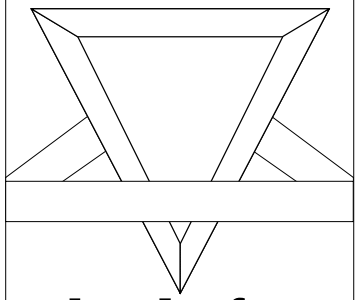
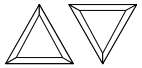
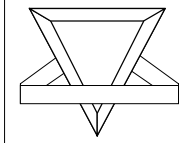


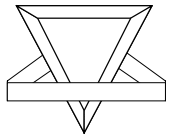
Chap. 7 Methods for Regression



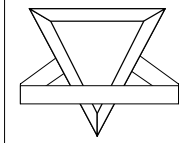


개요

- Regression
  - Sample에서 continuous-valued function을 estimate하는 것
- Regression 방법을 분류하는 기준 3가지
  - Approximating function 집합의 parameterization에 따라
  - Parameter estimation을 위한 optimization procedure에 따라
  - Interpretation capability에 따라



차례



7.1 Taxonomy: Dictionary vs. Kernel Representation

- 일반적인 approximating function의 parameterization

$$f_m(\mathbf{x}, \mathbf{w}, \mathbf{v}) = \sum_{i=1}^{m} w_i g_i(\mathbf{x}, \mathbf{v}_i) + w_0$$

  - $g_i(\mathbf{x}, \mathbf{v}_i)$: basis function
  - $\mathbf{v}_i = [v_{1i}, v_{2i}, \ldots, v_{pi}]$: basis function parameters
  - $\mathbf{w} = [w_0, \ldots, w_m]$: linear combination의 coefficients
- Regression의 목적
  - 가장 작은 mean squared-error를 가지는 $\mathbf{v}_i$, $\mathbf{w}$의 estimation

Dictionary method

- 주어진 basis function들의 집합(dictionary)에 의해 method가 정해진다.

Basis function의 특성에 의한 분류

1. Fixed basis function

$$f_m(\mathbf{x}, \mathbf{w}) = \sum_{i=1}^{m} w_i g_i(\mathbf{x}) + w_0$$

- Nonadaptive method

2. Adaptive basis function
   - Basis function이 data에 따라서 바뀔 수 있다.
   - 보통 같은 형태의 basis function을 사용한다.

$$f_m(\mathbf{x}, \mathbf{w}, \mathbf{v}) = \sum_{i=1}^{m} w_i g(\mathbf{x}, \mathbf{v}_i) + w_0$$

예제) multilayer perceptron (MLP)

$$g(\mathbf{x}, \mathbf{v}_i) = s(v_{i0} + \sum_{k=1}^{d} x_k v_{ik}) = s(\mathbf{x} \cdot \mathbf{v}_i)$$

- Sigmoid (logistic)

$$s(t) = \frac{1}{1 + \exp(-t)}$$

- Sigmoid (hyperbolic tangent)

$$s(t) = \tanh(t) = \frac{\exp(t) - \exp(-t)}{\exp(t) + \exp(-t)}$$

예제) Radial basis function (RBF) network

$$g(\mathbf{x}, \mathbf{v}_i) = g(\| \mathbf{x} - \mathbf{v}_i \|) = K\left(-\frac{\| \mathbf{x} - \mathbf{v}_i \|}{\alpha} \right)$$

- $g(t)$: univariate function

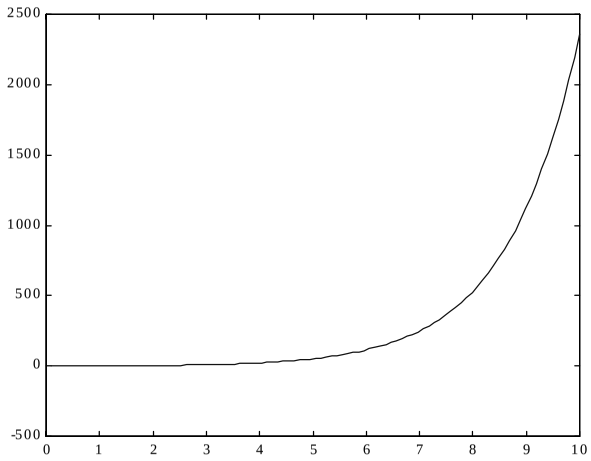
1. Local radial basis functions (kernel function K)

$$Gaussian: g(t) = \exp(-\frac{t^2}{2\alpha^2})$$

$$\text{Multiquadratic}: \ g(t) = (t^2 + b^2)^{-\alpha}$$

2. Nonlocal radial basis function

$$g(t) = t \quad \text{and} \quad g(t) = t^2 \ln(t)$$



$$g(t) = t^2 \ln(t)$$

MLP, RBF network의 graphical form

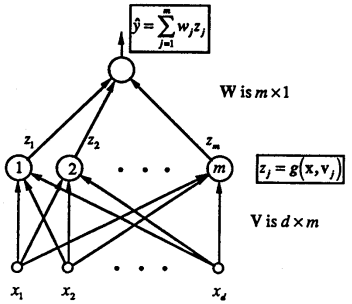


Fig. 7.1 Multilayer perceptron and radial basis function approximators, usually presented in graphical form as a network.

- Parameters: network weights
- Input (output) variables: input (output) nodes
- Basis function: hidden layer unit

Adaptive basis functions

- univariate
- **x**,**v** 에 대해서 symmetric

Basis function

- (nonlinear) feature
- Basis function의 optimal selection
  - Feature selection이라고 할 수 있다.

Kernel methods

$$f(x) = \sum_{i=1}^{n} K_i(\mathbf{x}, \mathbf{x}_i) y_i$$

- $K(x, x_i)$: 주로 다음의 특성을 만족시키는 symmetric function

$K(\mathbf{x}, \mathbf{x}') \geq 0$	Nonnegative
$K(\mathbf{x}, \mathbf{x}') = K(\|\| \mathbf{x} - \mathbf{x}' \|\|)$	Radially symmetric
$K(\mathbf{x}, \mathbf{x}) = \max$	$\mathbf{x} = \mathbf{x}'$일때 최대값
$\lim_{t \to \infty} K(t) = 0$	단조감소

Kernel function의 두 종류

1. Kernel function의 값이 training data의 x_i값에만 의존하는 경우
   - Nonadaptive
   - Nonadaptive dictionary method와 대응되며, 서로 변환이 가능하다.
2. Kernel function의 선택이 training data의 y 값에도 영향을 받는 경우
   - Adaptive kernel method

- Dictionary method에서 optimal least sqares solution이 발견되면 이는 바로 동등한 kernel method의 형태로 변환될 수 있다.
- 대부분의 adaptive method에서는 kernel method 보다는 dictionary method가 쓰인다.
  - 이는 dictionary method에서는 global function을 사용하며 모든 training data를 이용하기 때문이다.
- Nonadaptive kernel method에서는 고정된 kernel span α가 사용된다.

7.2 Linear Estimator

- Regression estimator가 다음의 superposition principle을 만족시키면 linear라 한다.

$$f_0(a\mathbf{y}'+b\mathbf{y}''|\,\mathbf{X}) = af_1(\mathbf{y}'|\,\mathbf{X}) + bf_2(\mathbf{y}''|\,\mathbf{X})$$

  - nonzero a, b
  - f_0, f_1, f_2: 하나의 approximating functions 집합에서의 3개의 estimate
  - $\mathbf{X}=(x_1, ..., x_n)$: predictor sample
  - y', y'' :두 개의 response 값들

- Linear approximating function의 두가지 표현 방법
  1. Fixed basis function의 linear combination
     - fixed basis functions의 선택: learning problem에 대한 priori knowledge
     - learning: empirical risk나 penalized risk의 최소화에 따른 linear coefficients의 선택
  2. Training data의 kernel average
     - kernel function의 형식: priori knowledge
- Kernel function의 두 가지 예
  - Kernel density estimation
  - Equivalent basis function representation of a linear estimator

- Kernel density estimation
  - Approximating function

$$\hat{p}(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n} K_\alpha(\mathbf{x}, \mathbf{x}_i)$$

    - Kernel function에 또 필요한 특성

$$\int_{-\infty}^{\infty} K(\mathbf{x}, \mathbf{x}')d\mathbf{x}' = 1 \quad \text{for any } \mathbf{x}$$

  - Kernel regression approximating function

$$f_a(\mathbf{x}, \mathbf{w}_n \,|\, \mathbf{x}_n) = \frac{\sum_{i=1}^{n} w_i K_\alpha(\mathbf{x}, \mathbf{x}_i)}{\sum_{i=1}^{n} K_\alpha(\mathbf{x}, \mathbf{x}_i)}$$

    - Local symmetric neighborhood near $\mathbf{x}$

- Least squares로 estimate된 linear model에 대한 두 가지 표현방법

$$\hat{y} = f(\mathbf{x}, \mathbf{w}^*) = \sum_{j=1}^{m} w_j^* g_j(\mathbf{x}) = \sum_{i=1}^{n} S(\mathbf{x}, \mathbf{x}_i) y_i$$

  - kernel function $S(\mathbf{x}, \mathbf{x}')$이 꼭 local일 필요는 없다.
- Basis function으로 표현된 linear model의 VC-dimension은 basis function의 개수와 같다.
  - 이를 이용해서 penalized linear model과 kernel estimator의 model complexity를 계산하고 효율적인 free parameter의 개수를 구할 수 있다.

7.2.1 Estimation of Linear Models and Equivalence of Representations

- Least squares solution for estimating **w**

$$\mathbf{Zw} \cong \mathbf{y}$$

$$\mathbf{Z} = \begin{bmatrix} g_1(\mathbf{x}_1) & \cdot & \cdot & \cdot & g_m(\mathbf{x}_1) \\ \cdot & & & & \cdot \\ \cdot & & & & \cdot \\ \cdot & & & & \cdot \\ g_1(\mathbf{x}_n) & \cdot & \cdot & \cdot & g_m(\mathbf{x}_n) \end{bmatrix} = [g_1(\mathbf{X}) | g_2(\mathbf{X}) | ... | g_m(\mathbf{X})]$$

- Empirical risk

$$R_{emp}(\mathbf{w}) = \frac{1}{n} \| \mathbf{Zw} - \mathbf{y} \|^2$$

- Solution

$$\mathbf{Z}^\mathbf{T}\mathbf{Zw} = \mathbf{Z}^\mathbf{T}\mathbf{y}$$

  - **Z**가 linearly independent하면 unique solution이 존재한다.(대부분의 경우가 그렇다. $m <= n$)

$$\mathbf{w}^* = (\mathbf{Z}^\mathbf{T}\mathbf{Z})^{-1}\mathbf{Z}^\mathbf{T}\mathbf{y}$$

- Parametric penalization이 linear estimator에 적용된 경우
  - Penalized risk functional

$$R_{pen}(\mathbf{w}) = \frac{1}{n}(\| \mathbf{Zw} - \mathbf{y} \|^2 + \mathbf{w}^\mathbf{T}\Phi\mathbf{w})$$

    - Φ: $m*m$ penalty matrix
    - Regularization parameter λ is absorbed in Φ
  - Solution

$$\mathbf{w}^* = (\mathbf{Z}^\mathbf{T}\mathbf{Z} + \Phi)^{-1}\mathbf{Z}^\mathbf{T}\mathbf{y}$$

- 이를 modified least squares로 풀 수 있다.

1. 다음의 행렬을 만든다.

$$\mathbf{U} = \begin{bmatrix} \mathbf{Z} \\ \mathbf{A} \end{bmatrix}, \quad \mathbf{v} = \begin{bmatrix} \mathbf{y} \\ \mathbf{0} \end{bmatrix}$$

- **Z**: 주어진 data matrix
- $\Phi = \mathbf{A}^T\mathbf{A}$
- **0**: *m*개의0을 가지는 column vector
- 결국은 관찰된 data에 인공적인 data를 첨가한 것과 같다.

2. 다음의 empirical risk functional을 minimize

$$R_{emp} = \frac{1}{n} \| \mathbf{Uw} - \mathbf{v} \|^2$$

- Solution

$$\mathbf{w}^* = (\mathbf{U}^T\mathbf{U})^{-1}\mathbf{U}^T\mathbf{v} = (\mathbf{Z}^T\mathbf{Z} + \mathbf{A}^T\mathbf{A})^{-1}\mathbf{Z}^T\mathbf{y}$$

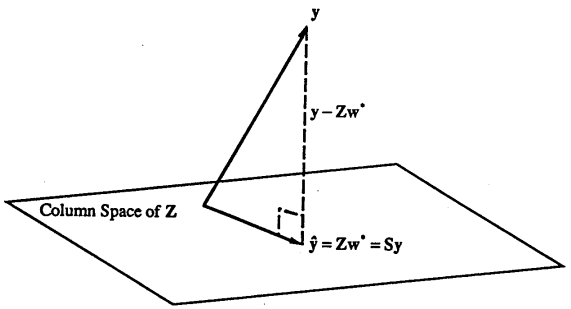
- Training data에 artificial example을 포함시키는 방법
  - training sample의 개수에 따라 얼마나 artificial example을 포함시킬 것인가에 대한 문제가 있다.

- 하나의 representation을 그와 동등한 다른 형태의 representation으로 analytical하게 변환하는 것이 가능하다.
  - 각각의 representation들의 computational efficiency, estimation of complexity, model interpretation 등이 서로 다르기 때문에 유용하다.
- (penalized) least squares solution에는 임의의 벡터 **y**를 column space **Z**에 project시키는 projection matrix **S**가 존재한다.

$$\hat{\mathbf{y}} = \mathbf{Zw}^* = \mathbf{Sy}$$

- Matrix **S**는 다음과 같다.

$$\mathbf{S} = \mathbf{Z}(\mathbf{Z}^T\mathbf{Z})^{-1}\mathbf{Z}^T, \quad \mathbf{S}_\Phi = \mathbf{Z}(\mathbf{Z}^T\mathbf{Z} + \Phi)^{-1}\mathbf{Z}^T$$

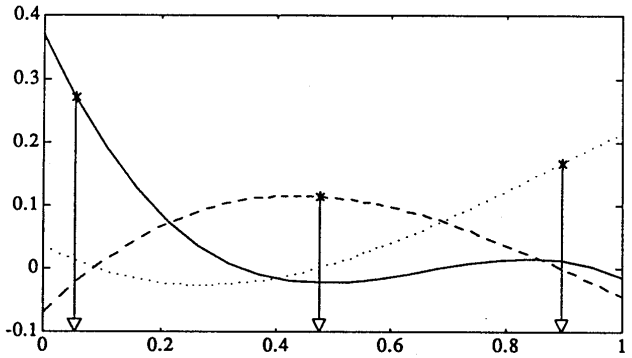


Matrix **S**는 optimal basis function estimate of $\mathbf{w}^*$와 동등한 kernel이라고 볼 수 있다.
임의의 kernel function은 다음과 같다.

$$S(\mathbf{x},\mathbf{x}_i) = g(\mathbf{x})(\mathbf{Z}^{\mathbf{T}}\mathbf{Z})^{-1}g^{\mathbf{T}}(\mathbf{x}_i)$$

$$S_{\Phi}(\mathbf{x},\mathbf{x}_i) = g(\mathbf{x})(\mathbf{Z}^{\mathbf{T}}\mathbf{Z}+\Phi)^{-1}g^{\mathbf{T}}(\mathbf{x}_i)$$

이렇게 만들어진 kernel function은 원래의 kernel function이 가지는 property들을 모두 만족시키지는 않는 것도 있다.



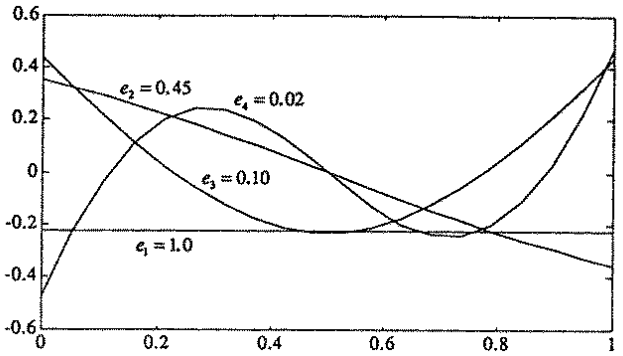
Kernel function이 symmetric하다면 이를 동등한 basis function expansion으로 변환하는 것이 가능하다.

- Kernel에 대한 eigen function decomposition

$$K(\mathbf{x},\mathbf{x}') = \sum_{i=1}^{\infty} e_i g_i(\mathbf{x}) g_i(\mathbf{x}')$$

- e_i: eigen values
- g_i(x): eigen function

Eigen value와 eigen function의 예



7.2.2 Analytic Form of Cross-Validation

- Matrix **S**에 의해 정의된 linear estimate를 가지고 leave-one-out cross-validation의 expected risk를 analytic하게 계산하는 것이 가능하다 (resampling에 비해서 계산상의 이점을 가진다).

$$\hat{y}_i' = \frac{1}{1-s_{ii}} \sum_{j=1, j\neq i}^{n} s_{ij} y_j \quad \text{or} \quad \hat{\mathbf{y}}' = \mathbf{S}'\mathbf{y}$$

- Leave-one-out cross-validation의 expected risk

$$\begin{aligned} y_i - \hat{y}_i' &= y_i - \frac{1}{1-s_{ii}} \sum_{j=1, j\neq i}^{n} s_{ij} y_j \\ &= \frac{(1-s_{ii}) y_i - \sum_{j=1, j\neq i}^{n} s_{ij} y_j}{1-s_{ii}} \\ &= \frac{y_i - \sum_{j=1}^{n} s_{ij} y_j}{1-s_{ii}} \\ &= \frac{y_i - \hat{y}_i}{1-s_{ii}} \end{aligned}$$

- 결과적인 risk functional

$$R(\mathbf{w}^*) \cong R_{cv}(\mathbf{w}^*) = \frac{1}{n} \sum_{i=1}^{n} \left(\frac{y_i - \hat{y}_i}{1-s_{ii}} \right)^2$$

7.2.3 Estimating Complexity of Penalized Linear Models

- Basis function을 사용하는 linear model의 경우: free parameter의 갯수가 model complexity이다.
- Free parameter의 갯수를 모를 때에는 kernel representation의 eigenvalue를 가지고 model complexity를 측정한다.
  - Eigen function은 orthogonal하며 positive symmetric kernel에 대한 eigen value는 nonnegative하다.
  - Degree of freedom은 eigen decomposition에서 significant term의 갯수이다.

- significance는 eigen value의 크기에 의해 결정된다.

- Projection matrix **S**의 rank가 model complexity이다.
- $\mathbf{S}_\lambda$가 penalized least square에 의해 정해진다면 eigenvalue의 값이 [0, 1]이기 때문에 계산이 복잡하다. 따라서 approximation이 이용된다.

$$h_l = trace(\mathbf{S}_\lambda), \quad h_l = trace(\mathbf{S}_\lambda \mathbf{S}_\lambda^T)$$

7.3 Nonadaptive Methods

- Nonadaptive methods (linear estimators)
- Statistics: local polynomial estimators, splines
- Neural nets: RBF networks
- Signal processing: wavelet methods

$$f_m(\mathbf{x}, \mathbf{w}) = \sum_{i=1}^{m} w_i g_i(\mathbf{x}) + w_0$$

7.3.1 Local Polynomial Estimators and Splines

- Spline: 일련의 local하게 정의된 low order polynomial들

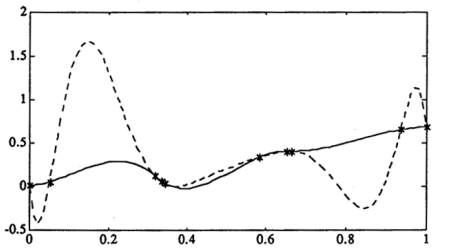


Fig. 7.6 A ninth order polynomial and a cubic spline interpolation of ten data points. The cubic spline provides an interpolation with minimum curvature.

- Natural cubic spline(입력 x가 1차원)
- Multivatirate spline(입력 **x**가 2차원 이상)
  - *d*개의 1차원 spline의 조합
  - Radial basis function(thin-plate splines)
- Aprroximating function

$$f_m(\mathbf{x}, \mathbf{w}, \mathbf{v}) = \sum_{j=1}^{m} w_j g_j(\mathbf{x}, \mathbf{v}_j) + w_0$$

  - basis function:spline basis
  - v_j: knot location
  - m: knot의 갯수

- 1. Nonadaptive knot selection
  - **x**만을 이용하여 knot 결정
  - knot이 결정되면 linear least square 문제
- 2. Adaptive knot selection
  - 함수값 y도 knot 결정에 이용

Regularization Framework와 Cubic Splines의 연결

- Regularization problem

$$R_{pen}(f) = \sum_{i=1}^{n} [f(x_i) - y_i]^2 + \lambda \int_a^b [f''(t)]^2 dt$$

  - λ: fixed complexity parameter
  - $a <= x_1 <= \ldots <= x_n <= b$
- Solution

$$f(x) = \sum_{i=1}^{n+2} w_j B_j(x)$$

- λ: fittting the data와 smoothness간의 trade-off 조절
- w_j의 결정: linear estimation problem with penalty
- Nonparametric penalty도 parametric하게 할 수 있다.
  - Penalty matrix

$$\phi_{ij} = \lambda \int B_i'(t) B_j''(t) dt$$

Multivariate function approximation을 위한 generalization of univariate splines

- Tensor product of d univariate splines
- Example1: Gaussian radial basis

$$g(\mathbf{x}, \mathbf{v}) = \prod_{j=1}^{d} \exp\left(\frac{-(x_j - v_j)^2}{\alpha}\right) = \exp\left(\frac{-\|\mathbf{x} - \mathbf{v}\|^2}{\alpha}\right)$$

- Example2: Tensor-product truncated power basis

$$g(\mathbf{x}, \mathbf{u}, \mathbf{v}) = \prod_{j=1}^{d} [u_j (x_j - v_j)]_+^q$$

7.3.2 Radial Basis Function Networks

- Approximating function

$$f_m(\mathbf{x},\mathbf{w}) = \sum_{j=1}^{m} w_j g\left(\frac{\| \mathbf{x}-\mathbf{v}_j \|}{\alpha_j}\right) + w_0$$

- Normalized RBF

$$f_m(\mathbf{x},\mathbf{w}) = \frac{\sum_{j=1}^{m} w_j g_j}{\sum_{k=1}^{m} g_k}$$

- Nonadaptive RBF training procedure
  1. Number of basis function(m, center의 갯수)의 선택
  2. Center v_j의 선택(unsupervised learning, GLA, SOM)
  3. Heuristic을 이용한 α_i의 결정
  4. Linear least square를 통한 w_i의 결정
- Advantage
  - Fast training procedure
  - Interpretability
- Scaling of input variable가 중요

7.4 Adaptive Dictionary Methods

- Multivariate problem: adaptive method에 대한 동기
- Adaptive method를 분류하는 기준
  - All basis functions of the same/different type
  - Type of basis functions
    - type of dimensionality reduction
    - bounded/unbounded basis functions
  - Optimization strategy

7.4.1 Additive Methods and Projection Pursuit Regression

- Additive model

$$f(\mathbf{x},\mathbf{V}) = \sum_{j=1}^{m} g_j(\mathbf{x},\mathbf{v}_j) + w_0$$

  - $\mathbf{v}_j$: internal parameters (kernel width)
  - $g_j(\mathbf{x},\mathbf{v}_j)$: can be kernel smoother
  - kernel width: data에 맞도록 조정된다.

Projection Pursuit

- Additive model의 specific form (univariate function)

$$f(\mathbf{x}, \mathbf{V}, \mathbf{W}) = \sum_{j=1}^{m} g_j(\mathbf{w}_j \cdot \mathbf{x}, \mathbf{v}_j) + w_0$$

- Invariant to affine coordinate transformation(rotation, scaling) of the input variables

Backfitting (find local minimum of the empirical risk)

- additive approximating function에 대한 empirical risk의 분해

$$R_{emp}(\mathbf{V}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - f(\mathbf{x}_i, \mathbf{V}))^2$$
$$= \frac{1}{n}\sum_{i=1}^{n}([y_i - \sum_{j\neq k} g_j(\mathbf{x}_i, \mathbf{v}_j) - w_0] - g_k(\mathbf{x}_i, \mathbf{v}_k))^2$$
$$= \frac{1}{n}\sum_{i=1}^{n}(r_i - g_k(\mathbf{x}_i, \mathbf{v}_k))^2$$

  - k번째를 제외한 basis function을 고정시키면 risk는 "unexplained" variance로 분해
  - Initial set of basis function들을 가지고 임의의 k에 대한 r_i를 구할 수 있다.
  - g_k의 parameter는 이 variance를 minimize하도록 조정된다.
  - 차례로 각 basis function을 estimate한다.

Backfitting algorithm

1. 모든 x에 대해 $g_j(\mathbf{x}, \mathbf{v}_j) \equiv 0$이 되도록 initialize
   w_0는 $1/n\Sigma y_i$
2. 모든 k에 대해 r_i를 계산하고 risk를 최소화하는 v_k를 구한다.
3. stopping criteria를 만족할 때까지 반복

- Projection pursuit
  - Specific form of backfitting
  - Steepest descent method

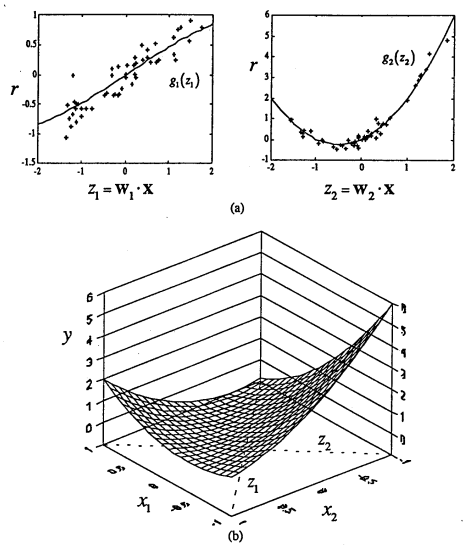


Fig. 7.13 Projection pursuit regression. (*a*) Projections are found that minimize unexplained variance. Smoothing is performed in this space to create adaptive basis functions. (*b*) The approximating function is a sum of the univariate adaptive basis functions.

- Projection pursuit algorithm

1. Backfitting과 같다.
2. a. Residual r_i계산

   b. Projection pursuit

   다음의 과정을 converge할 때까지 반복

   i. $\mathbf{w}_k$고정, risk를 최소화하는 $\mathbf{v}_k$를 찾는다.

   univariate smoother

   ii. 다음 식에 의하여 $\mathbf{w}_k$를 갱신

$$\mathbf{w}_k \leftarrow \mathbf{w}_k - \gamma \frac{\partial R_{emp}(\mathbf{w}_k)}{\partial \mathbf{w}_k}$$

   γ: learning rate

3. Backfitting과 같다.

- Implementation
  - SMART (smooth multiple additive regression technique)
- Smoothing technique
  - Supersmoother (Friedman, 1984)
  - Hermite polynomial
- Linear method g_j에 대해서는 global minimum을 구함
- Nonlinear g_j에 대한 convergence는 보장되지 않는다.
- Basis function의 growing, pruning 사용
- Complexity의 estimation은 어렵다.

7.4.2 Multilayer Perceptrons and Backpropagation

- Basis function

$$g_j(\mathbf{x}, \mathbf{v}_j) = s(\mathbf{x} \cdot \mathbf{v}_j)$$

- Universal approximator
- In terms of representation
  - MLP is a specific case of projection pursuit(고정된 basis function)
- Projection is a specific case of MLP
  - univariate basis function can be represented as a sum of shifter sigmoids

- Target function that vary significantly only in a few directions
  - projection pursuit outperforms MLP
- Estimating a large number of projections
  - MLP outperforms projection pursuit
- Two properties of MLP network
  - Smooth well-behaved sigmoid(saturation limit)
  - Regularization properties of the backpropagation algorithm

Backpropagation algorithm

- Outputlayer

$$\delta_0(k) = \hat{y}(k) - y(k)$$

$$\mathbf{w}_j(k+1) = \mathbf{w}_j(k) - \gamma\delta_0(k)z_j(k)$$

- Hidden layer

$$\delta_{1j}(k) = \delta_0(k)s'(a_j(k))w_j(k+1)$$

$$v_{ij}(k+1) = v_{ij}(k) - \gamma\delta_{1j}(k)x_i(k)$$

- Forward pass

$$a_j = \sum_{i=0}^{d} x_i v_{ij},$$

$$z_j = g(a_j),$$

$$z_0 = 1$$

Adding momentum

$$w(k+1) = w(k) - \gamma\delta(k)z(k) + \mu(w(k) - w(k-1))$$

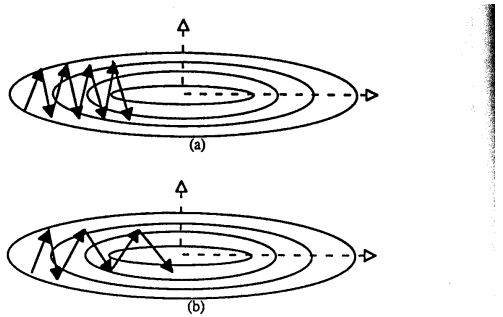


Fig. 7.14 (*a*) For error functionals with different curvatures in different directions, gradient descent with fixed steps produces oscillatory behavior with slow progress toward the valley of the error function. (*b*) Including a momentum term effectively smooths the oscillations, leading to faster convergence on the valley.

On-line (stochastic approximation) and batch

- On-line: less likely to be trapped in a local minimum
- Batch: accurate estimate of the true gradient

Learning rate

- Need to be reduced
- Problem dependent

Premature saturation

- Input value, weight가 큰 경우
- Sigmoid의 derivate가 0에 가까운 경우
- 시간이 오래 걸린다.
- Weight를 작게 준다(so tricky and important)

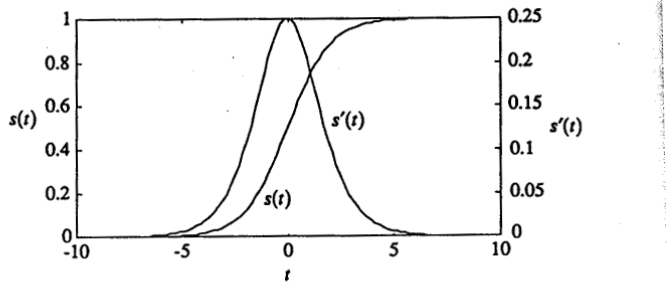


Fig. 7.15 For argument values with a large magnitude, the slope of the sigmoid function is very small, leading to slow convergence.

Complexity Control in MLP

- One-dimensional path through parameter space
  - The solution is on this path
  - and it depends on
    - 1. trainning data and its order
    - 2. the set of nonlinear approximating functions
    - 3. starting point on the path (initial parameter values)
    - 4. final point on the path (the stopping rules)
- 1, 2는 고정
- 3, 4가 regularization effect를 미친다.

- Strucrure on a set of approximating functions
  - 1. Initialization of parameters

$$S_i = \{A: f(\mathbf{x}, \mathbf{w}), \| \mathbf{w}^0 \| \leq c_i\}$$

    - $\mathbf{w}^0$:vextor of initial parameter values
    - *A*: optimization algorithm
    - c_i를 만족시키는 여러 가지값에 대해서 learning
    - 그 중에서 best (global minimum)을 찾는다.
    - time consuming
  - 2. Stopping rules
    - avoid overfitting
    - early stopping rules: 분석이 어렵다.

- 3. Dictionary representation
  - optimal number of hidden units
- 4. Penalization of (large) parameterization values
  - model complexity는 “penalized” risk functional을 minimize하는 것

$$R_{pen}(\omega, \lambda_i) = R_{emp}(\omega) + \lambda_i \| \mathbf{w} \|^2$$

  - 위의 식은 아래와 같다.

$$S_i = \{ f(\mathbf{x}, \mathbf{w}), \| \mathbf{w} \|^2 \leq c_i \}, \qquad c_1 < c_2 < c_3 \ldots$$

  - on-line version에서는 weight decay와 같다.

$$w(k+1) = w(k) - \gamma(\delta(k) z(k) + \lambda w(k))$$

- Regularization effect of initialization(실험)
  - Training data

$$y = \frac{(x-2)(2x+1)}{1+x^2}, \qquad x = [-5,10]$$

    - 15 training samples
    - *y* values: corrupted with gaussian noise
    - input *x* is prescaled to [-0.5, 0.5]
  - Network topology
    - 1 input, output node
    - 8 hidden unit, logical sigmoid
  - Backpropagation implementation
    - on-line, no momentum
    - learning rate: 0.5 (fixed)
    - the number of training epochs: 100,000

- Initialization bounds
  - c is set in [0,30]
  - 각각의 c에 대해서 30번 실험(random initial values)
- Global minimum에 가깝다고 추정되는 것을 선택
- Prediction performance
  - 주어진 c에 대한 best network의 MSE로 측정

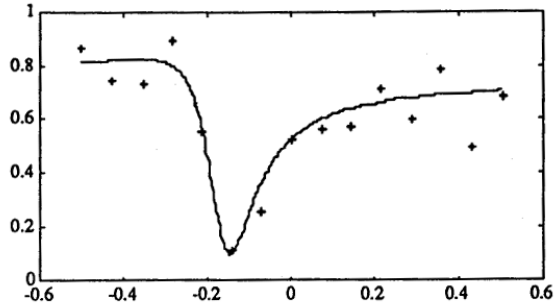
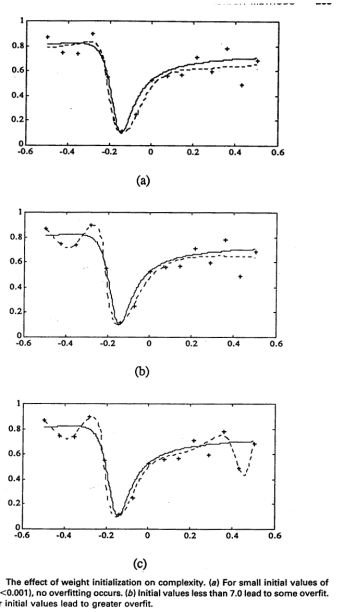


Fig. 7.16 True function and the training data used for the example.



ig. 7.17 The effect of weight initialization on complexity. (a) For small initial values of eights (<0.001), no overfitting occurs. (b) Initial values less than 7.0 lead to some overfit. c) Larger initial values lead to greater overfit.

7.4.3 Multivariate Adaptive Regression Splines

- MARS algorithm
  - Training data에서 adaptive하게 knot locations과 small subset of univariate splines를 선택
  - Recursive partitioning regression
  - Tensor-product splines
- Single linear ($q=1$) tensor product spline basis function

$$g(\mathbf{x},\mathbf{u},\mathbf{v},\Pi)=\prod_{k\in\Pi} b(x_k,u_k,v_k)$$

- *b*: univariate basis function
- **v**: knot locations
- **u**: orientation vector {-1,1}
- Π: subset of input variables (adaptively selected)
- knot locations: all possible combinations of individual coordinate values existing in the data

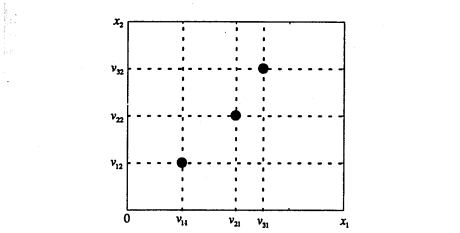


Fig. 7.18 Valid knot locations for MARS occur at all combinations of coordinate values existing in the data. For example, three data points in a two-dimensional input space lead to nine valid knot locations indicated by the intersections of the dashed lines.

MARS approximating function

$$f_m(\mathbf{x},\mathbf{v},\mathbf{U},\mathbf{V},\{\Pi_1,...,\Pi_m\}) = \sum_{j=1}^{m} w_j \prod_{k\in\Pi} b(x_k,u_{jk},v_{jk}) + w_0$$

- Greedy optimization 사용
- Tree 형태
  - left-right pair of basis function b^+, b^-
- If $\boldsymbol{g_{parent}}(\mathbf{x})$ denotes a parent node

$$g_{daughter+}(\mathbf{x}) = b^+(x_k,v_j)\cdot g_{parent}(\mathbf{x})$$

$$g_{daughter-}(\mathbf{x}) = b^-(x_k,v_j)\cdot g_{parent}(\mathbf{x})$$

  - v_j is knot for x_k
- Depth of a tree: interaction level

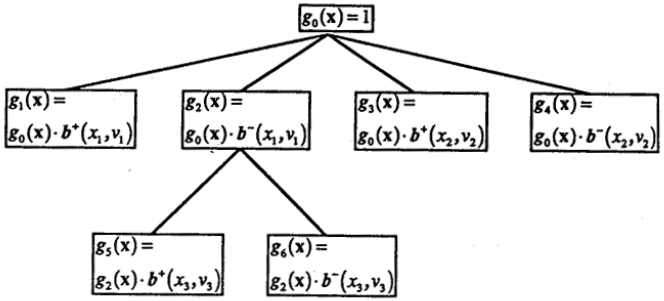


Fig. 7.19 Example of a MARS tree.

Measure of fit: generalized cross validation (gcv) estimate

Model complexity estimate:

1. Determine the degrees of freedom assuming a nonadaptive basis
2. Add a correction factor to take into account the adaptive basis construction

$$h_{mars} \approx (1+\eta)m$$

  - m: estimate for the equivalent degrees of freedom of estimating parameters **w**
  - η : adaptive correction factor (2 - 4)

- Search strategy
  - m_{max}: the maximum number of basis functions
  - t_{max}: limit in the interaction degree
1. Initialization, root node is $g_0(x) = 1$, w_0, via mean of the response data
2. Forward stepwise selection, repeat following until the tree has m_{max} nodes
   a. Exhaustive search over all valid nodes, all valid split variables, all valid knots. Create a pair of daughters, estimate w (linear problem), and estimate complexity h_{max}
   b. Incorporate the daughters into a tree that result in the largest decrease of prediction risk estimated using gcv.
3. Backward stepwise selection, repeat the following for m_{max} iterations
   a. Exhaustive search over all nodes in the tree, the change in model selection criterion *gcv* resulting from removal of each node.

   b. Delete the node that leads to the largest decrease of *gcv* or smallest increase.
4. Of the series of models created by the backward stepwise selection, choose the best *gcv* score model as the final model.

- Interpretation of MARS(by ANOVA decomposition)
  - Regrouping the additive terms in function approximation

$$\tilde{f}(\mathbf{x}) = \sum_{k=1}^{m} w_k g_k(\mathbf{x}) + w_0 = w_0 + \sum_{i=1}^{d} f_i(x_i) + \sum_{i,j=1}^{d} f_{ij}(x_i, x_j) + \ldots$$

  - Isolate the effect of a particular input variable
- Coordinate rotation에 민감
- 장점: speed of execution, interpretation, relatively automatic smoothing parameter selection

7.5 Adaptive Kernel Methods and Local Risk Minimization

- Local estimation
  - 하나의 data $\mathbf{x}_0$에 대한 estimation
- Local risk functional

$$R(\omega, \alpha; \mathbf{x}_0) = \int L(y, f(\mathbf{x}, \omega)) \frac{K_a(\mathbf{x}, \mathbf{x}_0)}{\kappa_\alpha(\mathbf{x}_0)} p(\mathbf{x}, y) d\mathbf{x} dy$$

$$\kappa_\alpha(\mathbf{x}_0) = \int K_\alpha(\mathbf{x}, \mathbf{x}_0) p(\mathbf{x}) d\mathbf{x}$$

  - $K_\alpha(\mathbf{x}, \mathbf{x}_0)$: estimation point $\mathbf{x}_0$근처의 local neighborhood(만약 1인 경우라면 global risk minimization과 같다.)
  - $\kappa_\alpha(\mathbf{x}_0)$: normalizing function

  - 실제로는 f 함수를 고정시키고 α를 찾는다.
  - Neighborhood size α controls model complexity
    - large α: low complexity
- Local empirical risk for estimation point $\mathbf{x}_0$

$$R_{emp-local}(\omega) = \frac{1}{n} \sum_{i=1}^{n} K_\alpha(\mathbf{x}_i, \mathbf{x}_0)(y_i - f(\mathbf{x}_i, \omega))^2$$

  - If approximating function $f(\mathbf{x}, w_0) = w_0$, zero-order model, risk is minimized when

$$f(\mathbf{x}_0) = w_0 = \frac{1}{n} \sum_{i=1}^{n} y_i K_\alpha(\mathbf{x}_i, \mathbf{x}_0)$$

  - and this is local average, kernel approximation

- Approximation은 한 점에서만 이루어진다.
  - Memory based: training data가 prediction 시점까지 존재해야 하므로
- Practical method for kernel width selection
  - *k*-nearest neighbors regression 등
- *k*-nearest neighbors technique
  - local risk minimization (take local average of the data)
  - locality: estimation point에 가까운 k개의 data points에 의해 정의된다.
- *k*의 조정 방법
  1. Nonadaptive approach
  2. Global adaptive approach
  3. Local adaptive approach(for the estimation point)

- Local model selection (small sample problem)
  - *k*의 결정
  - difficult
  - 따라서 practical하게는 global model selection을 한다.
  1. 주어진 k에 대해서 각 $\mathbf{x}_i$에 대한 local estimate를 계산
  2. 이 것을 가지고 global empirical risk을 계산

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

  3. 이를 최소화하는 적절한 *k*를 결정

  사실은 *k*를 estimation point $\mathbf{x}_0$에 따라 적절히 결정해야 하지만 어려우므로 위의 방법을 쓴다.

7.5.1 Generalized Memory-Based Learning

- Learn by example
- Local approximation using the past data
- Kernel width, distance scale: global하게 조정 (by cross-validation)
- 주어진 kernel width α, linear approximating function에 대한 local emprical risk

$$R_{emp-local}(\mathbf{w}, w_0) = \frac{1}{n}\sum_{i=1}^{n} K_\alpha(\mathbf{x}_i, \mathbf{x}_0)[\mathbf{w}\cdot\mathbf{x}_i + w_0 - y_i]^2$$

- Risk minimization (locally weighted scatterplot smoothing)
  1. Weighing the data by the kernel function

$$\mathbf{x}_i^{'} = \mathbf{x}_i K_\alpha(\mathbf{x}_i, \mathbf{x}_0), \qquad y_i^{'} = y_i K_\alpha(\mathbf{x}_i, \mathbf{x}_0)$$

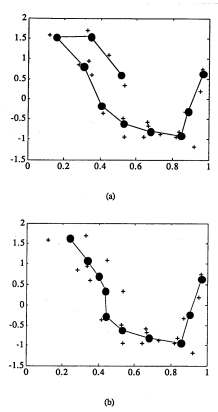
  2. Linear estimation
- GMBL kernel

$$K(\mathbf{x}, \mathbf{x}', \mathbf{v}) = (\sum_{k=1}^{d}(x_k - x_k^{'})^2 v_k^2)^{-q}$$

  - **v**: distance scaling 조정
  - *q*(> 0): width of kernel function 조정

7.5.2 Constrained Topological Mapping

- CTM:
  - Regression에 적합하도록 SOM을 수정한 것 (clustering via SOM + regression via piecewise-constant splines)
- Center of SOM:
  - Dynamically movable knots for spline regression
- Piecewise-constant spline approximation:
  - Training the SOM with m-dimensional feature space($m <= d$) using data samples $\mathrm{x}' = (\mathrm{x}_i, y_i)$ in $(d+1)$-dimensional input space



- Functionality가 보장되지 않는다. (x, y의 구별없음)
- Overcome:
  - Dimensionality reduction in the **x**-space
  - Feature space를 input으로 하여 각 knot에 대한 y 값을 kernel averaging으로 구한다.

- Number of knots:
  - model complexity
  - during the process, neighborhood width decreases
- CTM algorithm
  - discrete feature space $\mathbf{\Psi} = \{\Psi_{1,\ldots}\Psi_b\}$
  - data point $\mathbf{x}' = (\mathbf{x}_i, y_i)$
  - units $\boldsymbol{c}_j(k), j = 1, \ldots, b$

1. Determine the nearest unit to the data point

$$\mathbf{z}(k) = \Psi(\arg\min_j \sum_{l=1}^{d} (x_l(k) - c_{jl}(k-1))^2)$$

2. Update all units

$$\mathbf{c}_j(k) = \mathbf{c}_j(k-1) + \beta(k) K_{\alpha(k)}(\Psi(j), \mathbf{z}(k))(\mathbf{x}'(k) - \mathbf{c}_j(k-1)),$$
$$j = 1,\ldots,b \qquad k = k+1$$

3. Decrease the learning rate and neighborhood width

- CTM lacks in some key features

1. Picewise-linear vs. piecewise-constant
   - Less accurate than piecewise-linear
2. Control of model complexity
   - By user, iterative cross-validation
3. Adaptive regression via global variable selection
   - No information about variable importance
   - Adaptive predictor variable scaling
4. Batch vs. flow-through implementation
   - Flow-through has some disadvantage

- CTM batch version

0. Initialization. Center c_j와 distance scale parameter $v_l = 1$을 초기화
1. Projection.

$$\| \mathbf{c}_j - \mathbf{x}_i \|_{\mathbf{v}}^2 = \sum_{l=1}^{d} v_l^2 (c_{jl} - x_{il})^2$$

2. Conditional expectation (smoothing) in **x**-space. Update centers

$$F(\mathbf{z}, \alpha) = \frac{\sum_{i=1}^{n} \mathbf{x}_i K_\alpha(\mathbf{z}, \mathbf{z}_i)}{\sum_{i=1}^{n} K_\alpha(\mathbf{z}, \mathbf{z}_i)}$$

$$\mathbf{c}_j = F(\Psi(j), \alpha), \qquad j = 1, \ldots, b$$

3. Conditional expectation in *y*-space. Minimize the following.

$$R_{emp-local}(\mathbf{w}_j, w_{0j}) = \frac{1}{n} \sum_{i=1}^{n} K(\mathbf{z}_i, \Psi(j))[\mathbf{w}_j \cdot \mathbf{x}_i + w_{0j} - y_i]^2$$

4. Adaptive scaling. Determine new scaling parameter **v** for each of the *d* input variables using average sensitivity for each predictor dimension

$$v_l = \sum_{j=1}^{b} | w_{jl} |$$

If scaling parameters are normalized, they can be interpreted as variable importance.

5. Model selection. Decrease α, and repeat 1 - 4 until the leave-one-out cross-validation reaches a minimum.

7.6 Empirical Comparisons

- Comparisons of algorithms
- Almost fully automatic methods(많아야 한 두 개의 parameter를 user가 조정)
- Artificial datasets
- Robustness:
  - small change in trainning data가 가져오는 결과를 비교

7.6.1 Experimental Setup

- Comparison goal
  - Predictive performance
  - Non-expert user가 대상
  - Computational time은 거의 고려하지 않음
- Comparison methodology
  - 각각의 method는 4개의 setting으로 실행되고 그 중 가장 뛰어난 것을 채택. 이 것으로 각 method 간의 성능을 비교

- Experiment design
  - Types of functions used to generate samples
  - Properties of the training and test data sets
  - Specification of performance metric used for comparisons
  - Description of modeling methods
- Functions used
  - 8개의 representative 2-variable functions
  - Gaussian noise: no noise, medium noise, high noise
- Test data
  - No noise
- Performance metric
  - RMS of the test set

- Learning method implementations
  - Projection Pursuit Regression
  - MLP
  - Multivariate Adaptive Regression Spline
  - k-Nearest Neighbor
  - Generalized Memory Based Learning
  - Constrained Topological Mapping

	Best	*Worst*
Prediction accuracy (dense samples)	MLP	KNN, GMBL
Prediction accuracy (sparse samples)	GMBL, KNN	MARS, PP
Additive target functions	MARS, PP	KNN, GMBL
Harmonic target functions	CTM, MLP	PP
Radial target functions	MLP, PP	KNN
Robustness wrt parameter tuning	MLP, GMBL	PP
Robustness wrt sample properties	MLP, GMBL	PP, MARS

7.7 Combining Predictive Models

- Typical model combination
  1. 각각의 model을 training data로 training. Model들의 parameter가 고정.
  2. Linear combination of individual models
- Committee of networks
  - 각 stage에서의 risk minimization
- Stacking predictors
  - Resampling technique

- Committee of networks

$$f_{com}(\mathbf{x},\alpha) = \frac{1}{b}\sum_{j=1}^{b}\alpha_j f_j(\mathbf{x},\omega_j^*)$$

  - α_j: degree of belief

$$R(\alpha) = \frac{1}{n}\sum_{i=1}^{n}(f_{com}(\mathbf{x},\alpha) - y_i)^2$$

$$\sum_{j=1}^{b}\alpha_j = 1, \qquad \alpha_j \geq 0$$

Predictor stacking algorithm

1. Resampling. "Left-out" sample $(\mathbf{x}_i, y_i)$와 각각의 candidate method $f_j(\mathbf{x},\omega_j)$에 대해
   a. *n-1*개의 sample로 각 model estimate

$$f_{ij}(\mathbf{x}, \omega_{ij}^*)$$

   b. prediction for "left-out" sample을 저장

$$\hat{y}_{ij} = f_{ij}(\mathbf{x}, \omega_{ij}^*)$$

2. Estimation of linear coefficients.

$$R(\alpha) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \sum_{j=1}^{b}\alpha_j \hat{y}_{ij})^2$$

- Additional step
  - 모든 sample을 이용하여 final model을 estimate

$$f_j^*(\mathbf{x}, \omega_j^*)$$

  - Combined model의 구성

$$f(\mathbf{x}) = \sum_{j=1}^{b}\alpha_j^* f_j^*(\mathbf{x}, \omega_j^*)$$