HW1 TA hours

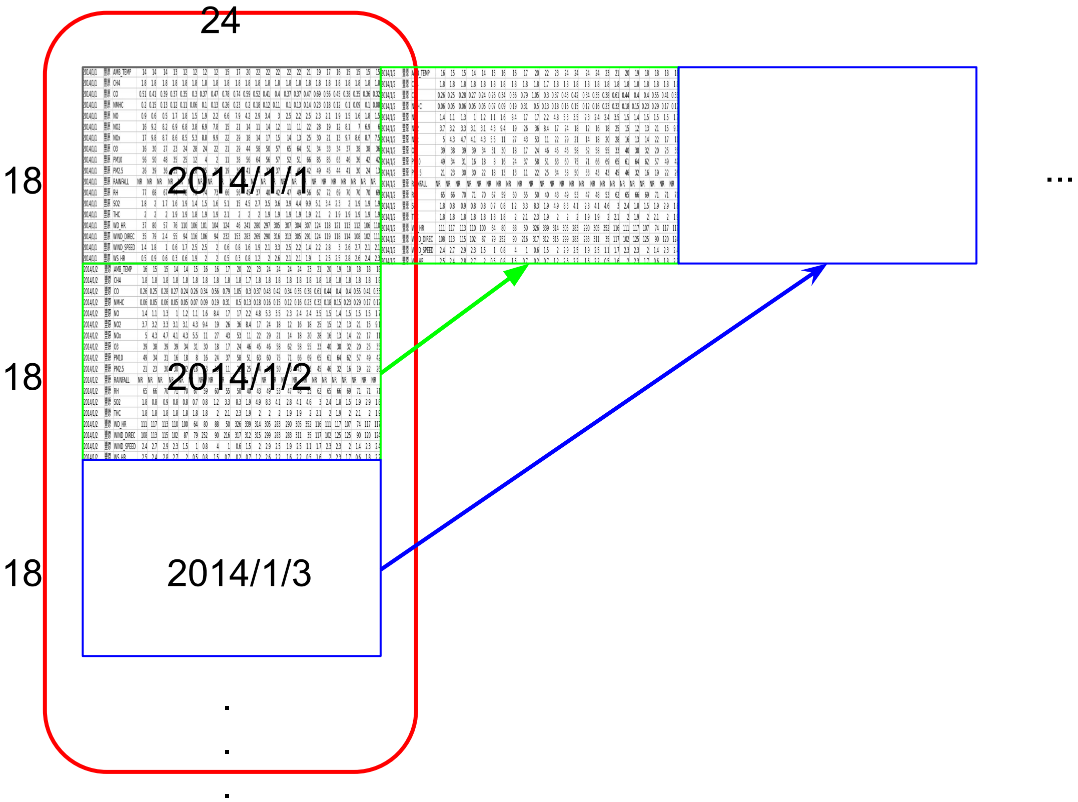
TAs
ntu.mlta@gmail.com

Outline

Simple linear regression using gradient descent (with adagrad)

1. 如何抽取feature
2. 實做linear regression
3. 使用步驟(2)的model預測pm2.5

如何抽取feature



如何抽取feature

(Pseudo code)

1. 宣告一個18維vector (Data)
2. for i_th row in training data :
3.   Data[i_th row%18].append(every element in i_th row)
4.   (可以順便處理rainfall的NR->設成0)

Data會變成一個

2014/1/1	2014/1/2	2014/1/3	...

的vector

如何抽取feature

如何抽取feature

(Pseudo code)

1. 宣告train_x儲存前9小時data，以及train_y紀錄第十小時pm2.5值
2. for i =1月、2月......
3. 　取樣每連續10個小時：
4. 　　　train_x.append(前9小時所有data)
5. 　　　train_y.append(第10小時pm2.5值)
6. 在train_x每筆data中加入bias

實做linear regression

(Pseudo code)

1. 宣告weight vector、初始learning rate、# of iteration
2. for i_th iteration :
3. y’ = train_x 和 weight vector 的 內積
4. L = y’ - train_y
5. gra = 2*np.dot((train_x)’ , L)
6. weight vector -= learning rate * gra

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

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

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

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

2. for i_th iteration :
3. y' = train_x 和 weight vector 的 內積
4. L = y' - train_y
5. gra = 2*np.dot((train_x)^T , L)
6. weight vector -= learning rate * gra

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.

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

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 24, 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.

實做linear regression

(Pseudo code)

1. 宣告weight vector、初始learning rate、# of iteration
   宣告prev_gra儲存每個iteration的gradient
2. for i_th iteration :
3. y’ = train_x 和 weight vector 的 內積
4. L = y’ - train_y
5. gra = 2*np.dot((train_x)’ , L)
   prev_gra += gra**2
   ada = np.sqrt(prev_gra)
6. weight vector -= learning rate * gra / ada

預測 PM 2.5

(Pseudo code)

1. read test_x.csv
2. every 18 rows :
3. test_x.append([1])
4. test_x.append(這9小時的data)
5. test_y = np.dot(weight vector, test_x)

Reference

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