EEC1509 - Machine Learning

Lesson #5 - Linear Regression with Multiple Variables

Ivanovitch Silva

September, 2018

Update repository

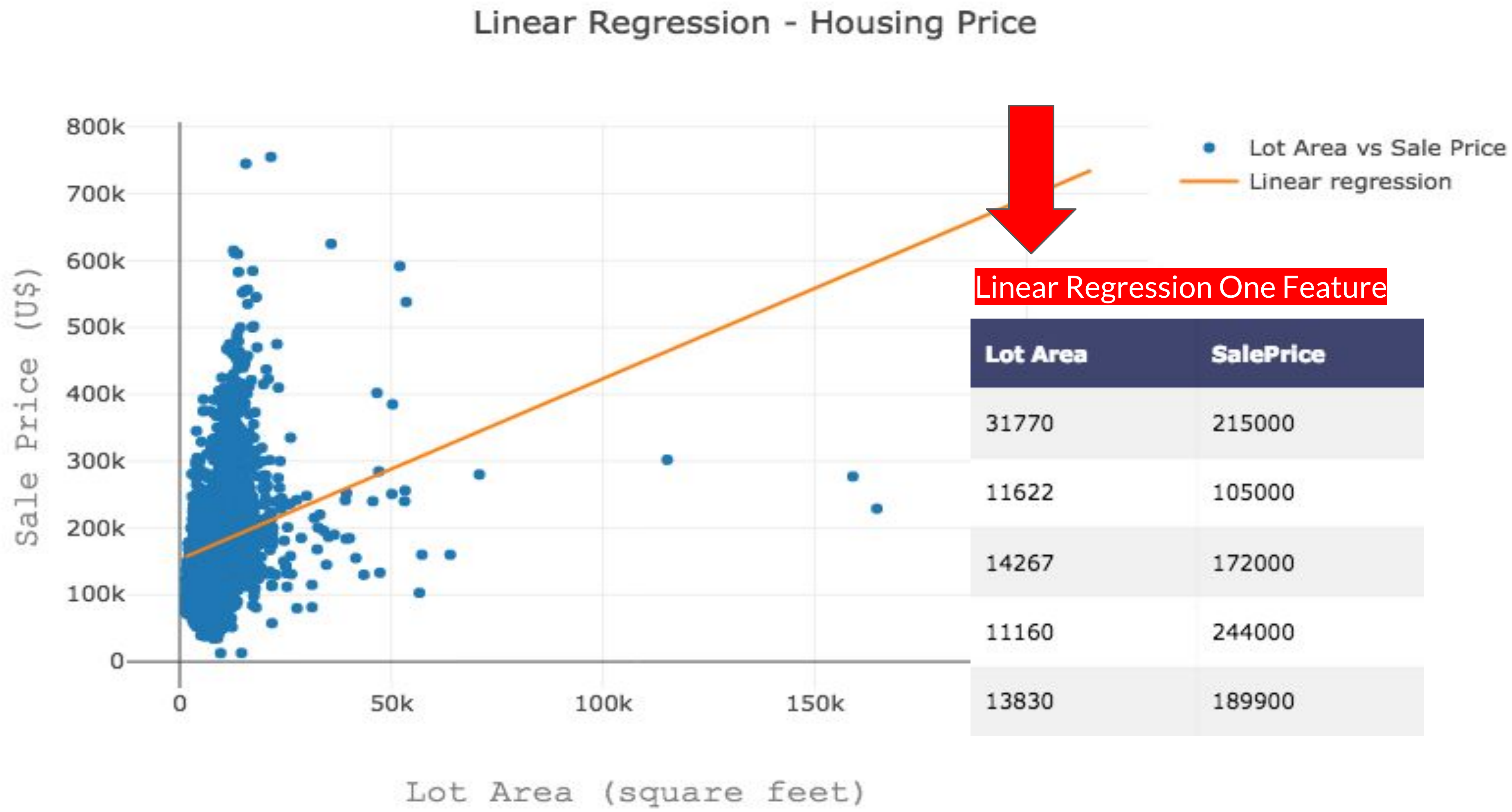
```
git clone https://github.com/ivanovitchm/EEC1509_MachineLearning.git
```

Ou ....

```
git pull
```

PREVIOUSLY ON...

lesson #4



Linear Regression One Feature

Lot Area	SalePrice
31770	215000
11622	105000
14267	172000
11160	244000
13830	189900

Training Set (m instances)

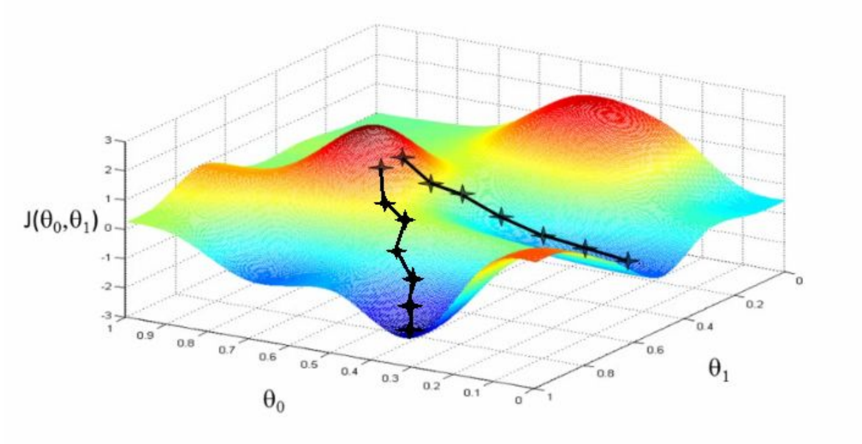
	Lot Area	SalePrice	
$x^{(1)}$	31770	215000	$y^{(1)}$
$x^{(2)}$	11622	105000	$y^{(2)}$
$x^{(3)}$	14267	172000	$y^{(3)}$
$x^{(4)}$	11160	244000	$y^{(4)}$
$x^{(5)}$	13830	189900	$y^{(5)}$

Cost Function

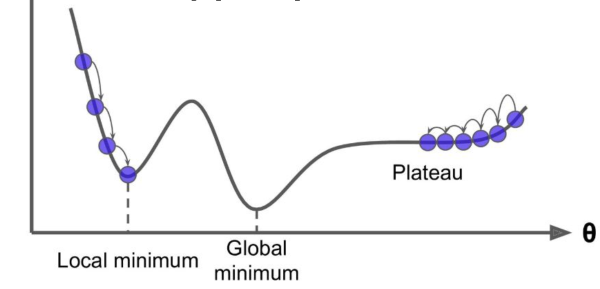
$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left[h_\theta(x^{(i)}) - y^{(i)}\right]^2$$

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

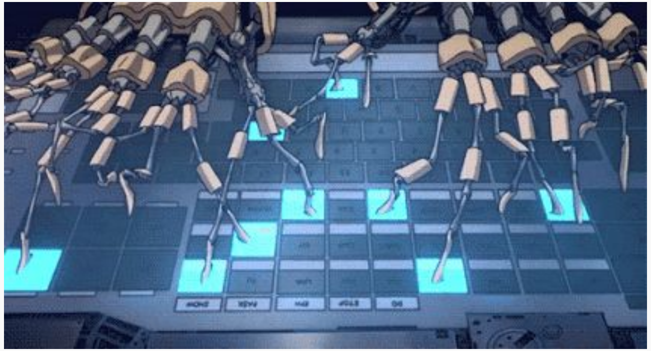
Gradient Descent



Hyperparameters



Stochastic Gradient Descent
Mini-batch Gradient Descent



- We are going to start by covering linear regression
  - Multiple variables
- We discuss the application of linear regression to **housing price prediction**

Linear Regression with Multiple Variables

Notation:

- m - number of training examples
- n - number of features
- $x^{(i)}$ - input features of i^{th} training example
- $x_j^{(i)}$ - value of feature j in i^{th} training example
- $y^{(i)}$ - target value of i^{th} training examples

$$x^{(2)} = \begin{bmatrix} 11622 \\ 5 \\ 1961 \\ 2010 \\ 105000 \end{bmatrix}$$

$$x_3^{(2)} = 1961$$

n = 4

m = 5

x_1	x_2	x_3	x_4	y
Lot Area	**Overall Qual**	**Year Built**	**Yr Sold**	**SalePrice**
31770	6	1960	2010	215000
11622	5	1961	2010	105000
14267	6	1958	2010	172000
11160	7	1968	2010	244000
13830	5	1997	2010	189900

Hypothesis (previously)

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

Multivariable case

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4$$

For convenience of notation, define $x_0=1$. In other words:

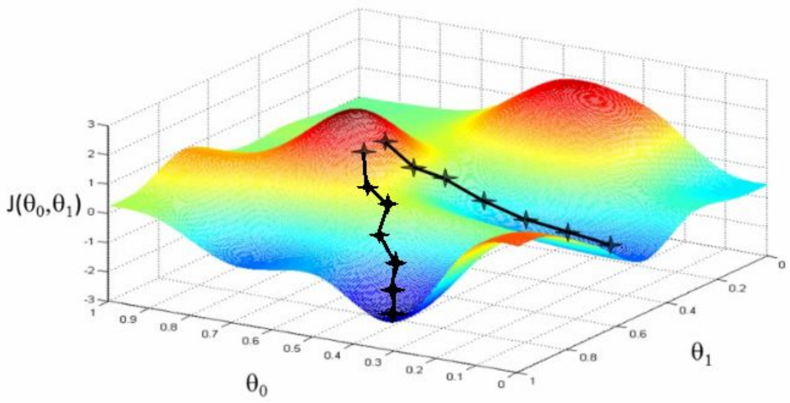
$x_0^{(i)}=1$

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4$$

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^{n+1} \qquad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix} \in \mathbb{R}^{n+1}$$

$$h_\theta(x) = \begin{bmatrix} \theta_0 & \theta_1 & \theta_1 & \dots & \theta_n \end{bmatrix} \times \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \theta^T x$$

Gradient Descent
(linear reg. multiple variables)



Hypothesis: $h_\theta(x) = \theta^T x = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_n x_n$

Parameters: $\theta_0, \theta_1, \theta_2, \ldots, \theta_n$

Cost function:

$$J(\theta_0, \theta_1, \theta_2, \ldots, \theta_n) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

Gradient descent:

repeat {

$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \ldots, \theta_n)$$

} Simultaneously update for every j (0 to n)

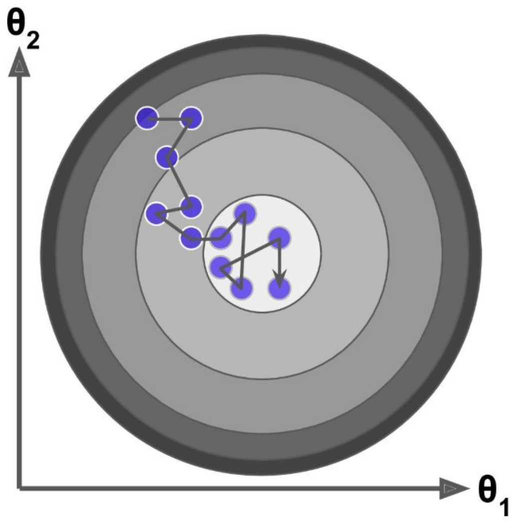
Gradient descent:

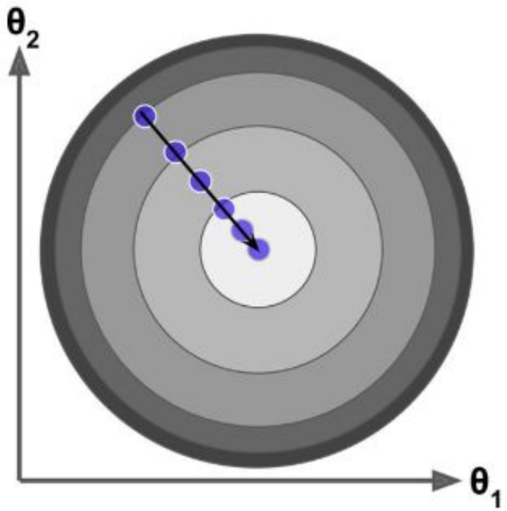
repeat until the convergence {

$$\theta_j = \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} \qquad \text{for } j = 0, \ldots, n$$

}

Gradient Descent: trick #1 - Feature Scaling





Gradient Descent: trick #1 - Feature Scaling

Z-Score or Standardization

$$z = \frac{x - \mu}{\sigma}$$

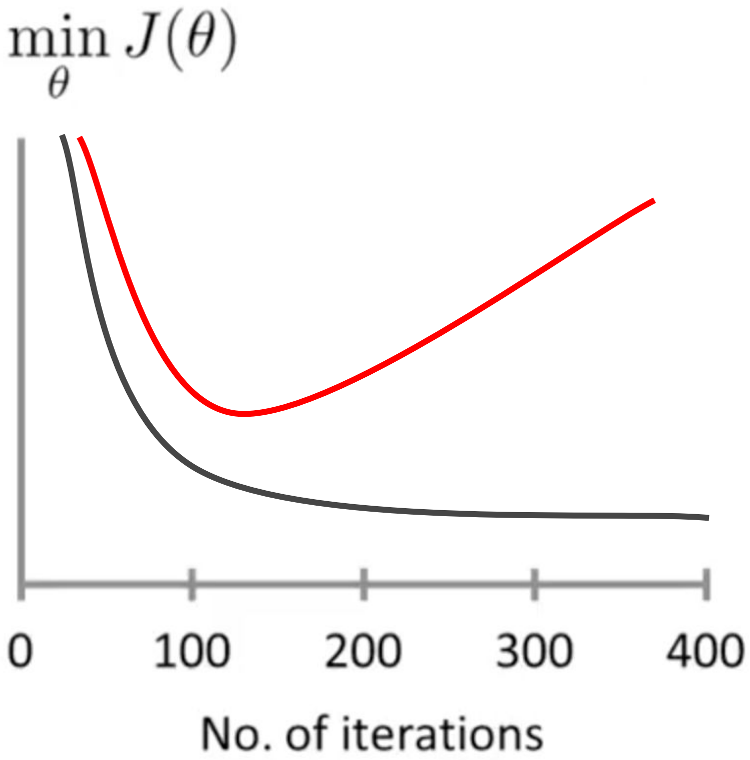
μ - 0
σ - 1

Min-Max Scaling

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

range (0,1)

Gradient Descent: trick #2 - Debugging α



1. Make a plot with number of iterations on the x-axis.
2. Now plot the cost function, J(θ) over the number of iterations of gradient descent.
3. If J(θ) ever increases, then you probably need to decrease α.
4. It has been proven that if learning rate α is sufficiently small, then J(θ) will decrease on every iteration.
5. Automatic convergence test

Gradient Descent: trick #2 - Debugging α

- If α is too small:
  - Slow convergence
- If α is too large:
  - $J(\Theta)$ may not decrease on every iteration;
  - $J(\Theta)$ may not converge.

To choice α:
…, 0.0001, ….., 0.001, …, 0.01, …, 0.1, …, 1, …, 10, ...

normal equation: method to solve for θ analytically

Normal Equation

Lot Area	Overall Qual	Year Built	Yr Sold	SalePrice
31770	6	1960	2010	215000
11622	5	1961	2010	105000
14267	6	1958	2010	172000
11160	7	1968	2010	244000
13830	5	1997	2010	189900

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

$$\begin{bmatrix} 1 & 31770 & 6 & 1960 & 2010 \\ 1 & 11622 & 5 & 1961 & 2010 \\ 1 & 14267 & 6 & 1958 & 2010 \\ 1 & 11160 & 7 & 1968 & 2010 \\ 1 & 13830 & 5 & 1997 & 2010 \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \\ \theta_5 \end{bmatrix} = \begin{bmatrix} \theta_0 + 31770\theta_1 + 6\theta_2 + 1960\theta_3 + 2010\theta_4 \\ \theta_0 + 11622\theta_1 + 5\theta_2 + 1961\theta_3 + 2010\theta_4 \\ \theta_0 + 14267\theta_1 + 6\theta_2 + 1958\theta_3 + 2010\theta_4 \\ \theta_0 + 11160\theta_1 + 7\theta_2 + 1968\theta_3 + 2010\theta_4 \\ \theta_0 + 13830\theta_1 + 5\theta_2 + 1997\theta_3 + 2010\theta_4 \end{bmatrix}$$

There is **no need** to do feature scaling with the normal equation.

The following is a comparison of gradient descent and the normal equation:

Gradient Descent	Normal Equation
Need to choose alpha	No need to choose alpha
Needs many iterations	No need to iterate
O (kn^2)	O (n^3), need to calculate inverse of X^TX
Works well when n is large	Slow if n is very large

With the normal equation, computing the inversion has complexity $\mathcal{O}(n^3)$. So if we have a very large number of features, the normal equation will be slow. In practice, when n exceeds 10,000 it might be a good time to go from a normal solution to an iterative process.

If $X^T X$ is **noninvertible,** the common causes might be having :

- Redundant features, where two features are very closely related (i.e. they are linearly dependent)
- Too many features (e.g. m ≤ n). In this case, delete some features or use "regularization" (to be explained in a later lesson).

Exercises

- Complete the notebook "Lesson 5 exercise.ipynb"
  - plus++
    - Transform the Section 7 into a Scikit-Learn's Pipeline
    - Implement gradient descent for multiple variables
    - Implement normal equation