Name : Suhas Raghavendra

ASU ID : 1233271600

Linear regression [7 pts]

In this homework, you will implement solution algorithms for linear regression.

Import libraries

Let's begin by importing some libraries.

In [1]:
```
print(__doc__)
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets
%matplotlib inline
```

```
Automatically created module for IPython interactive environment
```

Load dataset

Now, we are importing a dataset of diabetes. You can check the details on this dataset here: [https://scikit-learn.org/stable/datasets/toy_dataset.html#diabetes-dataset (https://scikit-learn.org/stable/datasets/toy_dataset.html#diabetes-dataset)](https://scikit-learn.org/stable/datasets/toy_dataset.html#diabetes-dataset).

The dataset consists of 442 observations with 10 attributes (X) that may affect the progression of diabetes (y). Ten baseline variables, age, sex, body mass index, average blood pressure, and six blood serum measurements were obtained for each of n = 442 diabetes patients, as well as the response of interest, a quantitative measure of disease progression one year after baseline.

In [2]:
```
# Load the diabetes dataset
diabetes_X, diabetes_y = datasets.load_diabetes(return_X_y=True)
print('The shape of the input features:',diabetes_X.shape)
print('The shape of the output varaible:',diabetes_y.shape)
```

```
The shape of the input features: (442, 10)
The shape of the output varaible: (442,)
```

We will choose just one attribute from the ten attributes as an input variable.

In [3]:
```
# Use only one feature
diabetes_X_one = diabetes_X[:, np.newaxis, 2]
print(diabetes_X_one.shape)
```

```
(442, 1)
```

Dataset split

Now, we split the dataset into two parts: training set and test set.

- training set: 422 samples
- test set: 20 samples

In [4]:
```
# Split the data into training/testing sets
diabetes_X_train = diabetes_X_one[:-20]
diabetes_X_test = diabetes_X_one[-20:]

# Split the targets into training/testing sets
diabetes_y_train = diabetes_y[:-20]
diabetes_y_test = diabetes_y[-20:]

print('Training input variable shape:', diabetes_X_train.shape)
print('Test input variable shape:', diabetes_X_test.shape)
```

```
Training input variable shape: (422, 1)
Test input variable shape: (20, 1)
```

Linear regression

Assume that we have a hypothesis

$$h_\theta(x) = \theta_0 + \theta_1 x.$$

Your tasks:

Implementation tasks:

- [2pts] implement {your own version} of the method of least-squares, compute and report θ_0 and θ_1 that minimize the residual sum of squares,)

$$\sum_{i=1}^{N} \frac{1}{2}(y^{(i)} - h_\theta(x^{(i)})^2$$

[IMPORTANT] Do not just call the least square function from libraries, for example, scipy.optimize.least_squares from scipy. Doing so will result in 0 point. Using helping functions such as numpy.linalg.inv is okay.

- [3pts] implement your own version of the gradient descent algorithm, compute and report θ_0 and θ_1 that minimize the mean squared error

$$\sum_{i=1}^{N} \frac{1}{N}(y^{(i)} - h_\theta(x^{(i)})^2$$

[NOTE] Notice that the loss function is mean-squared error. Implement the gradient descent update rule in a for loop. To check whether your computation is correct, consider using an API such as Scikit learn linearregression.

- [2pts] derive the analytical expression of the gradient if the loss is defined as

$$\sum_{i=1}^{N} \frac{1}{2}(y^{(i)} - h_\theta(x^{(i)})^2 + \frac{\lambda}{2}\|\theta\|_2^2,$$

  where $\theta = [\theta_0, \theta_1]^\intercal$

task 1

In [5]:
```
diabetes_X_b = np.c_[np.ones((len(diabetes_X_train), 1)), diabetes_X_train]
theta = np.linalg.inv(diabetes_X_b.T.dot(diabetes_X_b)).dot(diabetes_X_b.T)

print(f"Lease squares Theta 1 : {theta[1]}")
print(f"Least squares Theta 0 : {theta[0]}")
```

```
Lease squares Theta 1 : 938.2378612512634
Least squares Theta 0 : 152.91886182616173
```

MSE of Lease squares

In [6]:
```
X_b = np.c_[np.ones((diabetes_X_test.shape[0], 1)), diabetes_X_test]
y_pred_ls = X_b.dot(theta)
```

In [7]:
```
mse_ls = np.mean((y_pred_ls - diabetes_y_test) ** 2)
```

In [8]:
```
print(f"Mean Squared Error (Least Squares): {mse_ls}")
```

```
Mean Squared Error (Least Squares): 2548.072398725972
```

Task 2

In [9]:
```
learning_rate=0.1
n_iterations=10000

m = len(diabetes_y_train)
X_b = np.c_[np.ones((m, 1)), diabetes_X_train]
theta_gd = np.random.randn(2)
for iteration in range(n_iterations):
        gradients = 2/m * X_b.T.dot(X_b.dot(theta_gd) - diabetes_y_train)
        theta_gd -= learning_rate * gradients

intercept_gd, slope_gd = theta_gd
print(f"Gradient Descent Slope (m): {slope_gd}")
print(f"Gradient Descent Intercept (b): {intercept_gd}")
```

```
Gradient Descent Slope (m): 928.1305888847661
Gradient Descent Intercept (b): 152.92365245534137
```

mse of gradient descent

In [10]:
```
X_b = np.c_[np.ones((diabetes_X_test.shape[0], 1)), diabetes_X_test]
y_pred_ls = X_b.dot(theta_gd)
```

In [11]:
```
mse_gd = np.mean((y_pred_ls - diabetes_y_test) ** 2)
```

In [12]:
```
print(f"Mean Squared Error (Gradient Descent): {mse_gd}")
```

```
Mean Squared Error (Gradient Descent): 2559.7830831148212
```