National Taiwan University

Machine Learning: HW1

EE3 b03901016
陳昊

October 12, 2016

---

1 Linear regression

This program is aim to predict the value of $PM2.5$, according to the previous data of the weather station. There are not just the data of $PM2.5$ in the training set, but including some other features like $PM10$, NO_2, $RAINFALL$....,etc. The core concept of this program is based on **linear regression**, using **gradient descent**.

$$y = b + \sum w_i x_i$$
$$L = \sum_n (\hat{y}^n - y)^2$$

the above equations is the main idea of linear regression and gradient descent, where L is the loss function, b is bias, w is weights and x, $\hat{y}$ is the real data.

2 Method

The way I design this program is, first chain all the training data together, I create a table includes 18 arrays which contains different features respectively, then append all the data to the corresponding array. The next step is to decide which feature to use and how many hours(from 0 to 9) to trace back to construct the model. In addition to these features, I also use the method of **Momentum** and **Adagrad** to improve the efficiency of gradient descent. The final design of this program is with trace back features: (Since term $RAINFALL$ in training data has none numerical value 'NR', I change 'NR' to 0)

feature	$PM2.5$	$PM10$	NO_2	O_3	$WIND\_SPEED$	WS_HR	CH_4	$RAINFALL$
trace back	9	5	4	3	2	1	1	2

(My best grade on kaggle is base on this model.)

3 Regularization

Regularization is to add an extra term to the loss function to make it smoother, now the loss function is written as

$$y = b + \sum w_i x_i$$
$$L = \sum_n (\hat{y}^n - y)^2 + \lambda \sum w_i^2 \tag{1}$$

($\hat{y}^n$ is the real data, while y is the value that calculate by the model designed before.)
With regularization, the result should be better theoretically. However, I did not gain any improvement using regularization, it went worse than before even I tried several different values of λ. So I decided not to use it in my program.

4 Learning rate

In this program, I use some methods to improve the efficiency of gradient descent. The first one is **Momentum**, while the other is **Adagrad**.The trivial gradient descent has the formula:

$$w = w - \eta \sum_i \nabla Q_i(w) \tag{2}$$

where w is the weight of each feature, η is learning rate, $Q(w)$ is the loss function. In my best designed model, the value of η is set to 10.

4.1 Momentum

Momentum is a method to improve gradient descent, it stores the update Δw at each iteration, now we can rewrite the formula of as:

$$w = w - \eta \sum_i \nabla Q_i(w) + \alpha \Delta w \tag{3}$$

with the appropriate adjustment of α (from 0 to 1), we can make it faster to the destination of gradient descent and have better result. In my program, I design α to be 0.00038, which has the best performance.

4.2 Adagrad

Adagrad is another good way to adjust learning rate, it efficiently prevent the step being too big that may cause the loss function to explode and have efficient improvement further. We can now rewrite equation(2) as:

$$w_t = w_t - \eta \frac{\sum_i \nabla Q_i(w)}{\sqrt{\sum_{k=0}^{t-1} w_k^2}} + \alpha \Delta w \tag{4}$$

where w_k's are previous weights. In my observation, **Adagrad** is more useful with respect to **Momentum**.

Another version of **Adagrad** adds a new parameter *corr*, which sometimes have better performance than the former version. The formula is written as:

$$his\_grad = his\_grad \times corr + (1 - corr) \times w_i^2$$
$$w_t = w_t - \eta \frac{\sum_i \nabla Q_i(w)}{\sqrt{his\_grad}} + \alpha \Delta w$$

where his_grad are the sum of the historical gradient value.

However, I design the value *corr* to be 0 in my best model, the formula then behaves as same as equation(4).