Stein's unbiased risk estimator

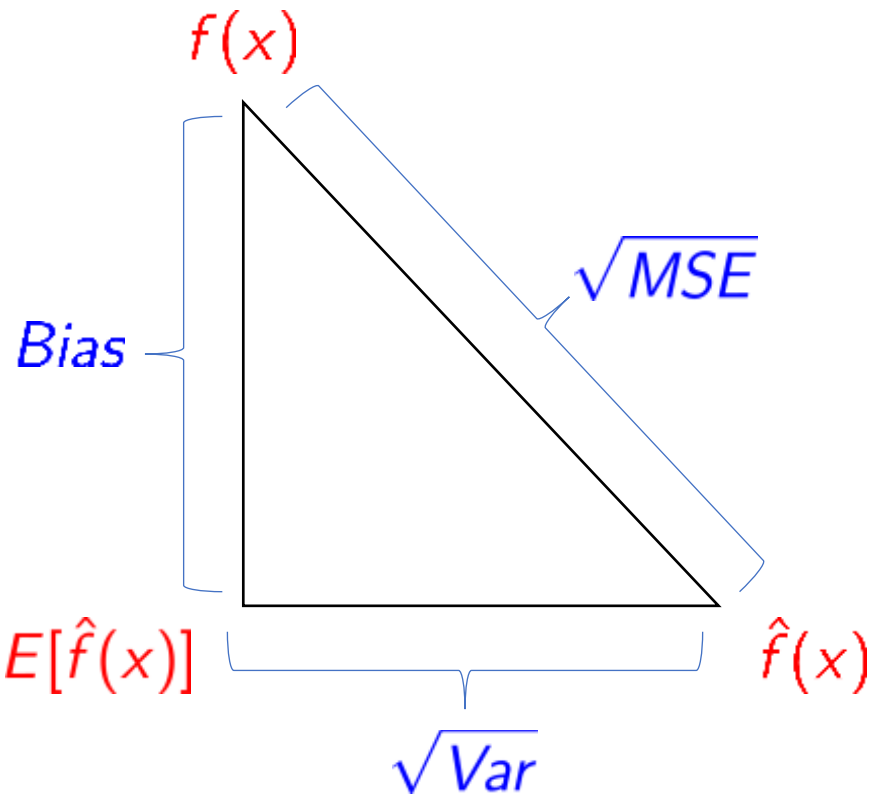
Model Selection

- The general task in machine learning is estimating a function.
  - We want to estimate :

$\hat{f}(x)$ estimated function

  - Where there is a true underlying function :

$f(x)$ true function



Definitions and Notations

assume $T = \{(x_i, y_i)\}_{i=1}^{n}$ be the training set.

$f(.) \rightarrow$ True function

$\hat{f}(.) \rightarrow$ Estimated function

also assume:

$y_i = f(x_i) + \epsilon_i$

where $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$

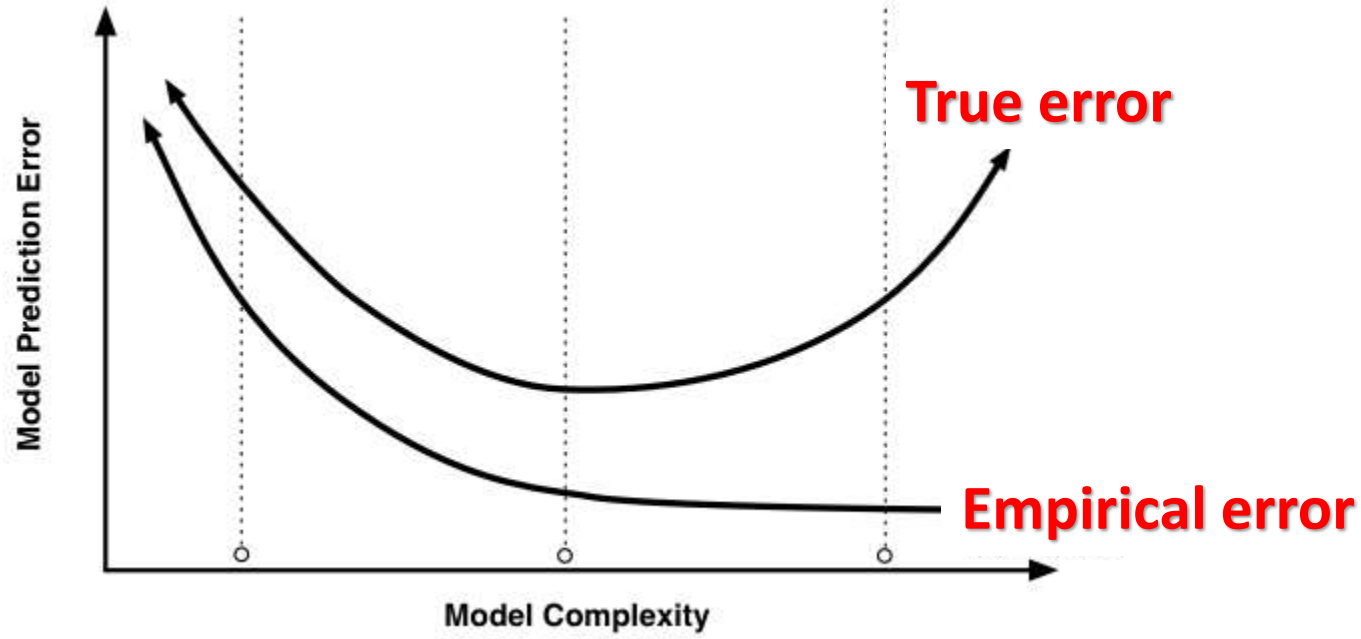
$\hat{y}_i = \hat{f}(x_i)$

$f_i \equiv f(x_i)$

$\hat{f}_i \equiv \hat{f}(x_i)$

- Empirical error:

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$



For point (x_0, y_0) we are interested in:

$$
\begin{aligned}
E[(\hat{y}_0 - y_0)^2] &= E[(\hat{f}_0 - f_0 - \epsilon_0)^2] \\
&= E\left[\left((\hat{f}_0 - f_0) - \epsilon_0\right)^2\right] \\
&= E[(\hat{f}_0 - f_0)^2 + \epsilon_0^2 - 2\epsilon_0(\hat{f}_0 - f_0)] \\
&= E[(\hat{f}_0 - f_0)^2] + E[\epsilon_0^2] - 2E[\epsilon_0(\hat{f}_0 - f_0)] \\
&= E[(\hat{f}_0 - f_0)^2] + \sigma^2 - 2E[\epsilon_0(\hat{f}_0 - f_0)]
\end{aligned}
$$

case 1

assume: $(x_0, y_0) \notin T$

In this case, since $\hat{f}$ is estimated only based on points in training set, therefore it is completely independent from (x_0, y_0)

$$\Rightarrow E[(y_0 - f)(\hat{f} - f)] = cov(y_0, \hat{f}_0) = 0$$

If summing up all m pints that are not in T.

$$\underbrace{\sum_{i=1}^{m}(\hat{y}_i - y_i)^2}_{\text{err}} = \underbrace{\sum_{i=1}^{m}(\hat{f}_i - f_i)^2}_{\text{Err}} + m\sigma^2$$

true error

empirical error

$$Err = err - m\sigma^2$$

Empirical error (err) is a good estimator of true error (Err) if the point (x_0, y_0) is not in the training set.

case 2

assume: $(x_0, y_0) \in T$

then: $2E[\epsilon_0(\hat{f}_0 - f_0)] \neq 0$

Stein's Lemma

If :

$x \sim \mathcal{N}(\theta, \sigma^2)$
and $g(x)$ is differentiable.

then

$$E[g(x)(x-\theta)] = \sigma^2 E[\frac{\partial g(x)}{\partial x}]$$

- Our problem:

$$E[\epsilon_0(\hat{f}_0 - f_0)] = \sigma^2 E[\frac{\partial(\hat{f}_0 - f_0)}{\partial \epsilon_0}]$$

$$= \sigma^2 E[\frac{\partial \hat{f}_0}{\partial \epsilon_0} - \frac{\partial f_0}{\partial \epsilon_0}]$$ (0)

$$= \sigma^2 E[\frac{\partial \hat{f}_0}{\partial \epsilon_0}]$$ (Chain rule)

$$= \sigma^2 E[\frac{\partial \hat{f}_0}{\partial y_0} \cdot \frac{\partial y_0}{\partial \epsilon_0}]$$ (1)

$$= \sigma^2 E[\frac{\partial \hat{f}_0}{\partial y_0}]$$

D_0

$$E[((\hat{y})_0 - y_0)^2] = E[((\hat{f})_0 - f_0)^2] + \sigma^2 - 2\sigma^2 E[D_0]$$

Sum over all n data points:

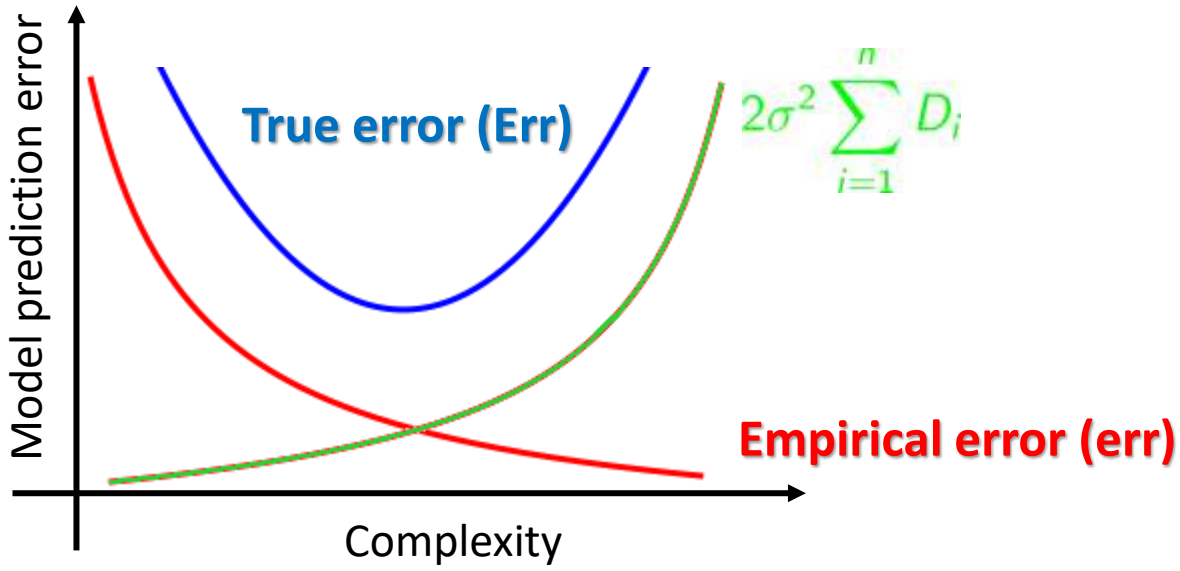
$$\underbrace{\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}_{\text{err}} = \underbrace{\sum_{i=1}^{n}(\hat{f}_i - f_i)^2}_{\text{Err}} + n\sigma^2 - 2\sigma^2 \sum_{i=1}^{n} D_i$$

$$Err = err - n\sigma^2 + 2\sigma^2 \underbrace{\sum_{i=1}^{n} D_i}_{\text{Complexity of model}}$$

Stein's Unbiased Risk Estimator (SURE)

- SURE gives us a very good insight about the behavior of true error

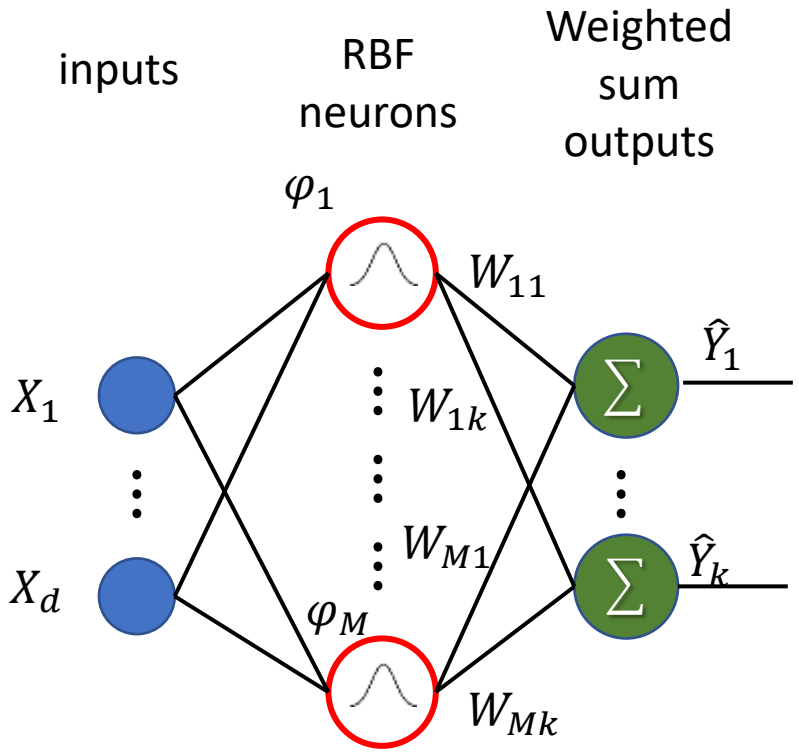
$$Err = err - n\sigma^2 + 2\sigma^2 \sum_{i=1}^{n} D_i$$



optimum complexity of RBF network

- How can we compute the optimum complexity of RBF network?

- Complexity of RBFNN controls by RBF units.
  - More RBF nodes, more complexity



Review of RBF Network

Inputs:

$$X = \begin{bmatrix} x_{11} & \cdots & x_{1n} \\ \vdots & \ddots & \vdots \\ x_{d1} & \cdots & x_{dn} \end{bmatrix}_{d \times n} = \begin{bmatrix} \vdots & \vdots & & \vdots \\ \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_n \\ \vdots & \vdots & & \vdots \end{bmatrix}_{d \times n}$$

Weights:

$$W = \begin{bmatrix} w_{11} & \cdots & w_{1k} \\ \vdots & \ddots & \vdots \\ w_{m1} & \cdots & w_{mk} \end{bmatrix}_{m \times k}$$

Radial Basis Functions:

$$\Phi = \begin{bmatrix} \phi_1(\mathbf{x}_1) & \cdots & \phi_1(\mathbf{x}_n) \\ \vdots & \ddots & \vdots \\ \phi_m(\mathbf{x}_1) & \cdots & \phi_m(\mathbf{x}_n) \end{bmatrix}_{m \times n}$$

Outputs:

$$Y = \begin{bmatrix} y_{11} & \cdots & y_{1n} \\ \vdots & \ddots & \vdots \\ y_{k1} & \cdots & y_{kn} \end{bmatrix}_{k \times n}$$

Review of RBF Network

- Output of network:

$$Y = W^T \Phi$$

- Objective function used for computing W

$$\min_W \left\| Y - W^T \Phi \right\|^2$$

- Close form solution for W

$$W = (\Phi \Phi^T)^{-1} \Phi Y^T$$

- In RBF network the estimated function is:

$$\hat{Y} = W^T \Phi$$

$$= \left((\Phi \Phi^T)^{-1} \Phi Y^T \right)^T \Phi$$

$$= Y \Phi^T \left((\Phi \Phi^T)^{-1} \right)^T \Phi$$

$$\Rightarrow \boxed{\hat{Y}^T = \Phi^T (\Phi \Phi^T)^{-1} \Phi Y^T}$$

Estimated function ($\hat{f}$)

H

$$\hat{Y}^T = HY^T$$

- Let's assume that the number of output nodes k is equal to 1
- Therefore the output of network will be a vector

- Let's to apply SURE to RBF.

$$\left.\begin{array}{l} D_i = \dfrac{\partial \hat{f}_i}{\partial y_i} \\ \hat{f}_i = \hat{y}_i = H_{i:}y \end{array}\right\} \quad D_i = \frac{\partial \hat{f}_i}{\partial y_i} = \frac{\partial H_{i:}y}{\partial y_i} = H_{ii}$$

- Then SURE will be:

$$Err = err - n\sigma^2 + 2\sigma^2 \sum_{i=1}^{n} H_{ii}$$

$$= err - n\sigma^2 + 2\sigma^2 Tr(H)$$

$$Tr(H) = Tr\left[\Phi^T(\Phi\Phi^T)^{-1}\Phi\right]$$

$$= Tr\left[\Phi\Phi^T(\Phi\Phi^T)^{-1}\right]$$

$$= Tr\left[I_m\right] = m$$

- Where m in RBF network is the number of RBF nodes.
- Therefore SURE for RBF network is:

$$Err = err - n\sigma^2 + 2\sigma^2 m$$

- For computing SURE, we need to know the value of σ. But we do not know it. Therefore we need to estimate it.

$$\sigma^2 = \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n-1}$$

- Since the $\hat{y}$ is the function of complexity (more complex, smaller σ^2), in practice, we do not consider the $\hat{y}$ to be the function of complexity and instead we consider it to be a low bias and high variance estimation (for example a line).

- With this assumption the σ will be considered to be constant and independent from the complexity of model.

- If we want to minimize true error in SURE, we have:

$$Err = err - n\sigma^2 + 2\sigma^2 \sum_{i=1}^{n} D_i$$

$$\Rightarrow \min Err = \boxed{\min err + 2\sigma^2 \sum_{i=1}^{n} D_i}$$

Function of complexity

This is a justification for regularization!

regularization

- In regularization, we add a penalty function to the objective function (which is a function of complexity) in order to control the complexity of model.

$$err + \underbrace{L(\theta)}_{\text{penalty function}}$$