4):

- The distance between $\mathbf{w}^t$ and $\mathbf{w}_{ML}$ is initial at the starting iterations is high and as the weights are updated for minimum error for predictions, the distance decreases and is very close to zero.
- The graph plotted for gradient descent solution (figure 1 and 2) is smooth as each data point is visited and the weights are updated. For SGD, the graphs looks a little erratic behavior observed in the distance between $\mathbf{w}^t$ and $\mathbf{w}_{ML}$ as we introduce randomness in the optimisation problem by taking data in batches for updation.
- The graph plotted for solving linear regression for the given train data, with bias and without bias (for both the cases the graph is same!)
- It is interesting to note that, for the case where the solution has bias term added to the weights, the distance between $\mathbf{w}_{MLE}$ and $\mathbf{w}^t$ is large at the initial rounds of iterations, but it converges as same as the solution for non - bias solution!

iv. 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 $\mathbf{w}_R$. Compare the test error (for the test data in the file *FMLA1Q1Datatest.csv*) of $\mathbf{w}_R$ with $\mathbf{w}_{ML}$. Which is better and why?

solution

The objective function for ridge regression is as follows:

$$\mathcal{L}(w, X, y) = \arg\min_{w} ||Xw - y||_2 + \lambda||w||^2 \qquad where\ \lambda > 0$$

where $||w||^2 = w^T w$.

As discussed in question (ii), the gradient descent update function (taking the additional term in consideration) is as follows:

$$\mathbf{w}^t = \mathbf{w}^{t-1} - \eta(2X^T(\hat{y} - y) + 2\lambda w)$$

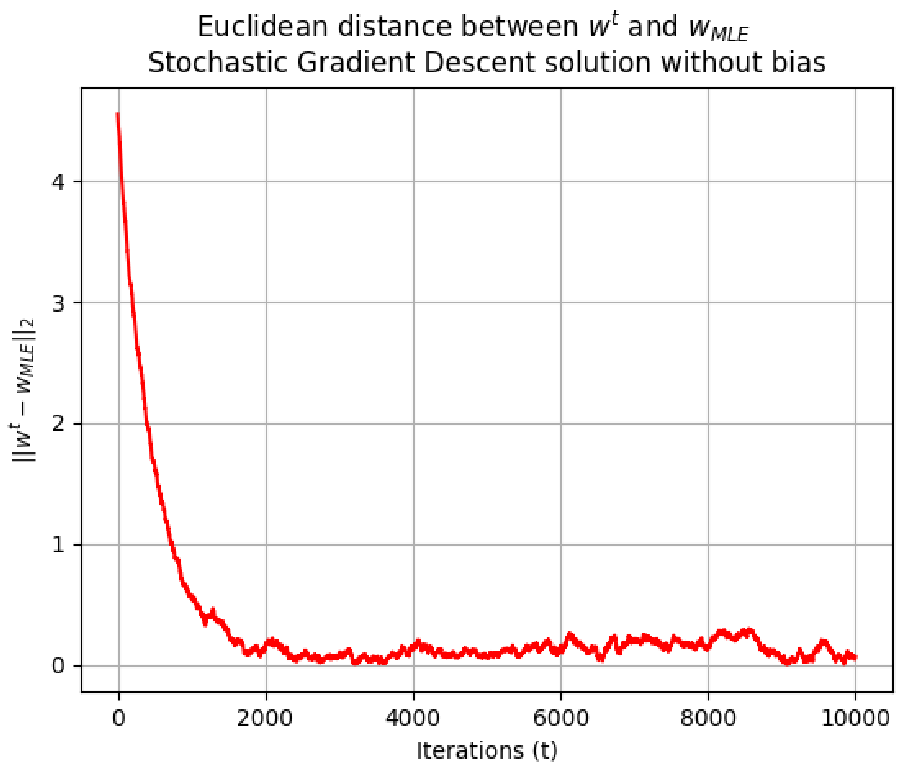

Figure 3: $||\mathbf{w}^t - \mathbf{w}_{MLE}||_2$ **for Stochastic Gradient Descent solution without bias**

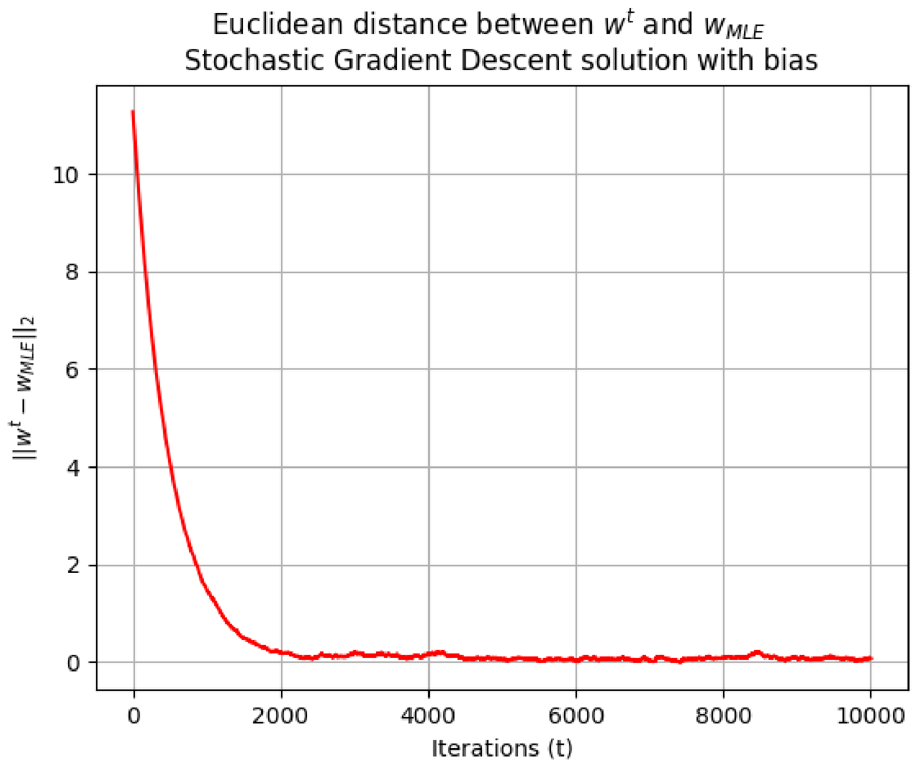

Figure 4: $||\mathbf{w}^t - \mathbf{w}_{MLE}||_2$ **for Stochastic Gradient Descent solution with bias**

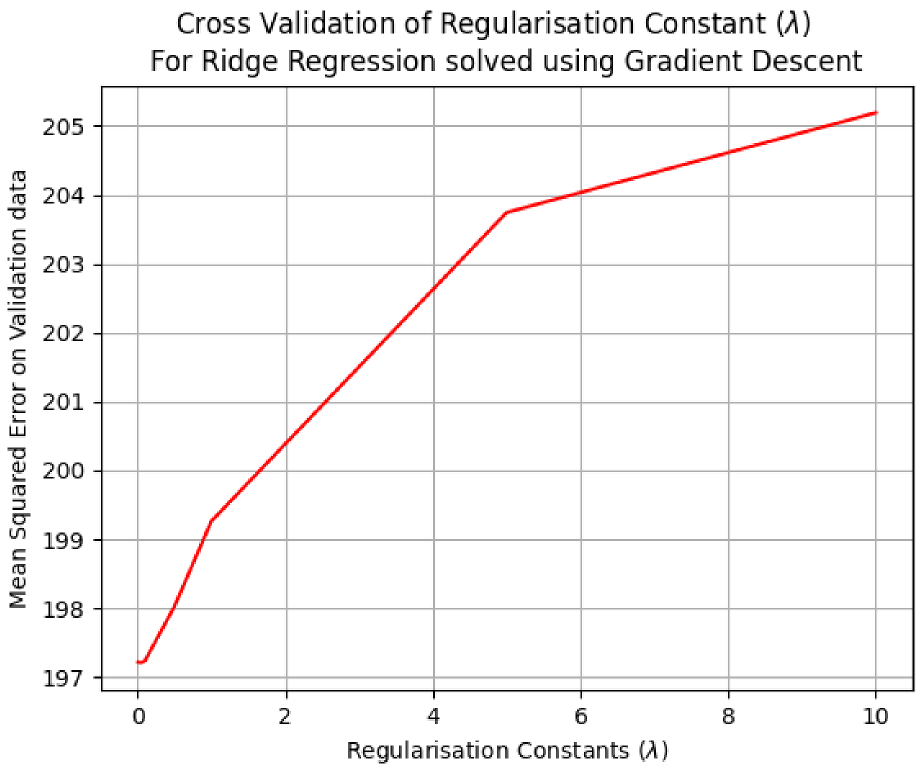

Figure 5: $||\mathbf{w}^t - \mathbf{w}_{MLE}||_2$ **for Stochastic Gradient Descent solution with bias**

The above equation is implemented in code using a for loop iterating over a fixed number of iterations. Refer code file: *linear_regression.py*.

For cross validation, the training data is split into validation and training sets and then different values of the regularisation constants (λs) are tested for minimum errors. A graph is plotted where errors corresponding with respect to different values of regularisation constants (λs). **From the graph it is evident that, the suitable regularisation constant for our problem is 0.01** . Refer Figure 5.

Now, regarding the test error for $\mathbf{w}_{ML}$ and $\mathbf{w}_R$, in this problem the error values are very close for chosen $\lambda = 0.01$. **In fact, test error for $\mathbf{w}_R$ is slightly more than $\mathbf{w}_{ML}$.**
The test error for $\mathbf{w}_{ML} = 142.76610663$
The test error for $\mathbf{w}_R = 142.80531666$

This phenomenon can be explained by understanding the relationship:

$$Covariance(\mathbf{w}_{ML}) = \sigma^2 (XX^T)^{-1}$$

$$Covariance(\mathbf{w}_R) = \sigma^2 (XX^T + \lambda I_n)^{-1}$$

In the above equation, σ^2 is the variance of the noise which is in the system, for which $\mathbf{w}_{ML}$ is calculated.

Also note that, the gradient descent solution for ridge regression for the best chosen regularisation constant is similar (or) same as the analytical solution of ridge regression with same chosen regularisation constant.

Thus the goodness of the weight term depends on the value $(XX^T)^{-1}$. For the given problem, the values of $(XX^T)^{-1}$ for both ridge and ordinary regression are same and addition of regularisation constant on the diagonal entries does not have a big impact because of this. Thus even when L2 regularisation is done, the error will be similar and only differ by very small value!

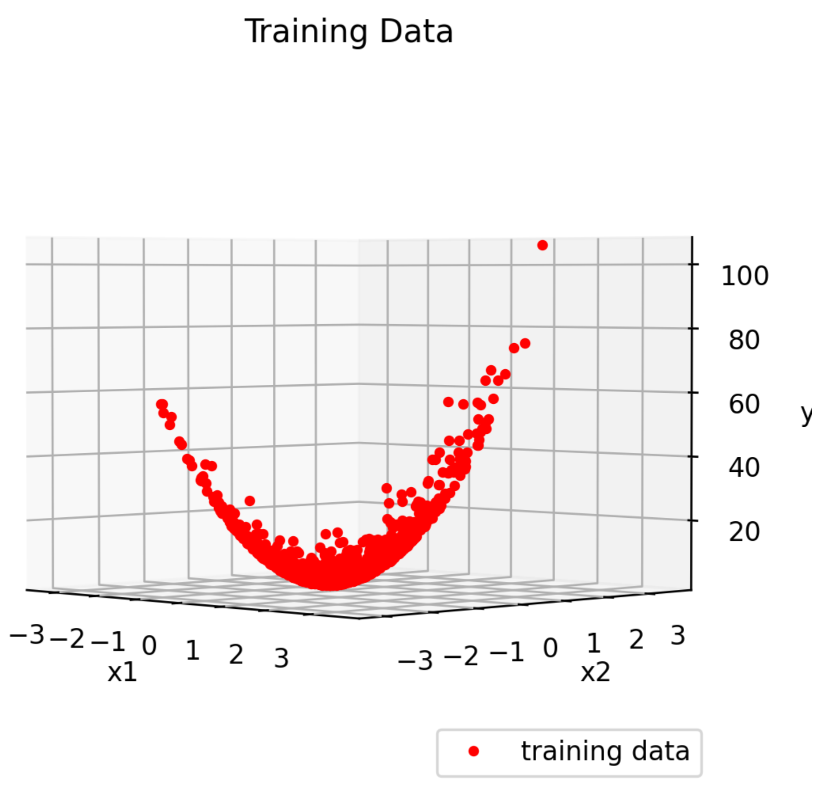

Figure 6: **3D Visualisation of Training Data**

v. 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 abetter kernel than the standard least squares regression.

Solution

Before going into implementing kernel regression, by doing some Exploratory Data Analysis (EDA), it is found that, the train data features have quadratic relationship. This is shown by visualising the train data. (Refer Figure 6)

From figure 6, the following observations are made:

- It appears that, the training data points are sampled from a 3D parabola (i.e.) Parabolic ellipsoid. It's standard form as follows:
$$z = \frac{(x-h)^2}{a^2} + \frac{(y-k)^2}{b^2} \qquad \textit{where } a > b$$
where a is the length of major axis and b is the length of minor axis. (h, k) is the point at which the paraboloid is centered.
- A standard analytical solution if done for the training data and in 3D space, it fits a plane to the curve. When a bias term is introduced in the solution, the plane is shifted offset to the origin by the bias term. (Refer Figure 7 and 8)
- Now that, it is evident that the features in the training data have a quadratic relationship (specifically a **standard elliptical paraboloid**), a corresponding fit is done. It is observed the there is some tilting (specifically a bit of yaw and pitch) to the curve and the fit. (Refer Figure 9)
- To adjust the turning, the rotation matrices are Incorporated into the equation (refer *rough_work.pdf* for the matrices) and a line search is performed to find the angles which fit the training data with less error. Again, the analytical solution is applied to the following data and we get a perfect fit! (Refer Figure 10)
- Thus it is evident that, a polynomial kernel of degree 2 (i.e.) a **quadratic kernel** or **radial basis kernel** would be a perfect choice to do do feature transformations!

Thus **Quadratic** and **Radial Basis** kernels are chosen to do regression and the quadratic kernel with L2 regularisation gives a mean squared error of 0.00980628 for the test data! (check the code for accurate value!).

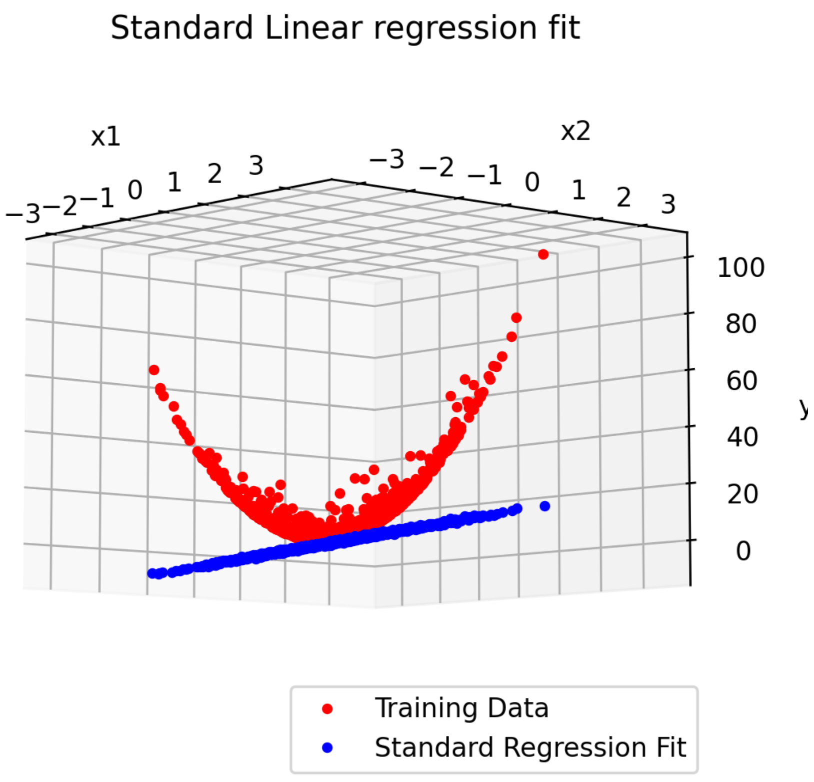

Figure 7: **Analytical solution without bias**

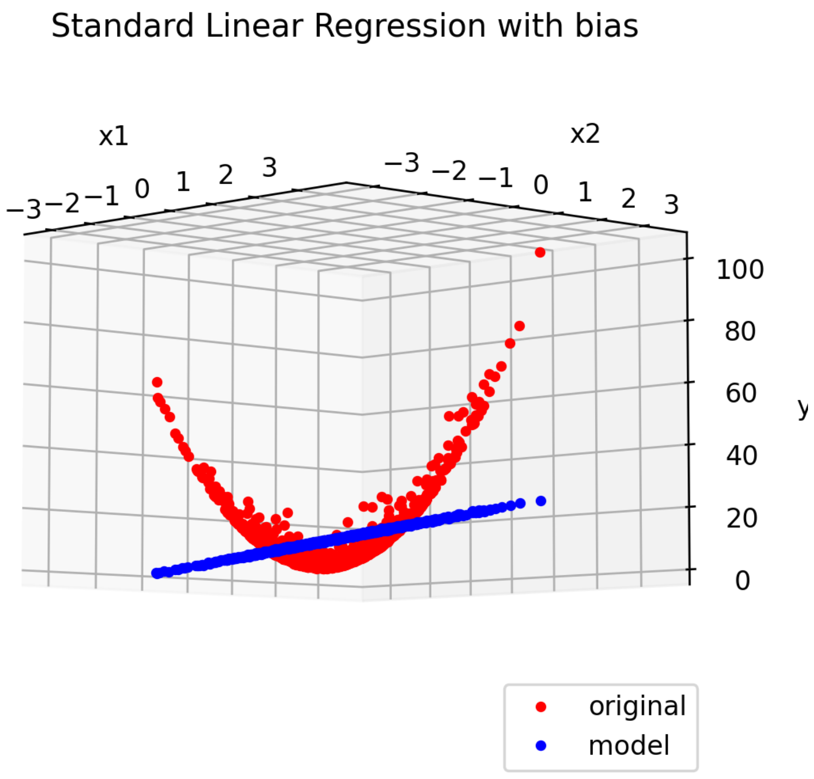

Figure 8: **Analytical solution with bias**

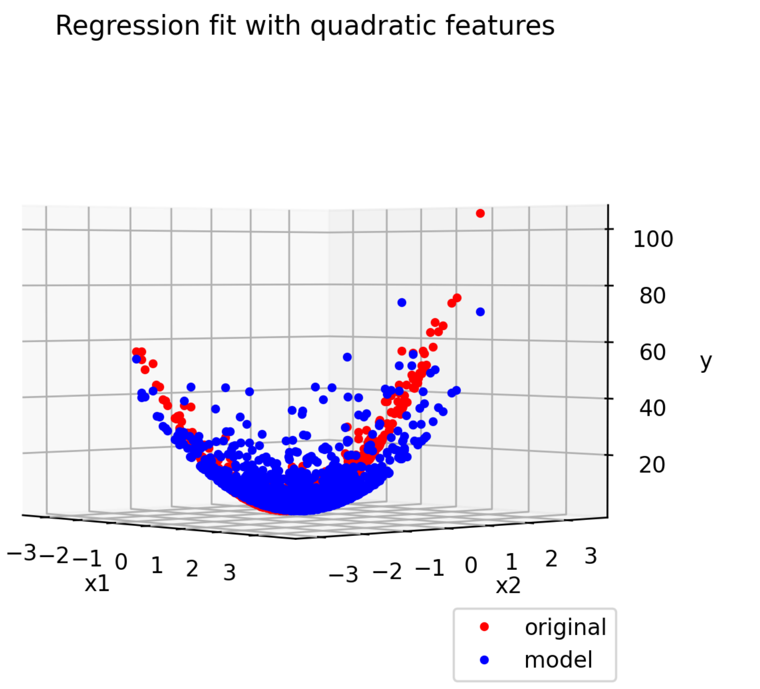

Figure 9: **Analytical solution using quadratic feature transformations**

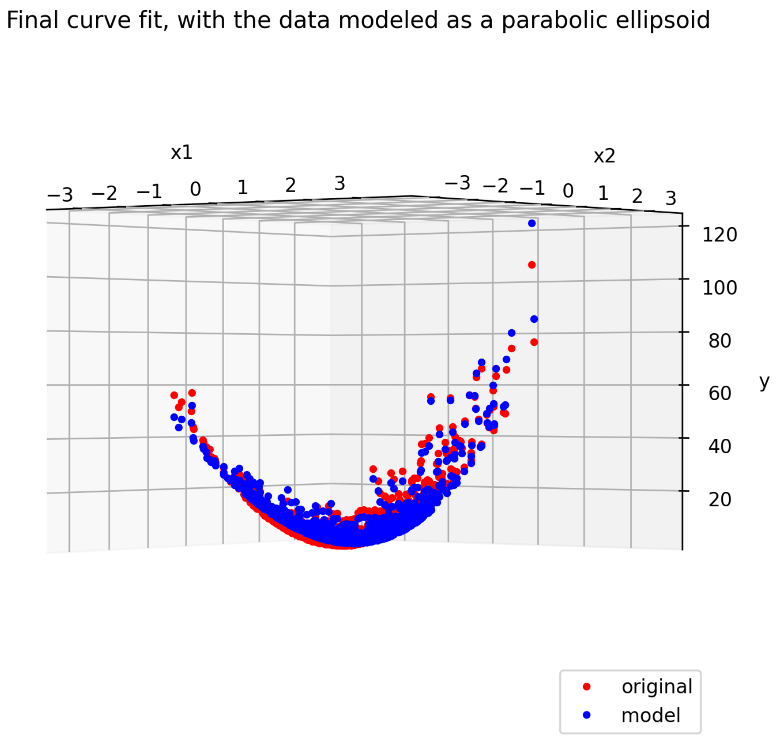

Figure 10: **Analytical solution using quadratic feature transformations including rotation matrices.**

3 References

i All the equations and mathematical concepts for Regression and related concepts are referred from the book: *Bishop, Christopher M. Pattern Recognition and Machine Learning. New York :Springer, 2006.*

ii Programming Language used: *Python3 - Van Rossum, G. Drake, F.L., 2009. Python 3 Reference Manual, Scotts Valley, CA: CreateSpace.*

iii Python Libraries used:

- *Numpy - Harris, C.R., Millman, K.J., van der Walt, S.J. et al. Array programming with NumPy. Nature 585, 357–362 (2020). DOI: 10.1038/s41586-020-2649-2*
- *Pandas - McKinney, W. others, 2010. Data structures for statistical computing in python. In Proceedings of the 9th Python in Science Conference. pp. 51–56.*
- *Matplotlib - J. D. Hunter, "Matplotlib: A 2D Graphics Environment", Computing in Science Engineering, vol. 9, no. 3, pp. 90-95, 2007*