Trương Sĩ Thi Vũ – USTHBI8-189

Machine Learning and Data Mining 1

Midterm report

Gradient descent with different methods

Contents

1. Problem setup

The given problem is to implement 4 different methods to perform gradient descent algorithm.

These methods are: *Batch Gradient Descent*, *Mini-batch Gradient Descent*, *Stochastic Gradient Descent*, and *Normal Equation*.

Preparation for the problem includes algorithms for these 4 methods, PyCharm code editor and some external libraries.

The data used for the problem is:

Size (m^2)	N^0 of floors	N^0 of rooms	Price (billion VND)
30	3	6	2.5
43	4	8	3.4
25	2	3	1.8
51	4	9	4.5
40	3	5	3.2
20	1	2	1.6

2. Methods and code implementation

Common variables part:

Notations for mathematics equations:

- m: number of data (or rows in the data table)
- n: number of inputs (or attributes)
- α: Learning rate

Given a dataset with x is a matrix of (m, n+1) and y is a matrix of (m,1).

$$X = \begin{bmatrix} x_0^{(1)} & x_1^{(1)} & \dots & x_n^{(1)} \\ x_0^{(2)} & x_1^{(2)} & \dots & x_n^{(2)} \\ & & \dots & \\ x_0^{(m)} & x_1^{(m)} & \dots & x_n^{(m)} \end{bmatrix} \qquad Y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \dots \\ y^{(m)} \end{bmatrix}$$

Theta is a matrix of (1, n+1)

$$[\theta_0 \ \theta_1 \ \theta_2 \dots \theta_m]$$

Cost function for all cases:

```
def cost_f(x, y, theta):
    n = len(x)
    predictions = x.dot(theta)
    cost = (1 / (2*n)) * np.sum(np.square(predictions - y))
    return cost
```

Batch Gradient Descent

- Find h(x) = $\theta^T x$
- Modify θ: $\theta_j := \theta_j - \alpha \frac{\partial}{\partial_{\theta_j}} E(\theta)$
- Find cost function = $E(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (\theta^T x^{(i)} - y^{(i)})^2$

Code implementation:

```
for i in range(iteration):
    prediction = np.dot(x, theta)
```

```
    theta = theta - (1 / n) * learning_rate * (x.T.dot((prediction - y)))
    thetas[i, :] = theta.T
    costs[i] = cost_f(x, y, theta)
```

Mini-batch Gradient Descent

- Divide the batch into smaller sets. In my example, I divided the data into 3 batches with 2 data rows in each set.
- Mix the x matrix by a random permutation
- For each iteration, run through a batch and do the exact same thing as BGD. However, this time, the thetas are updated after each batch.

Code implementation:

```
for i in range(iteration):
    indexes = np.random.permutation(n)
    x = x[indexes]
    y = y[indexes]
    for j in range(0, n, batch_size):
        x_j = x[j:j+batch_size]
        x_j = np.c_[np.ones(len(x_j)), x_j]  # Add one to the X matrix
        y_j = y[j:j+batch_size]
        prediction = np.dot(x_j, theta)
        theta = theta - (1 / batch_size) * learning_rate *
(x_j.T.dot((prediction - y_j)))
        thetas[i, :] = theta.T

    costs[i] = cost_f(np.c_[np.ones(len(x)),x],y,theta)
```

Stochastic Gradient Descent

- For each iteration, loop through the whole data set
- Selects a random index and work with the data of that row
- Perform the same operations as the previous methods

Code implementation:

```
for i in range(iteration):
    for j in range(n):
        random_index = np.random.randint(0,n)

        x_r = x[random_index, :].reshape(1, x.shape[1])
        y_r = y[random_index, :].reshape(1, 1)

        prediction = np.dot(x_r, theta)
        theta = theta - (1 / n) * learning_rate * (x_r.T.dot((prediction - y_r)))
        thetas[i, :] = theta.T
        cost = cost_f(x,y,theta)
    costs[i] = cost
```

Normal Equation

- The formula for this method is simple and we directly apply it

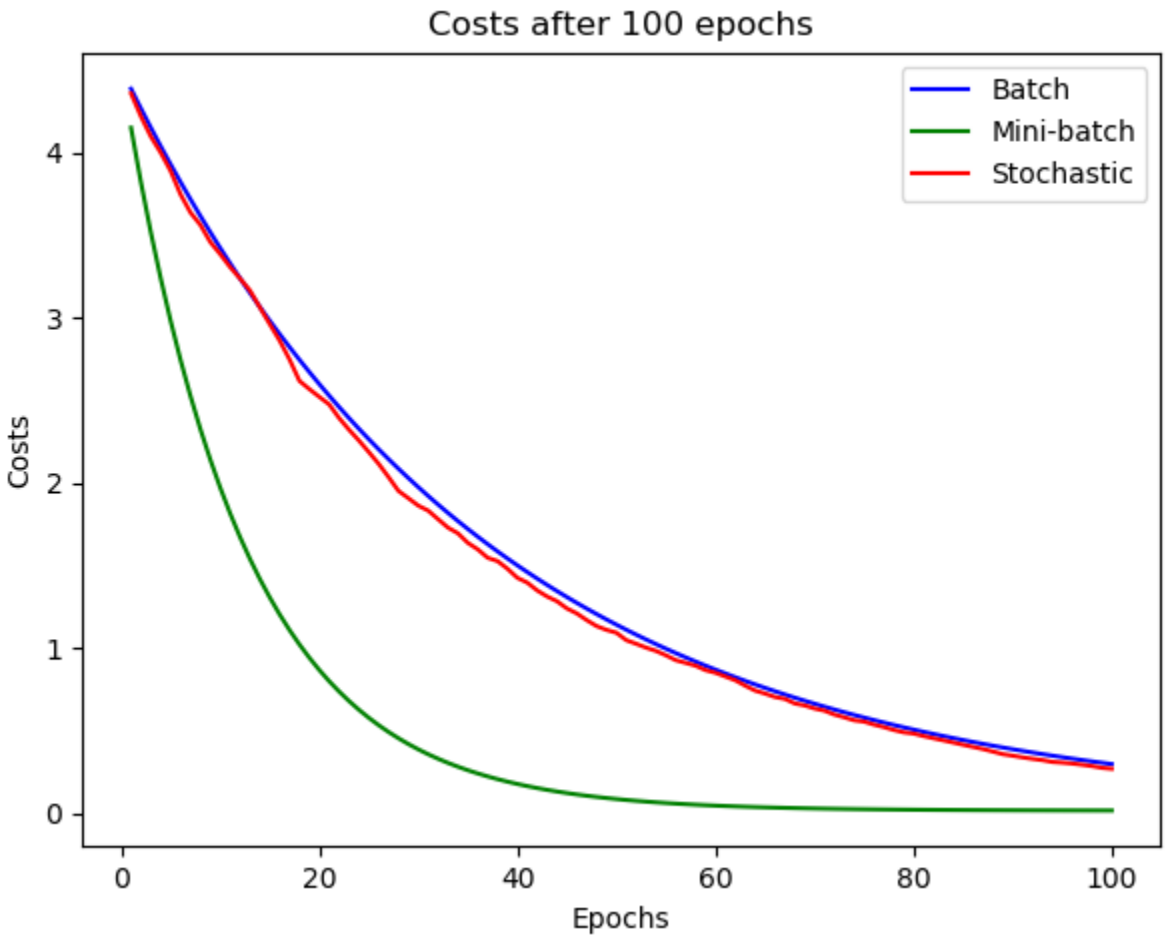
$$\theta = (X^T X)^{-1} X^T Y$$

Code implementation:

```
def normal_equation(x , y):
    print(np.linalg.inv(x.T.dot(x)).dot(x.T).dot(y))
```

3. Results and comments

Costs



From the results of the costs of 3 methods, we can clearly see the Mini-batch Gradient Descent works faster and is converging at epoch 100.

The red line shows the fluctuation of the cost, as data is randomized in the Stochastic Gradient Descent.

The blue line shows a stable drop of the cost function but not as effective as the green line.

The specific cost values for three methods are:

BGD: ***0.29693***
MGD: ***0.01579***
Stochastic: ***0.26850***

Thetas

Using the Normal Equation, we found out that the four theta values are:

[-0.15413669, 0.09406475, -0.4721223, 0.19064748]

However, the thetas for each method are:

BGD	[0.00153, 0.05959, 0.00489, 0.00979]
MGD	[0.00196, 0.07826, 0.00641, 0.01299]
Stochastic	[0.00152, 0.05922, 0.00484, 0.00968]

We can observe that the theta values are far from the acceptable result. To resolve this, I tried to increase the learning rate to 0.0001 and the iteration to 500000, and the results of thetas are:

BGD	[-0.16830, 0.09387, -0.44604, 0.18092]
MGD	[-0.15457, 0.09412, -0.47382, 0.19065]
Stochastic	[-0.16825, 0.09358, -0.44595, 0.18112]

We notice that the results are significantly close two the normal equation results. Let’s try to predict and test if these thetas are acceptable:

Size = 30 m^2

Number of floors = 4

Number of rooms = 5

The price estimated is 1.7683502181069344

Conclusion: The preferred method for this data is Mini-batch Gradient Descent. If the learning rate is 0.00001 and we run with 100 epochs, the theta values are far from correct. We can avoid this by adjusting the learning rate and the epochs. Another method to optimize these algorithms is Feature scaling.