Implementation Assignment #1

Alireza Mostafizi

Introduction

I coded this assignment in MATLAB. Attached to this report, you can find four MATLAB codes. The assignment is to implement linear regression with quadratic regularization to avoid over-fitting with the following regularized sum of squared error objective function:

$$\sum_{i=1}^{N}(W^T X_i - y_i)^2 + \lambda \left|\left|W\right|\right|^2$$

The data provided includes one set of training data with 100 data points, and one testing data with the same size. Each data point specifies 45 features, plus a dummy constant of 1 column to match up with the matrix calculation with the weight vector. Since some features have large values, it is beneficial to modify the data to a shorter range, either with normalization or linear modification of both the features and output. To do so, all the features are modified as following, so the new features can get the values only between 0 and 1.

$$x_{new} = \frac{x - Min(x)}{Max(x) - Min(x)}$$

$$y_{new} = \frac{y - Min(y)}{Max(y) - Min(y)}$$

In this assignment, optimization is approached by **Batch Gradient Descent Algorithm**. Since the number of data points is not large, batch updating can lead to the convergence in a short time. Thus, here is the gradient formula, and the algorithm to update weight vector:

$$E(W) = \frac{1}{2}\left(\sum_{i=1}^{N}(W^T X_i - y_i)^2 + \lambda \left|\left|W\right|\right|^2\right)$$

$$\nabla E(W) = \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \dots, \frac{\partial E}{\partial w_d}\right]^T$$

$$\frac{\partial E}{\partial w_k} = \sum_{i=1}^{N}(W^T X_i - y_i)x_{ik} + \lambda w_k$$

$$\nabla E(W) = \sum_{i=1}^{N}(W^T x_i - y_i)X_i + \lambda W$$

With this in mind, the algorithm can be summarized as following:

```
$W = W^0$
*Repeat {*
    $\nabla E(W) = \sum_{i=1}^{N}(W^T x_i - y_i)X_i + \lambda W$
    $W \leftarrow W - \alpha \nabla E(W)$
*} Until* $|\nabla E(W)| < \epsilon$
```

Where λ is the regularization coefficient, and α is the learning rate. Both of these coefficients are going to be analyzed and their effects on the outcome of the algorithm is going to be furtherly assessed. Based on my observation on the convergence of the algorithm and its general behavior in this particular case, I set the ϵ value to 0.1.

Part I: Exploring Different Learning Rates, $\boldsymbol{\alpha}$

The approach I took to explore the effect of different learning rates on the learning process, would be to apply different learning rates to the algorithm, and compare the learning curves and the quickness of the algorithm in terms of convergence. Thus, to do so, I ran the algorithm with the following values of learning rates. Time to convergence is captured as performance measurement, and results are presented in Figure 1 which shows Time to Convergence versus the log learning rate.

$$Log(\alpha) = \{-6, -5.9, -5.8, \ldots, -0.2, -0.1, 0\}$$

Then the α itself take the values ranging from 10^{-6} and 0. The λ in this experiment, for simplification, is set to 1. Each run has been going on for 100,000 trials, and if they had not been converged, their Time to Convergence is set to 10,000+.

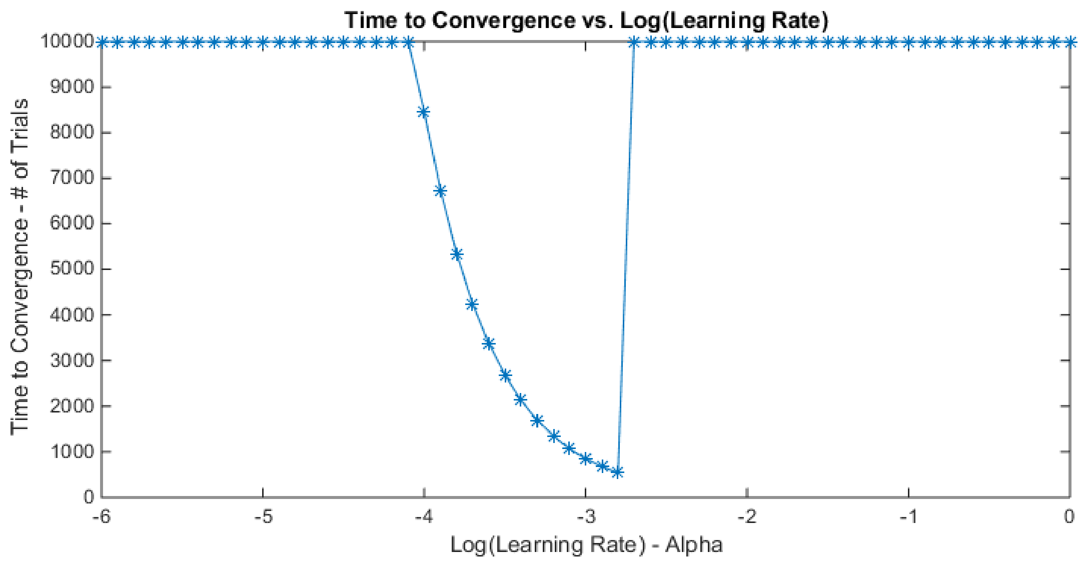

Figure 1. Time to Convergence vs. Log (Learning Rate)

Learning Rates										
$\log(\alpha)$	-3.6	-3.5	-3.4	-3.3	-3.2	-3.1	-3.0	-2.9	-2.8	-2.7
α	0.0002	0.0003	0.0004	0.0005	0.0006	0.0008	0.001	0.0013	0.0016	0.002
Trials	3373	2679	2128	1690	1343	1067	847	673	535	+10,000

Table 1. # Of Trials to Convergence vs. Learning Rate

To clarify how different learning rates affect learning process, Figure 2 shows three learning curves associated with three different learning rates.

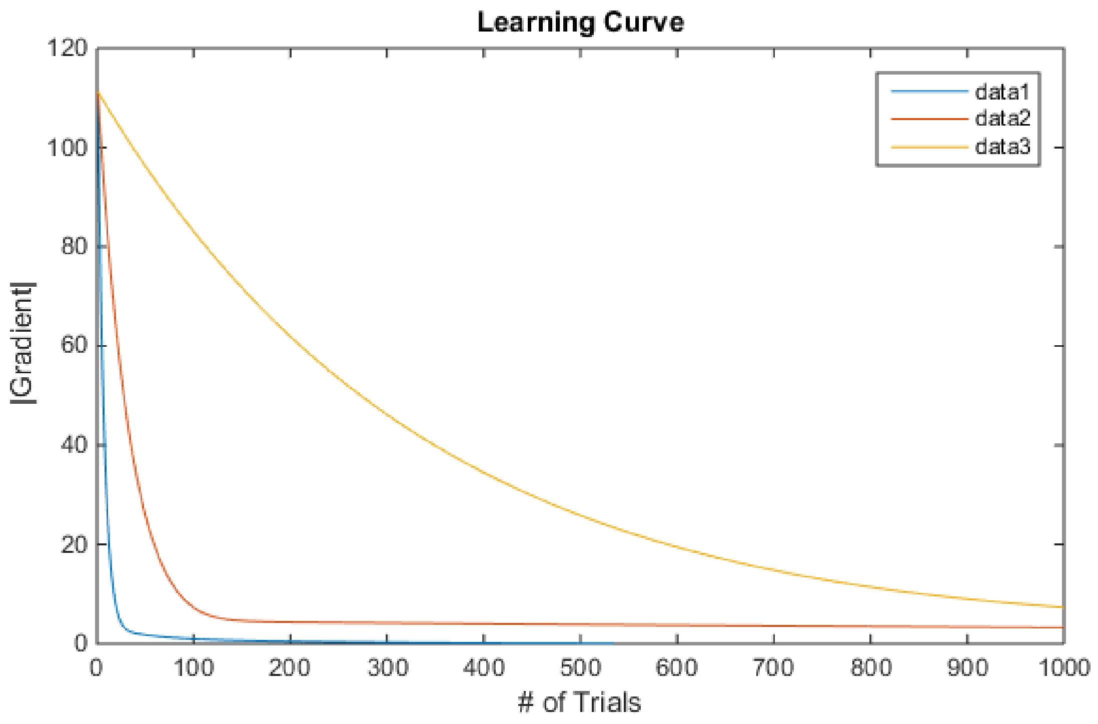

Figure 2. Learning curves associated with three different learning rates

As expected, the yellow curve, which is associate with learning rate of $2.5 * 10^{-6}$, is slowly converging to the optimal wright vector. On the other end, the blue line represents the optimal learning rate which equals to 0.0016, and it converges to the optimal W faster than the others do.

Discussion

The result are totally in line with the intuition. There is an optimum value for the learning rate, which minimizes the convergence time. As you decrease the learning rate, it takes longer for the algorithm to converge. On the other hand, if you increase the learning rate over that threshold, it may cause oscillation since the algorithm jumps over the optimum value of the objective function, which leads to longer convergence times and in some cases, failure to convergence. As you saw in the Table 1 and Figure 1, the optimum learning rate in this case is 0.0016.

Part II: Exploring Different Regularization Coefficient

Knowing that learning rate of 0.0016 works fine with the algorithm, we can play with λ to assess its effect on the algorithm. Like the previous section, $Log(\lambda)$ is set to vary between the following values:

$$\log(\lambda) = \{-3, -2.9, -2.8, \dots, 1.8, 1.9, 2.0\}$$

The the λ itself changes from 0.001 to 100. Figure 3 shows the results of sum of squared error on the training data, and the test data.

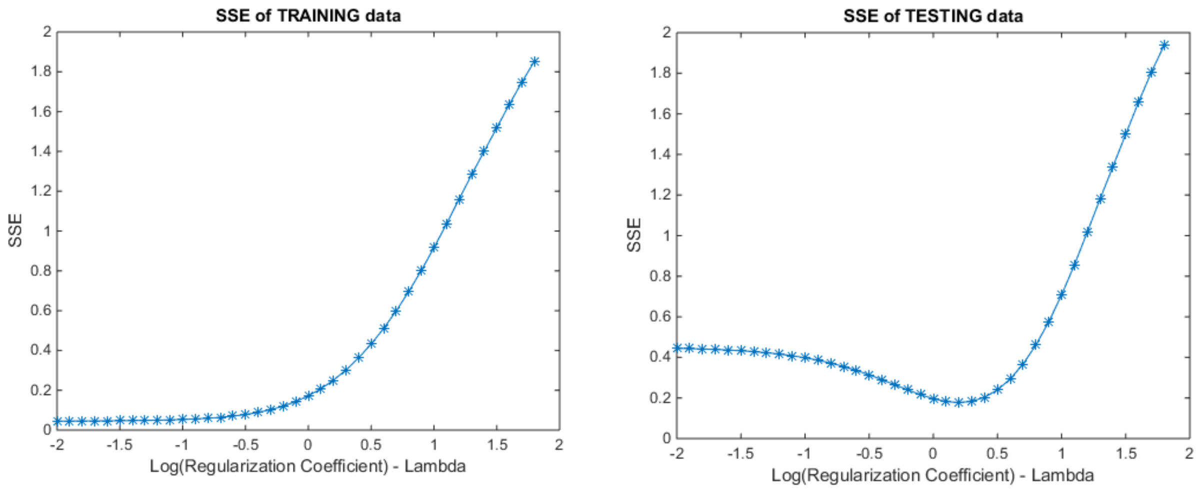

Figure 3. Sum of Squared Error of Training and Test Data vs. Log(Regularization Coefficient)

Discussion

As shown in the figure 3, the best regularization coefficient which leads to the least SSE on the testing data, is $10^{0.2} = 1.58$.

- As we expected, the trend of SSE is increasing as we increase λ, mainly because, when the λ is increased, the algorithm tends to fit simpler models to the data which inherently leads to higher errors on training data since it might not be able to explain the data in a way that a more complex model with higher W values can.

- However, an increase in λ to some point decreases the error on the testing data, and after a threshold, it starts to increase the SSE. This is because the complex model which comes with low regularization coefficients, cause over-fitting. When the λ is increased, over-fitting will be resolved to some point, and after that, under-fitting comes to the place due to over-simplification of the model.

Part III: Cross-Validation

As suggested in the assignment, 10-fold cross validation has been implemented on the training data. The results of sum of squared error are presented in Figure 4. Just like the previous section, I tried different values for regularization coefficient as following:

$$\log(\lambda) = \{-3, -2.9, -2.8, \dots, 1.8, 1.9, 2.0\}$$

The the λ itself changes from 0.001 to 100.

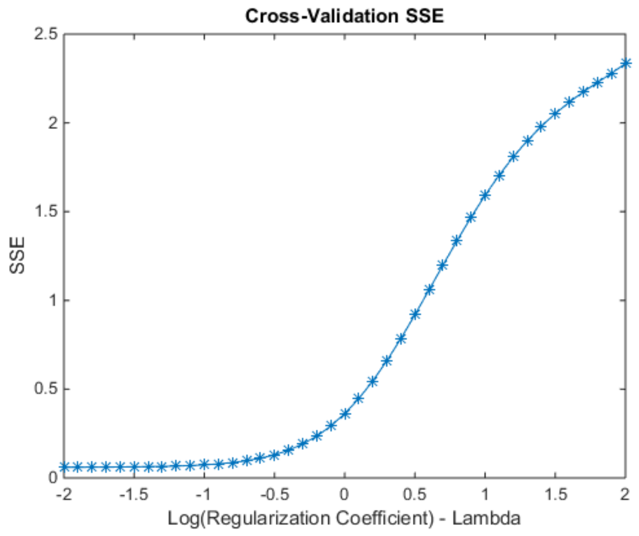

Figure 4. Sum of Squared Error of Cross-Validation on Training Data vs. Log (Regularization Coefficient)

Discussion

- Regarding choosing λ, generally speaking, I would say the highest possible value for λ, which provides the simplest weight vector, and at the same time, does not generate high errors should be picked. In this case, $10^{-0.5} = 0.32$ might be reasonable to choose since it is almost the threshold that after that, SSE starts to increase with the increase of λ.

- The SSE with the cross validation is a little bit larger than the SSE of the training data itself. The main reason of this phenomenon is that, in the case of cross-validation, we are choosing smaller sample to learn the model, and use that small sample, to validate the other part of the dataset. Then it rotates, but in each round, there is no data mutual between training part and validating part. On the other hand, the SSE coming from training dataset is calculated based on the model that is learnt using the same dataset. Therefore, it does make sense to expect higher SSE for cross-classification.

- The shape of the curve is close to the SSE on the training data, since the model is being both learned and validated by the same dataset, not with two different datasets.

Notes on running the code:

1. Main code is “Main.m,” and the others are functions.
2. Run the “Initialization” part in the main code first
3. Run the code associated to each part of the assignment separately to get the results and graphs.
4. Run the plot commands one by one.