AI1001 - Assignment 10

Shivram S
ai24btech11031@iith.ac.in

1 Univariate Linear Regression

A univariate linear function with input x and output y has the form $y = w_1x + w_0$ where w_0 and w_1 are real-valued coefficients (**weights**) to be learned. We define **w** to be the vector $\langle w_0, w_1 \rangle$ and define the linear function as $h_{\mathbf{w}} = w_1x + w_0$. The task of finding the $h_{\mathbf{w}}$ that best fits the data is called **linear regression**.

To fit a line to the data, we find the values of the weights that minimizes the empirical loss. We can calculate the empirical loss, using the L_2 loss function, as

$$Loss(h_{\mathbf{w}}) = \sum_{j=i}^{N} L_2(y_1, h_{\mathbf{w}})(x_j) = \sum_{j=1}^{N} (y_j - h_{\mathbf{w}}(x_j))^2$$

We obtain $\mathbf{w}^* = \text{argmin}_{\mathbf{w}}\, Loss(h_{\mathbf{w}})$ when the empirical loss is minimum and its partial derivatives with respect to w_0 and w_1 are zero.

$$\frac{\partial}{\partial w_0} \sum_{j=1}^{N} (y_j - h_{\mathbf{w}}(x_j))^2 = 0 \text{ and } \frac{\partial}{\partial w_1} \sum_{j=1}^{N} (y_j - h_{\mathbf{w}}(x_j))^2 = 0$$

These equations have a unique solution

$$w_1 = \frac{N(\sum x_j y_j) - (\sum x_j)(\sum y_j)}{N(\sum x_j^2) - (\sum x_j)^2} \text{ and } w_0 = \frac{(\sum y_j - w_1 (\sum x_j))}{N}$$

The space defined by all possible settings of the weights is called the **weight space**. For univariate linear regression the weight space is two-dimensional. The loss function is **convex** for every linear regression problem with the L_2 loss function. There are no local minima.

2 Gradient Descent

Gradient descent is a method that lets us minimize loss without solving for the zeroes of the derivatives. It involves computing an estimate of the gradient at a point, and moving downhill along the steepest path until we reach a local minimum.

Gradient descent was discovered by Cauchy, who noticed that for any $u = f(x, y, z, \dots)$, we could say that:

$$f(x + \alpha, y + \beta, z + \gamma, \dots) \approx u + \alpha X + \beta Y + \gamma Z + \dots$$

Where $X = f'_x$, $Y = f'_y$, $\dots$ are the partial derivatives of the function. We can take $\alpha = -\theta X$, $\beta = -\theta Y$, $\dots$ to obtain

$$f(x - \alpha X, y - \beta Y, z - \beta Z, \dots) \approx u - \theta(X^2 + Y^2 + Z^2 + \dots)$$

Preprint. Under review.

If θ is small enough, we get a value $\Theta = f(x - \theta X, y - \theta Y, z - \theta Z, \dots)$ which is less than u. We can repeat this until Θ vanishes or coincides with a minimum value.

In the case of linear regression, we can iteratively reach convergence by performing the operation

$$w_1 \leftarrow w_1 - \alpha \frac{\partial}{\partial w_1} Loss(\mathbf{w})$$

α is a parameter called the **learning rate**. It can be a fixed constant or can decay over time. We can substitute the values of the partial derivatives to get

$$w_0 \leftarrow w_o + \alpha \sum_j (y - h_{\mathbf{w}}(x_j)) \text{ and } w_1 \leftarrow w_1 + \alpha \sum_j (y - h_{\mathbf{w}}(x_j)) \times x_j$$

These updates are called the **batch gradient descent** learning rule for univariate linear regression (also called **deterministic gradient descent**). A step that covers all the training examples is called an **epoch**.

A faster variant, called **stochastic gradient descent** (SGD), randomly selects a small number of training examples at each step. For example, by taking a **minibatch** size of $N/100$, each step becomes 100 times faster. Since the error is proportional to $\sqrt{N}$, we need 10 times as many steps. Overall, this method is still 10 times faster than deterministic gradient descent.

Convergence of SGD is not necessarily guaranteed. It can oscillate around the minimum without converging. However, gradual decrement of the learning rate, α, can guarantee convergence. SGD is also useful in an online setting, where new data comes in one at a time, and a model needs to adapt to changes represented in the new data.

3 Multivariate Linear Regression

We can extend our approach to **multivariate linear regression** where each example $\mathbf{x}_j$ is an n-element vector, and out hypotheses are functions of the form

$$h_{\mathbf{w}}(\mathbf{x}_j) = w_0 + w_1 x_{j,1} + \cdots + w_n x_{j,n} = w_0 + \sum_i w_i x_{j,i}$$

If we invent a dummy attribute $x_{j,0}$, which is always equal to 1, we can write our hypotheses as dot products

$$h_{\mathbf{w}}(\mathbf{x}_j) = \mathbf{w} \cdot \mathbf{x}_j = \mathbf{w}^\top \mathbf{x}_j$$

The best vector of weights, $\mathbf{w}^*$ minimizes squared-error:

$$\mathbf{w}^* = \operatorname*{argmin}_{\mathbf{w}} \sum_j L_2(y_j, \mathbf{w} \cdot \mathbf{x}_j)$$

It is possible to solve analytically for the $\mathbf{w}$ that minimizes loss. If $\mathbf{y}$ is the vector of outputs and $\mathbf{X}$ be the **data matrix**, then the vector of predicted outputs is $\hat{\mathbf{y}} = \mathbf{X}\mathbf{w}$, and the squared-error loss over the training data is

$$L(\mathbf{w}) = \|\hat{\mathbf{y}} - \mathbf{y}\|^2 = \|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2$$

We can set the gradient to zero

$$\nabla_{\mathbf{w}} L(\mathbf{w}) = 2\mathbf{X}^\top(\mathbf{X}\mathbf{w} - \mathbf{y}) = 0$$

Rearranging, we find that the minimum-loss weight vector is

$$\mathbf{w}^* = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}$$

$(\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top$ is called the **pseudoinverse** of the matrix, and the above equation is called the **normal equation**.