SOFT COMPUTING : ISC0630E

ASSIGNMENT 2 : IIT2016067(Maanas Vohra)

Question Description

1. Use housing price dataset. Using various parameters in the datasets we need to predict the housing price.

    i. Predict housing price using normal equations with regularizer.

    ii. Predict housing price using gradient descent with regulariser.

    iii. Compare the results for both approaches

Design alpha and lambda parameters so that hypothesis will perform better.

Case 1: Use all dataset for training.

Case 2: 70% to design hypothesis and 30% to test hypothesis

Find the differences and conclude analysis

2. Use HP data to implement LWR .You may take neighbouring batch size of data.Discuss what will happen when tau is very small.

Introduction

We are given a dataset comprising of housing prices with various features(continuous as well as categorical), for which we need to find the parameters that will help us find an appropriate hypothesis for the model.

Hypothesis Function

$$h(x) = \sum_{i=0}^{n} \theta_i x_i = \theta^T x,$$

$$X = \begin{bmatrix} — (x^{(1)})^T — \\ — (x^{(2)})^T — \\ \vdots \\ — (x^{(m)})^T — \end{bmatrix}. \qquad \vec{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}.$$

The above formula approximates y as a linear function of x, called hypothesis. The hypothesis considers both Θ and x as (n + 1) size vectors.

(1) X = m X (n + 1) matrix, containing the training samples features in the rows, where each $x^{(i)}$ is a (n + 1) X 1 matrix.
(2) m = number of training samples
(3) Θ_i = the parameters
(4) x_0 = 1 for every sample.
(5) n = features count, not including x_0 for every training sample
(6) Y contains all the m target values from the training samples, so it's m X 1 matrix

Cost Function

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

In order to pick the right parameters, we want the hypothesis values for each training example to be close to y. Thus the cost function(similar to least square cost function) helps to identify if the parameters that we've chosen give the minimum cost value or not.

$$X\theta - \vec{y} = \begin{bmatrix} (x^{(1)})^T\theta \\ \vdots \\ (x^{(m)})^T\theta \end{bmatrix} - \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(m)} \end{bmatrix}$$

$$= \begin{bmatrix} h_\theta(x^{(1)}) - y^{(1)} \\ \vdots \\ h_\theta(x^{(m)}) - y^{(m)} \end{bmatrix}.$$

$$J(\theta) = \frac{1}{2m}(X\theta - \vec{y})^T(X\theta - \vec{y})$$

$$\frac{1}{2}(X\theta - \vec{y})^T(X\theta - \vec{y}) = \frac{1}{2}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$
$$= J(\theta)$$

Cost Function With Regularization

With regularization

$J(\theta) = \frac{1}{2m}(X\theta - \vec{y})^T(X\theta - \vec{y}) + \lambda/(2*m) * \sum_{j=1}^{n}\theta_j^2$

$J(\theta) = \frac{1}{2m}(X\theta - \vec{y})^T(X\theta - \vec{y}) + \lambda/(2*m) * \text{sum}(\theta(2{:}n).\hat{}2)$

In the cost function we add an additional “regularization” term in order to avoid overfitting of features. Overfitting happens when the model fits the training data more than required, and thus will lead to more error when we test it for a sample from the test data.

The regularization term will thus add to the cost and will “reduce” the effect of the parameters will contribute towards overfitting.

Gradient Descent

We want to choose Θ so as to minimise the cost function. We choose an initial value of Θ and then repeatedly make changes to it so that it minimises the cost function. This is a very natural algorithm that repeatedly takes a step in the direction of steepest decrease of J.

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta).$$

For every i = 0, 1, 2, … , n

α = Learning rate

$$\begin{aligned}
\frac{\partial}{\partial \theta_j} J(\theta) &= \frac{\partial}{\partial \theta_j} \frac{1}{2} \left(h_\theta(x) - y\right)^2 \\
&= 2 \cdot \frac{1}{2} \left(h_\theta(x) - y\right) \cdot \frac{\partial}{\partial \theta_j} (h_\theta(x) - y) \\
&= \left(h_\theta(x) - y\right) \cdot \frac{\partial}{\partial \theta_j} \left(\sum_{i=0}^{n} \theta_i x_i - y \right) \\
&= \left(h_\theta(x) - y\right) x_j
\end{aligned}$$

The derivative of the cost function(in the case of one training sample) is used in the gradient descent. Thus, for one training sample we have the LMS update rule:

$$\theta_j := \theta_j + \alpha \left(y^{(i)} - h_\theta(x^{(i)})\right) x_j^{(i)}.$$

For m samples of the training set we have,

Repeat until convergence {

$$\theta_j := \theta_j + \alpha \sum_{i=1}^{m} \left(y^{(i)} - h_\theta(x^{(i)})\right) x_j^{(i)} \qquad \text{(for every } j\text{)}.$$

}

$$\theta := \theta - \alpha \nabla J(\theta) \qquad \theta := \theta - \frac{\alpha}{m} X^T (X\theta - \vec{y})$$

Gradient

$$\nabla J(\theta) = \frac{1}{m} X^T (X\theta - \vec{y})$$

The convergence arises when $J_t(\Theta) - J_{t+1}(\Theta) <= \text{ERROR\_THRESHOLD}$

The above method is batch gradient descent, since at every step we look at all the samples of the training set.

Since the cost function is convex in nature, the local minima obtained for the cost function would be the global minima.

Gradient Descent with Regularization

With regularization

Gradient

$$\nabla J(\theta) = \frac{1}{m} X^T (X\theta - \vec{y}) + \lambda/m * \theta\ (2{:}n)$$

$$\theta := \theta - \alpha \nabla J(\theta)$$

By differentiating the cost function we get the above result for gradient descent. This reduces the effect of those features that contribute to overfitting. This does depend on taking an appropriate value of λ though. If the value of λ is too high, then it reduces the effect of almost every feature, and thus it will lead to underfitting of the data. If it is too low, then it doesn't have much effect and there is a chance that overfitting is still there.

Normal Equation

Since we want to minimise the cost function, we can set the derivative of the cost function to zero.

$$
\begin{aligned}
\nabla_\theta J(\theta) &= \nabla_\theta \frac{1}{2}(X\theta - \vec{y})^T (X\theta - \vec{y}) \\
&= \frac{1}{2}\nabla_\theta \left(\theta^T X^T X\theta - \theta^T X^T \vec{y} - \vec{y}^T X\theta + \vec{y}^T \vec{y}\right) \\
&= \frac{1}{2}\nabla_\theta \operatorname{tr}\left(\theta^T X^T X\theta - \theta^T X^T \vec{y} - \vec{y}^T X\theta + \vec{y}^T \vec{y}\right) \\
&= \frac{1}{2}\nabla_\theta \left(\operatorname{tr}\theta^T X^T X\theta - 2\operatorname{tr}\vec{y}^T X\theta\right) \\
&= \frac{1}{2}\left(X^T X\theta + X^T X\theta - 2X^T \vec{y}\right) \\
&= X^T X\theta - X^T \vec{y}
\end{aligned}
$$

$$\theta = (X^T X)^{-1} X^T \vec{y}.$$

The challenges here are :

(1) Non-invertibility of X^TX matrix

(2) High complexity of finding the inverse of a matrix : $O(N^3)$

Normal Equation with Regularization

If $\lambda > 0$,

$$\theta = \left(X^T X + \lambda \begin{bmatrix} 0 & & & \\ & 1 & & \\ & & 1 & \\ & & & \ddots \\ & & & & 1 \end{bmatrix} \right)^{-1} X^T y$$

invertible.

The gradient of the cost function is set to zero in order to evaluate the expression for parameters in the vectorized form. The main issue of non-invertibility is still there in this scenario.

Locally Weighted Linear Regression

This method makes the choice of features less critical.

We need to minimise the following cost function:

$$\min_{\theta} \sum_{j=1}^{m} w^j \left[h_\theta(x^j) - y^j \right]^2.$$

Where the weights w^j are generally represented by:

$$w^{(i)} = \exp\left(-\frac{(x^{(i)} - x)^2}{2\tau^2} \right)$$

Here, x represents the input which has only one feature. For (n + 1) features for which x and each input sample from the training data will be represented in form of vectors, we have :

$$w^{(i)} = \exp(-(x^{(i)} - x)^T (x^{(i)} - x)/(2\tau^2)),$$

Where τ represents the bandwidth which means how quickly the weight of a training example falls off with distance of its $x^{(i)}$ from the query point x.

$$W = \begin{bmatrix} w^1 & 0 & 0 \\ 0 & w^2 & 0 \\ \vdots & \ddots & \vdots \\ 0 & 0 & w^m \end{bmatrix}$$, where $w^{(i)}$ represents weight for i^{th} sample

$$J(\theta) = \frac{1}{2} \sum_{j=1}^{m} w^j \left[h_\theta(x^j) - y^j \right]^2 = \frac{1}{2} (X\theta - Y)^T W (X\theta - Y)$$

Gradient of cost function, normal equation and gradient descent

The gradient of the cost function can be used in gradient descent and normal equation.

$$\begin{aligned}\frac{\partial}{\partial\theta}J(\theta) &= \frac{1}{2}\frac{\partial}{\partial\theta}(X\theta-Y)^T W(X\theta-Y)\\ &= \frac{1}{2}\frac{\partial}{\partial\theta}(\theta^T X^T W X\theta-\theta^T X^T W Y-Y^T W X\theta+Y^T W Y)\}\\ &= (X^T W X\theta-X^T W Y)\end{aligned}$$

The gradient in the above equation can be used for gradient descent.

For normal equation:

$$\frac{\partial}{\partial\theta}J(\theta)=0,$$

$$(X^T W X\theta-X^T W Y)=0.$$

$$\theta = (X^T W X)^{-1} X^T W Y$$

We can also apply regularization in order to avoid overfitting.

Results:

For linear regression with regularization :

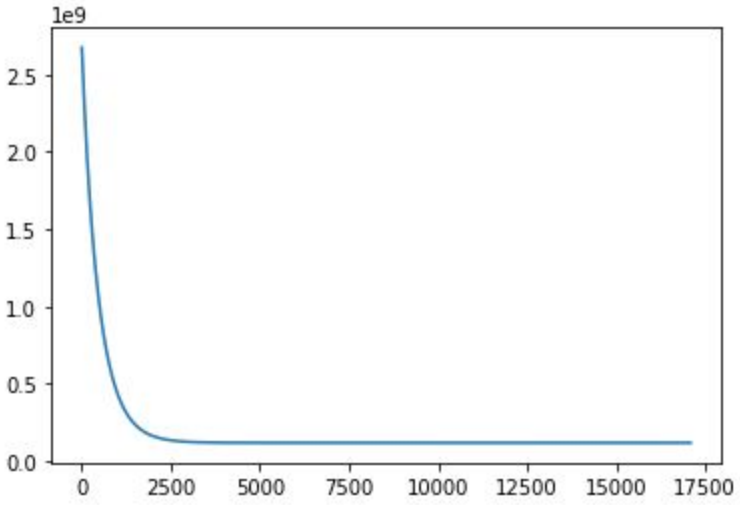
We take λ = 0.1, α = 0.001, ERROR_THRESHOLD = 0.001

(1) Gradient Descent:

Cost: 116344468.8618906

Iterations : 17089

Theta Values: [68121.59450896 7680.12586032 1351.52327014 7191.42938284
5685.13213484 2328.35786138 1725.0860881 2598.6506428
2681.62765466 5877.36829972 3652.61184557 3969.23458743



(2) Normal Equation:

Cost Function: 116344467.65890989

Theta Values: [68121.5970696 7680.77961523 1350.49089731 7191.19786875
5686.61602204 2327.86101151 1724.56688369 2599.7448518
2681.52744473 5876.86113663 3652.7041446 3968.9901632]

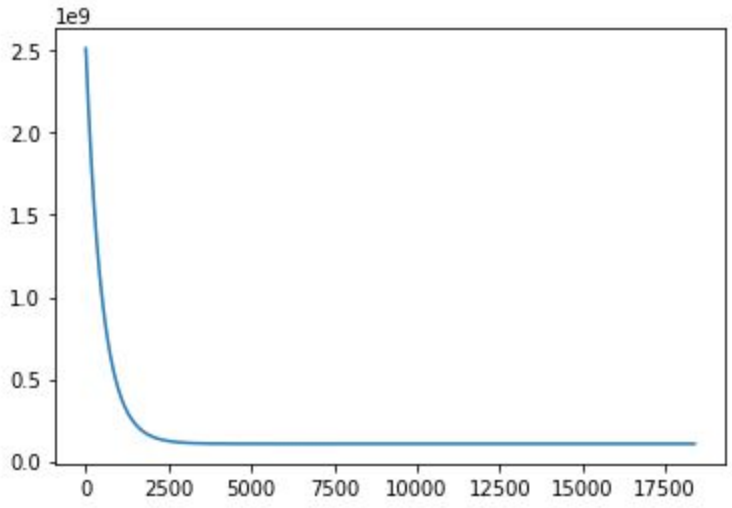
When dataset was randomly divided into 70% training and 30% test:

(1) Gradient Descent:

Cost: 128838076.08846697

Theta Values: [67887.74819169 8749.02470464 1402.86037433 7414.08847368
5688.11357881 2007.60580541 2160.30541606 2212.38046431
1976.3623115 5587.72008534 3382.06779906 3904.94108006]

Iterations: 16259



(2) Normal Equation:

Cost Function: 128838074.90943304

Theta Values: [67887.751878 8749.6842668 1401.92686687 7413.704708
5689.54574534 2007.03755567 2159.88346937 2213.48524141
1976.26849608 5587.27934957 3382.23874065 3904.59981973]

The testing error was recorded as : 90359909.995826

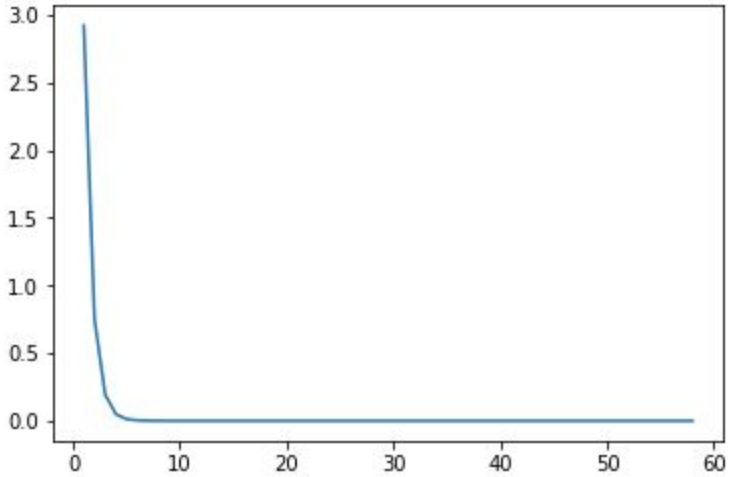
Result from both approaches : i.e. gradient descent and normal equation were almost equal because the cost function is a convex function and thus the local minima is the global minima.

For locally weighted linear regression:

We take $\lambda = 10^{-22}$, $\alpha = 10^{10}$, ERROR_THRESHOLD = 10^{-40}, $\tau = (20 / 546)$

input_var = [1, 4000,2 ,1 ,1 ,1 ,1 ,0 ,0 ,0 ,0 ,0] (converting “yes” to 1, “no” to 0)

Using gradient descent we get:
Total iterations: 58
Final cost: 9.060006210999755e-18
Final theta: [3101.57188523 -2362.88555452 -4063.51498634 -1766.32905256
-2888.0508516 1256.72655661 6672.96972752 -2275.01549192
-679.41165693 -2112.27513615 -2495.29173226 -1716.32170698]
Predicted value : 34454.59427890818



Using normal equation :
Final cost: 0.021671895274336297
Final theta: [3.84000000e+04 -1.28000000e+02 1.00000000e+00 -1.28000000e+02
1.97000000e+02 -6.40000000e+01 -1.46214844e+02 0.00000000e+00
-1.02500000e+01 0.00000000e+00 4.10000000e+01 0.00000000e+00]
Predicted value : 37984.86619125075

Observations:

For a smaller value of ϒ the measured and predicted values are almost on top of each other. By adjusting the meta-parameter we can get a non-linear model that is as strong as polynomial regression of any degree.

The predicted value is close to the curve obtained from the no weighting case when the value of τ is high.

When predicting using the locally weighted least squares case, we need to have the training set handy to compute the weighting function. In contrast, for the unweighted case one could have ignored the training set once the parameters are computed.