Machine Learning Assignment 1

Tung-Chun, Chiang R05922027

October 9, 2016

1 Linear regression function

```
def gradient_descent(x, y):
  # compute square error
  # x is feature vector and y is PM2.5 target value
  # W, b are weight vector and bias of this model
  cost_partial = np.dot(x, W) + b - y

  # gw is the gradient of W
  # C is regularization factor
  gw = np.dot(c_partial, x) + C * W

  # gb is the gradient of b
  gb = np.sum(c_partial)

  # update parameters
  W -= learningRate * gw
  b -= learningRate * gb

  return
```

2 My method

2.1 Data preprocessing

First, we replace the NR values in RAINFALL columns to 0.0.
For each month, we concatenate 20 days and use a sliding window with $window\_size$ 10 to scan the timeline to generate blocks (10 hours each block). For each block, We use all values in the first 9 hours as feature and the PM2.5 value of the remaining hour as ground truth. The total data size is $12 * (20 * 24 - 10 + 1) = 5652$.

2.2 Feature extraction

For each instance, we use origin feature x (dimension is $18*9 = 162$) and $x^2_{normalized}$ ($x_{normalized} = x/sum(x)$). The dimension of feature vectors is 324. We don't normalize origin feature x because it may lose some information of real values. Also, to add time information (such as the answer would be effected more by recent hours), we multiple the feature vector by [1., 1., ..., 9., 9.] (features of the first hour * 1., features of the second hour * 2., ..., and so on.) to enlarge the features of recent hours.

2.3 Cost function

$$\sum_{(x,y)} (W^T x + b - y)^2 + C \sum_{w} w^2$$

- W, b: weight vector and bias.
- (x, y): an instance pair, x means feature vector and y means ground truth.
- w: a weight in W.
- C: hyper parameter of regularization

2.4 Gradient descent

2.4.1 Adagrad

To get better results, the learning rate should decrease during training. We use Adagrad method to tune learning rate dynamically. $Adagrad$ divides learning rate by summation of square of gradients. When model training, the summation will increase and learning rate will decrease.

2.4.2 Momentum

We use $Momentum$ method to speed up training and get out of saddle points. Momentum method adds the current gradients with gradients in last iteration. So the model won't stuck in saddle points, where the gradients are close to 0.

2.4.3 Mini-batch

We use $Mini-batch$ method to speed up training. Unlike Stochastic Gradient Descent, which updates parameters each instance, Mini-batch updates parameters each batch, which contains $batch\_size$ instances. It saves time cost of matrix computations.

3 Experiments

3.1 Settings

In Regularization experiments, we use 324-dim features (x, $x^2_{normalized}$ and time factor mask), 10000 iterations, Momentum factor 0.3, batch size 100 and Adagrad method. Error function is $RMSE$. In Learning rate experiments, the settings are the same as Regularization experiments excluding Momentum factor 0.0 and Adagrad method is not used.

3.2 Regularization

Regularization makes the weights closer to 0 (more smooth) and the model would be simpler so that the variance is smaller. In Table 1, the results with smaller C vary larger than the results with larger C. By the way, the E_{in} is usually larger than E_{kaggle}. I think it is because the model is too simple (underfitting), or feature dimension is too small.

C, result	E_{in}	E_{kaggle}
100	5.72577	5.61991
10	5.76548	5.61620
1	5.81074	5.66569
0.1	5.73037	5.63915
0.01	5.72686	5.61323
0	5.72877	5.64137

Table 1: Results of different C with learning rate = $5e - 10$

3.3 Learning rate

The results of this experiment are showed in Figure 1. Large learning rates (larger than 1e-9, we don't show here because the error is extremely higher) stuck in high RMSE after some updates. if learning rate (1e-16, 1e-20) is too small, RMSE decreases very slow. Training with appropriate learning rates (1e-9, 1e-12) makes errors very low with fewer numbers of updating parameters.

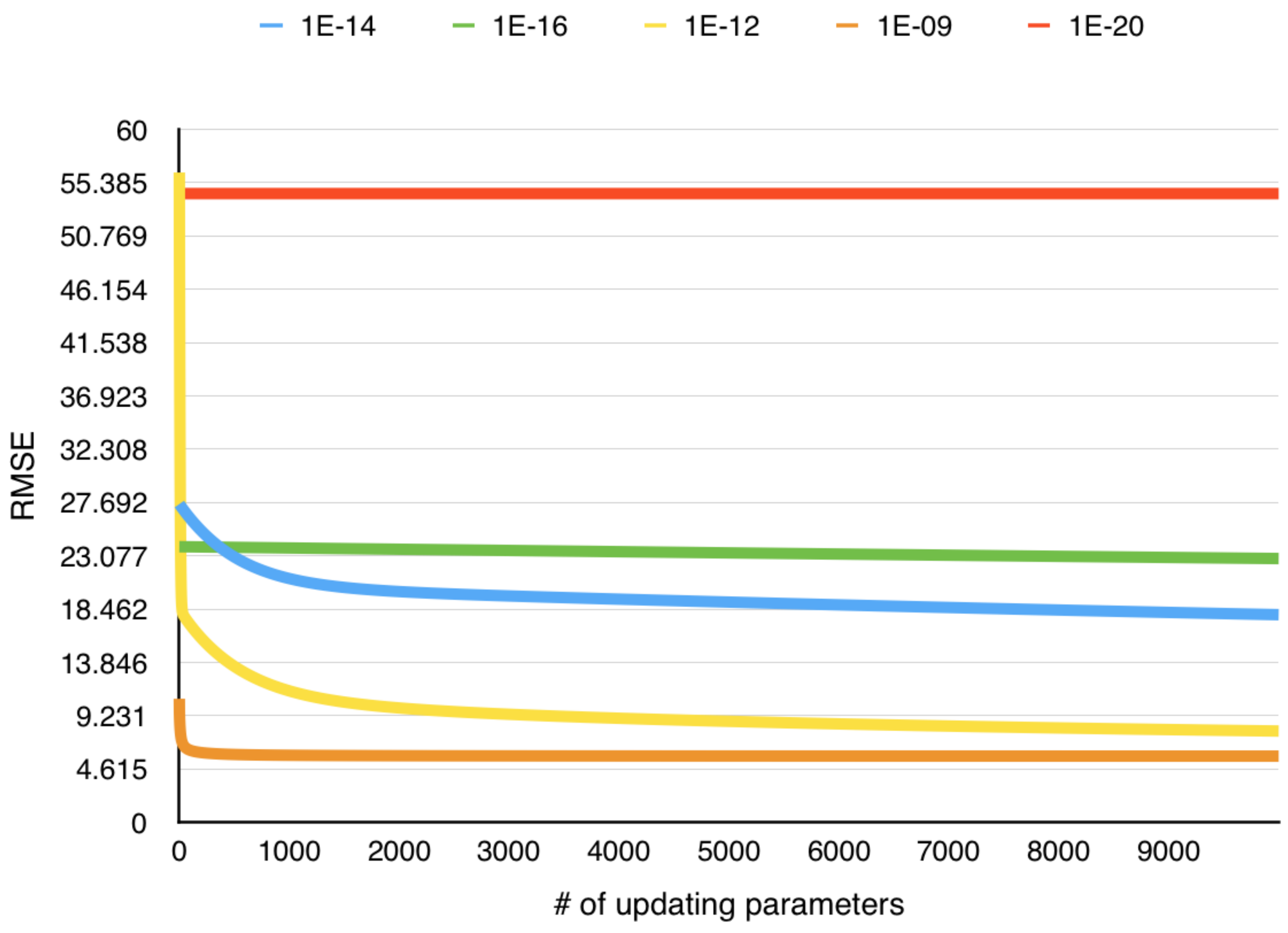

Figure 1: $RMSE_{in}$ of different learning rates with C = 0.1