
Machine Learning HW1

TA Hours

MLTAs
ntumlta2019@gmail.com

Outline

- Simple linear regression using gradient descent (with adagrad)
 - Extract features
 - Implement linear regression
 - Adagrad
 - Predict pm2.5

Extract Features

24

18

2014/1/1

18

2014/1/2

18

2014/1/3

...

Extract Features

Pseudo code

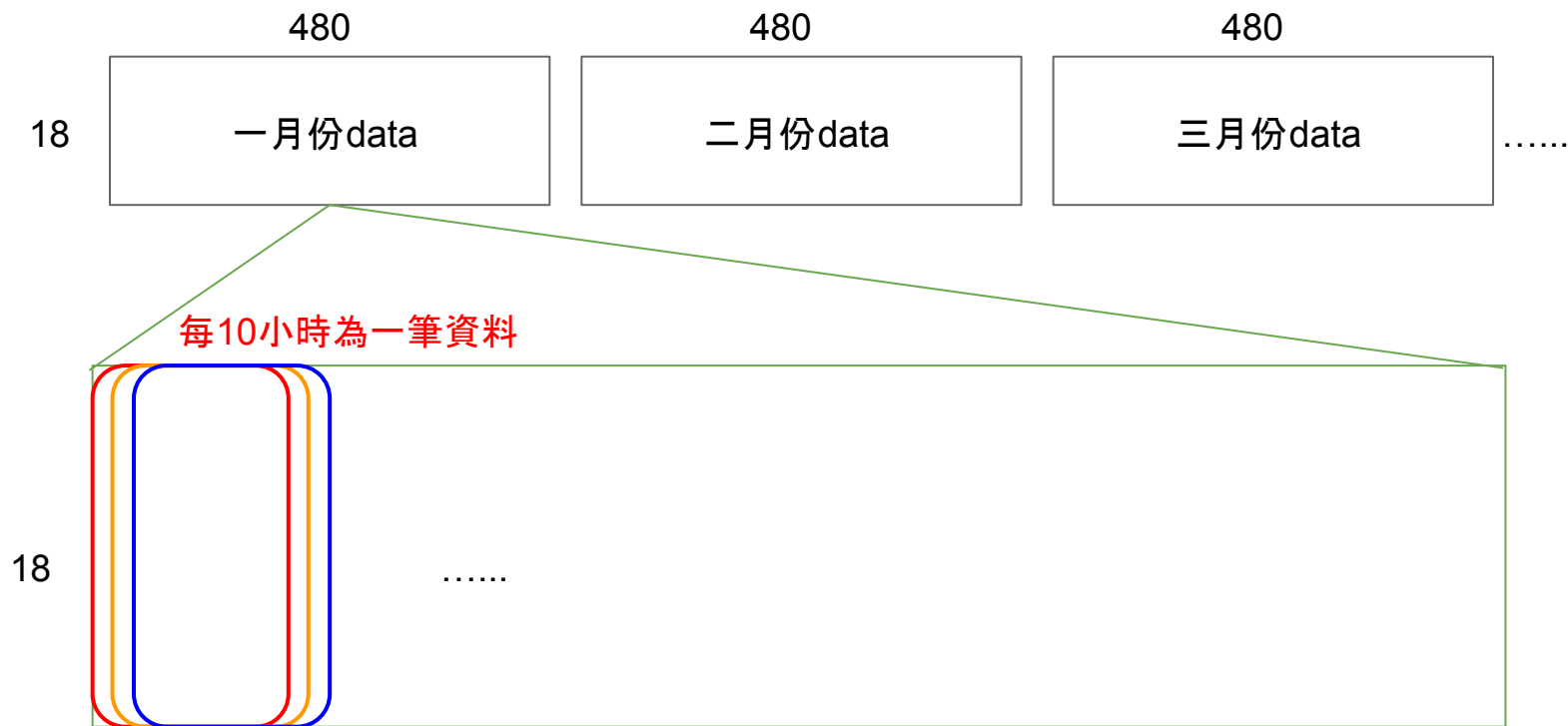
- 1 | Declare a 18-dim vector (Data)
- 2 | for i_th row in training data :
- 3 | Data[i_th row%18].append(every element in i_th row)

Data will become a vector like

2014/1/1	2014/1/2	2014/1/3
----------	----------	----------

...

Extract Features



Extract Features

Pseudo code

- 1 | Declare train_x for previous 9-hr data, and train_y for 10th-hr pm2.5
- 2 | for i in all the given data:
- 3 | sample every 10 hrs:
- 4 | train_x.append(previous 9-hr data)
- 5 | train_y.append(the value of 10th-hr pm2.5)
- 6 | add a bias term to every data in train_x

Implement linear regression

Pseudo code

- 1 | Declare weight vector, initial lr ,and # of iteration
- 2 | for i_th iteration :
- 3 | y' = the product of train_x and weight vector
- 4 | Loss = $y' - \text{train_y}$
- 5 | gradient = $2 * \text{np.dot}((\text{train_x})', L)$
- 6 | weight vector -= learning rate * gradient

$$\mathbf{y} = \mathbf{X}\mathbf{w} + \mathbf{b}$$

$$\mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

$$\mathbf{X} = \begin{pmatrix} x_1^T \\ x_2^T \\ \vdots \\ x_n^T \end{pmatrix} = \begin{pmatrix} x_{11} & \cdots & x_{1p} \\ x_{21} & \cdots & x_{2p} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{np} \end{pmatrix}$$

$$\mathbf{w} = \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_p \end{pmatrix}, \mathbf{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_p \end{pmatrix}$$

Implement linear regression

Pseudo code

- 1 | Declare weight vector, initial lr ,and # of iteration
- 2 | for i_th iteration :
- 3 | y' = the inner product of train_x and weight vector
- 4 | Loss = $y' - \text{train_y}$
- 5 | gradient = $2 * \text{np.dot}((\text{train_x})', L)$
- 6 | weight vector -= learning rate * gradient

3.


$$\begin{pmatrix} y_1' \\ y_2' \\ \vdots \\ y_n' \end{pmatrix} = \begin{pmatrix} x_{11} & \cdots & x_{1p} \\ x_{21} & \cdots & x_{2p} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{np} \end{pmatrix} \cdot \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_p \end{pmatrix}$$

4.

$$L = \begin{pmatrix} y_1' \\ y_2' \\ \vdots \\ y_n' \end{pmatrix} - \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

5.

$$\text{gradient} = 2 \times$$


$$\begin{pmatrix} x_{11} & x_{21} & \cdots & x_{n1} \\ x_{12} & x_{22} & \cdots & x_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ x_{1p} & x_{2p} & \cdots & x_{np} \end{pmatrix} \begin{pmatrix} y_1' - y_1 \\ y_2' - y_2 \\ \vdots \\ y_n' - y_n \end{pmatrix}$$

p-dim vector

Adagrad

$$\theta_{t+1,i} = \theta_{t,i} - \eta \cdot g_{t,i}.$$

In its update rule, Adagrad modifies the general learning rate η at each time step t for every parameter θ_i based on the past gradients that have been computed for θ_i :

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i}.$$

$G_t \in \mathbb{R}^{d \times d}$ here is a diagonal matrix where each diagonal element i , i is the sum of the squares of the gradients w.r.t. θ_i up to time step t , while ϵ is a smoothing term that avoids division by zero (usually on the order of $1e - 8$). Interestingly, without the square root operation, the algorithm performs much worse.

As G_t contains the sum of the squares of the past gradients w.r.t. to all parameters θ along its diagonal, we can now vectorize our implementation by performing an element-wise matrix-vector multiplication \odot between G_t and g_t :

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t.$$

One of Adagrad's main benefits is that it eliminates the need to manually tune the learning rate. Most implementations use a default value of 0.01 and leave it at that.

Adagrad

Pseudo code

- 1 | Declare weight vector, initial lr ,and # of iteration
 Declare prev_gra storing gradients in every previous iterations
- 2 | for i_th iteration :
- 3 | $y' = \text{the inner product of train_x and weight vector}$
- 4 | $\text{Loss} = y' - \text{train_y}$
- 5 | $\text{gradient} = 2 * \text{np.dot}((\text{train_x})', L)$
 $\text{prev_gra} += \text{gra}^2$
 $\text{ada} = \text{np.sqrt}(\text{prev_gra})$
- 6 | $\text{weight vector} -= \text{learning rate} * \text{gradient} / \text{ada}$

Predict PM2.5

Pseudo code

- 1 | read test_x.csv file
- 2 | for every 18 rows :
- 3 | test_x.append([1])
- 4 | test_x.append(9-hr data)
- 5 | test_y = np.dot(weight vector, test_x)

Reference

1. Adagrad :

https://youtu.be/yKKNr-QKz2Q?list=PLJV_el3uVTsPy9oCRY30oBPNLCo89yu49&t=705

2. RMSprop

<https://www.youtube.com/watch?v=5Yt-obwvMHI>

3. Adam

https://www.youtube.com/watch?v=JXQT_vxqwls