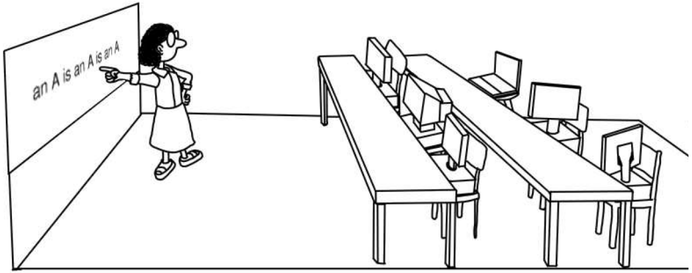
Machine Learning

Krystian Mikolajczyk & Deniz Gunduz

Department of Electrical and Electronic Engineering
Imperial College London



Part 3.2 summary

Learning with noisy data: Which $\mathcal{H} \sim \Phi(\mathbf{x})$ should we choose?

Given many hypotheses classes $\mathcal{H}_1, \mathcal{H}_2, \ldots$

Which $\mathcal{H}_i$ should we choose g from?

Use data to select g!

Structural Risk Minimization

Regularisation

Validation

Learning with noisy data: Structural Risk Minimization (SRM)

Structural Risk Minimization

- Hypothesis class: $\mathcal{H} = \cup_i \mathcal{H}_i$
- Generalization bound: for all i w.p. at least $1 - \delta$,
$$R(h) \leqslant \widehat{R}_n(h) + \Omega(n, \mathcal{H}_i, w_i\delta).$$
- SRM solution:
$$g = \mathrm{argmin}_{h \in \mathcal{H}} \widehat{R}_n(h) + \Omega(n, \mathcal{H}(h), w_i\delta)$$
    - $\mathcal{H}(h)$ is the simplest class h belongs to, and w_i is the class weight.
    - Training error $\widehat{R}_n(h)$ plus complexity term Ω (from VC inequality).
    - Identical to ERM if there is only one class $\mathcal{H}$.

Example

$$\mathcal{H}_1 : \Phi_1(\mathbf{x}) = (1, x_1, x_2)$$

$$\mathcal{H}_2 : \Phi_2(\mathbf{x}) = (1, x_1, x_2, x_1^2, x_1 x_2, x_2^2)$$

$$\mathcal{H}_3 : \Phi_3(\mathbf{x}) = (1, x_1, x_2, x_1^2, x_1 x_2, x_2^2, x_1^3, x_1^2 x_2, x_1 x_2^2, x_2^3, x_1^4, x_1^3 x_2, x_1^2 x_2^2, x_1 x_2^3, x_2^4)$$

From VC inequality:

$$\Omega(n, \mathcal{H}_i, w_i\delta) = \sqrt{\frac{8 d_{VC}(\mathcal{H}_i)}{n} \log \frac{2ne}{d_{VC}(\mathcal{H}_i)} + \frac{8}{n} \log \frac{4}{w_i \delta}}$$

$$= \begin{cases} 1.298 & i = 1, n = 100 \\ 1.612 & i = 2, n = 100 \\ 2.178 & i = 3, n = 100 \end{cases}$$

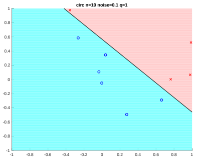
$\Omega(n, \mathcal{H}_i, \delta)$ is too large: $\Omega > 1$

Use $c\Omega(n, \mathcal{H}_i, \delta)$ instead, e.g., for $c = 0.1$

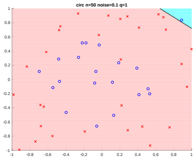
Use $\widehat{R}_n(h) + c\Omega(n, \mathcal{H}_i, \delta)$ to choose g.

Learning with noisy data: Structural Risk Minimization

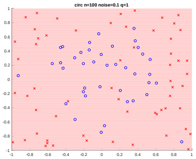
Columns $n \in \{8, 50, 100, 1000\}$, rows $q \in \{1, 2, 4\}$, % of trials selected by SRM, with train and test errors



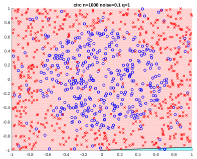
9% tr=0.189 te=0.481



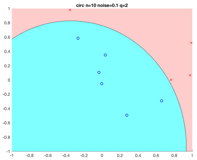
0% tr=0.332 te=0.462



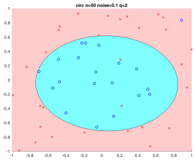
0% tr=0.364 te=0.44



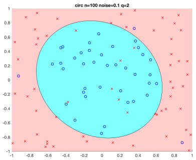
0% tr=0.402 te=0.417



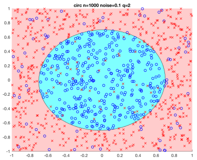
87% tr=0.004 te=0.334



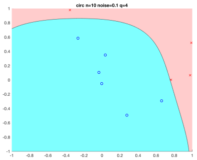
70% tr=0.085 te=0.185



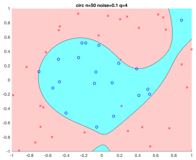
99% tr=0.097 te=0.150



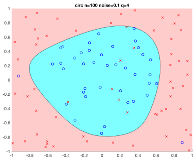
100% tr=0.106 te=0.110



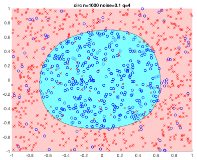
4% tr=0 te=0.328



30% tr=0.031 te=0.243



1% tr=0.084 te=0.186



0% tr=0.111 te=0.119

Example

In an ML regression task you decided to use a non-linear transformation of the input data with polynomials. You observe that the training error is zero but the test error is much larger

- Give an example of a target function $f(x) = ?$, polynomial feature transformation $\mathbf{z} = \Phi(x) = \{\ldots\}?$ and error function $\widehat{R} = ?$ for the given scenario. In what specific training setting this may occur and how to improve the test error?
- Assume you fix the polynomial degree to $k = 3$. What is the approximate number of data points that are needed if you want to guarantee that the test error of the predictor is within 0.1 from the training error with probability 0.99? Show your calculations.

Example

Suppose that n data points are generated according to a polynomial of degree q

$f(x) = y = ax^q + bx^{q-1} + \ldots$, where x is a real feature $x \in \mathbb{R}$ value uniformly distributed on $[0, 1]$

y is the target variable

consider the squared error $\widehat{R} = \frac{1}{n} \sum_n (x - y)^2$.

Given $n > q$ data points, if we chose the hypothesis class to be polynomials degree $n - 1$:

$\Phi(x) = \{1, x, x^2, \ldots, x^{n-1}\}$

the training error will be 0 (as any n points can be fitted properly),

but the test error of the learned polynomial will be large as the complexity of the hypothesis class is too high.

To improve the test error reduce the polynomials degree, use Legendre polynomials, and apply regularisation.

Example

We need to choose n large enough so that $|R - \widehat{R}_n| \leqslant 0.1$ with probability at least 99%.

The probability $P.() = 0.01$ of error larger than $\varepsilon = 0.1$

for the VC dimension $d_{VC} = k + 1 = 4$, since $\mathbf{z} = \Phi(x) = \{1, x, x^2, x^3\}$.

By VC's inequality we get

$$P\left(|R - \widehat{R}_n| > 0.1\right) \leqslant 4n^{d_{VC}} e^{-\varepsilon^2 n/8} = 4n^4 e^{-n \cdot 0.1^2/8} = 0.01.$$

trying e.g. $n \in \{1000, 10000, 50000\}$ one can quickly converge to 40000.

Example

A Transformer thinks that the following procedures would work well in learning from two-dimensional data sets of any size. Point out if there are any potential problems in the procedures:

(a) Use the transform

$$\Phi(x) = \begin{cases} (\underbrace{0, \ldots, 0}_{i-1}, 1, 0, \ldots) & \text{if } x = x_i; \\ (0, 0, \ldots, 0) & \text{otherwise.} \end{cases}$$

Hint:Take a few 2D points from a known target function, generate their transformed vectors and see if the prediction is still possible.

Example

Solution (a): This transformation is meaningless, as each data point is represented by its index only (which is arbitrarily assigned). In this way, any information present in the data is lost.

Example

(b) Use the transform $\Phi = (\Phi_1, \dots, \Phi_n)$ with

$$\Phi_i(x) = e^{-\frac{\|x - x_i\|^2}{2\gamma^2}}$$

using some very small γ.

Hint: Take a few 2D points from a known target function, generate their transformed vectors and see if the prediction is still possible.

Example

Solution (b): This is a valid feature transformation (SVM with Gauss kernel) and often works well in practice (however, γ should be tuned properly).

Example

(c) Use the transform Φ consisting of all

$$\Phi_{i,j}(x) = e^{-\frac{\|x-(i,j)\|^2}{2\gamma^2}}$$

with $i, j \in \{0, \frac{1}{100}, \dots, 1\}$

Hint: Take a few 2D points from a known target function, generate their transformed vectors and see if the prediction is still possible.

Example

Solution (c): This is also a valid feature transformation, assuming it is decided without looking at the data. Since the basis-points (i, j) of the transformation are fixed, the transform can adapt to the data less than the one in (b). In particular, if the range of the data is far from $[0, 1]^2$, it is not likely to work without further transformation; e.g., if $x \in [10^6, 10^6 + 1]^2$, all features are essentially zero, and it is computationally hard (due to numerical problems) to recover any information from the features. Furthermore, the transform is 10000 dimensional, so the resulting feature space and hypothesis class might be too rich, so strong regularization might be needed. (Again, even if the data is in $[0, 1]^2$, γ should be tuned properly.)

Regularisation: Linear regression

Minimize

$$\widehat{R}_n = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{w}^\top \mathbf{z}_i - y_i)^2$$
$$= \frac{1}{n}(Z\mathbf{w} - y)^\top (Z\mathbf{w} - y)$$

unconstrained solution $\Rightarrow \mathbf{w}_{lin} = (Z^\top Z)^{-1} Z^\top y.$

Will overfit if complexity of $\mathbf{z} = \Phi(\mathbf{x})$ is too high (e.g. high degree q)!

Constrain the function class!

- Hard constraint: $\mathcal{H}_Q$: $w_q = 0$ for all $q > Q$! (how to choose Q?)

  E.g., $\mathcal{H}_2$ is a constrained version of $\mathcal{H}_{10}$.

- Soft constraint: $\sum_{i=0}^{Q} w_q^2 \leqslant C$.

  E.g. Small polyn. coeffs $\rightarrow$ function value varies little with varying input.

Regularisation: Solving constrained optimization, L_2 penalty

Minimize

$$\widehat{R}_n = \frac{1}{n}(Z\mathbf{w} - y)^\top (Z\mathbf{w} - y)$$

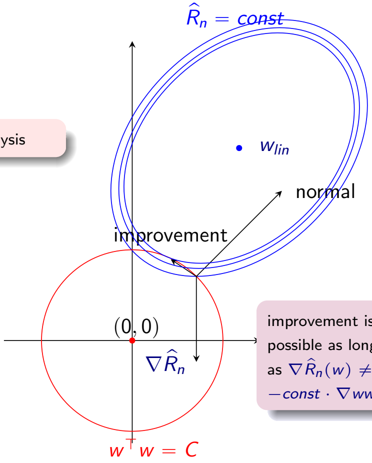
$$\text{subject to} \quad \mathbf{w}^\top \mathbf{w} \leqslant C$$

← Restricting $\mathcal{H}$ ← VC analysis

$\|\mathbf{w}\|_2^2 = \sum_i w_i^2 = \mathbf{w}^\top \mathbf{w}$

If $\mathbf{w}_{lin}^\top \mathbf{w}_{lin} \leqslant C$ then $g(\mathbf{z}, \mathbf{w}) = \mathbf{w}_{lin}^\top \mathbf{z}$

If $\mathbf{w}_{lin}^\top \mathbf{w}_{lin} > C$?



improvement is possible as long as $\nabla \widehat{R}_n(w) \neq -const \cdot \nabla ww^\top$

Loss: $\mathcal{L}_n(w) = \widehat{R}_n(w) + \frac{\lambda}{n}\|w\|_2^2 \quad (\lambda > 0)$

Lagrangian

Why $\frac{2\lambda}{n}$? $\widehat{R}_n = \frac{\mathbf{1}}{\mathbf{n}} \sum \ell(h(x), y)$

Optimum: $\nabla \widehat{R}_n(w_{reg}) = -\frac{2\lambda}{n} w_{reg}$

$$\nabla \widehat{R}_n(w_{reg}) + \frac{2\lambda}{n} w_{reg} = 0$$

Ridge regression solution

$$\lambda \downarrow \Leftrightarrow C \uparrow$$

Regularisation: Minimizing the Lagrangian

Minimize

$$\begin{aligned}\mathcal{L}_n(w) &= \widehat{R}_n(w) + \frac{\lambda}{n}\|w\|_2^2 \\ &= \frac{1}{n}(Zw-y)^\top(Zw-y) + \frac{\lambda}{n}\|w\|_2^2\end{aligned}$$

Setting $\nabla\mathcal{L}_n(w) = 0$

$$Z^\top(Zw-y) + \lambda w = 0$$

Optimal regularized weight – Ridge regression

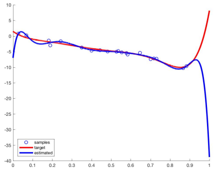
$$w_{reg} = (Z^\top Z + \lambda I)^{-1} Z^\top y$$

Unconstrained optimum

$$w_{lin} = (Z^\top Z)^{-1} Z^\top y$$

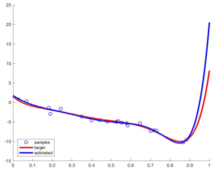
Regularisation: regression with Legendre polynomials

$\lambda = 0$



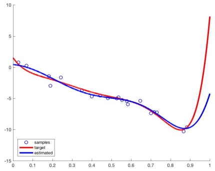
overfitting

$\lambda = 0.0001$



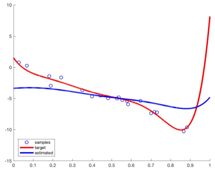
$\longrightarrow$

$\lambda = 1$



$\longrightarrow$

$\lambda = 100$



underfitting

Regularization: Tikhonov

Minimize Loss:

$$\mathcal{L}_n(w) = \widehat{R}_n(w) + \lambda\|\mathbf{w}\|_Q^2$$

Instead of $\|\mathbf{w}\|_2^2 = \mathbf{w}^\top\mathbf{w}$ put emphasis on certain weights

$$\|\mathbf{w}\|_Q^2 = \sum_{q=0}^{Q} \gamma_q w_q^2$$

Examples:

- $\gamma_q = 2^q$ low-order fit
- $\gamma_q = 2^{-q}$ high order fit

General form: Tikhonov regularizer

Tikhonov regularizer: $\|\mathbf{w}\|_Q^2 = \mathbf{w}^\top\Gamma^\top\Gamma\mathbf{w}$, with $Q = \Gamma^\top\Gamma$,

where Γ is a matrix with weights (and cross term weights)

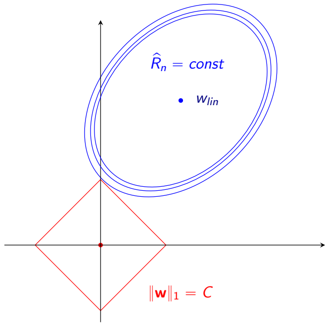
Regularisation: LASSO L_1 penalty

- Lasso: Least absolute shrinkage and selection operator
  - when $\mathbf{x} \in \mathbb{R}^d$ and $d \approx n$
  - need to reduce d to avoid overfitting
- Select useful dimensions d

  $\Rightarrow$ constrain $\|\mathbf{w}\|_0 = \sum_{j=1}^{d} \mathbb{I}\left(\mathbf{w}_i \neq 0\right)$

  Computationally very hard (non-convex)
- Instead, constrain $\|\mathbf{w}\|_1 = \sum_{j=1}^{d} |\mathbf{w}_j|$
  - Solve $\widehat{R}_n(\mathbf{w}) + \lambda\|\mathbf{w}\|_1$

The ellipse $\widehat{R}_n$ is likely to touch the corner of $\|\mathbf{w}\|_1 = C$ where a subset of dimensions is 0



Sparse solution!

Feature selection

Lasso

$$\text{minimize} \quad \widehat{R}_n = \frac{1}{n}(Z\mathbf{w} - \mathbf{y})^\top (Z\mathbf{w} - \mathbf{y})$$

$$\text{subject to} \quad \|\mathbf{w}\|_1 \leqslant C$$

Regularisation: Variants

Regularised Loss (Augmented Error)

$$\mathcal{L}_n(h) = \widehat{R}(h) + \lambda \underbrace{\Omega(h)}_{\textit{overfit penalty}}$$

VC inequality

$$R(h) \leqslant \widehat{R}(h) + \underbrace{\Omega(n, \mathcal{H}, \sigma)}_{\textit{complexity penalty}}$$

- $\mathcal{L}_n(h)$ is a better approximation of $R(h)$
- $\Omega(h)$ – regulariser
  - $\|\mathbf{w}\|_2^2$: ridge regression
  - $\|\mathbf{w}\|_1$: lasso
  - $\|\mathbf{w}\|_1 + \|\mathbf{w}\|_2^2$: elastic net
  - $\mathbf{w}^\top \Gamma^\top \Gamma \mathbf{w}$: Tikhonov
- Generalization bounds also hold

Occam's razor: Simpler is usually better

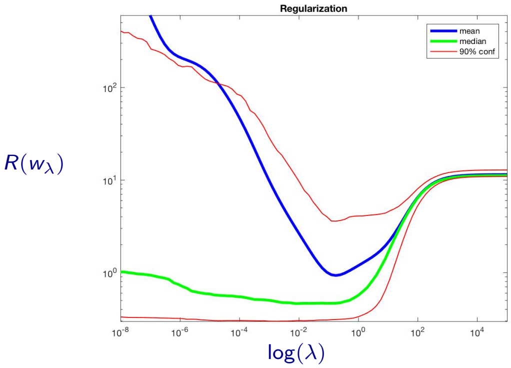
Regularize towards smoother, simpler functions. Why?

Because noise is not smooth!

How to select the regularization coefficient λ?

Noise types: stochastic ($N \sim \mathcal{N}(0, \sigma^2)$), deterministic (complexity e.g. Q)

1000 experiments of
polynomial fit with 20 points.



deterministic noise	$\uparrow$	overfitting	$\uparrow$
stochastic noise	$\uparrow$	overfitting	$\uparrow$
number of data points	$\uparrow$	overfitting	$\downarrow$
λ	$\uparrow$	overfitting	$\downarrow$

From regularisation point of view, deterministic noise behaves same as the stochastic noise

How to select λ ?Validation!

Validation

Regularisation

$$\mathcal{L}_n(h) = \widehat{R}_n(h) + \lambda \underbrace{\Omega(h)}_{overfit\ \ penalty}$$

Validation

$$\underbrace{\mathcal{L}_n(h)}_{direct\ \ estimation} = \widehat{R}_n(h) + \lambda \underbrace{\Omega(h)}_{overfit\ \ penalty}$$

1. Split the training data $\mathcal{D}$ into training $\mathcal{D}_{train}$ and validation $\mathcal{D}_{val}$ sets.
2. Train g on $\mathcal{D}_{train}$.
3. Estimate its performance on $\mathcal{D}_{val}$ ($v = |\mathcal{D}_{val}|$):

$$\check{R}_v(g) = \frac{1}{v} \sum_{(x_i,y_i)\in\mathcal{D}_{val}} \ell(g(x_i), y_i)$$

Very good estimate of $R(g)$

$$\mathbb{E}_{\mathcal{D}_{val}}\left[\check{R}_v(g)\right] \approx R(g) \leqslant \check{R}_v(g) + \underbrace{\Omega(v,\delta)}_{\sim\sqrt{\log(1/\delta)/v} \leftarrow\ one\ \ model\ \ only\ \ on\ \ v-points} \quad \text{w. p. } 1-\delta$$

- D_{val} is unbiased, small Hoeffding bound, only one g is considered.
- Select $\lambda^* = \text{argmin}_\lambda \check{R}_v(g)$, then train on the whole $\mathcal{D}$ with λ^*.

Validation: More generally

Validation

Given hypothesis classes $(\mathcal{H}_1, \lambda_1), \dots, (\mathcal{H}_i, \lambda_i), \dots, (\mathcal{H}_M, \lambda_M)$,

1. Split training data $\mathcal{D}$ into $\mathcal{D}_{train}$ and $\mathcal{D}_{val}$ sets.
2. Train g_i on $\mathcal{D}_{train}$, $\quad |\mathcal{D}_{train}| = n - v$
3. Select i_* such that $g_{i_*} = \text{argmin}_i \check{R}_v(g_i)$ on $|\mathcal{D}_{val}| = v$
   - i - not only regularisation, can be other hyperparameter.
4. Select $\mathcal{H}_{i_*}, \lambda_{i_*}$ and train new g on $\mathcal{D}_{val} \cup \mathcal{D}_{train}$ to get the final g_*.

- Cost? $\quad (n - v)$ and learning curves.
- if $v \uparrow$ then $\check{R}_v(g) \sim R(g) \uparrow$, but $|\mathcal{D}_{train}| = n - v \downarrow$ then $R(g) \uparrow$
- How big $|\mathcal{D}_{val}| = \frac{|D|}{5}$
- Why "validation" and not "test" set? Unlucky split of dataset $\mathcal{D}$?

Cross validation

$$\begin{array}{|c|c|c|c|c|}\hline \colorbox{red}{$\mathcal{D}_1$} & \mathcal{D}_2 & \mathcal{D}_3 & \cdots & \mathcal{D}_K \\ \hline \mathcal{D}_1 & \colorbox{red}{$\mathcal{D}_2$} & \mathcal{D}_3 & \cdots & \mathcal{D}_K \\ \hline \mathcal{D}_1 & \mathcal{D}_2 & \colorbox{red}{$\mathcal{D}_3$} & \cdots & \mathcal{D}_K \\ \hline \vdots & \vdots & \vdots & \ddots & \vdots \\ \hline \mathcal{D}_1 & \mathcal{D}_2 & \mathcal{D}_3 & \cdots & \colorbox{red}{$\mathcal{D}_K$} \\ \hline \end{array} \quad \begin{array}{l} \to \quad \check{R}_1(g_{\bar{1}}) \\ \to \quad \check{R}_2(g_{\bar{2}}) \\ \to \quad \check{R}_3(g_{\bar{3}}) \\ \\ \to \quad \check{R}_K(g_{\bar{K}}) \end{array}$$

Validation error of $g_{\bar{k}}$ on $\mathcal{D}_k$ is

$$\check{R}_k(g_{\bar{k}}) = \frac{1}{|\mathcal{D}_k|} \sum_{(\mathbf{x}_i, y_i) \in \mathcal{D}_k} \ell(g_{\bar{k}}(\mathbf{x}_i), y_i) \text{ with } \bar{n} = (K - 1/K)n$$

Cross-validation error

$$\check{R}_{CV} = \frac{1}{K} \sum_{k=1}^{K} \check{R}_k(g_{\bar{k}}), \qquad \mathbb{E}_{\mathcal{D}}\left[\check{R}_{CV}\right] \approx \mathbb{E}_{\mathcal{D}_k}\left[R(g_{\bar{k}})\right]$$

Typical K choices:

- $K = 10$: 10-fold cross validation
- $K = n$: leave-one-out cross validation

Learning curves: $\hat{R}$, $\check{R}$, R vs. iterations

Cross validation

K-fold Cross Validation

Given: hypothesis classes $(\mathcal{H}_1, \theta_1), \dots, (\mathcal{H}_i, \theta_i), \dots, (\mathcal{H}_M, \theta_M)$,

with θ – any hyperparameter

K training sets $\mathcal{D}_{\bar{k}} = \mathcal{D}_1 \cup \mathcal{D}_2 \cup \dots \xcancel{\mathcal{D}_k} \cup \dots \cup \mathcal{D}_K$, $|\mathcal{D}_{\bar{k}}| = \bar{n} = \frac{K-1}{K} n$

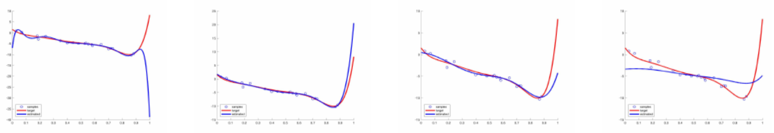
K validation sets $\mathcal{D}_k$, $|\mathcal{D}_k| = \bar{k} = \frac{n}{K}$

1. Train $g_{i,\bar{k}}$ with ERM for every $(\mathcal{H}_i, \theta_i)$ and every $\mathcal{D}_{\bar{k}}$
2. Compute $\check{R}_{CV}(g_i)$ for every g_i on all $\mathcal{D}_k$
3. Select $g_{i_*} = \operatorname{argmin}_{g_i} \check{R}_{CV}(g_i)$
4. Use $(\mathcal{H}_{i_*}, \theta_{i_*})$ and whole data set $\mathcal{D}$ to train final g_*

Regularized Loss Minimization

Regularization

$\lambda = 0$ $\lambda = 0.0001$ $\lambda = 1$ $\lambda = 100$



overfitting $\longrightarrow$ $\longrightarrow$ underfitting

Validation

$\mathcal{D}_1$	$\mathcal{D}_2$	$\mathcal{D}_3$	$\cdots$	$\mathcal{D}_K$	$\rightarrow$	$\breve{R}_1(g_{\bar{1}})$
$\mathcal{D}_1$	$\mathcal{D}_2$	$\mathcal{D}_3$	$\cdots$	$\mathcal{D}_K$	$\rightarrow$	$\breve{R}_2(g_{\bar{2}})$
$\mathcal{D}_1$	$\mathcal{D}_2$	$\mathcal{D}_3$	$\cdots$	$\mathcal{D}_K$	$\rightarrow$	$\breve{R}_3(g_{\bar{3}})$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$		
$\mathcal{D}_1$	$\mathcal{D}_2$	$\mathcal{D}_3$	$\cdots$	$\mathcal{D}_K$	$\rightarrow$	$\breve{R}_K(g_{\bar{K}})$

$$\breve{R}_{CV} = \frac{1}{K}\sum_{k=1}^{K} \breve{R}_k(g_{\bar{k}})$$

Regularized Loss Minimization (RLM)

- Hypothesis class $\mathcal{H} = \cup_i (\mathcal{H}_i, \lambda_i)$, with $i \in \mathbb{N}$ e.g. $\lambda_i \in \{0.0001, 0.001, \ldots\}$
- Augmented error:

$$\mathcal{L}_{\bar{k}}(\mathbf{w}, \lambda) = \widehat{R}_{\bar{k}}(\mathbf{w}) + \lambda\Omega(\mathbf{w})$$

$$\text{e.g. } \Omega(\mathbf{w}) \in \{\|\mathbf{w}\|_1, \|\mathbf{w}\|_2^2, \|\mathbf{w}\|_Q^2, \ldots\}$$

- RLM solution:
  - for all i, train on $\mathcal{D}_{\bar{k}}$: $g(\mathbf{w}_{\lambda_i}) = \text{argmin}_{\mathbf{w}} \mathcal{L}_{\bar{k}}(\mathbf{w}, \lambda_i)$
  - from all i, select on $\mathcal{D}_k$: $g(\mathbf{w}_{\lambda *}) = \text{argmin}_{\lambda_i} \breve{R}_k(g(\mathbf{w}_{\lambda_i}))$

Part 3 summary

- Non-linear feature transform: polynomial, Legendre
- Overfitting/underfitting: match the hypothesis class to data
- Structural risk minimization
- Regularisation: L_2, L_1 ...
- Validation