Nonlinear ridge regression
Risk, regularization, and cross-validation

Nando de Freitas

UNIVERSITY OF OXFORD

Outline of the lecture

This lecture will teach you how to fit nonlinear functions by using bases functions and how to control model complexity. The goal is for you to:

- ❑ Learn how to derive **ridge regression**.
- ❑ Understand the trade-off of fitting the data and **regularizing** it.
- ❑ Learn **polynomial regression**.
- ❑ Understand that, if basis functions are given, the problem of learning the parameters is still linear.
- ❑ Learn **cross-validation.**
- ❑ Understand model complexity and **generalization**.

Regularization

$d \times d$

All the answers so far are of the form

$n \times d$

$$\widehat{\boldsymbol{\theta}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

They require the inversion of $\mathbf{X}^T\mathbf{X}$. This can lead to problems if the system of equations is poorly conditioned. A solution is to add a small element to the diagonal:

$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$

$$\widehat{\boldsymbol{\theta}} = (\mathbf{X}^T\mathbf{X} + \delta^2 I_d)^{-1}\mathbf{X}^T\mathbf{y}$$

This is the ridge regression estimate. It is the solution to the following **regularised quadratic cost function**

$$J(\boldsymbol{\theta}) = (\mathbf{y} - \mathbf{X}\boldsymbol{\theta})^T(\mathbf{y} - \mathbf{X}\boldsymbol{\theta}) + \delta^2\boldsymbol{\theta}^T\boldsymbol{\theta}$$

Derivation

$$J(\theta) = (y - X\theta)^T (y - X\theta) + \delta^2 \theta^T \theta$$

$$\frac{\partial J(\theta)}{\partial \theta} = \frac{\partial}{\partial \theta}\left(\theta^T X^T X \theta - 2 y^T X \theta + y^T y + \delta^2 \theta^T \theta \right)$$

$$= 2 X^T X \theta - 2 X^T y + 2 \delta^2 I \theta$$

$$= 2 (X^T X + \delta^2 I) \theta - 2 X^T y$$

Equate to zero

$$\hat{\theta}_{\text{ridge}} = (X^T X + \delta^2 I)^{-1} X^T y$$

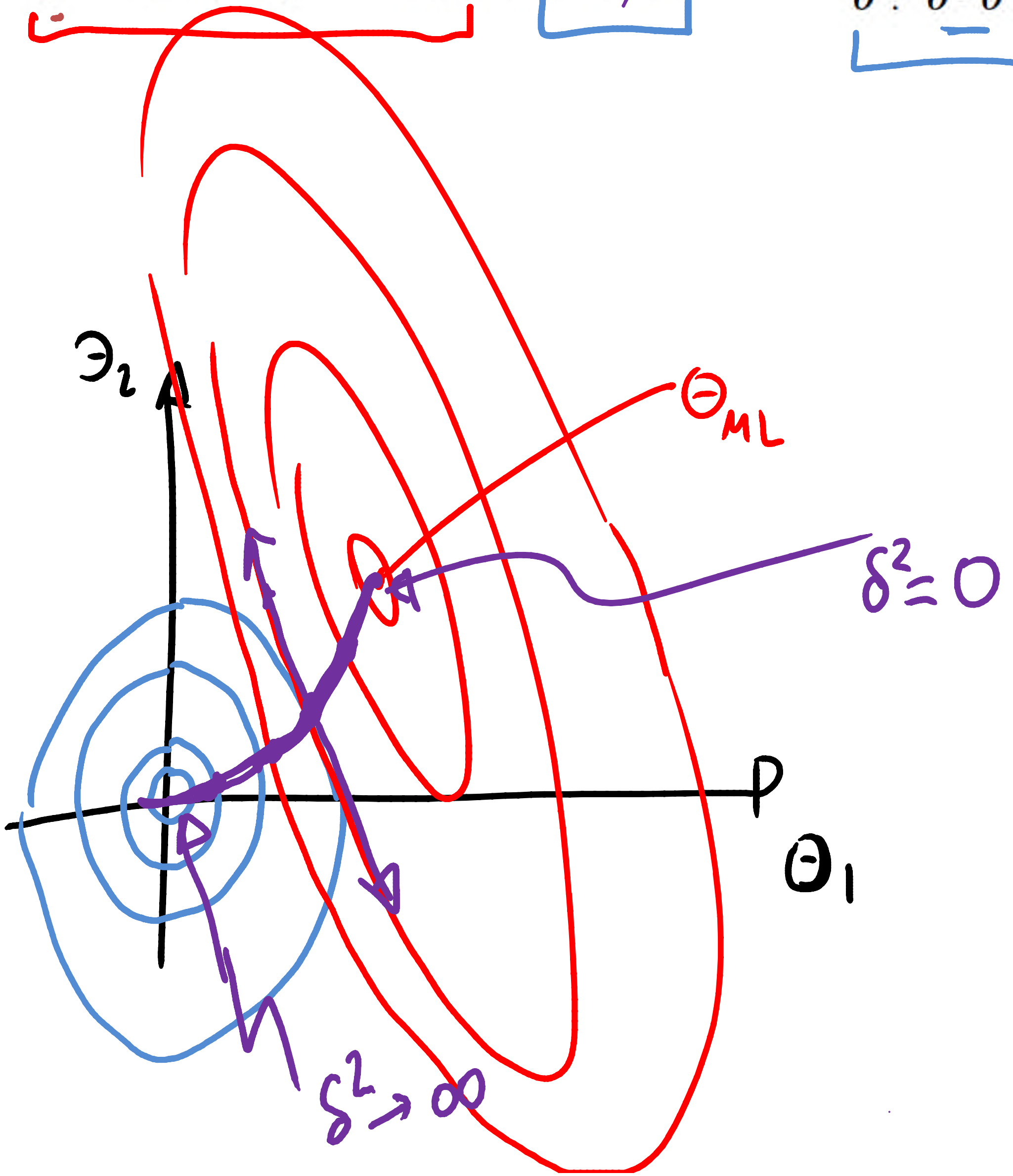
Ridge regression as constrained optimization

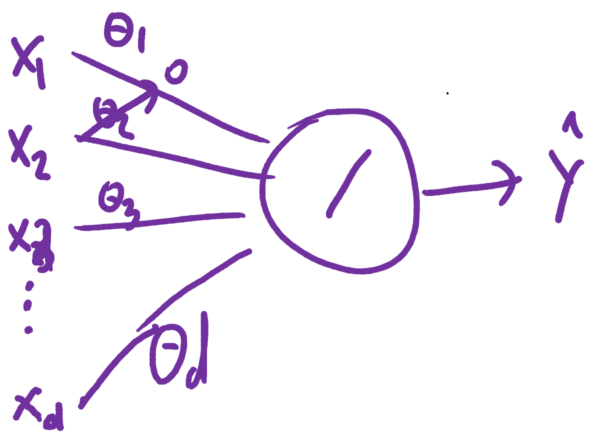
$$J(\boldsymbol{\theta}) = \underbrace{(\mathbf{y} - \mathbf{X}\boldsymbol{\theta})^T(\mathbf{y} - \mathbf{X}\boldsymbol{\theta})} + \underbrace{\delta^2\boldsymbol{\theta}^T\boldsymbol{\theta}} \equiv \underbrace{\min_{\boldsymbol{\theta}\,:\,\boldsymbol{\theta}^T\boldsymbol{\theta} \le t(\delta)}} \underbrace{\{(\mathbf{y} - \mathbf{X}\boldsymbol{\theta})^T(\mathbf{y} - \mathbf{X}\boldsymbol{\theta})\}}$$

$$\theta^T = [\theta_1 \quad \theta_2]$$

$$\theta^T\theta = [\theta_1 \; \theta_2]\begin{bmatrix}\theta_1 \\ \theta_2\end{bmatrix}$$

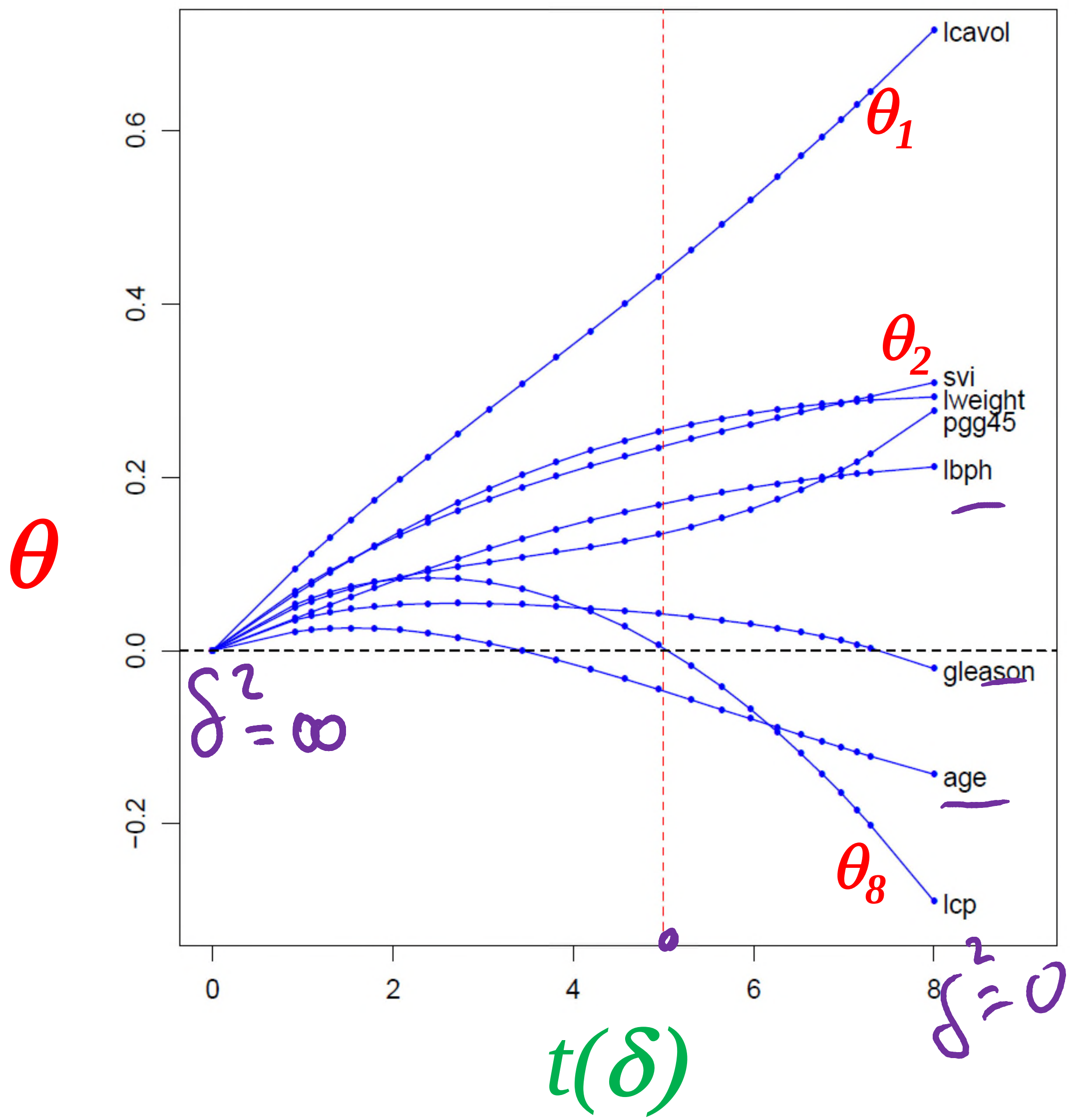
$$= \theta_1^2 + \theta_2^2 = \text{const}$$

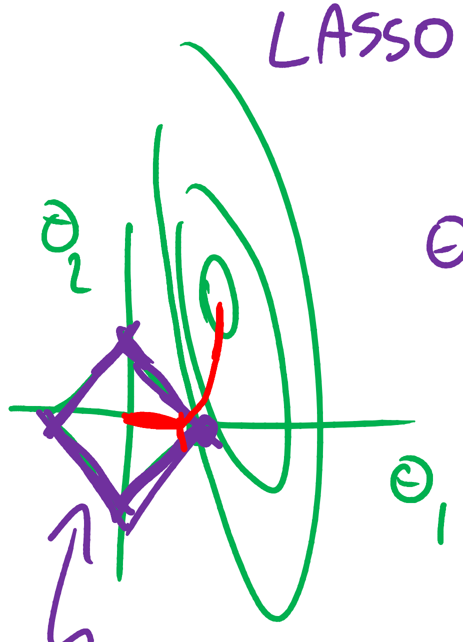




Regularization paths

As δ increases, $t(\delta)$ decreases and each θ_i goes to zero.





$\theta^T\theta = \|\theta\|_2^2$

$|\theta_1| + |\theta_2|$

$\|\theta\|_1 = \sum |\theta_i|$

[Hastie, Tibshirani & Friedman book]

Ridge regression and Maximum a Posteriori (MAP) learning

Bayes Rule

$$J(\boldsymbol{\theta}) = \underbrace{(\mathbf{y} - \mathbf{X}\boldsymbol{\theta})^T(\mathbf{y} - \mathbf{X}\boldsymbol{\theta})}_{E(\theta, X, y)} + \underbrace{\delta^2\boldsymbol{\theta}^T\boldsymbol{\theta}}$$

$$P(y|X,\theta) = \frac{1}{Z} e^{-E(\theta, X, y)}$$

$$P(y|X) = \int P(y|X,\theta)P(\theta)\, d\theta$$

Prior $\rightarrow$ $P(\theta) = \frac{1}{Z_2} e^{-\delta^2\theta^T\theta}$

$$\max_\theta \text{const}\, P(y|X,\theta)P(\theta) = \max_\theta \frac{P(y|X,\theta)P(\theta)}{P(y|X)}$$

Ridge regression and Maximum a Posteriori (MAP) learning

$$P(A|B) = \frac{P(B|A)\,P(A)}{P(B)}$$

$$J(\boldsymbol{\theta}) = \underbrace{(\mathbf{y}-\mathbf{X}\boldsymbol{\theta})^T(\mathbf{y}-\mathbf{X}\boldsymbol{\theta})}_{\text{likelihood}} + \underbrace{\delta^2\boldsymbol{\theta}^T\boldsymbol{\theta}}_{\text{prior}}$$

Posterior

$$P(\theta|x,y) = \frac{P(y|x,\theta)\,P(\theta)}{P(y|x)}$$

word

$$= \frac{P(y|x,\theta)\,P(\theta)}{\int P(y|\theta,x)\,P(\theta)\,d\theta}$$

Going nonlinear via basis functions

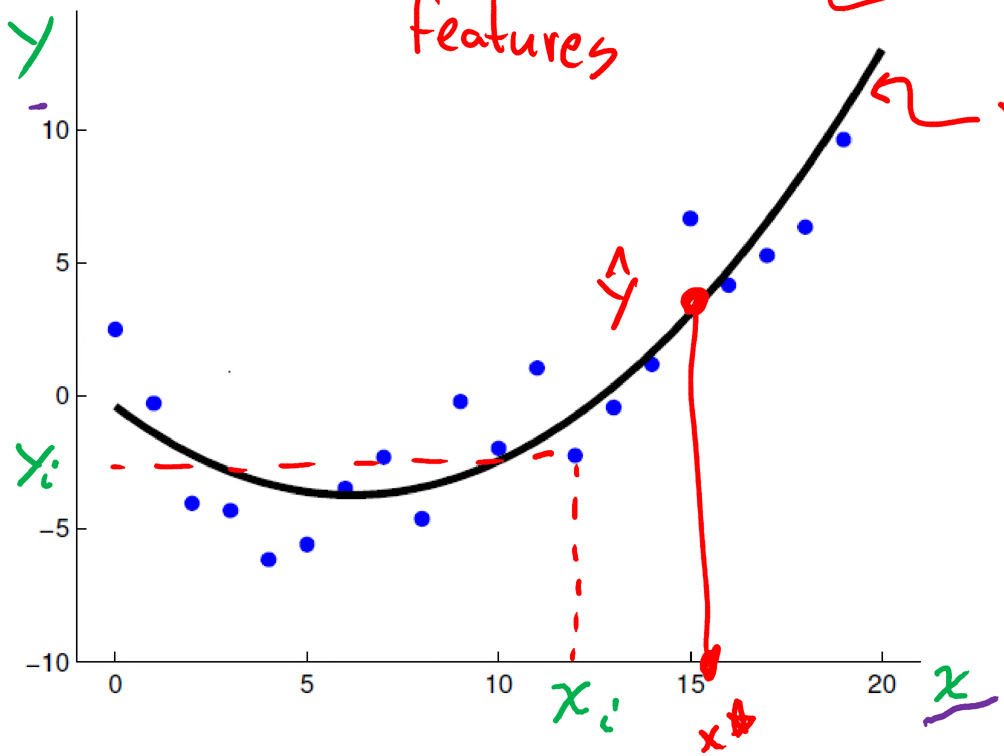
We introduce basis functions $\phi(\cdot)$ to deal with nonlinearity:

$$y(\mathbf{x}) = \phi(\mathbf{x})\boldsymbol{\theta} + \epsilon$$

For example, $\phi(x) = \underbrace{[1, x_i, x_i^2]}_{\text{features}}$

$$\hat{\Theta} = (\phi^T\phi + \delta^2 I)^{-1}\phi^T y$$

x	y
2.2	1
6.3	2.6
⋮	⋮

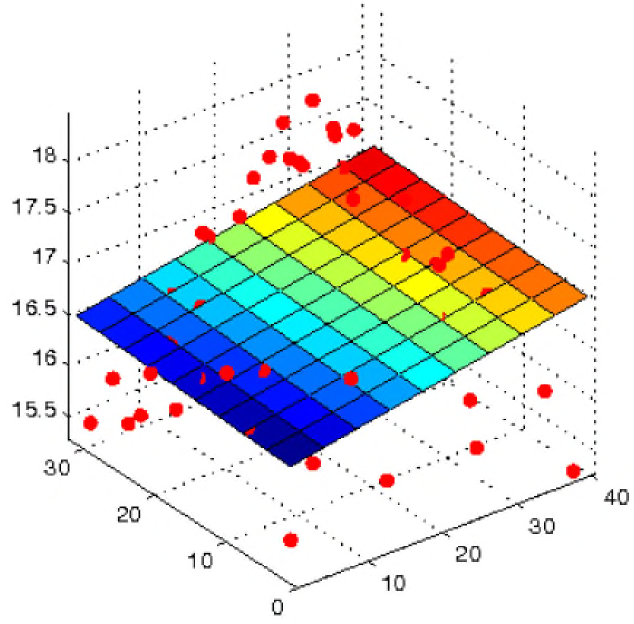


$$\hat{y} = \phi(x)\Theta = \Theta_0 + x\Theta_1 + x^2\Theta_2 = [1 \; x \; x^2]\begin{bmatrix}\Theta_0 \\ \Theta_1 \\ \Theta_2\end{bmatrix}$$

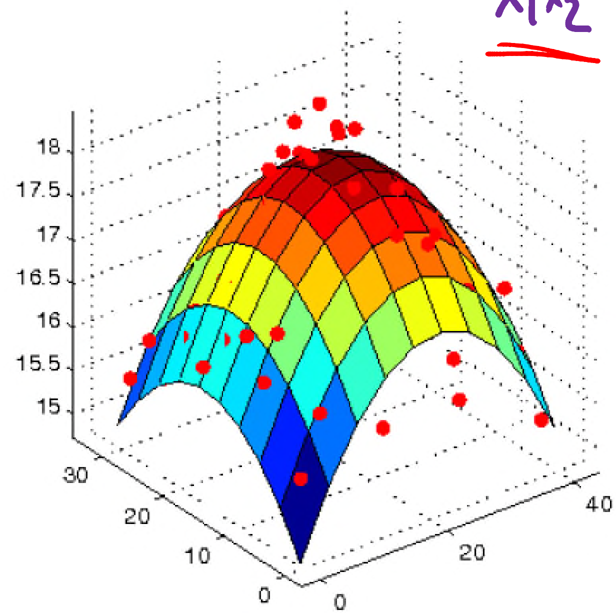
Going nonlinear via basis functions

$$y(\mathbf{x}) = \boldsymbol{\phi}(\mathbf{x})\boldsymbol{\theta} + \epsilon$$

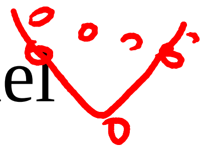
$$\boldsymbol{\phi}(\mathbf{x}) = [1, x_1, x_2] \qquad \boldsymbol{\phi}(\mathbf{x}) = [1, x_1, x_2, x_1^2, x_2^2]$$



$\underline{x_1 x_2}$

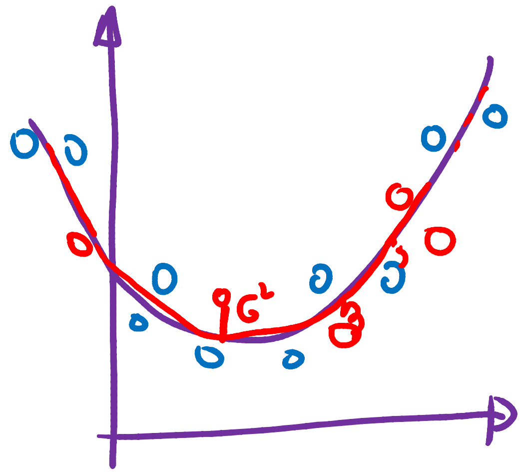


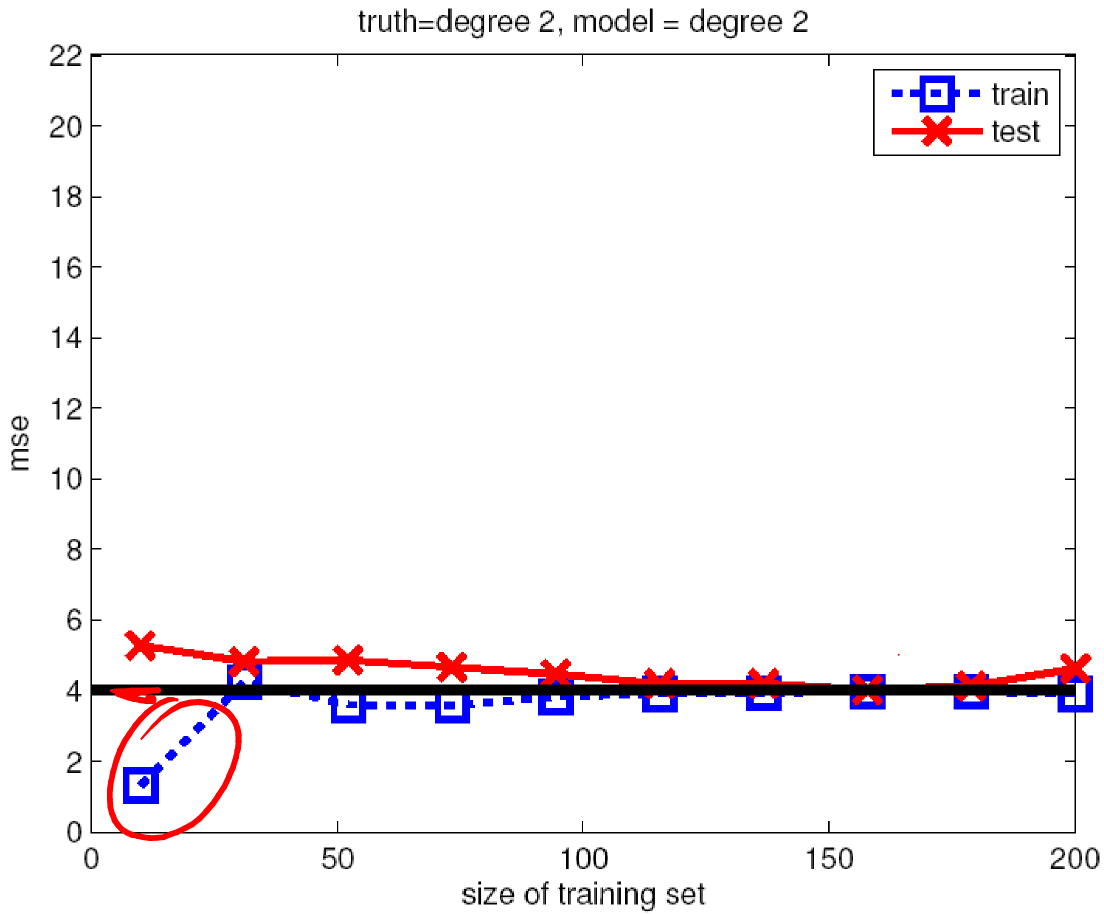
Effect of data when we have the right model



$$y_i = \underline{\theta_0 + x_i\,\theta_1 + x_i^2\,\theta_2} + \underline{\mathcal{N}(0, \sigma^2)}$$

$$\hat{y} = \theta_0 + x_i\theta_1 + x_i^2\theta_2$$





o Train data

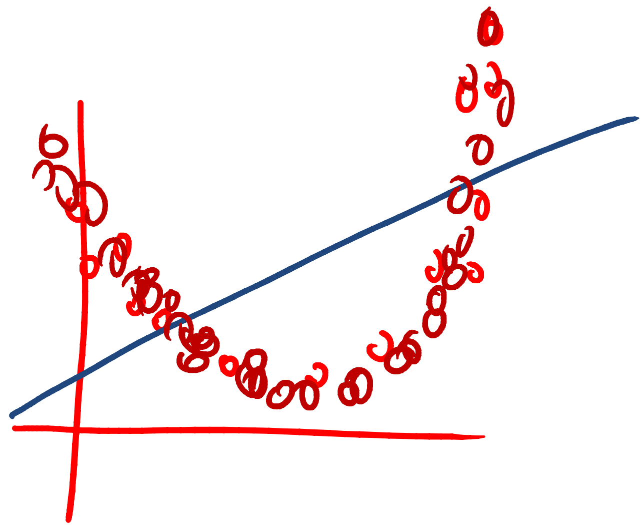
• Test data

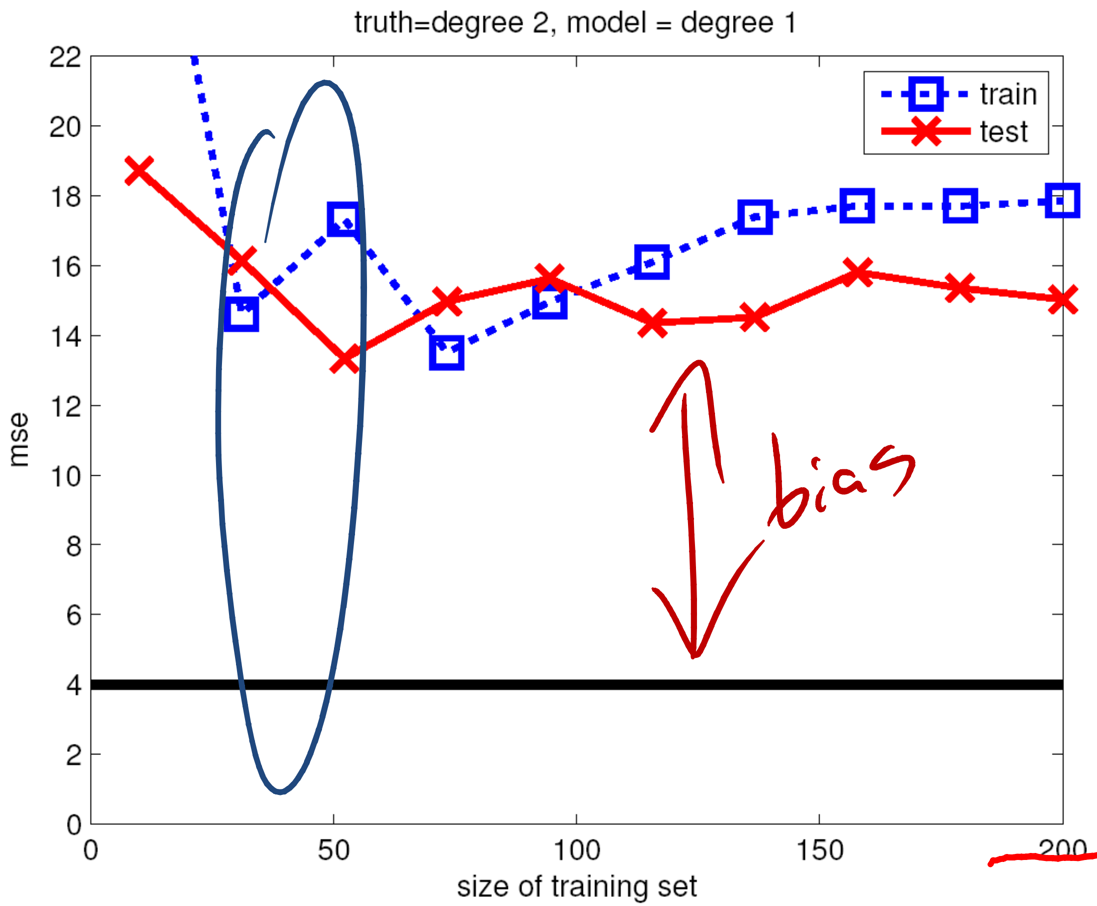
noise Error

Effect of data when the model is too simple

$$y_i = \theta_0 + x_i\,\theta_1 + x_i^2\,\theta_2 + \mathcal{N}(0, \sigma^2)$$

$$\hat{y} = \theta_0 + x_i\,\theta_1$$





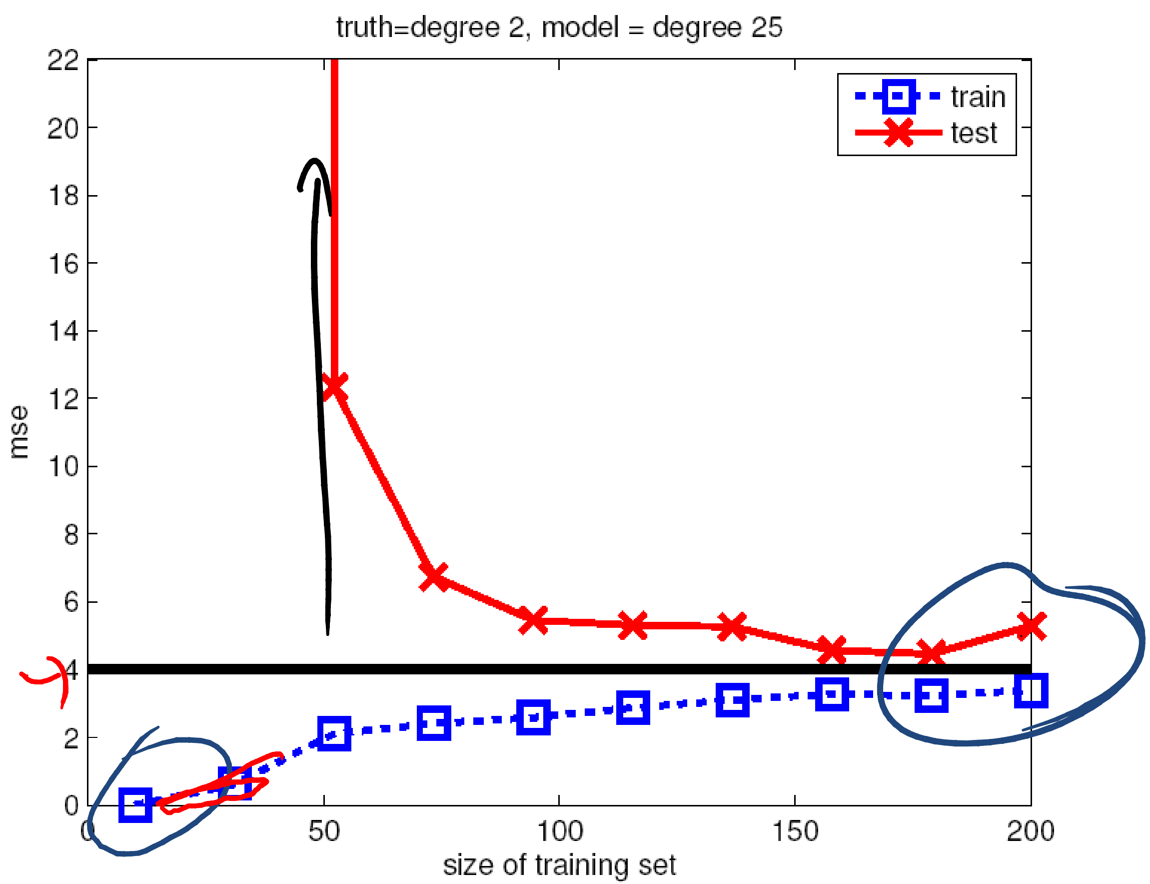
$\rightarrow i = 1 : N \rightarrow$

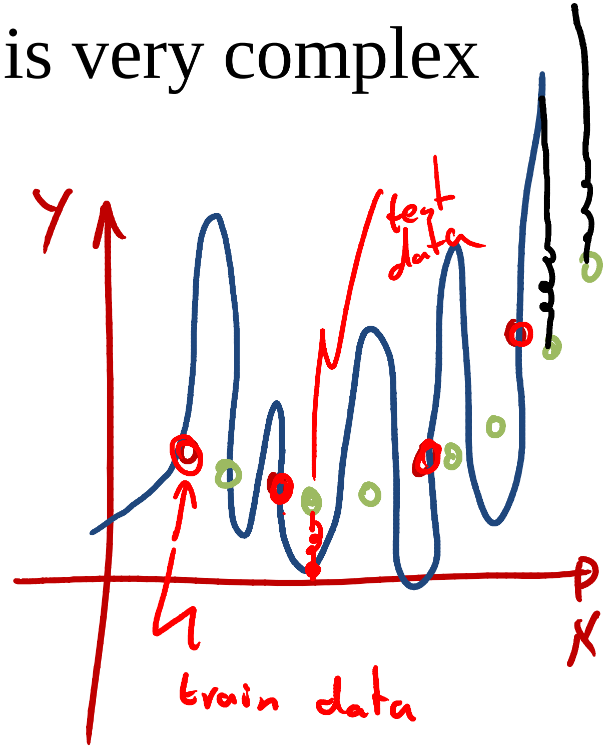
$N \rightarrow$ increases

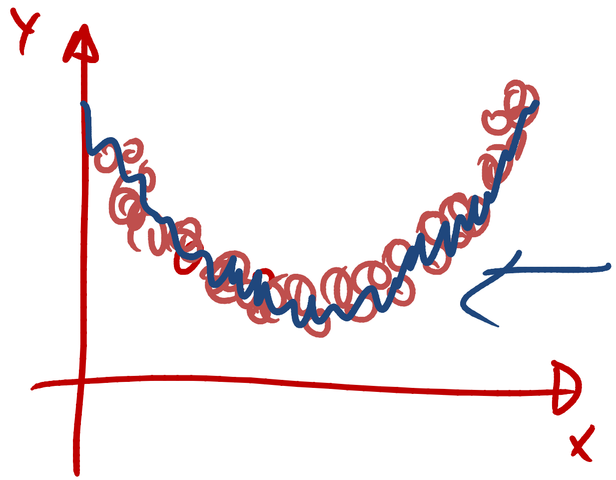
Effect of data when the model is very complex

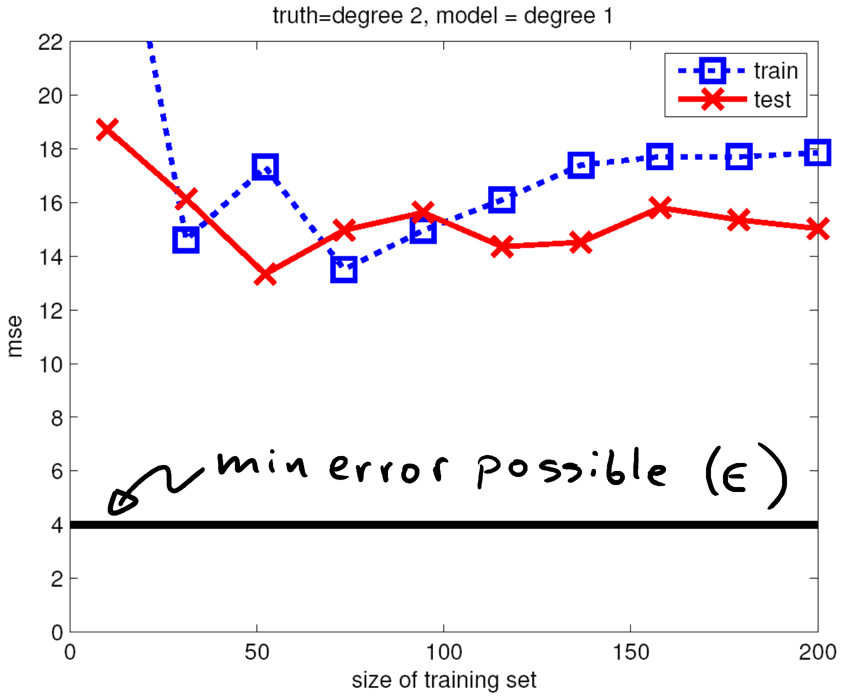
$$y_i = \theta_0 + x_i\,\theta_1 + x_i^2\,\theta_2 + \mathcal{N}(0, \sigma^2)$$

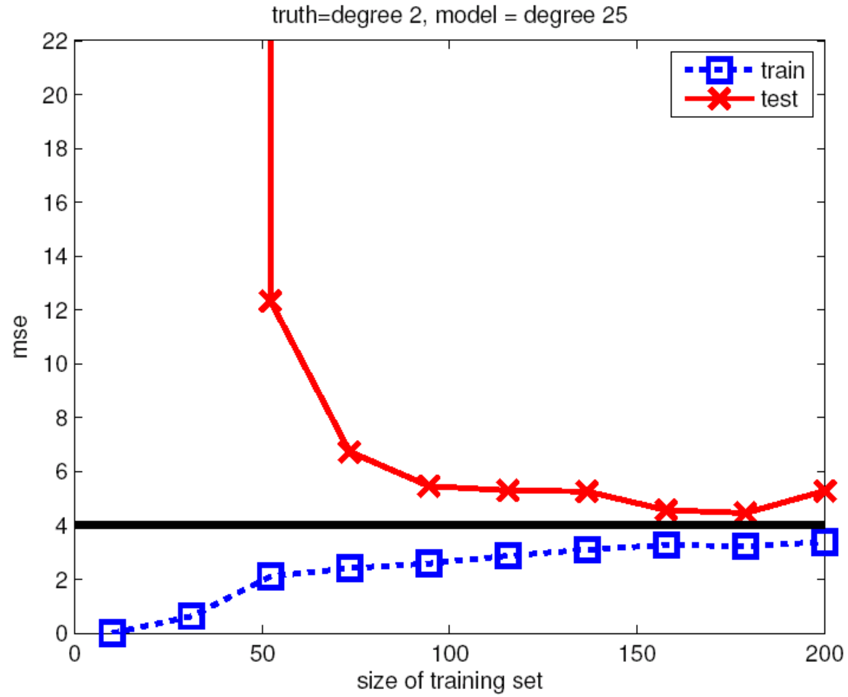
$$\hat{Y}_i = \Theta_0 + x_i\Theta_1 + \cdots + x_i^{25}\Theta_{25}$$

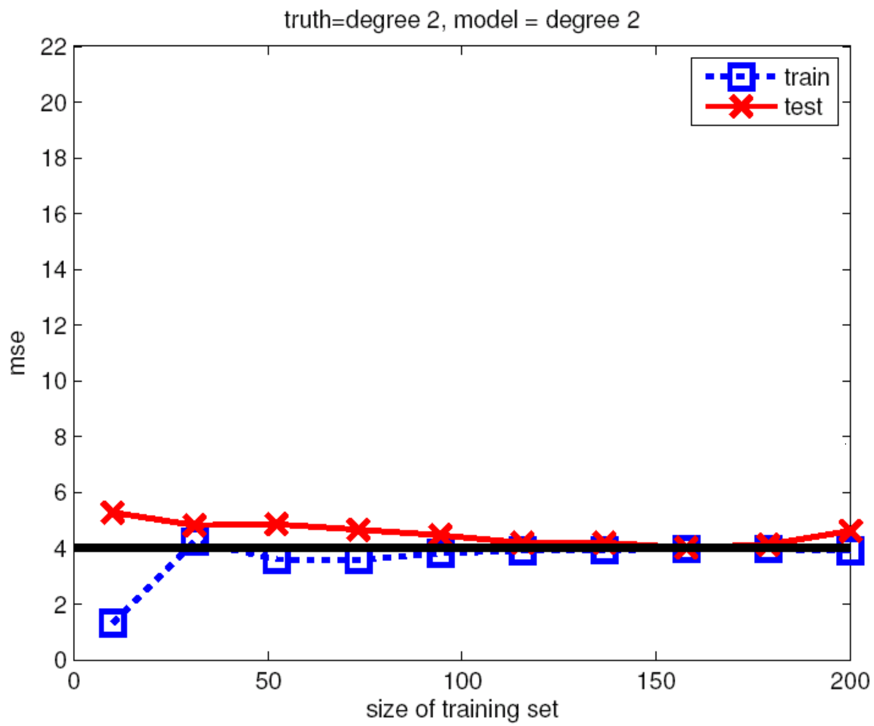












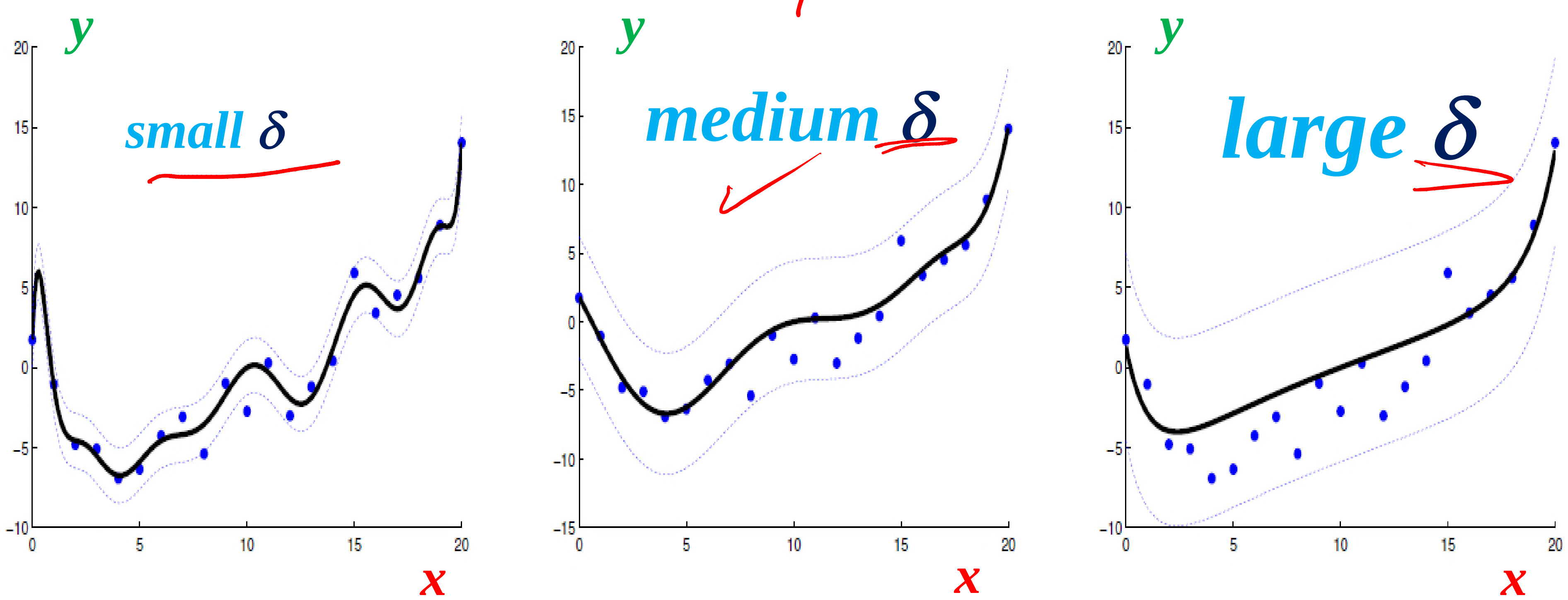
More data improves results, but only if the model has the right complexity.

Example: Ridge regression with a polynomial of degree 14

$$\hat{y}(x_i) = 1\,\theta_0 + x_i\,\theta_1 + x_i^2\,\theta_2 + \dots + x_i^{13}\,\theta_{13} + x_i^{14}\,\theta_{14}$$

$$\Phi_i = [\,1 \quad x_i \quad x_i^2 \quad \dots \quad x_i^{13} \quad x_i^{14}\,]$$

$$J(\theta) = (y - \Phi\,\theta)^T (y - \Phi\,\theta) + \delta^2\,\theta^T\theta$$



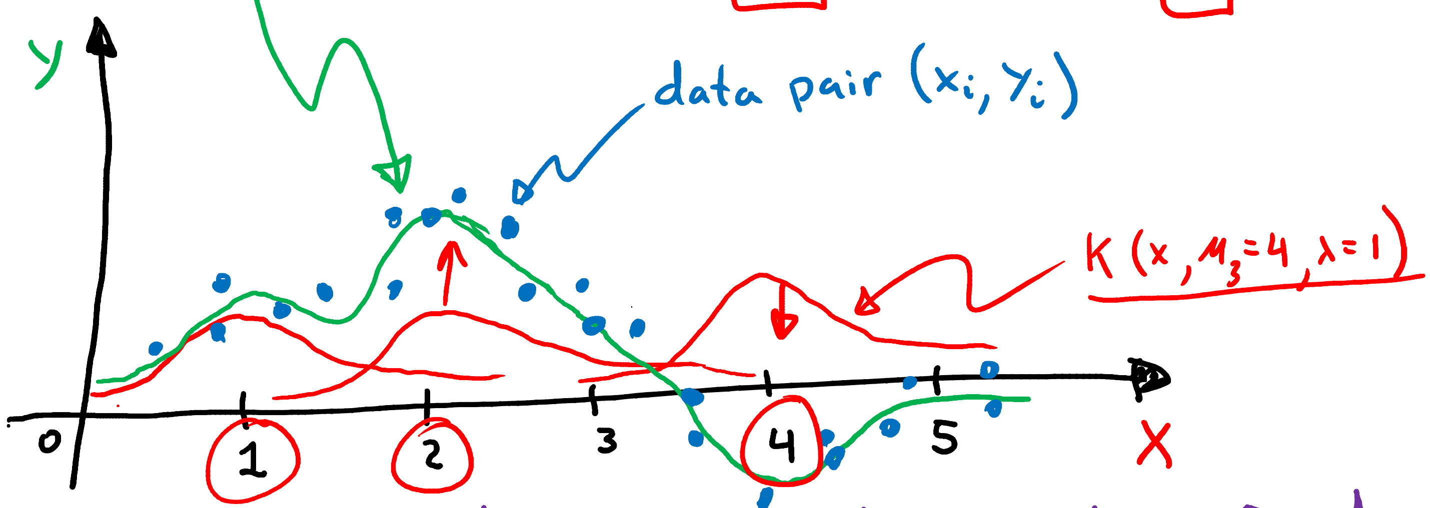
Kernel regression and RBFs

We can use kernels or radial basis functions (RBFs) as features:

$$\phi(\mathbf{x}_i) = [\kappa(\mathbf{x}_i, \boldsymbol{\mu}_1, \lambda), \ldots, \kappa(\mathbf{x}_i, \boldsymbol{\mu}_d, \lambda)], \quad e.g. \ \kappa(\mathbf{x}, \boldsymbol{\mu}_i, \lambda) = e^{(-\frac{1}{\lambda}\|\mathbf{x}-\boldsymbol{\mu}_i\|^2)}$$

$$\hat{y}(x_i) = \phi(x_i)\,\theta = 1\theta_0 + k(x_i, \mu_1, \lambda)\,\theta_1 + \ldots + k(x_i, \mu_d, \lambda)\,\theta_d$$

Example 1: $\hat{Y}(x) = e^{-\|x-1\|^2}\Theta_1 + e^{-\|x-2\|^2}\Theta_2 + e^{-\|x-4\|^2}\Theta_3$



The green curve is a weighted sum of the 3 red curves

$$\phi(x_i) = [1 \quad \kappa(x_i, \mu_1, \lambda) \quad \kappa(x_i, \mu_2, \lambda) \quad \kappa(x_i, \mu_3, \lambda)]$$

$\phi(x_i)$ is a vector with 4 entries. There are 3 bases.

The corresponding vector of parameters is $\underline{\theta} = \underbrace{[\theta_0 \quad \theta_1 \quad \theta_2 \quad \theta_3]^T}$

$$\boxed{\hat{y}_i = \phi(x_i)\,\underline{\theta}}$$

If we have $i = 1, \ldots, N$ data, let

$$\underset{N \times 1}{Y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} \qquad \underset{N \times 4}{\Phi} = \begin{bmatrix} \phi(x_1) \\ \phi(x_2) \\ \vdots \\ \phi(x_N) \end{bmatrix}$$

Then

$$\hat{Y} = \Phi \Theta$$

and

$$\hat{\Theta}_{LS} = \left(\Phi^T \Phi\right)^{-1} \Phi^T Y$$

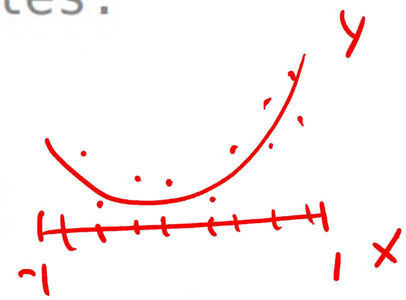
or

$$\hat{\Theta}_{ridge} = \left(\Phi^T \Phi + \delta^2 I\right)^{-1} \Phi^T Y$$

Hence, this is still linear regression, with X replaced by Φ.

Kernel regression in Torch

```lua
require 'torch'
require 'gnuplot'

local nData   = 10 -- Number of data samples.
local kWidth = 1 -- Kernel width.

local xTrain = torch.linspace(-1,1,nData)
local yTrain = torch.pow(xTrain,2)
local yTrain = yTrain + torch.mul(torch.randn(nData),0.1)

local function phi(x, y)
  return torch.exp(-(1/kWidth)*torch.sum(torch.pow(x-y,2)))
end
```



Kernel regression in Torch

$$\phi(x_i, x_j) = e^{-\frac{1}{\lambda}\|x_i - x_j\|^2}$$

$$\Phi = \begin{bmatrix} \phi_{11} & \cdots & \phi_{1n} \\ & & \\ & & \phi_{nn} \end{bmatrix}$$

```
local Phi = torch.Tensor(nData,nData)
for i=1,nData do
   for j=1,nData do
      Phi[i][j]=phi(xTrain[{{i}}],xTrain[{{j}}])
   end
end
```

$\sim \delta^2$

```
local regularizer = torch.mul(torch.eye(nData),0.001)
local theta = torch.inverse((Phi:t()*Phi) + regularizer) * Phi:t() * yTrain
```

$$\theta = \left[\Phi^T \Phi + \delta^2 I\right]^{-1} \Phi^T y$$

Kernel regression in Torch

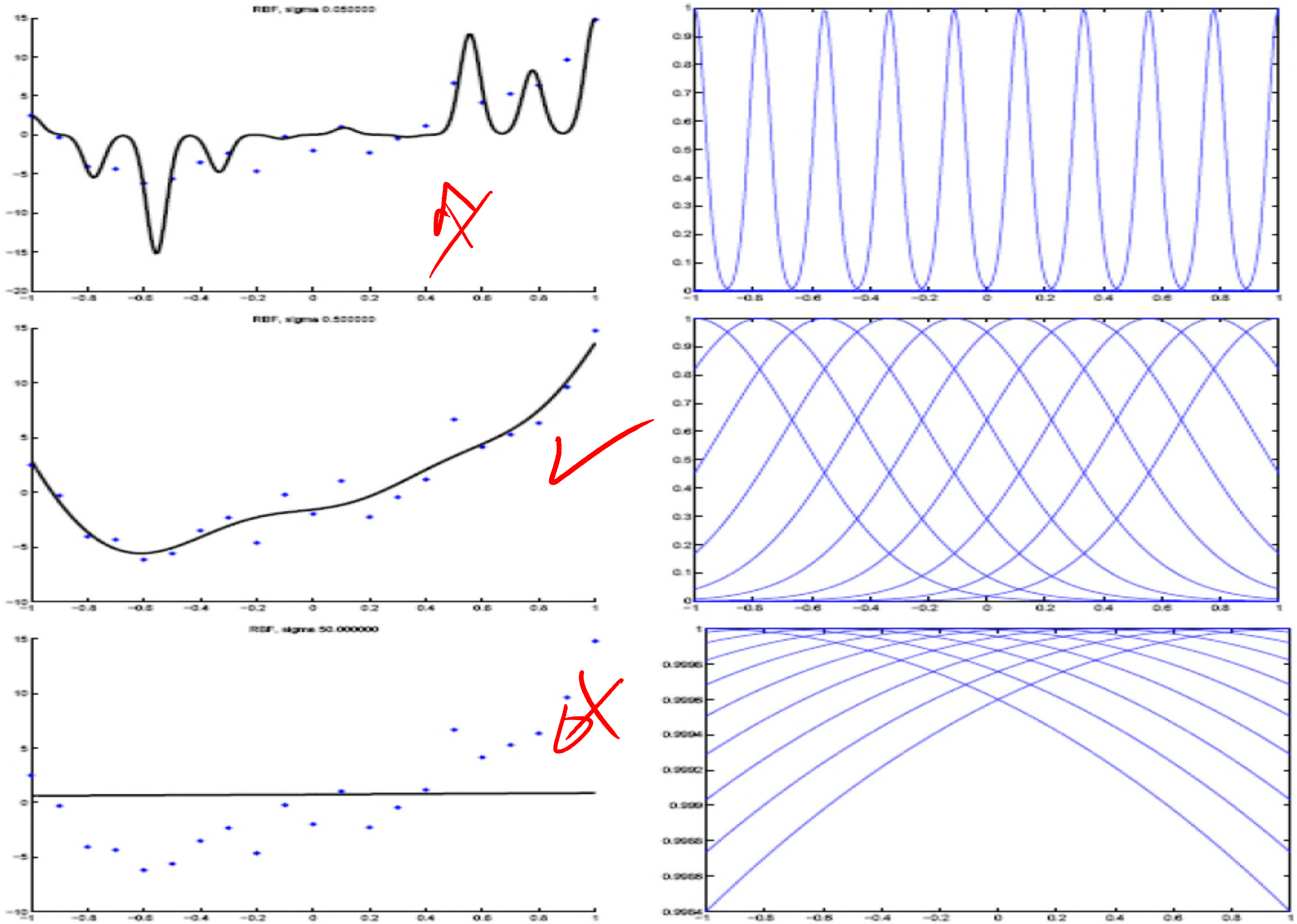
```
local nTestData  = 100  -- Number of test data samples
local xTest = torch.linspace(-1,1,nTestData)

local PhiTest = torch.Tensor(nData,nTestData)
for i=1,nData do
  for j=1,nTestData do
    PhiTest[i][j]=phi(xTrain[{{i}}],xTest[{{j}}])
  end
end
```

```
local yPred = PhiTest:t() * theta
```

```
gnuplot.plot({'Data',xTrain,yTrain,'+'},{'Prediction',xTest,yPred,'-'})
```

We can choose the locations μ of the ***basis functions*** *to be the inputs.*
That is, $\mu_i = x_i$. These basis functions are the known as ***kernels****.*
The choice of width λ is tricky, as illustrated below.

kernels



Too small λ

Right λ

Too large λ

The big question is how do we choose the regularization coefficient, the width of the kernels or the polynomial order?

Simple solution: **cross-validation**

1. Given training data $(X, Y)_{train}$, and some δ^2 guess, compute $\hat{\Theta}$
2. $\hat{Y}_{train} = X_{train}\hat{\Theta}$ (compute training set predictions)
3. $\hat{Y}_{test} = X_{test}\hat{\Theta}$

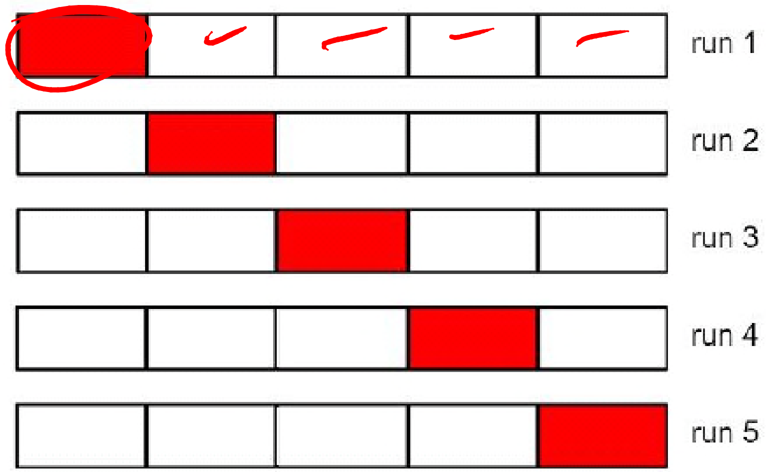
train | test

val

validation

q_1, q_2, q_3, λ, δ^2	Train error $\sum_{i \in train}(y_i - \hat{y}_i)^2$	Test error $\sum_{i \in test}(y_i - \hat{y}_i)^2$	max	min_max	avg
1 → 0.1	100	2	100		
3 | → 1	10	11	11	11	10.5
→ | → 10	1	19	19		x 10 ✗
11 → 50	20	0	20		x 10
1 → 100	100	1000	1000		

K-fold crossvalidation

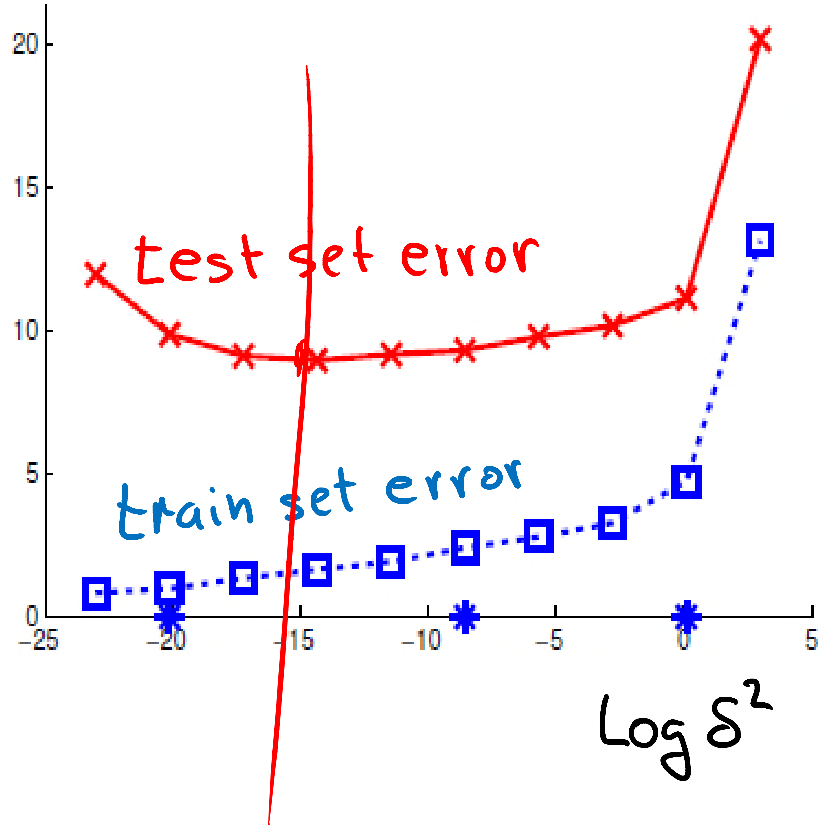


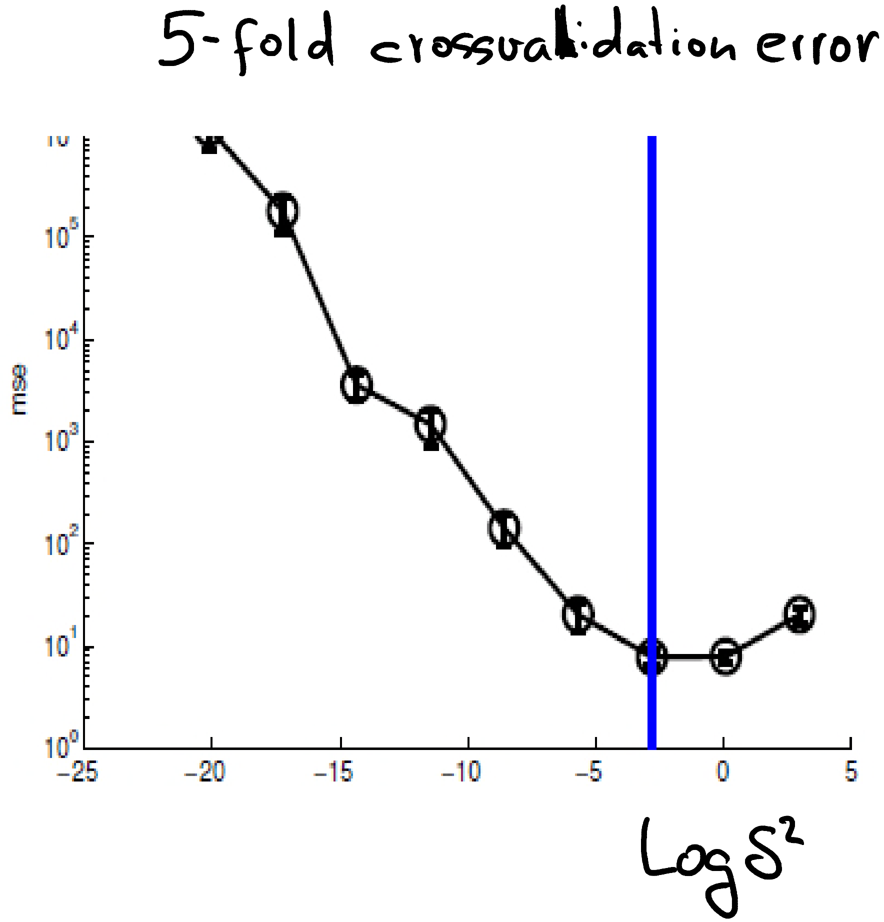
The idea is simple: we split the training data into K **folds**; then, for each fold $k \in \{1, \ldots, K\}$, we train on all the folds but the k'th, and test on the k'th, in a round-robin fashion.

It is common to use $K = 5$; this is called 5-fold CV.

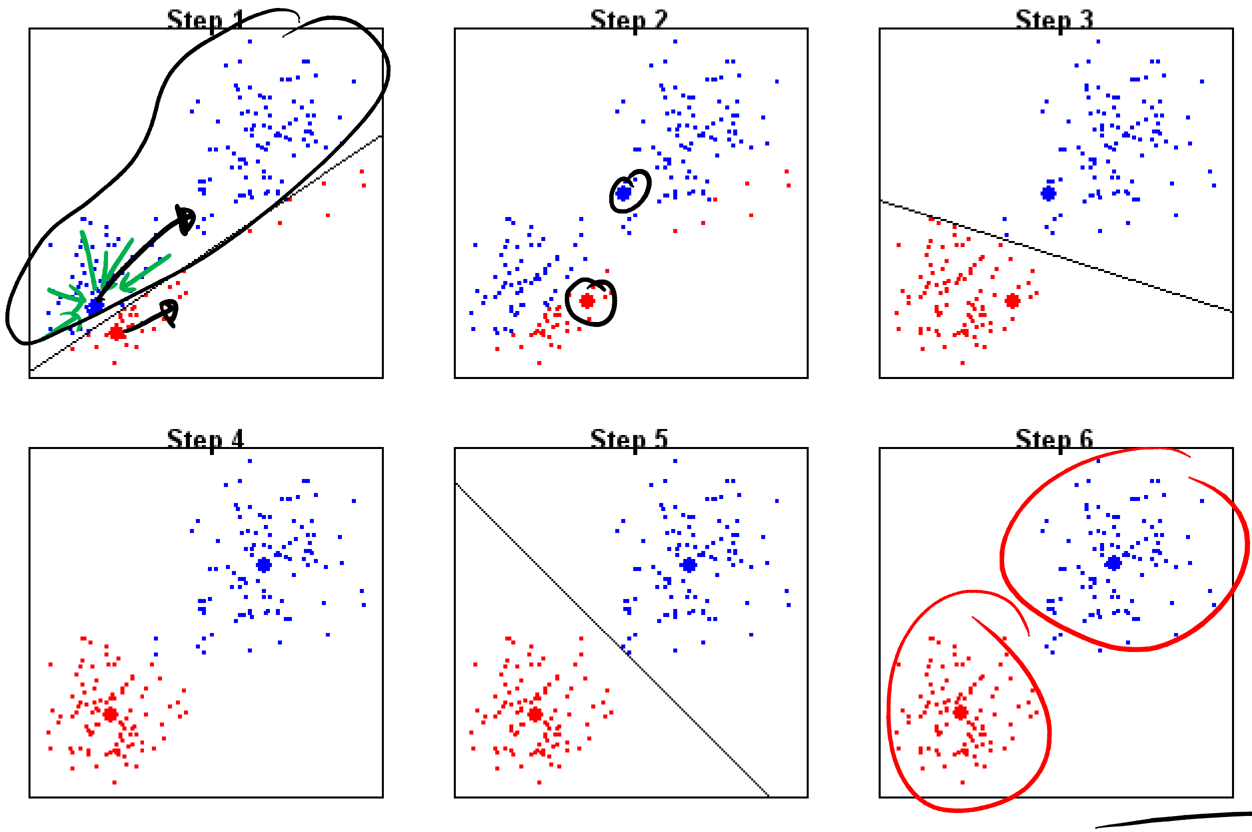
If we set $K = N$, then we get a method called **leave-one out cross validation**, or **LOOCV**, since in fold i, we train on all the data cases except for i, and then test on i.

Example: Ridge regression with polynomial of degree 14





Where cross-validation fails) (K-means)



$$E(\mathcal{D}, L) = \frac{1}{|\mathcal{D}|} \sum_{i \in \mathcal{D}} ||\mathbf{x}_i - \hat{\mathbf{x}}_i||^2$$

$$\boxed{\hat{\mathbf{x}}_i = \boldsymbol{\mu}_{z_i}}$$

0

$$z_i = \operatorname{argmin}_k ||\mathbf{x}_i - \boldsymbol{\mu}_k||_2^2$$

Next lecture

In the next lecture, we delve into the world of optimization.

Please revise your multivariable calculus and in particular the definition of **gradient**