Coursera Machine Learning Notes

Eddie Shim

September 27, 2016

1 Regression

1.1 Linear Regression

To perform a univariate linear regression, take a dataset of two variables and optimize the linear equation $y = \theta_1 x + \theta_0$. Use the least squares method which minimizes the convex parabolic equation $\sum_{i=1}^{m}(f(x) - y)^2$, where $(f(x)) - y)^2$ represents the residual squared. To find the minimal $\vec{\theta} = [\theta_0; \theta_1]$, calculate where the derivative of the equation equals 0.

- **Cost function for linear regression** (where $\vec{\theta} \in \mathbb{R}^{n+1}$, where n is the number of independent variables):

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

- **Gradient descent** (Note: must update all θ_j simultaneously):

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

  ex: for 2 variable case:

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})$$
$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x^{(i)}$$

- **Feature scaling**: normalizing all independent variables to an appropriate range. Purpose is to speed up gradient descent.

$$x_i = \frac{x_i - \mu}{\text{range of } x}$$

- Note: if α is too large, gradient descent may skip minimum point. However, if α is too little, it may take too long to converge.

- **Normal equation** an alternative to gradient descent:

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

Gradient Descent	**Normal Equation**
- need to choose an α - needs many iterations - works well even when n is large	- no need for α - no need to iterate, one calculation - need to compute $(X^T X)^{-1}$ which runs in $O(n^3)$ runtime - slow if n is very large

Table 1: Pros and Cons

- **Vectorization**: to speed up loops.
  e.g. Transform the following code from:

```
for i = 1:3
   for j = 1:m
      theta(i) := theta(i) - alpha * (1/m)*(h_theta(x(j))-y(j))*x(i);
   end
end
```

  into:

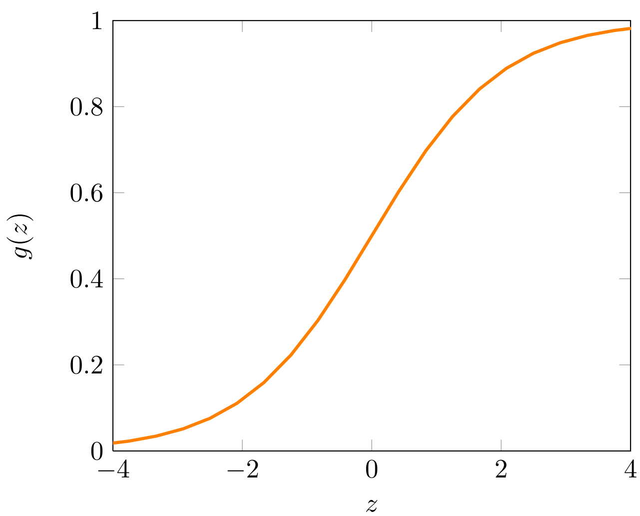
```
theta = theta - alpha * delta;
```

  where $\theta \in \mathbb{R}^{n+1},\ \alpha \in \mathbb{R},\ \delta \in \mathbb{R}^{n+1}$

1.2 Logistic Regression

- A classification algorithm (not really regression, which predicts continuous variable given continuous variable)

$$h_\theta(x) = g(\theta^T x),$$
$$\text{where } g(z) = \frac{1}{1+e^{-z}} \quad \text{(sigmoid function)}$$



- Nice properties:
  - $0 \leq h_\theta(x) \leq 1$, good for properties of a probability
  - At $z = 0$, $g(z) = 0.5$
  - Converges to $g(z) = 1$ quickly as g increases, and vice versa for 0
  - We can use these properties to output the probability that an input exists in one of two binary states (1 or 0)
- Terminology: the probability of the output of results equaling 1:

$$h_\theta(x) = p(y = 1|\ x; \theta)$$

- Predict $y = 1$ if $\theta^T x \geq 0 \Leftrightarrow$ if $h_\theta(x) = g(\theta^T x) \geq 0.5$
- Cost Function:

$$J(\vec{\theta}) = \frac{1}{m} \sum_{i=1}^{m} \text{cost}(h_\theta(x), y)$$

  where cost $(h_\theta(x), y) =$

$$\begin{cases} -log(h_\theta(x)) & \text{if } y = 1 \\ -log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$

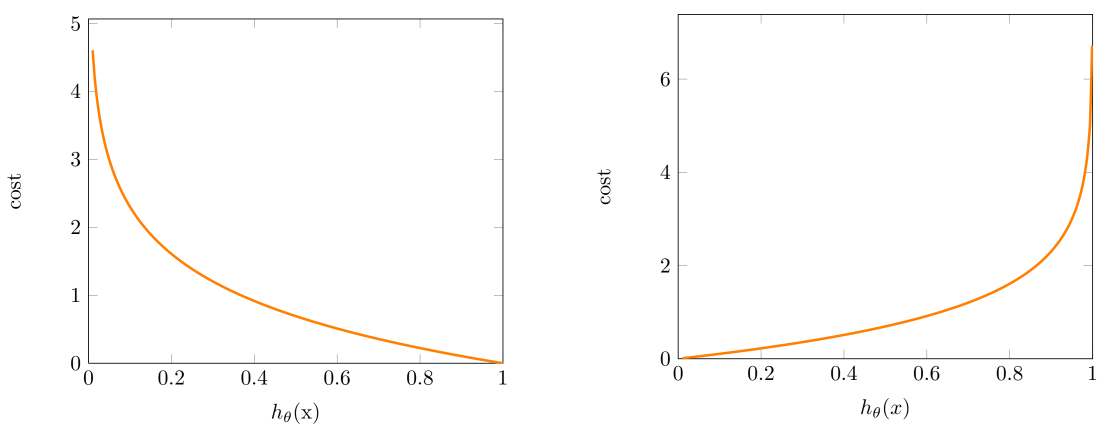


Figure 1: Left: $-log(h_\theta(x))$; Right: $-log(1 - h_\theta(x))$

- We can create a more succinct function instead of using a piecewise function.
  Cost function for logistic regression:

$$\text{cost}(h_\theta(x), y) = -y * log(h_\theta(x)) - (1 - y) * log(1 - h_\theta(x))$$

$$J(\vec{\theta}) = -\frac{1}{m} \Big(\sum_{i=1}^{m} y^{(i)} * log(h_\theta(x^{(i)})) + (1 - y^{(i)}) * log(1 - h_\theta(x^{(i)})) \Big)$$

1.3 Neural Networks