SOFT COMPUTING : ISCO630E

Assignment 1

Group Assignment

Group Members:

IIT2016067: Maanas Vohra

IIT2016076: Swapnil Gharat

IIT2016086: Pranav Mahajan

IIT2016506: Anant Chaturvedi

Question Description

We are given a housing price dataset. Using various parameters in the datasets we need to predict the housing price.

i. Predict housing price using normal equations method.

ii. Predict housing price using gradient descent method.

iii. Compare the results for both approaches.

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.

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

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

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

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

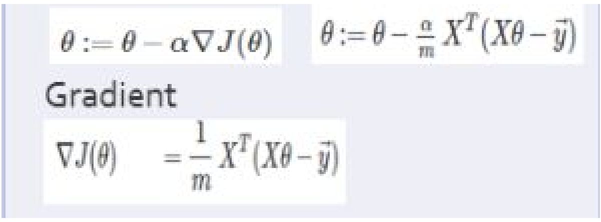
$$
\begin{aligned}
\frac{\partial}{\partial \theta_j} J(\theta) &= \frac{\partial}{\partial \theta_j} \frac{1}{2} \left(h_\theta(x) - y\right)^2 \\
&= 2 \cdot \frac{1}{2} \left(h_\theta(x) - y\right) \cdot \frac{\partial}{\partial \theta_j} \left(h_\theta(x) - y\right) \\
&= \left(h_\theta(x) - y\right)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

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,

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



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

Gradient

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

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.

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 : $O(N^3)$

Result

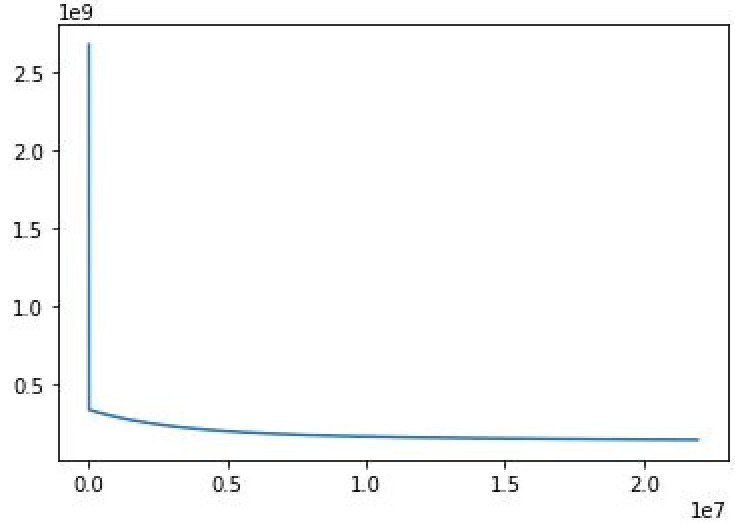
For the values of α = 0.00000006, ERROR_THRESHOLD = 1

For Linear Regression:

Iterations: 21941708

Final Cost : 148578628.6529754

Final Theta Values: [1.43972725e+03, 4.67550485e+00, 5.49545140e+03, 5.70824361e+03, 6.63408322e+03, 2.18302619e+03, 1.81306569e+03, 2.59884975e+03, 6.78432566e+02, 3.71615216e+03, 3.34088153e+03, 2.38764377e+03]



For Normal Equation:

```
Final Cost: 116323325.68174723
Final Theta Values: [-4.03835043e+03  3.54630297e+00  1.83200347e+03  1.43355585e+04
  6.55694571e+03  6.68777889e+03  4.51128383e+03  5.45238554e+03
  1.28314063e+04  1.26328904e+04  4.24482900e+03  9.36951324e+03]
```