Multivariate Linear Regression

Notation:

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

Hypothesis:

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_n x_n$$

- For convenience of notation, define $x_0 := 1$. ($x_0^{(i)} = 0$)

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

- Then:

$$h_\theta = \boldsymbol{\theta}^T \mathbf{x}$$

- Multivariate Linear Regression.

Gradient Descent for Multiple Variables

- Cost function:

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

- Gradient descent:
  - Repeat:

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

(simultaneously update for every $j = 0, 1, \ldots, n$).

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_j^{(i)}$$

Feature Scaling

Idea: make sure features are on a similar scale.

E. g.:

- x_1: size (0 - 2000 squared feet)
- x_2: number of bedrooms (1-5)

They can rescaled to:

- x_1: $\dfrac{\text{size (sq. feet)}}{2000}$;
- x_2: $\dfrac{\text{num. of bedrooms}}{5}$.

Get every feature into approximately a $-1 \leq x_i \leq 1$ range.

Mean Normalization

Replace x_i with $x_i - \mu_i$ to make features have approximately zero mean. (Do not apply to $x_0 = 1$.)

E. g.:

- x_1: $\dfrac{\text{size} - 1000}{2000}$;
- x_2: $\dfrac{\#\ \text{bedrooms} - 2}{5}$.

$\Rightarrow \quad -0.5 \leq x_1 \leq 0.5$, $-0.5 \leq x_1 \leq 0.5$.

Generally:

$$x_i := \frac{x_i - \mu_i}{S_i},$$

where μ_i is the average value for the feature and S_i , the range of values (maximum minus minimum).

Learning Rate

- Making sure gradient descent is working correctly:
  - Plot $\underset{\boldsymbol{\theta}}{min}\ J(\boldsymbol{\theta})$ against the number of iterations;
  - $J(\boldsymbol{\theta})$ should decrease after every iteration!
  - Otherwise, try using a smaller α .
- For a sufficiently small α , $J(\boldsymbol{\theta})$ should decrease on every iteration;
  - But if α is too small, gradient descent can be slow to converge.

Features and Polynomial Regression

Example:

Housing prices prediction:

$$h_\theta(\mathbf{x}) = \theta_0 + \theta_1 \cdot \text{frontage} + \theta_2 \cdot \text{depth}$$

- Defining new features...

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

where x is the land area (*i. e.*, frontage times depth).

- Adding non-linear features:

$$\begin{gathered} h_\theta(\mathbf{x}) = \theta_0 + \theta_1 \cdot x_1 + \theta_2 \cdot x_2 + \theta_3 \cdot x_3 \\ = \theta_0 + \theta_1 \cdot (\text{size}) + \theta_2 \cdot (\text{size})^2 + \theta_3 \cdot (\text{size})^3 \end{gathered}$$

(Only adding the quadratic term would imply that, with a sufficiently large value for size, the price would go down.)

Feature scaling becomes even more crucial:

$$\text{size } \in [1, 1000] \Rightarrow (\text{size})^2 \in [1, 10^6], (\text{size})^3 \in [1, 10^9]$$

- An alternative non-linear version:

$$h_\theta(\mathbf{x}) = \theta_0 + \theta_1(\text{size}) + \theta_2\sqrt{(\text{size})}$$

Normal Equation

Method to solve $\underset{\theta}{min}\ J(\boldsymbol{\theta})$ for $\boldsymbol{\theta}$ analytically.

- Intuition: for one dimension ($\theta \in \mathbb{R}$):

$$\frac{\partial}{\partial\theta}J(\theta) = \frac{\partial}{\partial\theta}(a\theta^2 + b\theta + c) := 0$$

- $\theta \in \mathbb{R}^{n+1}$:

$$\frac{\partial}{\partial\theta_j}J(\theta_0, \theta_1, \ldots \theta_m) = \frac{\partial}{\partial\theta_j}\left(\frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2\right) := 0,$$
$$\forall j \in \{0, 1, \ldots, m\}$$

Example:

x_0	Size (sq. ft.) x_1	# of bedrooms x_2	# of floors x_3	Age of home x_4	Price in $ 1000s (y)
1	2104	5	1	45	460
1	1416	3	2	40	232
1	1534	3	2	30	315
1	852	2	1	36	178

$$X = \begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{bmatrix} \qquad y = \begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \end{bmatrix}$$

Then:

$$\boldsymbol{\theta} = (X^TX)^{-1}X^Ty$$

Generally:

- m examples: $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})$;
- n features:

$$x^{(i)} = \begin{bmatrix} x_0^{(i)} \\ x_1^{(i)} \\ x_2^{(i)} \\ \vdots \\ x_n^{(i)} \end{bmatrix} \in \mathbb{R}^{n+1} \quad \Rightarrow \quad X = \begin{bmatrix} -(x^{(1)})^T - \\ -(x^{(2)})^T - \\ \cdots \\ -(x^{(m)})^T - \end{bmatrix} \quad y = \begin{bmatrix} y^1 \\ y^2 \\ \vdots \\ y^m \end{bmatrix}$$

where $X_{m\times(n+1)}$ is called a design matrix.

So, $\boldsymbol{\theta}$ can be computed as: $\boldsymbol{\theta} = (X^TX)^{-1}X^Ty$.

```
theta = pinv(X' * X) * X' * y
```

- When using the normal equation, feature scaling is not necessary.

Comparison:

Gradient Descent	**Normal Equation**
Need to choose α	No need to choose α
Needs many iterations	Don't need to iterate
Works well even when the number of features (n) is large, $e.\ g.$, $n \geq 10^4$	Slow if n is very large (due to $(X^TX)^{-1}$)

- What if X^TX is non-invertible?
    - Redundant features (linearly dependent):
        - $E.\ g.$: x_1: size in sq. ft., x_2: size in m^2;
        - Delete one the features.
    - Too many features ($e.\ g.$, $m \leq n$):
        - Delete some features, or use regularization.