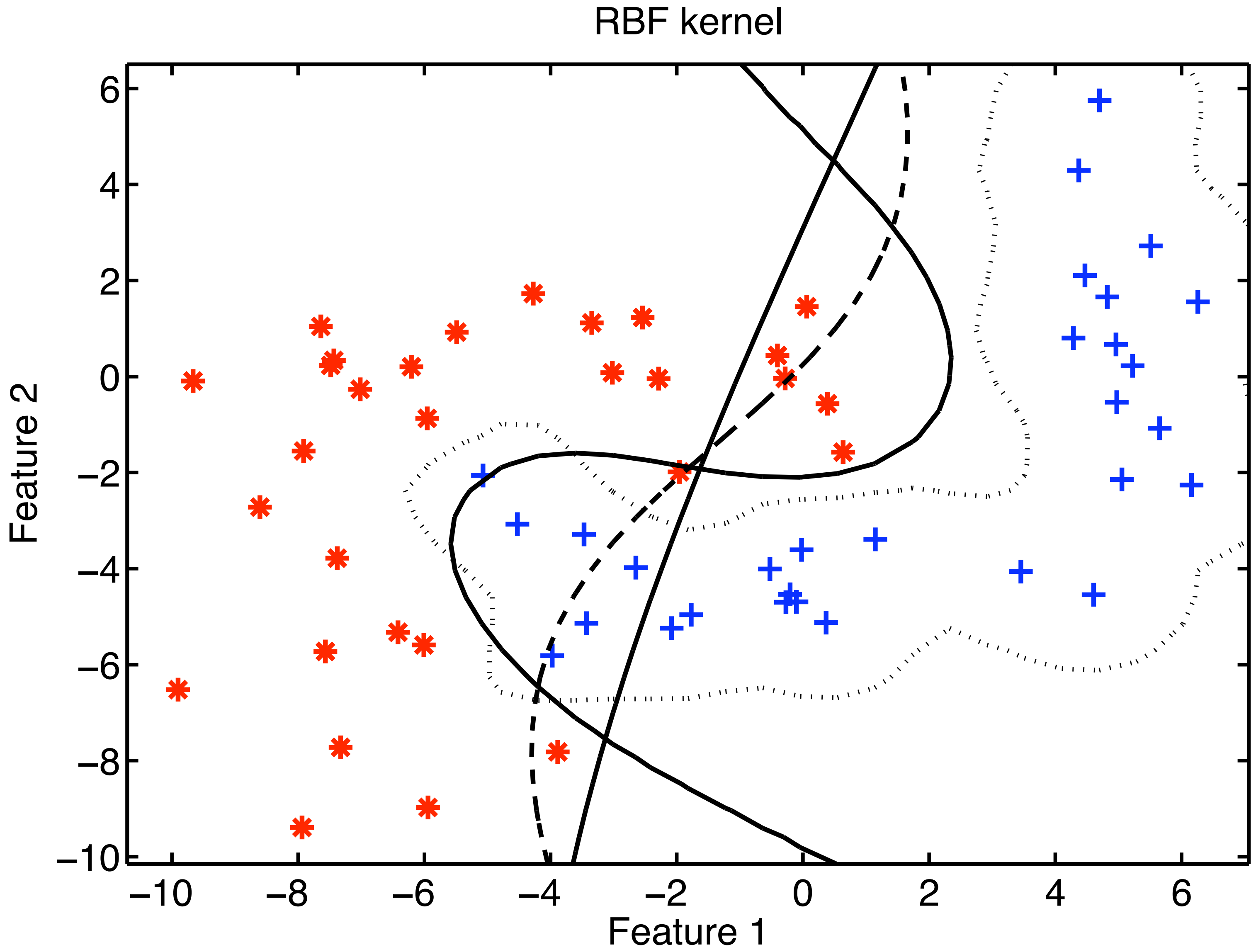
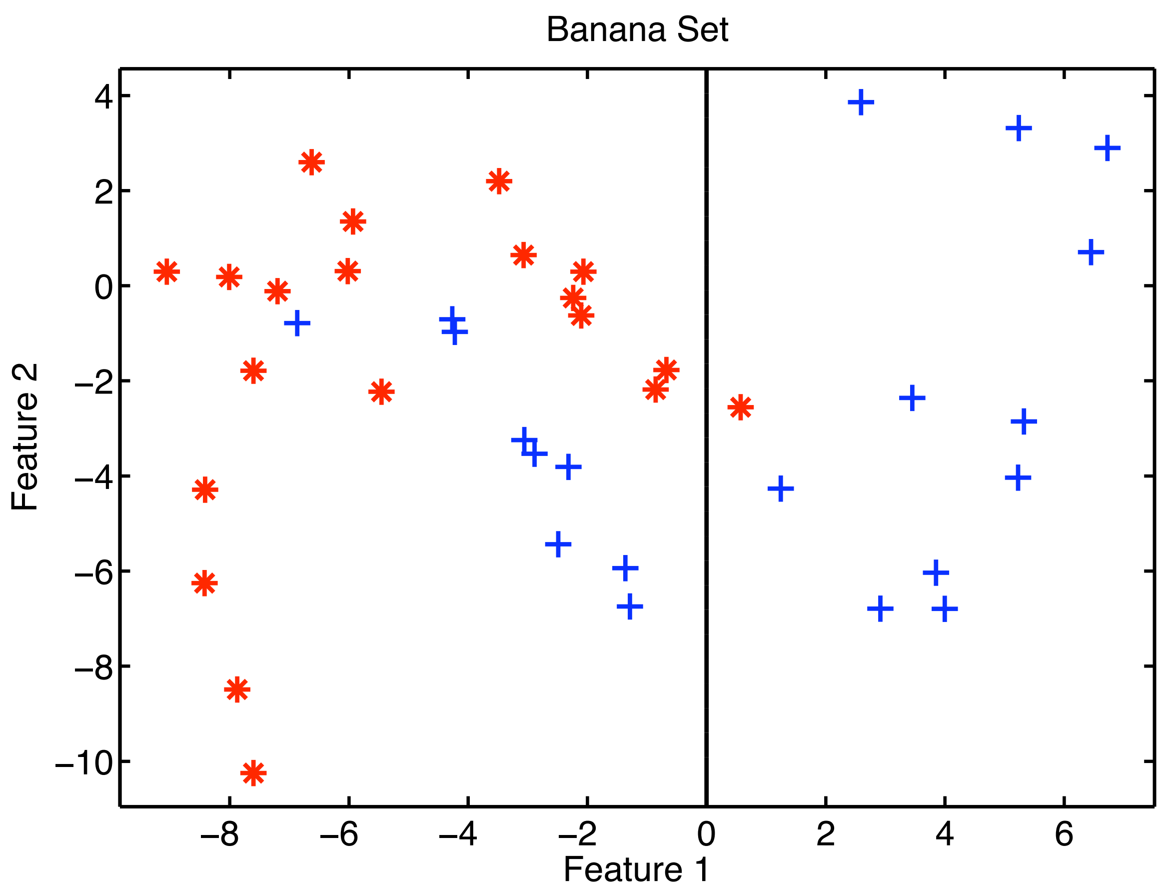
Complexity and Support Vector Classifiers



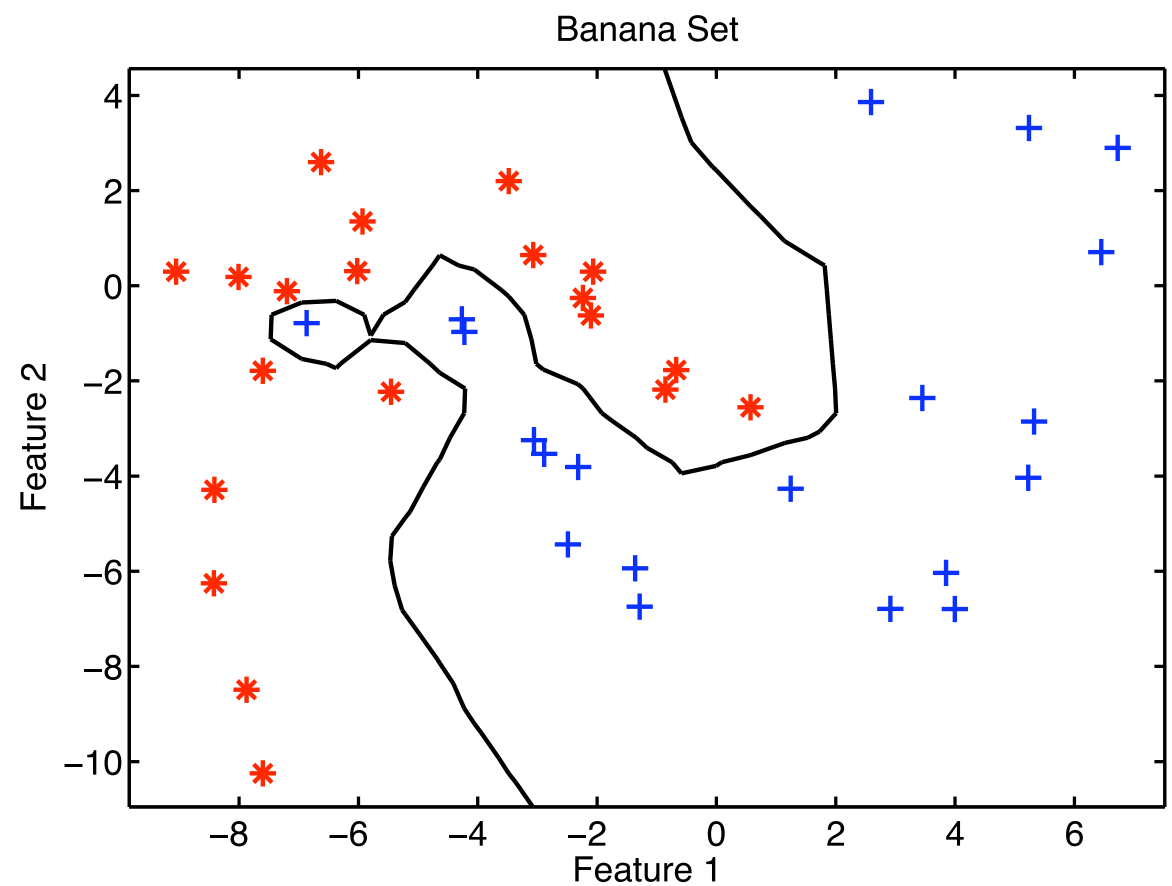
David M.J. Tax

TU Delft

Complexity?

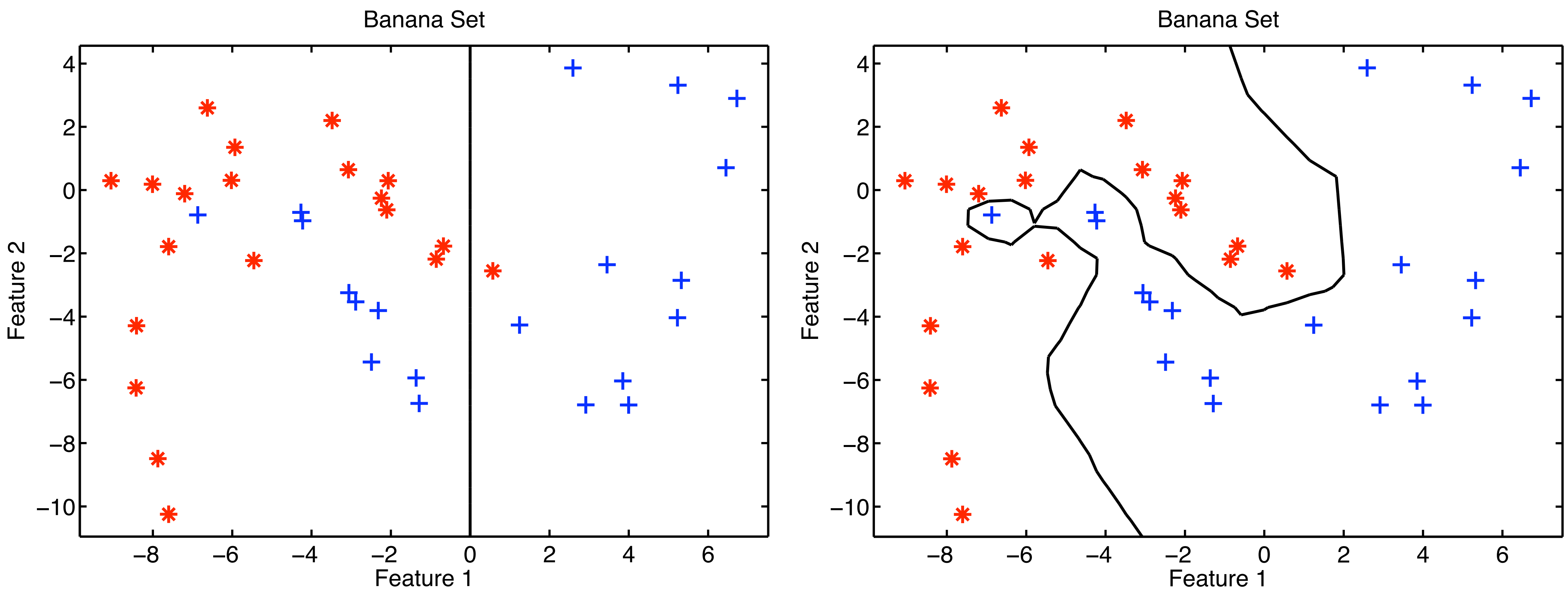


Banana Set
Feature 2
Feature 1
4
2
0
−2
−4
−6
−8
−10
−8
−6
−4
−2
0
2
4
6



Banana Set
Feature 2
Feature 1
4
2
0
−2
−4
−6
−8
−10
−8
−6
−4
−2
0
2
4
6

Complexity?



The complexity of a classifier indicates the ability to fit to any data distribution

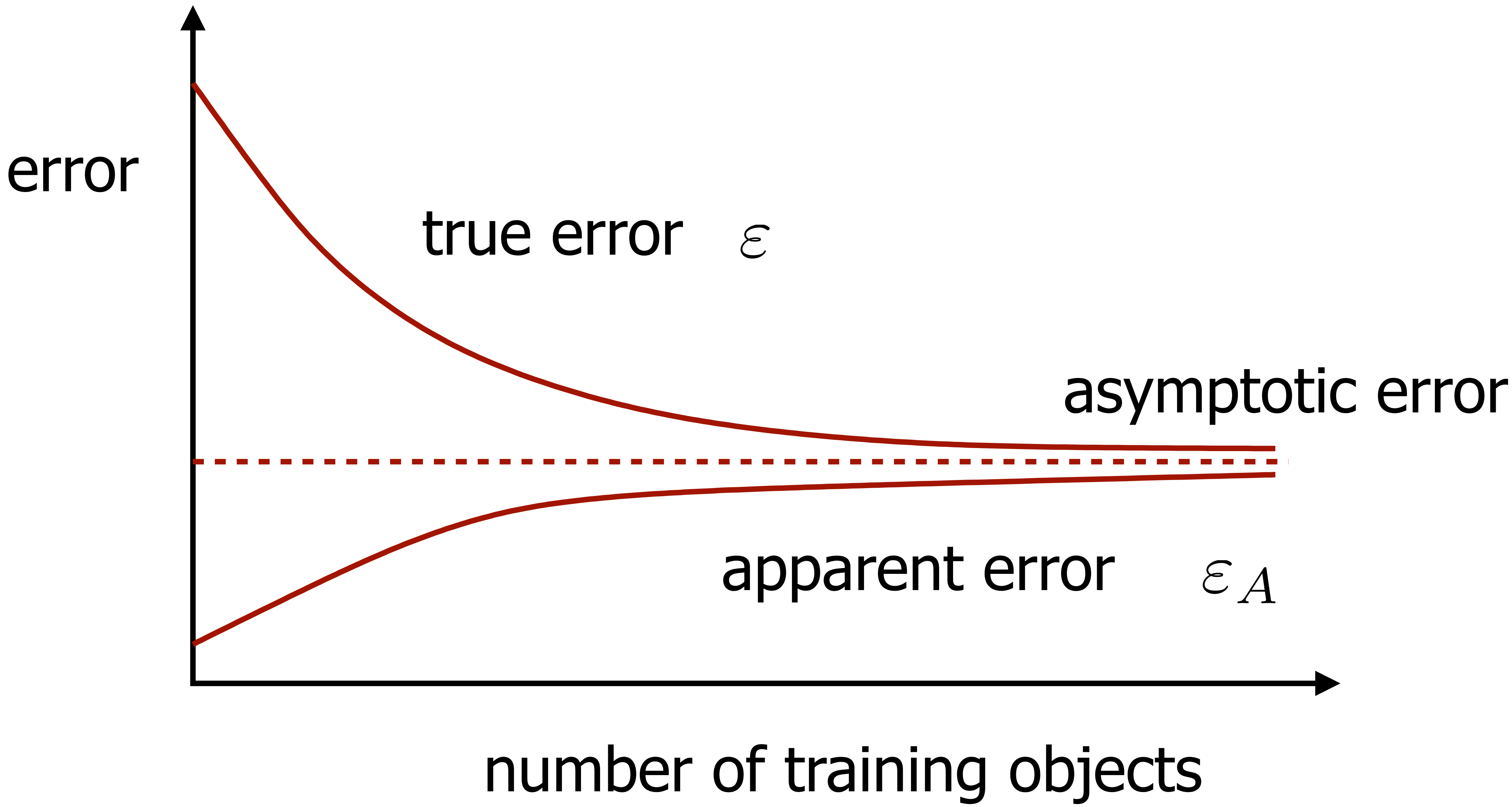
TU Delft

Which complexity to choose?

- A simple classifier fits to only a few specific data distributions
- A complex classifier fits to almost all data distributions
- So, we should always use a complex classifier! (?)

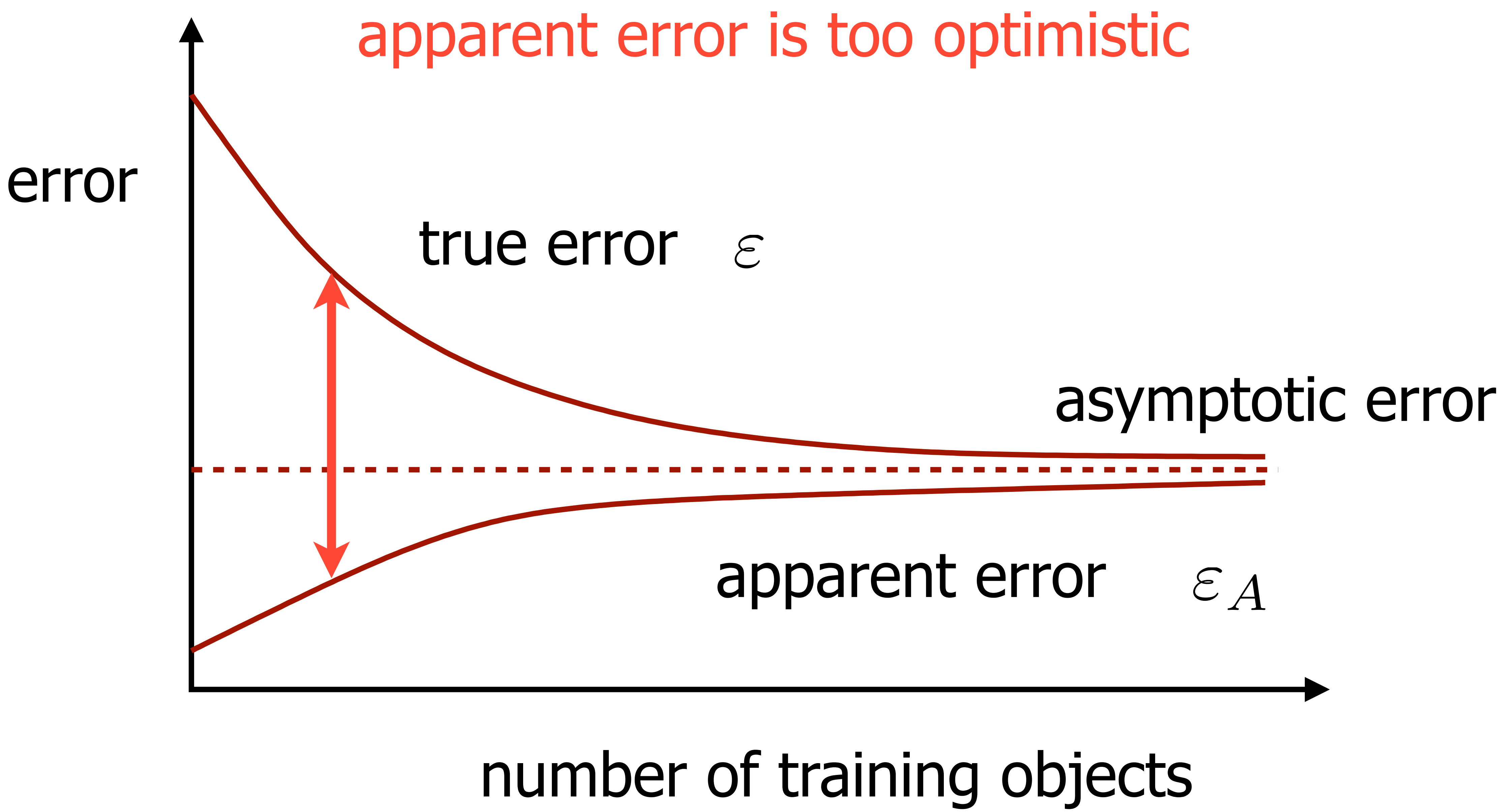
TU Delft

The learning curve

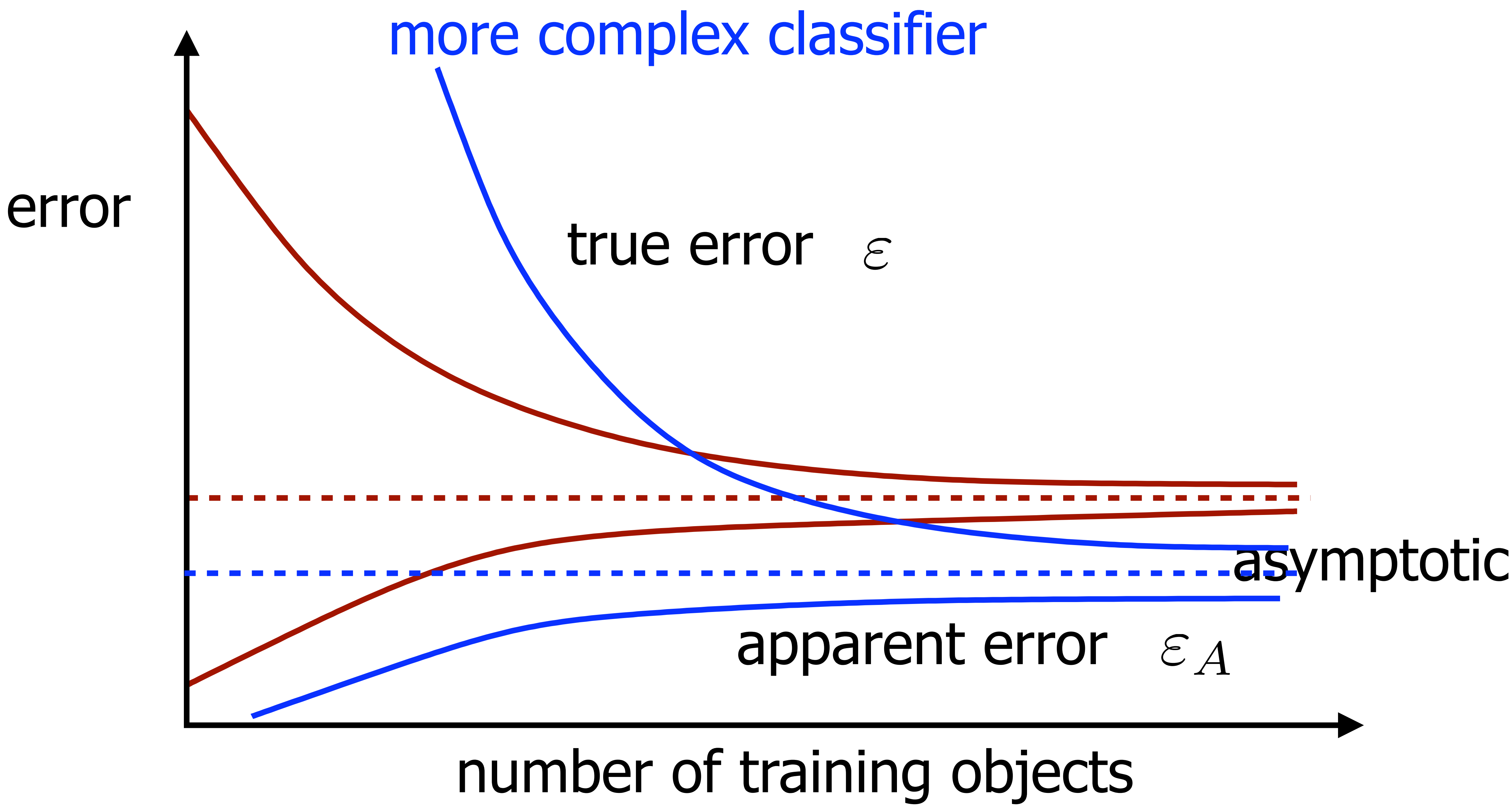


TU Delft

The learning curve

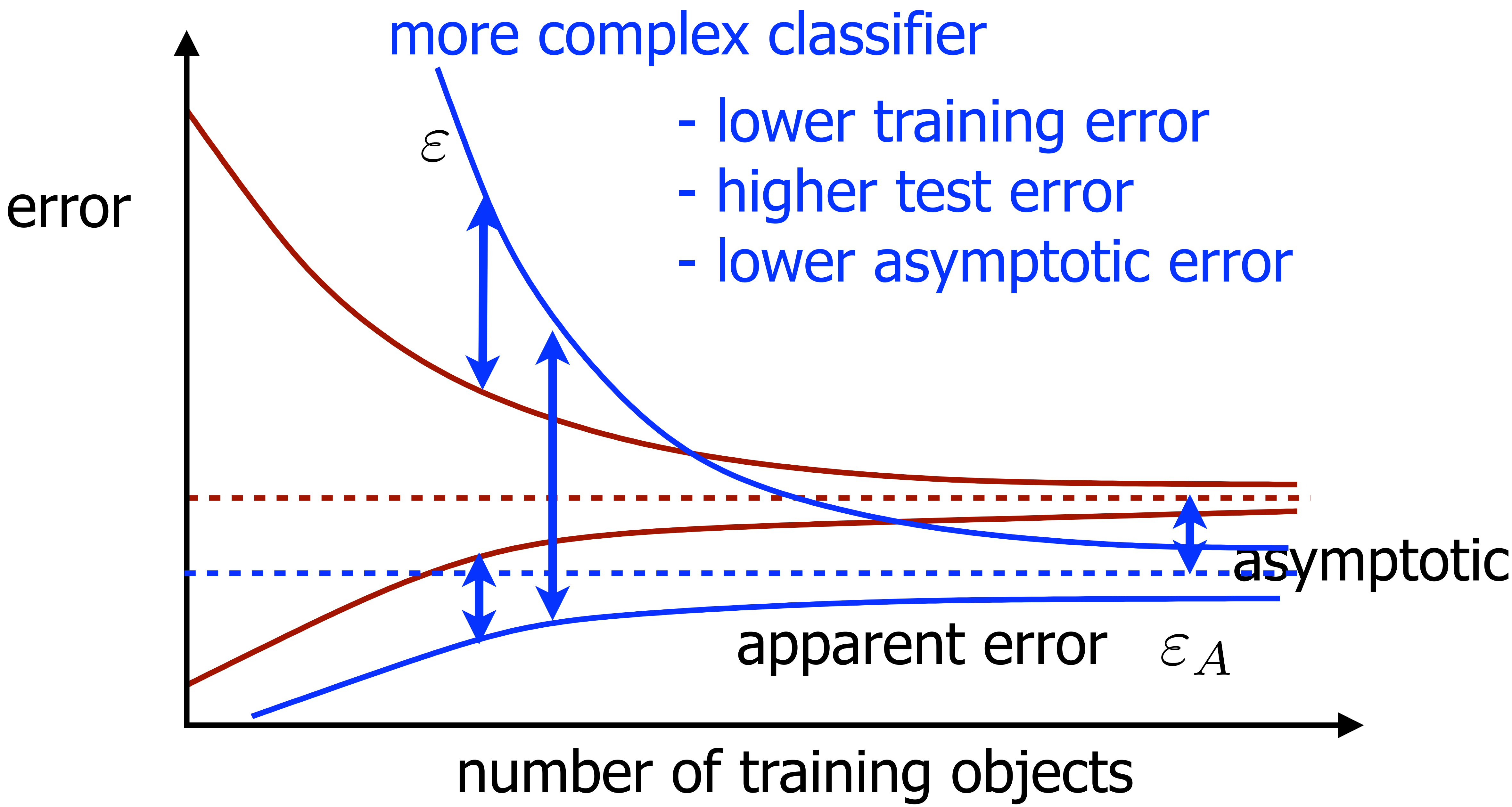


The learning curve



TU Delft

The learning curve



So, complex or not?

- Complex classifiers are good when you have sufficient number of training objects
- When a small number of training objects is available, you overtrain
- Use a simple classifier when you don’t have many training examples

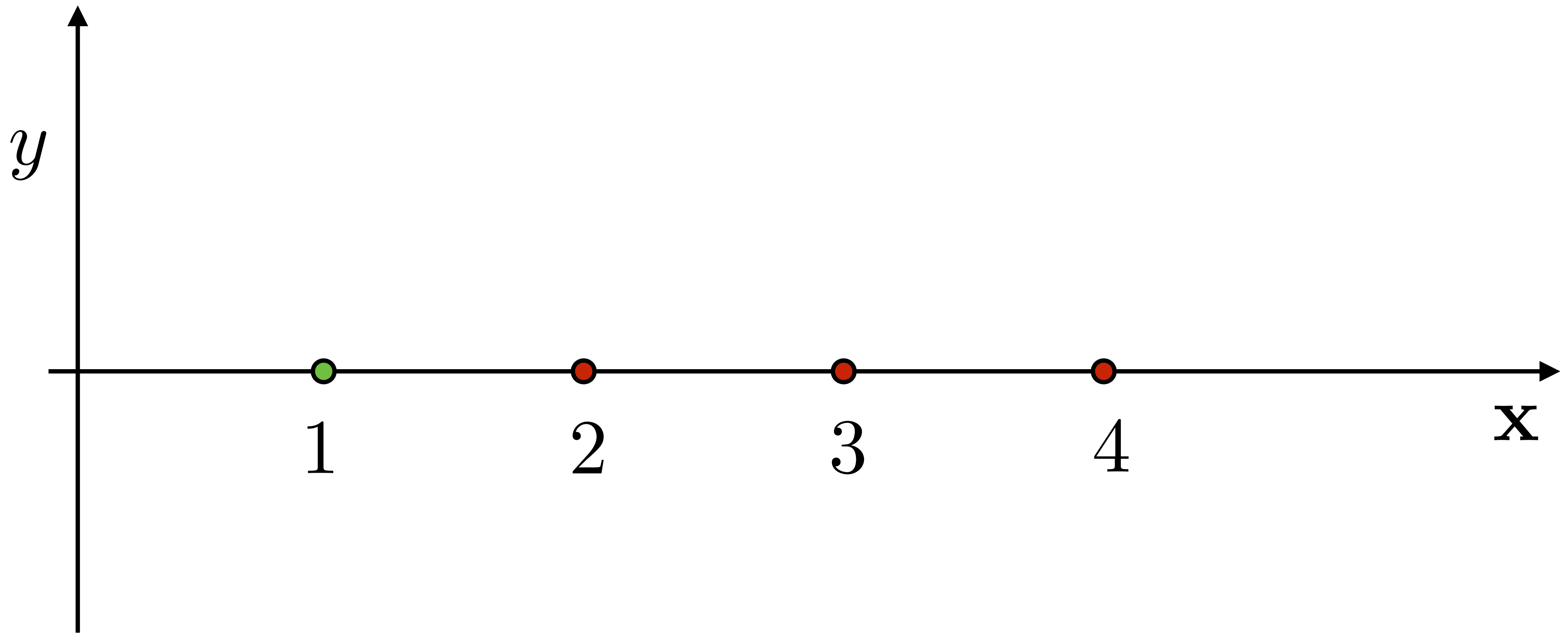
Choose the complexity according to the available training set size

TUDelft

Use a linear classifier for:

$$y = \mathrm{sign}(\mathbf{w}^T\mathbf{x} + w_0)$$

- What are the optimal **w** and w_0?



Last time: linear classifiers

- Perceptron

$$J(\mathbf{w}) = \sum_{\text{misclassified}\,\mathbf{x}_i} -y_i \mathbf{w}^T \mathbf{x}_i$$

- Nearest mean

$$J(\boldsymbol{\mu}) = \sum_i \frac{(\mathbf{x}_i - \mu_{y_i})^2}{2\sigma^2}$$

- Least squares

$$J(\mathbf{w}) = E[|y - \mathbf{w}^T \mathbf{x}|^2]$$

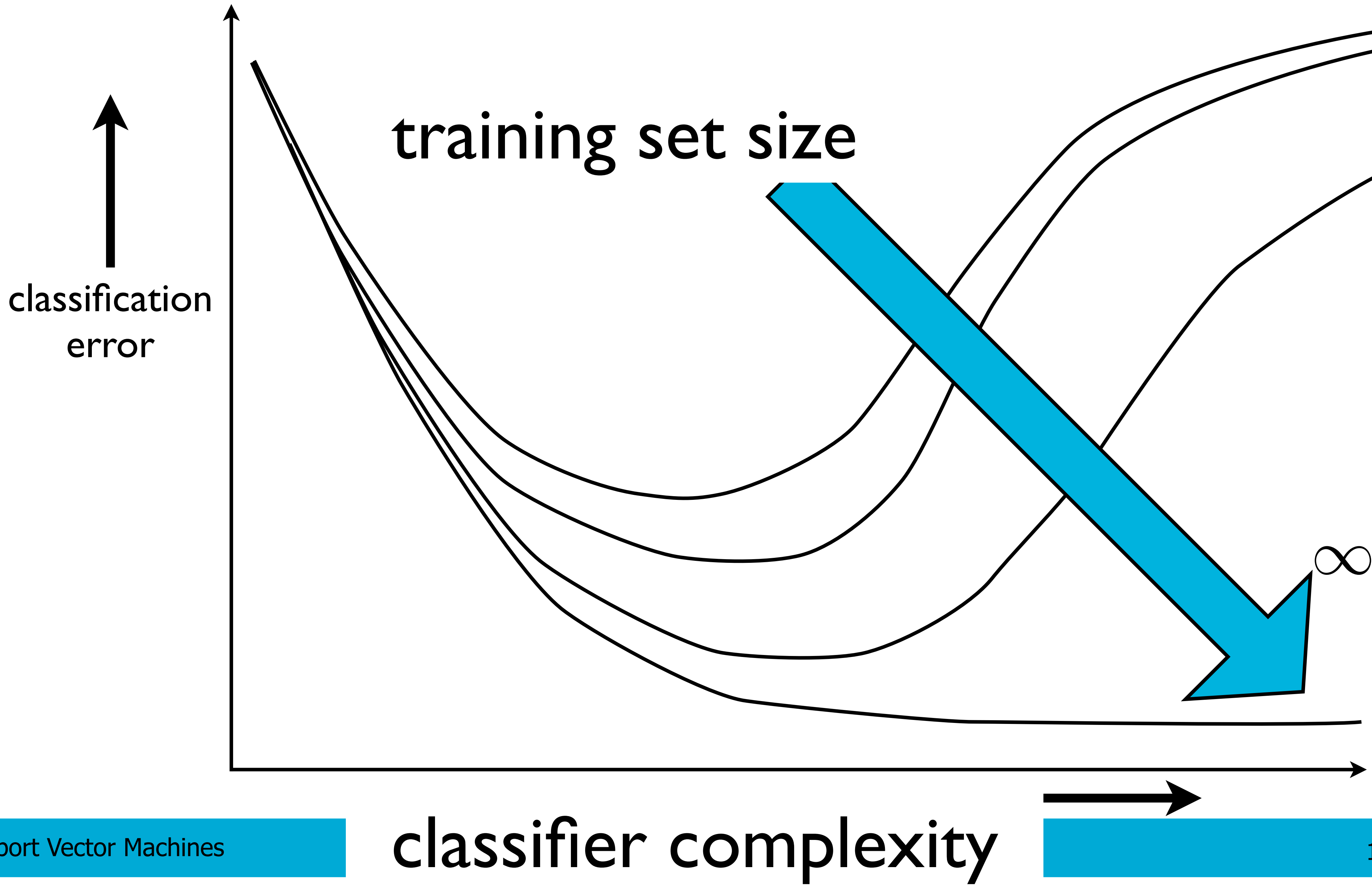
- Logistic classifier

$$L = \prod_{i=1}^{n_1} p(\mathbf{x}_i^{(1)}|\omega_1) \prod_{i=1}^{n_2} p(\mathbf{x}_i^{(2)}|\omega_2)$$

TU Delft

Bias-Variance and the Feature curve

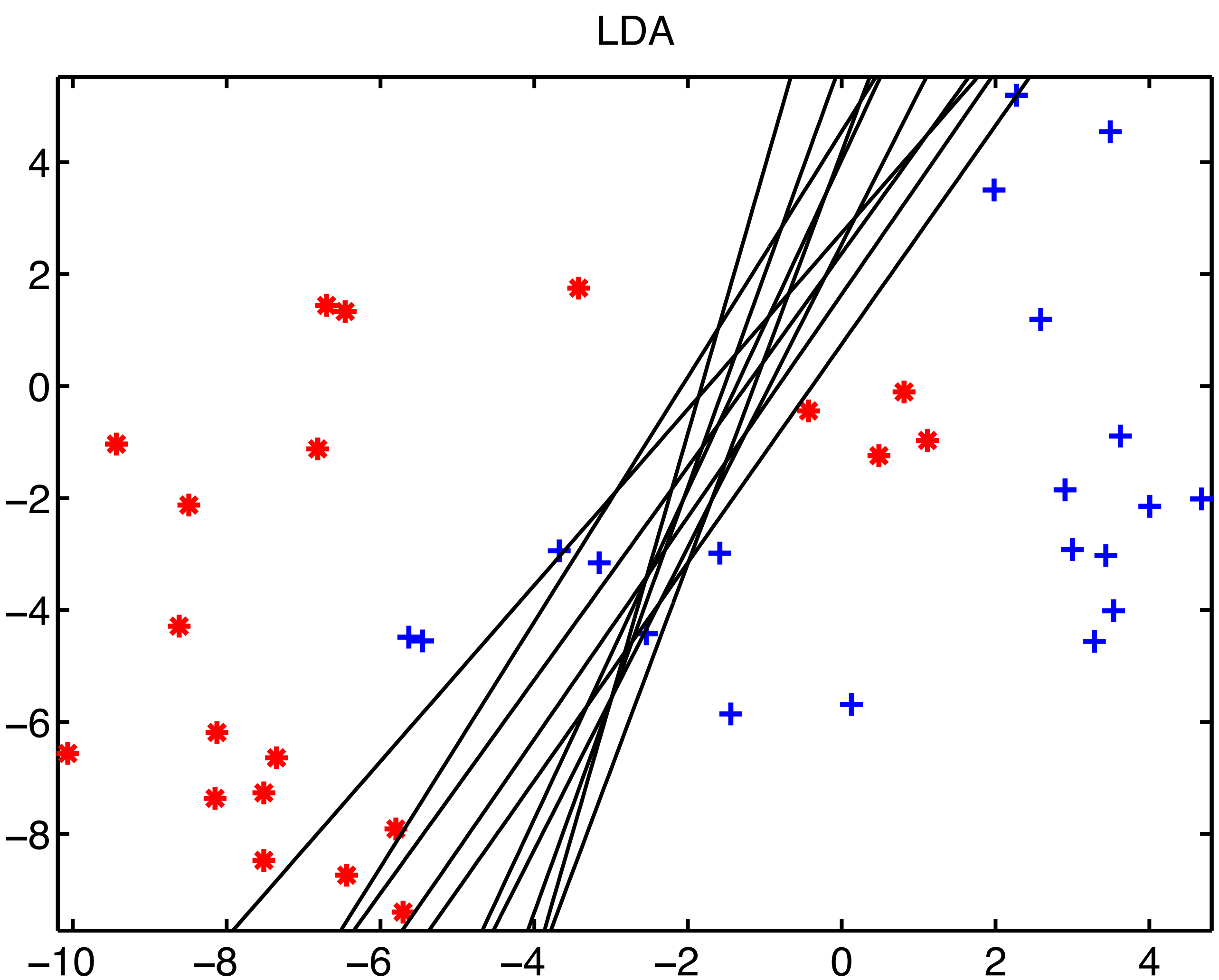
Different classifiers: may be better on different problems

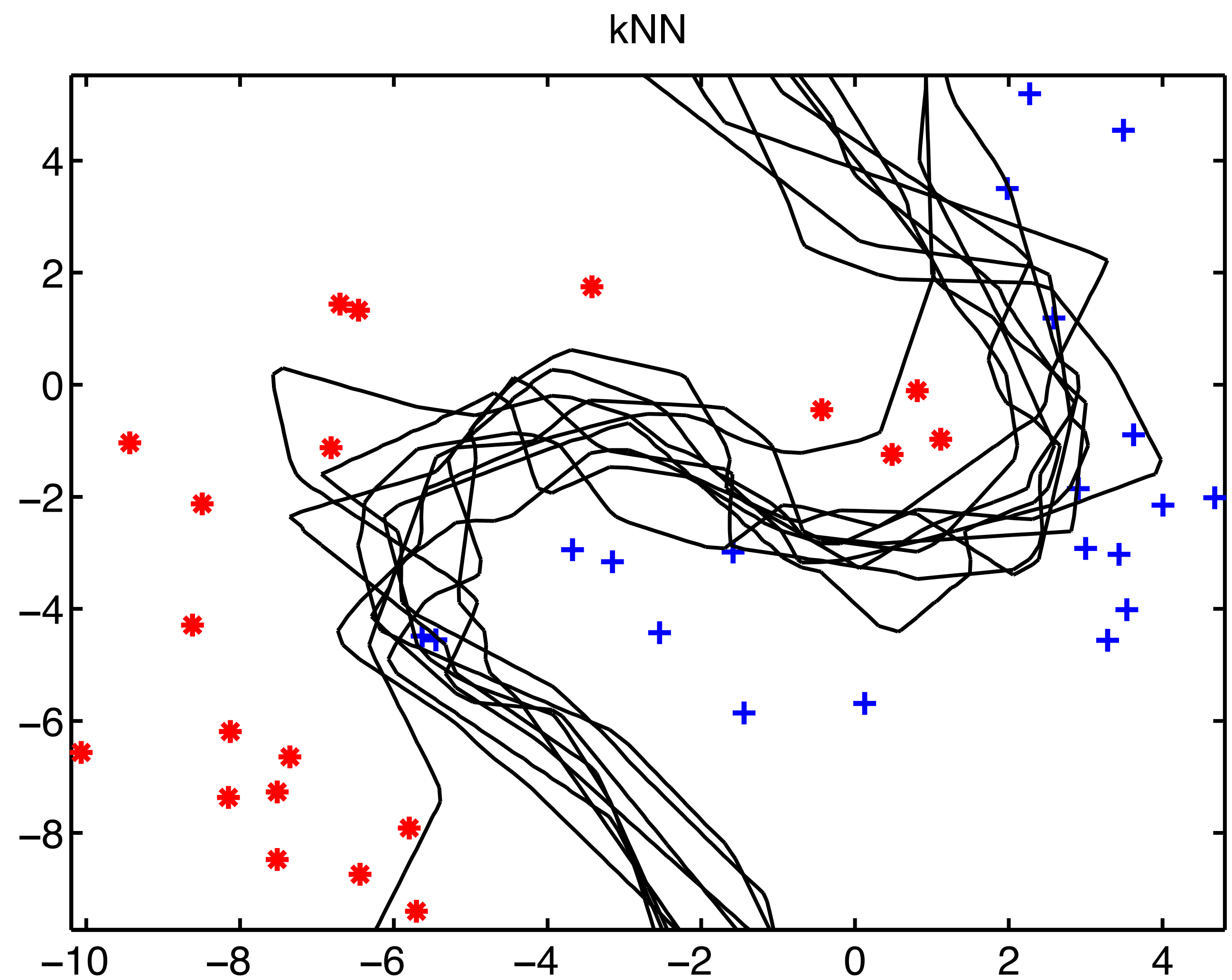


Bias-variance dilemma

- Compare LDA with kNN:

low variance, high bias



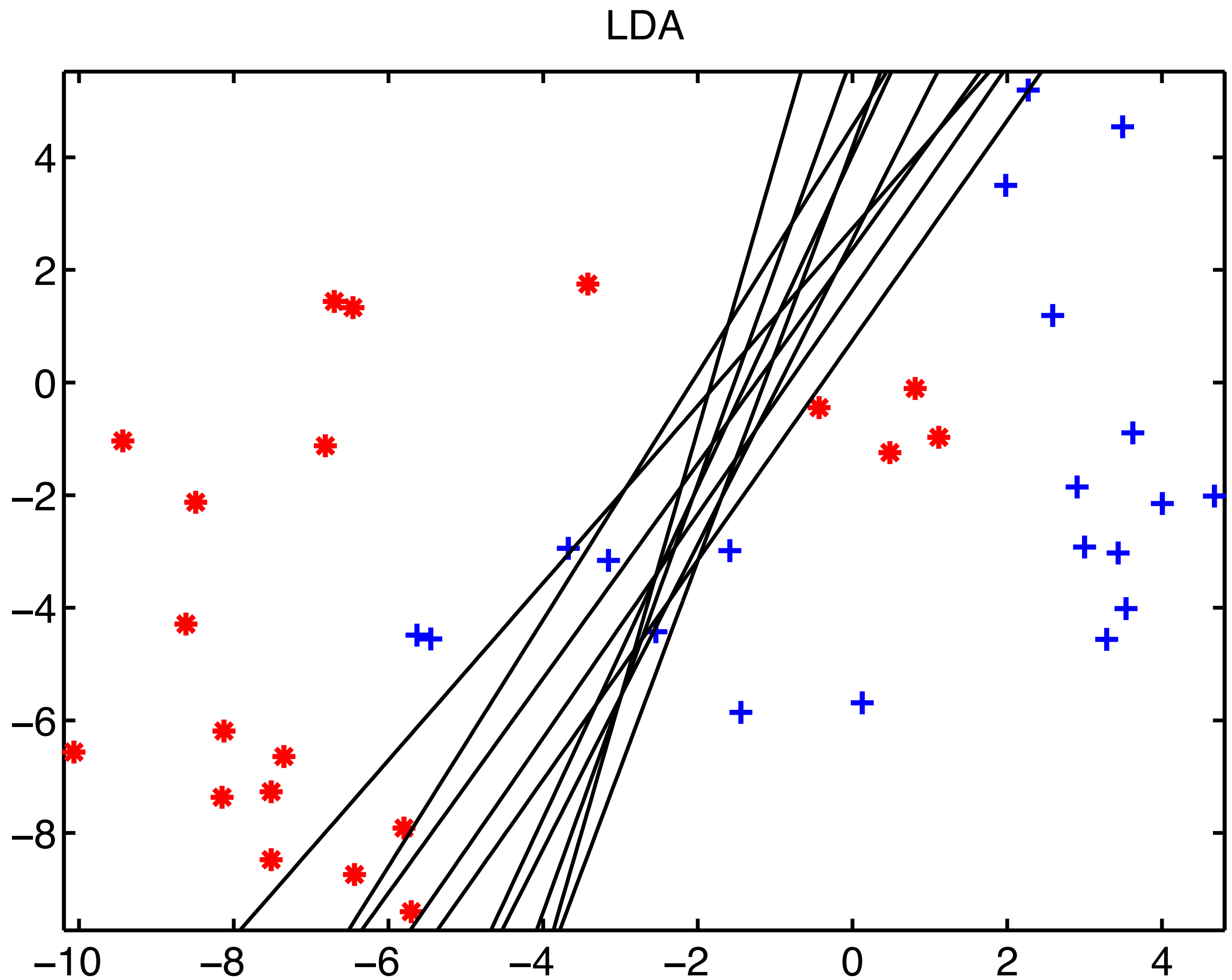


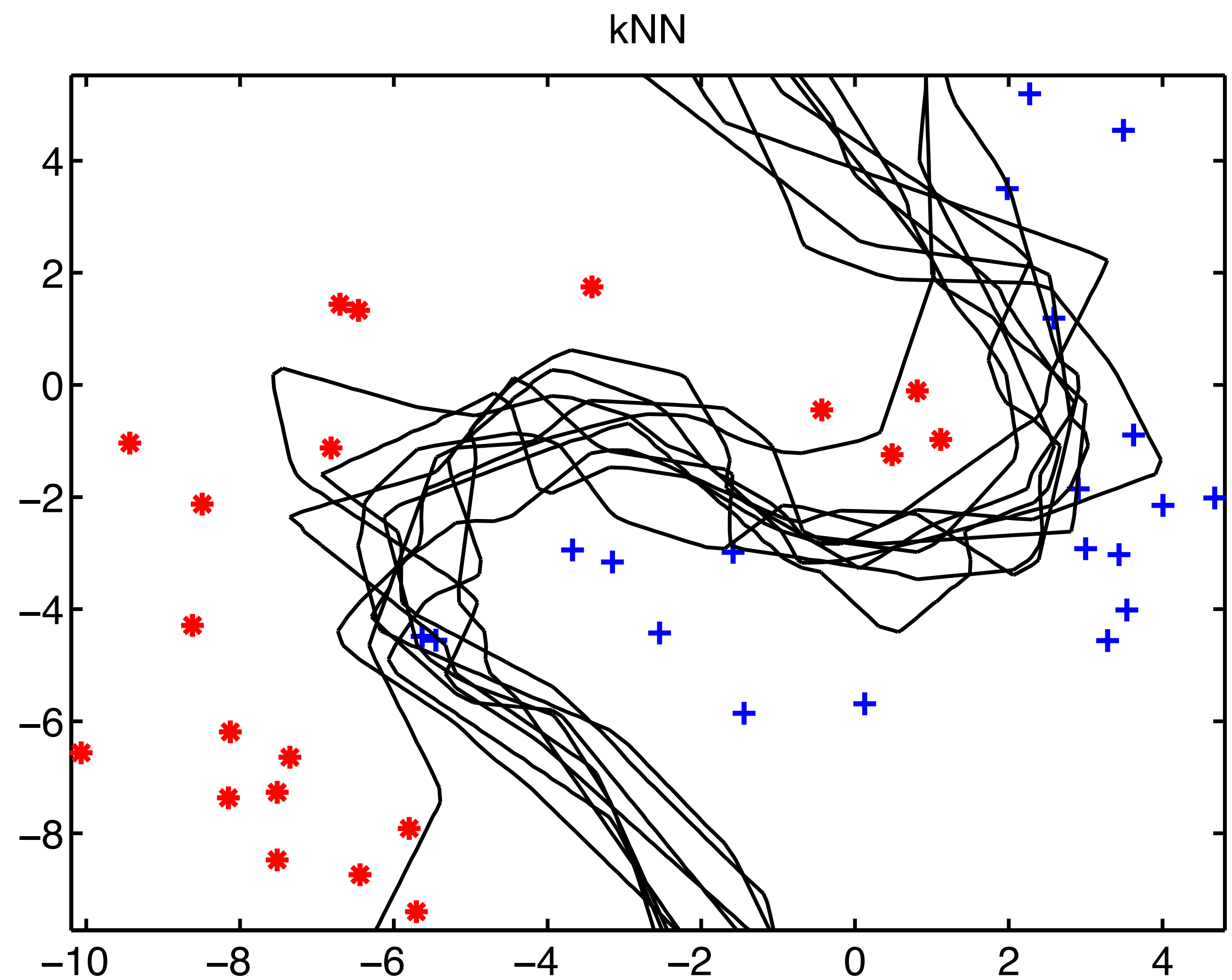
high variance, low bias

Bias-variance dilemma

- How to measure/influence the complexity of a classifier?

low variance, high bias





high variance, low bias

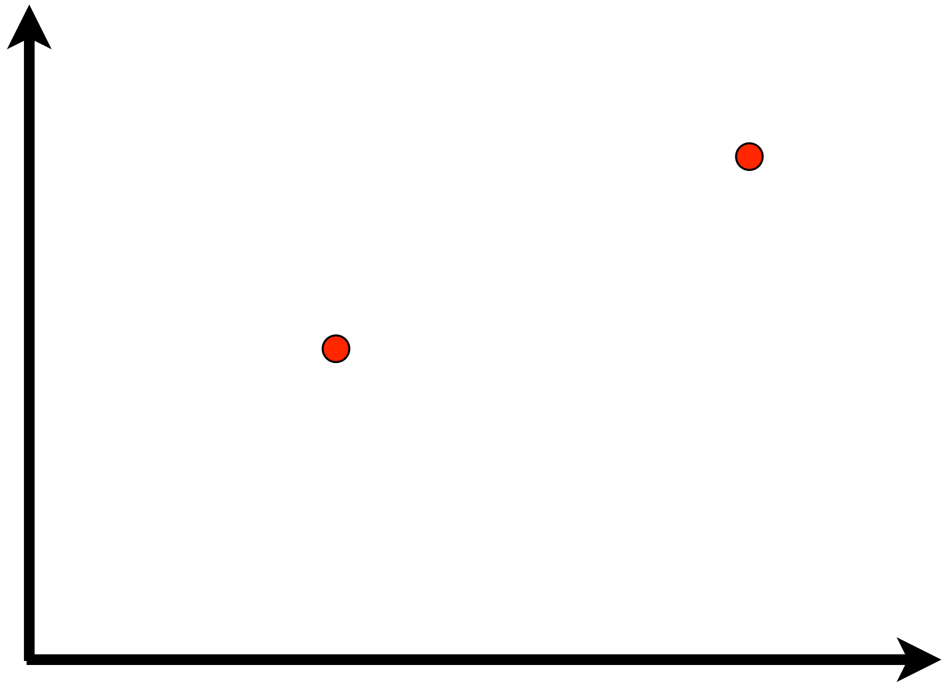
TU Delft
Delft University of Technology

Contents

- What is complexity?
- How to characterise complexity?
- How to adapt the complexity?
- Measuring complexity: VC-dimension
- Support vector classifiers
  - Constrained optimisation
  - Class overlap
  - Kernel trick
- Examples
- Conclusions

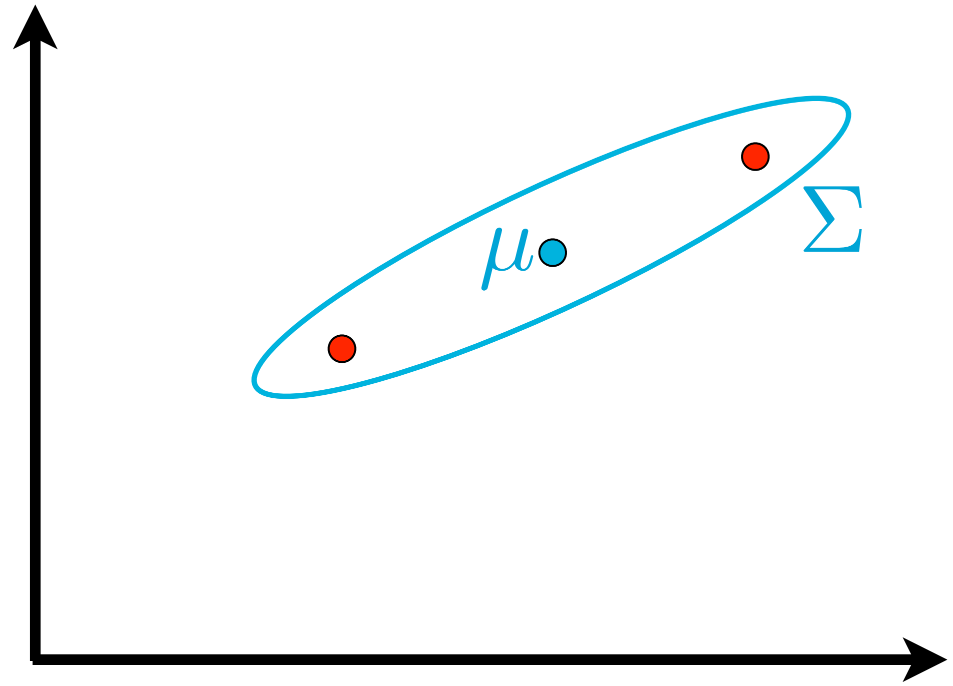
Regularization Normal-based classif.

- Estimate a Gaussian in a 2D feature space:



- Estimate mean and covariance matrix μ Σ

Regularization Normal-based classif.

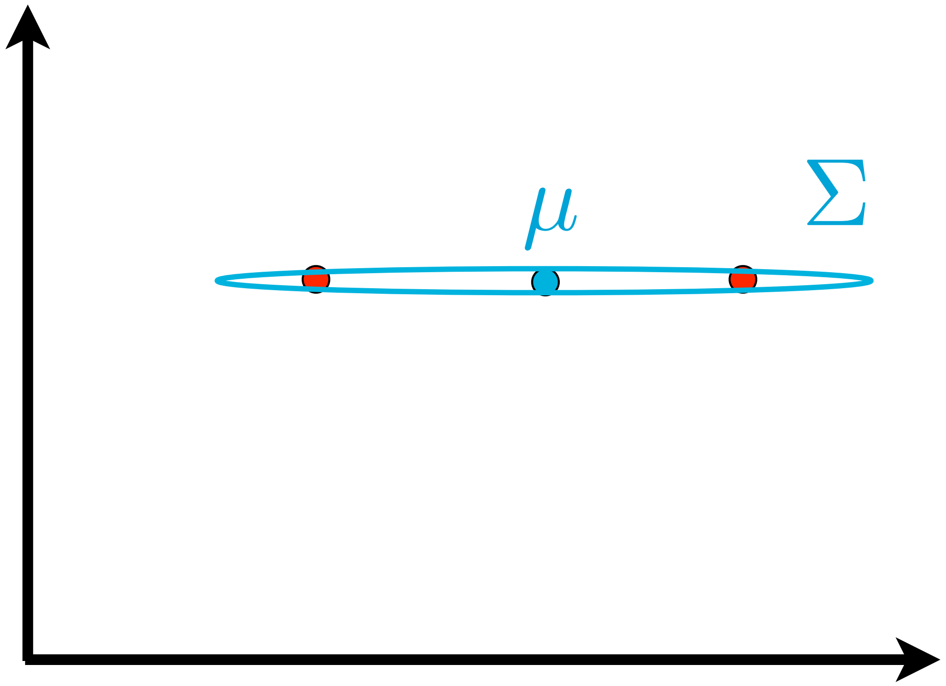


μ Σ

a Gaussian in a 2D feature space:

Regularization Normal-based classif.

- Estimate a Gaussian on 2 points:



$$\hat{\Sigma} = \begin{bmatrix} 5 & 0 \\ 0 & 0 \end{bmatrix}$$

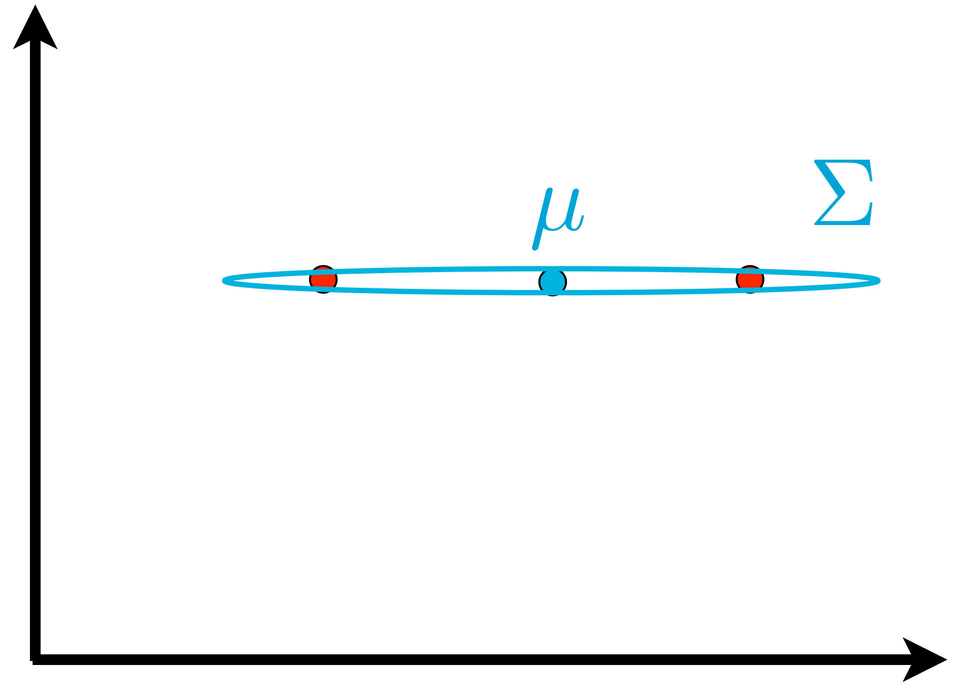
- For a normal-based classifier I need the inverse:

$$(\mathbf{x}-\mu)^T \Sigma^{-1} (\mathbf{x}-\mu)$$

- That is: $\hat{\Sigma}^{-1} = \begin{bmatrix} \frac{1}{5} & 0 \\ 0 & \frac{1}{0} \end{bmatrix}$

Regularization Normal-based classif.

- Estimate a Gaussian on 2 points:



$$\hat{\Sigma} = \begin{bmatrix} 5 & 0 \\ 0 & 0 \end{bmatrix}$$

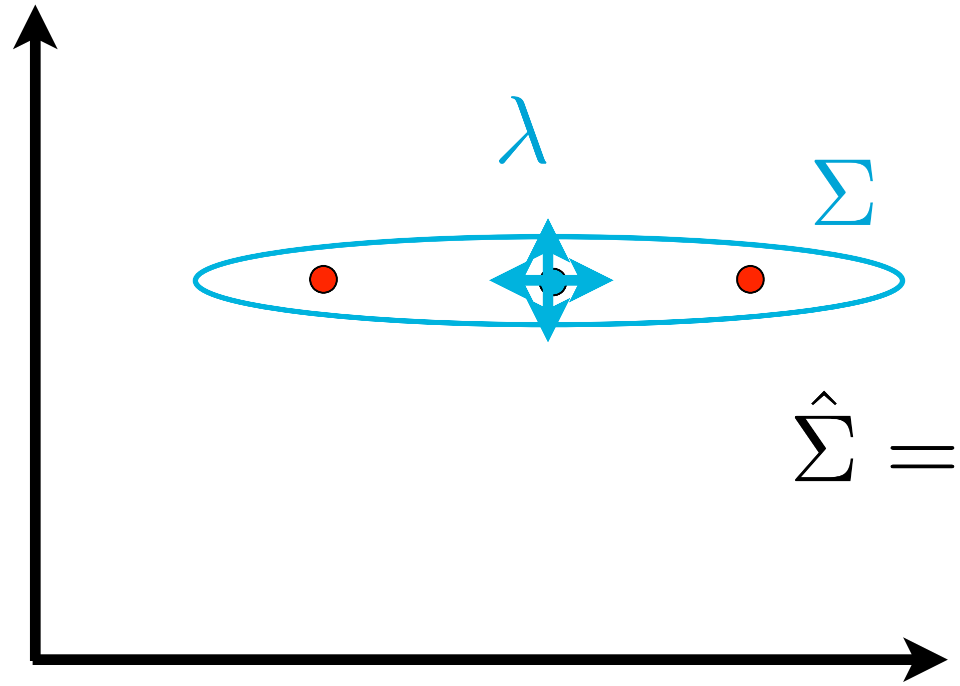
- For a normal-based classifier I need the inverse:

$$(\mathbf{x}-\mu)^T \Sigma^{-1} (\mathbf{x}-\mu)$$

- That is: $\hat{\Sigma}^{-1} = \begin{bmatrix} \frac{1}{5} & 0 \\ 0 & \frac{1}{0} \end{bmatrix}$

Regularization Normal-based classif.

- Add a bit of artificial noise to the Gaussian



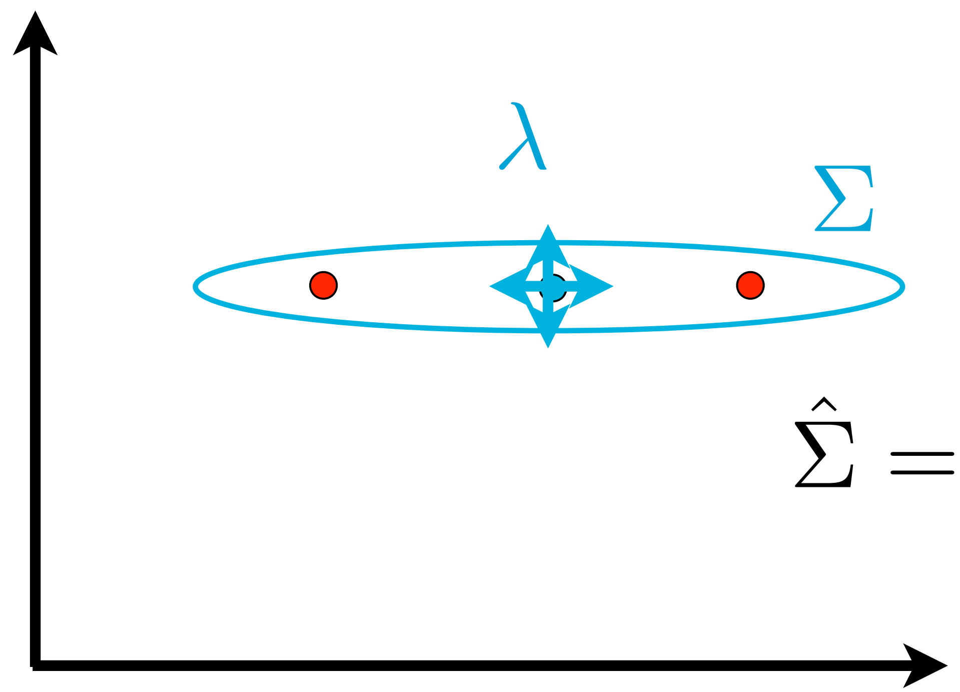
$$\hat{\Sigma} = \begin{bmatrix} 5+\lambda & 0 \\ 0 & 0+\lambda \end{bmatrix}$$

- Now the inverse is well-defined:

- That is: $\hat{\Sigma}^{-1} = \begin{bmatrix} \frac{1}{5+\lambda} & 0 \\ 0 & \frac{1}{0+\lambda} \end{bmatrix}$

Regularization Normal-based classif.

- Add a bit of artificial noise to the Gaussian



$$\hat{\Sigma} = \begin{bmatrix} 5+\lambda & 0 \\ 0 & 0+\lambda \end{bmatrix}$$

- Now the inverse is well-defined:

- That is: $\hat{\Sigma}^{-1} = \begin{bmatrix} \frac{1}{5+\lambda} & 0 \\ 0 & \frac{1}{0+\lambda} \end{bmatrix}$

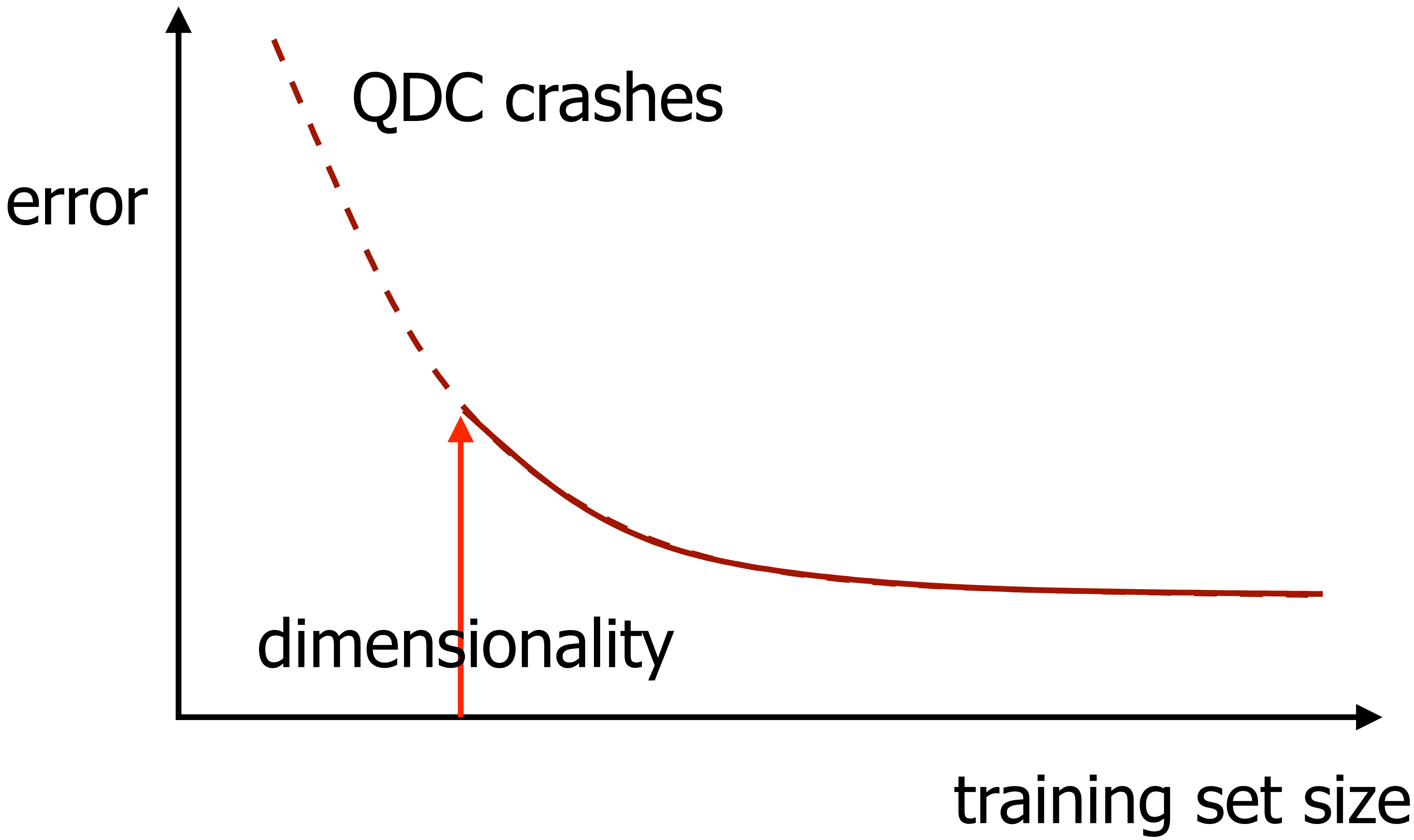
Quadratic classifier

$$g_i(\mathbf{x}) = -\frac{1}{2}\log(\det \Sigma_i) - \frac{1}{2}(\mathbf{x}-\mu_i)^T \Sigma_i^{-1}(\mathbf{x}-\mu_i) + \log(p(\omega_i))$$

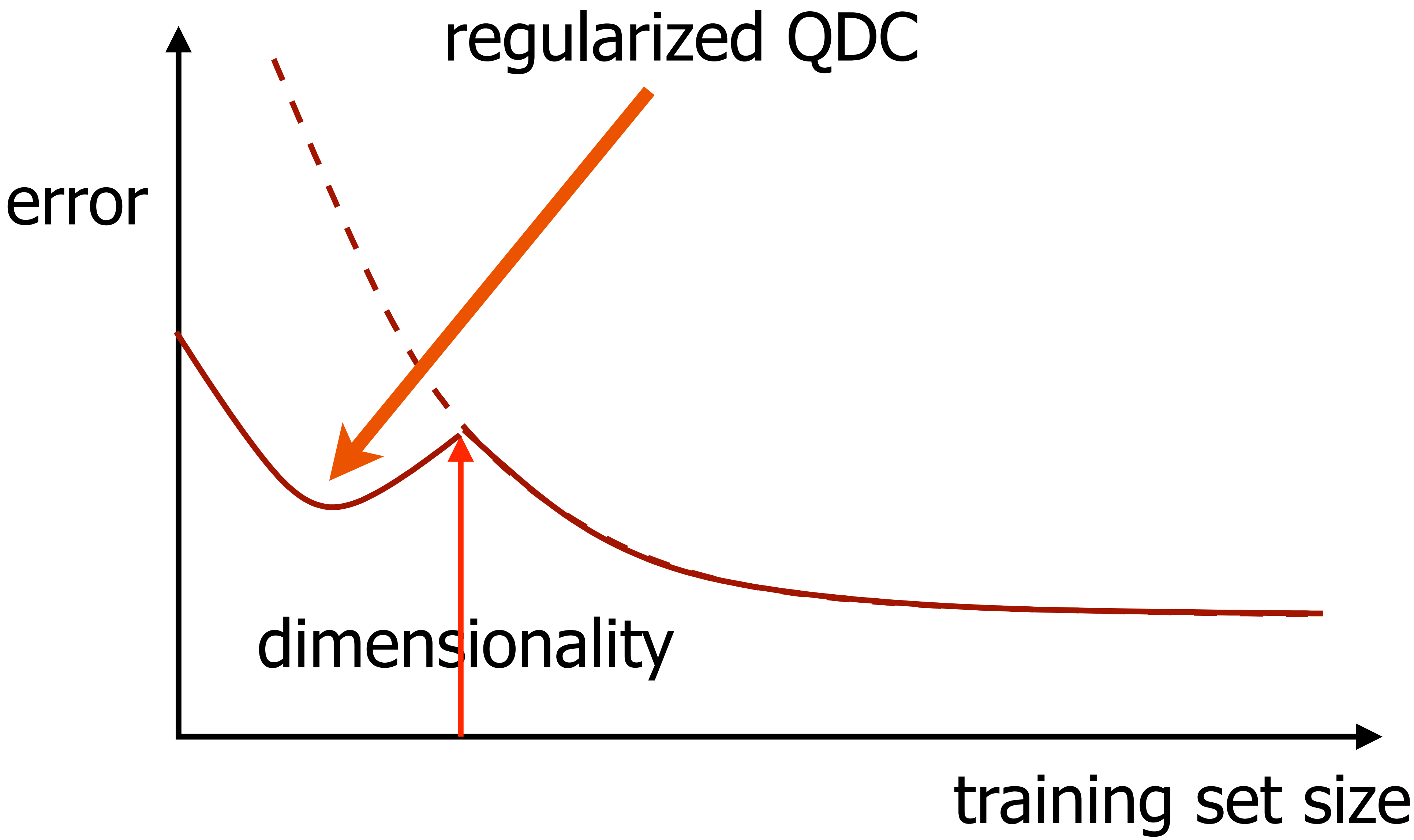
- When insufficient data is available (nr of obj's is smaller than dimensionality), the inverse covariance matrices are not defined: the classifier is not defined
- **Regularization** solves it: $\tilde{\Sigma}_i = \Sigma_i + \lambda \mathbb{I}$
- With very large regularisation $\lambda \to \infty$ (and equal class priors) you get: nearest mean classifier

TU Delft

Quadratic classifier



Quadratic classifier

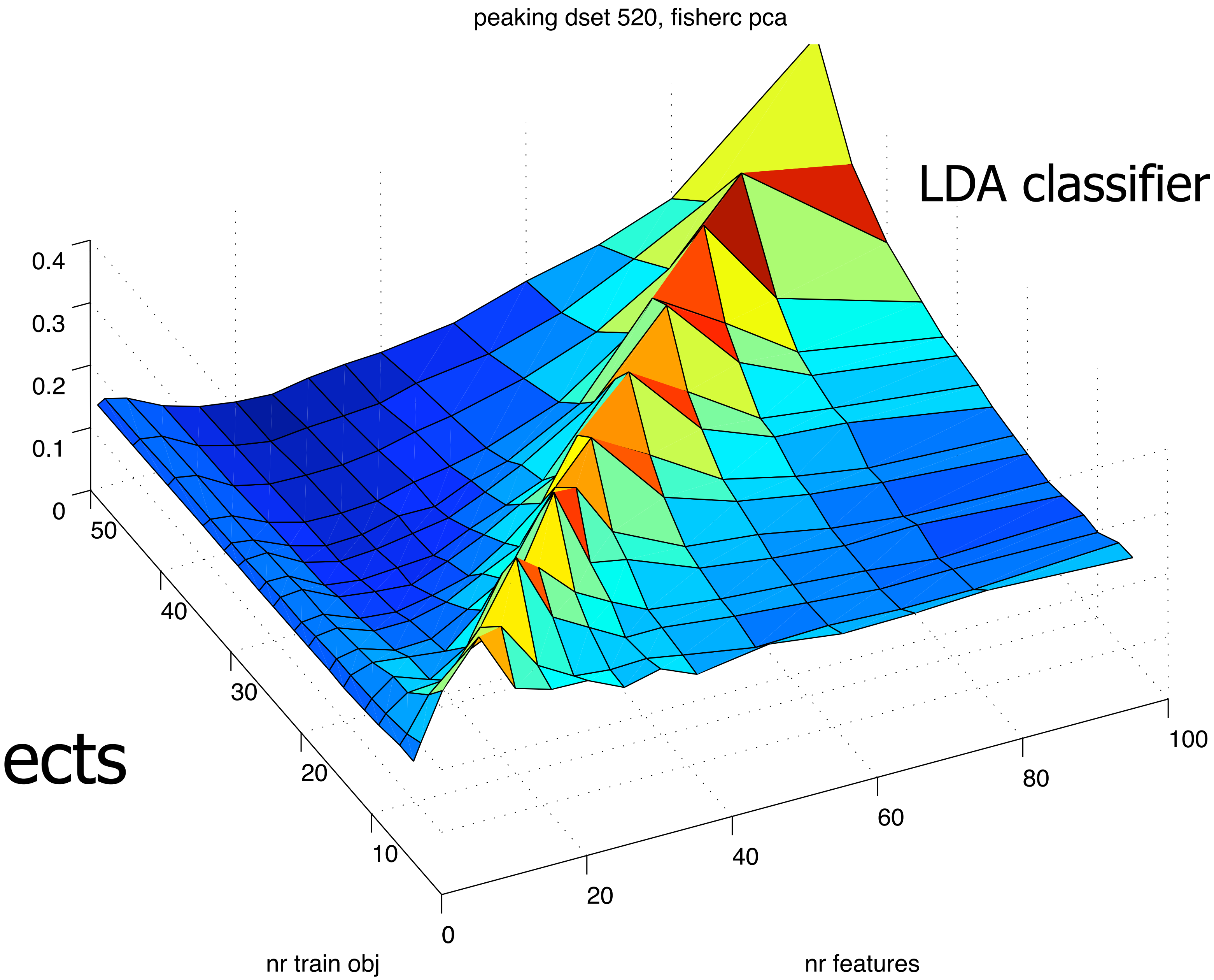


LDA classifier

$$f(\mathbf{x}) = (\mu_B - \mu_A)^T \hat{\Sigma}^{-1} \mathbf{x} + C$$

- The same phenomenon appears, but because the covariance matrix is averaged over two classes, the peak appears for small sample size
- Instead of regularization the *pseudo-inverse* of the covariance matrix is used.

Learning & feature curve



LDA classifier

number of training objects

number of features

(Concordia dataset, 4000 obj in 1024D)

TUDelft

Regularizer

- Optimizing on purely the training set may cause overfitting

$$\min_{\theta} \varepsilon_A(\theta)$$

training error

- Often it is wise to 'punish' unwanted (too flexible) solutions.
- Then need a regulariser $\Omega(\theta)$, that 'measures' this flexibility

$$\min_{\theta} \varepsilon_A(\theta) + \lambda\Omega(\theta)$$

regularization parameter regularizer

TU Delft

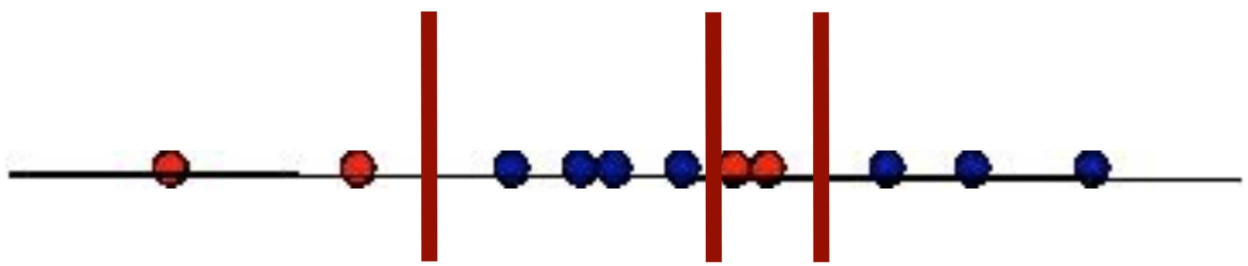
How to characterize complexity?

- How can we measure the complexity or flexibility of a classifier?
- An indication might be the number of free parameters?

- Take a 1D example:

1D classifier

- We can define N thresholds, giving it N parameters
- The more thresholds, the more complex our classifier becomes:

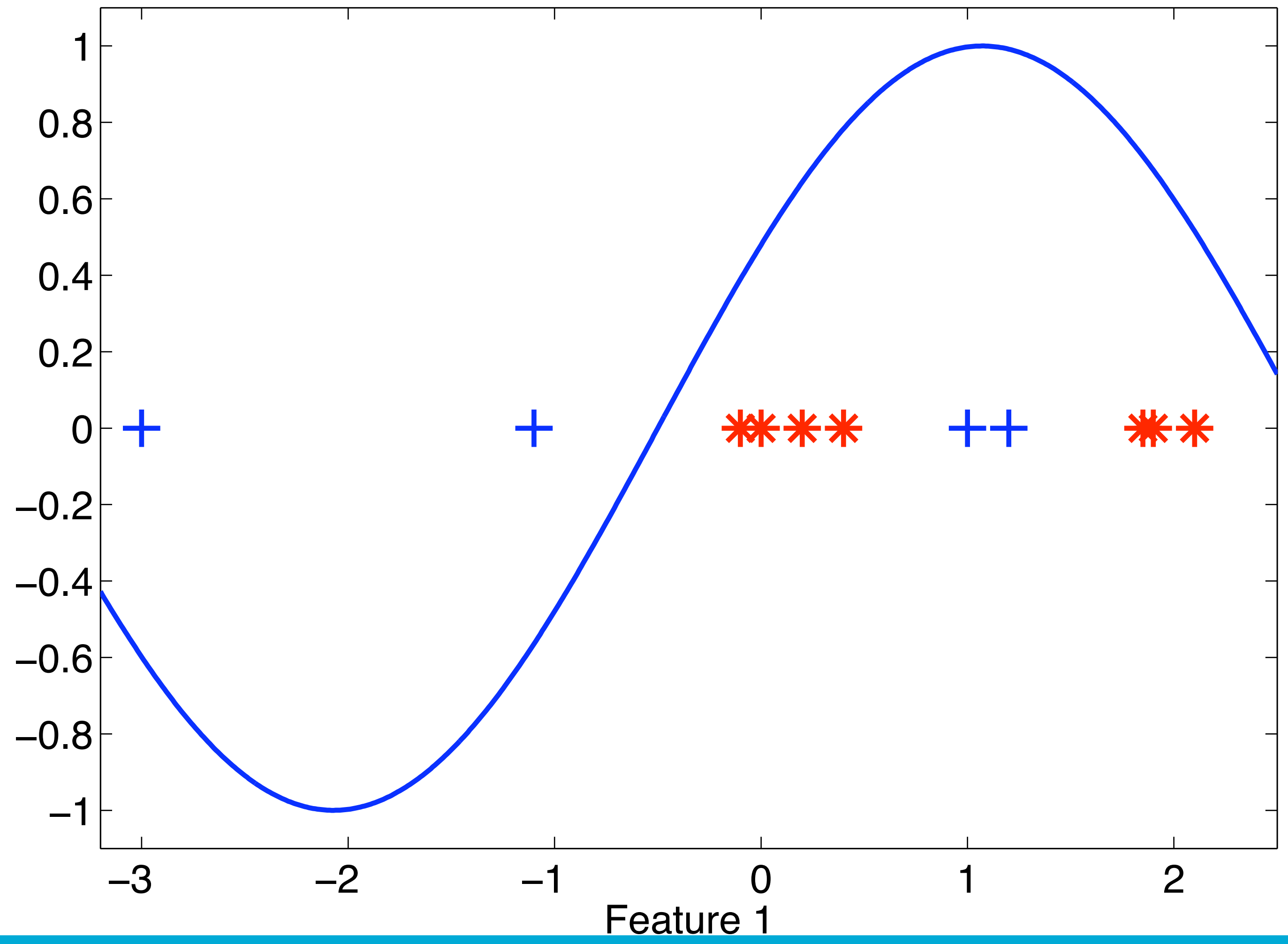


- So the complexity of the classifier = number of free parameters?

1D classifier (2)

- I can define another classifier:

$$f(x) = \mathrm{sign}(\sin(\omega x))$$

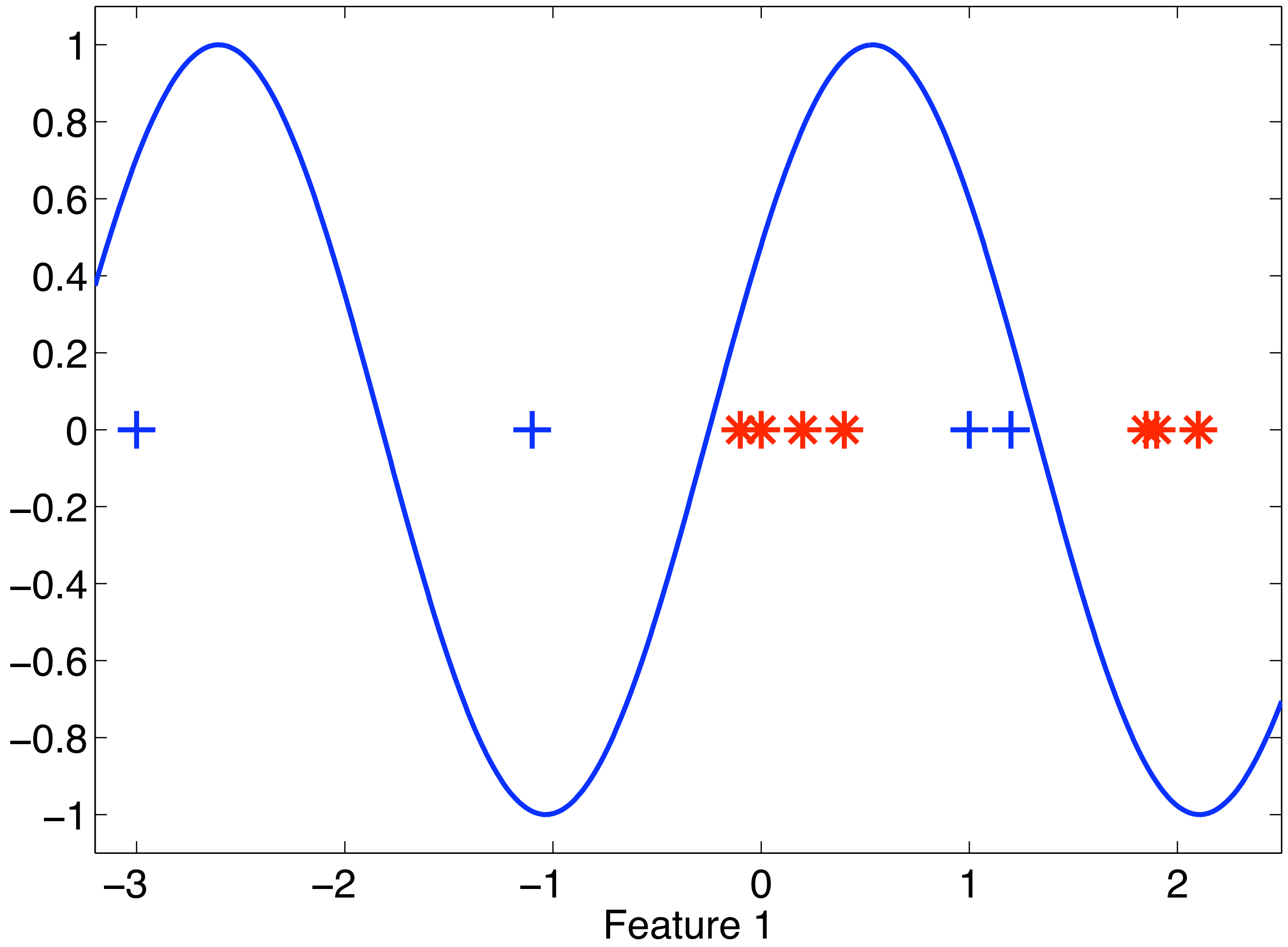


$$\omega = 1$$

1D classifier (2)

- I can define another classifier:

$$f(x) = \mathrm{sign}(\sin(\omega x))$$

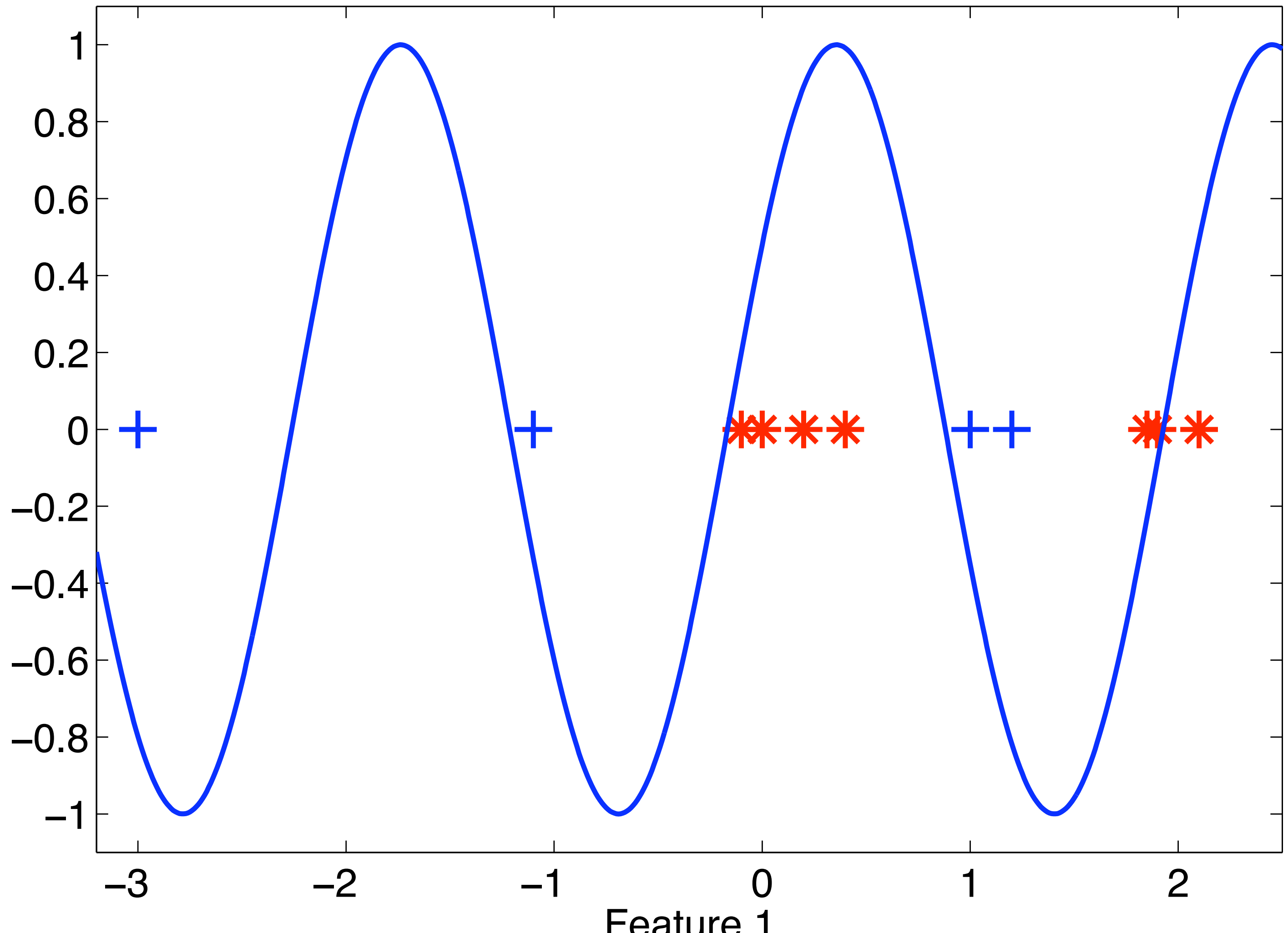


$$\omega = 2$$

1D classifier (2)

- I can define another classifier:

$$f(x) = \text{sign}(\sin(\omega x))$$

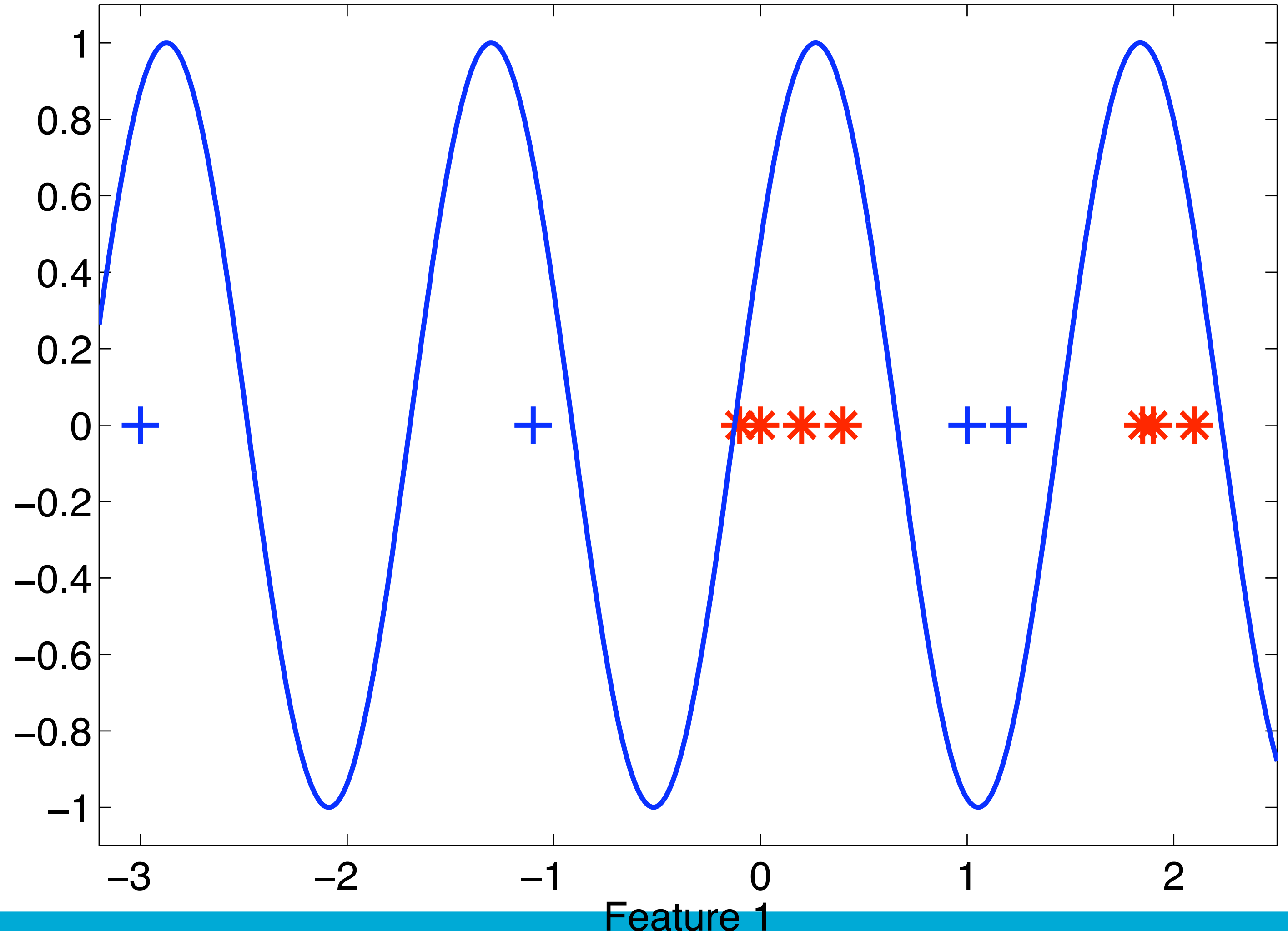


$$\omega = 3$$

1D classifier (2)

- I can define another classifier:

$$f(x) = \text{sign}(\sin(\omega x))$$



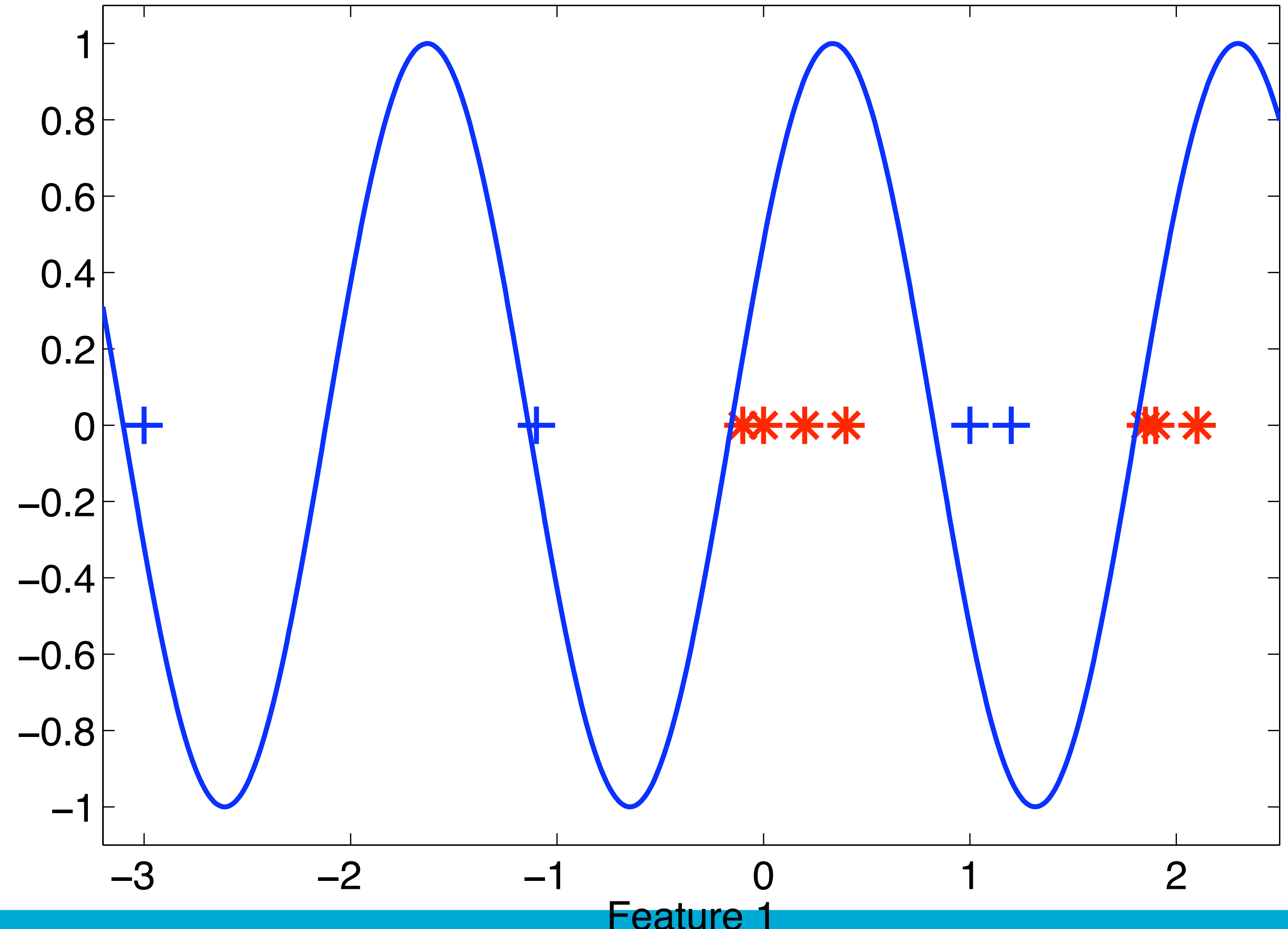
$$\omega = 4$$

TU Delft

1D classifier (2)

- I can define another classifier:

$$f(x) = \text{sign}(\sin(\omega x))$$



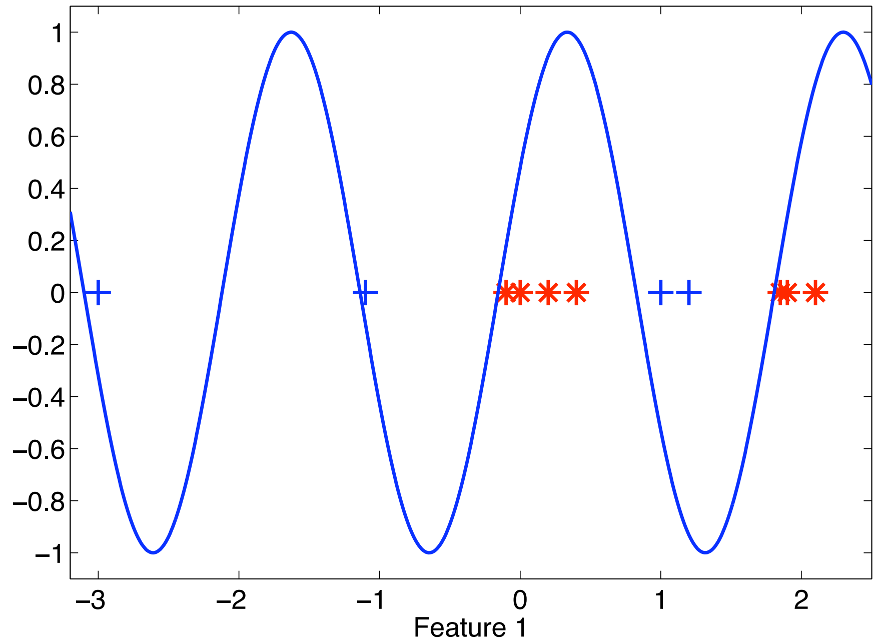
$$\omega = 3.2$$

1D classifier (3)

- I can define another classifier:

$$f(x) = \text{sign}(\sin(\omega x))$$

- It appears that by changing the frequency, you (almost) always can separate the red and the blue:



- But the number of parameters is 1.

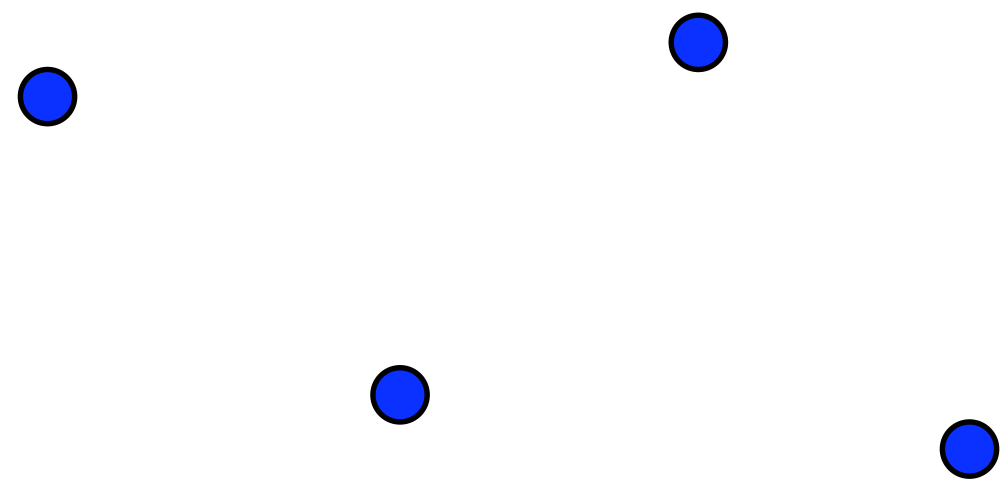
TUDelft

Measuring complexity

- By changing the regularization parameter, the complexity is changed
- Is there a direct way to measure complexity?
- Yes: the VC-dimension (Vapnik-Chervonenkis dimension) of a classifier

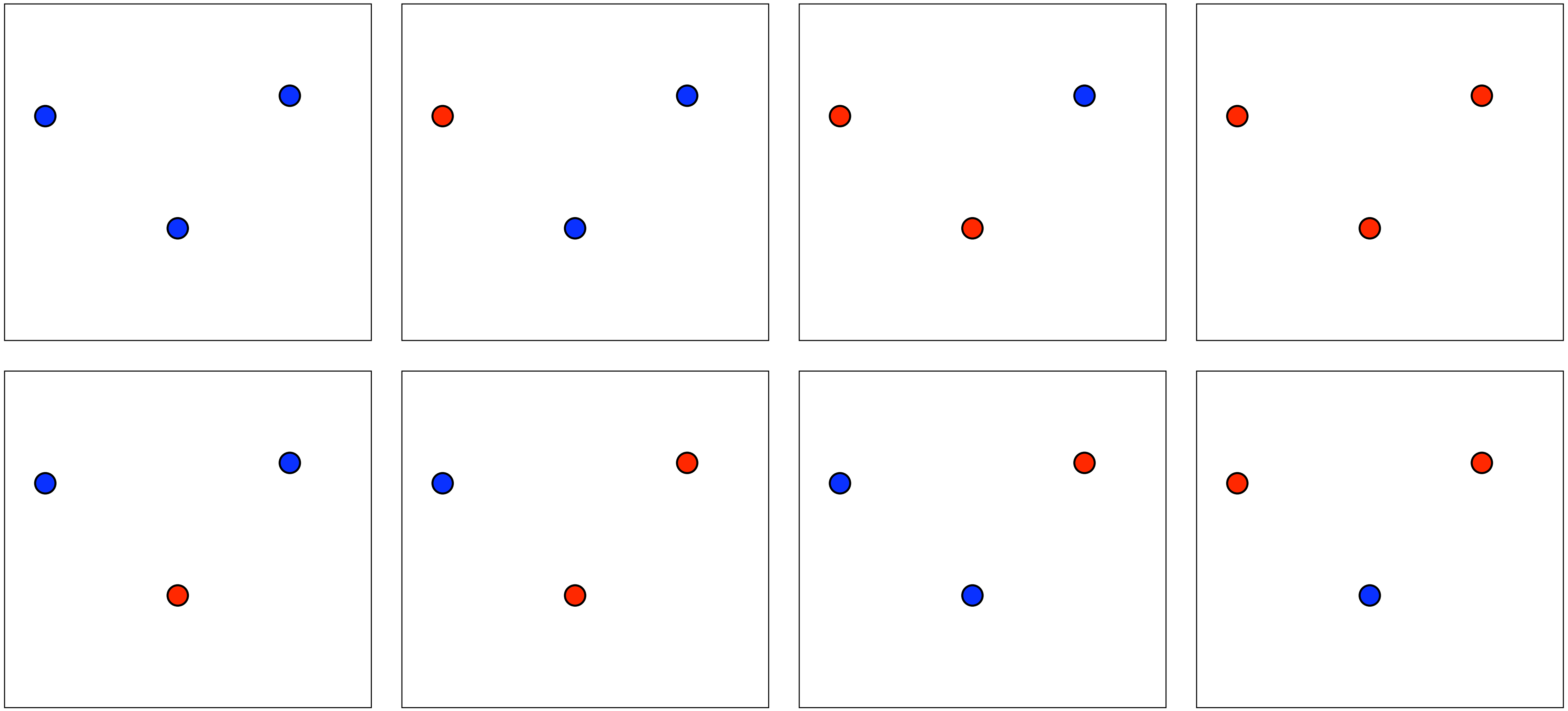
VC-dimension

- h: the VC-dimension of a classifier:



The largest number of vectors **h** that can be separated in all the 2^h possible ways.

VC-dim for linear classifier



- For linear classifier: $h = p + 1$

TU Delft

Use of VC-dimension

- Unfortunately, only for a very few classifiers the VC-dimension is known
- Fortunately, when you know h of a classifier, you can bound the true error of the classifier

Bounding the true error

not for exam

With probability at least $1-\eta$ the inequality holds:

$$\varepsilon \leq \varepsilon_A + \frac{\mathcal{E}(N)}{2}\left(1+\sqrt{1+\frac{\varepsilon_A}{\mathcal{E}(N)}}\right)$$

where

$$\mathcal{E}(N) = 4\frac{h(\ln(2N/h)+1)-\ln(\eta/4)}{N}$$

V.Vapnik, Statistical learning theory, 1998

- When h is small, the true error is close to the apparent error

TUDelft

Is this bound practical?

- The given bound (and others) is very loose
- The worst case scenario is assumed: objects can be randomly labeled
- In practice, features are chosen such that objects from one class are nicely clustered and can be separated from other classes

TU Delft

Compactness hypothesis

Representations of real world objects are close. There is no ground for any generalization on representations that do not obey this demand.

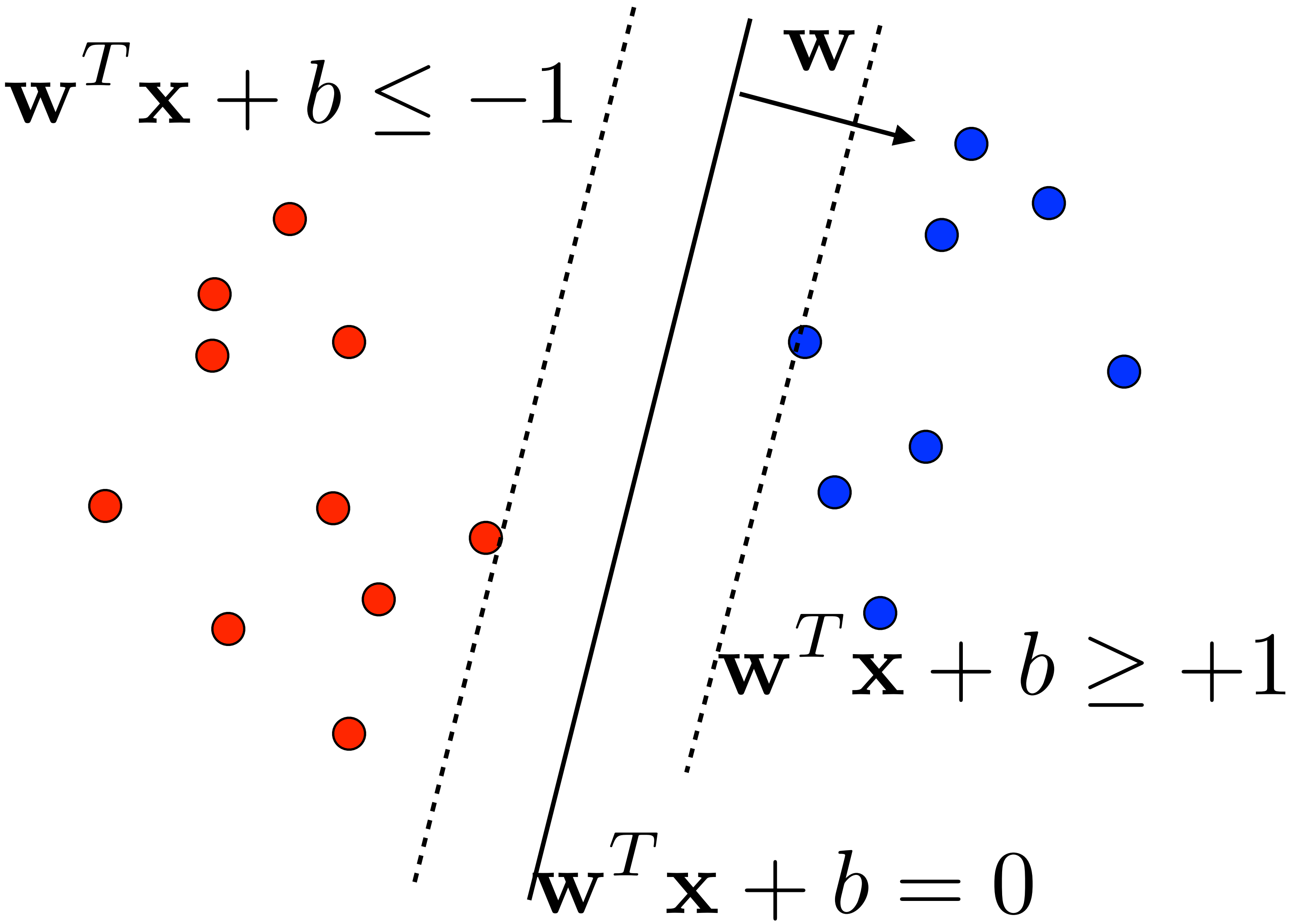
A.G. Arkadev and E.M. Braverman,
Computers and Pattern Recognition, 1966

Changing h for linear classifiers

- The linear classifier had $h = p + 1$
- By putting some constraints on this linear classifier, the VC dimension can be reduced
- Assume a linearly separable dataset, constrain the weights such that the output of the classifier is always larger than one:

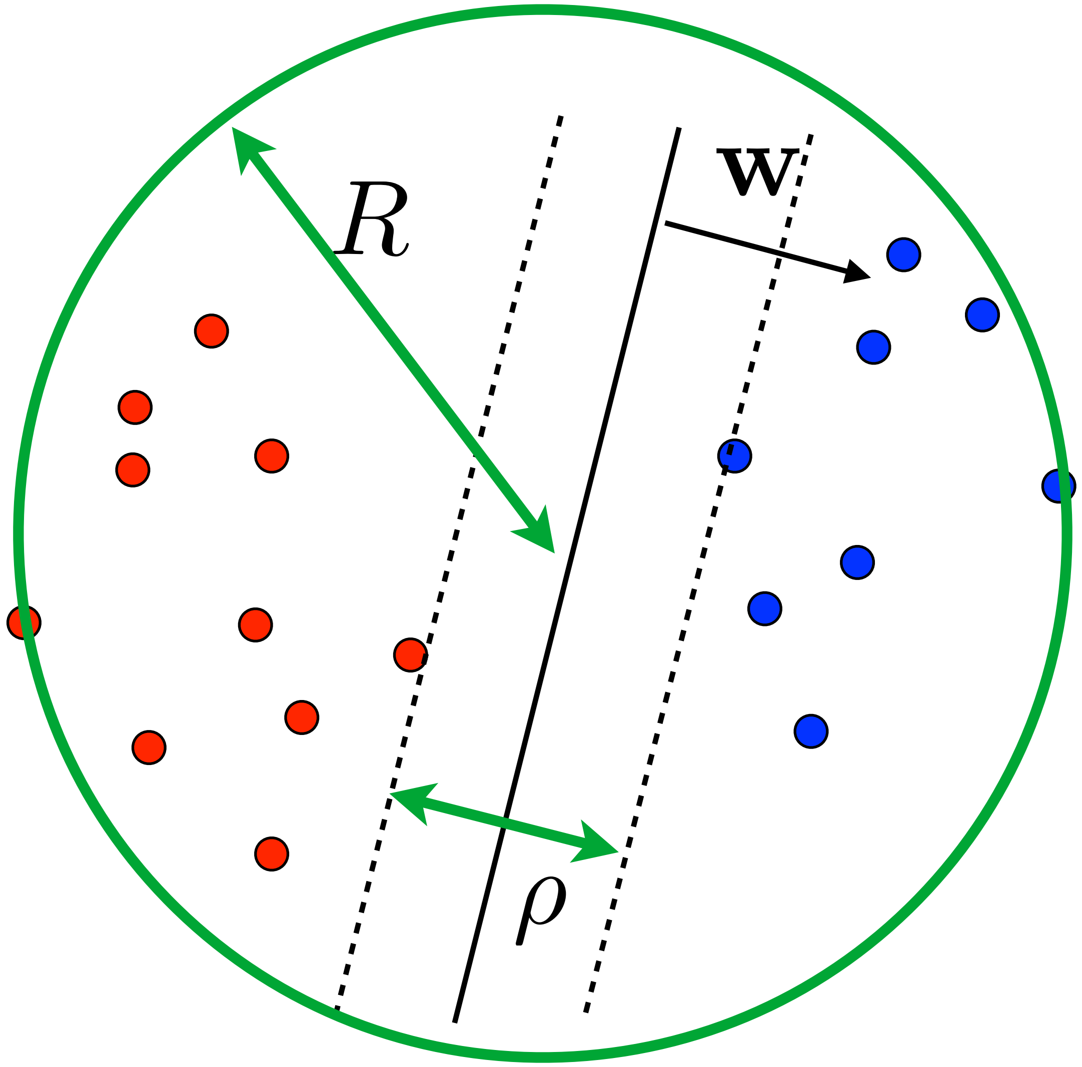
$$\begin{aligned} \mathbf{w}^T\mathbf{x}_i + b \geq +1, \quad & \text{for } y_i = +1 \\ \mathbf{w}^T\mathbf{x}_i + b \leq -1, \quad & \text{for } y_i = -1 \end{aligned}$$

Constraining the weights



The constraints force the weights to have a minimum value: 'canonical hyperplane'

h for a canonical hyperplane



$$h \leq \min\left(\left\lfloor \frac{R^2}{\rho^2} \right\rfloor, p\right) + 1$$

Finding a good classifier

- To find a classifier with small VC-dimension:
  1. minimize the dimensionality p
  2. minimize the radius R
  3. maximize the margin ρ
- Using simple algebra $\|\mathbf{w}\|^2 = \dfrac{2}{\rho^2}$
- So:

$$\min \frac{1}{2}\|\mathbf{w}\|^2$$
$$\begin{aligned} \mathbf{w}^T\mathbf{x}_i + b &\geq +1, \quad \text{for } y_i = +1 \\ \mathbf{w}^T\mathbf{x}_i + b &\leq -1, \quad \text{for } y_i = -1 \end{aligned}$$

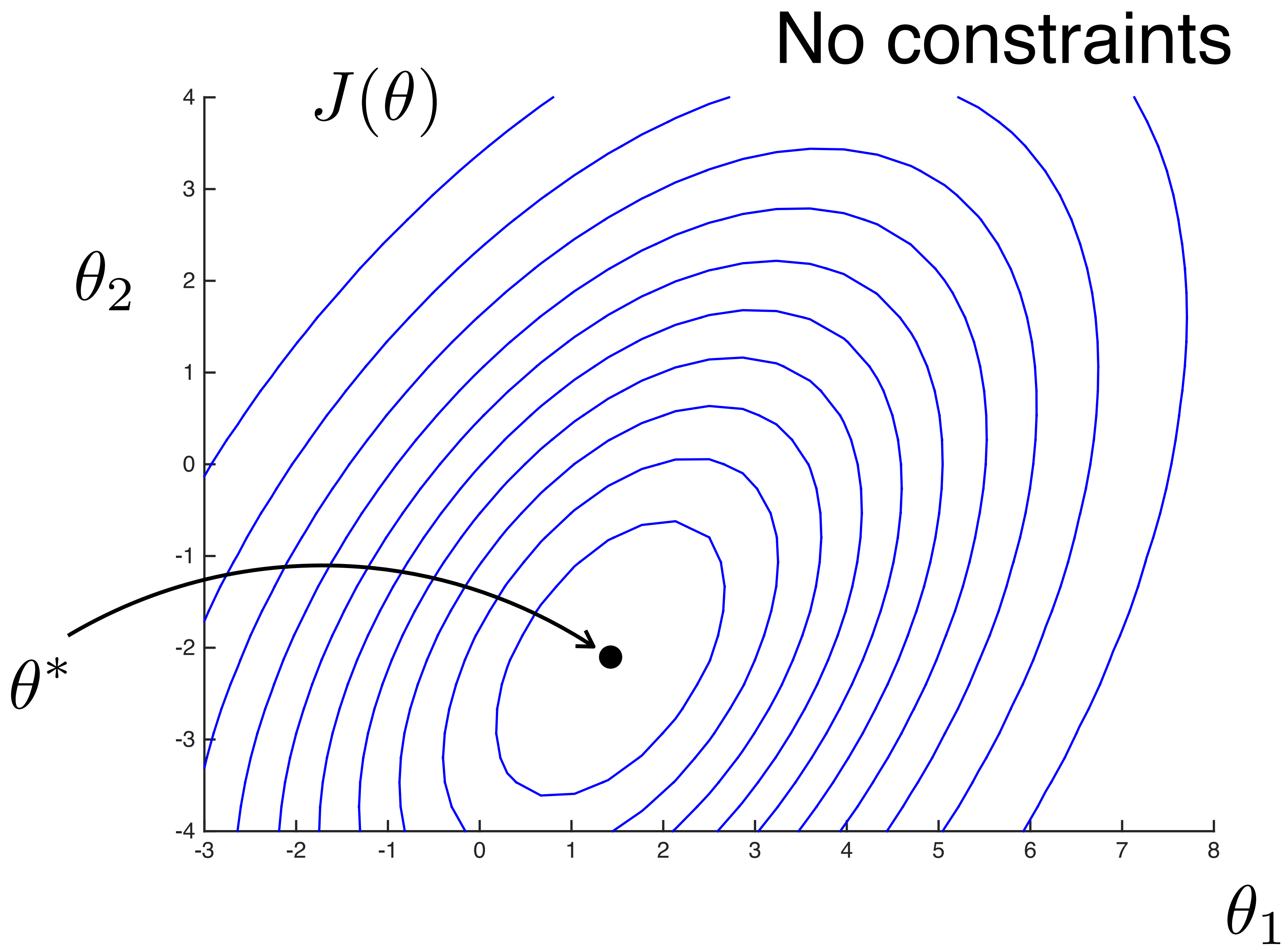
Finding a good classifier

- To find a classifier with small VC-dimension:
  1. minimize the dimensionality p
  2. minimize the radius R
  3. maximize the margin ρ
- Using simple algebra $\|\mathbf{w}\|^2 = \dfrac{2}{\rho^2}$
- So:

$$\boxed{\begin{array}{l} \min \frac{1}{2}\|\mathbf{w}\|^2 \\ \mathbf{w}^T\mathbf{x}_i + b \geq +1, \quad \text{for } y_i = +1 \\ \mathbf{w}^T\mathbf{x}_i + b \leq -1, \quad \text{for } y_i = -1 \end{array}}$$

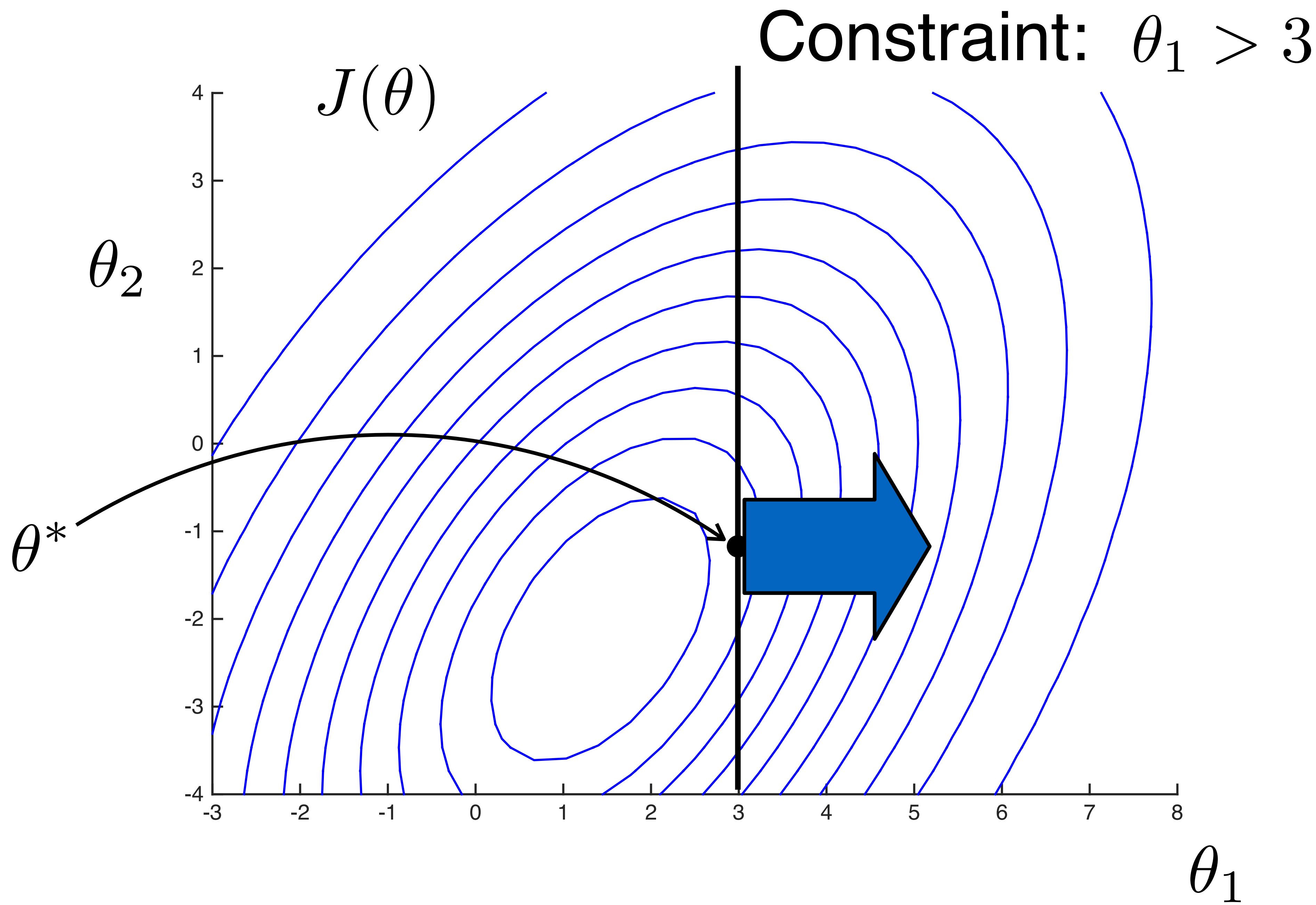
Support vector classifier

Constrained optimisation

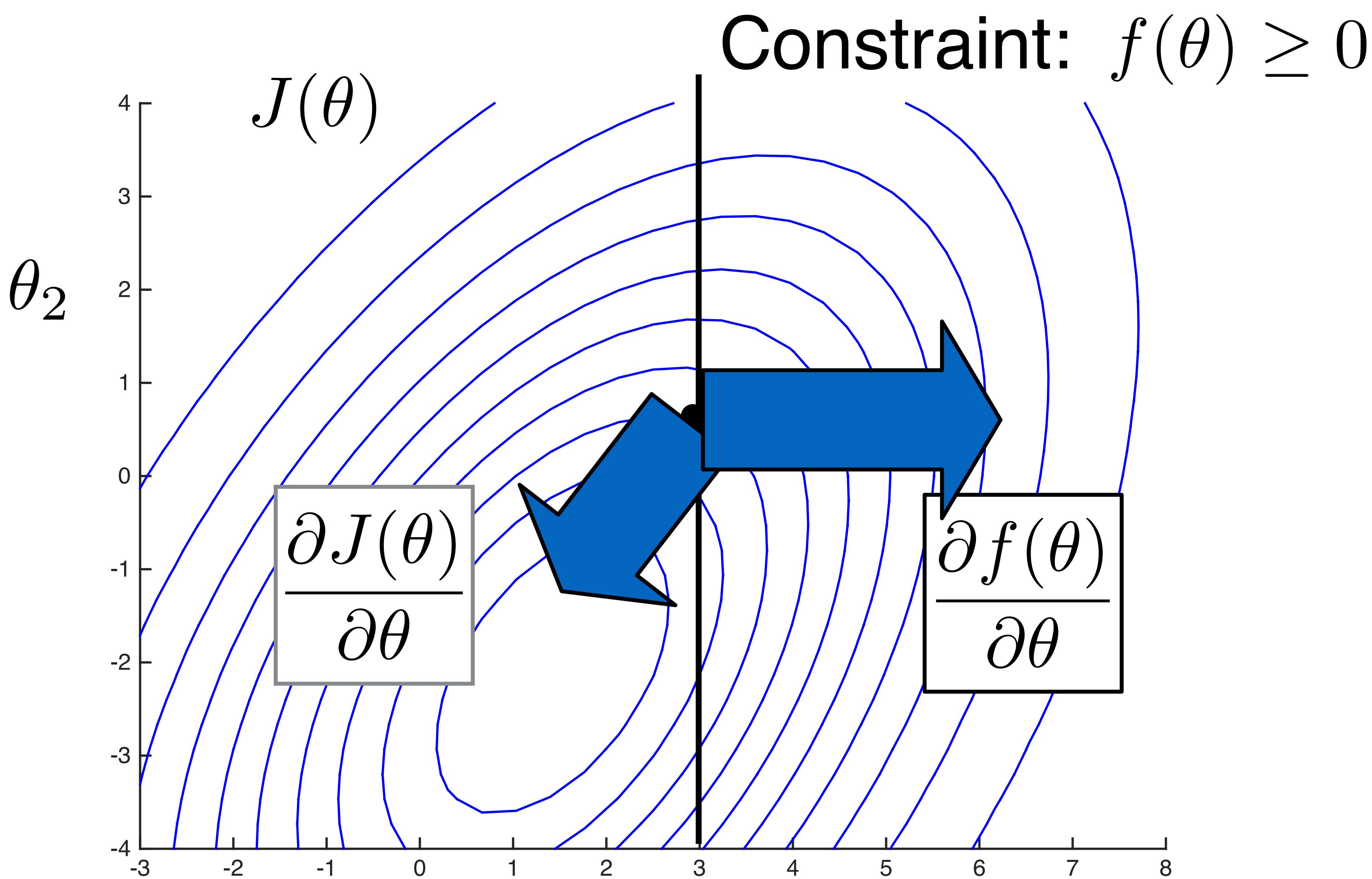


TUDelft

Constrained optimisation

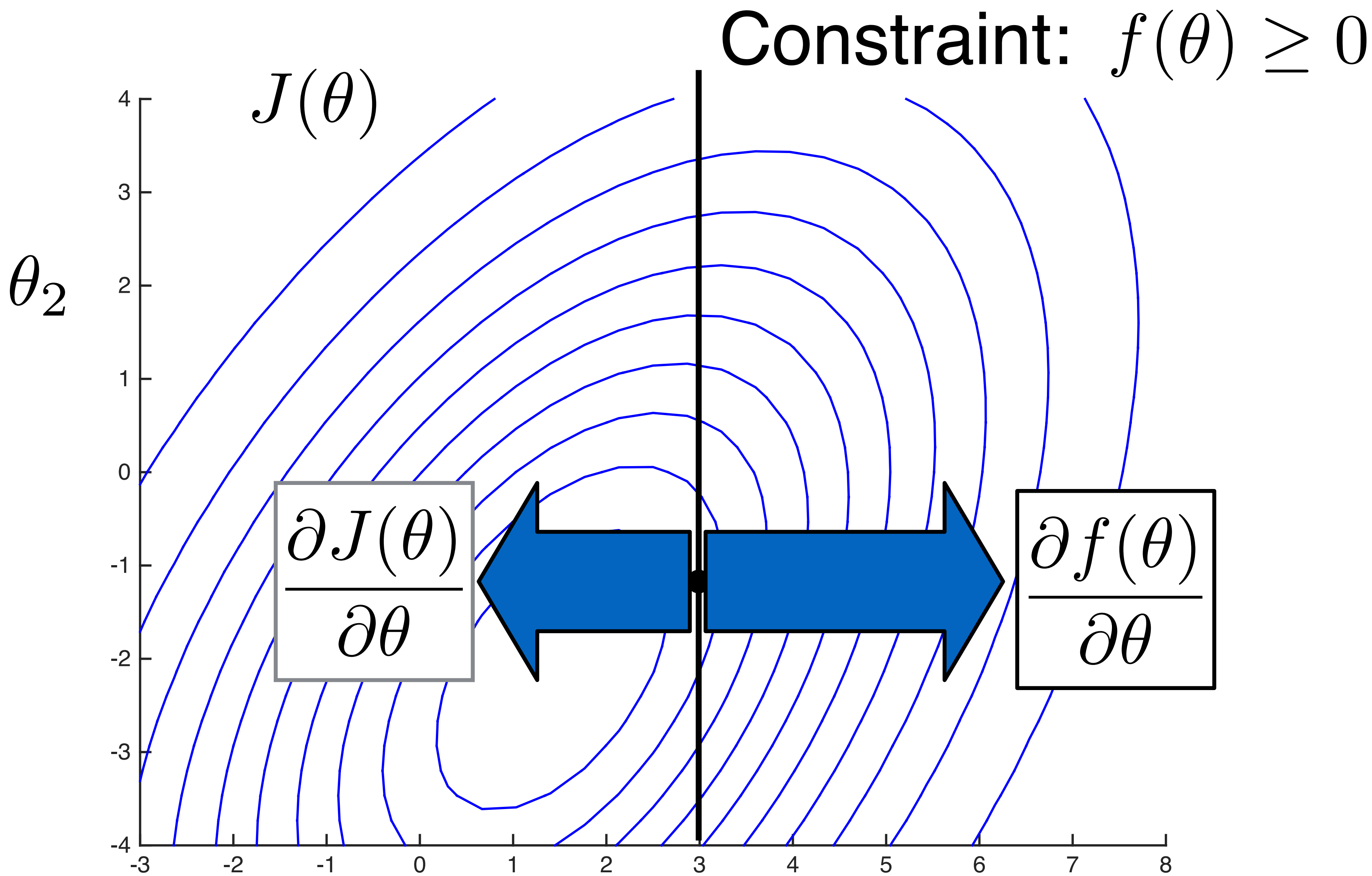


Constrained optimisation: where?



$$\frac{\partial J(\theta)}{\partial \theta} = \lambda \frac{\partial f(\theta)}{\partial \theta}$$

Constrained optimisation: where?



$$\frac{\partial J(\theta)}{\partial \theta} = \lambda \frac{\partial f(\theta)}{\partial \theta}$$

Constrained optimisation

- When you want to optimise

$$\min_{\theta} J(\theta)$$

$$\text{subject to } f(\theta) \geq 0$$

- introduce 'Lagrange Multiplier' λ and define the Lagrangian:

$$\mathcal{L}(\theta, \lambda) = J(\theta) - \lambda f(\theta)$$

- Take the derivative with respect to θ and λ, and set to zero.

For the support vector classifier

$$\min \frac{1}{2}\|\mathbf{w}\|^2$$

$$\mathbf{w}^T\mathbf{x}_i + b \geq +1, \quad \text{for } y_i = +1$$

$$\mathbf{w}^T\mathbf{x}_i + b \leq -1, \quad \text{for } y_i = -1$$

- This can be shortened:

$$\min \frac{1}{2}\|\mathbf{w}\|^2$$

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq +1 \quad \text{for all } i$$

- Constraint:

$$f_i(\mathbf{w}) = y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1$$

TU Delft

For the support vector classifier

$$\min \frac{1}{2}\|\mathbf{w}\|^2$$
$$\mathbf{w}^T\mathbf{x}_i + b \geq +1, \quad \text{for } y_i = +1$$
$$\mathbf{w}^T\mathbf{x}_i + b \leq -1, \quad \text{for } y_i = -1$$

- This can be rewritten into the Lagrangian:

$$\mathcal{L}(\mathbf{w}, \alpha) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_i \alpha_i \left(y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1\right)$$

where in the literature α_i are used as the Lagrange multipliers

TU Delft

Minimizing the Lagrangian

- Have to take the derivative with respect to **w**, b, and α_i and set the derivative to 0
- For instance:

$$\mathcal{L}(\mathbf{w}, \alpha) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_i \alpha_i \left(y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1\right)$$

$$\frac{\partial \mathcal{L}(\mathbf{w}, \alpha)}{\partial \mathbf{w}} = \mathbf{w} - \sum_i \alpha_i y_i \mathbf{x}_i = 0$$

- Solve:

$$\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$$

TU Delft

Optimizing the classifier

- Solving w.r.t. **w** and b is “simple”.
- Solving w.r.t. α_i gives:

$$\max_{\alpha} \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j}^{N} y_i y_j \alpha_i \alpha_j \mathbf{x}_i^T \mathbf{x}_j$$

$$\alpha_i \geq 0 \quad \forall i$$

$$\sum_{i=1}^{N} \alpha_i y_i = 0$$

$$\mathbf{w} = \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i$$

- Quadratic Programming Problem

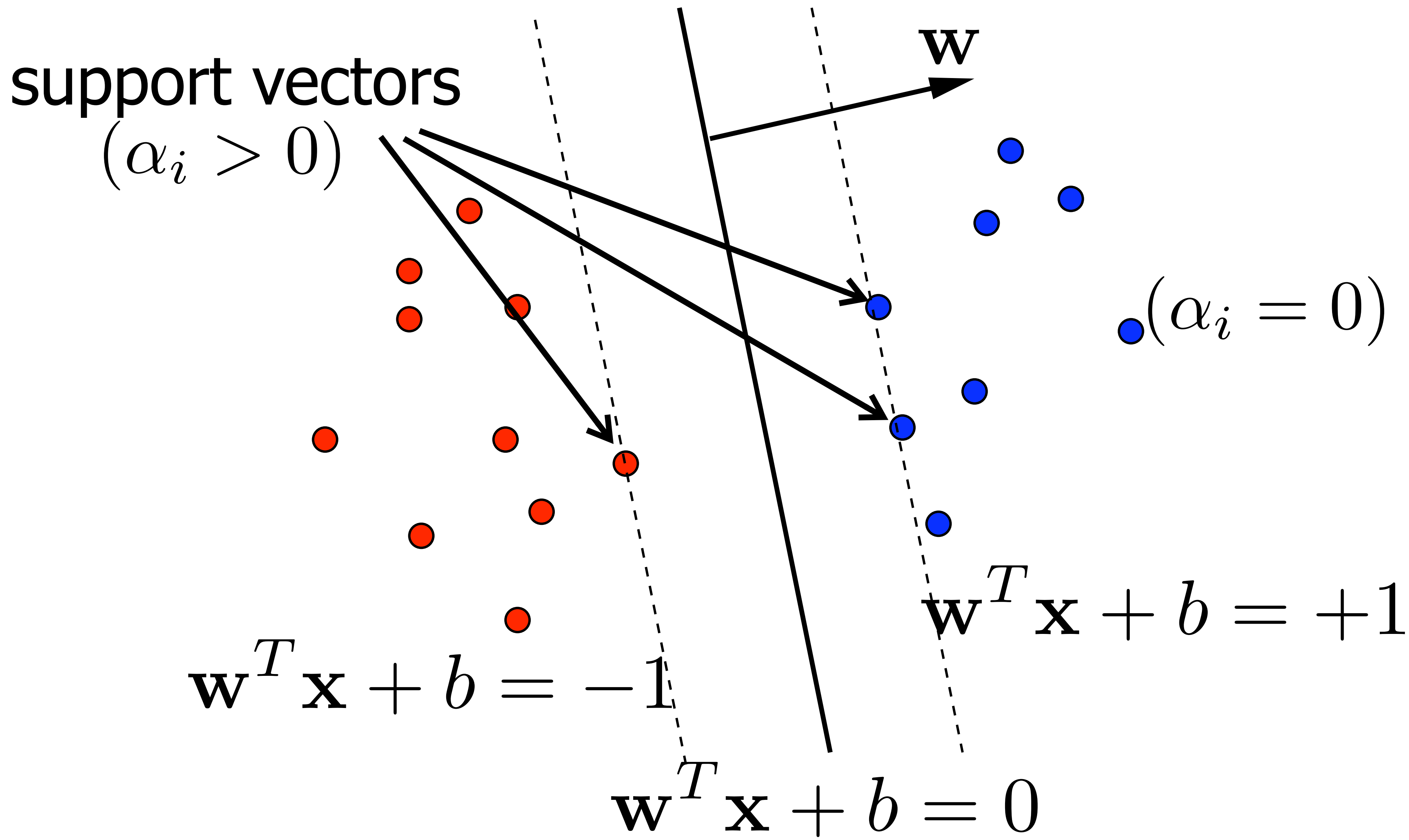
Support vectors

- The classifier becomes:

$$\mathbf{w} = \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i$$

- The solution is expressed in terms of objects, not features.
- Only a few weights become non-zero
- The objects with non-zero weight are called the support vectors

Support vector classifier

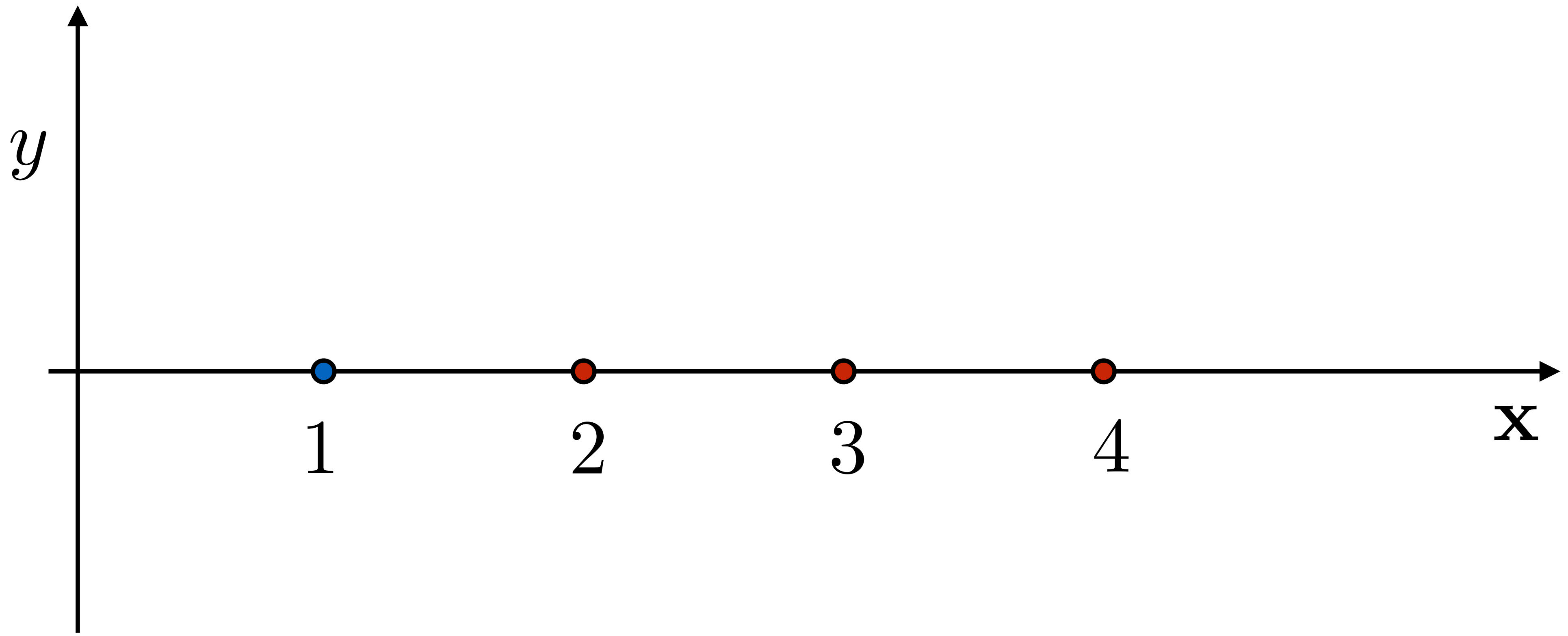


TU Delft

Use a linear classifier for:

$$y = \text{sign}(\mathbf{w}^T \mathbf{x} + w_0)$$

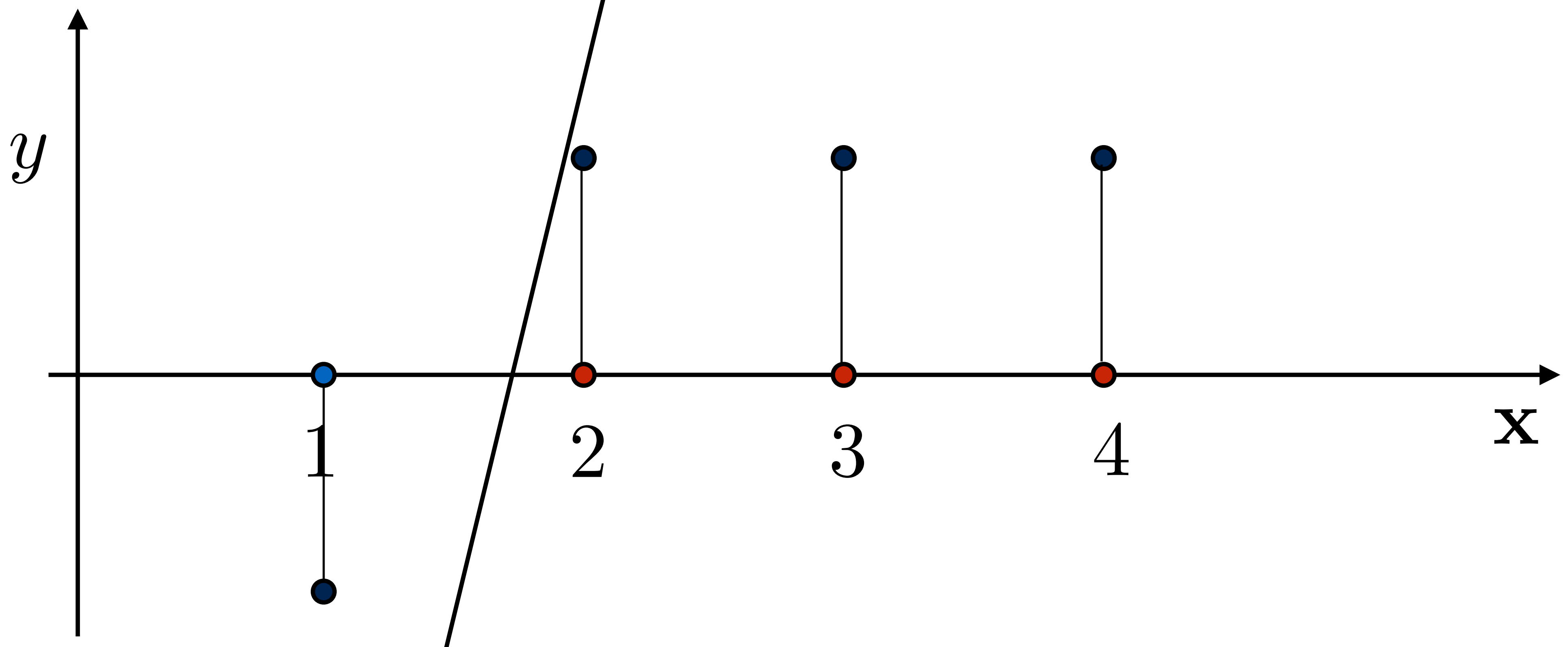
- What are the optimal **w** and w_0?



Use a linear classifier for:

$$y = \mathrm{sign}(\mathbf{w}^T \mathbf{x} + w_0)$$

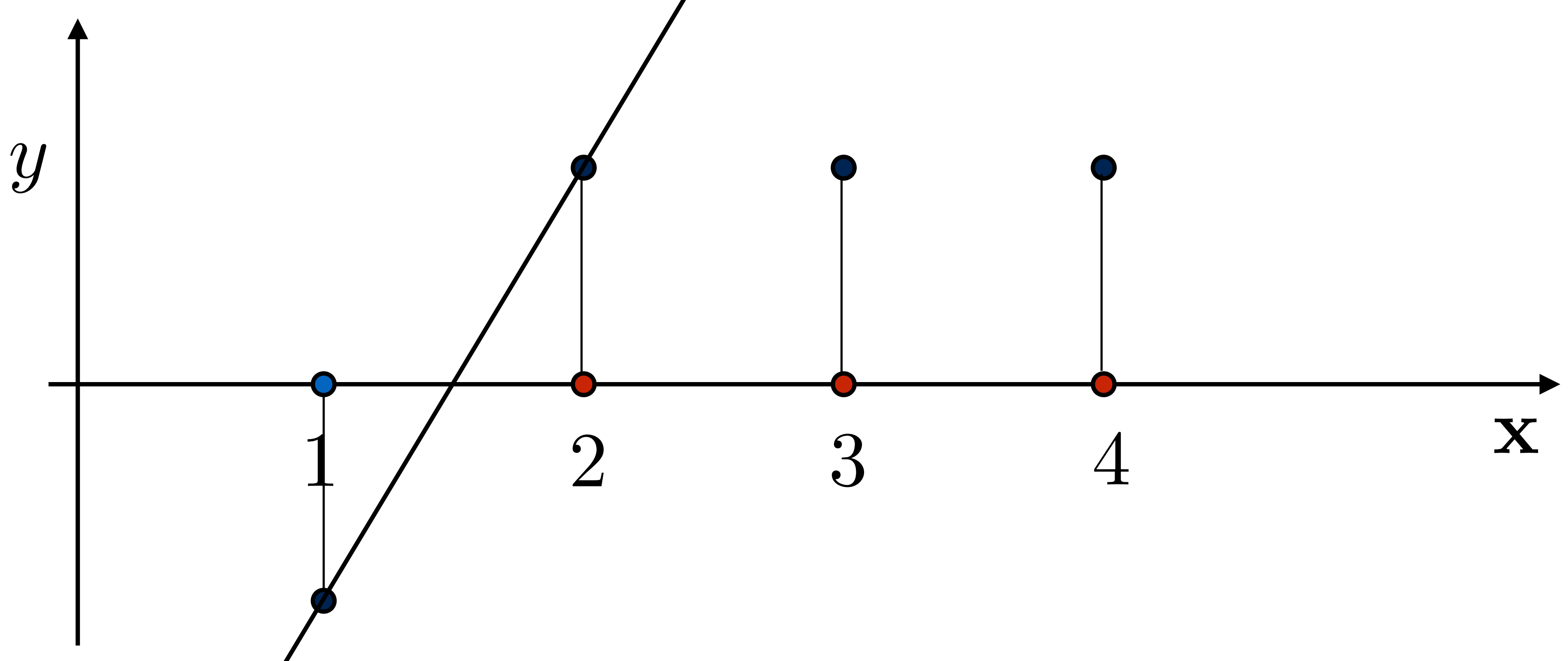
- What are the optimal **w** and w_0?



Use a linear classifier for:

$$y = \mathrm{sign}(\mathbf{w}^T \mathbf{x} + w_0)$$

- What are the optimal **w** and w_0?



Conclusions

- Always tune the complexity of your classifier to your data (#training objects, dimensionality, class overlap, class shape...)
- Complexity of a classifier is an elusive concept; number of parameters is not sufficient
- VC-dimension is complexity measure, taking a worst case approach
- Support vector classifier tries to minimise its VC-dimension