Linear Regression (Cont'd) and Classification

Satyanarayana Gajjarapu
AI24BTECH11009
Department of Artificial Intelligence
ai24btech11009@iith.ac.in

1 Regularization in Linear Regression

In univariate linear regression there is no chance of overfitting, but in multivariable linear regression there is a possibility of overfitting. Thus **regularization** is used on multivariable linear regression to avoid overfitting. With regularization, the total cost of hypothesis is minimized.

$$Cost(h) = EmpLoss(h) + \lambda Complexity(h)$$

For linear functions the complexity can be specified as a function of weights.

$$Complexity(h_{\mathbf{w}}) = L_q(\mathbf{w}) = \sum_i |w_i|^q$$

L_1 regularization has an important advantage because it tends to produce a **sparse model** which often sets many weights to zero. L_1 regularization takes the dimensional axes seriously, while L_2 treats them as arbitrary. The L_2 function is spherical, which makes it rotationally invariant.

2 Linear Classification

Linear functions can be used for classification as well as regression. A **decision boundary** is a line or a surface that separates two classes. A linear decision boundary is called a **linear separator** and data which consists it is called **linearly separable**. **Threshold function** outputs either 0 or 1 by passing the result of the linear function **w**·**x**.

$$h_{\mathbf{w}}(\mathbf{x}) = Threshold(\mathbf{w} \cdot \mathbf{x}) \text{ where } Threshold(z) = 1 \text{ if } z \geq 0 \text{ and 0 otherwise}$$

Gradient descent cannot be used to minimize the loss by finding weights because the gradient is zero almost everywhere in weight space except at those points where **w**·**x** = 0, because there the gradient is undefined. A simple weight update rule called the **perceptron learning rule** is used, which is similar to the update rule for linear regression. This rule converges to a solution only when the provided data is linearly separable.

2.1 Logistic regression

The hypothesis $h_{\mathbf{w}}(\mathbf{x})$ is not differentiable and discontinuous, which makes learning with the perceptron rule very unpredictable. These issues can be resolved by approximating the hard threshold function with a continuous, differentiable function like logistic function.

$$Logistic(z) = \frac{1}{1 + e^{-z}}$$

$$h_{\mathbf{w}}(\mathbf{x}) = Logistic(\mathbf{w} \cdot \mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w} \cdot \mathbf{x}}}$$

The process of fitting the weights of this model to minimize loss on a data set is called **logistic regression**. The hypotheses no longer output 0 or 1, because of using logistic function. So the L_2 loss function can be used for optimization with gradient descent. Let the logistic function is denoted by g. The derivation of the gradient is the same as for linear regression up to a certain point.

$$\begin{aligned}
\frac{\partial}{\partial w_i} Loss(\mathbf{w}) &= \frac{\partial}{\partial w_i}(y - h_{\mathbf{w}}(\mathbf{x}))^2 \\
&= 2(y - h_{\mathbf{w}}(\mathbf{x})) \times \frac{\partial}{\partial w_i}(y - h_{\mathbf{w}}(\mathbf{x})) \\
&= -2(y - h_{\mathbf{w}}(\mathbf{x})) \times g'(\mathbf{w} \cdot \mathbf{x}) \times \frac{\partial}{\partial w_i} \mathbf{w} \cdot \mathbf{x} \\
&= -2(y - h_{\mathbf{w}}(\mathbf{x})) \times g'(\mathbf{w} \cdot \mathbf{x}) \times x_i
\end{aligned}$$

The derivative of logistic function g is

$$g'(\mathbf{w} \cdot \mathbf{x}) = g(\mathbf{w} \cdot \mathbf{x})(1 - g(\mathbf{w} \cdot \mathbf{x})) = h_{\mathbf{w}}(\mathbf{x})(1 - h_{\mathbf{w}}(\mathbf{x}))$$

The update rule for minimizing the loss is

$$w_i \leftarrow w_i + \alpha(y - h_{\mathbf{w}}(\mathbf{x})) \times h_{\mathbf{w}}(\mathbf{x})(1 - h_{\mathbf{w}}(\mathbf{x})) \times x_i$$

where α is the learning rate.