‘cs229’—Notes

Malcolm

Started 1st October 2024

Chapter 1

Supervised learning

Given a dataset of n *training examples* $\{(x^{(i)}, y^{(i)}); i = 1\}, \ldots, n\}$—a *training set*—where $\boldsymbol{x}$ represents the *features* and $\boldsymbol{y}$ the "output" or *target* variable we are trying to predict. If not already obvious, we denote the vector space of $\boldsymbol{x}$ as $\mathcal{X}$ and that of the outputs $\boldsymbol{y}$ as $\mathcal{Y}$.

Our goal is, given a training set, to learn a function $h : \mathcal{X} \mapsto \mathcal{Y}$ so that $h(x)$ is a "good" predictor for the corresponding y. This function h is called a *hypothesis*.

When trying to predict a continuous target variable, we call this a *regression* problem; whereas when y can take on only a small number of discrete values we call that a *classification* problem.

1.0.1 Linear Regression

Say we decide to approximate y as a linear function of x:

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

Where $\boldsymbol{\theta}$ represents the *parameters/weights* (parametrising the space of linear functions mapping from $\mathcal{X}$ to $\mathcal{Y}$). We can simplify our notation as such: (by convention letting $x_0 = 1$, aptly named the *intercept* term)

$$h(x) = \sum_{i=0}^{d} \theta_i x_i = \boldsymbol{\theta}^T \boldsymbol{x}$$

In order to formalise a measure of proximity between the predicted value $h(x)$ and the target y, we define a *cost function*:

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

This particular cost function implies an *ordinary least squares* regression model.

1.0.2 LMS algorithm

Our cost function $J(\theta)$ gives us a measure of prediction accuracy. We want to choose θ so as to minimise $J(\theta)$. Starting with an initial set of θ, we need a search algorithm that repeatedly changes θ in an attempt to minimise $J(\theta)$. Here we consider the *gradient descent* algorithm, which, given some initial θ, repeatedly performs the update:

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

Where the update is simultaneously performed for all values of $j = 0, \ldots, d$. α is called the *learning rate* (how much we move in the direction the gradient points in).

Intuition
Consider attempting to minimise the least mean squares (LMS) cost function for a single training example:

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

This gives us the update rule:

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

(we use the notation $a := b$ to denote (in a script) overwriting a with b) Notice the property of the LMS update rule that the magnitude of the update is proportional to the *error* term $\left(y^{(i)} - h_\theta(x^{(i)})\right)$ (this means that predictions further off the mark result in a greater correction to θ).
(next page)

Batch Gradient Descent