II – Principles of Supervised Learning

Contents:

- *Bias-variance trade-off for supervised problems*
- *Overfitting, underfitting and test set*
- *Implicit and explicit regularisations*
- *Case-study: Ridge Regression*

In the previous chapter, we have:

1. Specified different models for different supervised learning tasks (regression and classification),
2. Specified loss functions and associated empirical risks,
3. Used a finite training set to minimise the empirical risk,
4. Found parameters of our models $f_{\boldsymbol{\theta}}(\boldsymbol{\phi}^{(i)})$.

Now what could possibly go wrong with our models?

Generalisation!

Is it able to work on new, independent from training, data?

The bias-variance trade-off

- The whole aim of training supervised models is to generalise well on unseen data. In practice, we would like to minimise $\mathbb{E}_{\boldsymbol{x},y}\big(\ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}), y)\big)$ that we approximate by $\mathrm{R}_{\text{train}} = \frac{1}{n}\sum_{i=1}^{n} \ell\left(\left(f_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right), y^{(i)}\right)\right)$
- In a regression context, suppose there exists f such that $y^{(i)} = f\left(\boldsymbol{x}^{(i)}\right) + \epsilon^{(i)}$, with $\mathbb{E}\left[\epsilon^{(i)}\right] = 0, \mathbb{E}\left[\epsilon^{(i)^2}\right] = \sigma_\epsilon^2$
- We build a model f_θ of f minimising the squared error $\ell\left(\left(f_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right), y^{(i)}\right)\right) = \left(y^{(i)} - f_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right)\right)^2$
- We can show that the expected risk on a **test example $\tilde{\boldsymbol{x}}$** (never seen by the model during training) decomposes as

$$\mathbb{E}\left[\left(y - f_{\boldsymbol{\theta}}(\tilde{\boldsymbol{x}})\right)^2\right] = \mathbb{E}[f_{\boldsymbol{\theta}}(\tilde{\boldsymbol{x}}) - f(\tilde{\boldsymbol{x}})]^2 + \mathbb{E}[(f_{\boldsymbol{\theta}}(\tilde{\boldsymbol{x}}) - \mathbb{E}[f_{\boldsymbol{\theta}}(\tilde{\boldsymbol{x}})])^2] + \sigma_\epsilon^2$$

$$\mathbb{E}\left[\left(y - f_{\boldsymbol{\theta}}(\tilde{\boldsymbol{x}})\right)^2\right] = \mathrm{Bias}[f_{\boldsymbol{\theta}}(\tilde{\boldsymbol{x}})]^2 + \mathrm{Var}[f_{\boldsymbol{\theta}}(\tilde{\boldsymbol{x}})] + \sigma_\epsilon^2$$

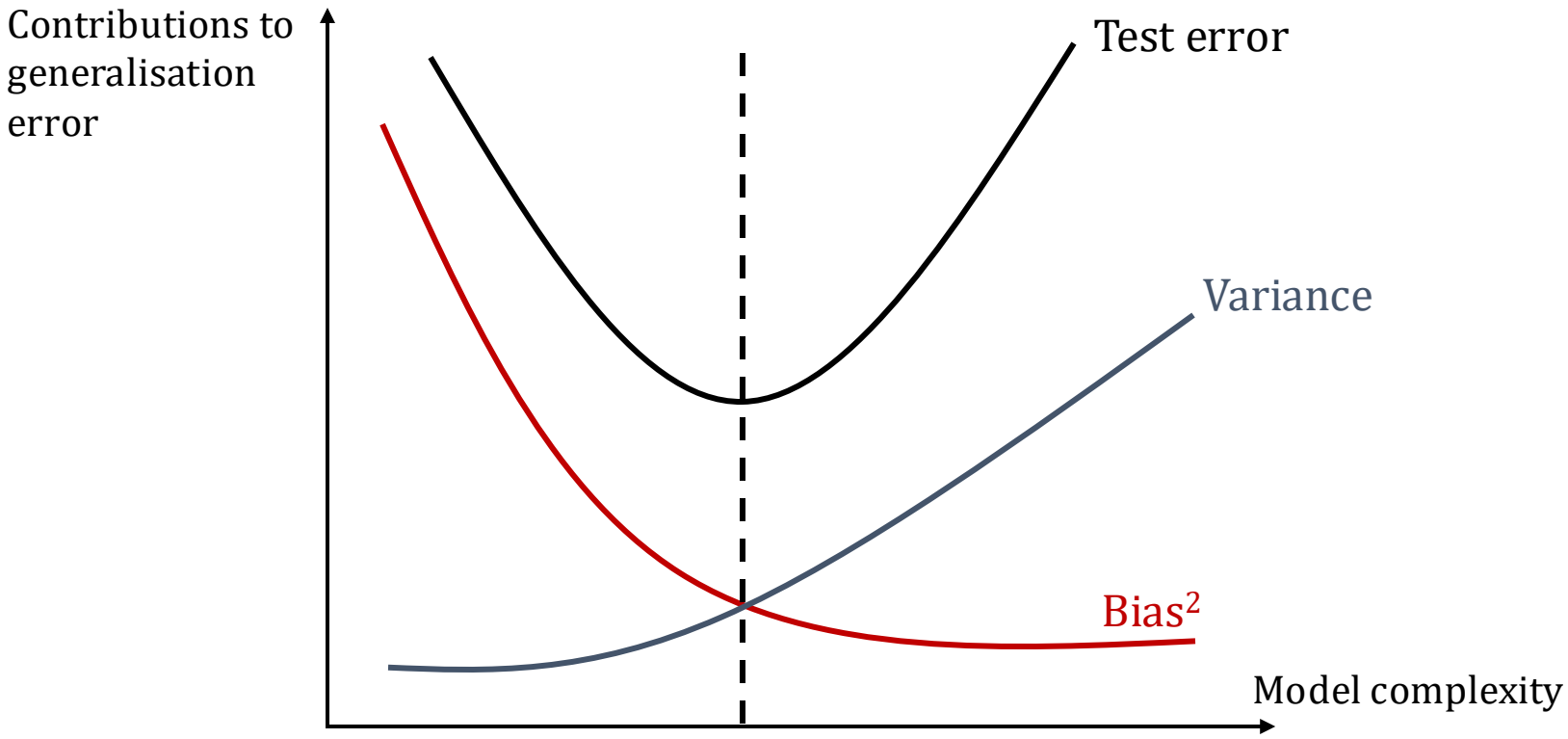
Modelling error | Model variability | Irreducible error

Note: Every terms are >0

Note: a similar expression holds for classification

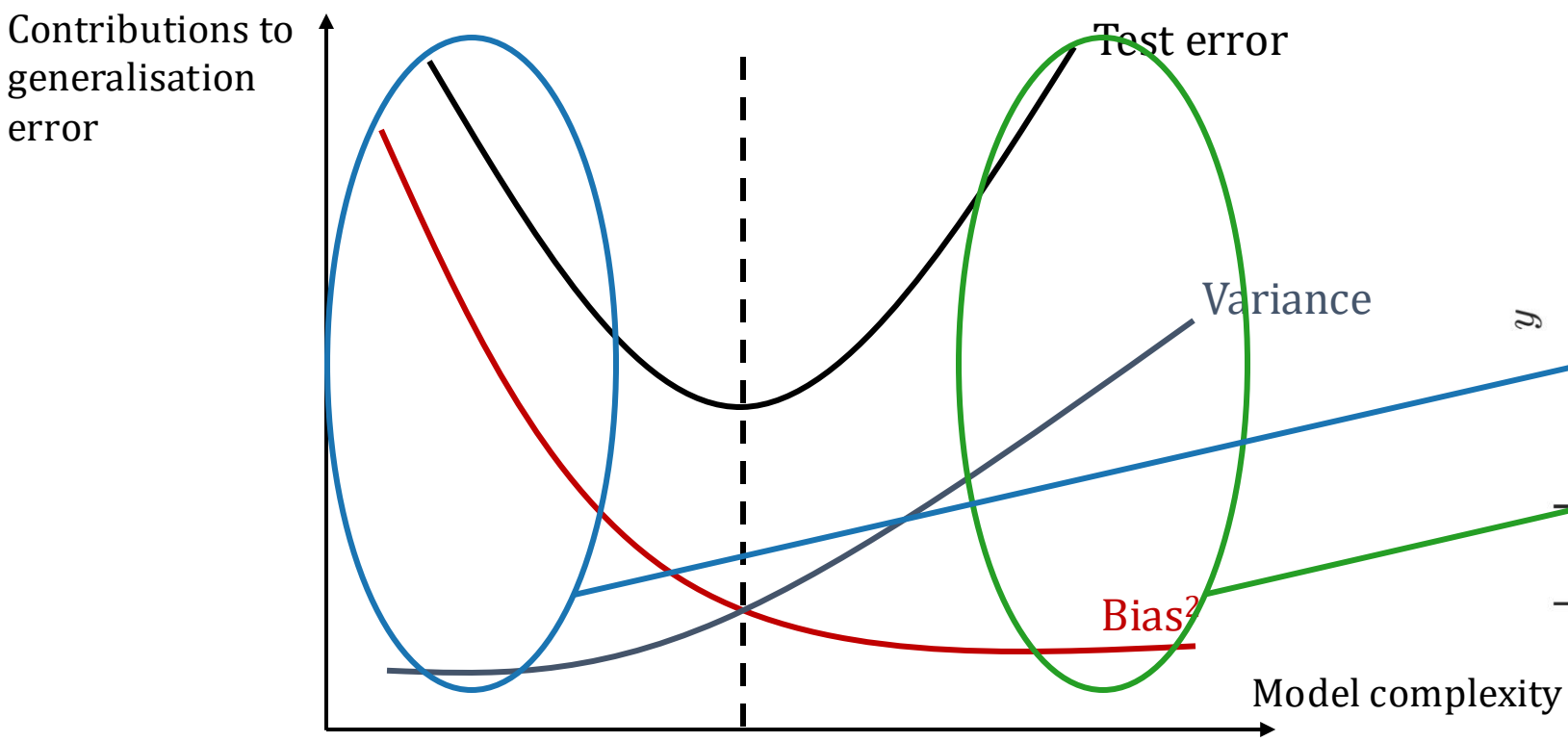
Model complexity and generalisation error

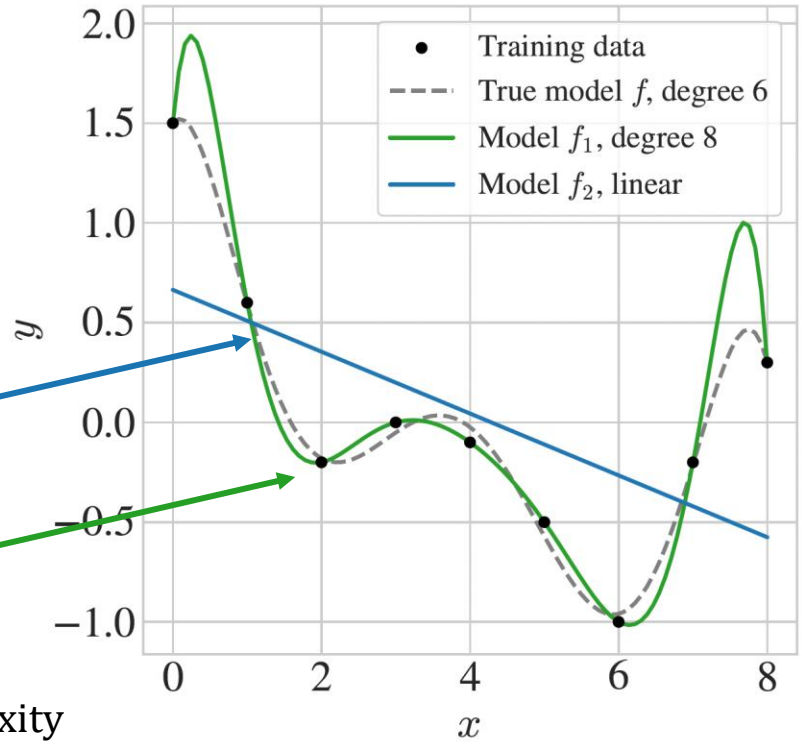
- **"Simple" models** have **large bias** because they constrain very much the function class that is therefore far from the truth, but they usually have **low variance** and are robust to variations of the training set
- "**Complex" models** (with a lot of parameters for instance) have **small bias** but **large variance**



Model complexity and generalisation error

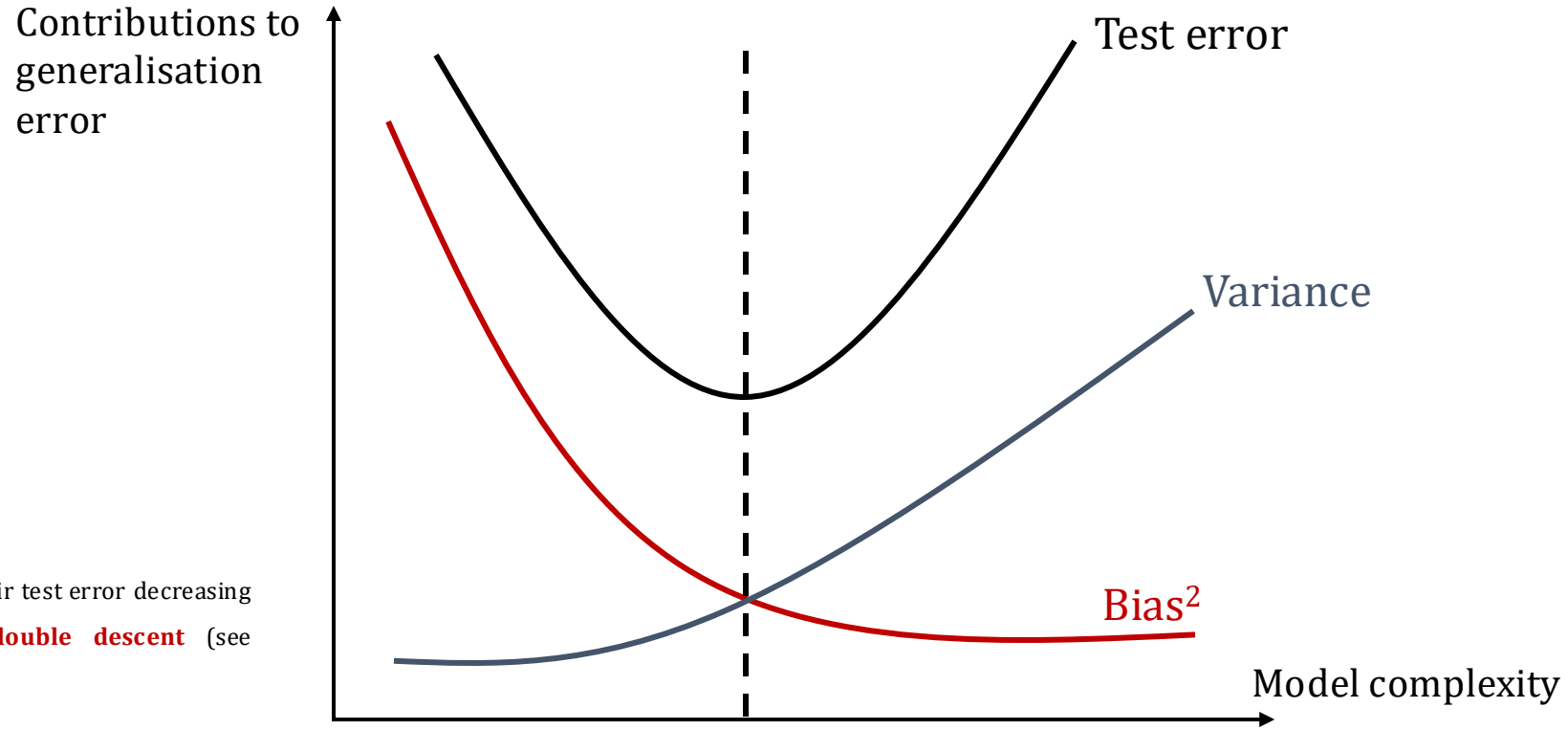
- **"Simple" models** have **large bias** because they constrain very much the function class that is therefore far from the truth, but they usually have **low variance** and are robust to variations of the training set
- "**Complex" models** (with a lot of parameters for instance) have **small bias** but **large variance**





Model complexity and generalisation error

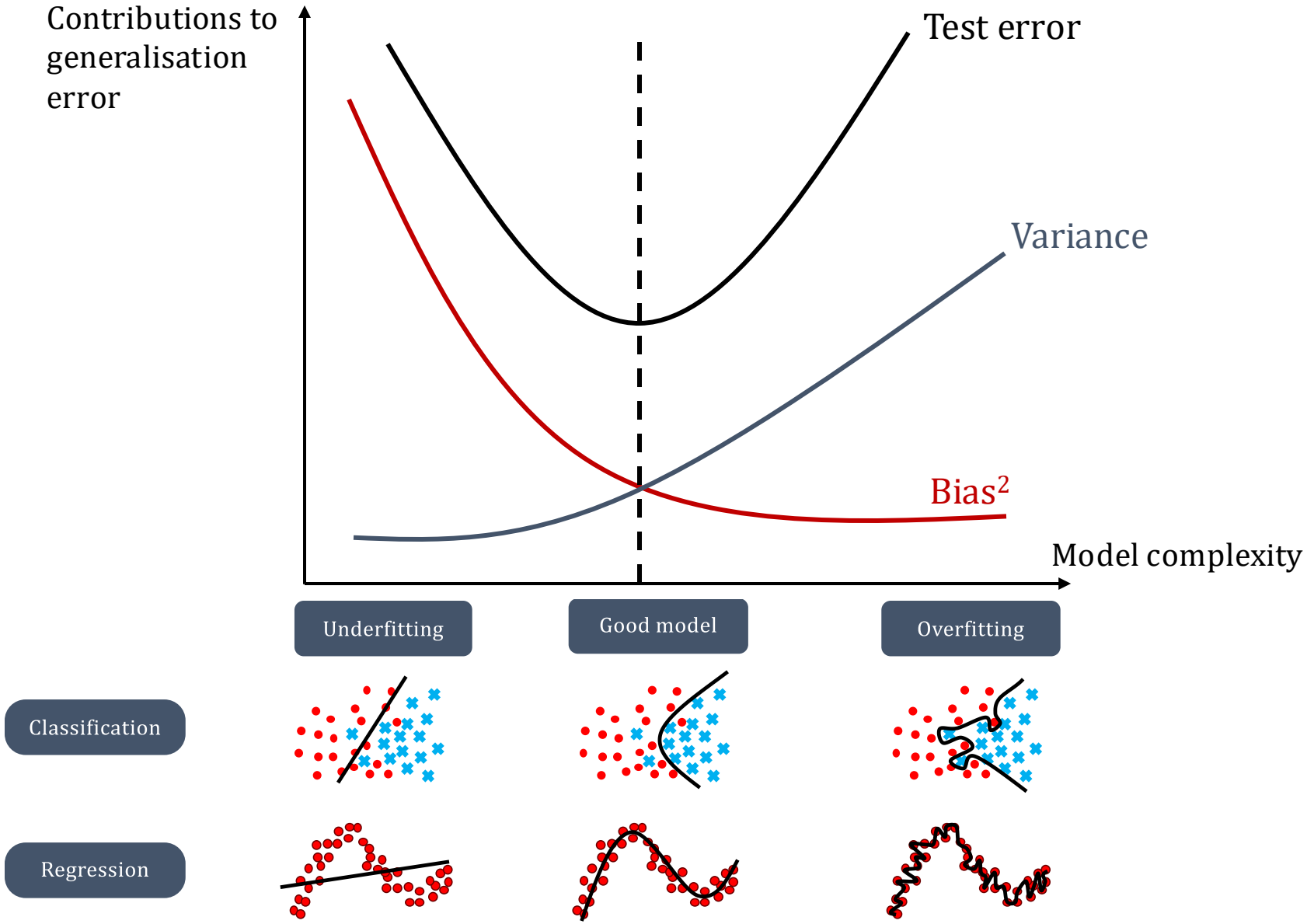
- **"Simple" models** have **large bias** because they constrain very much the function class that is therefore far from the truth, but they usually have **low variance** and are robust to variations of the training set
- "**Complex" models** (with a lot of parameters for instance) have **small bias** but **large variance***



*Heavily overparametrized have their test error decreasing again, a phenomenon dubbed **double descent** (see Belkin+18 and Nakkiran+19)

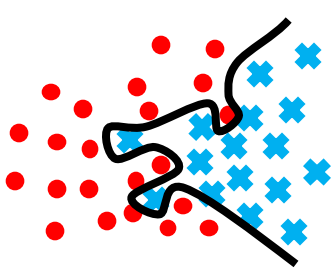
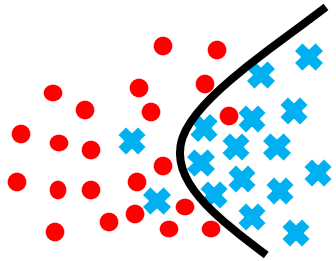
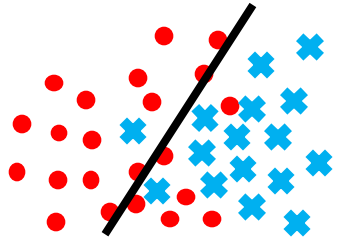
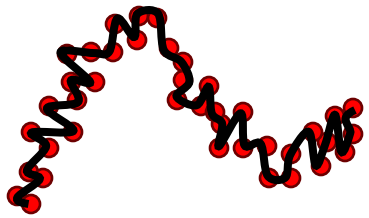
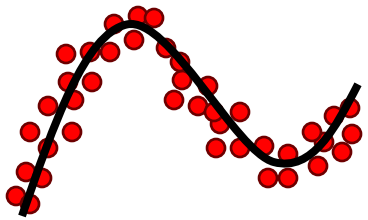
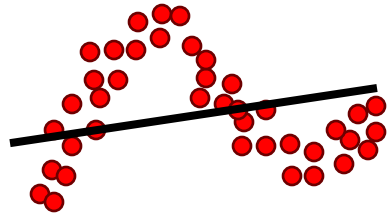
Models need to be built such that they are not too flexible to fit the noise in the data but also not too restrictive to avoid bias

Model complexity and generalisation error

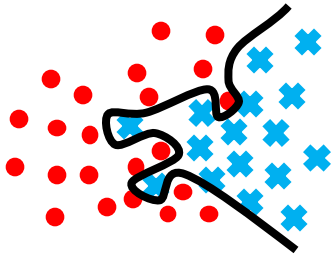
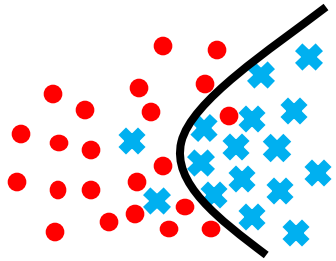
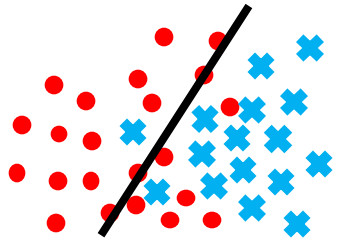
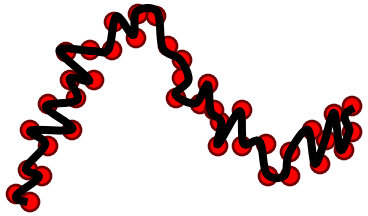
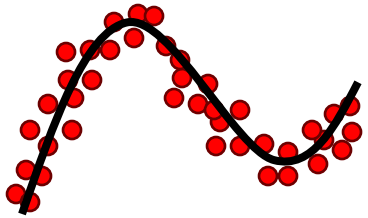
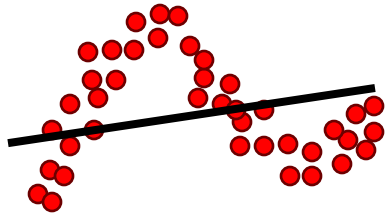


Generalisation in practice

- To measure if we really learnt something useful in supervised learning in practice, we use a **test dataset** that the model has never seen but for which we know the labels and check that $R_{\text{train}}(\hat{\theta})$ and $R_{\text{test}}(\hat{\theta})$ are of the same order
- Based on the previous view, there are two regimes where things could go wrong: high variance and low bias models vs high bias and low variance models, respectively defining **overfitting** and **underfitting**

	Overfitting	Good model	Underfitting
Classification			
Regression			
	$R_{\text{train}}(\hat{\boldsymbol{\theta}}) \ll R_{\text{test}}(\hat{\boldsymbol{\theta}})$	$R_{\text{train}}(\hat{\boldsymbol{\theta}}) \approx R_{\text{test}}(\hat{\boldsymbol{\theta}})$	$R_{\text{train}}(\hat{\boldsymbol{\theta}}) \approx R_{\text{test}}(\hat{\boldsymbol{\theta}})$ but large

Some fixes to under and over fitting

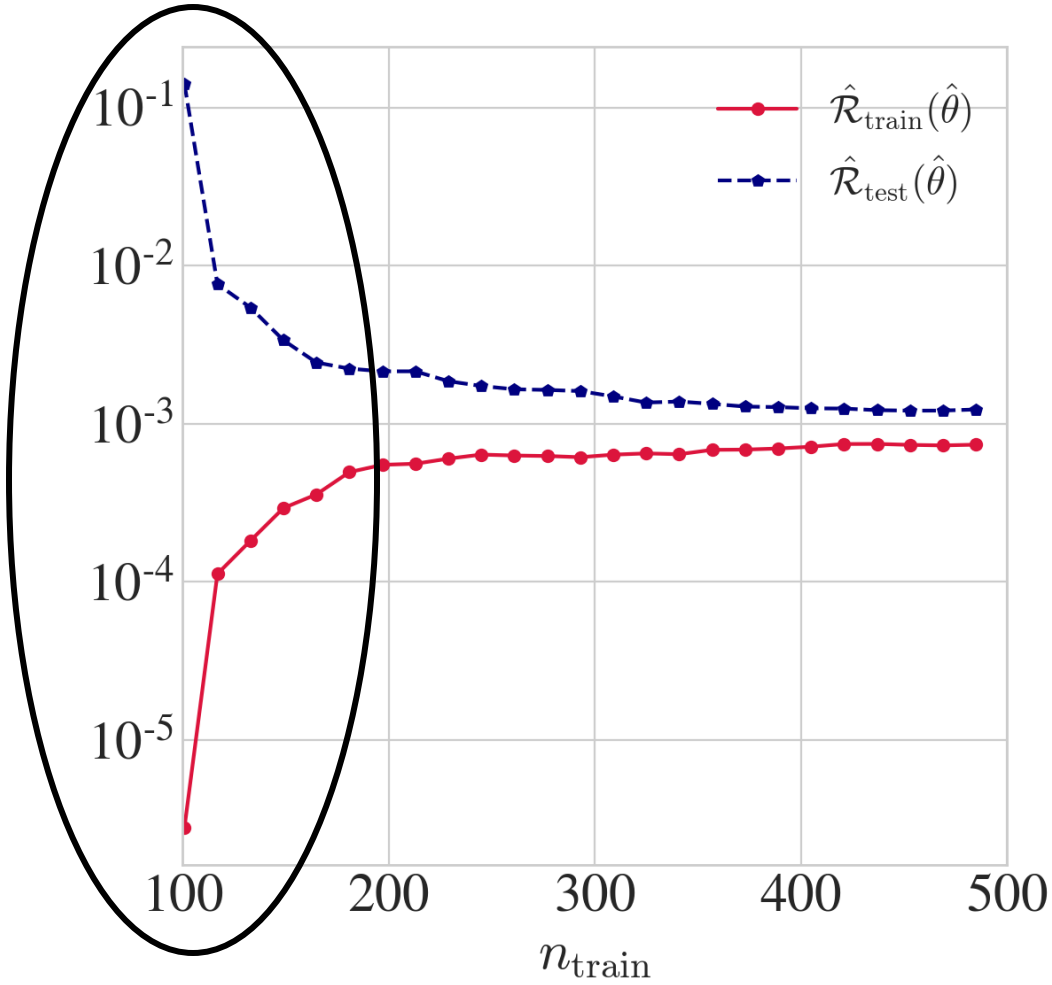
	Overfitting	Good model	Underfitting
Classification			
Regression			
	$R_{\text{train}}(\hat{\boldsymbol{\theta}}) \ll R_{\text{test}}(\hat{\boldsymbol{\theta}})$	$R_{\text{train}}(\hat{\boldsymbol{\theta}}) \approx R_{\text{test}}(\hat{\boldsymbol{\theta}})$	$R_{\text{train}}(\hat{\boldsymbol{\theta}}) \approx R_{\text{test}}(\hat{\boldsymbol{\theta}})$ but large
Possible fixes	• Add more data • Remove features • Stop the training earlier • Add **regularisation**	• You did a good job!	• Try a more complex model • Train your model longer

Overfitting in linear model

- Back to our regression problem: what happens when $n_{\text{train}} \approx d'$? Remember the solution

$$\hat{\boldsymbol{\theta}} = \left(\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi}\right)^{-1}\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{y}$$

Overfitting!



Linear regression with $d' = 100$ features

A gentle introduction to regularisation

- One generic way to deal with **overfitting** is by penalising 'complex' models and restrain the class of learned functions: this is called explicit **regularization** and appears in the optimization problem as a constraint on parameter space

$$\hat{\theta} = \underset{\boldsymbol{\theta}\in\Theta}{\operatorname{argmin}} R(\boldsymbol{\theta})$$

Unregularized optimization

$$\hat{\theta} = \underset{\boldsymbol{\theta}\in\Theta}{\operatorname{argmin}} R(\boldsymbol{\theta}) \text{ s.t. } P(\boldsymbol{\theta}) \leq \epsilon$$

Regularized optimization

- The function $P(\boldsymbol{\theta})$ is called **penalty function** and the regularized problem can in fact be written equivalently as (by Lagrange duality)

$$\hat{\theta} = \underset{\boldsymbol{\theta}\in\Theta}{\operatorname{argmin}} R(\boldsymbol{\theta}) + \lambda P(\boldsymbol{\theta})$$

1. λ is a **regularization** (hyper)**parameter**
2. $P(\boldsymbol{\theta})$ can take different forms depending on the penalty we chose to impose on the set of parameters

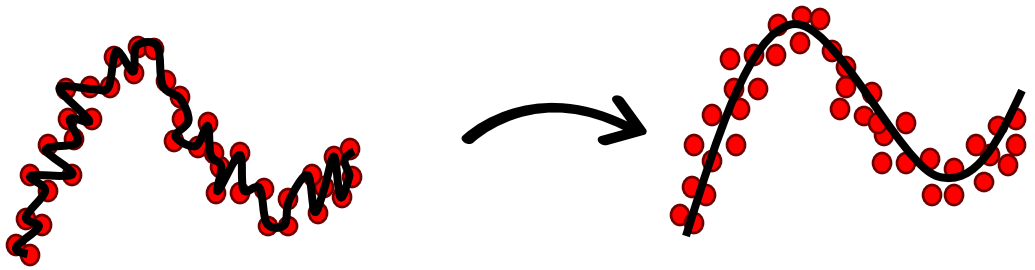
- Common penalty functions are the L_p-norms with $p = 1$ or 2

$$P(\boldsymbol{\theta}) = \|\boldsymbol{\theta}\|_p$$

Regularisation as a smoothness constraint

- By solving the regularized optimization with **an appropriate value of λ**, we can reduce the overfitting

Unregularized optimization



Regularized optimization

$$\hat{\theta} = \underset{\boldsymbol{\theta}\in\Theta}{\operatorname{argmin}}\, R(\boldsymbol{\theta}) \qquad \hat{\theta} = \underset{\boldsymbol{\theta}\in\Theta}{\operatorname{argmin}}\, R(\boldsymbol{\theta}) + \lambda\|\boldsymbol{\theta}\|_2^2$$

- Let us apply L_2 regularisation to our linear regression model which now minimises the regularised risk

$$\hat{\theta} = \underset{\boldsymbol{\theta}\in\Theta}{\operatorname{argmin}} \frac{1}{n}\|\boldsymbol{\Phi\theta} - \boldsymbol{Y}\|_2^2 + \lambda\|\boldsymbol{\theta}\|_2^2$$

- This can be minimised analytically and gives

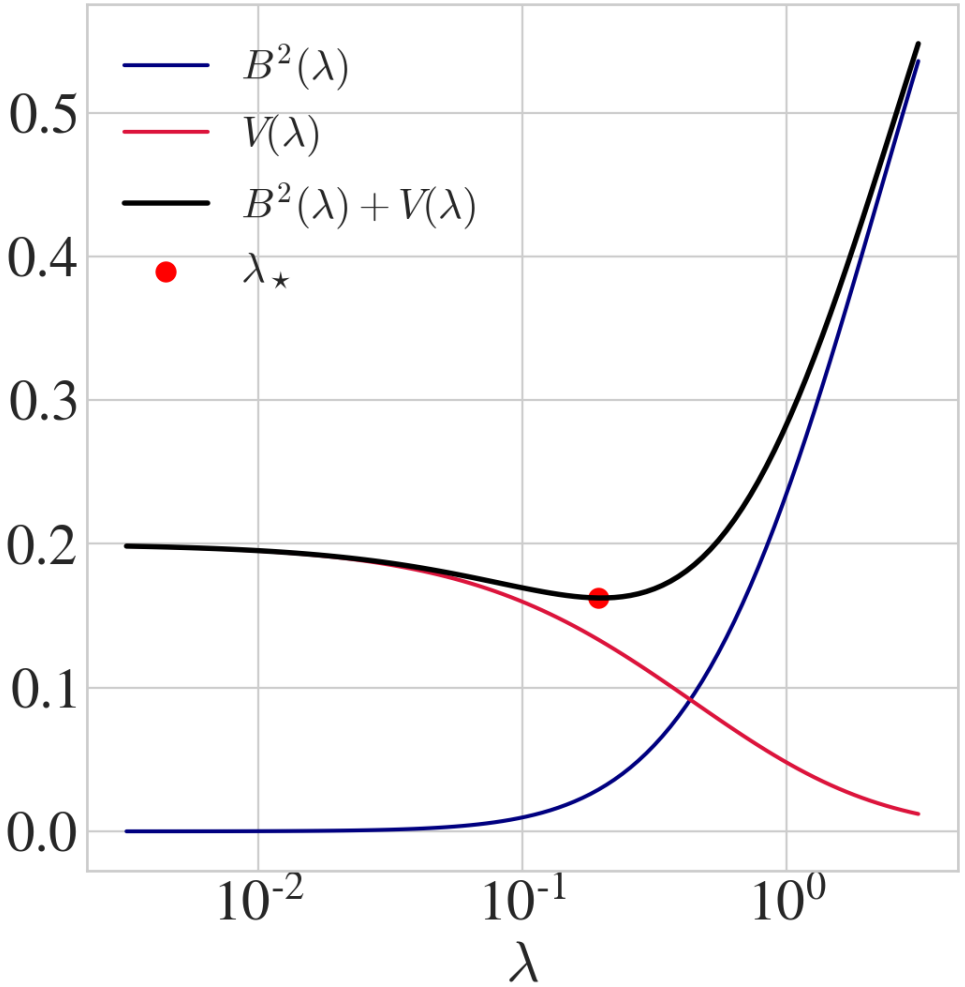
$$\hat{\theta} = \frac{1}{n}\left(\frac{\boldsymbol{\Phi}^{\mathbf{T}}\boldsymbol{\Phi}}{n} + \lambda\boldsymbol{I}\right)^{-1}\boldsymbol{\Phi}^{\mathbf{T}}\boldsymbol{Y}$$

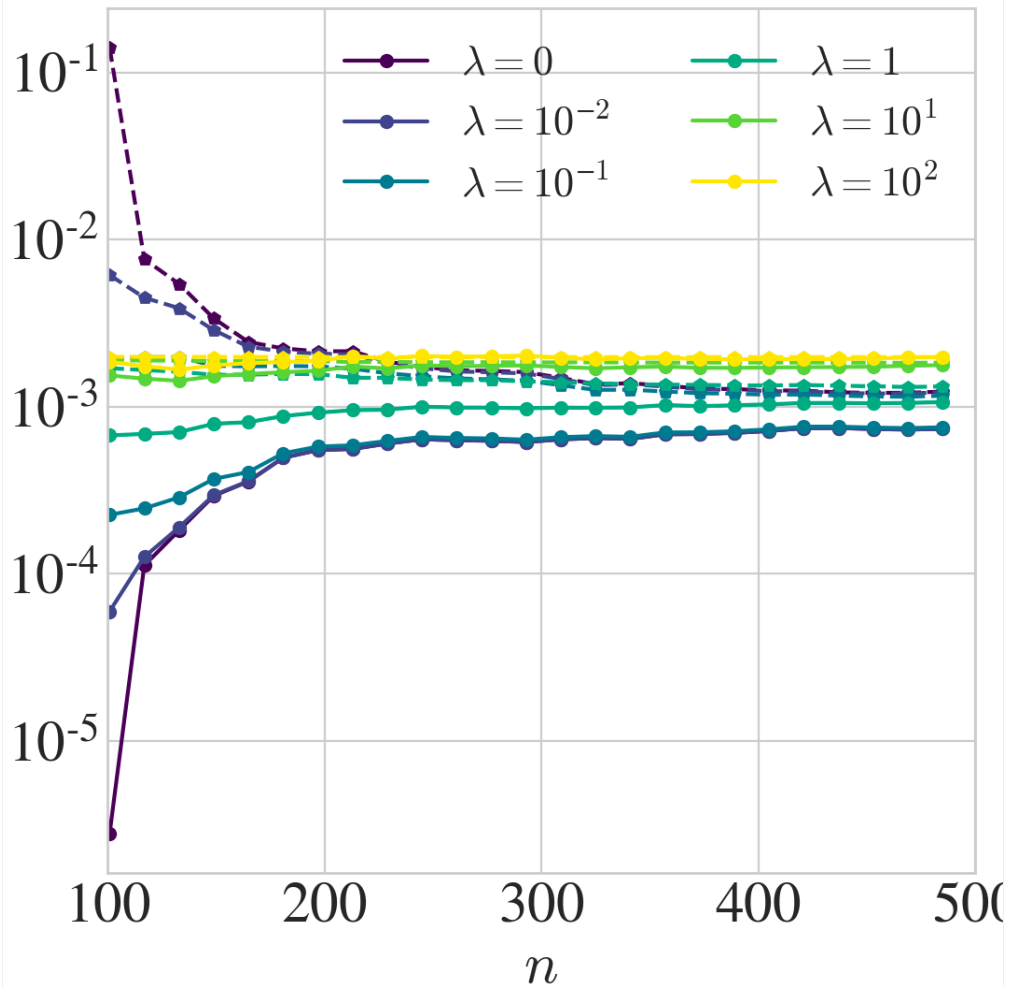
where the matrix is now invertible for $\lambda > 0$.

Regularisation as a smoothness constraint

The penalty impact the generalisation error by increasing the bias and decreasing the variance with λ

→ **Regularisation helps reducing overfitting and improves the generalisation performances!**





A word about hyperparameters

- One **hyperparameter**: the regularisation parameter λ

Hyperparameter: parameter that is **not learned** during the optimisation.
Examples include **regularisation parameters**, **depth of trees**, **learning rate** in optimisation.

- For the regularisation parameter: a value that is too large introduces a large bias, while if too small and close to zero, we do not solve the overfitting issue
- Usually, we use **grid search** to find the hyperparameter performing best on a third dataset: the **validation set**

- **Training set**: training data used to to learn parameters of the model during optimisation,
- **Test set**: independent set used to evaluate and compare the models (should in principle be used once by a model)
- **Validation set**: used to fit hyperparameters of the model by varying it and keeping the value minimising the validation error R_{valid}.

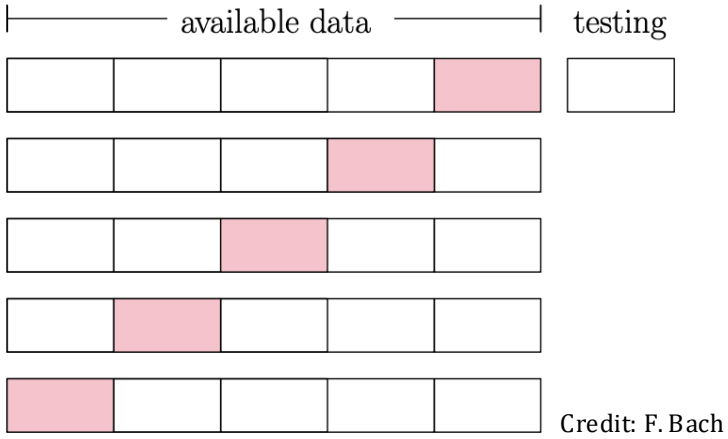
A word about hyperparameters

- One **hyperparameter**: the regularisation parameter λ

Hyperparameter: parameter that is **not learned** during the optimisation.
Examples include **regularisation parameters**, **depth of trees**, **learning rate** in optimisation.

- For the regularisation parameter: a value that is too large introduces a large bias, while if too small and close to zero, we do not solve the overfitting issue
- Usually, we use **grid search** to find the hyperparameter performing best on a third dataset: the validation set or **cross-validation**

Cross-validation: Split the available data into k folds and train the model k times changing the chunk of validation set. At the end, average the obtained errors.



A one slide-view of regularization as a prior

- Let us consider that $y^{(i)} = f_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}) + \epsilon^{(i)}$, meaning our model is correct up to an error with $\epsilon^{(i)}$ that we assume i.i.d.
- In this, case we can write the **likelihood** of the data as

$$p(\boldsymbol{Y}|\boldsymbol{X}, \boldsymbol{\theta}) = \prod_i p(y^{(i)}|\boldsymbol{x}^{(i)}, \boldsymbol{\theta})$$

- Using the **Bayes theorem**, we can express the posterior probability distribution of the parameters given the dataset as

$$p(\boldsymbol{\theta}|\boldsymbol{X}) \propto p(\boldsymbol{Y}|\boldsymbol{X}, \boldsymbol{\theta})p(\boldsymbol{\theta})$$

- Assuming a Gaussian likelihood $\epsilon^{(i)} \sim \mathcal{N}(0, \sigma_\epsilon^2)$, $\theta_i \sim \mathcal{N}(0, \sigma_\theta^2)$, and taking the log of this expression yields

$$\log p(\boldsymbol{\theta}|\boldsymbol{X}) = -\frac{1}{2\sigma_\epsilon^2}\sum_i \left(y^{(i)} - \hat{y}^{(i)}\right)^2 - \frac{1}{2\sigma_\theta^2}\sum_i \theta_i^2 + \text{ cst.}$$

- Finally,

$$\max_\theta \log p(\boldsymbol{\theta}|\boldsymbol{X}) \Leftrightarrow \min_\theta \sum_i \left(y^{(i)} - \hat{y}^{(i)}\right)^2 + \lambda \|\boldsymbol{\theta}\|^2, \qquad \text{with } \lambda = \sigma_\epsilon^2/\sigma_\theta^2$$

The maximum a posteriori estimator under Gaussian likelihood and Gaussian prior is equivalent to minimizing the square loss with an L_2 penalty. The **prior plays the role of the regularization**, and the **square-loss comes from the Gaussian error** assumption.