# CS5014 Machine Learning

## Lecture 10 Regularisation

Lei Fang

School of Computer Science, University of St Andrews

Spring 2021

University
of
St Andrews

# Some responses

What one hot encoding actually does ?

- regression ?
- classification ?

Why Newton's step use Hessian's inverse $\boldsymbol{H}^{-1}$ ?

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \boldsymbol{H}_t^{-1}\boldsymbol{g}_t$$

- long story: it optimise local approximated quadratic function
- short story: inversely related to curvature (H measures curvature in higher dimensions)
  - larger curvature (pointy and curvy) $\Rightarrow$ shorter steps;
  - smaller curvature (flatter)

# Today's topic

Nonlinear model

- fixed basis models

Regularisation

- $l_2$ penalty: ridge regression
- $l_1$ penalty: lasso regression

# Towards nonlinear models: regression

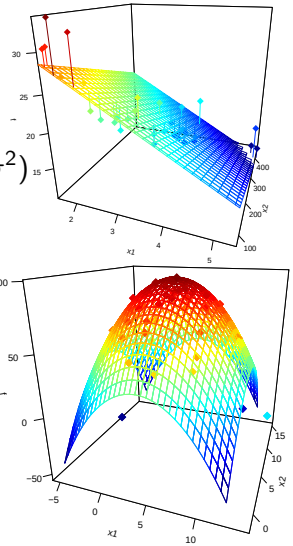
For linear regression:

$$P(y|\boldsymbol{x},\boldsymbol{\theta}) = N(\underbrace{f(\boldsymbol{x};\boldsymbol{\theta})}_{\boldsymbol{\theta}^T\boldsymbol{x}\text{: linear}}, \sigma^2) \text{ or } y = \underbrace{f(\boldsymbol{x};\boldsymbol{\theta})}_{\boldsymbol{\theta}^T\boldsymbol{x}} + \epsilon, \quad \epsilon^{(i)} \sim N(0, \sigma^2)$$

The regression function $f$ is assumed linear

$$f(\boldsymbol{x};\boldsymbol{\theta}) = \boldsymbol{\theta}^T\boldsymbol{x}$$

- *i.e.* fitting lines/hyperplanes
- in real life, a lot of relationships are not linear
- and we do not know what $f(\boldsymbol{x})$ should look like !



# Towards nonlinear models: classification

For logistic regression:

$$P(y|\boldsymbol{x},\boldsymbol{\theta}) = \text{Ber}(\sigma(\underbrace{f(\boldsymbol{x};\boldsymbol{\theta})}_{\boldsymbol{\theta}^T\boldsymbol{x}\text{: linear}}) = \sigma^y(1-\sigma)^{(1-y)}$$

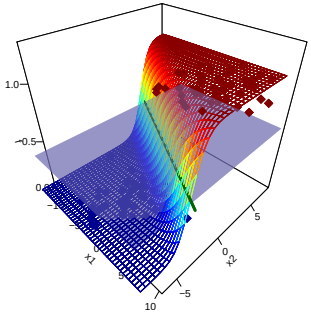
To predict the label $y$ for any input $\boldsymbol{x}$:

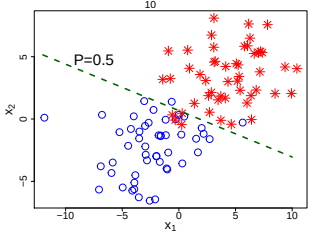
$$y = 1 \text{ if } P(y=1|\boldsymbol{x},\boldsymbol{\theta}) > 0.5 \quad y = 0 \text{ if otherwise}$$

Note that the **decision boundary** is linear (hyperplane or line)

$$P(y=1|\boldsymbol{x},\boldsymbol{\theta}) = 0.5 \Rightarrow \boldsymbol{\theta}^T\boldsymbol{x} = 0$$

- *i.e.* separating data by lines/hyperplanes
- in reality, we do know what $f$ should be; plane or a more general surface

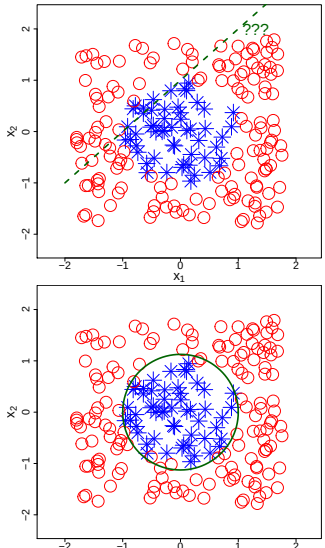


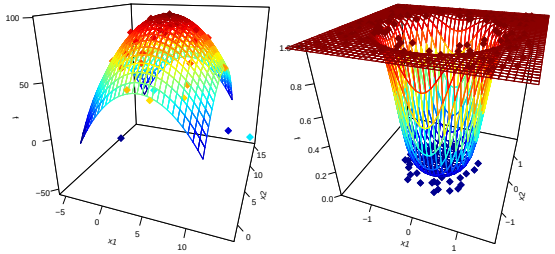
# Nonlinear classification data

What if your data looks like this ?

- no linear descision boudary or $f(\boldsymbol{x};\boldsymbol{\theta}) = \boldsymbol{\theta}^T\boldsymbol{x}$ seems making much sense
- but a non-linear boundary makes more sense
  - the classification rule is actually $||\boldsymbol{x}||_2^2 = x_1^2 + x_2^2 \leq 1$
  - distance to $\mathbf{0}$ is less than 1
  - the boundary is a circle
  - I know it because I generated the data



# Nonlinear model: polynomial model




Both models are actually 2nd order polynomial:

$$f(\boldsymbol{x};\boldsymbol{\beta}) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_1 x_2 + \beta_4 x_1^2 + \beta_5 x_2^2 = \sum_{k=0}^{5} \beta_j \phi_j(\boldsymbol{x}) = \boldsymbol{\beta}^T \boldsymbol{\phi}(\boldsymbol{x})$$

- where

$$\boldsymbol{\phi}(\boldsymbol{x}) = [\underbrace{1}_{\phi_0(\boldsymbol{x})}, \underbrace{x_1}_{\phi_1(\boldsymbol{x})}, \underbrace{x_2}_{\phi_2(\boldsymbol{x})}, \underbrace{x_1 x_2}_{\phi_3(\boldsymbol{x})}, \underbrace{x_1^2}_{\phi_4(\boldsymbol{x})}, \underbrace{x_2^2}_{\phi_5(\boldsymbol{x})}]^T$$

- it expands $\boldsymbol{x} = [1, x_1, x_2]^T$ to a larger vector

# Nonlinear response from linear model

Note that you get a free nonlinear model by transforming the input $\boldsymbol{X}$

$$\boldsymbol{X} = \begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} \\ \vdots & \vdots & \vdots \\ 1 & x_1^{(m)} & x_2^{(m)} \end{bmatrix} \Rightarrow \boldsymbol{\Phi} = \begin{bmatrix} \phi(\boldsymbol{x}^{(1)}) \\ \phi(\boldsymbol{x}^{(2)}) \\ \vdots \\ \phi(\boldsymbol{x}^{(m)}) \end{bmatrix} = \begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} & x_1^{(1)}x_2^{(1)} & (x_1^{(1)})^2 & (x_2^{(1)})^2 \\ 1 & x_1^{(2)} & x_2^{(2)} & x_1^{(2)}x_2^{(2)} & (x_1^{(2)})^2 & (x_2^{(2)})^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_1^{(m)} & x_2^{(m)} & x_1^{(m)}x_2^{(m)} & (x_1^{(m)})^2 & (x_2^{(m)})^2 \end{bmatrix}$$

- remember superscript $(i)$ index data samples; and subscript index features
- $\boldsymbol{\Phi}$ is a $m \times 6$ matrix

The expanded model for regression is

$$\boldsymbol{y} = \boldsymbol{\Phi}\beta + \boldsymbol{\epsilon}$$

- still a linear model w.r.t $\phi$, the expanded new features
- all existing results apply: gradient descent, normal equation (replace $\boldsymbol{X}$ with $\boldsymbol{\Phi}$)

$$\beta_{ls} = (\boldsymbol{\Phi}^T\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^T\boldsymbol{y}$$

# Polynomial basis function expansion

Consider predictors with $n = 2$ input features,

$$\boldsymbol{x} = [1, x_1, x_2]^T \text{ (1 is dummy variable)}$$

Second order polynormial, $\phi$ expands $\boldsymbol{x}$ to

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

- what if the input size is $n$ rather than 2? $(1 + n + \binom{n}{2} + n) \in O(n^2)$

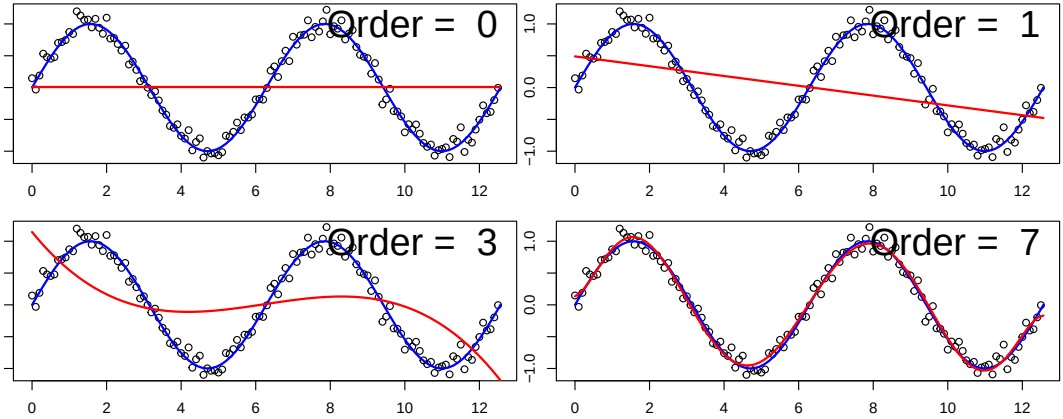
Third order polynormial, it becomes

$$\phi(\boldsymbol{x}) = [1, x_1, x_2, x_1 x_2, x_1^2, x_2^2, \underbrace{x_1 x_2^2, x_1^2 x_2, x_1^3, x_2^3}_{\text{3-rd order terms}}]^T$$

Higher order leads to larger number of basis; it may not necessary be a good thing though

# Example: polynormial fitting

True regression function: $\mathbf{f} = \sin(\mathbf{x})$; and $y^{(i)} = \sin(x^{(i)}) + \epsilon^{(i)}$, $\epsilon \sim N(0, 0.2^2)$



# Other basis function $\phi$

$\phi(\boldsymbol{x})$ can take other forms

- each $\phi_k(\boldsymbol{x})$ is a $R^n \rightarrow R$ transformation;
  - so $\phi$ is a $R^n \rightarrow R^p$ transformation: previous example: $n = 3$, $p = 6$
  - obviously, if $\phi(\boldsymbol{x}) = \boldsymbol{I}\boldsymbol{x}$, we recover ordinary linear regression
    - it is a specific case of basis expansion model
- 2nd order polynomial: $\phi_k(\boldsymbol{x}) = x_j^2$, $x_j x_{j'}$, $x_j$, or 1; for $k = 1 \ldots p$
- $\phi_k(x) = \log(x), \sqrt{x}$ can take other (literally any) nonlinear forms

# Fixed basis models

Fixed basis model assumes

$$f(\boldsymbol{x};\boldsymbol{\beta}) = \sum_{k=0}^{p-1} \beta_k \phi_k(\boldsymbol{x}) = \boldsymbol{\beta}^T \boldsymbol{\phi}(\boldsymbol{x})$$

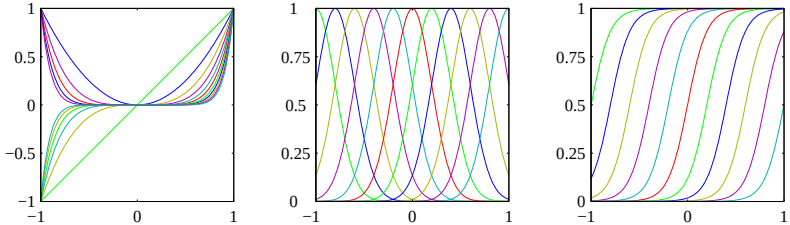
- "fixed" because the basis functions need to be manually specified
  - the contrary is adaptive basis (the basis are learnt)
- still linear w.r.t each new feature input $\phi_k$
  - all linear learning results apply to $\boldsymbol{\beta}$
- no longer linear in the orginal input $\boldsymbol{x}$

# Two popular basis in ML literature

In traditional Machine Learning literature, radial basis function (RBF) and sigmoid (or "tanh") are popular

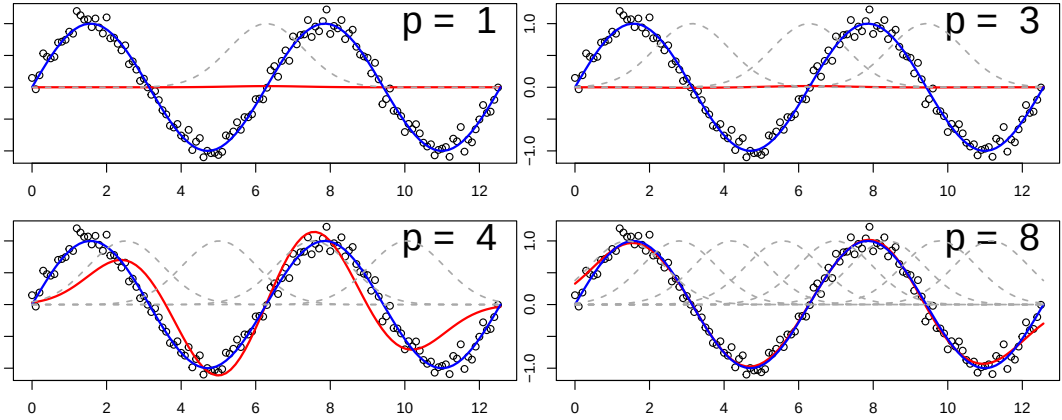
$$\underbrace{\phi_k(\boldsymbol{x}) = \exp\left(-\frac{||\boldsymbol{x} - \boldsymbol{\mu}_k||_2^2}{2s^2}\right)}_{\text{radial basis}}, \quad \underbrace{\phi_k(x) = \frac{1}{1 + \exp\left(-\frac{x-\mu_k}{s}\right)}}_{\text{sigmoid basis}}$$

- they are location based: depending on $\mu$; while polynomial basis are not (lower left)!
- very general basis and can fit various different models



# Example: radial basis function (RBF) with varying number of basis

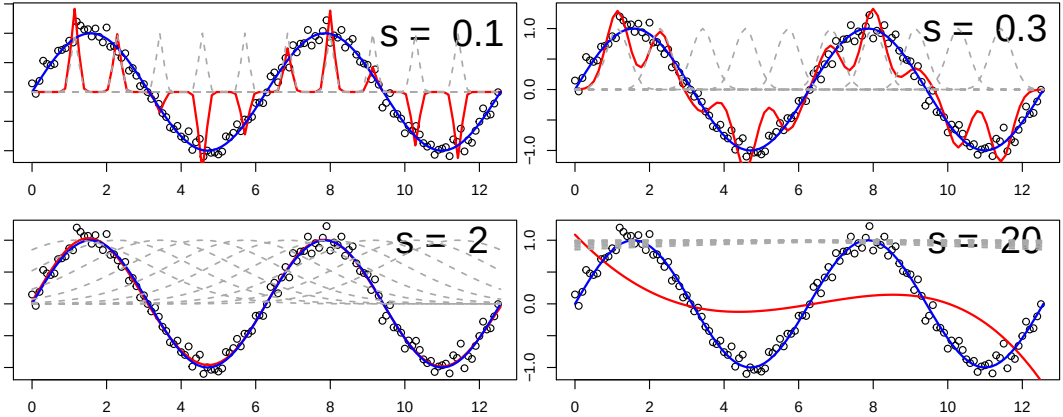
True regression function: $\mathbf{f} = \sin(\mathbf{x})$; and $y^{(i)} = \sin(x^{(i)}) + \epsilon^{(i)}$, $\epsilon \sim N(0, 0.2^2)$
More RBF basis (dashed gray lines) fits better; location $\mu_k$ matter ($p = 3$ is an (unlucky) example);



# Example: radial basis function (RBF) with varying scale $s$

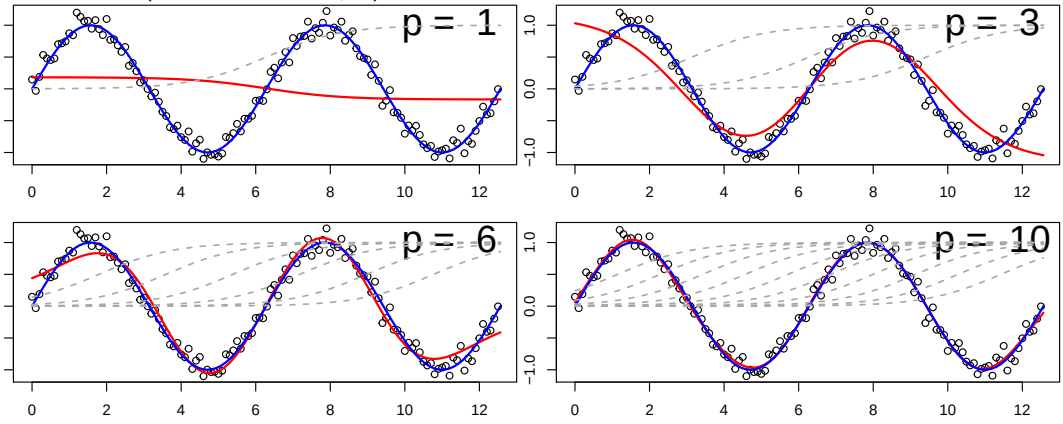
Smaller scale $s \Rightarrow$ wiggly predictions;

Large scale $s \Rightarrow$ flatter predictions, not good either ! ($p = 10$ basis for all four cases)



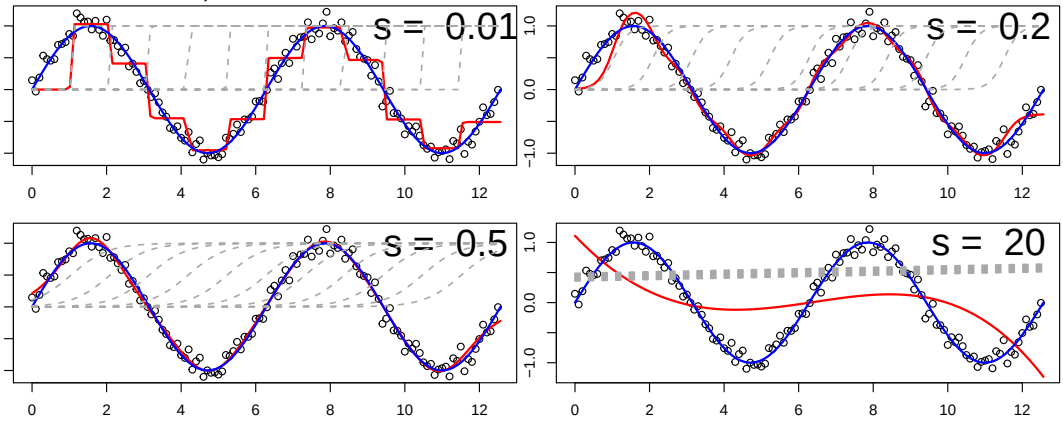
# Example: sigmoid basis with varying number of basis

More basis (more locations $\mu_k$) fits data better again



# Example: sigmoid basis with varying scale $s$

Smaller scale $s \Rightarrow$ step functions; Large scale $s \Rightarrow$ smoother predictions ($p = 11$ basis for all four cases)



# Interpretation of fixed basis models

In abstract vector space, functions are vectors, say $\phi_k(\boldsymbol{x})$

$$f(\boldsymbol{x};\boldsymbol{\beta}) = \sum_{k=0}^{p-1} \beta_k \phi_k(\boldsymbol{x})$$

- $f$: a linear combination of vectors $\{\phi_0, \phi_1, \ldots, \phi_p\}$
- e.g. $f(x) = \beta_0 + \beta_1 x$ is a linear combination of two functions (vectors): $\phi_0(x) = 1$ and $\phi_1(x) = x$

$$f(x) = \beta_0\phi_0(x) + \beta_1\phi_1(x)$$

- linear regression: $\hat{f} \in \text{span}(\{1, x\})$ with $\hat{\beta}_0, \hat{\beta}_1$
  - projection of $f$ to the subspace
- larger functional space $\text{span}(\{\phi_k(x)\}_1^p)$ ($p$ increases) $\Rightarrow$ better fit
  - until $f$ exactly lives in the span (which can be bad!)
  - regardless of $\phi_k$'s shapes as long as nonlinear and "different" (linear independent)

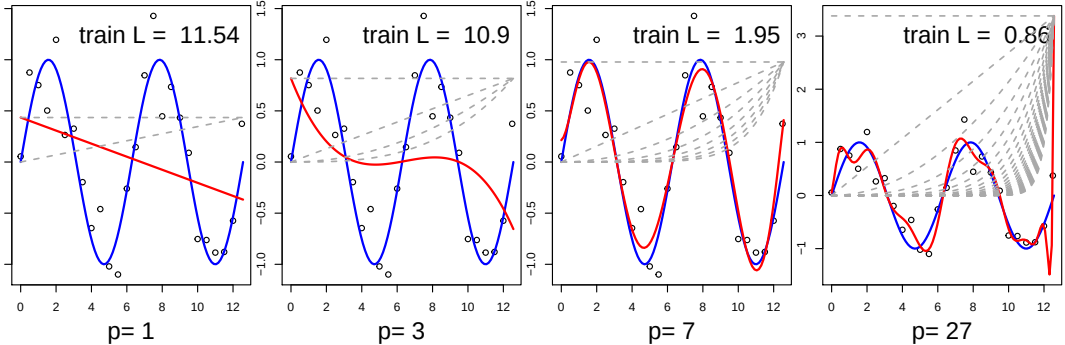
# Flexibility resolves underfitting but introduce overfitting

Training loss:

$$L(\beta) = \sum_{i=1}^{m} (y^{(i)} - \beta^T \phi(\boldsymbol{x}^{(i)}))^2$$

More basis functions ($p$ increases) $\Rightarrow$ flexible model (larger functional subspace)

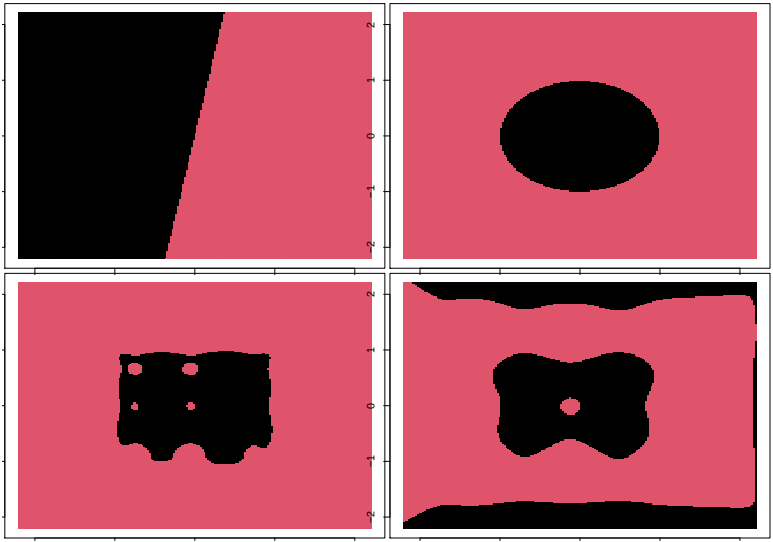
- MLE loss (squared error in training) uniformly decreases



# Classification example: polynomial basis

The training NLL drops

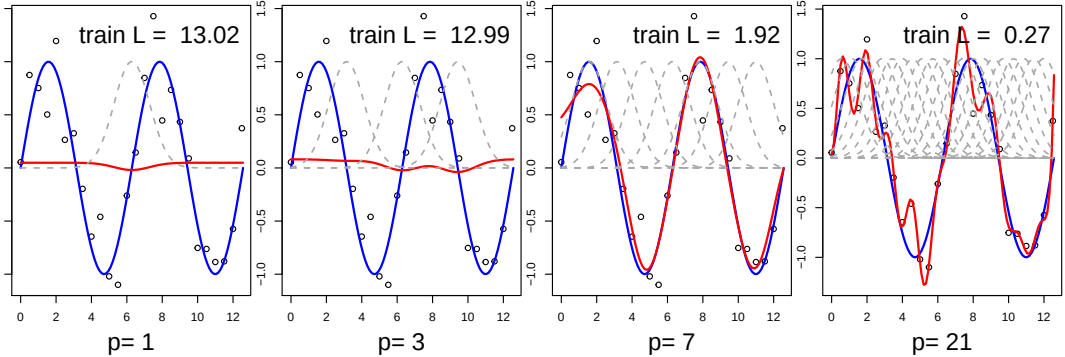
- possible to achieve 100% training accuracy
- regardless of dataset
- just keep expanding



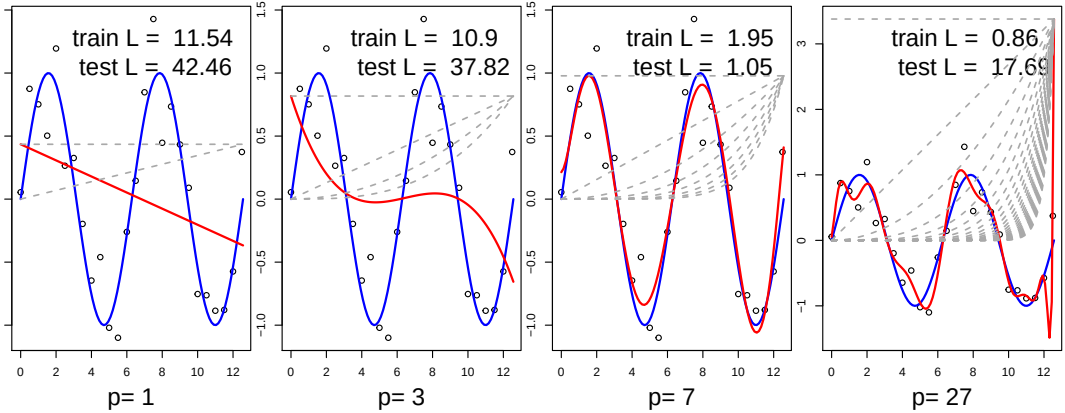
# Flexibility resolves underfitting but introduce overfitting (RBF example)

The problem applies to all basis (no matter what shape)

- as long as you expands the basis $\Rightarrow$ smaller training error
- MLE always favour flexible models



# Testing results reveals severe overfitting



# Regularisation: ridge regression

For linear regression, we add a $l_2$ norm penalty to the loss

$$L_{\text{ridge}}(\boldsymbol{\beta}) = \underbrace{\sum_{i=1}^{m}(y^{(i)} - \boldsymbol{\beta}^T\boldsymbol{\phi}(\boldsymbol{x}^{(i)}))^2}_{\text{previous NLL loss}} + \lambda \underbrace{\boldsymbol{\beta}^T\boldsymbol{\beta}}_{L_2 \text{ penalty}}$$

- as NLL (squared error loss) term always favour flexible models
- we add a penalty term: $\boldsymbol{\beta}^T\boldsymbol{\beta} = ||\boldsymbol{\beta}||_2^2$; its $l_2$ norm
- large $\boldsymbol{\beta} \Rightarrow$ higher penalty
  - "regularise" the value of $\boldsymbol{\beta}$: large value of $\boldsymbol{\beta}$ is discouraged
- large $\lambda \Rightarrow$ large penalty: controls the penalty

# Learning of ridge regression

Ridge regression has closed form solution (no surprise: still a quadratic function of $\beta$):

$$\beta_{\text{ridge}} = \underset{\beta}{\text{argmin}} \sum_{i=1}^{m} (y^{(i)} - \beta^T \phi(\boldsymbol{x}^{(i)}))^2 + \lambda \beta^T \beta = \underset{\beta}{\text{argmin}} \underbrace{(\boldsymbol{y} - \boldsymbol{\Phi}\beta)^T (\boldsymbol{y} - \boldsymbol{\Phi}\beta) + \lambda \beta^T \beta}_{L_{\text{ridge}}(\beta)}$$

Take the derivative

$$\nabla_\beta L_{\text{ridge}} = -2(\boldsymbol{y} - \boldsymbol{\Phi}\beta)^T \boldsymbol{\Phi} + 2\lambda \beta^T \qquad (\text{note } \nabla \beta^T \beta = 2\beta^T)$$

And set it to zero

$$-2(\boldsymbol{y} - \boldsymbol{\Phi}\beta)^T \boldsymbol{\Phi} + 2\lambda \beta^T = \boldsymbol{0} \Rightarrow \boldsymbol{\Phi}^T \boldsymbol{\Phi}\beta + \lambda\beta = \boldsymbol{\Phi}^T \boldsymbol{y}$$

$$\Rightarrow (\boldsymbol{\Phi}^T \boldsymbol{\Phi} + \lambda \boldsymbol{I})\beta = \boldsymbol{\Phi}^T \boldsymbol{y} \qquad (\text{note } \lambda\beta = \lambda \boldsymbol{I}_{p \times p}\beta)$$

$$\Rightarrow \beta_{\text{ridge}} = (\boldsymbol{\Phi}^T \boldsymbol{\Phi} + \lambda \boldsymbol{I})^{-1} \boldsymbol{\Phi} \boldsymbol{y}$$

# Learning for logistic regression (with $l_2$ penalty)

Similarly, we can derive the results for logistic regression ($l_2$ penalty)

$$\beta_{\text{ridge}} = \underset{\beta}{\text{argmin}} \underbrace{-\left(\sum_{i=1}^{m} y^{(i)} \log \sigma^{(i)} + (1 - y^{(i)}) \log(1 - \sigma^{(i)})\right)}_{\text{Negative log likelihood}} + \underbrace{\lambda \beta^T \beta}_{l_2 \text{ penalty}}$$

The gradient is

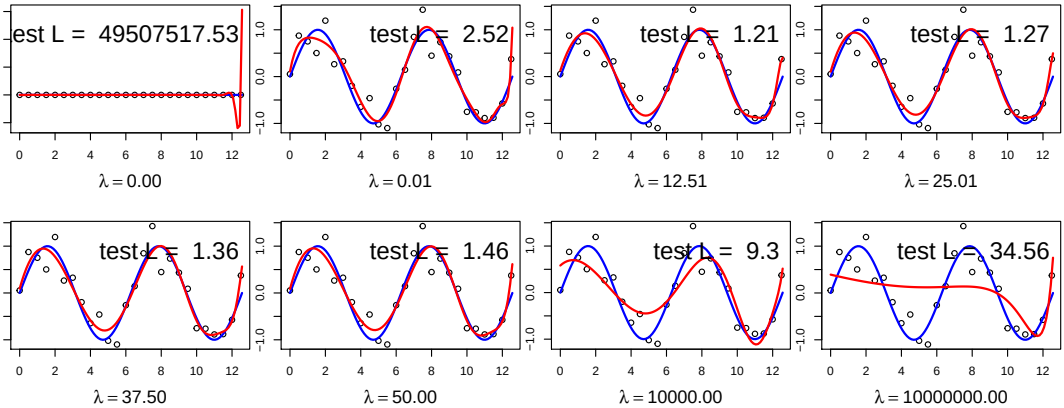
$$\nabla_\beta L_{\text{ridge}} = \underbrace{-(\boldsymbol{y} - \sigma(\boldsymbol{\Phi}\beta))^T \boldsymbol{\Phi}}_{\text{same as before but negated}} + \underbrace{2\lambda\beta^T}_{\nabla \text{penalty}} \quad \text{(I am abusing the notation for } \sigma \text{ as before)}$$

The Hessian is

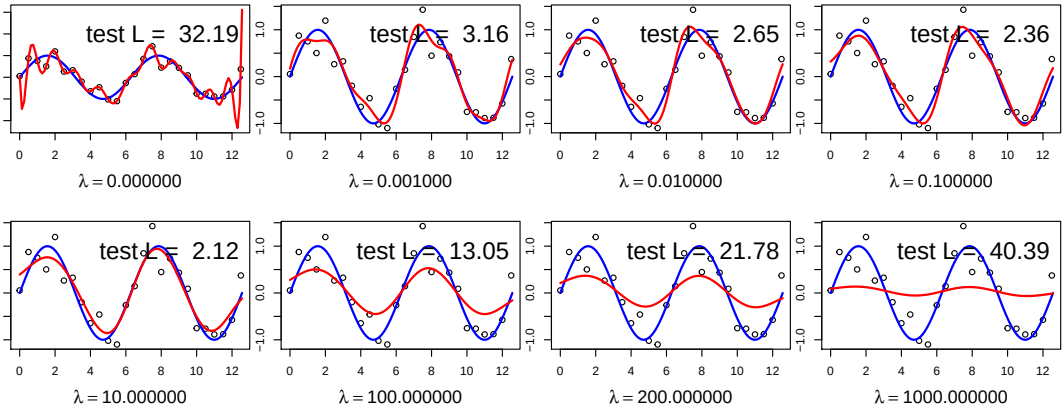
$$H_{\text{ridge}} = -\boldsymbol{X}^T \boldsymbol{D} \boldsymbol{X} + \lambda \boldsymbol{I}$$

- "-": minimise the negative LL (directional curvature is bending upwards)
- check lecture 6 for $\boldsymbol{D}$

# Example: Polynomial basis regression (28 basis) with $l_2$ penalty



# Example: Radial basis regression (28 basis) with $l_2$ penalty



# $l_2$ penalty in practice

We do not penalise $\beta_0$ (intercept), $\boldsymbol{I}_{p\times p}$ is usually replaced with $\boldsymbol{I}'_{p\times p} = \begin{bmatrix} 0 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ 0 & 0 & \dots & 0 \\ 0 & 0 & \dots & 1 \end{bmatrix}$

!!! Ridge regression is sensitive to the way you scale your data !

- should consider transformation that make the $l_2$ norm to 1: standardisation (columns of $\boldsymbol{\Phi}$'s norm is 1, i.e. $||\phi_k||_2 = 1$)
- other scaling may not produce optimal result
- sk-learn use $l_2$ penalty by default for logistic regression
  - for numerical stability reason (remember logistic regression's matrix inversion?)

# Let's demystify what ridge regression does

$$\beta_{\text{ridge}} = (\boldsymbol{\Phi}^T\boldsymbol{\Phi} + \lambda \boldsymbol{I})^{-1}\boldsymbol{\Phi}\boldsymbol{y}$$

- obviously, $\lambda \to 0$, $\beta_{\text{ridge}} \to \beta_{\text{ml}}$ (which is $\boldsymbol{\Phi}^T\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}\boldsymbol{y}$)
- $\lambda \to \infty$, assume $\boldsymbol{\Phi}$ are formed by orthogonormal basis (columns), i.e. $\boldsymbol{\Phi}^T\boldsymbol{\Phi} = \boldsymbol{I}_{p\times p}$

$$\beta_{\text{ridge}} = (\boldsymbol{I} + \lambda \boldsymbol{I})^{-1}\boldsymbol{\Phi}\boldsymbol{y} = \frac{1}{\lambda+1}\boldsymbol{\Phi}\boldsymbol{y}$$

  $\beta \to \mathbf{0}$ when $\lambda \to \infty$
- how about those sane $\lambda$?

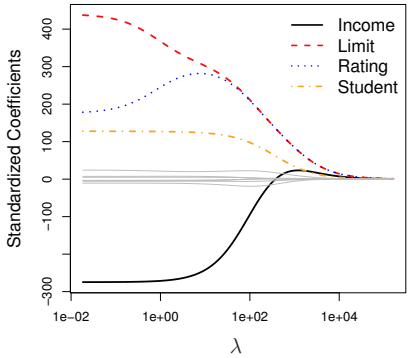
$$\beta_{\text{ridge}} = \frac{1}{\lambda+1}\boldsymbol{\Phi}\boldsymbol{y} = \frac{1}{\lambda+1}\underbrace{(\boldsymbol{\Phi}^T\boldsymbol{\Phi})^{-1}}_{\boldsymbol{I}^{-1}=\boldsymbol{I}}\boldsymbol{\Phi}\boldsymbol{y} = \underbrace{\frac{1}{\lambda+1}}_{\text{btw [0,1]}}\beta_{\text{ml}}$$

  it shrinks the ML estimator by some percentage.

Orthogonormal basis are common: e.g. Fourier basis. If $\boldsymbol{\Phi}$ is not, the result is still similar: just replace 1 with the eigen value of $\boldsymbol{\Phi}^T\boldsymbol{\Phi}$

# Example

$\lambda \to 0$ (to the left): $\beta_{\text{ridge}} \to \beta_{\text{ml}}$
$\lambda \to \infty$ (to the right): $\beta_{\text{ridge}} \to \mathbf{0}$
other $\lambda$: somewhere in between; being shrinked by a percentage




taken from An introduction to statistical learning; Chapter 6, James et al.

# Lasso regression $l_1$ penalty

There is another popular choice for penalty: $l_1$ norm

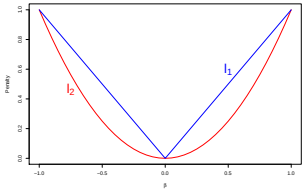
$$L_{\text{lasso}}(\beta) = \underbrace{\sum_{i=1}^{m}(y^{(i)} - \beta^T \phi(\boldsymbol{x}^{(i)}))^2}_{\text{previous NLL loss}} + \lambda \underbrace{||\beta||_1}_{l_1 \text{ penalty}}$$

- $l_1$ norm: $||\boldsymbol{x}||_1 = \sum_{j=1}^{n} |x_j|$, $l_2$ norm: $||\boldsymbol{x}||_2^2 = \sum_{j=1}^{n} x_j^2 = \boldsymbol{x}^T\boldsymbol{x}$
- ridge shrinks (discount) the ML estimator by some ratio: $\beta_k^{\text{ridge}} = \delta\beta_k^{\text{ml}}$ $(0 < \delta < 1)$
- lasso directly shut them $\beta_k^{\text{lasso}} = 0$
- why though?

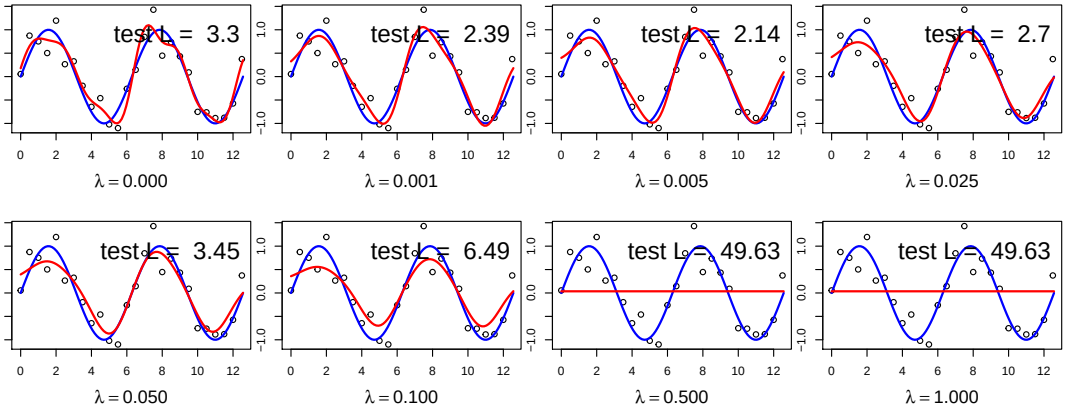
Assume applying gradient descent and $x \to 0^+$

$$\nabla_x l_1(x) = -1; \quad \nabla_x l_1(x) = -x$$

Penalty $l_1$ is constantly (strong); but $l_2$ diminishes



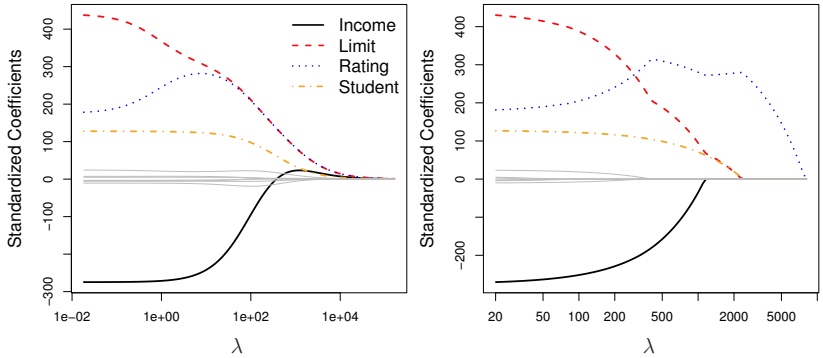
# Example: Radial basis regression (28 basis) with $l_1$ penalty



# Compare $l_1$ and $l_2$ penalty

Compare and contrast ridge and lasso: ($\lambda$ has similar effect, i.e. regularisation)

- ridge shrinks the estimator by some percentage (smooth curve)
- Lasso more black and white (feature selection + learning)

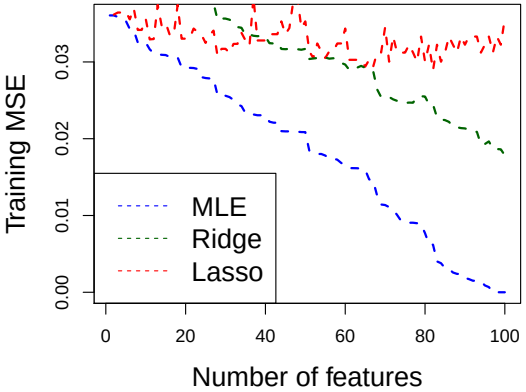


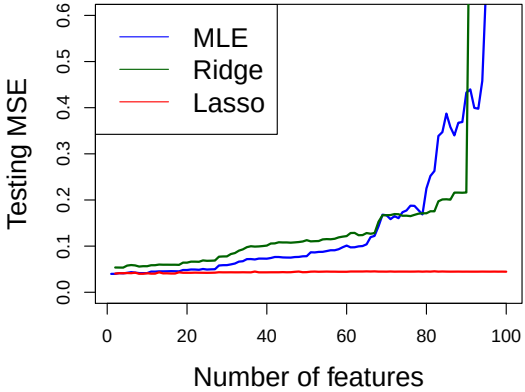

taken from An Introduction to Statistical Learning by James et al.

# Compare $l_1$ and $l_2$ penalty

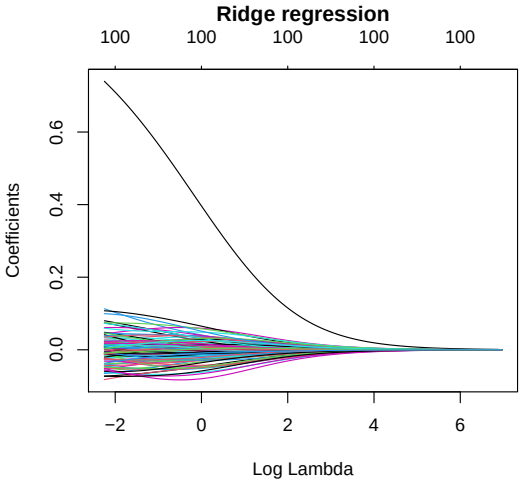
Consider $\boldsymbol{X}$ is a $m = 100$, $n = 100$ (features) matrix sampled from Gaussian ($N(0, 1)$)

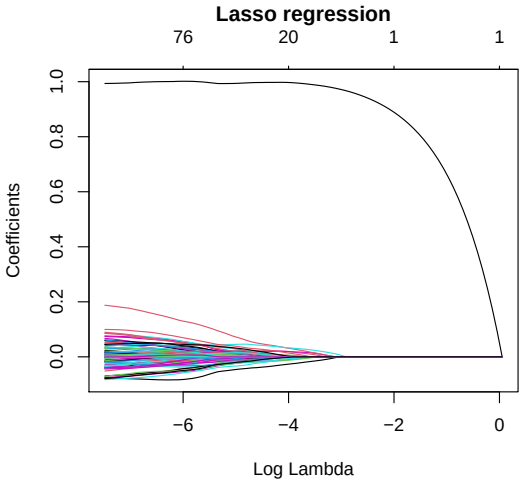
$y^{(i)} = x_1^{(i)} + \epsilon$; $\epsilon \sim N(0, \sigma^2 = 0.2^2)$



# The full path



# Another view: regularisation is MAP estimation

Regularisation are just Maximum A Posteriori (MAP) estimator

$$P(\beta|\mathbf{\Phi}, \boldsymbol{y}) \propto \underbrace{P(\boldsymbol{y}|\beta, \mathbf{\Phi})}_{\text{Likelihood}} \underbrace{P(\beta)}_{\text{Prior}}$$

MAP estimation maximise the posterior instead

$$\beta_{\text{MAP}} = \underset{\beta}{\text{argmax}}\, P(\beta|\mathbf{\Phi}, \boldsymbol{y})$$

Why?

$$\log P(\beta|\mathbf{\Phi}, \boldsymbol{y}) \propto \log P(\boldsymbol{y}|\beta, \mathbf{\Phi}) + \log P(\beta)$$

if we assume $P(\beta) = N(0, \gamma \boldsymbol{I})$ (a multivariate Gaussian with diagonal covariance)

$$\log P(\beta) = C - \frac{1}{2\gamma}\beta^T\beta$$

Therefore: $\beta_{\text{MAP}} = \beta_{\text{ridge}}$

# Suggested reading

- MLAPP 7.5, 13.3, 13.4*
- ESL 3.4
- interesting paper*: A comparison of numerical optimizers for logistic regression by Thomas Minka. https://tminka.github.io/papers/logreg/minka-logreg.pdf

Exercise for this lecture (you can discuss it with me in lab session or with your classmates)
ESL 3.12 Show that the ridge regression estimatees can be obtained by ordinary least squares regression on an augmented data set. We augment the centered matrix $\boldsymbol{X}$ with p additional rows $\sqrt{\lambda}\boldsymbol{I}$, and augment $y$ with p zeros. By introducing artificial data having response zero, the fitting forces a shrinkage.