Machine Learning - Homework 2

- Linear Regression -

ChangYoon Lee

Dankook University, Korea Republic
32183641@gmail.com

Abstract. Linear regression is one of the simplest prediction models in machine learning algorithms. The following model fits the best model by changing the coefficients and the intercept variables of the linear model. Two main approaches are used to implement the linear model, and these methods are called gradient descent and least square. Also, the error of the model is evaluated by the error function, which is the mean squared error (MSE). In this paper, we will explain the concepts that are used in the linear regression model, and then present the implemented code for tasks 1 and task 2. Task 1 represents the gradient descent method, and task 2 represents the least square method. After explaining the implemented codes, we will compare the generated prediction models with the built-in libraries.

Keywords: Linear Regression, Gradient Descent, Normal Equation, Least Square, Mean Squared Error, MSE

1 Introduction

Linear regression is used to predict the value of a variable based on the value of another variable. The variable that we want to predict is called the dependent variable. The variable that we are using to predict the other variables are called the independent variable (About linear regression).

In this form of operation, independent variables are chosen that best predict the value of the dependent variable, based on the coefficients of the linear equation. It minimizes discrepancies between the predicted and actual output values by fitting a straight line or surface. To perform the linear regression, we use two main concepts used in building the linear regression model: gradient descent and least square. Also, the error of the prediction by using the following prediction model, which is called cost, is calculated by using the mean squared error (MSE) formula.

In this paper, we will discuss the basic concepts that are used to implement the linear regression model: gradient descent and least square. Then, we will discuss how we evaluate the performance of the prediction model by evaluating the error: mean squared error (MSE). After discussing all the concepts, we will present the implemented python code for task 1 and task 2, which represents each model that applies gradient descent and the least square method of the linear regression model. In the result, we will show the prediction graph according to the given data set and the coefficients for the predicted model. Also, there will be comparisons between the built-in linear regression models and the implemented models.

2 Concepts

2.1 Linear Regression

Linear regression is one of the machine learning models for predicting the real-valued label. It is supervised learning, which finds the label of the other data based on the prediction model that is generated by using the given labels or answers.

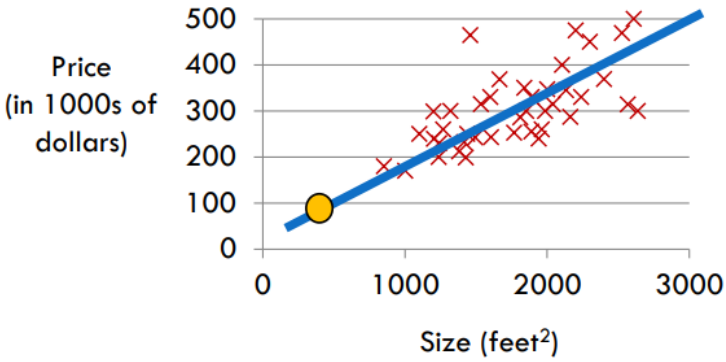


Figure 1 - Housing Price Prediction (Lee, *[Slide] 02. Linear Regression*)

As we mentioned, the linear regression problem deals with predicting the real-valued output. Figure 1 shows one example of the linear regression model that predicts the housing price based on the size of the house. We can achieve this goal by following the three steps: build a hypothesis, train the model (Optimization), and make predictions with the trained model.

- **Build hypothesis**: we can set the linear relationship between the given input and given output.
- **Train the model**: we train the model and find the best coefficients and intercepts that involve the following linear equation that is used in the model.
- **Make predictions with the trained model**: we predict the output for some input with the trained model.

Then, how could we determine whether the chosen coefficients and intercepts are the best parameters for the model? We evaluate the error, which is the gap between the prediction results and actual values, to find out the performance of the resulting model. The higher error means that the model is not describing the current situation well (Lee, [Slide] 02. Linear Regression). This error is also equally treated as cost and calculated by the cost function.

$$J(\theta_0, \theta_1) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

Figure 2 - Mean Squared Error (MSE) Function (Lee, *[Slide] 02. Linear Regression*)

In the linear regression model, the cost function is usually calculated through the mean squared error (MSE). Figure 1 presents the following cost function. The main goal of implementing the linear regression model is to find the best coefficients that minimize the value of the cost function. The approaches to finding these coefficients end up in the method, such as gradient descent and the least square method.

2.2 Least Square

The least square method, which uses a normal equation, is one of the ways to implement the linear regression model. It sets a goal to minimize the sum of the error squared by using the formula of matrix operations.

$$\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)})-y^{(i)}) = \frac{1}{m}\sum_{i=1}^{m}\left(\theta_0+\theta_1 x^{(i)}-y^{(i)}\right) = 0$$
$$\theta_0 = \frac{\sum_{i=1}^{m}\left(y^{(i)}-\theta_1 x^{(i)}\right)}{m}$$

Figure 3 - Result of Partial Derivative of Theta 0 (Lee, *[Slide] 02. Linear Regression*)

$$\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)})-y^{(i)})x^{(i)} = \frac{1}{m}\sum_{i=1}^{m}\left(\theta_0 x^{(i)}+\theta_1 {x^{(i)}}^2 - y^{(i)}x^{(i)}\right) = 0$$
$$\theta_1 = \frac{\sum_{i=1}^{m}\left(y^{(i)}-\theta_0\right)x^{(i)}}{\sum_{i=1}^{m}{x^{(i)}}^2}$$

Figure 4 - Result of Partial Derivative of Theta 1 (Lee, *[Slide] 02. Linear Regression*)

To get the following matrix operation formula, we first get the partial derivatives of the theta 0 and theta 1 from the cost function which is presented in Figure 2. By setting each derivative to 0 and solving the equation for each parameter, we can get the variables for the linear regression model.

$$\begin{bmatrix}1 & a^{(0)}\\ 1 & a^{(1)}\\ 1 & a^{(2)}\end{bmatrix}\begin{bmatrix}x_0\\ x_1\end{bmatrix} = \begin{bmatrix}b^{(0)}\\ b^{(1)}\\ b^{(2)}\end{bmatrix} \quad Ax = b$$

Figure 5 - Equation Form of the Linear Model (Lee, *[Slide] 02. Linear Regression*)

Now, let's normalize this formula into the matrix operation form by using the normal equation, which is presented in linear algebra. Figure 5 presents the equation form of the linear model based on the coefficients of the function. To get the coefficients that minimize the following equation, we use the gradient and the Hessian.

$$\frac{\partial f(x)}{\partial x_k} = \frac{\partial}{\partial x_k} \sum_{i=1}^{n} b_i x_i = b_k.$$

Figure 6 - Gradients of the Linear Function (Lee, *[Slide] 02. Linear Regression*)

$$\begin{aligned}
\frac{\partial f(x)}{\partial x_k} &= \frac{\partial}{\partial x_k} \sum_{i=1}^{n} \sum_{j=1}^{n} A_{ij} x_i x_j \\
&= \frac{\partial}{\partial x_k} \left[\sum_{i \neq k} \sum_{j \neq k} A_{ij} x_i x_j + \sum_{i \neq k} A_{ik} x_i x_k + \sum_{j \neq k} A_{kj} x_k x_j + A_{kk} x_k^2 \right] \\
&= \sum_{i \neq k} A_{ik} x_i + \sum_{j \neq k} A_{kj} x_j + 2 A_{kk} x_k \\
&= \sum_{i=1}^{n} A_{ik} x_i + \sum_{j=1}^{n} A_{kj} x_j = 2 \sum_{i=1}^{n} A_{ki} x_i,
\end{aligned}$$

Figure 7 - Gradients of Quadratic Functions (Lee, *[Slide] 02. Linear Regression*)

$$\frac{\partial^2 f(x)}{\partial x_k \partial x_\ell} = \frac{\partial}{\partial x_k} \left[\frac{\partial f(x)}{\partial x_\ell} \right] = \frac{\partial}{\partial x_k} \left[2 \sum_{i=1}^{n} A_{\ell i} x_i \right] = 2 A_{\ell k} = 2 A_{k \ell}.$$

Figure 8 - Hessian of Quadratic Functions (Lee, *[Slide] 02. Linear Regression*)

According to the properties of gradient and hessian, we can get the result of the derivatives from the cost function which are presented in Figures 6, 7, and 8.

$$\begin{aligned}
&\|Ax - b\|_2^2 = (Ax - b)^T (Ax - b) = x^T A^T A x - 2 b^T A x + b^T b \\
&\nabla_x (x^T A^T A x - 2 b^T A x + b^T b) = \nabla_x x^T A^T A x - \nabla_x 2 b^T A x + \nabla_x b^T b \\
&\qquad = 2 A^T A x - 2 A^T b \\
&x = (A^T A)^{-1} A^T b
\end{aligned}$$

Figure 9 - Normal Equation of the Linear Model (Lee, *[Slide] 02. Linear Regression*)

In short, by applying the following results of derivatives, when we process the derivative operation measured by the Euclidean norm, we can get the normal equation of x for the following linear regression model.

The least square method is a powerful and simple method to get the coefficients of the linear regression model. By simply processing the matrix operations, we can get the best coefficients and intercepts for the following model. However, the least square method is not always available to get the prediction model. If the matrix of the given data set is not invertible, such situations where redundant features are in the matrix, or the size of the rows is smaller than the size of the columns, we cannot apply the least square method to implement the linear regression model.

2.3 Gradient Descent

The limitation of the least square method is that it cannot be applied when the matrix of a given data set is not invertible. In this case, we must use the gradient descent method to implement the linear regression model. The gradient descent method can be applied to any objective, not just for the squared error (Lee, *[Slide] 02. Linear Regression*).

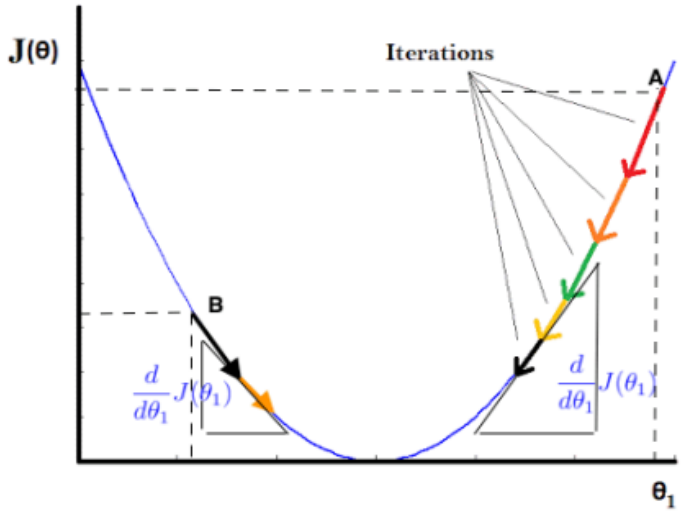


Figure 10 - Visualization of Gradient Descent (Lee, *[Slide] 02. Linear Regression*)

The gradient descent method is based on the iterative method. It first starts with random coefficients which are in some boundary. Then, it keeps changing the coefficients and operates the loops until it gets closer to the optimum point, where the error is minimized. Figure 10 shows the iterative loop of how the gradient descent operation is processed. In applying the gradient descent method, we must consider some points: a sign of the gradient, types of the batch, learning rate, and local minimum.

repeat until convergence {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \quad (\text{for } j = 0 \text{ and } j = 1)$$

}

Correct: Simultaneous update	Incorrect:
$\text{temp0} := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$	$\text{temp0} := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$
$\text{temp1} := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$	$\theta_0 := \text{temp0}$
$\theta_0 := \text{temp0}$	$\text{temp1} := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$
$\theta_1 := \text{temp1}$	$\theta_1 := \text{temp1}$

Figure 11 - Gradient Descent Algorithm (Lee, *[Slide] 02. Linear Regression*)

The first consideration in the gradient descent method is that we must be careful of the sign of the gradients. If the gradient is positive, we must move to the left to reach the optimal point. In contrast, if the gradient is negative, we must move to the right to reach the optimal point. In the aspect of the formula, these two considerations are the same. Therefore, we can present the following convergence in Figure 11. One important thing is that all the parameters should be updated simultaneously. We can check from Figure 11, that if the parameters are not simultaneously updated, the updated parameter can affect the result of the other parameter.

$$
\begin{aligned}
&\text{repeat until convergence } \{ \\
&\quad \theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) \\
&\quad \theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) \cdot x^{(i)} \\
&\}
\end{aligned}
$$

Figure 12 - Gradient Descent Algorithm for each Theta (Lee, *[Slide] 02. Linear Regression*)

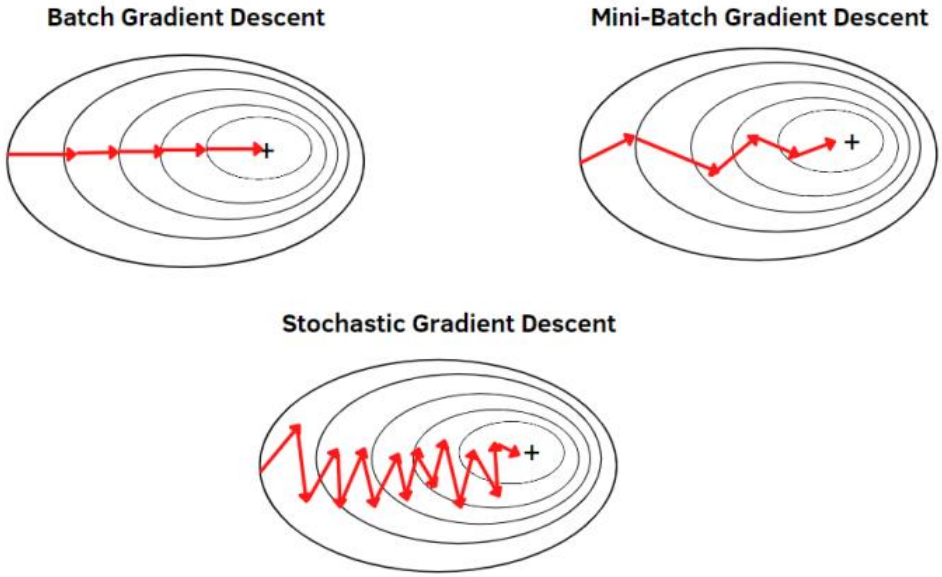


Figure 13 - Gradient Descent Operations due to the Batch Size (Lee, *[Slide] 02. Linear Regression*)

The second consideration is the batch size, which also means the data samples that are used in each iteration. In Figure 12, there is a variable m, which presents the size of the batch. If the batch size is small, which is called batch GD, the gradient will go in the correct direction, but the processing speed will be slow and can occur at the local minimum. However, if the batch size is big, which is called stochastic GD (SGD), each iteration will perform a broader search, but the gradient will go in the wrong direction. Therefore, setting the batch to some number between them, which is called mini-batch GD, is most preferred for the model. Figure 13 shows the visualization of the gradient descent operation due to the different batch sizes.

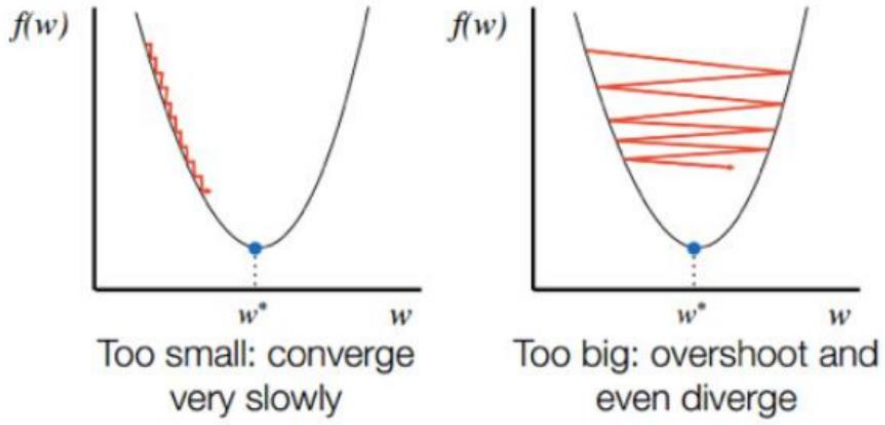


Figure 14 - Difference in Convergence due to the Learning Rate (Lee, *[Slide] 02. Linear Regression*)

The third consideration is the learning rate. If the learning rate of the model is too small, the convergence of the gradient will be very slow. Meanwhile, if the learning rate of the model is too big, the gradient will overshoot and in the worst case, it will diverge. As a result, choosing a proper learning rate is hard. Even if the learning rate is determined dynamically, as the model approaches the minimum, the gradient will automatically take the smaller step. Therefore, we must try with the small one and make sure the algorithm works correctly, and after that, we should increase it gradually (Lee, *[Slide] 02. Linear Regression*). Figure 14 shows the visualization of each presented case.

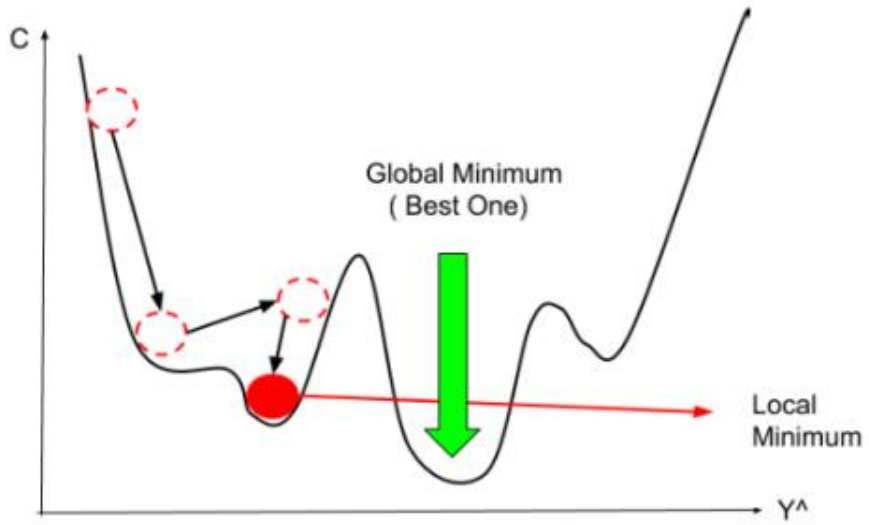
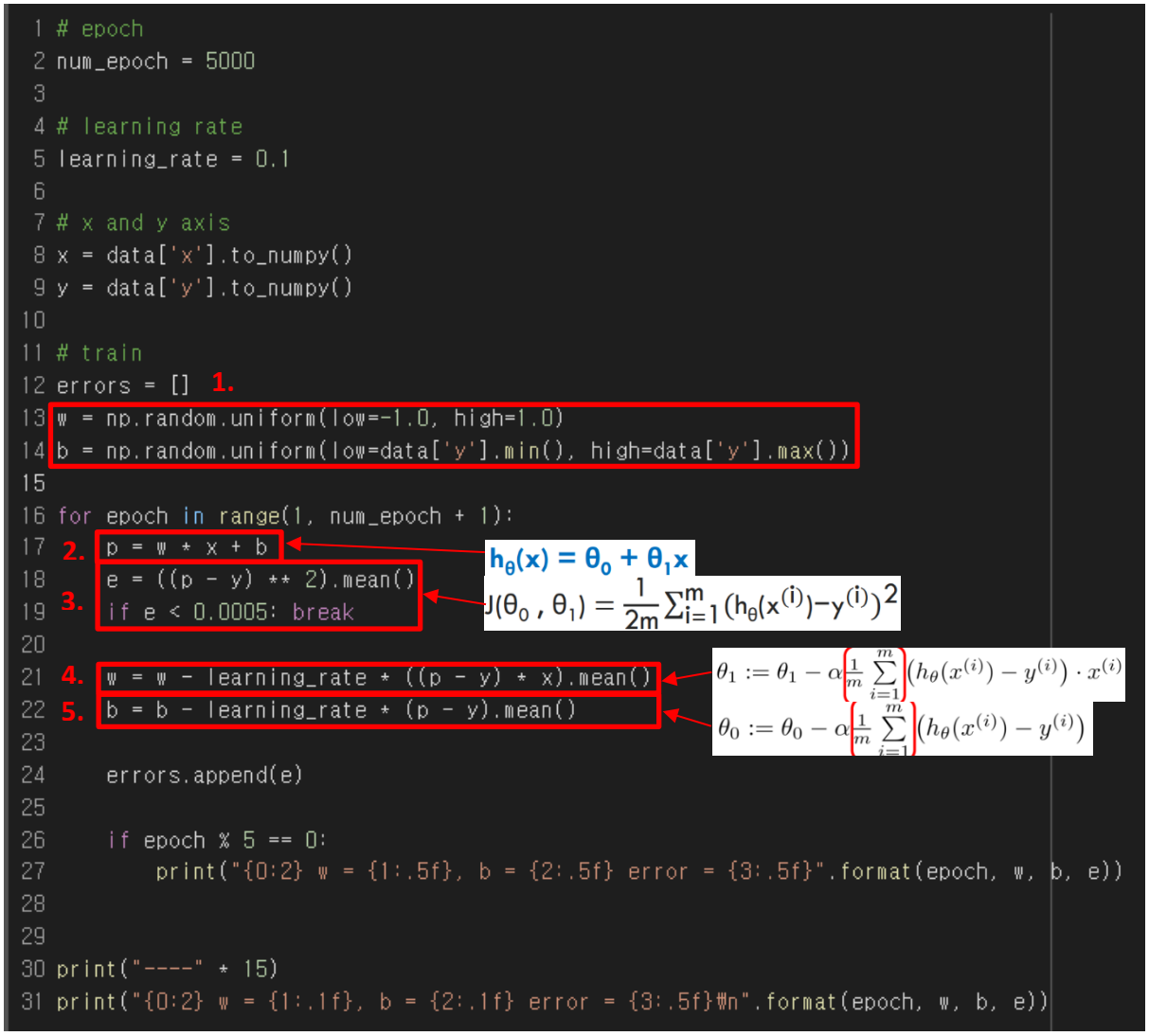


Figure 15 - Local Minimum (Lee, *[Slide] 02. Linear Regression*)

The last consideration is the local minimum. The local minimum becomes a problem when the gradient recognizes some minimum point is the optimal point, which is not, and stops the operation so that it cannot reach the global minimum. This problem is affected by the starting point of the operation, too small a learning rate, and the batch size. Therefore, by setting the right direction for the operation, proper step size, and using SGD, we can resolve the following consideration.

3 Implementation

3.1 Task 1




```
# epoch
num_epoch = 5000

# learning rate
learning_rate = 0.1

# x and y axis
x = data['x'].to_numpy()
y = data['y'].to_numpy()

# train
errors = []
w = np.random.uniform(low=-1.0, high=1.0)
b = np.random.uniform(low=data['y'].min(), high=data['y'].max())

for epoch in range(1, num_epoch + 1):
    p = w * x + b
    e = ((p - y) ** 2).mean()
    if e < 0.0005: break

    w = w - learning_rate * ((p - y) * x).mean()
    b = b - learning_rate * (p - y).mean()

    errors.append(e)

    if epoch % 5 == 0:
        print("{0:2} w = {1:.5f}, b = {2:.5f} error = {3:.5f}".format(epoch, w, b, e))


print("----" * 15)
print("{0:2} w = {1:.1f}, b = {2:.1f} error = {3:.5f}₩n".format(epoch, w, b, e))
```

Figure 16 - Implementation of the Gradient Descent Method

Figure 16 shows the implemented code for task 1, which presents the linear regression model that involves the gradient descent method and means squared error (MSE). The following code operates in 5 steps, and each step is presented in Figure 16.

1. First, we set the initial random bounded weight and bias for the linear regression model, so that we can initialize the optimal point in the error function.

2. Then, it defines the prediction function for the linear regression model.

3. It calculates the error, and if the error is smaller than 0.0005, it stops the operation

4. In steps 4 and 5, it calculates the next weight and bias according to the formula presented in Figure 16.

Each step of the code follows the formula operation which is presented in section 2.3. By using the following code, we can predict with the linear regression model for the given data set by applying the gradient descent method.

3.2 Task 2

```python
import numpy as np


# real data of x and y axis
x = data['x'].to_numpy()
y = data['y'].to_numpy()
x1 = x.reshape((len(x), 1))

# x2 -> x power of 2
# bias_x -> bias of x
x2 = np.power(x1, 2)
x_bias = np.ones((len(x1), 1))

# marix_x -> x axis with bias which is in the matrix form
matrix_x = np.append(x_bias, x1, axis=1)
matrix_x = np.append(matrix_x, x2, axis=1)

# tanspose_x -> transposed matrix of the following x matrix
# tx_dot_mx -> dot operation between transposed matrix and original matrix
transpose_x = np.transpose(matrix_x)
tx_dot_mx = transpose_x.dot(matrix_x)

# inverse_x -> inveres matrix of the following x matrix
inverse_x = np.linalg.inv(tx_dot_mx)

# mx_dot_y -> dot operation between x matrix and y matrix
mx_dot_y = transpose_x.dot(y)

# theta -> coefficients of the prediction graph
theta = inverse_x.dot(mx_dot_y)
```

1. $\begin{bmatrix} 1 & a_2^0 & a_1^0 \\ 1 & a_2^1 & a_1^1 \\ 1 & a_2^2 & a_1^2 \end{bmatrix} \begin{bmatrix} X_0 \\ X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} b^0 \\ b^1 \\ b^2 \end{bmatrix}$

2. $x = (A^T A)^{-1} A^T b$

Figure 17 - Implementation of the Least Square Method by using normal equation

Figure 17 shows the implemented code for task 2, which presents the linear regression model that involves the least square method. The following code operates in 2 steps, and each step is presented in Figure 17.

1. First, we define the matrix for the prediction formula. At this point, we have a concern about the formula that we are going to use as the prediction model. The model that we are going to use in task 2 is the polynomial formula. Therefore, not only implemented the 2 by 3 matrix, but we also generated a 3 by 3 matrix by appending multiple coefficient vectors.

2. Then, by processing the operation, which is presented in number 2 of Figure 17, we generated the coefficient and intercept matrix as a result.

Each step of the code follows the formula operation which is presented in section 2.2. By using the following code, we can predict with the linear regression model for the given data set by applying the least square method.

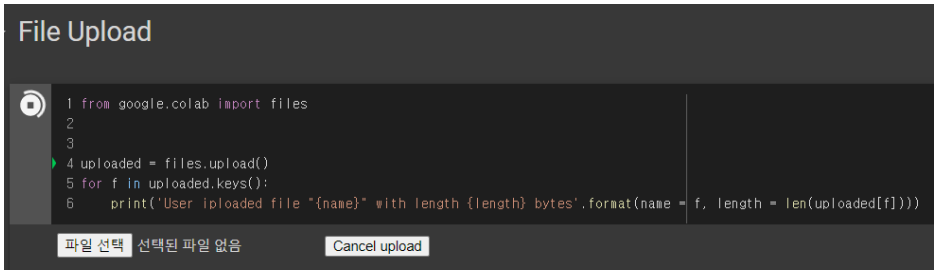
4 Build Environment

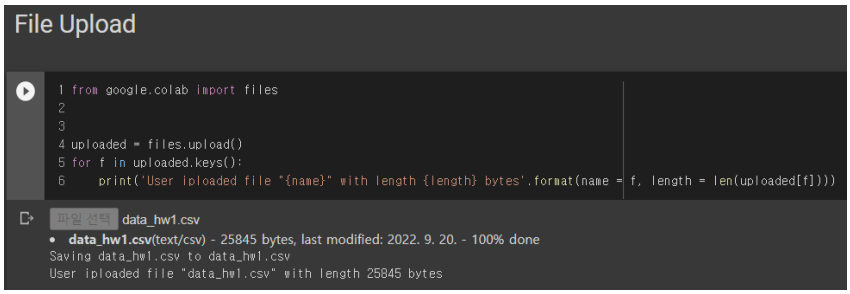
The following build environments are required to execute the implemented code for task 1 and task 2.

Building Environment:

- The environment that can open the ipython notebook: COLAB, Jupyter Notebook
- Each cell in the notebook should be executed sequentially.

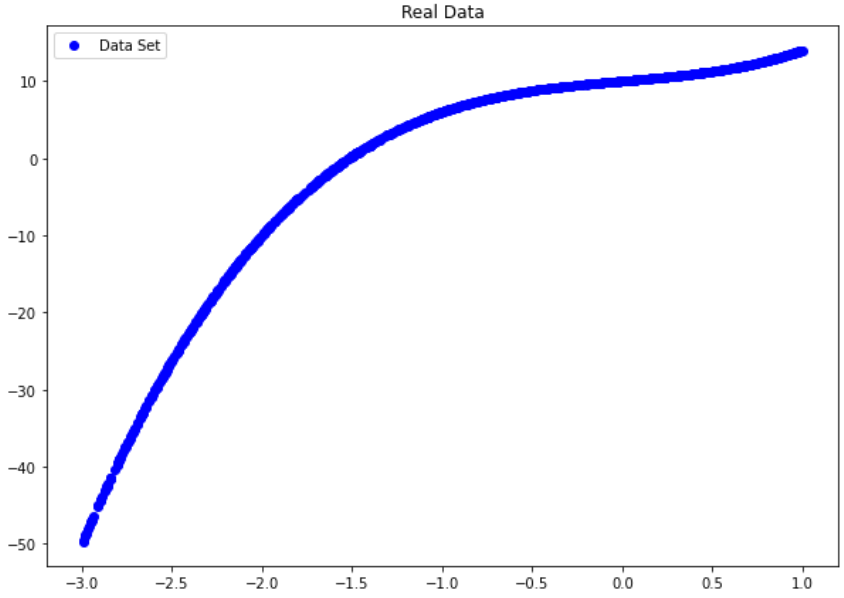
File Upload: Before cell execution, the given data set must be uploaded first.





5 Results

- Visualization of the original given data set of x and y

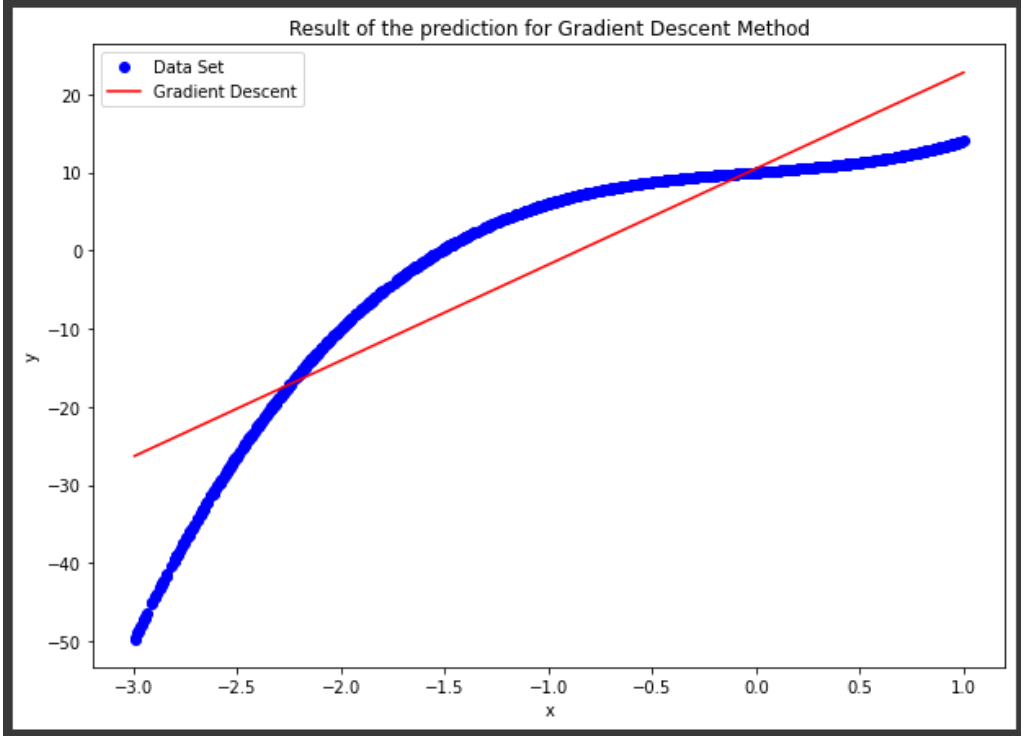


- The train result of the linear regression model of task 1

```
4850 w = 12.28807, b = 10.49321 error = 51.71361
4855 w = 12.28807, b = 10.49321 error = 51.71361
4860 w = 12.28807, b = 10.49321 error = 51.71361
4865 w = 12.28807, b = 10.49321 error = 51.71361
4870 w = 12.28807, b = 10.49321 error = 51.71361
4875 w = 12.28807, b = 10.49321 error = 51.71361
4880 w = 12.28807, b = 10.49321 error = 51.71361
4885 w = 12.28807, b = 10.49321 error = 51.71361
4890 w = 12.28807, b = 10.49321 error = 51.71361
4895 w = 12.28807, b = 10.49321 error = 51.71361
4900 w = 12.28807, b = 10.49321 error = 51.71361
4905 w = 12.28807, b = 10.49321 error = 51.71361
4910 w = 12.28807, b = 10.49321 error = 51.71361
4915 w = 12.28807, b = 10.49321 error = 51.71361
4920 w = 12.28807, b = 10.49321 error = 51.71361
4925 w = 12.28807, b = 10.49321 error = 51.71361
4930 w = 12.28807, b = 10.49321 error = 51.71361
4935 w = 12.28807, b = 10.49321 error = 51.71361
4940 w = 12.28807, b = 10.49321 error = 51.71361
4945 w = 12.28807, b = 10.49321 error = 51.71361
4950 w = 12.28807, b = 10.49321 error = 51.71361
4955 w = 12.28807, b = 10.49321 error = 51.71361
4960 w = 12.28807, b = 10.49321 error = 51.71361
4965 w = 12.28807, b = 10.49321 error = 51.71361
4970 w = 12.28807, b = 10.49321 error = 51.71361
4975 w = 12.28807, b = 10.49321 error = 51.71361
4980 w = 12.28807, b = 10.49321 error = 51.71361
4985 w = 12.28807, b = 10.49321 error = 51.71361
4990 w = 12.28807, b = 10.49321 error = 51.71361
4995 w = 12.28807, b = 10.49321 error = 51.71361
5000 w = 12.28807, b = 10.49321 error = 51.71361
------------------------------------------------------------
5000 w = 12.3, b = 10.5 error = 51.71361
```

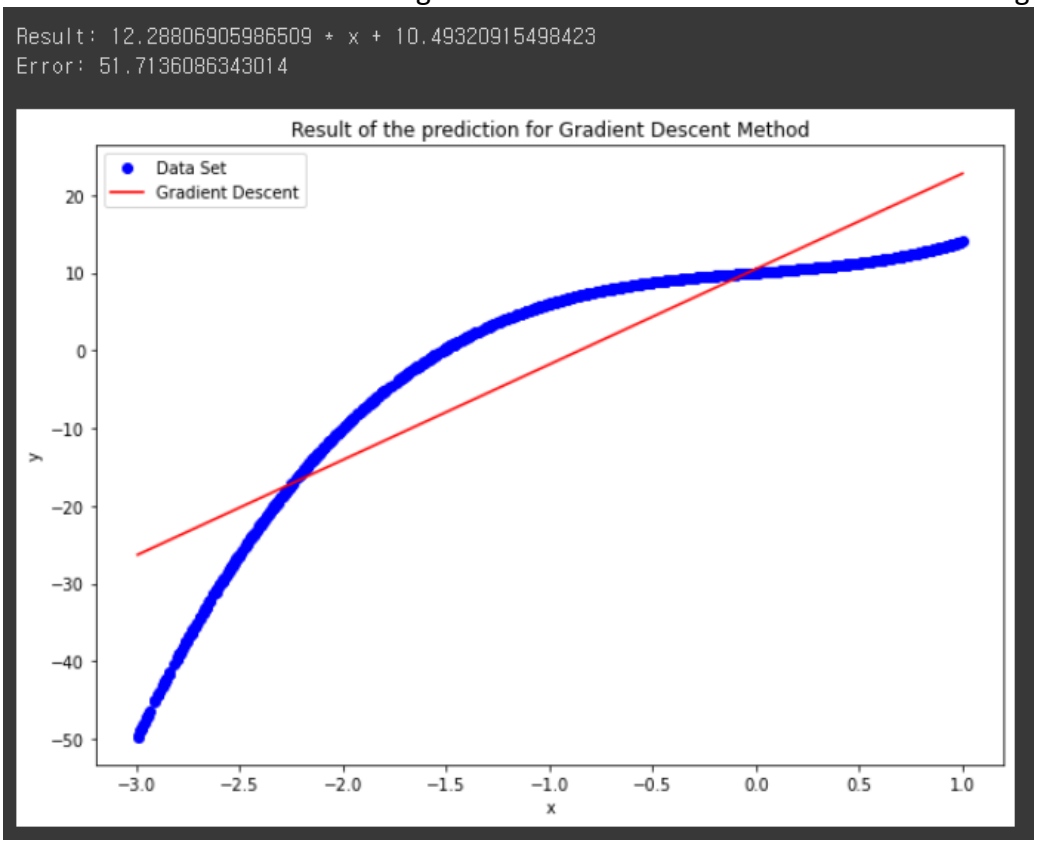
- Visualization of the linear regression model of task 1 with the original data set

```
Result: 12.288069059865078 * x + 10.493209154984212
Error: 51.713608634301416
```



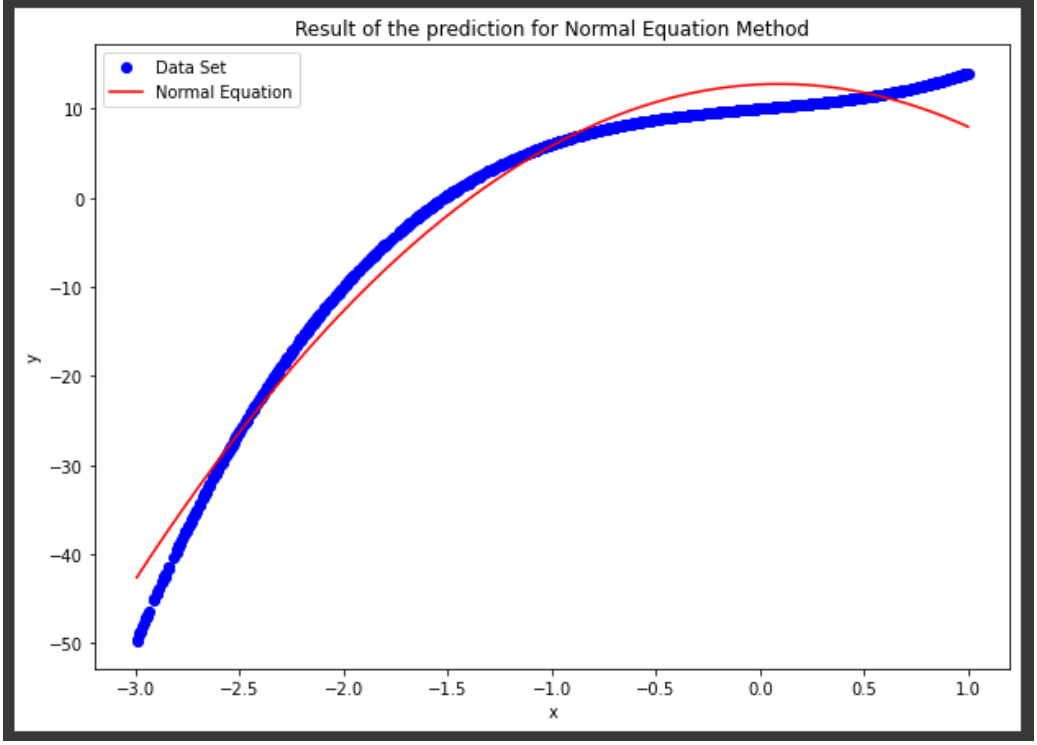
- Visualization of the linear regression model of the Scikit-Learn with the original data set

```
Result: 12.28806905986509 * x + 10.49320915498423
Error: 51.7136086343014
```



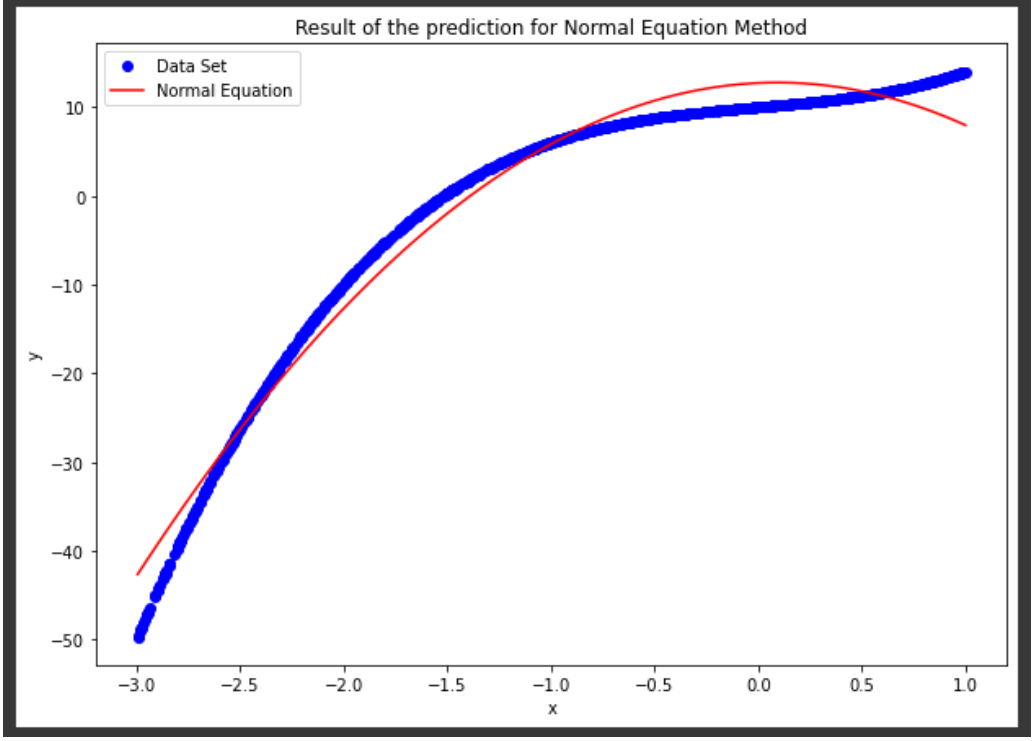
- Visualization of the linear regression model of task 2 with the original data set

```
Result: -5.82074893158833 * x**2 + 1.0570819678946108 * x + 12.704822547462781
Error: 5.496605324334822
```



- Visualization of the linear regression model of the Scikit-Learn with the original data set

```
Result: -5.8207489315883265 * x**2 + 1.0570819678946002 * x + 12.704822547462763
Error: 5.496605324334822
```



6 Evaluation

Figure 18 - Results of the Gradient Descent Method of Implemented and Built-In Code

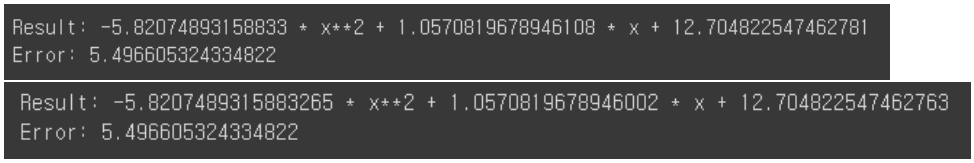


Figure 19 - Results of the Least Square Method of Implemented and Built-In Code

Figures 18 and 19 show the implemented linear regression model of task 1 and task 2, which are each implemented by our code and the sci-kit learn library. As we can see, the coefficients and the intercept are the same, which means that the implemented linear regression models are successfully generated. Also, the linear regression model of task 2 shows a better performance than the model of task 1. It means that as the number of coefficients increases in the linear regression model, the performance gets better. In short, we generated the fine-granted linear regression model, and the model of task 2, which is polynomial, shows better performance than the model of task 1, which is linear.

7 Conclusion

Linear regression fits a linear model with the coefficients to minimize the residual sum of the squares between the observed targets in the data set, and the targets predicted by the linear approximation (Sklearn.linear_model.linear regression). This model can be implemented by applying such methods: gradient descent and the least square method. Also, the performance of the linear regression model can be measured by the error function, which is the mean squared error (MSE).

In this paper, we have explained the concept of linear regression, gradient descent, least square, and mean square error (MSE). Then, we presented tasks 1 and task 2, which represented the method of gradient descent and the least square method. By using each implemented code, we generated each prediction model for each task. After all this process, we evaluated our prediction model and compared each model to the built-in linear regression model. As a result, the coefficients and the intercept between the prediction models are the same.

By understanding this paper, we can understand the basic concepts of the linear regression model and the formulas that are used to implement the following model: gradient descent, least square, and the mean square error (MSE). Also, we can understand the considerations that occur by applying the following concept, which is mainly about gradient descent.

Citations

[1] *About linear regression*. IBM. (n.d.). Retrieved September 20, 2022, from https://www.ibm.com/topics/linear-regression

[2] Lee, K. H. (n.d.). [Slide] 02. Linear Regression.

[3] *Sklearn.linear_model.linearregression*. scikit. (n.d.). Retrieved September 21, 2022, from https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html