Institute for Advanced Studies
in Basic Sciences
Gava Zang, Zanjan, Iran

Implementation of Linear Regression in Python

Presented by: Reza Pishkar (r.pishkar@iasbs.ac.ir)
Supervised by: Dr. Parvin Razzaghi (p.razzaghi@iasbs.ac.ir)

Winter 2024

Linear Regression

- Machine learning method used to model the relationship between a dependent variable and one or more independent variables by fitting a linear equation (line, plane or hyperplane) to observed data.
- It predicts the value of the dependent variable based on the input values of the independent variables.
- The goal is to minimize the difference between predicted and actual values.

$$y = f(x) = \sum_j w_j x_j + b$$

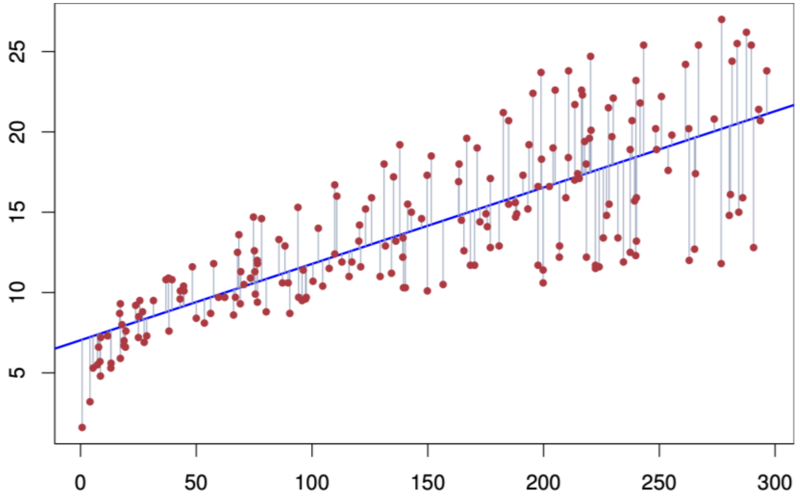


Image Source: https://web.stanford.edu/class/stats202/notes/Linear-regression/Simple-linear-regression.html

Implementing Linear Regression

- Direct Algebraic Method
- Gradient Descent Method

Direct Algebraic Method

Prediction:

$$y = f(x) = \sum_j w_j x_j + b \ \rightarrow (expand)$$

$$\rightarrow \begin{bmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1m} \\ x_{21} & x_{22} & \dots & x_{2m} \\ \dots & .. & \dots & \dots \\ x_{n1} & x_{n2} & \dots & x_{nm} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \dots \\ w_m \end{bmatrix} \rightarrow \vec{y} = X\vec{w}$$

Direct Algebraic Method

Prediction:

$$y = f(x) = \sum_j w_j x_j + b \;\rightarrow (expand)$$

$$\rightarrow \begin{bmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1m} \\ x_{21} & x_{22} & \dots & x_{2m} \\ \dots & .. & \dots & \dots \\ x_{n1} & x_{n2} & \dots & x_{nm} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \dots \\ w_m \end{bmatrix} \rightarrow \vec{y} = X\vec{w}$$

$$\rightarrow \begin{bmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1m} & 1 \\ x_{21} & x_{22} & \dots & x_{2m} & 1 \\ \dots & .. & \dots & \dots & 1 \\ x_{n1} & x_{n2} & \dots & x_{nm} & 1 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \dots \\ w_m \\ b \end{bmatrix} \rightarrow \vec{y} = X\vec{w}$$

Direct Algebraic Method

Training:

$$\vec{w} = (X^T X)^{-1} X^T t$$

Direct Algebraic Method Pros & Cons

- Finds optimal weights and bias directly.
- Memory-intensive for large datasets.

$$\vec{w} = (X^T X)^{-1} X^T t$$

Gradient Descent Method

Purpose

Used to find the minimum arguments (parameters) of a function by iteratively updating them in the direction that reduces the function's value.

Gradient Descent: Steps

1 - Calculate the Gradient:

Compute the gradient of the function with respect to the parameters (This indicates the direction of the steepest ascent):

$$\nabla f(x) = \frac{\partial f(x)}{\partial x}$$

2 - Update Parameters:

Adjust the parameters in the opposite direction of the gradient to minimize the function:

$$x = \ x - \ \eta \, \nabla f(x)$$

where η is the learning rate.

Gradient Descent Method

Minimizing squared error loss function:

$$L = \frac{1}{2}(y-t)^2 \rightarrow \frac{1}{2n}\left\|\vec{y}-\vec{t}\right\|^2$$

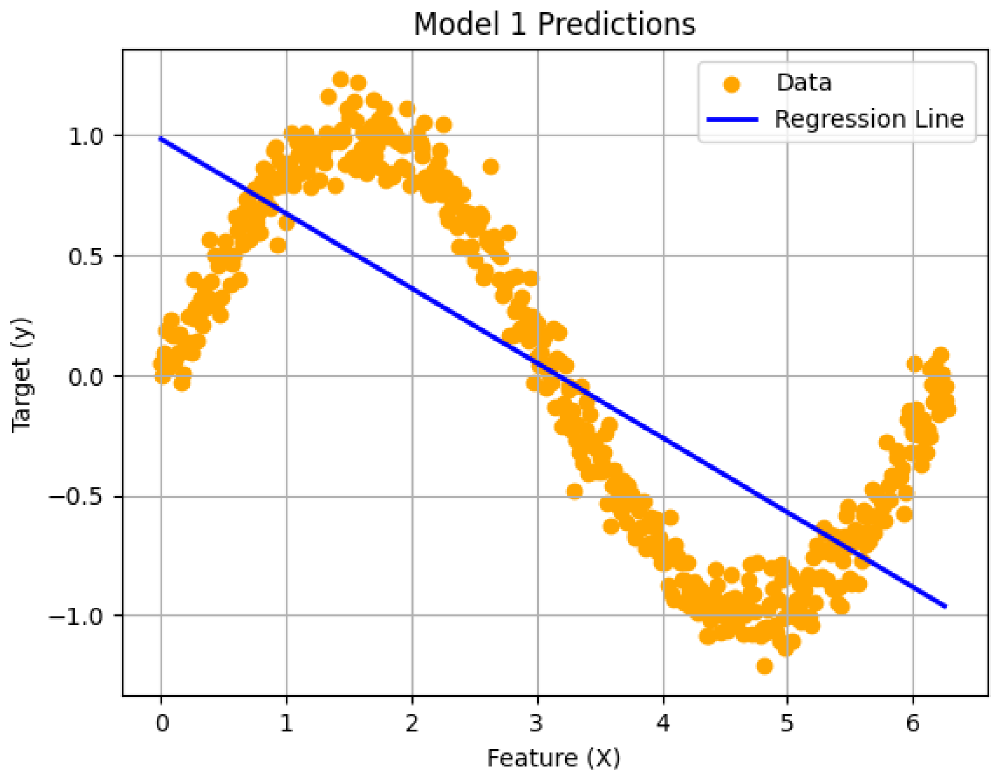
$$\nabla_w L = \frac{1}{n}X^T(X\vec{w}-\vec{y})$$

$$\vec{w} = \vec{w} - \eta\nabla_w L$$

Gradient Descent Pros & Cons

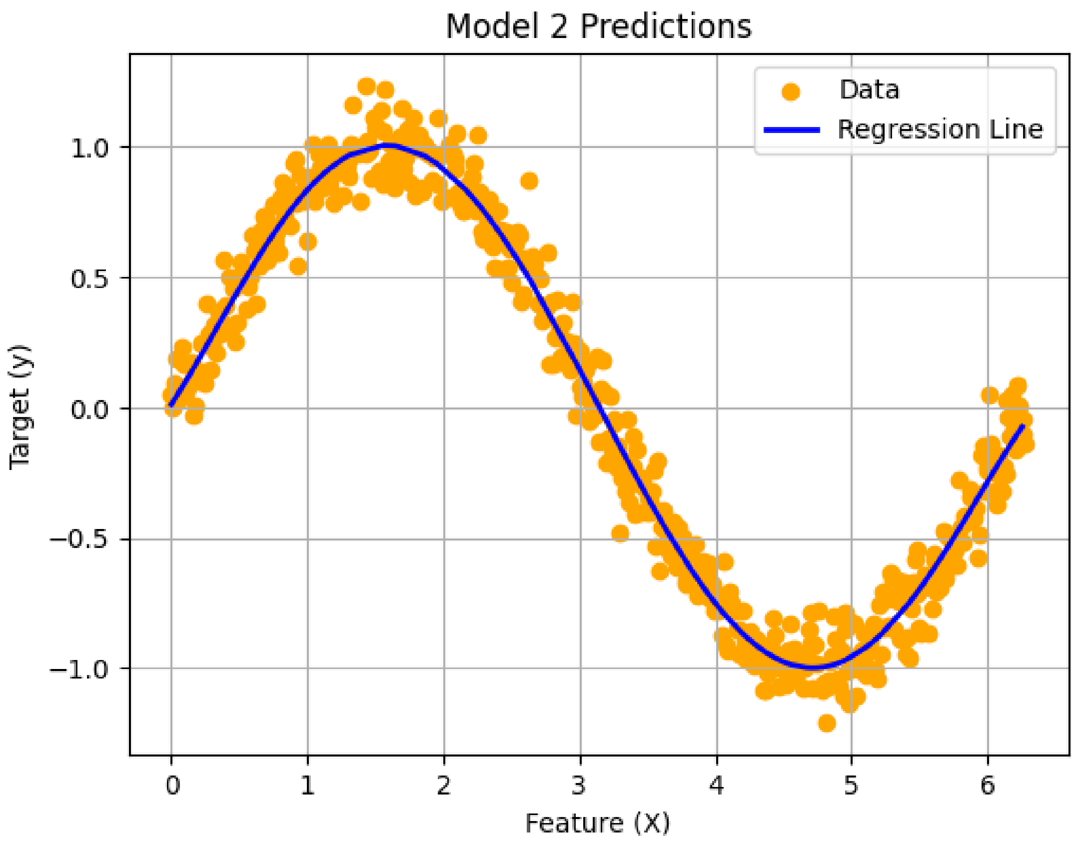
- By using mini-batch or stochastic gradient descent we decrease the amount of memory required to train our model
- However unlike algebraic method gradient descent does not guarantee converging into best optimal point and we might end up in local optima.
- We also have to deal with hyperparameters such as learning rate & batch size.

Basis Functions



MSE = 0.23783243185682448

Basis Functions



MSE = 0.009871901235453064

Basis Functions

- Polynomial:

$$\phi_j(x) = x^j$$

- Gaussian Basis Functions:

$$\phi_j(x) = \exp\left\{-\frac{(x-\mu_j)^2}{2s^2}\right\}$$

- Sigmoidal Basis Functions:

$$\phi_j(x) = \sigma\left(\frac{x-\mu_j}{s}\right)$$

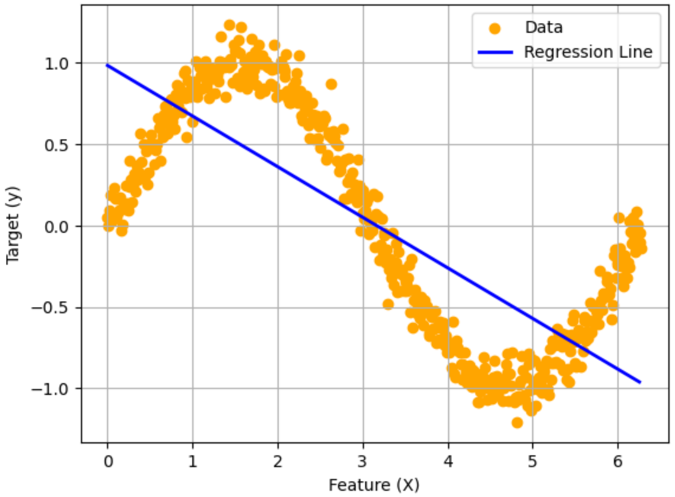
Polynomial Basis Functions

$$\begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix} \rightarrow \Phi_2\left(\begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix}\right) \rightarrow \begin{bmatrix} 1 & x_1^1 & x_1^2 \\ 1 & x_2^1 & x_2^2 \\ 1 & \dots & \dots \\ 1 & x_n^1 & x_n^2 \end{bmatrix}$$

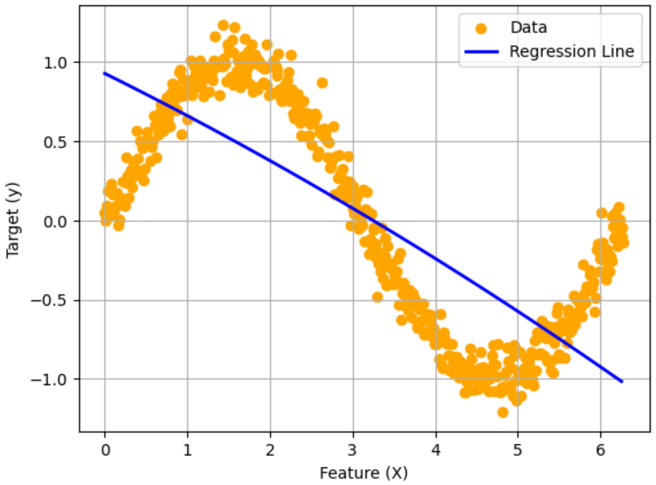
Polynomial Basis Functions

$$\begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix} \rightarrow \Phi_m\left(\begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix}\right) \rightarrow \begin{bmatrix} 1 & x_1^1 & x_1^2 & \dots & x_1^m \\ 1 & x_2^1 & x_2^2 & \dots & x_2^m \\ 1 & \dots & \dots & \dots & \dots \\ 1 & x_n^1 & x_n^2 & \dots & x_n^m \end{bmatrix}$$

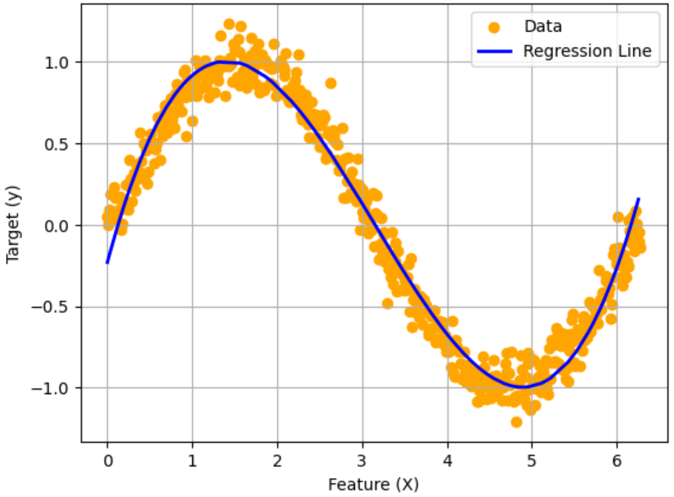
Polynomial Basis Functions



1st Degree Polynomial



2nd Degree Polynomial



3rd Degree Polynomial

Perquisites

- Python
- Object oriented programming
- Numpy
- Matplotlib