KALINGA INSTITUTE OF INDUSTRIAL TECHNOLOGY (KIIT)

Deemed to be University U/S 3 of UGC Act, 1956

Name : Abhishek Mallick

Roll : 21051706

ML_CS-3

Machine Learning

Assignment 1 – Simple Linear Regression

Github

AIM: The aim of this assignment is to implement a simple linear regression model using gradient descent.

- Batch gradient descent with convergence criteria and averaging cost function.
- Cost function advantage comparison.
- Stochastic and mini-batch gradient descent implementation.

Dataset used:

Independent/Predictor Variable -
https://drive.google.com/file/d/1cFZHElm5ebn1OolgmrLOxw91T8am3S5C/view?usp=sharing

Dependent/Response Variable -

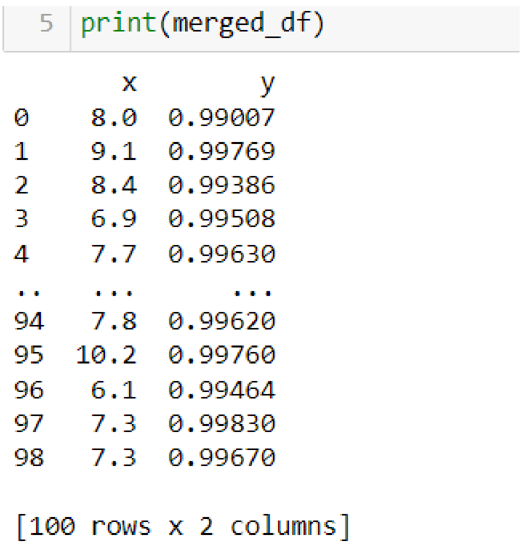
https://drive.google.com/file/d/1rY3oTHxa1FT3fdcjWuKKQ4DOXnU8AeQM/view?usp=sharing

Cost function used:

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

1. Use linear regression to fit a straight line to the given database. Set your learning rate to 0.5. What are the cost function value and learning parameters values after convergence? Also, mention the convergence criteria you used.

Merged dataframe representation:

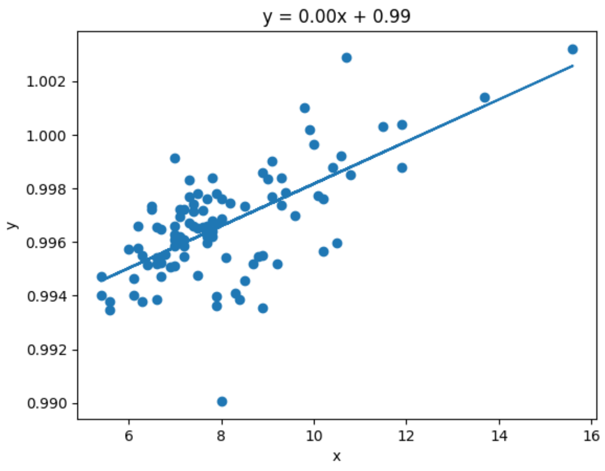


```
print(merged_df)
```

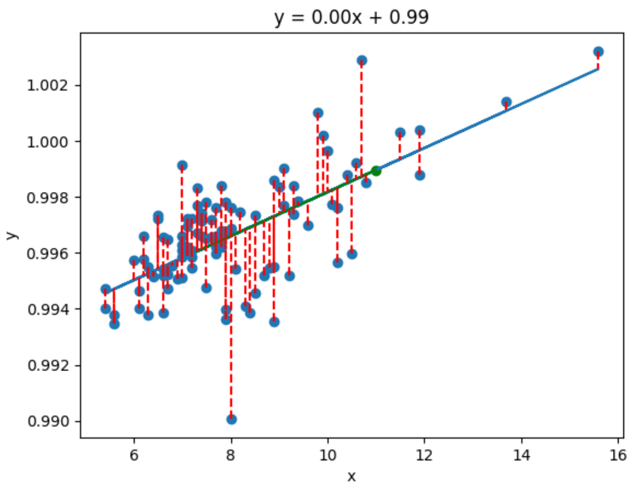
```
        x        y
0     8.0  0.99007
1     9.1  0.99769
2     8.4  0.99386
3     6.9  0.99508
4     7.7  0.99630
..    ...      ...
94    7.8  0.99620
95   10.2  0.99760
96    6.1  0.99464
97    7.3  0.99830
98    7.3  0.99670

[100 rows x 2 columns]
```

Regression Plot:



Regression Plot with errors:



Coefficient of Determination: 0.4381850455791929

Using Cost Function:

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

```
Epoch 0, Cost: x    0.0
y    0.0
dtype: float64, theta0: x    0.975739
y    0.975739
dtype: float64, theta1: x    0.686877
y    0.686877
dtype: float64
Gradient: theta0 = x    0.0
y    0.0
dtype: float64, theta1 = x    0.0
y    0.0
dtype: float64
Epoch 0, Cost: x    0.0
y    0.0
dtype: float64, theta0: x    0.975739
y    0.975739
dtype: float64, theta1: x    0.686877
y    0.686877
dtype: float64
Epoch 1, Cost: x    0.0
y    0.0
dtype: float64, theta0: x    0.975739
y    0.975739
dtype: float64, theta1: x    0.686877
y    0.686877
dtype: float64
Converged at epoch 1
```

```
Learning Rate: 0.5
Theta0: 0.975739
Theta1: 0.686877
Final Cost: 1.1821341459840557e-06
```

Using Convergence Criteria:

previous_cost - current_cost).any() < cost_threshold

2. The cost function that we are using in this assignment is different than the one we used in class. Can you think of the advantage of averaging the cost?

We used here ->

$J(\theta_0, \theta_1)$ = 1/2m ∑(i=1 to m) [($h\theta$(x(i)) - y(i))^2]

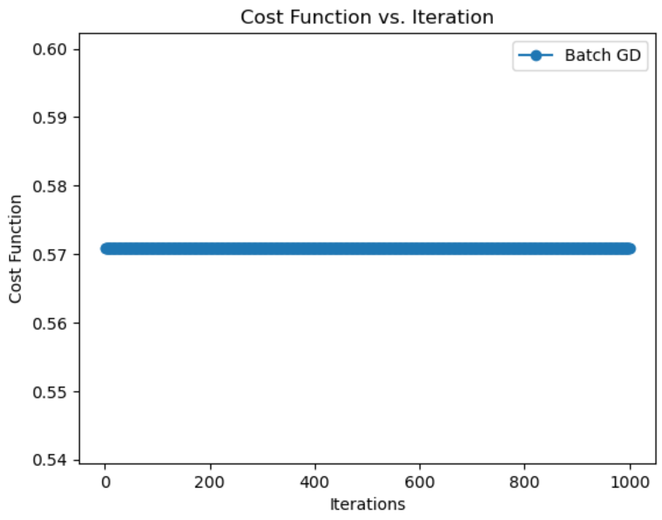
where:

- m is the number of data points
- $h\theta(x(i))$ is the predicted value for the ith data point using the model parameters θ_0 and θ_1
- $y(i)$ is the actual value for the ith data point

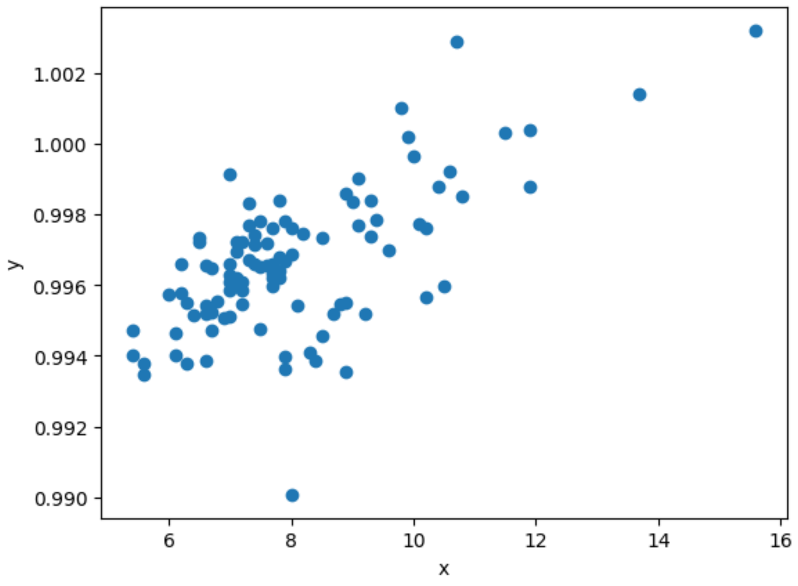
advantage of averaging the cost function ->

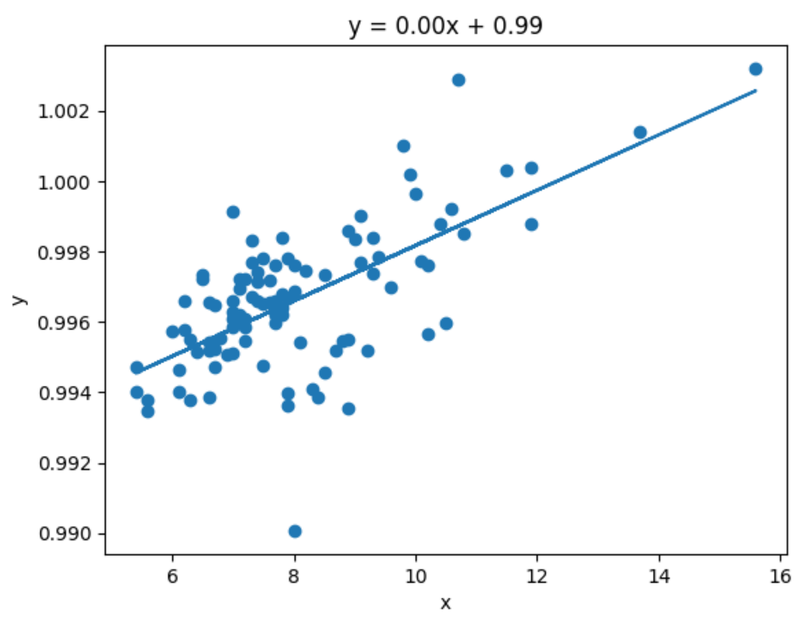
1. **Normalization:**
   - Averaging normalizes the cost function by the number of data points.
   - By averaging, we make them comparable across datasets.
2. **Gradient Stability:**
   - Averaging can lead to smoother gradients, which can be beneficial for optimization algorithms like gradient descent. Smoother gradients help the algorithm avoid getting stuck in local minima and converge to a better solution.
3. **Noise Reduction:**
   - Averaging can help reduce the impact of noise or outliers in the data. Individual data points that deviate significantly from the overall trend can have a large influence on the cost function without averaging.

function v/s iteration graph



4. Plot the given dataset on a graph and also print the straight line you obtained in question 1 to show how it fits the data.





5. Choose a suitable learning rate, then implement stochastic and min-batch gradient descent, plot the cost function against iteration, and observe how your cost function changes compared to batch gradient descent.

stochastic_gradient_descent logic:

```
def stochastic_gradient_descent(X, Y, learning_rate, iterations):
    m = 0
    b = 0
    cost_history = []

    for i in range(iterations):
        for j in range(len(Y)):
            # Update parameters using stochastic gradient descent
            m = m - learning_rate * dEdm(m, b, X[j], Y[j])
            b = b - learning_rate * dEdb(m, b, X[j], Y[j])

        # Calculate and store the cost
        cost = np.mean((m*X + b - Y)**2)
        cost_history.append(cost)

    return m, b, cost_history
```

Stochastic Gradient Descent - Coefficients:

Slope (m): 0.633266587828981

Intercept (b): 0.04594171242258456

mini_batch_gradient_descent logic:

```
def mini_batch_gradient_descent(X, Y, learning_rate, batch_size, iterations):
    m = 0
    b = 0
    cost_history = []

    for i in range(iterations):
        permutation = np.random.permutation(len(Y))
        X_shuffled = X[permutation]
        Y_shuffled = Y[permutation]

        for j in range(0, len(Y), batch_size):
            X_batch = X_shuffled[j:j+batch_size]
            Y_batch = Y_shuffled[j:j+batch_size]
            m = m - learning_rate * dEdm(m, b, X_batch, Y_batch)
            b = b - learning_rate * dEdb(m, b, X_batch, Y_batch)

        cost = np.mean((m*X + b - Y)**2)
        cost_history.append(cost)

    return m, b, cost_history
```

Mini-Batch Gradient Descent - Coefficients:

Slope (m): 0.6044388922065748

Intercept (b): -0.20040960583085976

Submitted By:

Abhishek Mallick

21051706

ML_CS-3

https://github.com/Abhishek-Mallick/Machine-Learning-KIIT