Linear Regression and Polynomial Regression with Gradient descent

Zehua Zhao

School of Software Engineering, Chongqing University, Chongqing 400044, China

1 Linear Regression

1.1 univariate linear regression

This algorithm only has one variable with the variable being x.

Hypothesis: $h_\theta(x) = \theta_0 + \theta_1 x$

θ_i is the parameter of the model. We want to choose θ_0 and θ_1 so that $h_\theta(x)$ is close to y for our training examples (x, y). So, it is a minimization problem, the objective function is defined as:

$$\min_\theta \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 \tag{1}$$

m is the size of the training examples, $h_\theta(x^{(i)})$ is the predicted value and $y^{(i)}$ is the real value.

1.2 multivariate linear regression

In fact, we usually have more than one variable, so we propose the multivariate linear regression.

Hypothesis: $h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \cdots + \theta_n x_n$

This model is similar with the univariate linear regression, but we have n variables here. The objective is defined as:

$$\min_\theta \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 \tag{2}$$

2 Polynomial Regression

Sometimes, the linear regression does not fit the data well, so we propose the polynomial regression, which allows us to use the machinery of linear regression to fit very complicated, even very non-linear functions.

Hypothesis: $h_\theta(x) = \theta_0 + \theta_1 x^1 + \theta_2 x^2 + \cdots + \theta_n x^n$

In order to solve the model, we could set x^i as x_i:

$$\begin{aligned} h_\theta(x) &= \theta_0 + \theta_1 x^1 + \theta_2 x^2 + \cdots + \theta_n x^n \\ &= \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n \end{aligned} \tag{3}$$

3 Gradient Descent

We often use Gradient Descent to learn the parameters.

3.1 description

Algorithm 1 Gradient Descent

Input: $J(\theta)$
Output: $\min_{\theta} J(\theta)$
1: Start with a $\theta = (\theta_0, \theta_1, \cdots, \theta_n)$
2: Keep changing $\theta = (\theta_0, \theta_1, \cdots, \theta_n)$ to reduce $J(\theta)$ until we hopefully end up at a minimum

3.2 Gradient Descent for linear regression

Algorithm 2 Gradient Descent for linear regression

Input: $J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$,$\theta = (0, 0, \cdots, 0)$,$\alpha$
Output: $\min_{\theta} J(\theta)$
$\frac{\partial}{\partial\theta_j} = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$
2: **repeat**
$\theta_j = \theta_j - \alpha\frac{\partial}{\partial\theta_j}$
4: **until** convergence

α is the learning rate. If α is too small, gradient descent can be slow, if it is too large, gradient descent can overshoot the minimum, it may fail to converge or even diverge. And because the features are too large, in order to make sure the features are on a similar scale, we need to do the feature scaling first.

4 Experiment

4.1 setup

We set $\alpha = 0.5$ and the iterations is 1000. The metric is the sums of squared error. And we choose 20% examples as the test dataset, and 80% as the training dataset.

4.2 results

The sums of squared error of linear regression is 11.64640378, and the sums of squared error of polynomial regression is 10.14587703.

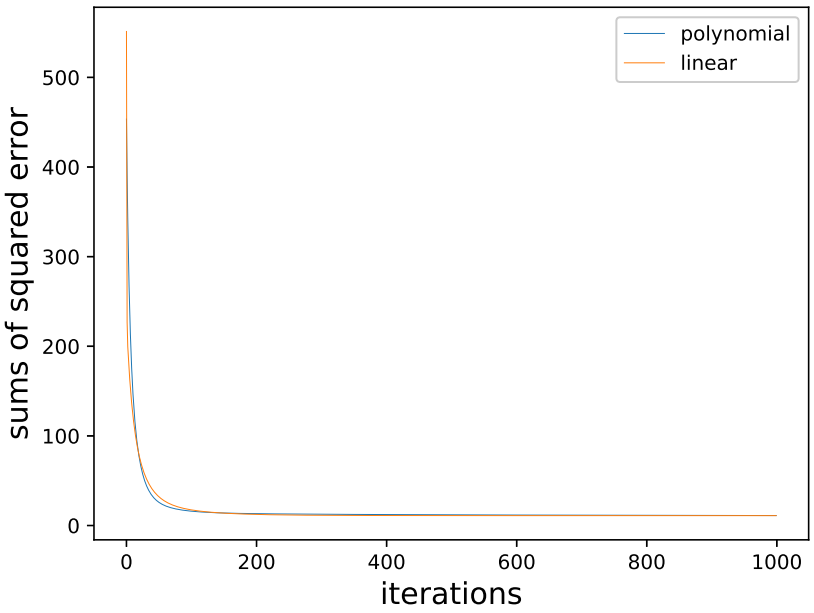

Fig. 1: sums of squared error