Notes on machine learning[*]

Daniel Miller

August 16, 2016

1 Supervised machine learning

1.1 Univariate linear regression

The basic idea is as follows. We have a set $\mathbf{x} = \{x^{(1)}, \ldots, x^{(m)}\}$ of "input variables," lying in some domain D, a set $\mathbf{y} = \{y^{(1)}, \ldots, y^{(m)}\}$ "output" or "target" variables in some range R, i.e. a map $[1, \ldots, m] \to D \times R$. Given this, we want to select a "hypothesis function" $h\colon D \to R$, such that $h(x) = y$ is a good fit for the data, i.e. $h(x^{(i)}) \approx y^{(i)}$ for some reasonable definition of $\approx$.

Univariate linear regression concerns $D = \mathbf{R}$, $R = \mathbf{R}$, and $h_\theta(x) = \theta_0 + \theta_1 x$. We try to find θ that minimizes the "cost function"

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

The function $J_{\mathbf{x},\mathbf{y}}$ is quadratic in θ, so it should be easy to find the minimum point.

1.2 Gradient descent

Basic idea, we have some function $J(\theta)$ that we would like to minimize. Start with some θ_0, then put $\theta_{n+1} = \theta_n - \alpha \nabla J(\theta_n)$. That is, we walk in the direction that J is decreasing most rapidly.

Apply gradient descent to the cost function $J_{\mathbf{x},\mathbf{y}}$ above. One has:

$$\frac{\mathrm{d}}{\mathrm{d}\theta_0} J_{\mathbf{x},\mathbf{y}}(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})$$

$$\frac{\mathrm{d}}{\mathrm{d}\theta_1} J_{\mathbf{x},\mathbf{y}} J(\theta_0, \theta_1) = \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x^{(i)}$$

[*] From Andrew Ng's Coursera class

So one repeats the following updates until convergence:

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

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

Here, our cost function $J_{\mathbf{x},\mathbf{y}}$ has a single minimum (it is a convex function), so any local optimum is actually global.

This is actually "batch" gradient descent, i.e. each step uses all the training examples $x^{(i)}, y^{(i)}$.

1.3 Multivariate linear regression

Suppose we have multiple "features (variables)" of our data set. That is, we have $\{\mathbf{x}_j\}$. Our hypothesis will be:

$$h_\theta(x) = \sum_{i=0}^{n} \theta_i x_i.$$

For convenience, put $x_0 = 1$. So the "feature vector" is $x = (x_0, \dots, x_n)$, and our "parameter vector" is $\theta = (\theta_0, \dots, \theta_n)$. Our hypothesis is $h_\theta(x) = \theta^{\mathrm{t}} x$. Define the "cost function" $J(\theta)$ and apply gradient descent as above.

It is useful to use *feature scaling*, which ensures that features lie in the range $[-1, 1]$ or something similar. Also, we use *mean normalization*, to ensure all features but $\mathbf{x}_0$ have zero mean. The general rule is:

$$\mathbf{x}_i = \frac{\mathbf{x}_i - \mu_i}{s_i},$$

where μ_i is the mean and s_i is the standard deviation.

It is important to choose the "learning rate" α well. One way to do this is, given α, plot the points $(n, J(\theta_n))$. One can "declare convergence" if $J(\theta)$ decreases by less than some given quantity in one iteration. If $J(\theta_n)$ increases, try using a smaller α.

1.4 Polynomial regression

If we want to make a hypothesis h_θ that depends not just linearly, but polynomially, on $\mathbf{x}$, just set $\mathbf{x}_n = \mathbf{x}^n$ and feature-scale. Really, we can add $\mathbf{x}^\eta$ for any real η.

1.5 Normal equation

This is a method to solve for θ analytically, instead of approximating it. Let X be the matrix $(\mathbf{x}_0, \dots, \mathbf{x}_n)$. Then $\theta = (X^{\mathrm{t}} X)^{-1} X^{\mathrm{t}} y$. The Octave command is `pinv(X'*X)*X'*y`. The normal equation is very slow if n is large, since the cost of inverting an $n \times n$ matrix is $O(n^3)$. If $X^{\mathrm{t}} X$ is non-invertible, then Octave's function `pinv` does the "right" thing anyways. Usually, $X^{\mathrm{t}}X$ is non-invertible if there are redundant (i.e., linearly dependent) features, or too many features ($m \leqslant n$).