# Machine Learning HW1 Report

電機三 郭笛萱

B03901009

## Problem 1 : Linear Regression Function by Gradient Decsent

```
mini_len = 12 //    for mini-batch
for k in range (0, 100000, 1): // total iteration
    loss = 0      //initialization
    diffb = 0
    for all features m:
        for i in range(0, feature_m_term, 1):
            diff_feature[i] = 0

    for i in range(0, len(data2_PM25)/mini_len,1): // look through all data
        t = 0
        w_s = 0
        for all features m:          //calculate the sum of wi * xi
            for j in range(0, feature_m_term, 1):
              t += float(data2_feature_m[i+mini][j]) *float(w_feature_m[j])
              w_s += float(w_feature_m[j]) ** 2
        for all features_m: // calculate the gradient
            for j in range(0, feature_m_term, 1):
                diff_m[j] += 2 * (float(data2_PM25[i+mini][PM25_term]) - (t + b)) *
                                (-float(data2_feature_m[i+mini][j]))
                            + 2 * x * w_feature_m[j]      //regularization
        diffb += 2 * (float(data2_PM25[i+mini][PM25_term]) - (t + b)) * (-1)
        loss += (float(data2_PM25[i+mini][PM25_term]) - (t + b)) **2 + x * w_s

    for all feature_m:    //update the weights
          for j in range(0, feature_m_term, 1):
              w_feature_m[j] -= n_feature_m[j] * diff_feature_m[j]

    //update bias
    b -= n_b * diffb
```

## Problem 2: Describe Your Method

1. Select features : calculate the correlation coefficients between PM2.5 and the other features. I found that the correlation between PM2.5 & PM10 is the highest --- 0.75, and the correlation between the other features and PM2.5 are low. I only chose features having correlation above 0.3 as input.

   The best case I got is setting the term of PM10 to 6, PM2.5 to 7, and CO, NMHC, NO2, NOx, O3, SO2, THC to 1.

2. Mini-batch : mini-batch is a method which cuts the data into groups, and update the weight after one group instead of after looking all the data. I try to cut the data into 16, 12, 8 groups, set the iteration to 80000, 50000, 30000, and found that cutting data into 12 groups and using 50000 iterations is the best combination.

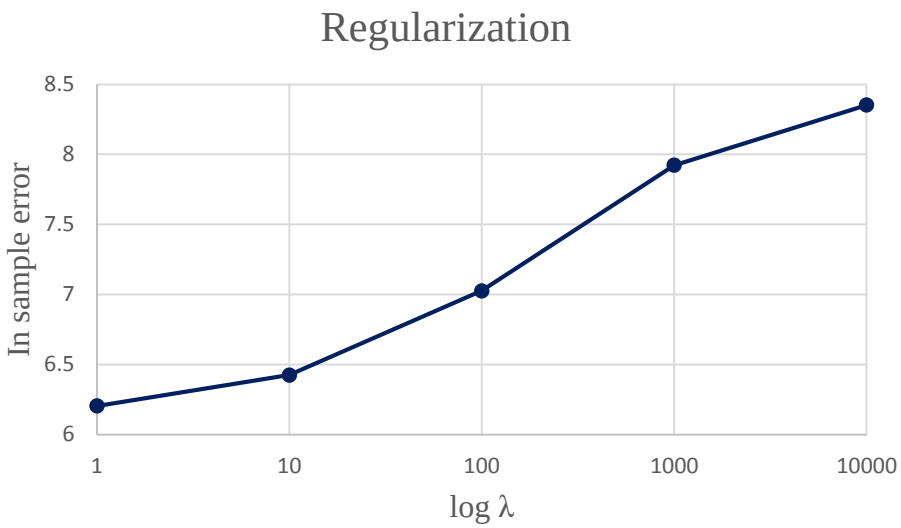
3. Loss function : I choose the loss function to be $\sum(\hat{y} - (\sum w_i\, x_i) - \sum w_i\, {x_i}^2)^2$, which includes 9 features in linear term and PM2.5 & PM10 in quadratic form. After testing, choosing four terms of PM2.5 & one term of PM10 in quadratic form gives the best result.

4. Adagrad : I have tried to add adagrad in my program, but the result will be worse than not adding adagrad.

5. Delta function : I add the concept of slope into my program. If the slope of the previous PM2.5 data > a certain number, than I took the slope into concern. After trying some combination, finally I chose slope > 30 to be the final parameter.

## Problem 3: Discussion on Regularization

After trying some values of $\lambda$, I find that adding regularization does not enhance the performance. The results are shown as the following figure. As $\lambda$ increases, the in sample error, $\sum(y - (\sum w_i \times x_i + b))^2$, will increase, which is not what we expect. Guess it's because regularization should be added when overfitting exists, but in our model, overfitting is not notable.




## Problem 4 : Discussion on Learning Rate

We can separate the discussion on learning rate into two parts.

The first part is the situation when adding Adagrad. When adding Adagrad, we will divide the learning rate with a number related to all the gradient of the features, so we have to set the learning rate higher than the model without using Adagrad. After several testing, I choose to use 100. If choosing the value of learning rate higher than 100, the learning rate will be too high, and the loss function will increase quickly. On the other hand, choosing lower than 100 will make the learning rate too small, and the loss function can't converge within 10000 iteration.

For the model not using Adagrad, I choose the value of learning rate to be $1 \times 10^{-8}$, which is almost the same order as the learning rate using Adagrad after dividing the learning rate with a big number.