Loss Functions and Numerical Optimization

Ravi Kothari, Ph.D.
ravi.kothari@ashoka.edu.in

"If you cannot measure it, you cannot improve it - Lord Kelvin"

Learning – A Formal Definition

Based on N (possibly noisy) observations $\mathcal{X} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{N}$ *of the input and output of a fixed though unknown system* $f(x)$*, construct an estimator* $\hat{f}(x;\theta)$ *so as to minimize,*

$$E\left[\left(L(f(x) - \hat{f}(x;\theta))\right)\right]$$

- L is the *loss function* and specifies *how* we want to measure the difference between the "desired output" ($f(x)$) and the "actual output" ($\hat{f}(x;\theta)$)

Learning – A Formal Definition

Based on N (possibly noisy) observations $\mathcal{X} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{N}$ of the input and output of a fixed though unknown system $f(x)$, construct an estimator $\hat{f}(x;\theta)$ so as to minimize,

$$E\left[\Big(L(f(x) - \hat{f}(x;\theta))\Big)\right]$$

- L is the *loss function* and specifies *how* we want to measure the difference between the “desired output” ($f(x)$) and the “actual output” ($\hat{f}(x;\theta)$)
- We have seen a loss function already; recall our discussion when we did the Bayes classifier

Some Loss Functions

Some Loss Functions

- *Classification*: $f(x) \in \{\omega_1, \omega_2, \ldots, \omega_c\}$

$$L(f(x), \hat{f}(x;\theta)) = \begin{cases} 0, & f(x) = \hat{f}(x;\theta) \\ 1, & f(x) \neq \hat{f}(x;\theta) \end{cases}$$

Some Loss Functions

- *Classification*: $f(x) \in \{\omega_1, \omega_2, \ldots, \omega_c\}$

$$L(f(x), \hat{f}(x;\theta)) = \begin{cases} 0, & f(x) = \hat{f}(x;\theta) \\ 1, & f(x) \neq \hat{f}(x;\theta) \end{cases}$$

- *Regression*: $f(x) \in \mathcal{R}^m$

$$L(f(x), \hat{f}(x;\theta)) = \left(f(x) - \hat{f}(x;\theta)\right)^2$$

Some Loss Functions

- *Classification*: $f(x) \in \{\omega_1, \omega_2, \ldots, \omega_c\}$

$$L(f(x), \hat{f}(x;\theta)) = \begin{cases} 0, & f(x) = \hat{f}(x;\theta) \\ 1, & f(x) \neq \hat{f}(x;\theta) \end{cases}$$

- *Regression*: $f(x) \in \mathcal{R}^m$

$$L(f(x), \hat{f}(x;\theta)) = \left(f(x) - \hat{f}(x;\theta)\right)^2$$

- *Density Estimation*:

$$L(\hat{f}(x;\theta)) = -\log \hat{f}(x;\theta)$$

Some Loss Functions

- *Classification*: $f(x) \in \{\omega_1, \omega_2, \ldots, \omega_c\}$

$$L(f(x), \hat{f}(x;\theta)) = \begin{cases} 0, & f(x) = \hat{f}(x;\theta) \\ 1, & f(x) \neq \hat{f}(x;\theta) \end{cases}$$

- *Regression*: $f(x) \in \mathcal{R}^m$

$$L(f(x), \hat{f}(x;\theta)) = \left(f(x) - \hat{f}(x;\theta)\right)^2$$

- *Density Estimation*:

$$L(\hat{f}(x;\theta)) = -\log \hat{f}(x;\theta)$$

The loss function may not be *symmetric* (in fact, it is not in many cases e.g. in medical diagnosis, a false negative is MUCH worse than a false positive)

Learning

Learning

- Once a loss function is defined, learning becomes an optimization process that minimizes the loss function by adjusting the parameters (θ)

Learning

- Once a loss function is defined, learning becomes an optimization process that minimizes the loss function by adjusting the parameters (θ)
- This is the *prevalent and popular* definition of learning (though completely inappropriate in my view – more on it later)

Numerical Optimization

Numerical Optimization

- The general idea is to start with some values of the parameters, present a training pattern, adjust the parameters to reduce the loss function, present the next training pattern and repeating until the loss function is below an acceptable threshold (or stops decreasing)

Numerical Optimization

- The general idea is to start with some values of the parameters, present a training pattern, adjust the parameters to reduce the loss function, present the next training pattern and repeating until the loss function is below an acceptable threshold (or stops decreasing)
- i.e.

$$\theta' = \theta + \Delta\theta \tag{1}$$

such that,

$$L(t+1) < L(t) \tag{2}$$

Numerical Optimization

- The general idea is to start with some values of the parameters, present a training pattern, adjust the parameters to reduce the loss function, present the next training pattern and repeating until the loss function is below an acceptable threshold (or stops decreasing)
- i.e.
$$\theta' = \theta + \Delta\theta \tag{1}$$
such that,
$$L(t+1) < L(t) \tag{2}$$
- A popular approach to finding θ is based on *gradient descent*

Gradient Descent

Gradient Descent

- Using Taylor Series ($\theta' = \theta + \Delta\theta$),

$$L(\theta') = L(\theta) + \nabla L(\theta)_\theta^T \Delta\theta + \dots \tag{3}$$

Gradient Descent

- Using Taylor Series ($\theta' = \theta + \Delta\theta$),

$$L(\theta') = L(\theta) + \nabla L(\theta)_\theta^T \Delta\theta + \dots \tag{3}$$

- To lower the LHS,

$$\nabla L(\theta)_\theta^T \Delta\theta < 0 \tag{4}$$

Gradient Descent

- Using Taylor Series ($\theta' = \theta + \Delta\theta$),

$$L(\theta') = L(\theta) + \nabla L(\theta)_\theta^T \Delta\theta + \dots \tag{3}$$

- To lower the LHS,

$$\nabla L(\theta)_\theta^T \Delta\theta < 0 \tag{4}$$

- This happens maximally when,

$$\Delta\theta \propto -\nabla L(\theta)_\theta \tag{5}$$

Gradient Descent

- Using Taylor Series ($\theta' = \theta + \Delta\theta$),
$$L(\theta') = L(\theta) + \nabla L(\theta)_\theta^T \Delta\theta + \dots \tag{3}$$
- To lower the LHS,
$$\nabla L(\theta)_\theta^T \Delta\theta < 0 \tag{4}$$
- This happens maximally when,
$$\Delta\theta \propto -\nabla L(\theta)_\theta \tag{5}$$
- i.e.,
$$\Delta\theta = -\eta \nabla L(\theta)_\theta \tag{6}$$
where, η is the step size. So,
$$\theta' = \theta - \eta L(\theta)_\theta \tag{7}$$

Example

Example

- Say we want to find the minimum of,

$$L(\theta) = \theta_1^2 + (\theta_2 - 3)^2 \tag{8}$$

Example

- Say we want to find the minimum of,

$$L(\theta) = \theta_1^2 + (\theta_2 - 3)^2 \tag{8}$$

- Let the starting guess be,

$$\theta = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 0.7 \end{bmatrix} \tag{9}$$

Example

- Say we want to find the minimum of,

$$L(\theta) = \theta_1^2 + (\theta_2 - 3)^2 \tag{8}$$

- Let the starting guess be,

$$\theta = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 0.7 \end{bmatrix} \tag{9}$$

- So,

$$\nabla L(\theta) = \begin{bmatrix} \frac{\partial L(\theta)}{\partial(\theta_1)} \\ \frac{\partial L(\theta)}{\partial(\theta_2)} \end{bmatrix} = \begin{bmatrix} 2\theta_1 \\ 2\theta_2 - 6 \end{bmatrix} \tag{10}$$

Example

- Say we want to find the minimum of,

$$L(\theta) = \theta_1^2 + (\theta_2 - 3)^2 \tag{8}$$

- Let the starting guess be,

$$\theta = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 0.7 \end{bmatrix} \tag{9}$$

- So,

$$\nabla L(\theta) = \begin{bmatrix} \frac{\partial L(\theta)}{\partial(\theta_1)} \\ \frac{\partial L(\theta)}{\partial(\theta_2)} \end{bmatrix} = \begin{bmatrix} 2\theta_1 \\ 2\theta_2 - 6 \end{bmatrix} \tag{10}$$

- Using $\eta = 0.1$ in,

$$\theta(t+1) = \theta(t) - \eta \nabla_{\theta(t)} \tag{11}$$

we get,

$$\theta = \begin{bmatrix} 1 \\ 0.7 \end{bmatrix} = -0.1 \begin{bmatrix} 2\theta_1 \\ 2\theta_2 - 6 \end{bmatrix} = \begin{bmatrix} 1 \\ 0.7 \end{bmatrix} - 0.1 \begin{bmatrix} 2 \\ -4.4 \end{bmatrix} = \begin{bmatrix} 0.8 \\ 1.14 \end{bmatrix} \tag{12}$$

Example

- Say we want to find the minimum of,

$$L(\theta) = \theta_1^2 + (\theta_2 - 3)^2 \tag{8}$$

- Let the starting guess be,

$$\theta = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 0.7 \end{bmatrix} \tag{9}$$

- So,

$$\nabla L(\theta) = \begin{bmatrix} \frac{\partial L(\theta)}{\partial(\theta_1)} \\ \frac{\partial L(\theta)}{\partial(\theta_2)} \end{bmatrix} = \begin{bmatrix} 2\theta_1 \\ 2\theta_2 - 6 \end{bmatrix} \tag{10}$$

- Using $\eta = 0.1$ in,

$$\theta(t+1) = \theta(t) - \eta \nabla_{\theta(t)} \tag{11}$$

we get,

$$\theta = \begin{bmatrix} 1 \\ 0.7 \end{bmatrix} = -0.1 \begin{bmatrix} 2\theta_1 \\ 2\theta_2 - 6 \end{bmatrix} = \begin{bmatrix} 1 \\ 0.7 \end{bmatrix} - 0.1 \begin{bmatrix} 2 \\ -4.4 \end{bmatrix} = \begin{bmatrix} 0.8 \\ 1.14 \end{bmatrix} \tag{12}$$

Other Optimization Methods

Other Optimization Methods

- Gradient descent can get trapped in a local minima (like all local search methods)

Other Optimization Methods

- Gradient descent can get trapped in a local minima (like all local search methods)
- It can be slow to converge. One can use higher order derivatives. They are expensive to compute though provide a better approximation of the error surface and can converge faster

Other Optimization Methods

- Gradient descent can get trapped in a local minima (like all local search methods)
- It can be slow to converge. One can use higher order derivatives. They are expensive to compute though provide a better approximation of the error surface and can converge faster
- Quasi-Newton's method (such as *Broyden–Fletcher–Goldfarb–Shanno* (BFGS)) do not explicitly compute the *Hessian* but rather approximate it. They usually converge much faster than first-order gradient descent. We will look at some if time permits