Machine Learning HW1

b03901056 孫凡耕 (kaggle : b03901056_daikon)

Q1：Linear Regression Function by Gradient Descent：

- Amount of data: I used one month of data to update all parameters once, which is 480 hours of data.
- Method of Gradient Descent: I tried simple gradient descent, and then adagrad and momentum, finally adadelta. I found that adadelta performs the best overall. So the code I pasted here represents my implementation of adadelta.

#assuming utilizing only the previous 9 hours of the value of PM2.5 to predict the present value of PM2.5

#b: bias term, w[i]: the weight of (i+1) hour(s) before present time of PM2.5 value

#res: the value of one particular i for $(\hat{y}^i - (b + \sum_{j=0}^{8} w_j x_j^i))$

#adad_gw: the sum of squares of previous gradients with an attenuation coefficient ρ (rho)

#adad_dw: the sum of squares of previous Δw with an attenuation coefficient ρ (rho)

#epsilon: to avoid division by zero while calculating Δw

```
res = [-b for j in range(480-9)]    #res[i] = -b
for i in range(9, 480):
    res[i-9] += dat[pm25, i]    #res[i] = y-b
    for j in range(9):
        res[i-9] -= w[j]*dat[ pos[j], i-j-1]    #res[i] = y-b+sigma_of_x
#rho = 0.95, epsilon = 1e-8
dw = [ 0 for i in range(9) ]    #initialize to 0
gra_w = [ 0 for i in range(9) ]    #initialize to 0
for i in range(9):
    for j in range(9, 480):
        gra_w[i] -= 2*res[j-9]*dat[pm25, j-i-1]    #the gradient of a particular paramter
    adad_gw[i] = rho*adad_gw[i]+(1.-rho)*(gra_w[i]**2)    #calculating adad_gw
    dw[i] = -gra_w[i]*(adad_dw[i]+epsilon)**0.5/(adad_gw[i]+epsilon)**0.5    #calculating △w
    adad_dw[i] = rho*adad_dw[i]+(1.-rho)*(dw[i]**2)    #calculating adad_dw
    w[i] += dw[i]    #update parameters
```

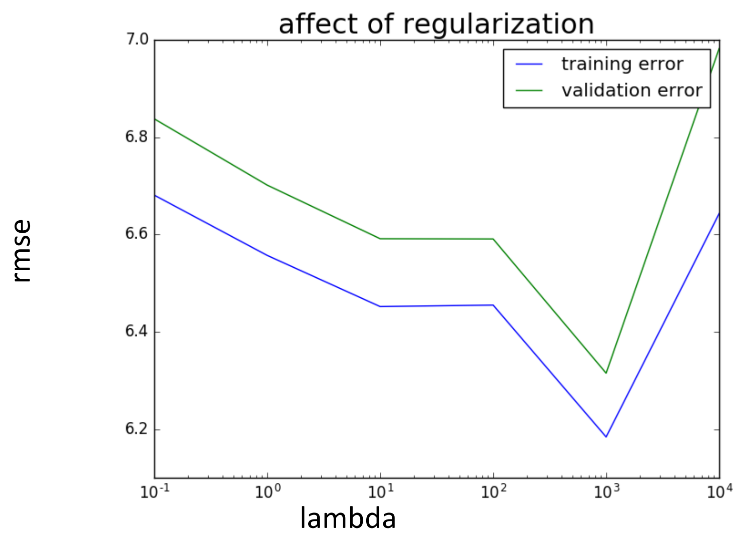
Q2：Describe Your Method：

- Mini-Batch: I cut the data into 12 separate mini-batches according the corresponding month. The reason to do so is that if I trained my model over the whole batch, I would have trained on some data that actually doesn't exist, because we only have the first 20 days of data for each month.
- Feature Extraction: Not all 18 features in the data is highly correlated to the prediction of PM2.5. So I first calculate the correlation coefficient of PM2.5 with 17 other features. Then I started to try to combine features starting from high correlation coefficient with PM2.5 and examine the result manually. Finally I found out that PM2.5, PM10, O3, NO2 and RAINFALL will all contribute positively to reduce RMSE.
- Including Rainfall: At first I didn't include RAINFALL in my training dataset, because the correlation coefficient is way to low and there are too many NRs in it. However, after I browsed the internet to find out what influences PM2.5, I realized that a positive precipitation will cause PM2.5 to drop drastically most of the time. So I included RAINFALL and without exception helps to reduce RMSE.
- Number of Terms: Just like feature extraction, I found the best number of terms for each features by trial

and error. At last, [PM2.5 * 9, PM10 * 5, O3 * 3, RAINFALL * 2, NO2 * 1] and [PM2.5^2 * 4, PM10^2 * 1] yields the best result.

➢ Use Adadelta: I felt that adagrad is too slow in hundreds of iteration, so I learnt from internet to implement a extension of adagrad that seeks to reduce its monotonically decreasing learning rate.

Q3：Discussion on regularization：



training set: 1~15-th days in a month,

validation set: 16~20-th days in a month

iteration number: 50

method: adadelta

features and terms:

*PM2.5 * 9, PM10 * 5, O3 * 3,*

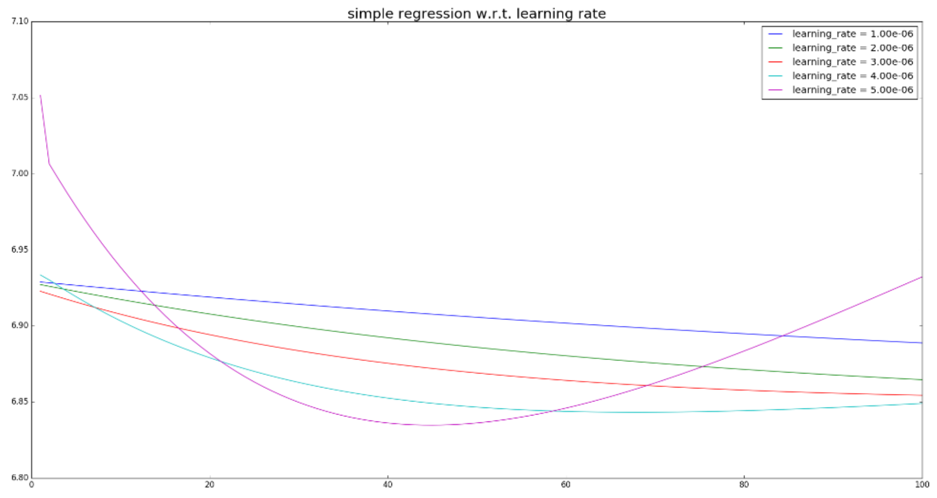
*RAINFALL * 2, NO2 * 1*

lambda value: 1e-1, 1e0, 1e1, 1e2, 1e3, 1e4

(all initialize to 0)

➢ We can see that when lambda = 1000, the performance of the model will be the best. So if the correct lambda value is chosen, we are able to improve the accuracy of the model on validation set. However, if the value of lambda is too large or too small, the results will be even worse.

Q4：Discussion on Learning Rate：



training set: 1~15-th days in a month,

validation set: 16~20-th days in a month

iteration – rmse graph

iteration: from 1 to 100

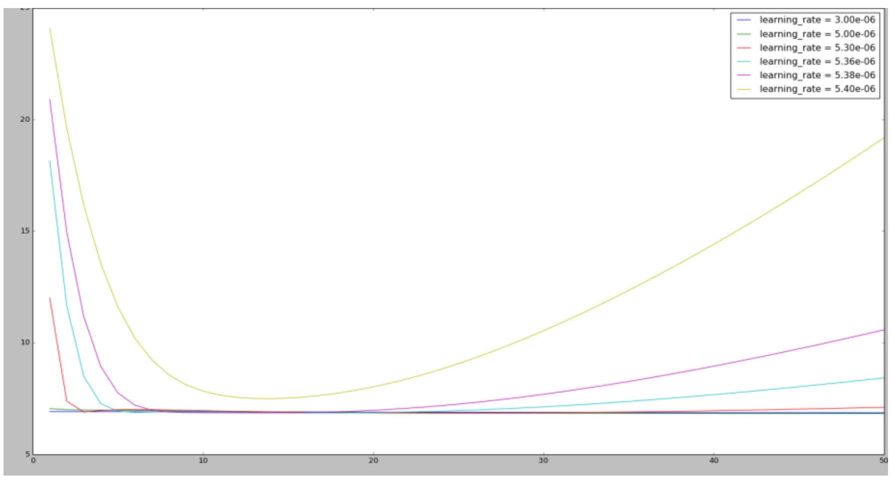
method: const. learning rate gradient descent

features and terms:

*PM2.5 * 1*

learning rate:1e-6, 2e-6, 3e-6, 4e-6, 5e-6

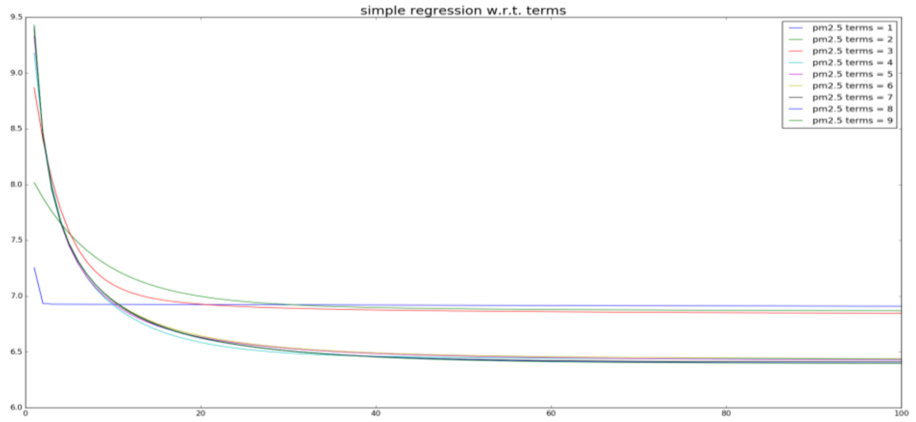
(all initialize to 0)



➢ From the graph above, we can see that if the learning rate is too small, such as the blue line, it may takes much more iterations to reach the minimum. However, if the learning rate is too large, such as the purple line, we can see that we can "probably" find the optimal point, but only if we got lucky. For the yellow line in the graph at the right(another case), we can see that it never touches the minimal rmse. So the best plan is to choose an appropriate learning rate, which is about the red line for the first graph.

Q5：Other Discussion：

- **The effect of terms：(use PM2.5 for example)**



training set: 1~15-th days in a month,

validation set: 16~20-th days in a month

iteration – rmse graph

iteration: from 1 to 100

method: const. learning rate gradient descent

learning rate: 4e-7

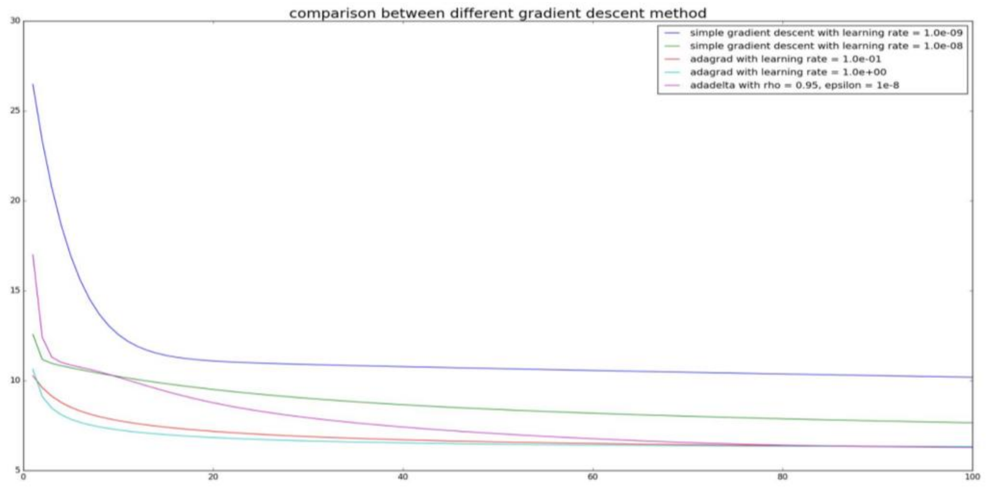
feature: PM2.5

terms:1, 2, 3, 4, 5, 6, 7, 8, 9

(all initialize to 0)

➢ We can see that the less terms we concerned about, the faster the model will converge. At the same time, we can see that although the improvement from 1 to 3 terms are tiny, the improvement from 3 to 4 terms is huge, so that's why it's pretty difficult to find the best parameters in a short period of time.

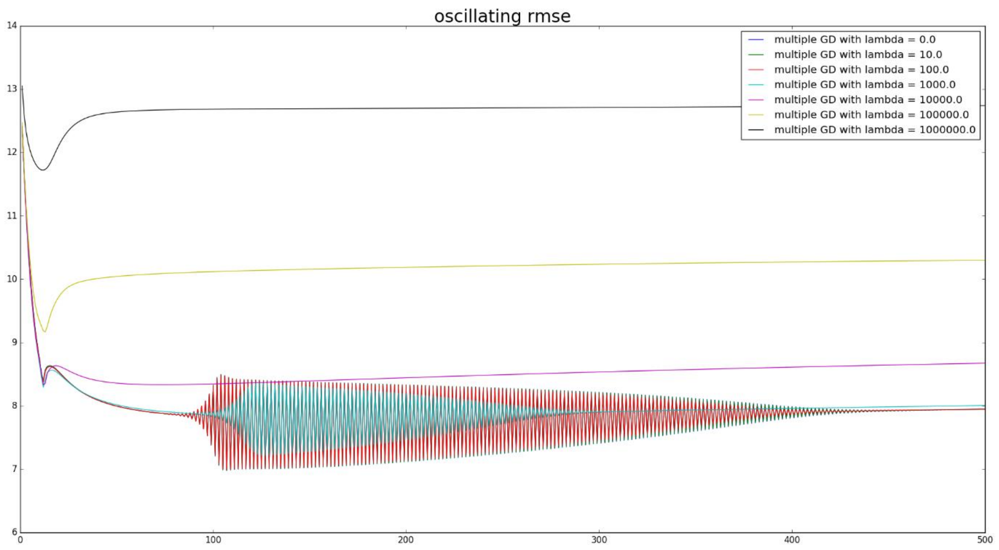
- **Comparison between constant learning rate gradient descent, adagrad and adadelta:**



We can see that adagrad and adadelta converge way faster than simple gradient descent. Adagrad also performs better than adadelta in the early stage. However in the late stage, since the learning rate of adagrad decreases monotonically, adadelta can reach the optimal point faster.

- The disadvantage of adadelta：

Since the learning rate in adadelta depends on previous gradients and updates, it's kind of like adagrad plus momentum. So when when the model almost reaches the lower rmse point, the outcomes may start to oscillate and never really come closer to the optimal point. This phenomenon depends on the value of epsilon and rho, but it's always better to switch to adagrad when we are close enough to the optimal point. However, I didn't implement it because it's hard to determine condition to switch. The phenomenon is shown below.


oscillating rmse
multiple GD with lambda = 0.0
multiple GD with lambda = 10.0
multiple GD with lambda = 100.0
multiple GD with lambda = 1000.0
multiple GD with lambda = 10000.0
multiple GD with lambda = 100000.0
multiple GD with lambda = 1000000.0