Batch GD - MLR

Wednesday, 6 March 2024 10:04 pm

→ All training data (rows) is taken into consideration to take a single step. i.e. to calculate the slope

→ n input cols → (n+1) coefficients

x_1	$x_2 \cdots x_m$	y_i	$\hat{y}_i$
x_{11}	$x_{12} \cdots x_{1m}$	y_1	$\hat{y}_1$
x_{21}	$x_{22} \cdots x_{2m}$	y_2	$\hat{y}_2$
x_{i1}	$x_{i2} \cdots x_{im}$	y_i	$\hat{y}_i$

$$\hat{y}_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_m x_{im}$$

$$\hat{y}_i = \beta_0 + [x_{i1}\ x_{i2}\ x_{i3} \ldots x_{im}] \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \\ \vdots \\ \beta_m \end{bmatrix}$$

e.g.

x_1	x_2	y
x_{11}	x_{12}	y_1
x_{21}	x_{22}	y_2

1) Random values

$\beta_0 = 0 \quad \beta_1, \beta_2 = 1$

2) epoch = 100, $\eta = 0.1$

$$\beta_0 = \beta_0 - \frac{\partial L}{\partial \beta_0} \qquad \beta_1 = \beta_1 - \frac{\partial L}{\partial \beta_1} \qquad \beta_2 = \beta_2 - \frac{\partial L}{\partial \beta_2} \quad \cdots \quad \beta_n = \beta_n - \frac{\partial L}{\partial \beta_n}$$

n-dim → (n+1) partial derivatives

$$L = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 = \frac{1}{n}\left[(y_1 - \hat{y}_1)^2 + (y_2 - \hat{y}_2)^2\right]$$

(n: # of rows)

$\frac{\partial L}{\partial \beta_0}$

$$L = \frac{1}{2}\left[(y_1 - \hat{y}_1)^2 + (y_2 - \hat{y}_2)^2\right]$$

$$= \frac{1}{2}\left[(y_1 - \beta_0 - \beta_1 x_{11} - \beta_2 x_{12})^2 + (y_2 - \beta_0 - \beta_1 x_{21} - \beta_2 x_{22})^2\right]$$

$$\frac{\partial L}{\partial \beta_0} = \frac{1}{2}\left[(-2)(y_1 - \hat{y}_1) + (-2)(y_2 - \hat{y}_2)\right]$$

$$= \frac{-2}{2}\left[(y_1 - \hat{y}_1) + (y_2 - \hat{y}_2)\right]$$

$$= \frac{-2}{n}\left[(y_1 - \hat{y}_1) + (y_2 - \hat{y}_2) + \ldots + (y_n - \hat{y}_n)\right]$$

$$\boxed{\frac{\partial L}{\partial \beta_0} = \frac{-2}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)}$$

→ mean

$\frac{\partial L}{\partial \beta_1}$

$$L = \frac{1}{2}\left[(y_1 - \hat{y}_1)^2 + (y_2 - \hat{y}_2)^2\right]$$

$$= \frac{1}{2}\left[(y_1 - \beta_0 - \beta_1 x_{11} - \beta_2 x_{12})^2 + (y_2 - \beta_0 - \beta_1 x_{21} - \beta_2 x_{22})^2\right]$$

$$\frac{\partial L}{\partial \beta_1} = \frac{1}{2}\left[(-2) x_{11}(y_1 - \hat{y}_1) + (-2) x_{21}(y_2 - \hat{y}_2)\right]$$

$$= \frac{-2}{2}\left[x_{11}(y_1 - \hat{y}_1) + x_{21}(y_2 - \hat{y}_2)\right]$$

$$= \frac{-2}{n}\left[x_{11}(y_1 - \hat{y}_1) + x_{21}(y_2 - \hat{y}_2) + \ldots + x_{n1}(y_n - \hat{y}_n)\right]$$

$$\boxed{\frac{\partial L}{\partial \beta_1} = \frac{-2}{n} \sum_{i=1}^{n} x_{i1}(y_i - \hat{y}_i)}$$

$$\rightarrow \boxed{\frac{\partial L}{\partial \beta_m} = \frac{-2}{n} \sum_{i=1}^{n} x_{im}(y_i - \hat{y}_i)}$$

Problems :

$$L = \frac{1}{n}\left[(y_1-\hat{y}_1)^2 + (y_2-\hat{y}_2)^2 + \ldots + (y_n-\hat{y}_n)^2\right]$$

for each epoch
↓
for each coefficient
↓
Calc n-derivatives ($n = \#$ rows)

e.g. $n = 1000$
features $= 5 \rightarrow 6$ coefficients
epochs $= 50$

In each epoch, for 1 coefficient we need to calculate 1000 derivatives!

* 1) Slow on big data (Computationally intensive)

total computations = epochs × (features + 1) × n
= 50 × 6 × 1000
= 300,000

* 2) hardware limitations → Need to load all X_train data at once to calculate $\hat{y}$. Which requires RAM.

```python
class BatchGradientDescent:

    def __init__(self,learning_rate,epochs):
        self.lr = learning_rate
        self.epochs = epochs

    def fit(self,X,y):

        self.intercept = 0
        self.coef = np.ones(shape=(X.shape[1],))

        for _ in range(self.epochs):

            y_pred = np.dot(X,self.coef) + self.intercept

            bias_slope = np.sum(y-y_pred)*(-2/X.shape[0])
            self.intercept = self.intercept - (self.lr*bias_slope)

            coef_slope = np.dot(X.T,y-y_pred)*(-2/X.shape[0])
            self.coef = self.coef - (self.lr*coef_slope)


    def predict(self,X):
        return np.dot(X,self.coef)+self.intercept
```

→ Instead of loop, we use vectorization