华南理工大学

South China University of Technology

The Experiment Report of Machine Learning

SCHOOL: SCHOOL OF SOFTWARE ENGINEERING

SUBJECT: SOFTWARE ENGINEERING

Author:
Yi Lan

Supervisor:
Mingkui Tan

Student ID：
201630664673

Grade:
Undergraduate

October 24, 2018

Linear Regression and Gradient Descent

Two methods (Closed-formed and Gradient Descent) are used in this experiment to implement linear regression. All code is written with python and tested in python3 platform.

I. Introduction

Different from linear classification, the target values of linear regression are discrete, and the model's responsibility is to use the feature values to predict a value which should be as close as the actual value. However, the linear classification model just needs to try its best to put the sample in correct side.

Simple linear regression describes the linear relationship between a predictor variable, plotted on the x-axis, and a response variable, plotted on the y-axis. The goal of linear regression is to learn a hypothesis/model f: X ↦ Y

This experiment aims to implement linear regression using two methods in the "Linear Regression" section of the machine learning course, and experience the effects of various hyperparameters on model training and explore how to train a better model.

II. Methods and Theory

Define:

Input space: $X = \mathbb{R}^m$, means features, covariates, predictors, etc.

Output space: *Y*, can be many different types of predictions.

Goal model: a hypothesis/model f: X ↦ Y

In supervised learning, we set the input, output pairs with the given dataset:

$$D = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$$

and what we want is the "best" model based on *D*.

Predict ŷ for unseen x based on f(x).

About model function f(X; W), we define:

- Parameters: $\mathbf{w} \in \mathbb{R}^m, w_0 \in \mathbb{R}$
- Input: **x**, where $x_j \in \mathbb{R} \quad for \quad j \in 1, ...m$
- Model function:

$$\begin{aligned} f(\mathbf{x}; w_0, \mathbf{w}) &= w_0 + w_1 x_1 + ... + w_m x_m \\ &= \sum_{j=1}^{m} w_j x_j + w_0 \\ &= \mathbf{w}^\top \mathbf{x} + w_0 \end{aligned}$$

We use *Loss Function* to measure the performance of our linear regression model. In this experiment, we use *Least Squared Loss* as loss function:

$$\begin{aligned} \mathcal{L}_D(\mathbf{w}) &= \frac{1}{2}\sum_{i=1}^{n}(y_i - f(\mathbf{x}_i; \mathbf{w}))^2 \\ &= \frac{1}{2}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \end{aligned}$$

Our training goal is to find minimizer of least squared loss:

$$w^* = \arg\min_w \mathcal{L}_D(w)$$

1. Closed-form Solution for Linear Regression

For least squared loss function

$$\mathcal{L}_D(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{n}(y_i - \mathbf{w}^\top \mathbf{x}_i)^2 = \frac{1}{2}(\mathbf{y} - \mathbf{X}\mathbf{w})^\top(\mathbf{y} - \mathbf{X}\mathbf{w})$$

Then

$$\begin{aligned} \mathcal{L}_D(\mathbf{w}) &= \frac{1}{2}(\mathbf{y} - \mathbf{X}\mathbf{w})^\top(\mathbf{y} - \mathbf{X}\mathbf{w}) \\ &= \frac{1}{2}(\mathbf{y}^\top\mathbf{y} - 2\mathbf{w}^\top\mathbf{X}^\top\mathbf{y} + \mathbf{w}^\top\mathbf{X}^\top\mathbf{X}\mathbf{w}) \\ \frac{\partial \mathcal{L}_D(\mathbf{w})}{\partial \mathbf{w}} &= \frac{1}{2}\left(\frac{\partial \mathbf{y}^\top\mathbf{y}}{\partial \mathbf{w}} - \frac{\partial 2\mathbf{w}^\top\mathbf{X}^\top\mathbf{y}}{\partial \mathbf{w}} + \frac{\partial \mathbf{w}^\top\mathbf{X}^\top\mathbf{X}\mathbf{w}}{\partial \mathbf{w}}\right) \\ &= \frac{1}{2}(-2\mathbf{X}^\top\mathbf{y} + (\mathbf{X}^\top\mathbf{X} + (\mathbf{X}^\top\mathbf{X})^\top)\mathbf{w}) \\ &= -\mathbf{X}^\top\mathbf{y} + \mathbf{X}^\top\mathbf{X}\mathbf{w} \end{aligned}$$

And then

$$\begin{aligned} \frac{\partial \mathcal{L}(\mathbf{w})}{\partial \mathbf{w}} &= -\mathbf{X}^\top\mathbf{y} + \mathbf{X}^\top\mathbf{X}\mathbf{w} = 0 \\ &\Rightarrow \mathbf{X}^\top\mathbf{X}\mathbf{w} = \mathbf{X}^\top\mathbf{y} \\ &\Rightarrow \mathbf{w} = (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{y} \end{aligned}$$

Finally, we solve for optimal parameters **w***

$$\mathbf{w}^* = (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{y} = \arg\min_{\mathbf{w}} \mathcal{L}_D(\mathbf{w})$$

2. Gradient Descent for Linear Regression

As for some critical issues in closed-form solution, for example, many matrices are not invertible, so we have to find another solution to deal with this case and make linear regression more easily.

Learning based on *Gradient Descent* is done through optimization. The main tool is *Gradients* (vector of partial derivatives):

$$\frac{\partial \mathcal{L}_D(\mathbf{w})}{\partial \mathbf{w}} = \begin{bmatrix} \frac{\partial \mathcal{L}_D(w_1)}{\partial w_1} \\ \frac{\partial \mathcal{L}_D(w_2)}{\partial w_2} \\ \vdots \\ \frac{\partial \mathcal{L}_D(w_n)}{\partial w_n} \end{bmatrix}$$

We use $d = -\frac{\partial \mathcal{L}_D(w)}{\partial w}$ as the direction of optimization and minimize loss by repeated gradient steps:

a) Compute gradient of loss with respect to parameters $\frac{\partial \mathcal{L}_D(w)}{\partial w}$

b) Update parameters with learning rate η:

$$w' \rightarrow w - \eta\frac{\partial \mathcal{L}_D(w)}{\partial w}$$

Learning rate η has a large impact on convergence. if η is large, the loss curve would be oscillatory and may even diverge. On the other hand, too small η would make loss curve converge too slowly.

III. EXPERIMENT

Data Set Source:
https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/regression/housing_scale

The data set includes 526 samples with 123 features and 1 target value.

Experiment Steps:

1. **Closed-form Solution**
   a) Load and preprocess the experiment data.
   b) Initialize model parameters 1-diemond matrix as ones.
   c) Code the compute formula.
   d) Calculate the model parameters and update it.
   e) Using the trained model to calculate the training loss and testing loss.
   f) Output result.
2. **Gradient Descent Solution**
   a) Load and preprocess the experiment data.
   b) Define and initialize hyper-parameters: learning rate and the max epochs of training.
   c) Initialize model parameters 1-diemond matrix as ones.
   d) Code the *Least Squared Loss* function and define it.
   e) Calculate gradient G toward loss function from all samples.
   f) Update model parameters using $w' = w - \eta G$. η is learning rate.
   g) Calculate the training loss and testing loss using current model. Record them.
   h) Output monitor information.
   i) Repeat (e) to (h) until the circulation end.
   j) Drawing graph of training losses and testing losses, as well as with the number of iterations.

Experiment Hyper-parameters Selection Scheme:

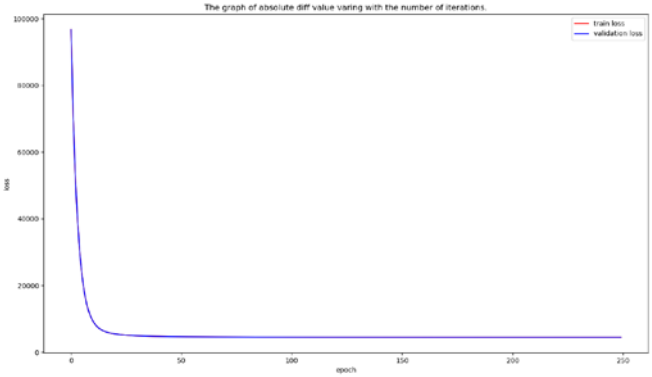
1. Closed-form Solution
   Train Loss: 4530.4540
   Test Loss: 1095.7302
2. Gradient Descent Solution

Learning rate	Max epochs	Train loss	Test loss
0.00200	50	3.3268E+47	3.3268E+47
	100	9.7480E+89	9.7480E+89
	150	2.8563E+132	2.8563E+132
	200	8.3694E+174	8.3694E+174
	250	2.4523E+217	2.4523E+217
	300	7.1857E+259	7.1857E+259
0.00100	50	4.2372E+03	4.2372E+03
	100	4.1128E+03	4.1128E+03
	150	4.0807E+03	4.0807E+03
	200	4.0703E+03	4.0703E+03
	250	4.0667E+03	4.0667E+03
	300	4.0654E+03	4.0654E+03
0.00050	50	4.6992E+03	4.6992E+03
	100	4.2393E+03	4.2393E+03
	150	4.1518E+03	4.1518E+03
	200	4.1132E+03	4.1132E+03
	250	4.0924E+03	4.0924E+03
	300	4.0808E+03	4.0808E+03
0.00010	50	9.1925E+03	9.1925E+03
	100	6.8748E+03	6.8748E+03
	150	5.6868E+03	5.6868E+03
	200	5.0579E+03	5.0579E+03
	250	4.7134E+03	4.7134E+03
	300	4.5171E+03	4.5171E+03
0.00005	50	3.3268E+47	3.3268E+47
	100	9.7480E+89	9.7480E+89
	150	2.8563E+132	2.8563E+132
	200	8.3694E+174	8.3694E+174
	250	2.4523E+217	2.4523E+217
	300	7.1857E+259	7.1857E+259
0.00001	50	4.2372E+03	4.2372E+03
	100	4.1128E+03	4.1128E+03
	150	4.0807E+03	4.0807E+03
	200	4.0703E+03	4.0703E+03
	250	4.0667E+03	4.0667E+03
	300	4.0654E+03	4.0654E+03

We can see that when learning rate is 0.001 and the number of epochs is 250, the loss curve can converge best and fast.

epoch	Train loss	Test loss
1st	93517.2296	93517.2296
10th	8222.1024	8222.1024
20th	4763.9566	4763.9566
30th	4252.8098	4252.8098
40th	4082.9357	4082.9357
50th	4006.0401	4006.0401
60th	3963.2036	3963.2036
70th	3935.7319	3935.7319
80th	3916.5539	3916.5539
90th	3902.4888	3902.4888
110th	3891.8628	3891.8628
120th	3883.6808	3883.6808
130th	3877.2977	3877.2977
140th	3872.2704	3872.2704
150th	3868.2818	3868.2818
160th	3865.0986	3865.0986
170th	3862.5453	3862.5453

180th	3860.4882	3860.4882
190th	3858.8241	3858.8241
200th	3857.4726	3857.4726
210th	3856.3709	3856.3709
220th	3855.4693	3855.4693
230th	3854.7284	3854.7284
240th	3854.1173	3854.1173
250th	3853.6111	3853.6111



IV. CONCLUSION

Obviously, the final loss of closed-form solution is less than which comes from gradient descent solution. However, in most cases, it's impossible to use this solution as many matrices are not invertible. Even invertible, when they are so complex that calculate their invertible matrices will cost much time and lots of memory for the matrix product.

Therefore, we need some more common solutions for most cases. Gradient descent solution is one of them. Calculating the inverse of matrix is unnecessary which can reduce a lot of computation. On the other hand, gradient descent solution can only find the locally optimal solution, but not the global optimal solution. It needs more improvement.