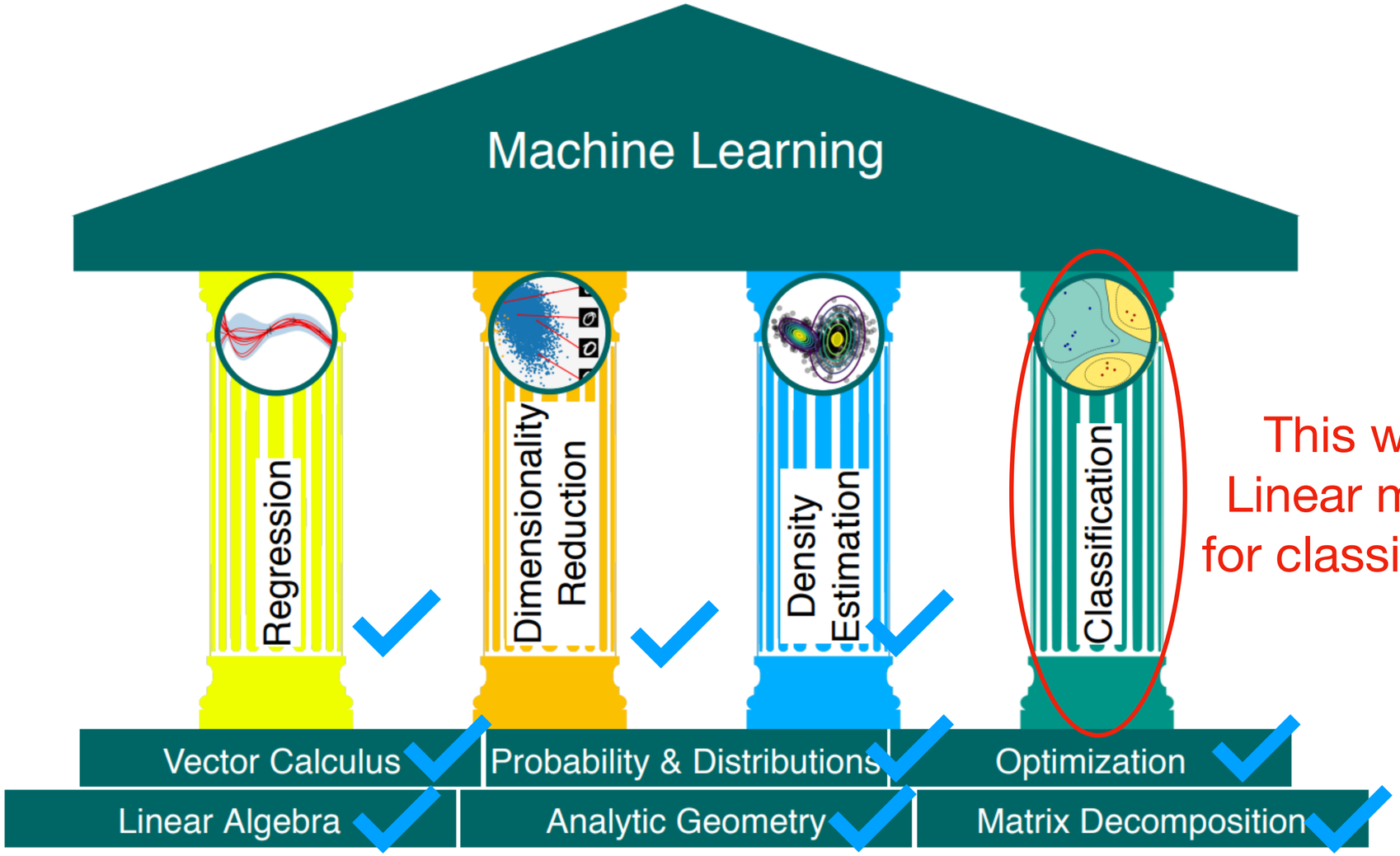
Classification

Week 11 - Introduction to ML / Thang Bui / ANU / 2023 S2

Housekeeping

- *Assignment 4* is now available on Wattle (due next week - W12 Monday)
- *No more* tutorials
- *Exam* timetable is available
  - Past exam papers released on Wattle
- Guest lecture: W12 Monday October 23
  - Dr Zheng Yuan, King's College London
  - *Examinable*!
  - Please do show up!

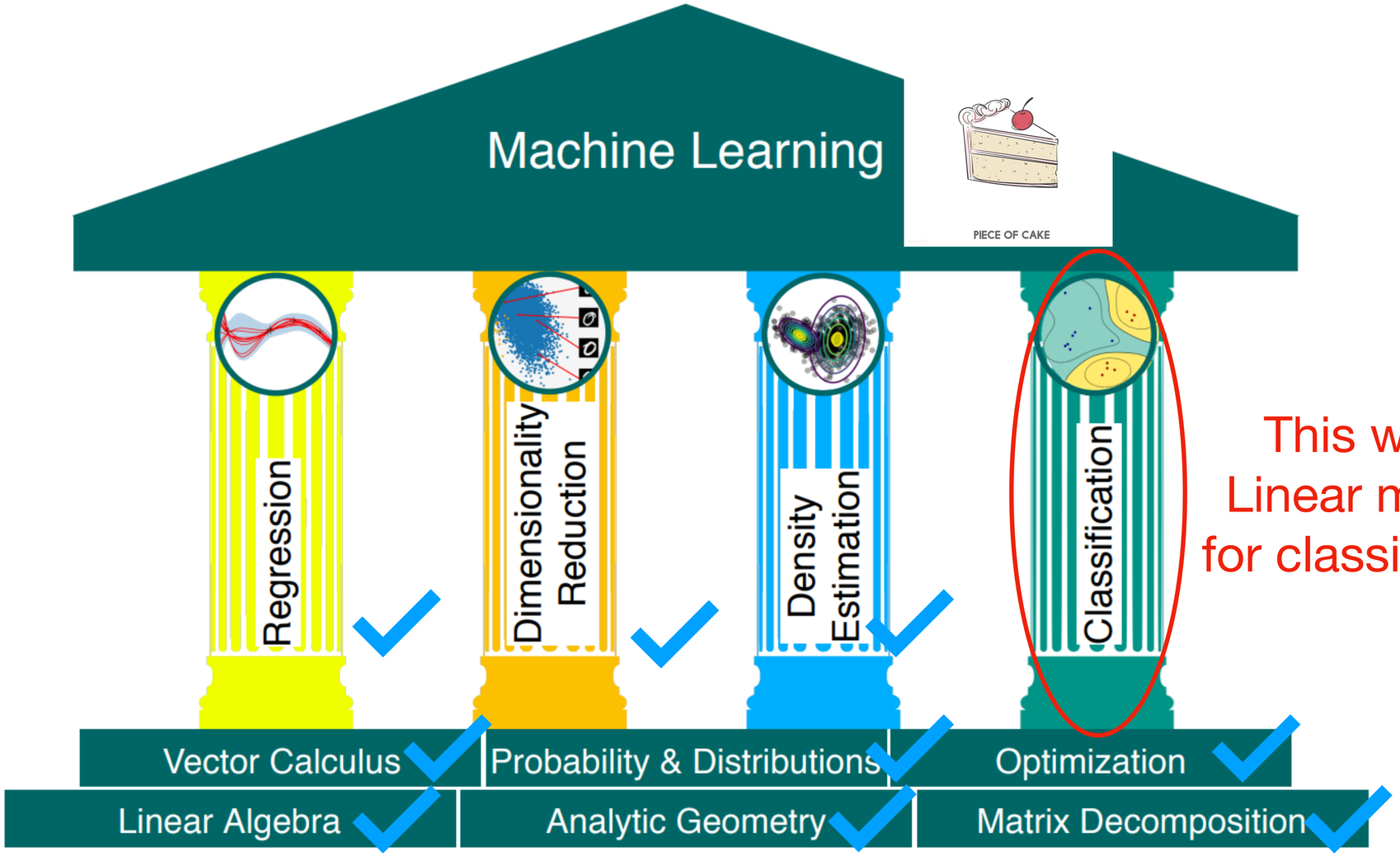
Foundations of ML



This week:
Linear models
for classification

Source: Maths for ML Textbook

Foundations of ML



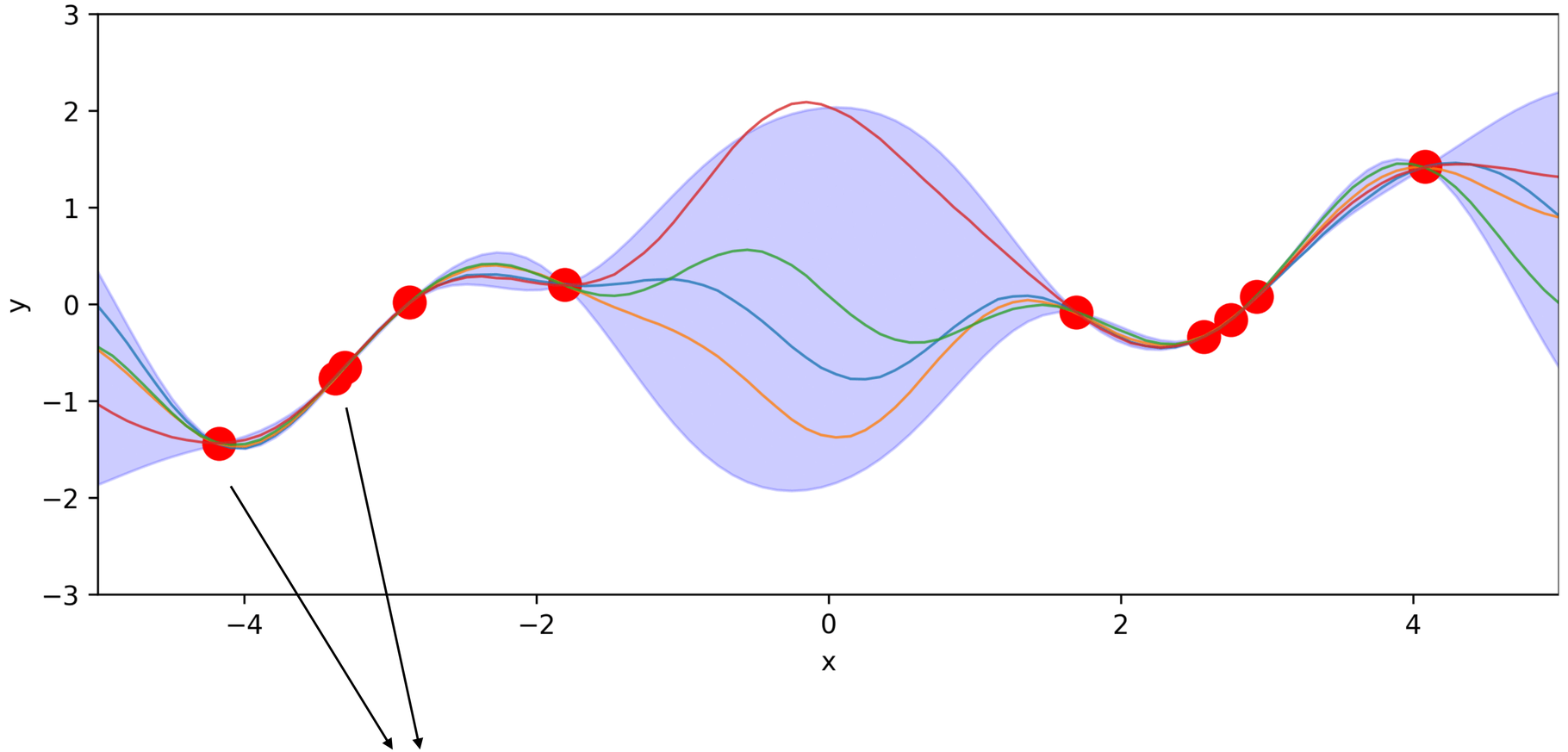
This week:
Linear models
for classification

Source: Maths for ML Textbook

Overview

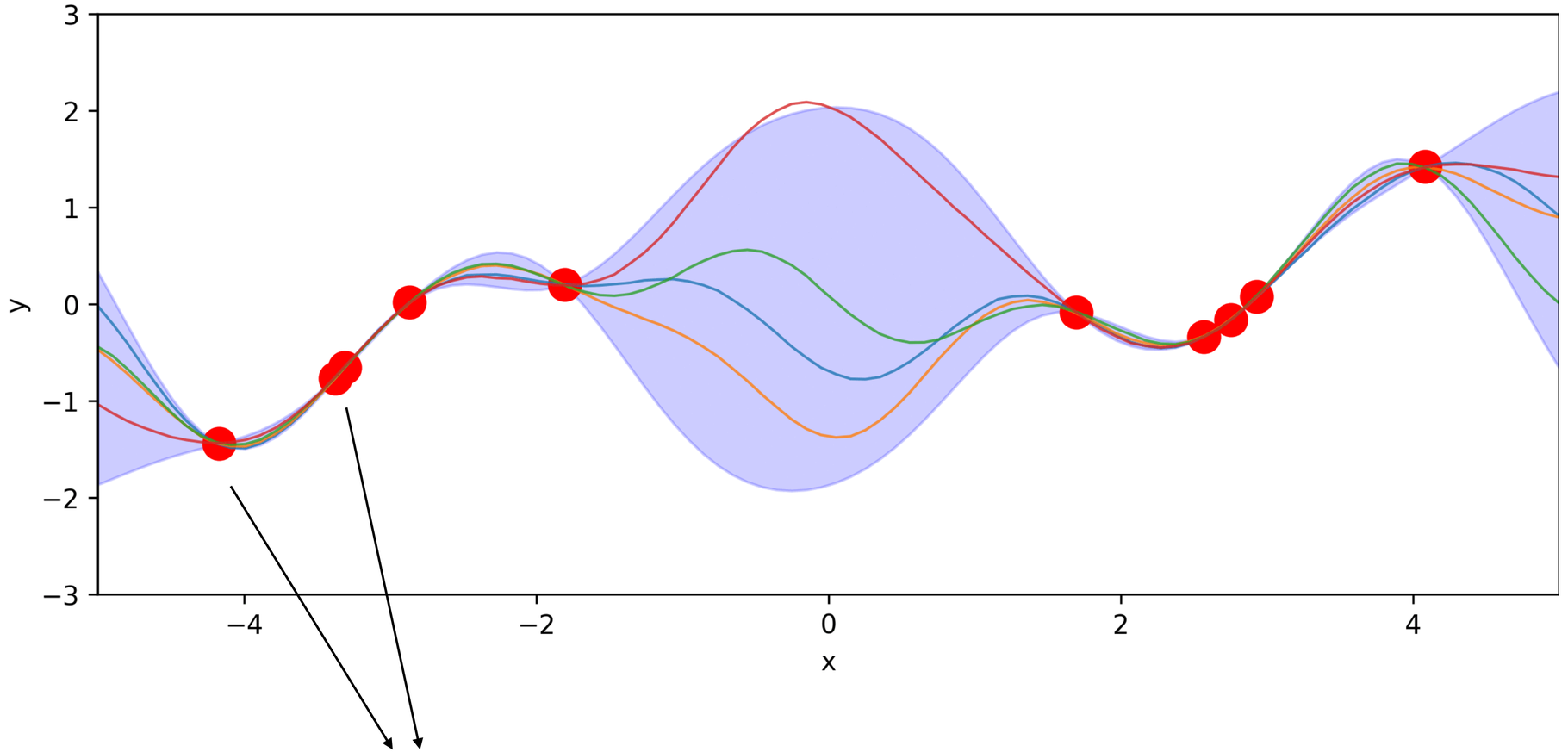
1. **Recap: Linear regression**
2. From least squares to binary logistic classification
3. From binary to multi-class classification
4. Evaluation
5. Linearly separable data - perceptron algorithm

Recap: regression and linear regression



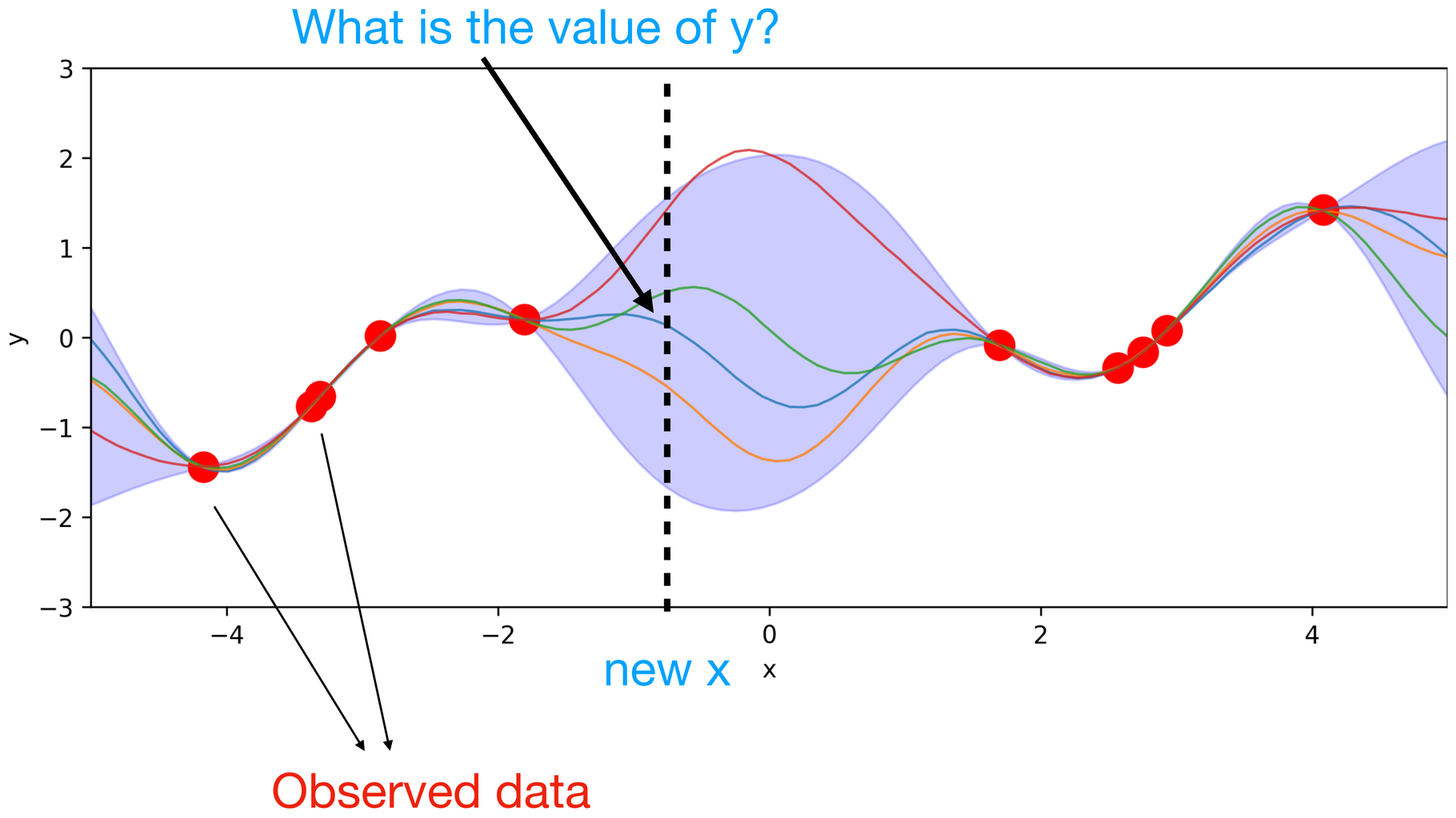
Observed data

Recap: regression and linear regression



Observed data

Recap: regression and linear regression



Data and linear assumption

Data and linear assumption

Training data N input, output pairs $\mathcal{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)\}, \mathbf{x}_n \in \mathbb{R}^D, y_n \in \mathbb{R}$

Each row is a data point

Each column is a feature dimension

$$X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1D} \\ x_{21} & x_{22} & \cdots & x_{2D} \\ \vdots & \vdots & \vdots & \vdots \\ x_{N1} & x_{N2} & \cdots & x_{ND} \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

One scalar per data point

Data and linear assumption

Training data N input, output pairs $\mathcal{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)\}, \mathbf{x}_n \in \mathbb{R}^D, y_n \in \mathbb{R}$

Each row is a data point

Each column is a feature dimension

$$X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1D} \\ x_{21} & x_{22} & \cdots & x_{2D} \\ \vdots & \vdots & \vdots & \vdots \\ x_{N1} & x_{N2} & \cdots & x_{ND} \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

One scalar per data point

Assumptions:

- Underlying function is **linear**, $f_\theta(\mathbf{x}) = \sum_{d=1}^{D} \theta_d x_d = \theta^\intercal \mathbf{x}, \theta \in \mathbb{R}^D$

$$\begin{bmatrix} f_\theta(\mathbf{x}_1) \\ f_\theta(\mathbf{x}_2) \\ \vdots \\ f_\theta(\mathbf{x}_N) \end{bmatrix} = \begin{bmatrix} \theta^\intercal \mathbf{x}_1 \\ \theta^\intercal \mathbf{x}_2 \\ \vdots \\ \theta^\intercal \mathbf{x}_N \end{bmatrix} = X\theta$$

- Due to measurement noise, observed y is a noisy version of $f(\mathbf{x})$

Data and linear assumption

Training data N input, output pairs $\mathcal{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)\}, \mathbf{x}_n \in \mathbb{R}^D, y_n \in \mathbb{R}$

Each row is a data point

Each column is a feature dimension

$$X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1D} \\ x_{21} & x_{22} & \cdots & x_{2D} \\ \vdots & \vdots & \vdots & \vdots \\ x_{N1} & x_{N2} & \cdots & x_{ND} \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

One scalar per data point

Assumptions:

- Underlying function is **linear**, $f_\theta(\mathbf{x}) = \sum_{d=1}^{D} \theta_d x_d = \theta^\intercal \mathbf{x}, \theta \in \mathbb{R}^D$
- Due to measurement noise, observed y is a noisy version of $f(\mathbf{x})$

$$\begin{bmatrix} f_\theta(\mathbf{x}_1) \\ f_\theta(\mathbf{x}_2) \\ \vdots \\ f_\theta(\mathbf{x}_N) \end{bmatrix} = \begin{bmatrix} \theta^\intercal \mathbf{x}_1 \\ \theta^\intercal \mathbf{x}_2 \\ \vdots \\ \theta^\intercal \mathbf{x}_N \end{bmatrix} = X\theta$$

Test time: given a new input $\mathbf{x}^*$, prediction = $f(\mathbf{x}^*) = \theta^\intercal \mathbf{x}^*$

Objective function

Objective function

Training data N input, output pairs $\mathscr{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)\}, \mathbf{x}_n \in \mathbb{R}^D, y_n \in \mathbb{R}$

$$X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1D} \\ x_{21} & x_{22} & \cdots & x_{2D} \\ \vdots & \vdots & \vdots & \vdots \\ x_{N1} & x_{N2} & \cdots & x_{ND} \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} \quad \approx \quad \begin{bmatrix} f_\theta(\mathbf{x}_1) \\ f_\theta(\mathbf{x}_2) \\ \vdots \\ f_\theta(\mathbf{x}_N) \end{bmatrix} = \begin{bmatrix} \theta^\intercal \mathbf{x}_1 \\ \theta^\intercal \mathbf{x}_2 \\ \vdots \\ \theta^\intercal \mathbf{x}_N \end{bmatrix} = X\theta$$

Objective function

Training data N input, output pairs $\mathscr{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)\}, \mathbf{x}_n \in \mathbb{R}^D, y_n \in \mathbb{R}$

$$X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1D} \\ x_{21} & x_{22} & \cdots & x_{2D} \\ \vdots & \vdots & \vdots & \vdots \\ x_{N1} & x_{N2} & \cdots & x_{ND} \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} \quad \approx \quad \begin{bmatrix} f_\theta(\mathbf{x}_1) \\ f_\theta(\mathbf{x}_2) \\ \vdots \\ f_\theta(\mathbf{x}_N) \end{bmatrix} = \begin{bmatrix} \theta^\intercal \mathbf{x}_1 \\ \theta^\intercal \mathbf{x}_2 \\ \vdots \\ \theta^\intercal \mathbf{x}_N \end{bmatrix} = X\theta$$

Desideratum: y is well approximated by $f_\theta(\mathbf{x}) = \theta^\intercal \mathbf{x}$

Objective function

Training data N input, output pairs $\mathcal{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)\}, \mathbf{x}_n \in \mathbb{R}^D, y_n \in \mathbb{R}$

$$X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1D} \\ x_{21} & x_{22} & \cdots & x_{2D} \\ \vdots & \vdots & \vdots & \vdots \\ x_{N1} & x_{N2} & \cdots & x_{ND} \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} \quad \approx \quad \begin{bmatrix} f_\theta(\mathbf{x}_1) \\ f_\theta(\mathbf{x}_2) \\ \vdots \\ f_\theta(\mathbf{x}_N) \end{bmatrix} = \begin{bmatrix} \theta^\intercal \mathbf{x}_1 \\ \theta^\intercal \mathbf{x}_2 \\ \vdots \\ \theta^\intercal \mathbf{x}_N \end{bmatrix} = X\theta$$

Desideratum: y is well approximated by $f_\theta(\mathbf{x}) = \theta^\intercal \mathbf{x}$

Loss function: $L(\theta) = \frac{1}{N} \sum_{n=1}^{N} (y_n - f_\theta(\mathbf{x}_n))^2 = \frac{1}{N} \|\mathbf{y} - X\theta\|_2^2 = \frac{1}{N} (\mathbf{y} - X\theta)^\intercal (\mathbf{y} - X\theta)$

Objective function

Training data N input, output pairs $\mathcal{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)\}, \mathbf{x}_n \in \mathbb{R}^D, y_n \in \mathbb{R}$

$$X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1D} \\ x_{21} & x_{22} & \cdots & x_{2D} \\ \vdots & \vdots & \vdots & \vdots \\ x_{N1} & x_{N2} & \cdots & x_{ND} \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} \quad \approx \quad \begin{bmatrix} f_\theta(\mathbf{x}_1) \\ f_\theta(\mathbf{x}_2) \\ \vdots \\ f_\theta(\mathbf{x}_N) \end{bmatrix} = \begin{bmatrix} \theta^\intercal \mathbf{x}_1 \\ \theta^\intercal \mathbf{x}_2 \\ \vdots \\ \theta^\intercal \mathbf{x}_N \end{bmatrix} = X\theta$$

Desideratum: y is well approximated by $f_\theta(\mathbf{x}) = \theta^\intercal \mathbf{x}$

Loss function: $L(\theta) = \frac{1}{N} \sum_{n=1}^{N} (y_n - f_\theta(\mathbf{x}_n))^2 = \frac{1}{N} \|\mathbf{y} - X\theta\|_2^2 = \frac{1}{N} (\mathbf{y} - X\theta)^\intercal (\mathbf{y} - X\theta)$

We want to find θ that minimises the loss function. Closed-form analytic solution!

$$\theta = (X^\intercal X)^{-1} X^\intercal \mathbf{y}$$

Regularised least squares

Regularised least squares

Overfitting occurs because the model is too **complex** (θ has too many large entries), while there are too limited training sample.

We want to *penalise* the amplitude of parameters by **regularisation.**

Regularised least squares = least squares + regularisation

$$L_\lambda(\theta) = \frac{1}{N}\sum_{n=1}^{N}(y_n - f_\theta(\mathbf{x}_n))^2 + \lambda\|\theta\|_p^p$$

Data-fit
Lower = better fit

Regulariser
Lower = simpler model

Hyperparameter

Regularised least squares

Overfitting occurs because the model is too **complex** (θ has too many large entries), while there are too limited training sample.

We want to *penalise* the amplitude of parameters by **regularisation.**

Regularised least squares = least squares + regularisation

$$L_\lambda(\theta) = \frac{1}{N}\sum_{n=1}^{N}(y_n - f_\theta(\mathbf{x}_n))^2 + \lambda\|\theta\|_p^p$$

Data-fit
Lower = better fit

Regulariser
Lower = simpler model

Hyperparameter

Loss function when p = 2: $L(\theta) = \frac{1}{N}\|\mathbf{y} - X\theta\|_2^2 + \lambda\|\theta\|_2^2 = \frac{1}{N}(\mathbf{y} - X\theta)^\intercal(\mathbf{y} - X\theta) + \lambda\|\theta\|_2^2$

We want to find θ that minimises the loss function. Closed-form analytic solution!

$$\theta = (X^\intercal X + N\lambda \mathrm{I})^{-1}X^\intercal \mathbf{y}$$

The probabilistic perspective

Week 6

Bayes' rule: $P(\theta \mid \mathscr{D}) = \frac{P(\theta, \mathscr{D})}{P(\mathscr{D})} = \frac{P(\theta)\, P(\mathscr{D} \mid \theta)}{P(\mathscr{D})}$

Posterior
belief about θ after knowing $\mathscr{D}$

$=$

$$\frac{\textbf{Prior}\ \text{prior belief about } \theta \quad \times \quad \textbf{Likelihood}\ \text{likelihood of } \theta \text{ given } \mathscr{D}}{\textbf{Marginal likelihood}\ \text{or evidence}}$$

likelihood averaged over all potential θ

The probabilistic perspective

Week 6

Bayes' rule: $P(\theta \mid \mathscr{D}) = \frac{P(\theta, \mathscr{D})}{P(\mathscr{D})} = \frac{P(\theta)\, P(\mathscr{D} \mid \theta)}{P(\mathscr{D})}$

Posterior
belief about θ after knowing $\mathscr{D}$

$=$

Prior
prior belief about θ

$\times$

Likelihood
likelihood of θ given $\mathscr{D}$

$\div$

Marginal likelihood
or evidence

likelihood averaged over all potential θ

- Maximum likelihood (ML/MLE), $\operatorname{argmax}_\theta\ p(\mathscr{D} \mid \theta)$ is equiv. empirical loss minimisation (ERM)
- Maximum a-posteriori (MAP), $\operatorname{argmax}_\theta\ p(\mathscr{D} \mid \theta) p(\theta)$ is equiv. regularised ERM
- Exact/approximate Bayesian inference

Linear regression - maximum likelihood

Training data N input, output pairs $\mathscr{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)\}, \mathbf{x}_n \in \mathbb{R}^D, y_n \in \mathbb{R}$

Assumptions:

- Underlying function is **linear**, $f_\theta(\mathbf{x}) = \sum_{d=1}^{D} \theta_d x_d = \theta^\intercal \mathbf{x}, \theta \in \mathbb{R}^D$

$$\begin{bmatrix} f_\theta(\mathbf{x}_1) \\ f_\theta(\mathbf{x}_2) \\ \vdots \\ f_\theta(\mathbf{x}_N) \end{bmatrix} = \begin{bmatrix} \theta^\intercal \mathbf{x}_1 \\ \theta^\intercal \mathbf{x}_2 \\ \vdots \\ \theta^\intercal \mathbf{x}_N \end{bmatrix} = X\theta$$

- Due to measurement noise, observed y is a noisy version of $f(\mathbf{x})$

Linear regression - maximum likelihood

Training data N input, output pairs $\mathscr{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)\}, \mathbf{x}_n \in \mathbb{R}^D, y_n \in \mathbb{R}$

Assumptions:

- Underlying function is **linear**, $f_\theta(\mathbf{x}) = \sum_{d=1}^{D} \theta_d x_d = \theta^\intercal \mathbf{x}, \theta \in \mathbb{R}^D$

$$\begin{bmatrix} f_\theta(\mathbf{x}_1) \\ f_\theta(\mathbf{x}_2) \\ \vdots \\ f_\theta(\mathbf{x}_N) \end{bmatrix} = \begin{bmatrix} \theta^\intercal \mathbf{x}_1 \\ \theta^\intercal \mathbf{x}_2 \\ \vdots \\ \theta^\intercal \mathbf{x}_N \end{bmatrix} = X\theta$$

- Due to measurement noise, observed y is a noisy version of $f(\mathbf{x})$

Likelihood: $p(\mathscr{D} \mid \theta) = \prod_n \mathcal{N}(y_n; f(x_n), \sigma^2) = \prod_n \mathcal{N}(y_n; x_n^\intercal \theta, \sigma^2)$

Factorise across data points

Mean = linear mapping

Measurement noise
Constant across data points

Linear regression - maximum likelihood

Training data N input, output pairs $\mathcal{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)\}, \mathbf{x}_n \in \mathbb{R}^D, y_n \in \mathbb{R}$

Assumptions:

- Underlying function is **linear**, $f_\theta(\mathbf{x}) = \sum_{d=1}^{D} \theta_d x_d = \theta^\intercal \mathbf{x}, \theta \in \mathbb{R}^D$

$$\begin{bmatrix} f_\theta(\mathbf{x}_1) \\ f_\theta(\mathbf{x}_2) \\ \vdots \\ f_\theta(\mathbf{x}_N) \end{bmatrix} = \begin{bmatrix} \theta^\intercal \mathbf{x}_1 \\ \theta^\intercal \mathbf{x}_2 \\ \vdots \\ \theta^\intercal \mathbf{x}_N \end{bmatrix} = X\theta$$

- Due to measurement noise, observed y is a noisy version of $f(\mathbf{x})$

Likelihood: $p(\mathcal{D} \mid \theta) = \prod_n \mathcal{N}(y_n; f(x_n), \sigma^2) = \prod_n \mathcal{N}(y_n; x_n^\intercal \theta, \sigma^2)$

Factorise across data points

Mean = linear mapping

Measurement noise
Constant across data points

Maximum likelihood, $\operatorname{argmax}_\theta \, p(\mathcal{D} \mid \theta)$ **is equiv. to least squares**

Linear regression - Maximum a posteriori

Training data N input, output pairs $\mathscr{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)\}, \mathbf{x}_n \in \mathbb{R}^D, y_n \in \mathbb{R}$

Assumptions:

- Underlying function is **linear**, $f_\theta(\mathbf{x}) = \sum_{d=1}^{D} \theta_d x_d = \theta^\intercal \mathbf{x}, \theta \in \mathbb{R}^D$

$$\begin{bmatrix} f_\theta(\mathbf{x}_1) \\ f_\theta(\mathbf{x}_2) \\ \vdots \\ f_\theta(\mathbf{x}_N) \end{bmatrix} = \begin{bmatrix} \theta^\intercal \mathbf{x}_1 \\ \theta^\intercal \mathbf{x}_2 \\ \vdots \\ \theta^\intercal \mathbf{x}_N \end{bmatrix} = X\theta$$

- Due to measurement noise, observed y is a noisy version of $f(\mathbf{x})$

Linear regression - Maximum a posteriori

Training data N input, output pairs $\mathcal{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)\}, \mathbf{x}_n \in \mathbb{R}^D, y_n \in \mathbb{R}$

Assumptions:

- Underlying function is **linear**, $f_\theta(\mathbf{x}) = \sum_{d=1}^{D} \theta_d x_d = \theta^\intercal \mathbf{x}, \theta \in \mathbb{R}^D$

$$\begin{bmatrix} f_\theta(\mathbf{x}_1) \\ f_\theta(\mathbf{x}_2) \\ \vdots \\ f_\theta(\mathbf{x}_N) \end{bmatrix} = \begin{bmatrix} \theta^\intercal \mathbf{x}_1 \\ \theta^\intercal \mathbf{x}_2 \\ \vdots \\ \theta^\intercal \mathbf{x}_N \end{bmatrix} = X\theta$$

- Due to measurement noise, observed y is a noisy version of $f(\mathbf{x})$

Likelihood: $p(\mathcal{D} \mid \theta) = \prod_n \mathcal{N}(y_n; f(x_n), \sigma^2) = \prod_n \mathcal{N}(y_n; x_n^\intercal \theta, \sigma^2)$

Prior: $p(\theta) = \mathcal{N}(\theta; \mathbf{0}, \sigma_o^2 \mathrm{I}) = \prod_d \mathcal{N}(\theta_d; 0, \sigma_o^2)$

Factorise across dimensions — Zero mean — Same variance for all dimensions

Linear regression - Maximum a posteriori

Training data N input, output pairs $\mathcal{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)\}, \mathbf{x}_n \in \mathbb{R}^D, y_n \in \mathbb{R}$

Assumptions:

- Underlying function is **linear**, $f_\theta(\mathbf{x}) = \sum_{d=1}^{D} \theta_d x_d = \theta^\intercal \mathbf{x}, \theta \in \mathbb{R}^D$

$$\begin{bmatrix} f_\theta(\mathbf{x}_1) \\ f_\theta(\mathbf{x}_2) \\ \vdots \\ f_\theta(\mathbf{x}_N) \end{bmatrix} = \begin{bmatrix} \theta^\intercal \mathbf{x}_1 \\ \theta^\intercal \mathbf{x}_2 \\ \vdots \\ \theta^\intercal \mathbf{x}_N \end{bmatrix} = X\theta$$

- Due to measurement noise, observed y is a noisy version of $f(\mathbf{x})$

Likelihood: $p(\mathcal{D} \mid \theta) = \prod_n \mathcal{N}(y_n; f(x_n), \sigma^2) = \prod_n \mathcal{N}(y_n; x_n^\intercal \theta, \sigma^2)$

Prior: $p(\theta) = \mathcal{N}(\theta; \mathbf{0}, \sigma_o^2 \mathrm{I}) = \prod_d \mathcal{N}(\theta_d; 0, \sigma_o^2)$

Factorise across dimensions — Zero mean — Same variance for all dimensions

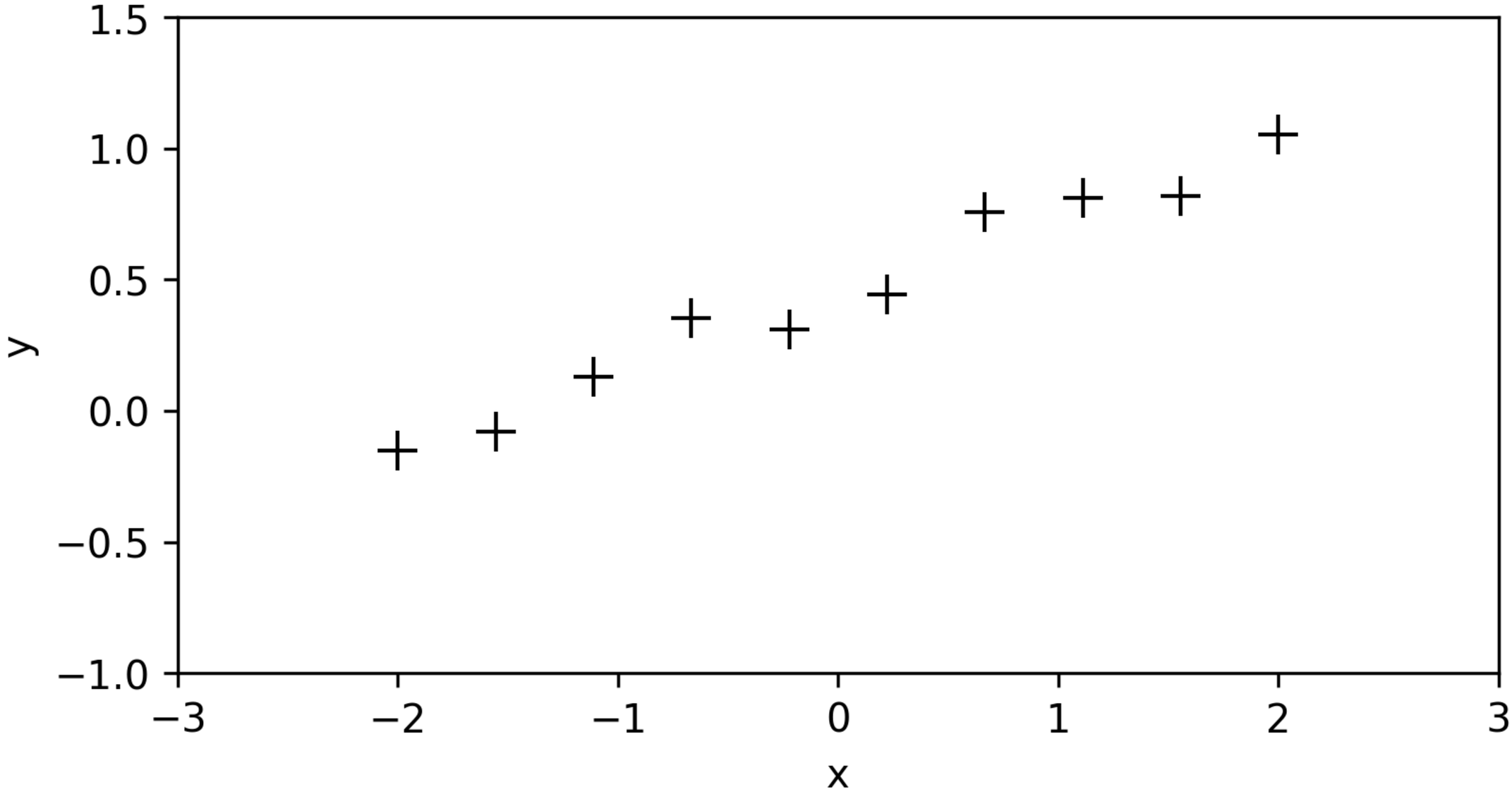
Maximum a posteriori (MAP), $\operatorname{argmax}_\theta \, p(\mathcal{D} \mid \theta) p(\theta)$ **is equiv. to regularised least squares**

Overview

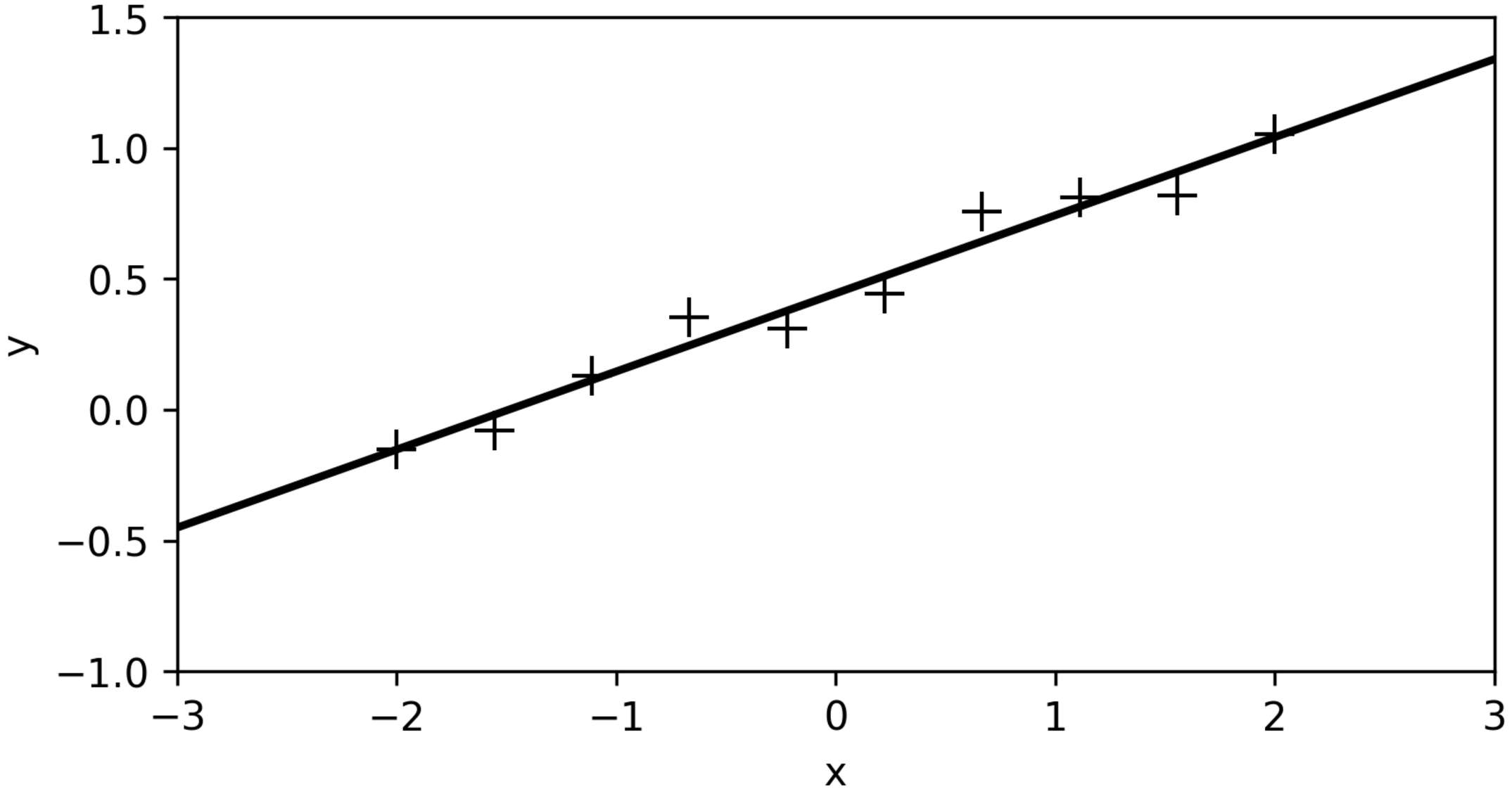
1. Recap: Linear regression
2. **From least squares to binary logistic classification**
3. From binary to multi-class classification
4. Evaluation
5. Linearly separable data - perceptron algorithm

Linear regression - an example

Linear regression - an example

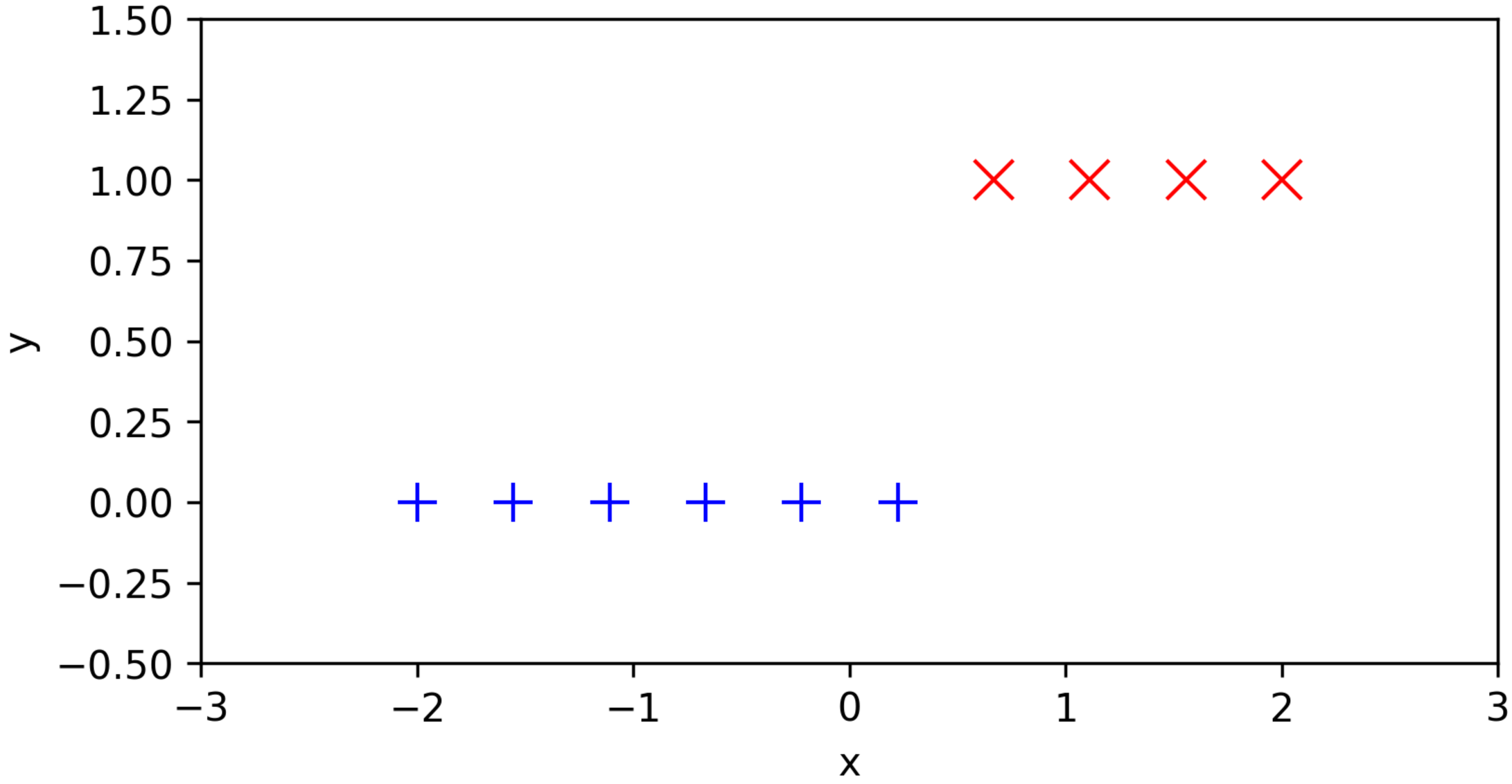


Linear regression - an example

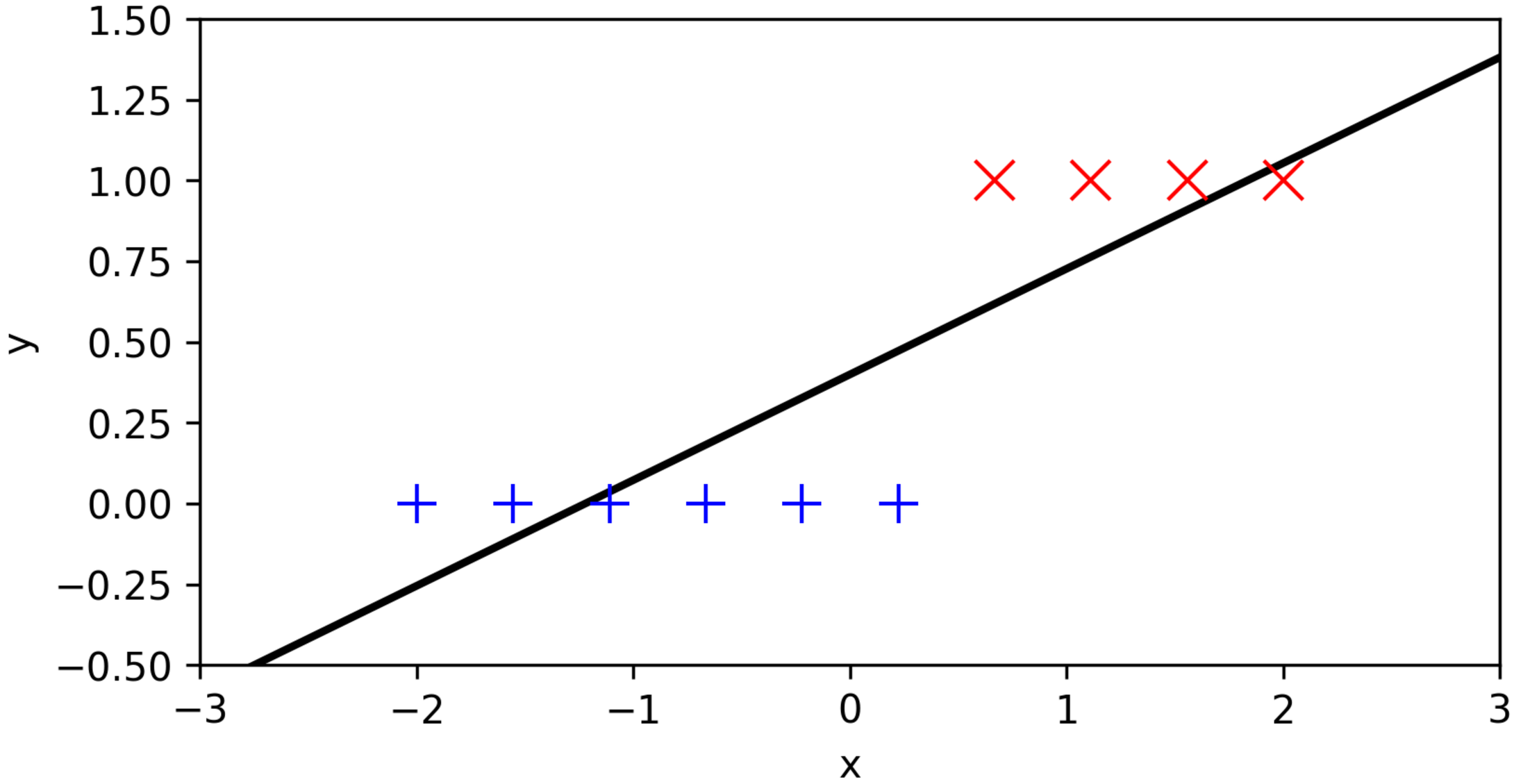


Can we use least squares to do binary classification? An example

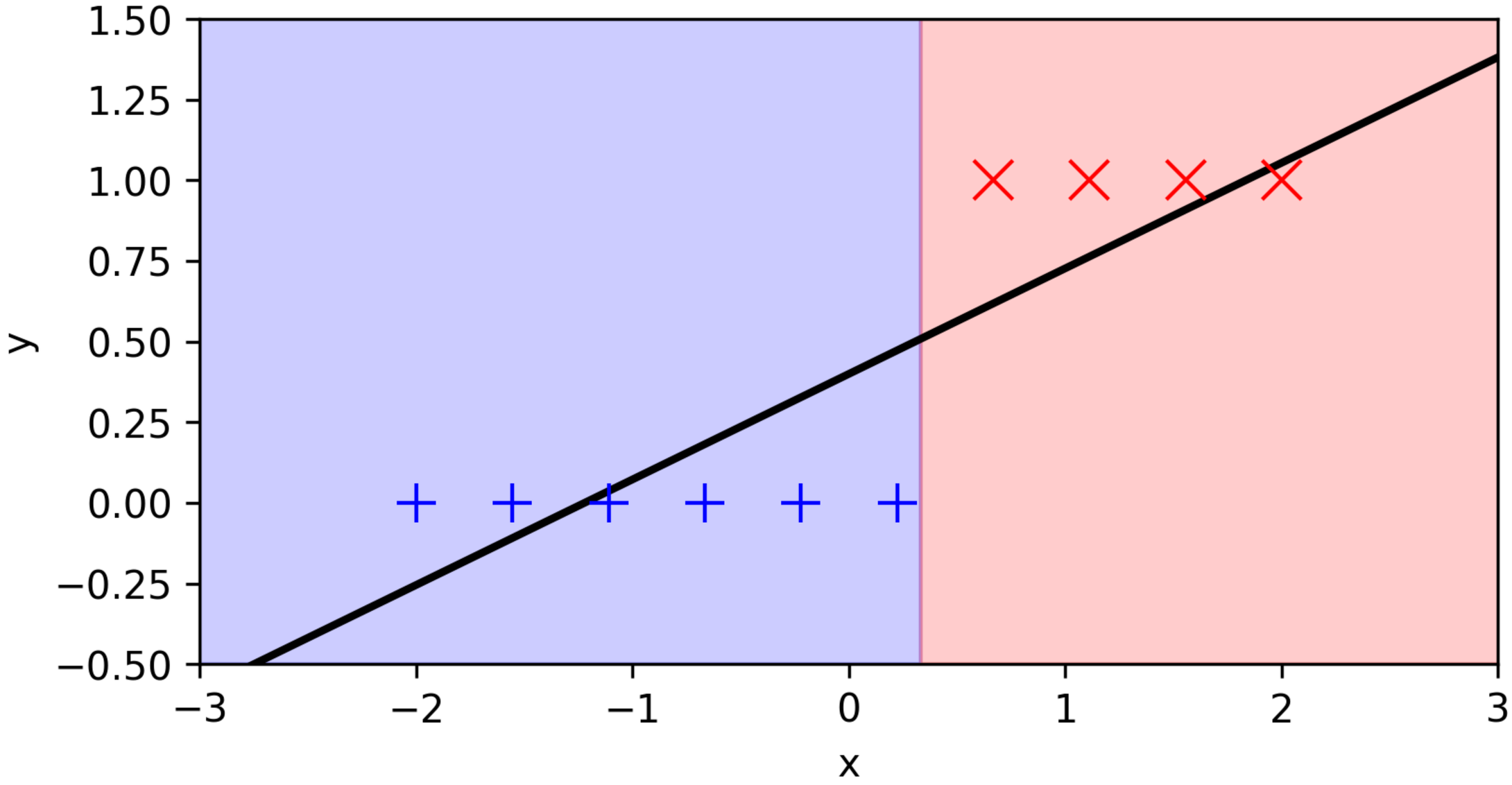
Can we use least squares to do binary classification? An example



Can we use least squares to do binary classification? An example

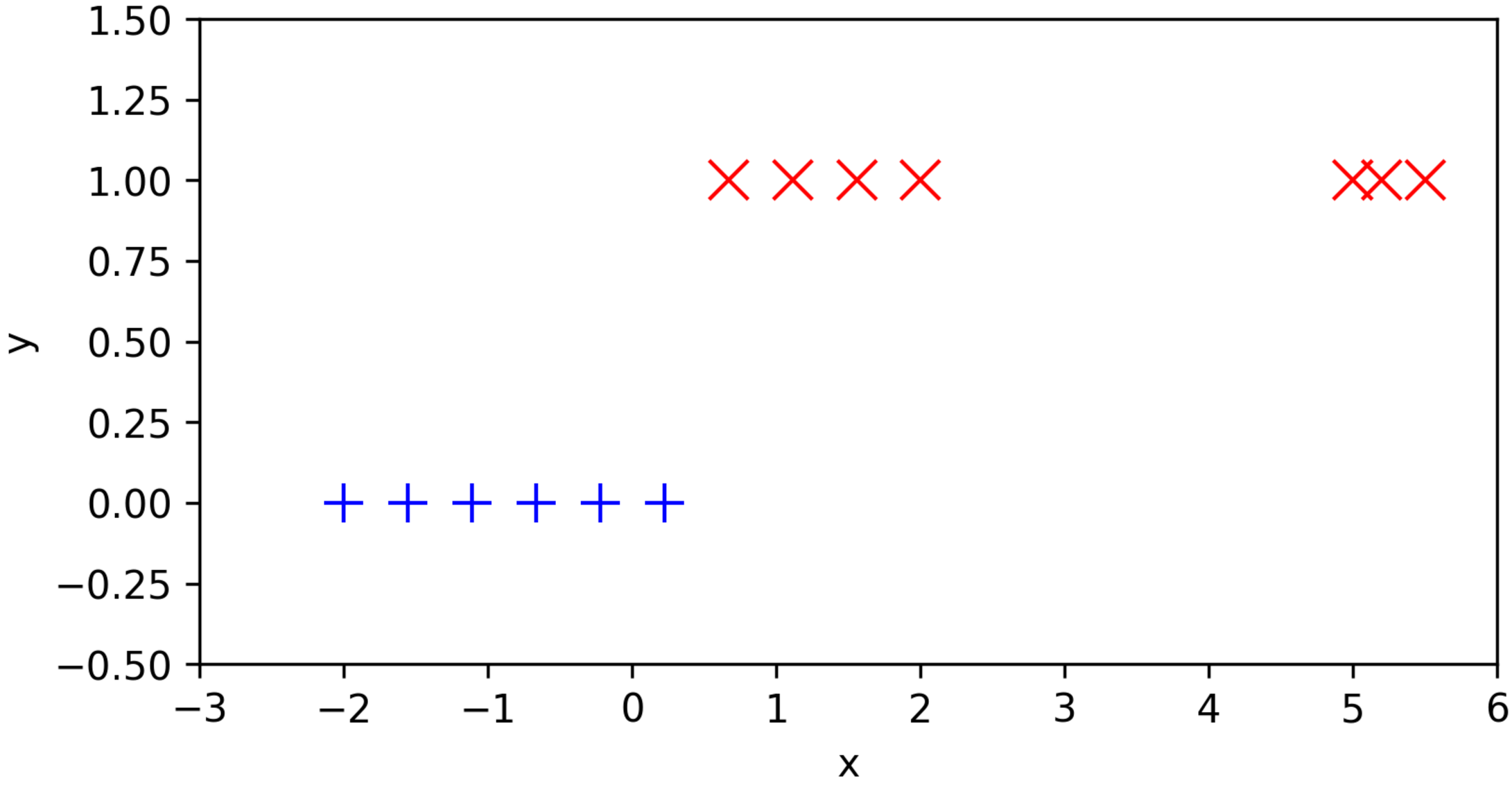


Can we use least squares to do binary classification? An example

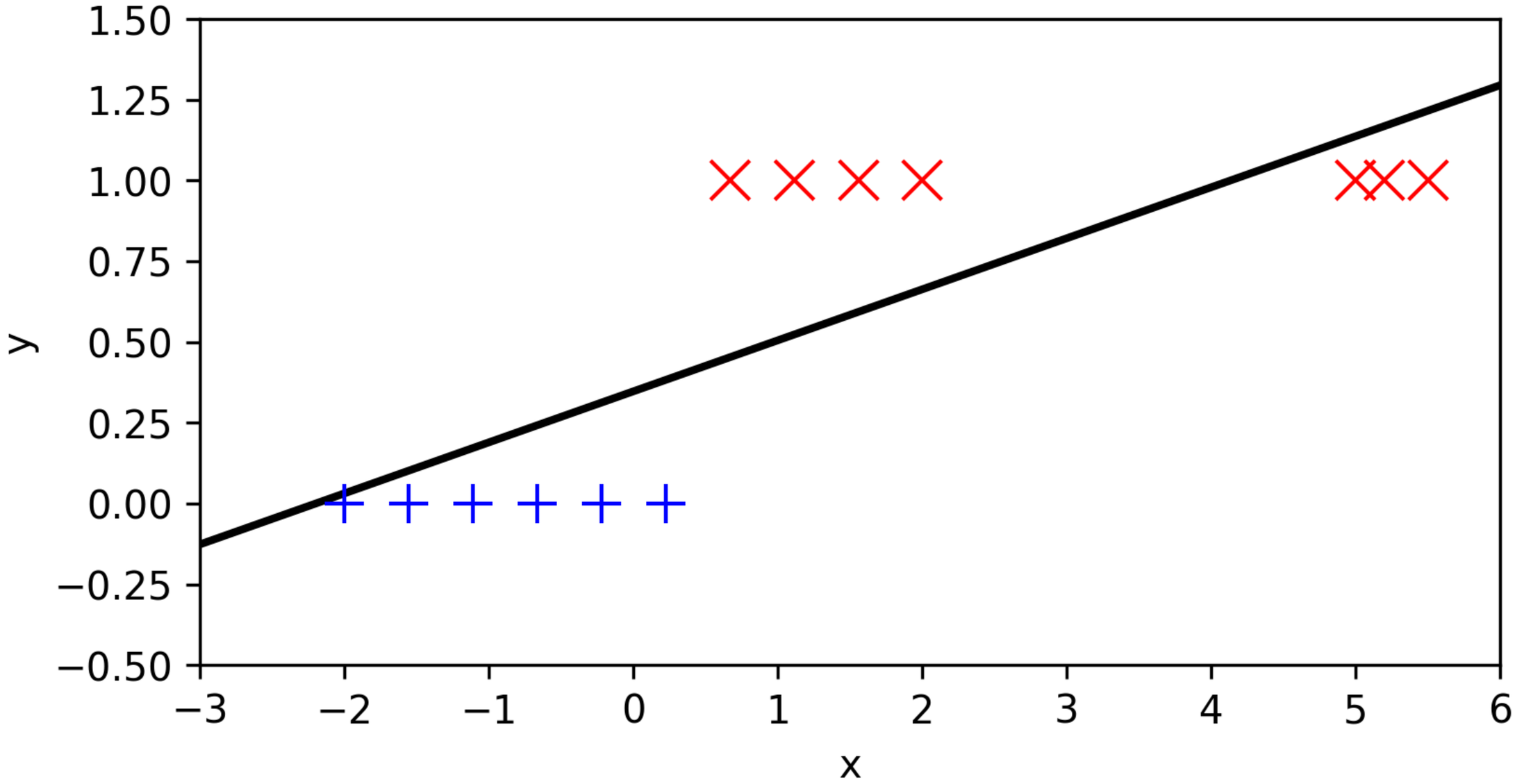


Can we use least squares to do binary classification? One more example

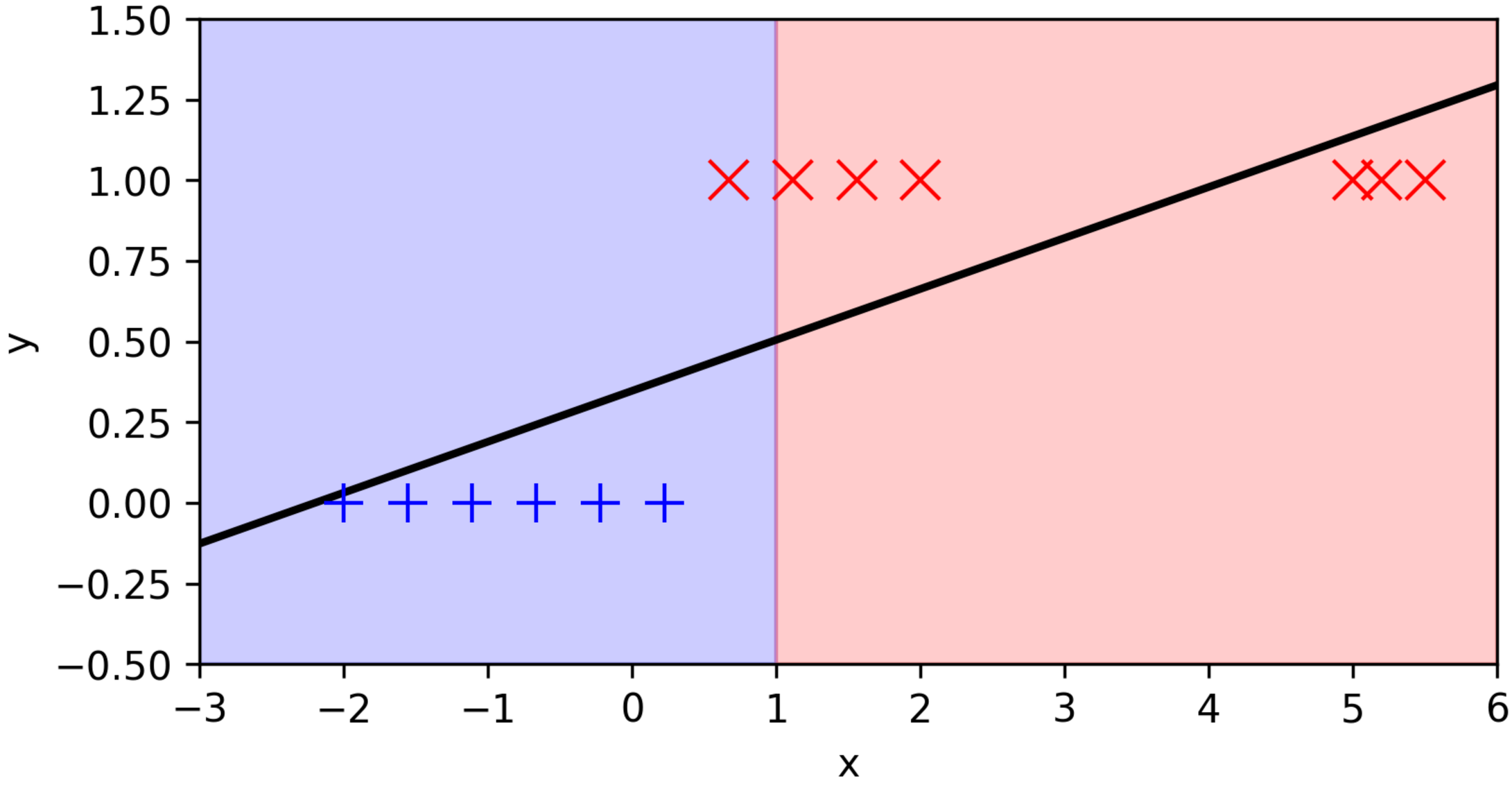
Can we use least squares to do binary classification? One more example



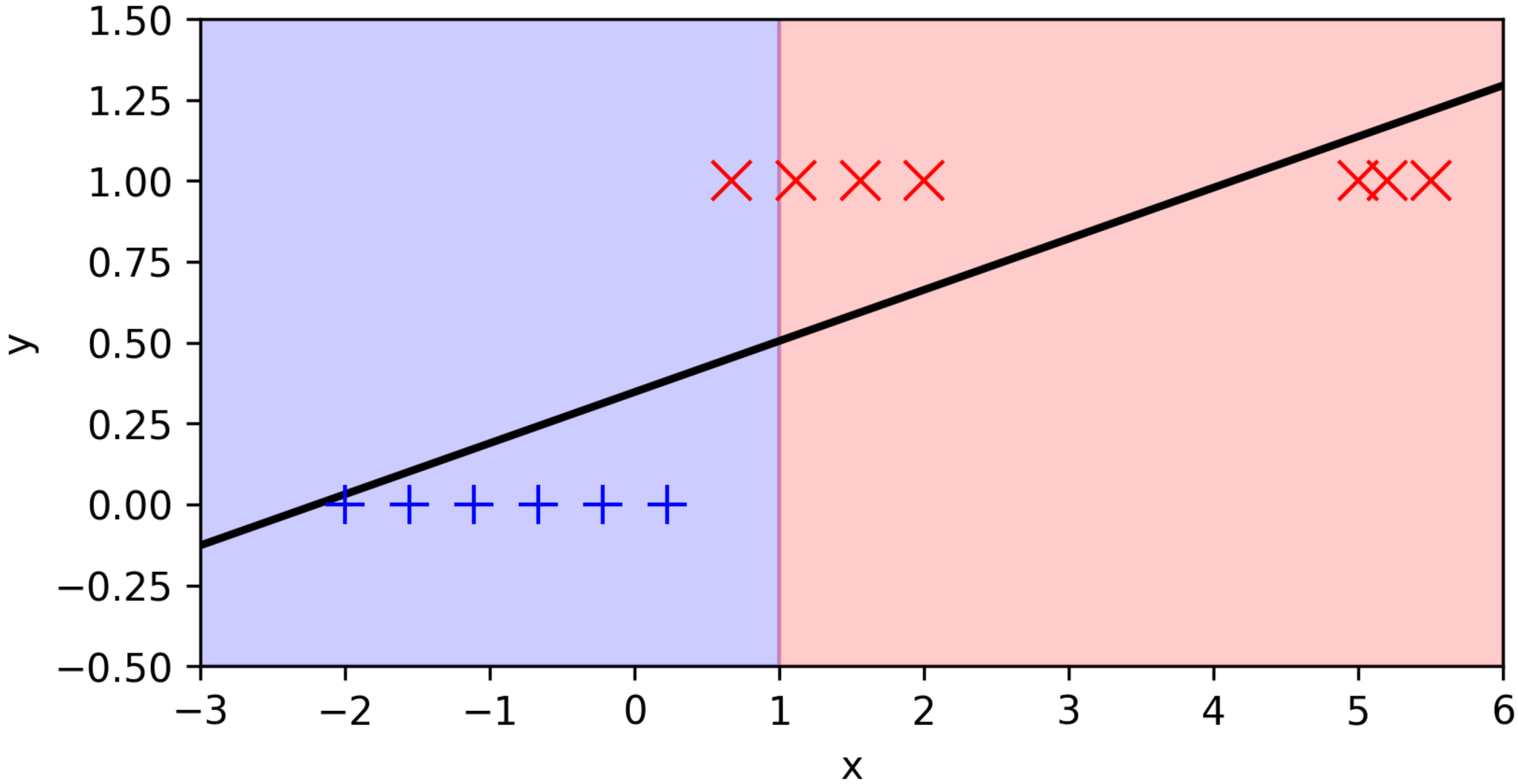
Can we use least squares to do binary classification? One more example



Can we use least squares to do binary classification? One more example



Can we use least squares to do binary classification? One more example



Least squares for classification: Lack robustness. Also fundamentally unsound - remember the Gaussian assumption!

Binary logistic classification - problem set-up

Binary logistic classification - problem set-up

Training data N input, output pairs $\mathscr{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)\}, \mathbf{x}_n \in \mathbb{R}^D,\ y_n \in \{0,1\}$

Each row is a data

Each column is a feature

$$X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1D} \\ x_{21} & x_{22} & \cdots & x_{2D} \\ \vdots & \vdots & \vdots & \vdots \\ x_{N1} & x_{N2} & \cdots & x_{ND} \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

One value per data point
0 or 1

Binary logistic classification - problem set-up

Training data N input, output pairs $\mathscr{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)\}, \mathbf{x}_n \in \mathbb{R}^D,\ y_n \in \{0,1\}$

Each row is a data

Each column is a feature

$$X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1D} \\ x_{21} & x_{22} & \cdots & x_{2D} \\ \vdots & \vdots & \vdots & \vdots \\ x_{N1} & x_{N2} & \cdots & x_{ND} \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

One value per data point
0 or 1

We construct a **classifier** that learns the class probabilities: $p(y = c \,|\, x)$

Binary logistic classification - problem set-up

Training data N input, output pairs $\mathscr{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)\}, \mathbf{x}_n \in \mathbb{R}^D,\ y_n \in \{0,1\}$

Each row is a data

Each column is a feature

$$X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1D} \\ x_{21} & x_{22} & \cdots & x_{2D} \\ \vdots & \vdots & \vdots & \vdots \\ x_{N1} & x_{N2} & \cdots & x_{ND} \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

One value per data point
0 or 1

We construct a **classifier** that learns the class probabilities: $p(y = c \,|\, x)$

For binary classification, by laws of probabilities: $p(y = 0 \,|\, x) + p(y = 1 \,|\, x) = ?$

Binary logistic classification - problem set-up

Training data N input, output pairs $\mathscr{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)\}, \mathbf{x}_n \in \mathbb{R}^D$, $y_n \in \{0,1\}$

Each row is a data

Each column is a feature

$$X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1D} \\ x_{21} & x_{22} & \cdots & x_{2D} \\ \vdots & \vdots & \vdots & \vdots \\ x_{N1} & x_{N2} & \cdots & x_{ND} \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

One value per data point
0 or 1

We construct a **classifier** that learns the class probabilities: $p(y = c \,|\, x)$

For binary classification, by laws of probabilities: $p(y = 0 \,|\, x) + p(y = 1 \,|\, x) = ?$ 1

Binary logistic classification - problem set-up

Training data N input, output pairs $\mathscr{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)\}, \mathbf{x}_n \in \mathbb{R}^D,\ y_n \in \{0,1\}$

Each row is a data

Each column is a feature

$$X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1D} \\ x_{21} & x_{22} & \cdots & x_{2D} \\ \vdots & \vdots & \vdots & \vdots \\ x_{N1} & x_{N2} & \cdots & x_{ND} \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

One value per data point
0 or 1

We construct a **classifier** that learns the class probabilities: $p(y = c \,|\, x)$

For binary classification, by laws of probabilities: $p(y = 0 \,|\, x) + p(y = 1 \,|\, x) = ?$ 1

Imagine: $p(y = 1 \,|\, x) = g_\theta(x)$ then $p(y = 0 \,|\, x) = 1 - g_\theta(x)$. **Constraint**: $0 \leq g_\theta(x) \leq 1, \forall x$

Binary logistic classification - problem set-up

Training data N input, output pairs $\mathscr{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)\}, \mathbf{x}_n \in \mathbb{R}^D,\ y_n \in \{0,1\}$

Each row is a data

Each column is a feature

$$X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1D} \\ x_{21} & x_{22} & \cdots & x_{2D} \\ \vdots & \vdots & \vdots & \vdots \\ x_{N1} & x_{N2} & \cdots & x_{ND} \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

One value per data point
0 or 1

We construct a **classifier** that learns the class probabilities: $p(y = c \,|\, x)$

For binary classification, by laws of probabilities: $p(y = 0 \,|\, x) + p(y = 1 \,|\, x) = ?$ 1

Imagine: $p(y = 1 \,|\, x) = g_\theta(x)$ then $p(y = 0 \,|\, x) = 1 - g_\theta(x)$. **Constraint**: $0 \leq g_\theta(x) \leq 1, \forall x$

Prediction with threshold = 0.5: $y = 1$ if $g_\theta(x) \geq 0.5$ and $y = 0$ if $g_\theta(x) < 0.5$.

Binary logistic classification - logistic function

Binary logistic classification - logistic function

Reminder: Linear regression: $f_\theta(\mathbf{x}) = \sum_{d=1}^{D} \theta_d x_d = \theta^\intercal \mathbf{x}, \theta \in \mathbb{R}^D$

$$\begin{bmatrix} f_\theta(\mathbf{x}_1) \\ f_\theta(\mathbf{x}_2) \\ \vdots \\ f_\theta(\mathbf{x}_N) \end{bmatrix} = \begin{bmatrix} \theta^\intercal \mathbf{x}_1 \\ \theta^\intercal \mathbf{x}_2 \\ \vdots \\ \theta^\intercal \mathbf{x}_N \end{bmatrix} = X\theta$$

Binary logistic classification - logistic function

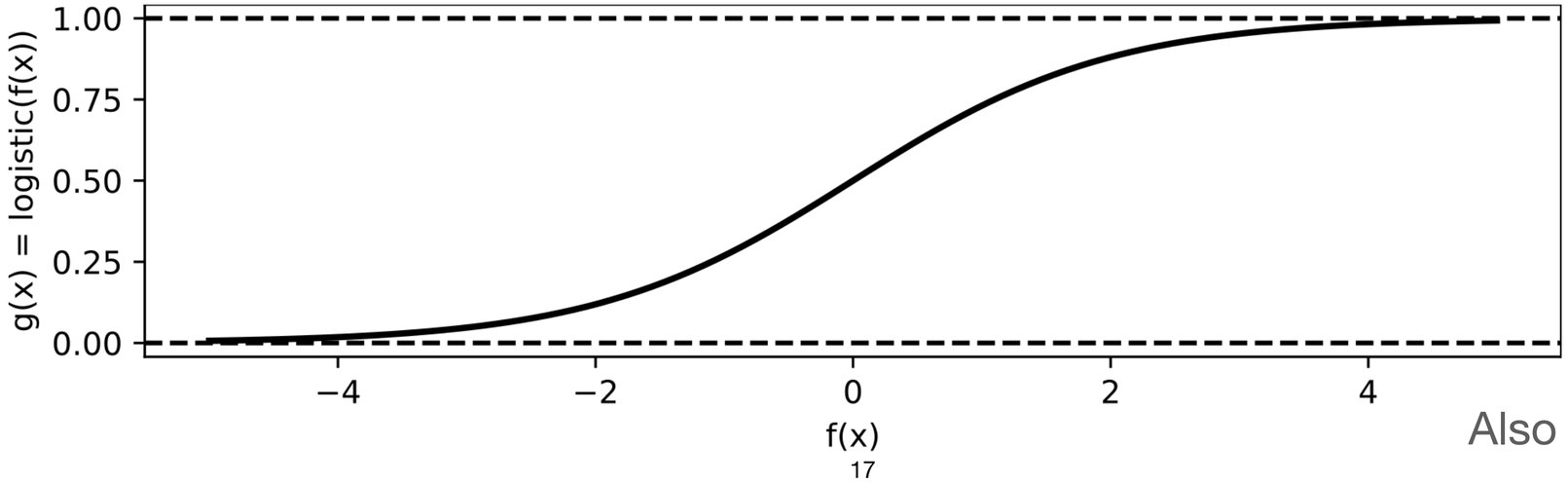
Reminder: Linear regression: $f_\theta(\mathbf{x}) = \sum_{d=1}^{D} \theta_d x_d = \theta^\intercal \mathbf{x}, \theta \in \mathbb{R}^D$

$$\begin{bmatrix} f_\theta(\mathbf{x}_1) \\ f_\theta(\mathbf{x}_2) \\ \vdots \\ f_\theta(\mathbf{x}_N) \end{bmatrix} = \begin{bmatrix} \theta^\intercal \mathbf{x}_1 \\ \theta^\intercal \mathbf{x}_2 \\ \vdots \\ \theta^\intercal \mathbf{x}_N \end{bmatrix} = X\theta$$

Want: $0 \leq g_\theta(x) \leq 1, \forall x$

Idea: 'squash' $f_\theta(x)$ through a *logistic sigmoid* function $g_\theta(x) = \sigma(f_\theta(x))$, where

$$\sigma(z) = \frac{\exp(z)}{1 + \exp(z)} = \frac{1}{1 + \exp(-z)}$$



Also called sigmoid

Binary logistic classification - maximum likelihood

Binary logistic classification - maximum likelihood

We can write down the ***likelihood*** of parameters given one data point:

$$p(y_n \,|\, x_n, \theta) = \begin{cases} g_\theta(x_n), & \text{if } y_n = 1 \\ 1 - g_\theta(x_n), & \text{if } y_n = 0 \end{cases}$$

Binary logistic classification - maximum likelihood

We can write down the ***likelihood*** of parameters given one data point:

$$p(y_n \,|\, x_n, \theta) = \begin{cases} g_\theta(x_n), & \text{if } y_n = 1 \\ 1 - g_\theta(x_n), & \text{if } y_n = 0 \end{cases}$$

We want to *maximise* the likelihood or *minimise* the negative log-likelihood:

$$\begin{aligned} \mathscr{L}_n(\theta) &= -\log p(y_n \,|\, x_n, \beta) = \begin{cases} -\log(g_\theta(x_n)), & \text{if } y_n = 1 \\ -\log(1 - g_\theta(x_n)), & \text{if } y_n = 0 \end{cases} \\ &= -y_n \log(g_\theta(x_n)) - (1 - y_n)\log(1 - g_\theta(x_n)) \end{aligned}$$

Binary logistic classification - maximum likelihood

We can write down the ***likelihood*** of parameters given one data point:

$$p(y_n \,|\, x_n, \theta) = \begin{cases} g_\theta(x_n), & \text{if } y_n = 1 \\ 1 - g_\theta(x_n), & \text{if } y_n = 0 \end{cases}$$

We want to *maximise* the likelihood or *minimise* the negative log-likelihood:

$$\begin{aligned} \mathscr{L}_n(\theta) &= -\log p(y_n \,|\, x_n, \beta) = \begin{cases} -\log(g_\theta(x_n)), & \text{if } y_n = 1 \\ -\log(1 - g_\theta(x_n)), & \text{if } y_n = 0 \end{cases} \\ &= -y_n \log(g_\theta(x_n)) - (1 - y_n)\log(1 - g_\theta(x_n)) \end{aligned}$$

For N datapoints:

$$\mathscr{L}(\theta) = \frac{1}{N} \sum_{n=1}^{N} -y_n \log(g_\theta(x_n)) - (1 - y_n)\log(1 - g_\theta(x_n))$$

Often called the **binary cross-entropy loss**

Binary logistic classification - gradients

Binary logistic classification - gradients

Objective: $\mathscr{L}(\theta) = \frac{1}{N}\sum_{n=1}^{N} -y_n \log(g_\theta(x_n)) - (1-y_n)\log(1-g_\theta(x_n))$

Notes: $g_\theta(x) = \sigma(f_\theta(x))$, σ is a logistic sigmoid, and $f_\theta(\mathbf{x}) = \sum_{d=1}^{D} \theta_d x_d = \theta^\intercal \mathbf{x}, \theta \in \mathbb{R}^D$

Show: $\frac{\mathrm{d}\mathscr{L}(\theta)}{\mathrm{d}\theta} = \frac{1}{N}\sum_{n=1}^{N} (g_\theta(x_n) - y_n)x_n^T$

Binary logistic classification - optimisation

Binary logistic classification - optimisation

We can use *gradient descent [Week 7]* to optimise $\mathscr{L}(\theta)$.

(i) Have some random starting point for θ

(ii) Update θ to decrease the loss function $\mathscr{L}(\theta)$, $\theta \leftarrow \theta - \gamma \left[\frac{\mathrm{d}\mathscr{L}(\theta)}{\mathrm{d}\theta}\right]^{\mathsf{T}}$

(iii) Repeat (ii) until reaching a minimum (convergence) [global minimum for logistic reg]

Regularised logistic classification

Similar to least squares, we want to *penalise* the amplitude of parameters by **regularisation:**

$$\mathscr{L}_\lambda(\theta) = \text{cross-entropy loss} + \lambda \|\theta\|_p^p$$

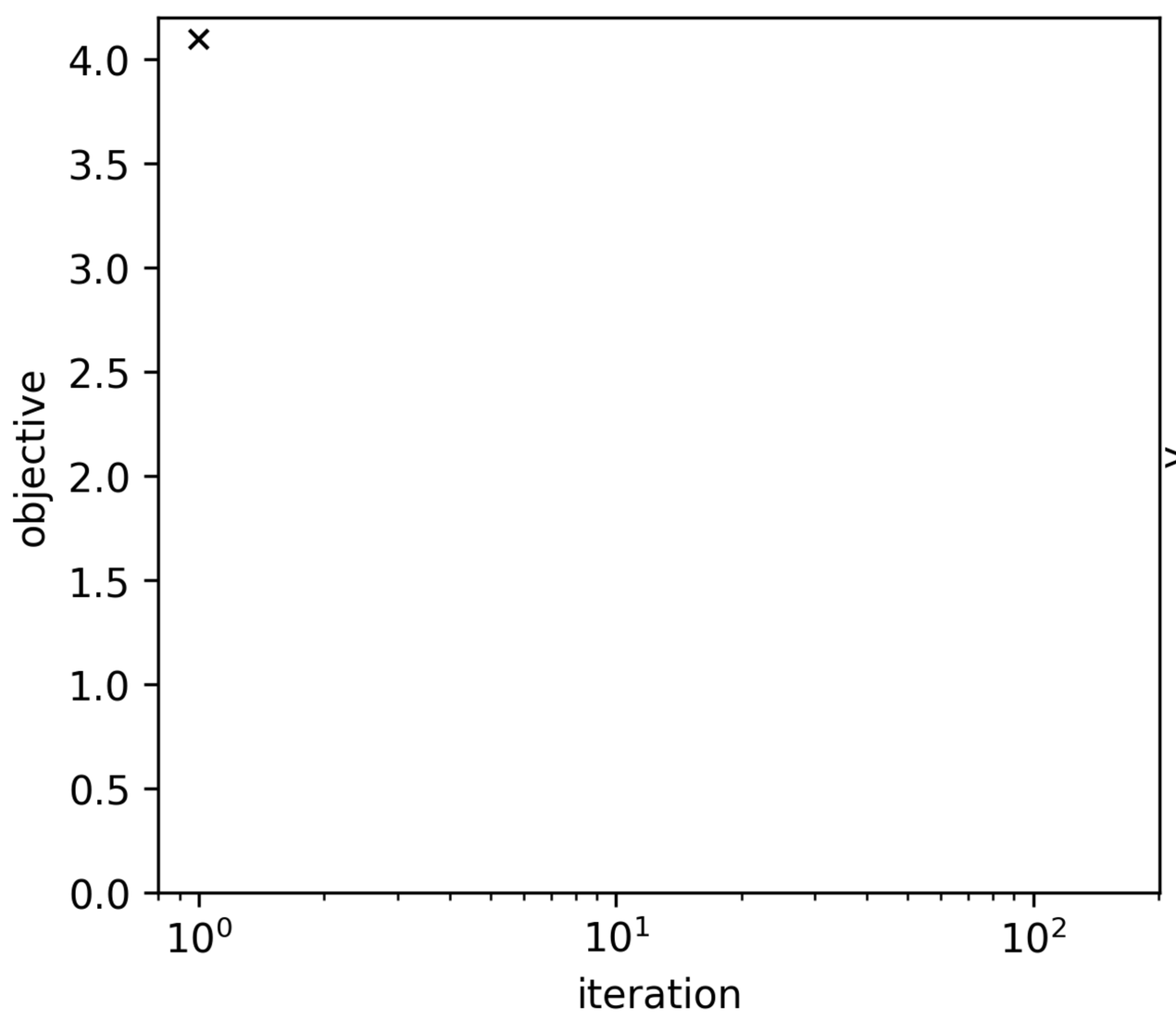
Data-fit
Lower = better fit

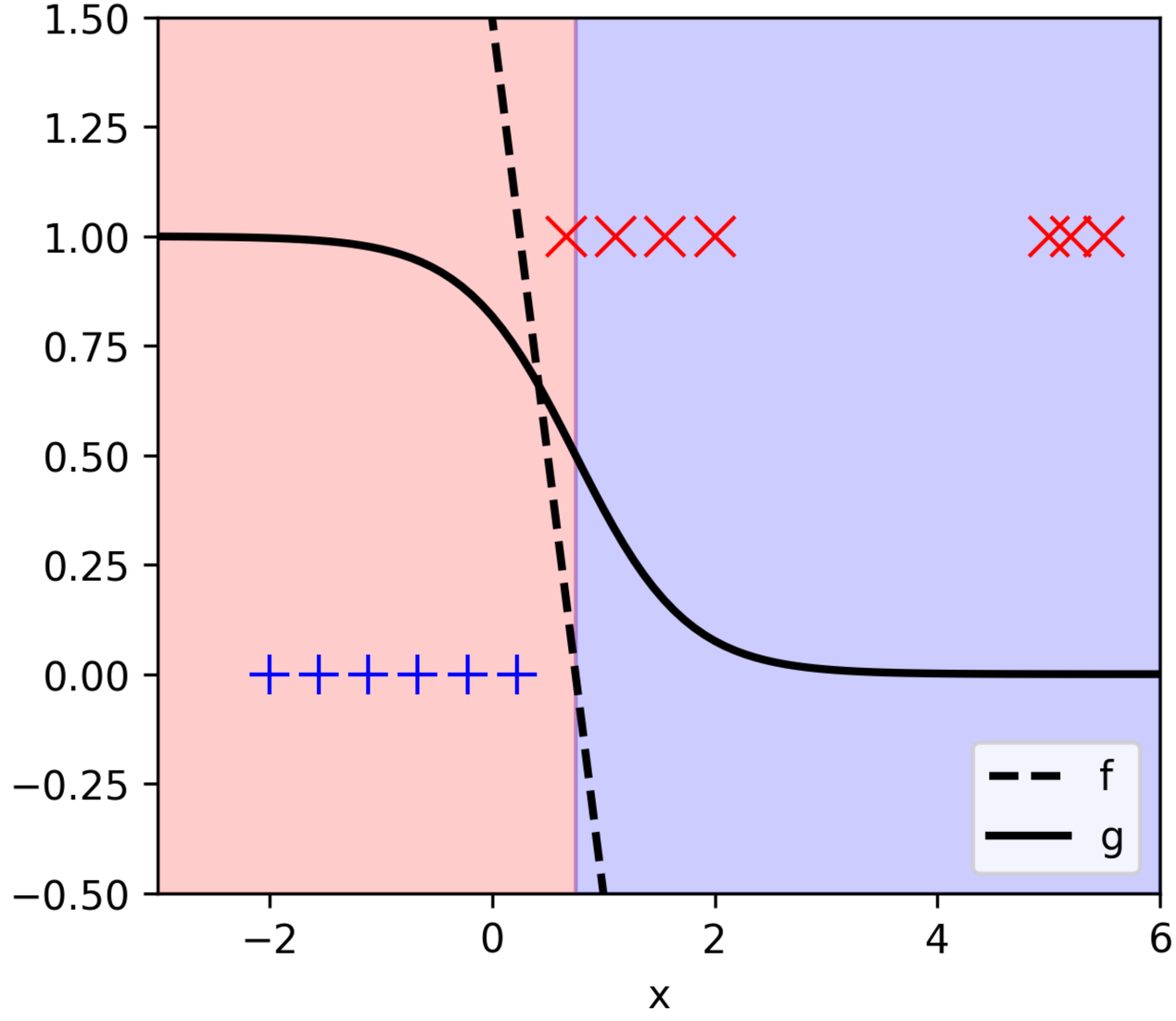
Regulariser
Lower = simpler model

Hyperparameter

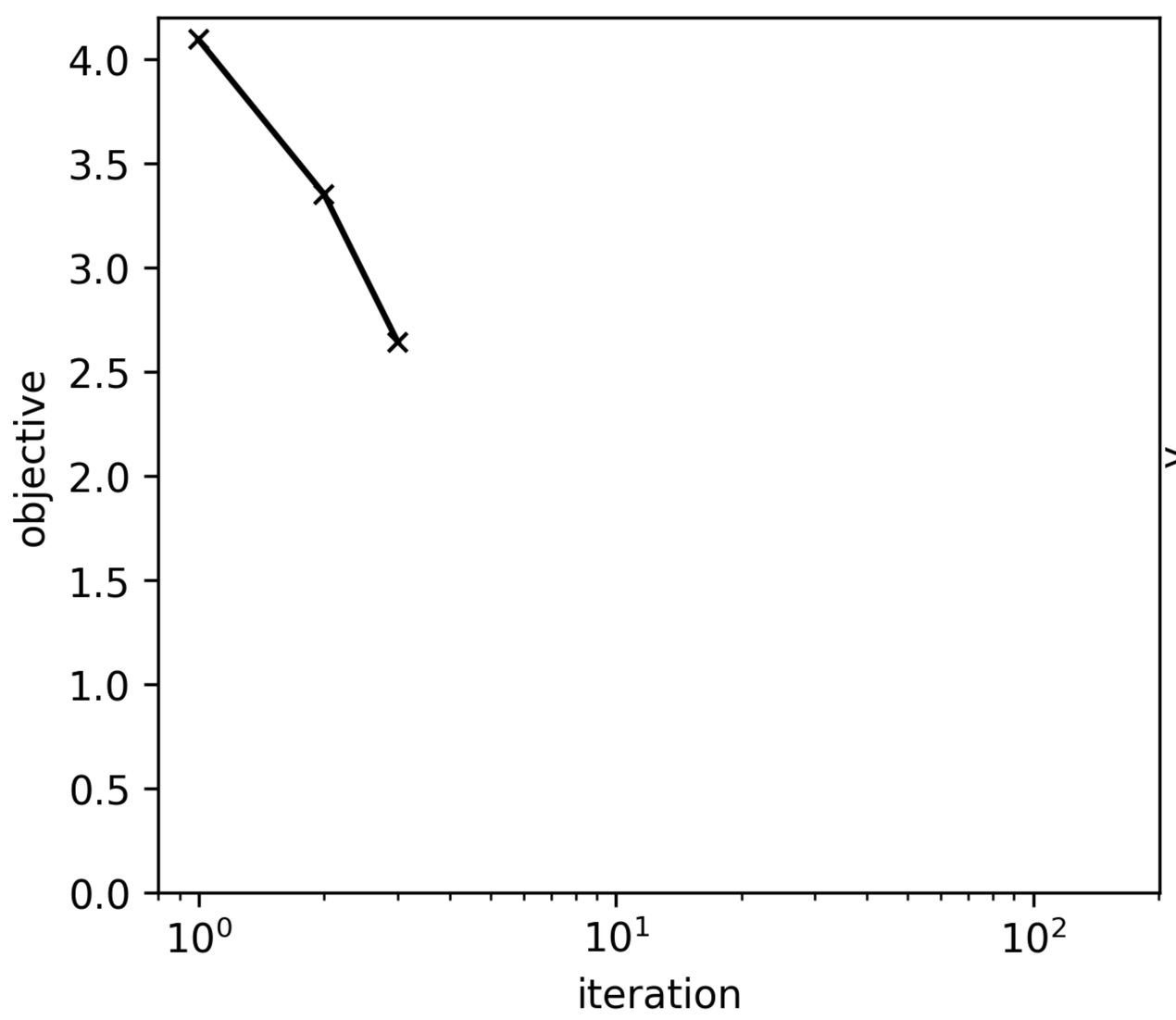
Binary logistic classification - example

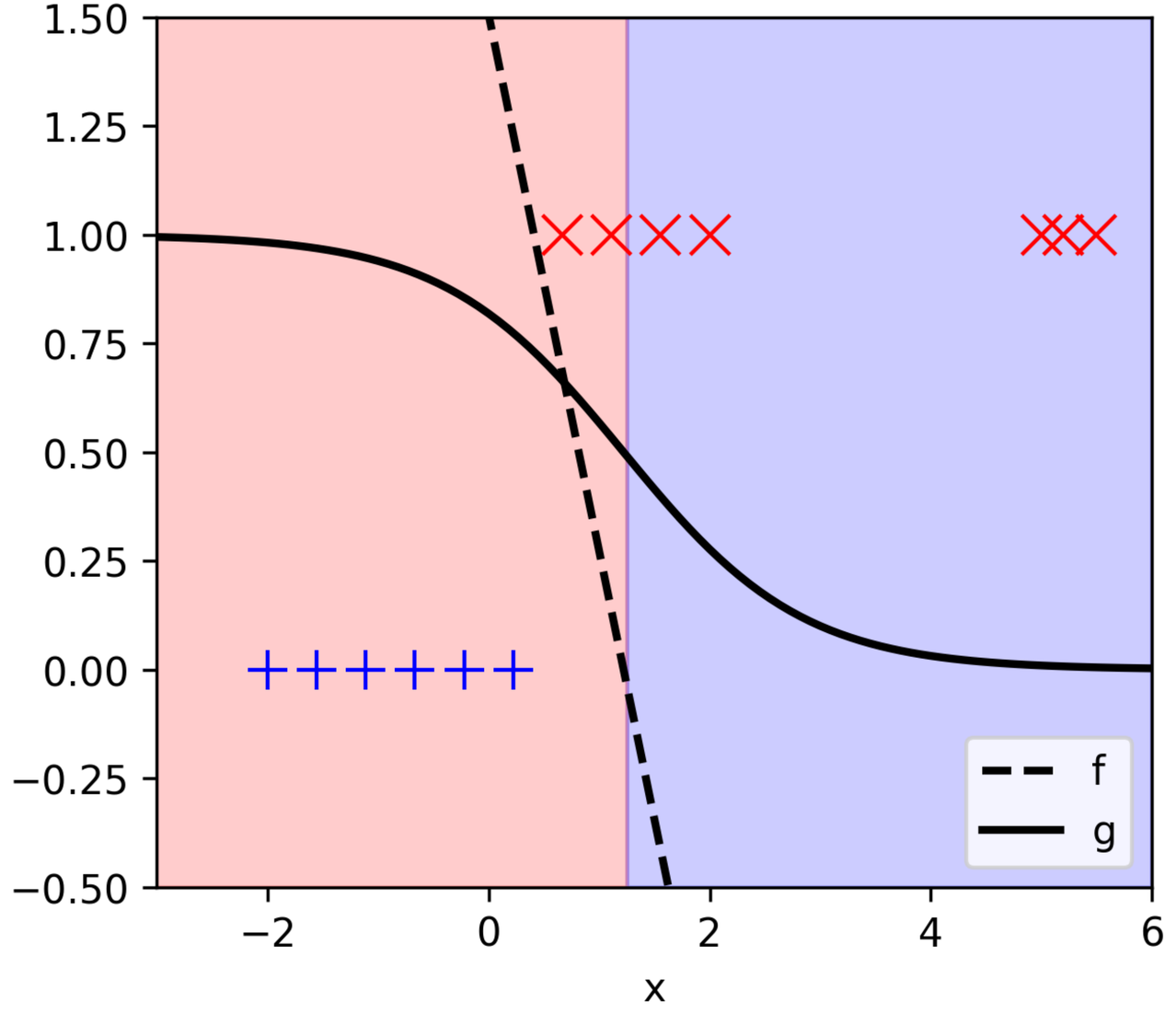
Binary logistic classification - example



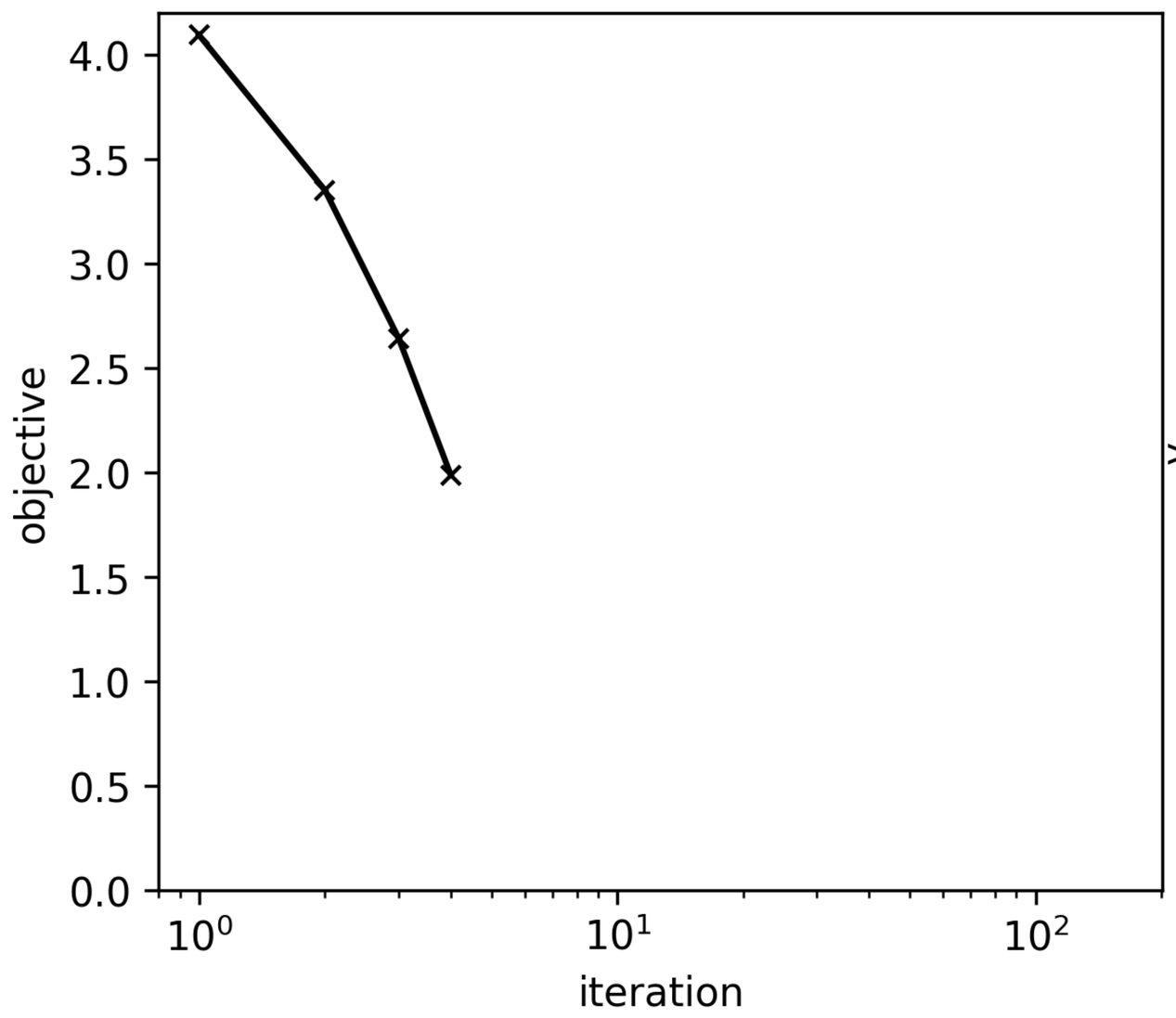


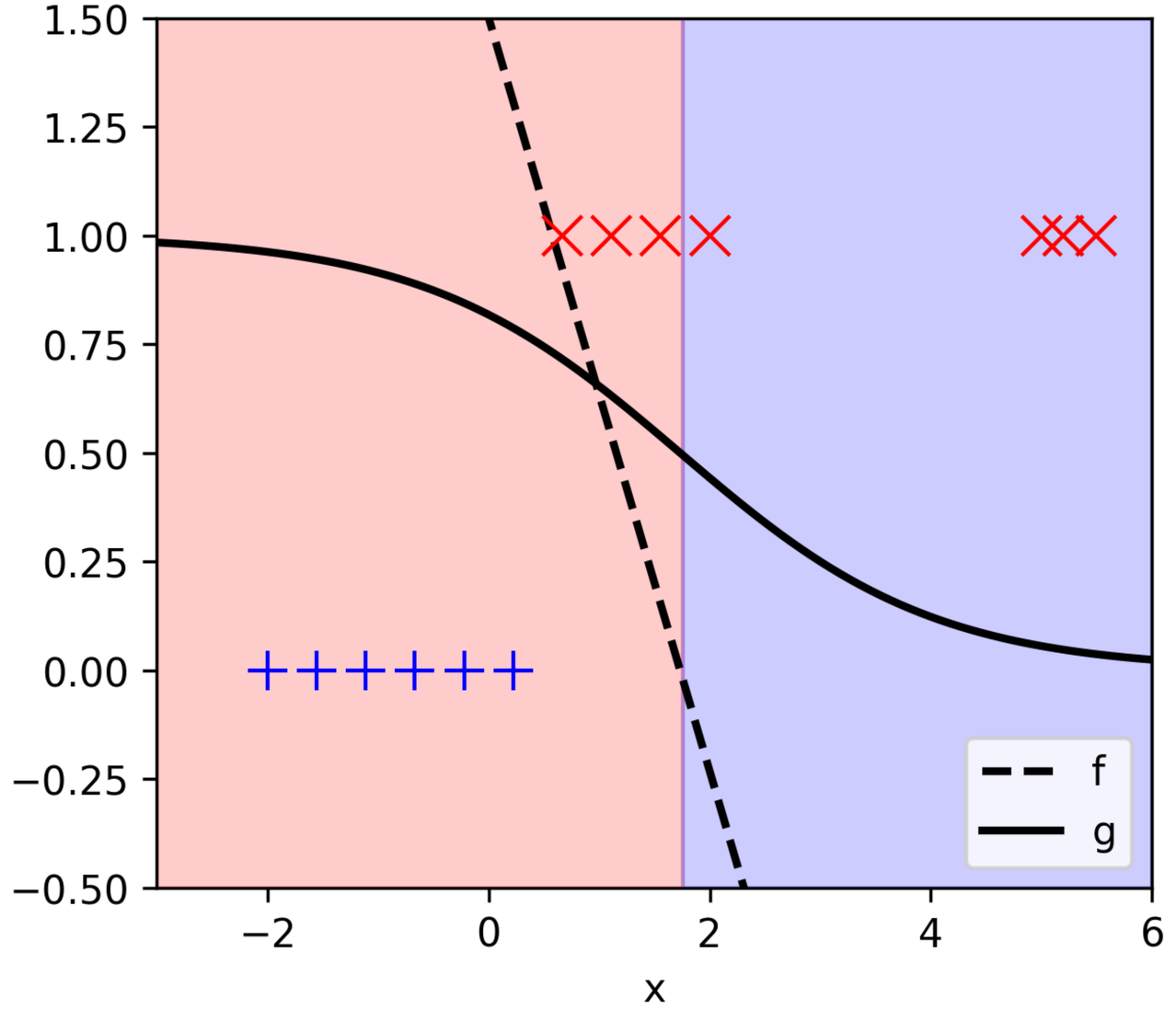
Binary logistic classification - example



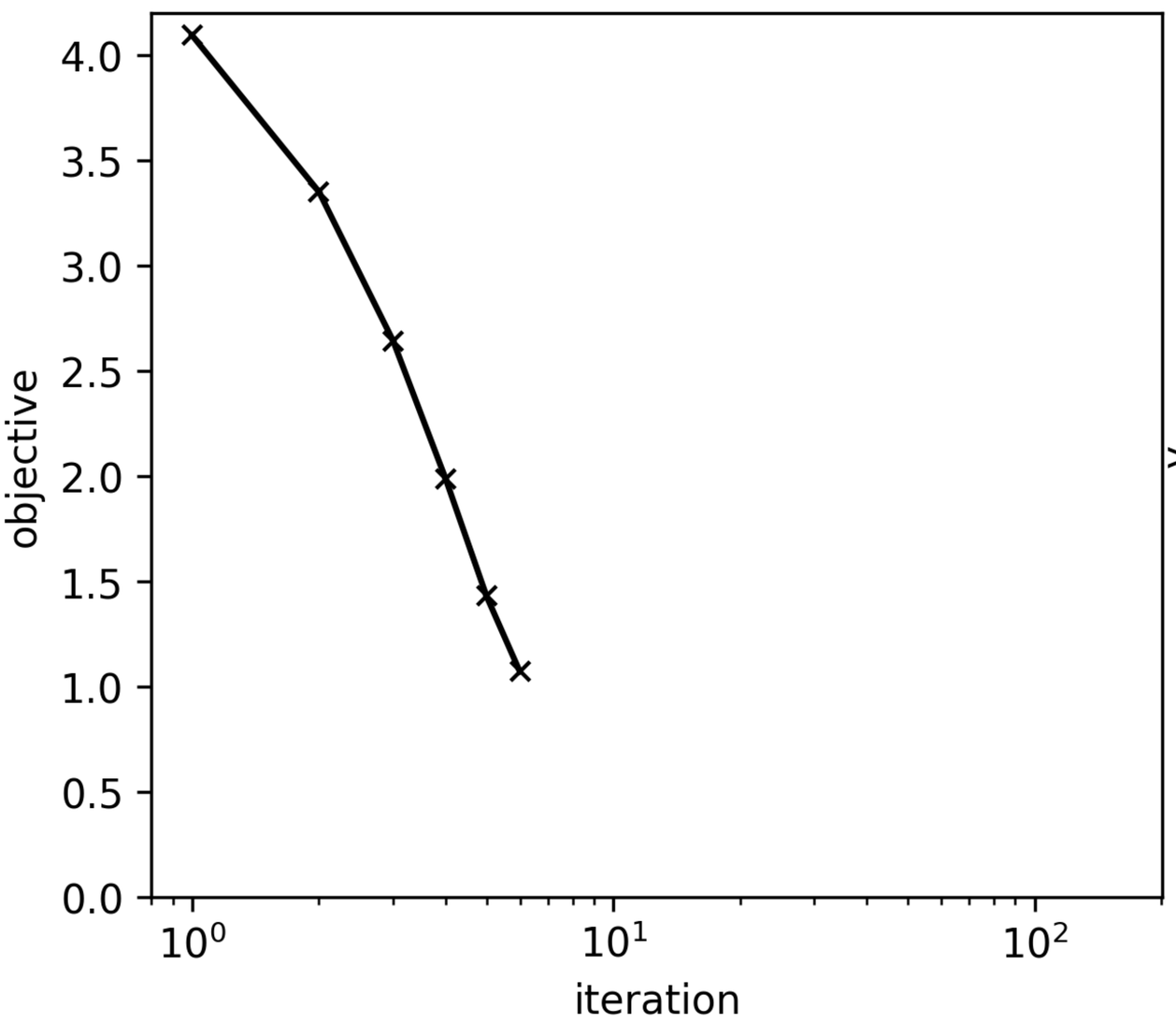


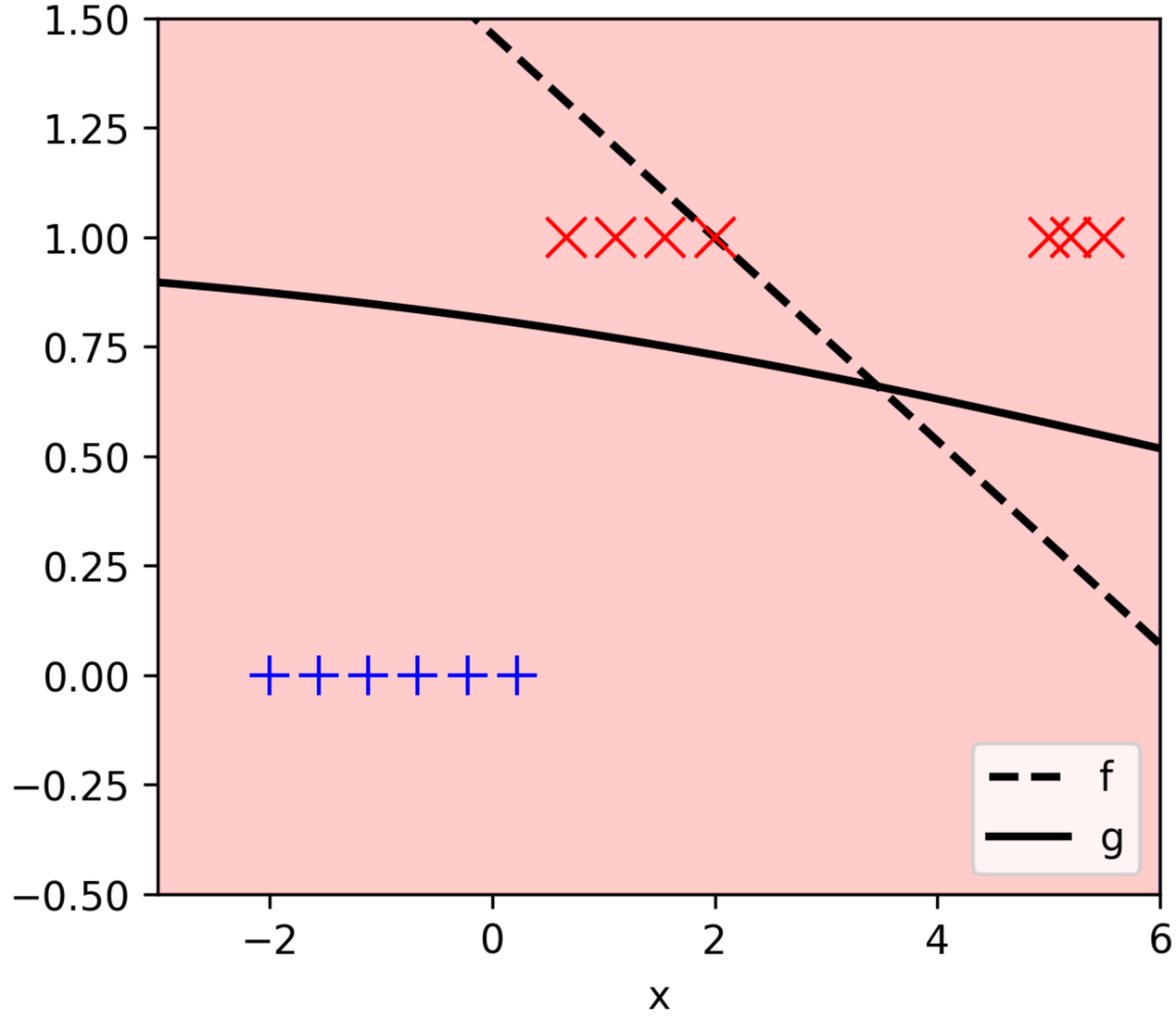
Binary logistic classification - example



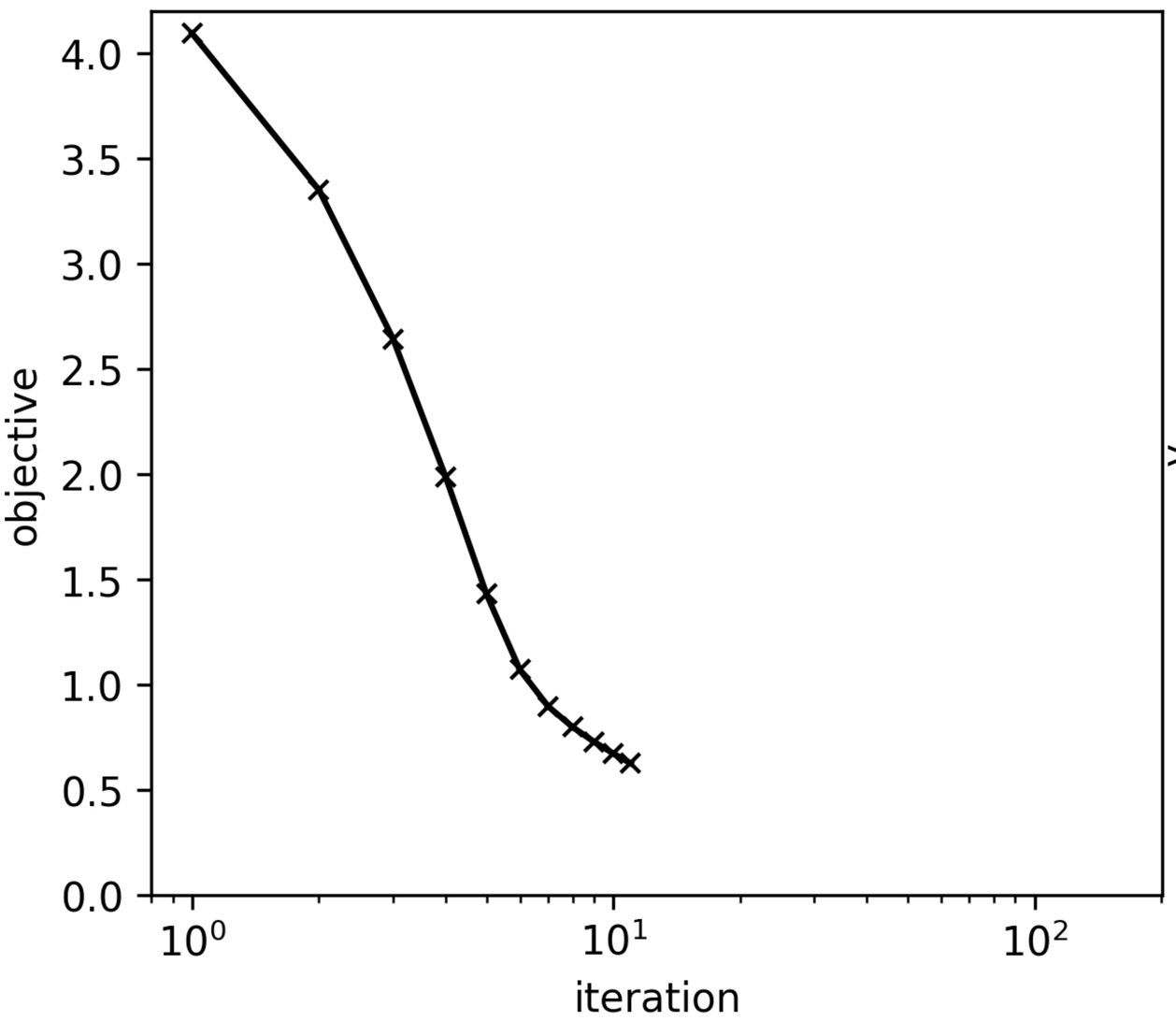


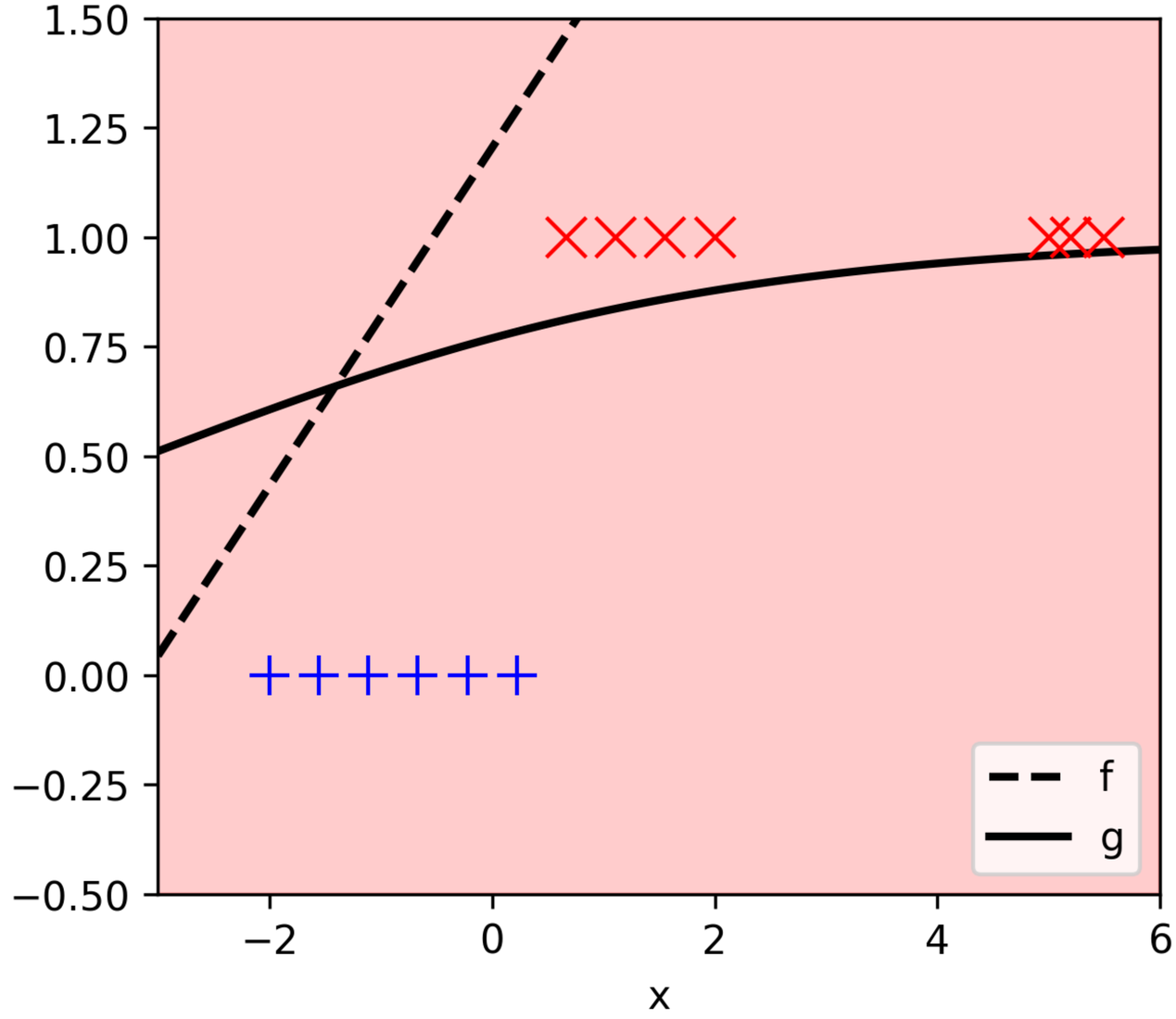
Binary logistic classification - example



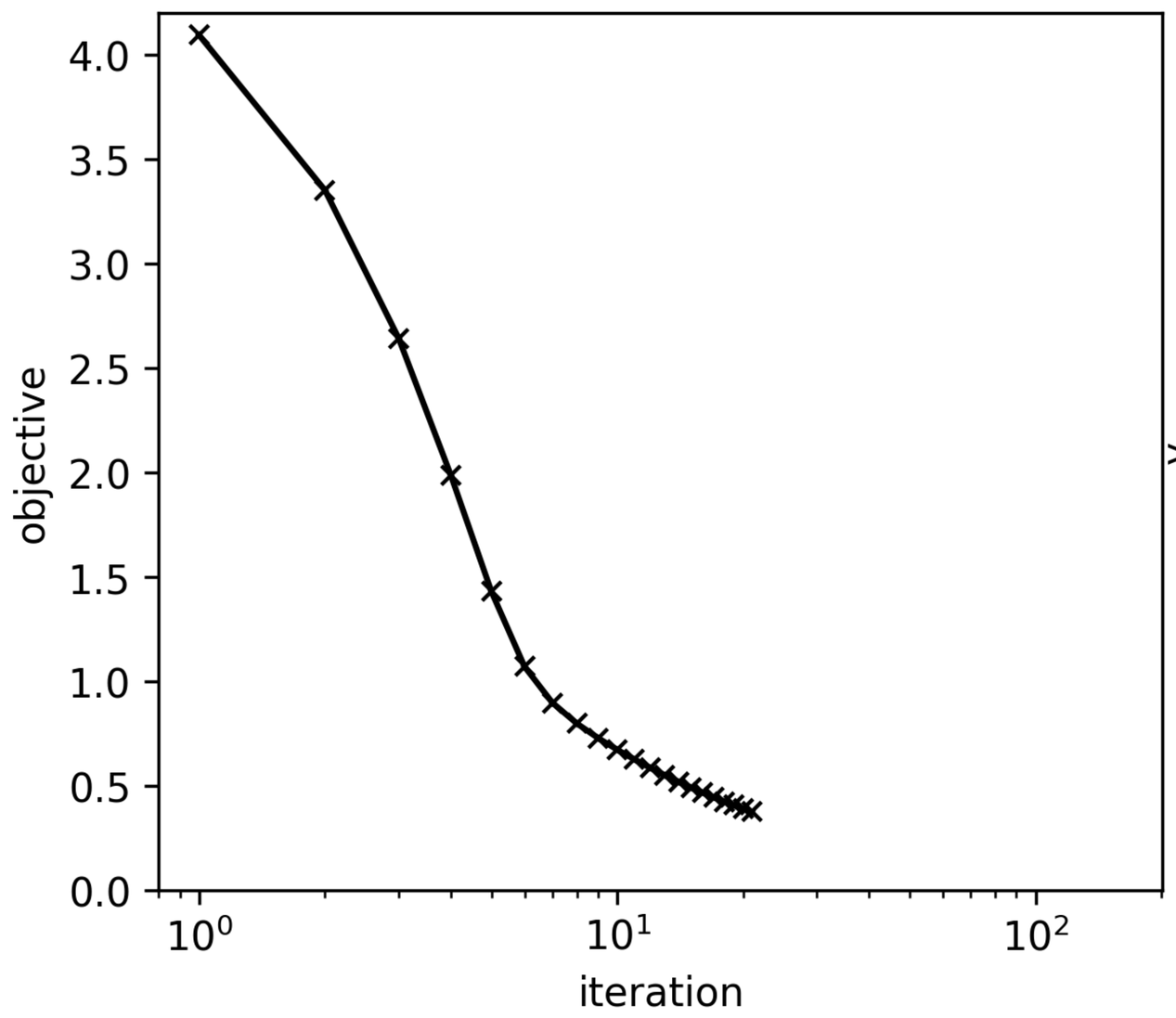


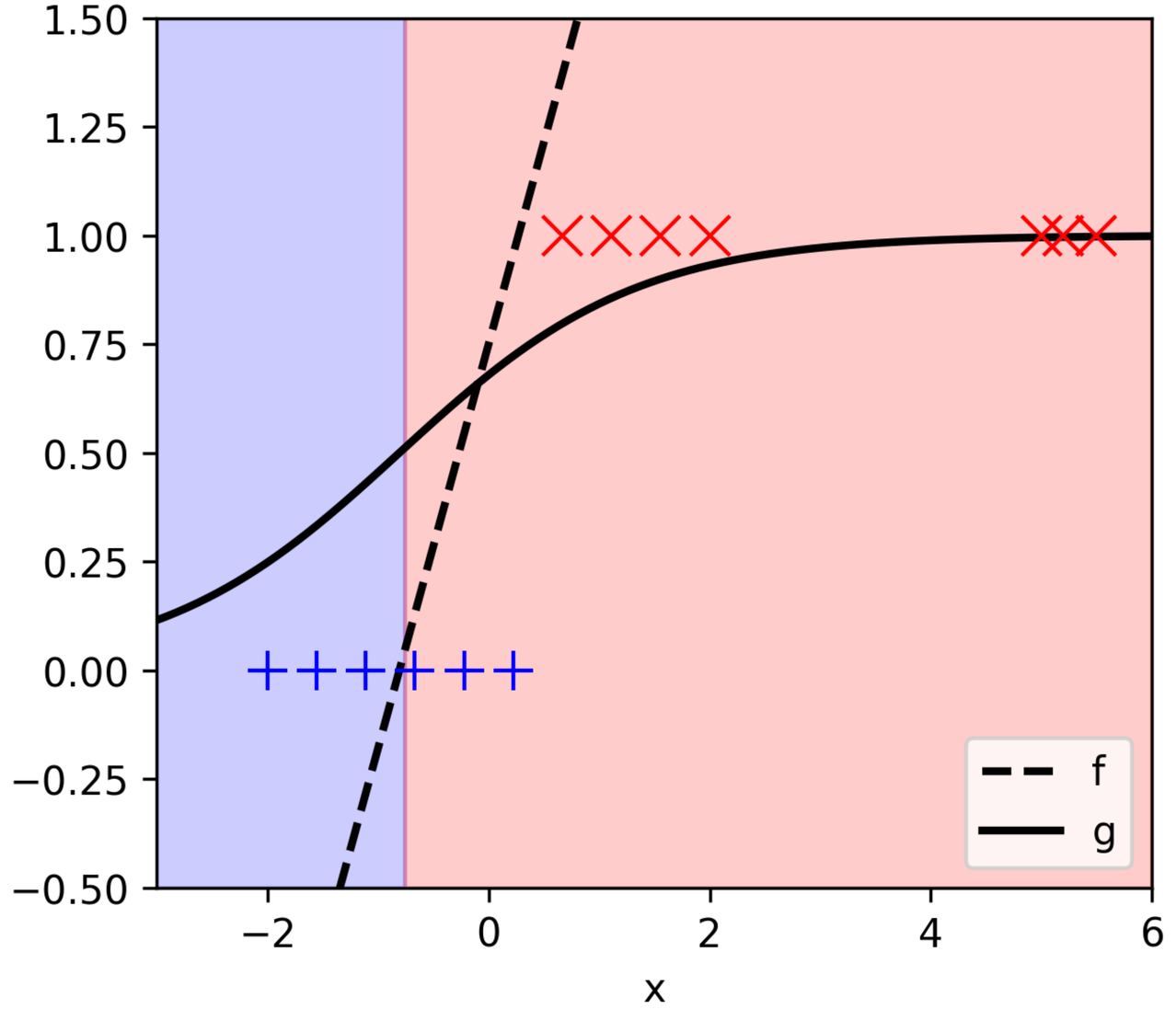
Binary logistic classification - example



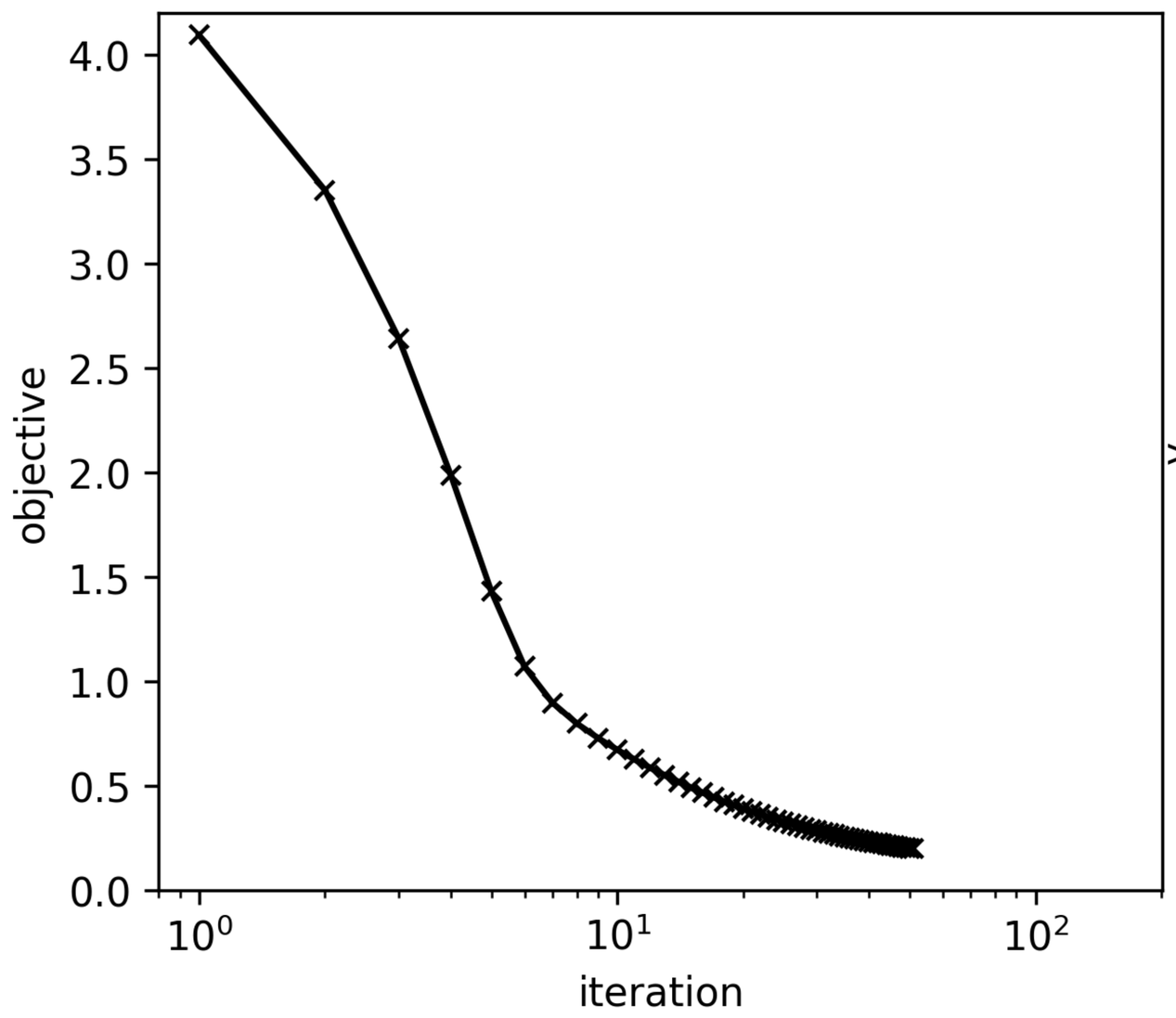


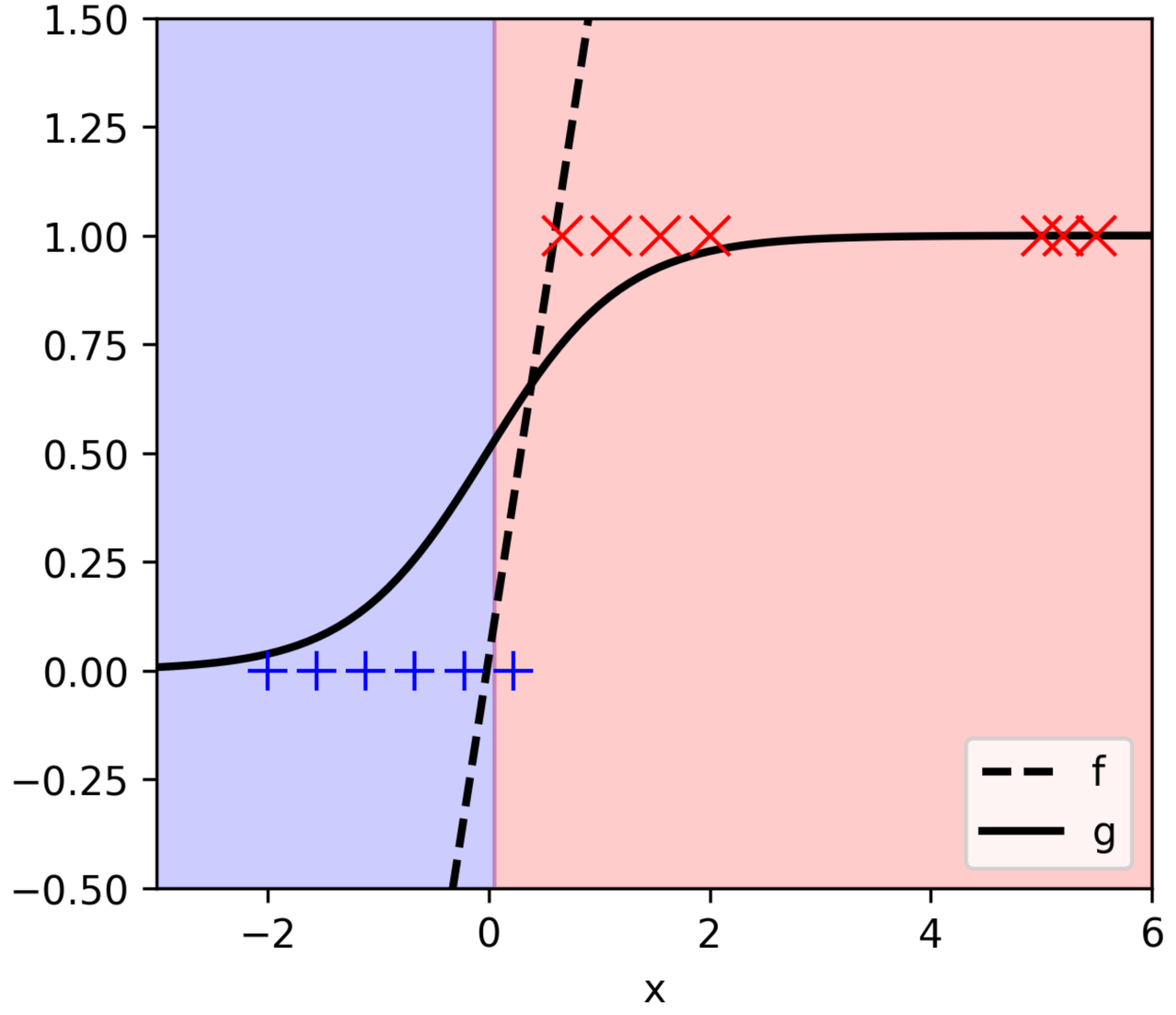
Binary logistic classification - example



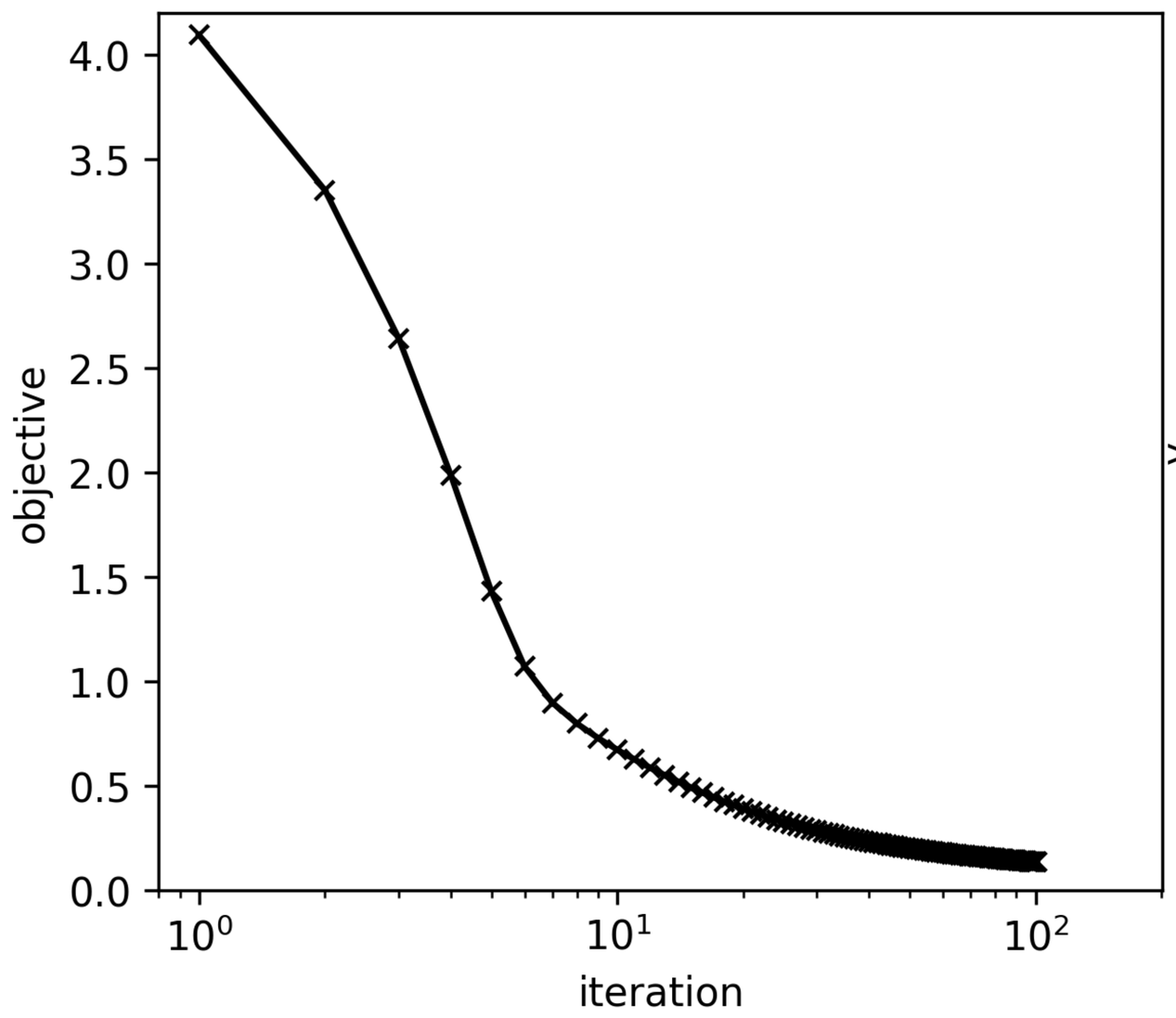


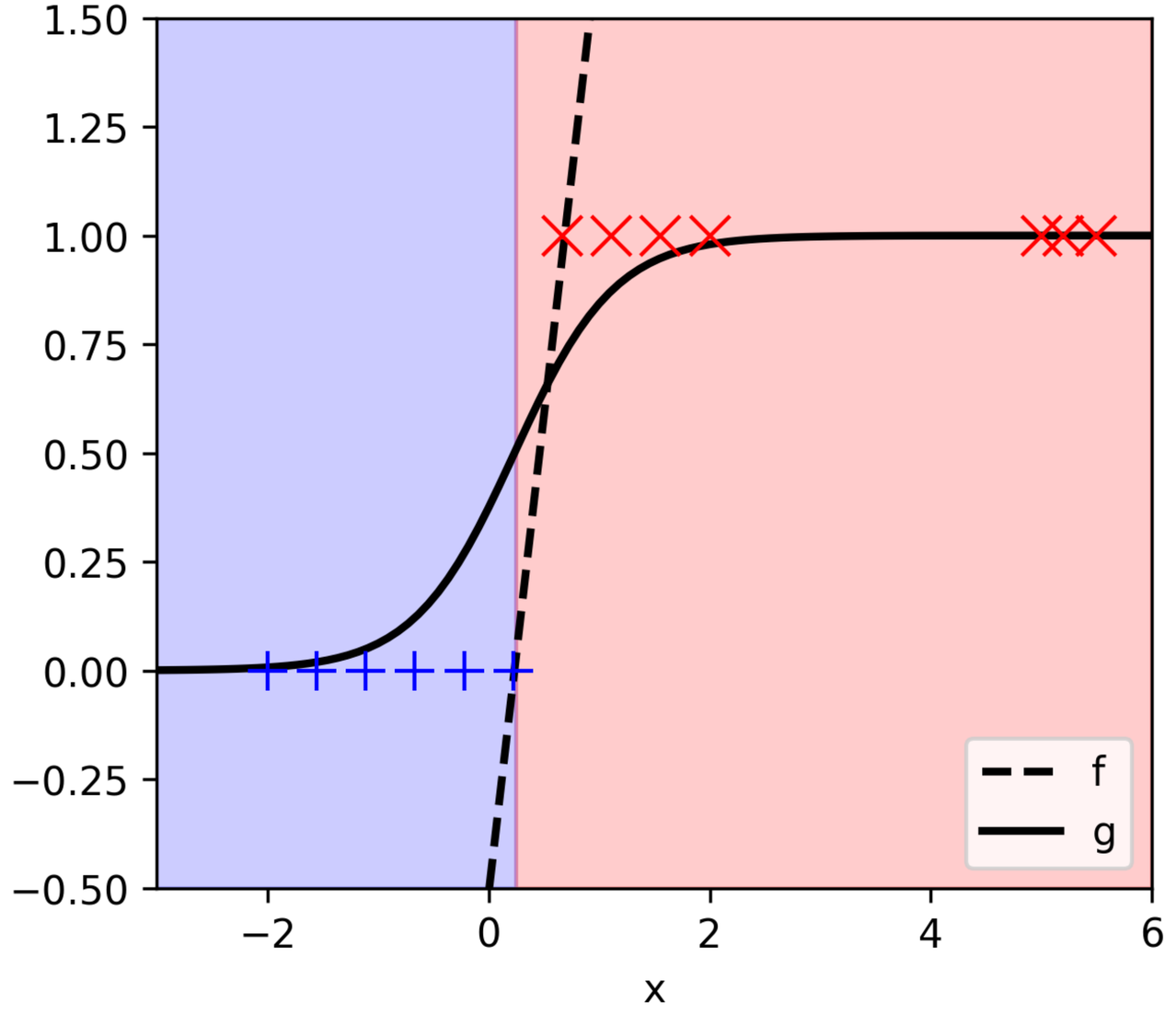
Binary logistic classification - example



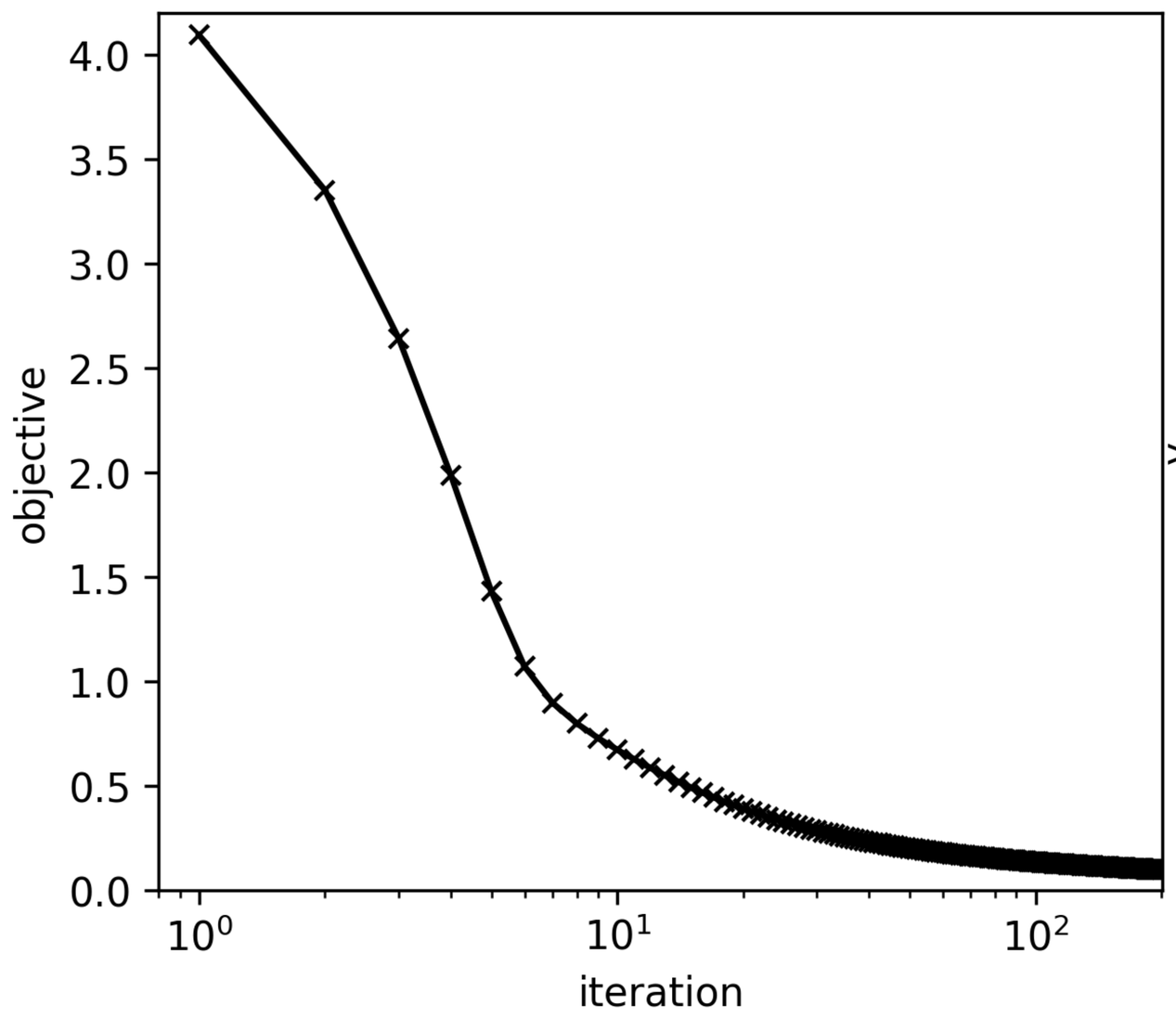


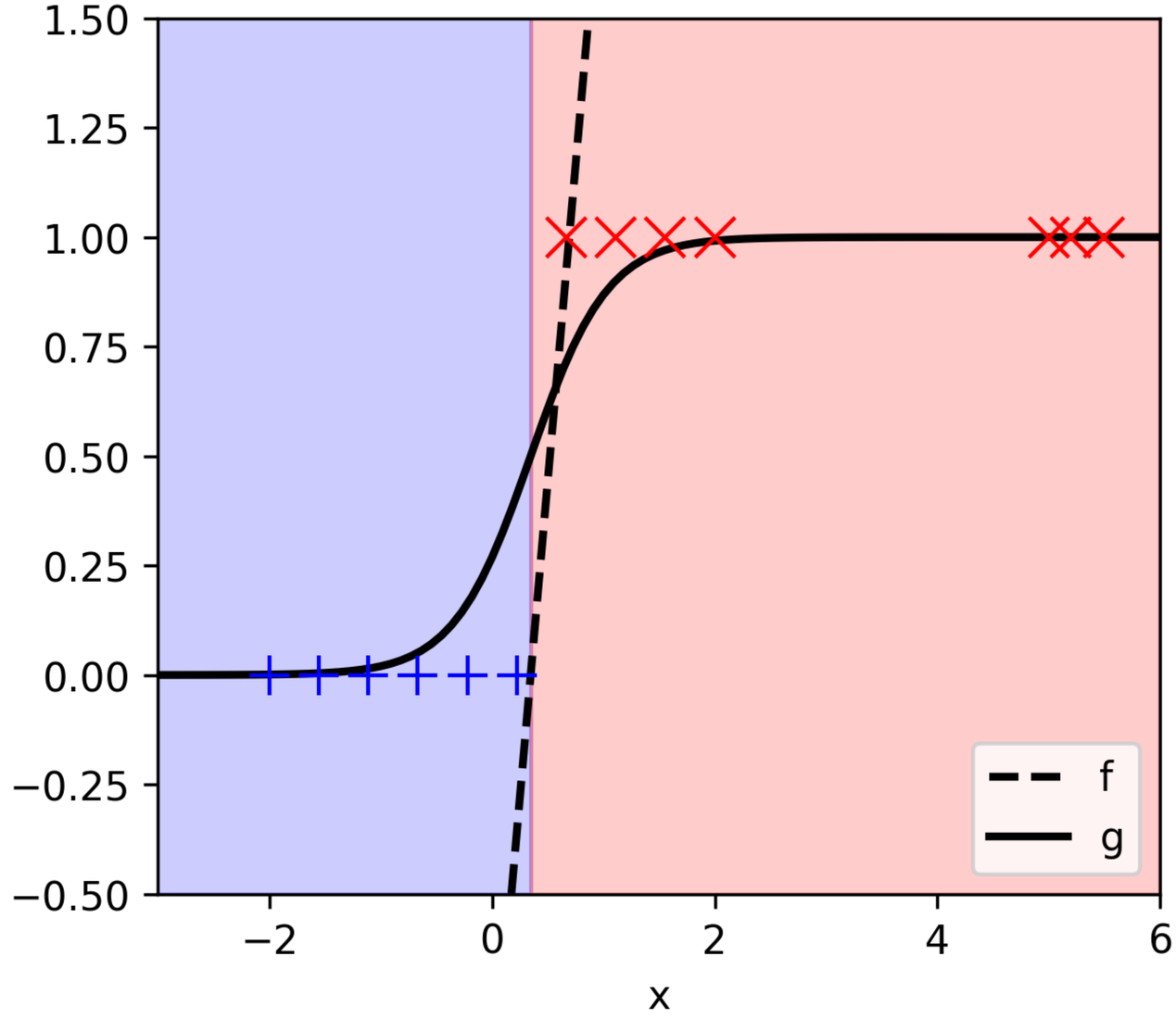
Binary logistic classification - example



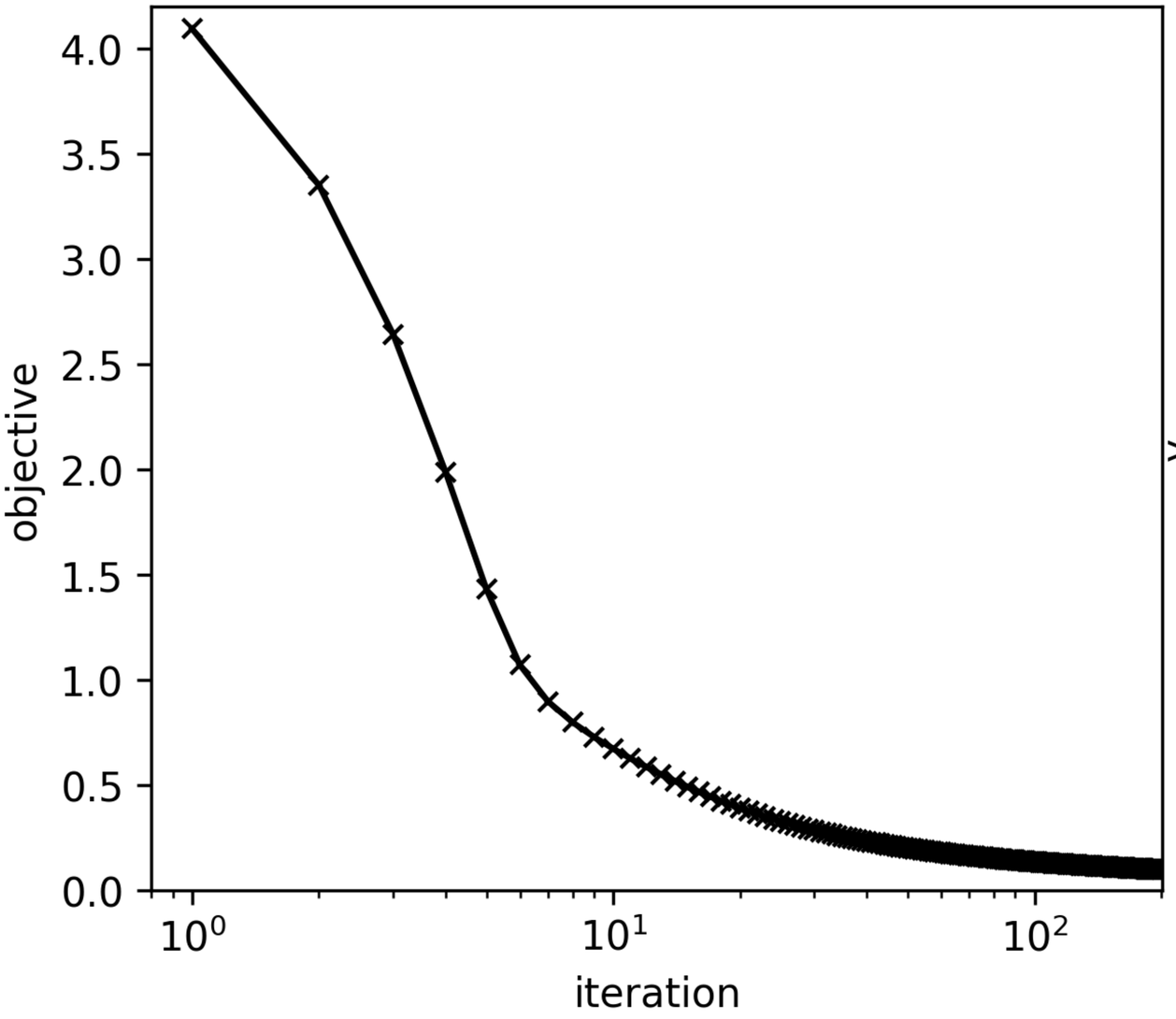


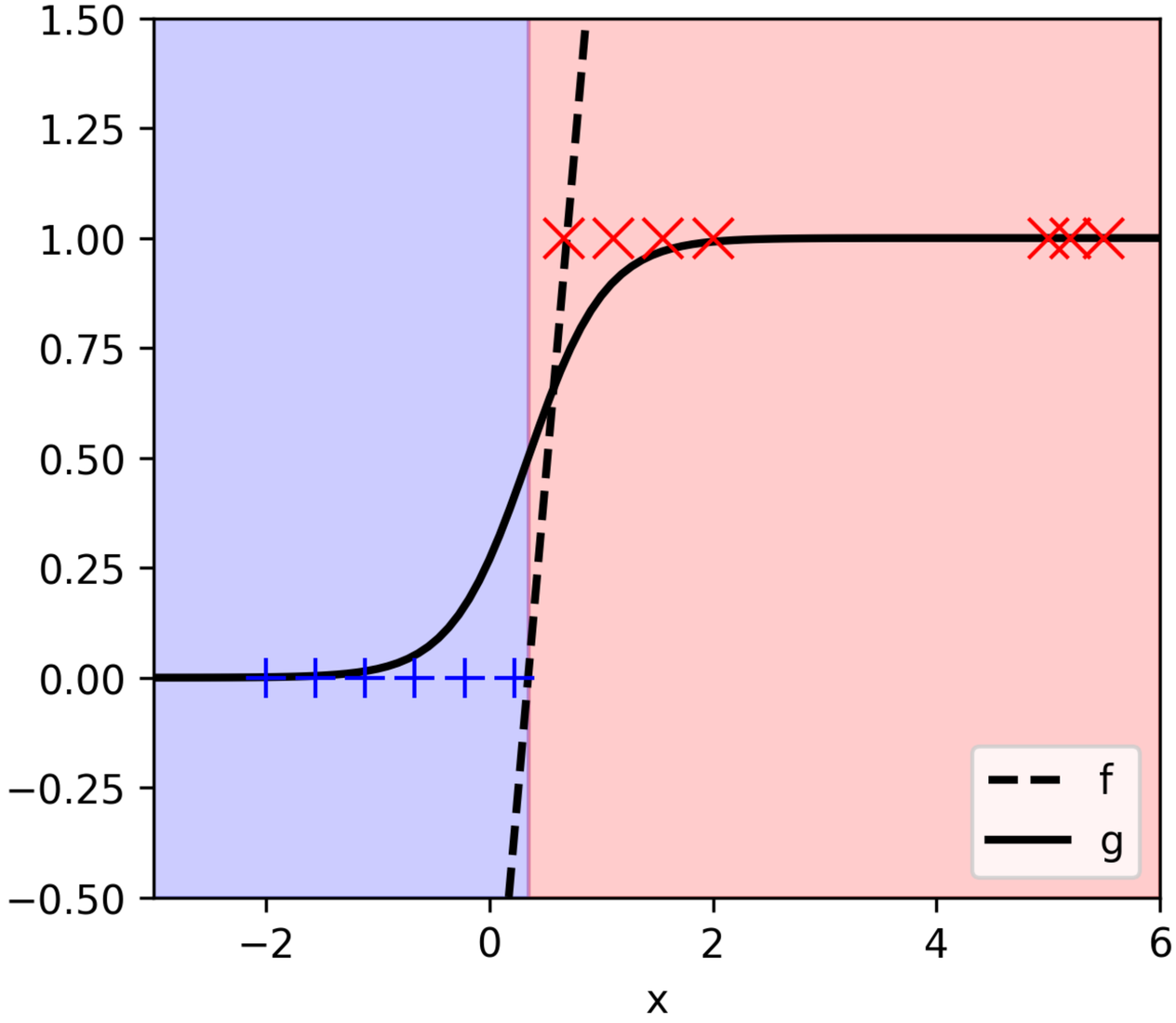
Binary logistic classification - example





Binary logistic classification - example





More examples: https://playground.tensorflow.org/

Binary logistic classification - benefits

Binary logistic classification - benefits

(i) Simple to implement, fast, widely used in industry

(ii) Can easily support sparse features

(iii) Easy to interpret as *log-odds* is a linear function of parameters

(iv) Easy to extend to multi-class and other kinds of outputs

(v) Easy to include new features (similar to regression) or kernelise (SML COMP4670)

Notes: often called logistic *regression*, but it is a classification task! And the underlying mapping f is still linear in the parameters.

Binary logistic classification vs linear regression

	Linear regression	Logistic regression
Underlying mapping: $f_\theta(\mathbf{x}) = \sum_{d=1}^{D} \theta_d x_d = \theta^\intercal \mathbf{x}, \theta \in \mathbb{R}^D$		**Same**
Likelihood:	$p(\mathcal{D} \mid \theta) = \prod_n \mathcal{N}(y_n; f_\theta(x_n), \sigma^2)$	$p(\mathcal{D} \mid \theta) = \prod_n \text{Bernoulli}(y_n; g_\theta(x_n))$ $g_\theta(x) = \sigma(f_\theta(x))$, σ is logistic sigmoid
Closed form: $\theta = (X^\intercal X + N\lambda \mathbf{I})^{-1} X^\intercal \mathbf{y}$ or $\theta = (X^\intercal X)^{-1} X^\intercal \mathbf{y}$		✗ **Have to use numerical optimisation**

Overview

1. Recap: Linear regression
2. From least squares to binary logistic classification
3. **From binary to multi-class classification**
4. Evaluation
5. Linearly separable data - perceptron algorithm

From binary to multi-class: set up

We construct a **classifier** that learns the class probabilities: $p(y = c \mid x)$

From binary to multi-class: set up

We construct a **classifier** that learns the class probabilities: $p(y = c \,|\, x)$

Training data $\mathscr{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)\}, \mathbf{x}_n \in \mathbb{R}^D,\ y_n \in \{0,1\}$

Training data $\mathscr{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)\}, \mathbf{x}_n \in \mathbb{R}^D,\ y_n \in \{1,\ldots,C\},\ C$ classes

From binary to multi-class: set up

We construct a **classifier** that learns the class probabilities: $p(y = c \,|\, x)$

Training data $\mathscr{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)\}, \mathbf{x}_n \in \mathbb{R}^D,\ y_n \in \{0,1\}$

By laws of probabilities: $p(y = 0 \,|\, x) + p(y = 1 \,|\, x) = 1$

Training data $\mathscr{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)\}, \mathbf{x}_n \in \mathbb{R}^D,\ y_n \in \{1,\ldots,C\},\ C$ classes

By laws of probabilities: $p(y = 1 \,|\, x) + p(y = 2 \,|\, x) + \ldots + p(y = C \,|\, x) = 1$

From binary to multi-class: set up

We construct a **classifier** that learns the class probabilities: $p(y = c \,|\, x)$

Training data $\mathscr{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)\}, \mathbf{x}_n \in \mathbb{R}^D,\ y_n \in \{0,1\}$

By laws of probabilities: $p(y = 0 \,|\, x) + p(y = 1 \,|\, x) = 1$

Imagine: $p(y = 1 \,|\, x) = g_\theta(x)$ then $p(y = 0 \,|\, x) = 1 - g_\theta(x)$. **Constraint**: $0 \leq g_\theta(x) \leq 1, \forall x$

Training data $\mathscr{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)\}, \mathbf{x}_n \in \mathbb{R}^D,\ y_n \in \{1,\ldots, C\},\ C$ classes

By laws of probabilities: $p(y = 1 \,|\, x) + p(y = 2 \,|\, x) + \ldots + p(y = C \,|\, x) = 1$

Imagine: $p(y = 1 \,|\, x) = g_\theta^{(1)}(x), p(y = 2 \,|\, x) = g_\theta^{(2)}(x), \ldots, p(y = C \,|\, x) = g_\theta^{(C)}(x).$

Constraint: $0 \leq g_\theta^{(c)}(x) \leq 1, \forall x, c,$ and $\sum_c g_\theta^{(c)}(x) = 1$

From binary to multi-class: sigmoid to softmax

Have: $f_\theta(\mathbf{x}) = \sum_{d=1}^{D} \theta_d x_d = \theta^\intercal \mathbf{x}, \theta \in \mathbb{R}^D$. **Want:** $0 \leq g_\theta(x) \leq 1, \forall x$

Idea: 'squash' $f_\theta(x)$ through a *logistic sigmoid* function $g_\theta(x) = \sigma(f_\theta(x))$, where

$$\sigma(z) = \frac{\exp(z)}{1 + \exp(z)} = \frac{1}{1 + \exp(-z)}$$

From binary to multi-class: sigmoid to softmax

Have: $f_\theta(\mathbf{x}) = \sum_{d=1}^{D} \theta_d x_d = \theta^\intercal \mathbf{x}, \theta \in \mathbb{R}^D$. **Want**: $0 \leq g_\theta(x) \leq 1, \forall x$

Idea: 'squash' $f_\theta(x)$ through a *logistic sigmoid* function $g_\theta(x) = \sigma(f_\theta(x))$, where

$$\sigma(z) = \frac{\exp(z)}{1 + \exp(z)} = \frac{1}{1 + \exp(-z)}$$

Construct: $f_\theta^{(c)}(\mathbf{x}) = \sum_{d=1}^{D} \theta_{cd} x_d = \theta_c^\intercal \mathbf{x}, \theta_c \in \mathbb{R}^D$, one linear function mapping for each class

From binary to multi-class: sigmoid to softmax

Have: $f_\theta(\mathbf{x}) = \sum_{d=1}^{D} \theta_d x_d = \theta^\intercal \mathbf{x}, \theta \in \mathbb{R}^D$. **Want**: $0 \leq g_\theta(x) \leq 1, \forall x$

Idea: 'squash' $f_\theta(x)$ through a *logistic sigmoid* function $g_\theta(x) = \sigma(f_\theta(x))$, where

$$\sigma(z) = \frac{\exp(z)}{1 + \exp(z)} = \frac{1}{1 + \exp(-z)}$$

Construct: $f_\theta^{(c)}(\mathbf{x}) = \sum_{d=1}^{D} \theta_{cd} x_d = \theta_c^\intercal \mathbf{x}, \theta_c \in \mathbb{R}^D$, one linear function mapping for each class

Want: $0 \leq g_\theta^{(c)}(x) \leq 1, \forall x, c$, and $\sum_c g_\theta^{(c)}(x) = 1$

From binary to multi-class: sigmoid to softmax

Have: $f_\theta(\mathbf{x}) = \sum_{d=1}^{D} \theta_d x_d = \theta^\intercal \mathbf{x}, \theta \in \mathbb{R}^D$. **Want**: $0 \leq g_\theta(x) \leq 1, \forall x$

Idea: 'squash' $f_\theta(x)$ through a *logistic sigmoid* function $g_\theta(x) = \sigma(f_\theta(x))$, where

$$\sigma(z) = \frac{\exp(z)}{1 + \exp(z)} = \frac{1}{1 + \exp(-z)}$$

Construct: $f_\theta^{(c)}(\mathbf{x}) = \sum_{d=1}^{D} \theta_{cd} x_d = \theta_c^\intercal \mathbf{x}, \theta_c \in \mathbb{R}^D$, one linear function mapping for each class

Want: $0 \leq g_\theta^{(c)}(x) \leq 1, \forall x, c$, and $\sum_c g_\theta^{(c)}(x) = 1$

Idea: 'squash' $f_\theta^{(c)}(x)$ through a *softmax:* $g_\theta^{(c)}(x) = \text{softmax}(f_\theta^{(c)}(x)) = \frac{\exp(f_\theta^{(c)}(x))}{\sum_{k=1}^{C} \exp(f_\theta^{(k)}(x))}$

From binary to multi-class: cross-entropy

From binary to multi-class: cross-entropy

Have:

$$p(y_n \mid x_n, \theta) = \begin{cases} g_\theta(x_n), & \text{if } y_n = 1 \\ 1 - g_\theta(x_n), & \text{if } y_n = 0 \end{cases}$$

Have: $p(y_n = c \mid x_n, \theta) = g_\theta^{(c)}(x_n)$

From binary to multi-class: cross-entropy

Have:

$$p(y_n \,|\, x_n, \theta) = \begin{cases} g_\theta(x_n), & \text{if } y_n = 1 \\ 1 - g_\theta(x_n), & \text{if } y_n = 0 \end{cases}$$

minimise negative log-likelihood

$$\mathscr{L}(\theta) = \frac{1}{N} \sum_{n=1}^{N} - y_n \log(g_\theta(x_n)) - (1 - y_n) \log(1 - g_\theta(x_n))$$

Often called the **binary cross-entropy loss**

Have: $p(y_n = c \,|\, x_n, \theta) = g_\theta^{(c)}(x_n)$ **minimise** neg log-likelihood $\mathscr{L}(\theta) = -\frac{1}{N} \sum_{n=1}^{N} \log(g_\theta^{(y_n)}(x_n))$

called the **multi class cross-entropy loss**

From binary to multi-class: cross-entropy

Have:

$$p(y_n \,|\, x_n, \theta) = \begin{cases} g_\theta(x_n), & \text{if } y_n = 1 \\ 1 - g_\theta(x_n), & \text{if } y_n = 0 \end{cases}$$

minimise negative log-likelihood

$$\mathscr{L}(\theta) = \frac{1}{N} \sum_{n=1}^{N} - y_n \log(g_\theta(x_n)) - (1 - y_n) \log(1 - g_\theta(x_n))$$

Often called the **binary cross-entropy loss**

Have: $p(y_n = c \,|\, x_n, \theta) = g_\theta^{(c)}(x_n)$ **minimise** neg log-likelihood $\mathscr{L}(\theta) = -\frac{1}{N} \sum_{n=1}^{N} \log(g_\theta^{(y_n)}(x_n))$

called the **multi class cross-entropy loss**

If *one-hot coding or 1-of-C coding* is used, $\mathbf{t}_n = (t_{n1}, t_{n2}, \ldots, t_{nC})$

where $t_{nc} = 1$ if $y_n = c$ and 0 otherwise:

$$\mathscr{L}(\theta) = -\frac{1}{N} \sum_{n=1}^{N} \sum_{c=1}^{C} t_{nc} \log(g_\theta^c(x_n))$$

From binary to multi-class: cross-entropy

Have:

$$p(y_n \,|\, x_n, \theta) = \begin{cases} g_\theta(x_n), & \text{if } y_n = 1 \\ 1 - g_\theta(x_n), & \text{if } y_n = 0 \end{cases}$$

minimise negative log-likelihood

$$\mathscr{L}(\theta) = \frac{1}{N} \sum_{n=1}^{N} - y_n \log(g_\theta(x_n)) - (1 - y_n) \log(1 - g_\theta(x_n))$$

Gradients: $\dfrac{\mathrm{d}\mathscr{L}(\theta)}{\mathrm{d}\theta} = \dfrac{1}{N} \sum_{n=1}^{N} (g_\theta(x_n) - y_n) x_n^T$

Often called the **binary cross-entropy loss**

Have: $p(y_n = c \,|\, x_n, \theta) = g_\theta^{(c)}(x_n)$ **minimise** neg log-likelihood $\mathscr{L}(\theta) = -\dfrac{1}{N} \sum_{n=1}^{N} \log(g_\theta^{(y_n)}(x_n))$

called the **multi class cross-entropy loss**

If *one-hot coding or 1-of-C coding* is used, $\mathbf{t}_n = (t_{n1}, t_{n2}, \ldots, t_{nC})$

where $t_{nc} = 1$ if $y_n = c$ and 0 otherwise:

Gradients: $\dfrac{\mathrm{d}\mathscr{L}(\theta)}{\mathrm{d}\theta_c} = \dfrac{1}{N} \sum_{n=1}^{N} (g_\theta^{(c)}(x_n) - t_{nc}) x_n^T$

$$\mathscr{L}(\theta) = -\frac{1}{N} \sum_{n=1}^{N} \sum_{c=1}^{C} t_{nc} \log(g_\theta^c(x_n))$$

Alternative approaches

Alternative approaches

One-vs-one approach

- A binary classifier is built for every pairwise combination of classes. A total of $\frac{C(C-1)}{2}$ classifiers for C classes.
- Each classifier receives the samples of a pair of classes from the original training set, and learns to distinguish these two classes.
- At prediction time, a voting scheme is applied: all $\frac{C(C-1)}{2}$ classifiers are applied and the class that has the highest number of **1** predictions is chosen by the combined classifier.

Alternative approaches

One-vs-one approach

- A binary classifier is built for every pairwise combination of classes. A total of $\frac{C(C-1)}{2}$ classifiers for C classes.
- Each classifier receives the samples of a pair of classes from the original training set, and learns to distinguish these two classes.
- At prediction time, a voting scheme is applied: all $\frac{C(C-1)}{2}$ classifiers are applied and the class that has the highest number of **1** predictions is chosen by the combined classifier.

One-vs-all (one-vs-rest) approach

- Classifier is built for each class and the remaining classes are grouped into one large "negative" class. A total of C binary classifiers.
- For a new case x^*, predicting the label c for which the corresponding classifier has the highest confidence score (e.g., the probability in logistic regression)

Overview

1. Recap: Linear regression
2. From least squares to binary logistic classification
3. From binary to multi-class classification
4. **Evaluation**
5. Linearly separable data - perceptron algorithm

Binary classification - confusion matrix

True label

Predicted label	$y = -1$	$y = 1$	Total
$\hat{y} = -1$			
$\hat{y} = 1$			
Total			

Binary classification - confusion matrix

True label

Predicted label	$y = -1$	$y = 1$	Total
$\hat{y} = -1$	True negative (TN)		
$\hat{y} = 1$			
Total			

Binary classification - confusion matrix

True label

Predicted label	$y = -1$	$y = 1$	Total
$\hat{y} = -1$	True negative (TN)		
$\hat{y} = 1$		True positive (TP)	
Total			

Binary classification - confusion matrix

		True label		
		$y = -1$	$y = 1$	Total
Predicted label	$\hat{y} = -1$	True negative (TN)		
	$\hat{y} = 1$	False positive (FP)	True positive (TP)	
	Total			

Binary classification - confusion matrix

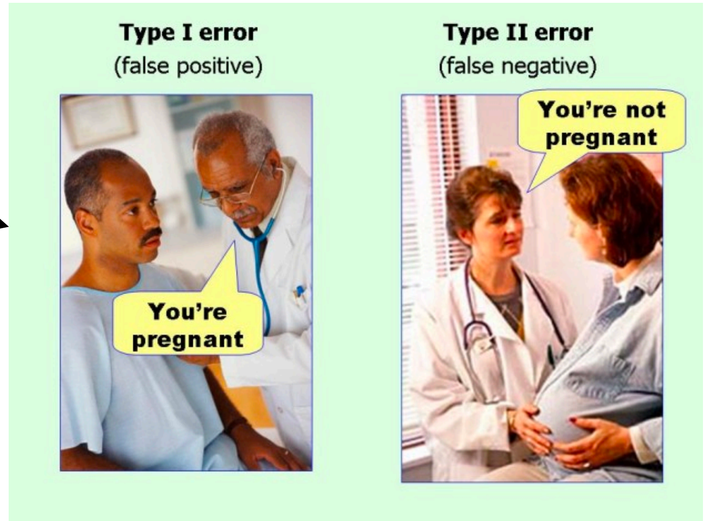
		True label $y = -1$	True label $y = 1$	Total
Predicted label	$\hat{y} = -1$	True negative (TN)	False negative (FN)	
	$\hat{y} = 1$	False positive (FP)	True positive (TP)	
	Total			

Binary classification - confusion matrix

		True label		
		$y = -1$	$y = 1$	Total
Predicted label	$\hat{y} = -1$	True negative (TN)	False negative (FN)	$\hat{N}_n$
	$\hat{y} = 1$	False positive (FP)	True positive (TP)	$\hat{N}_p$
	Total	N_n	N_p	N

Binary classification - confusion matrix

	True label $y=-1$	True label $y=1$	Total
Predicted label $\hat{y}=-1$	True negative (TN)	False negative (FN)	$\hat{N}_n$
Predicted label $\hat{y}=1$	False positive (FP)	True positive (TP)	$\hat{N}_p$
Total	N_n	N_p	N



Binary classification - popular metrics

		True label		
		$y = -1$	$y = 1$	Total
Predicted label	$\hat{y} = -1$	True negative (TN)	False negative (FN)	$\hat{N}_n$
	$\hat{y} = 1$	False positive (FP)	True positive (TP)	$\hat{N}_p$
	Total	N_n	N_p	N

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{N} \qquad \text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \qquad \text{True positive rate (TPR)} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

$$\text{Error} = \frac{\text{FP} + \text{FN}}{N} \qquad \text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \qquad \text{False positive rate (FPR)} = \frac{\text{FP}}{\text{FP} + \text{TN}}$$

Binary classification - popular metrics

	True label $y=-1$	True label $y=1$	Total
Predicted label $\hat{y}=-1$	True negative (TN)	False negative (FN)	$\hat{N}_n$
Predicted label $\hat{y}=1$	False positive (FP)	True positive (TP)	$\hat{N}_p$
Total	N_n	N_p	N

Questions: compute these metrics for

1. A perfect classifier
2. All classified as positive
3. All classified as negative
4. Random guessing

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{N}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

$$\text{True positive rate (TPR)} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

$$\text{Error} = \frac{\text{FP} + \text{FN}}{N}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

$$\text{False positive rate (FPR)} = \frac{\text{FP}}{\text{FP} + \text{TN}}$$

Binary classification - too many metrics?

Ratio	Name
FP/N	False positive rate, Fall-out, Probability of false alarm
TN/N	True negative rate, Specificity, Selectivity
TP/P	True positive rate, Sensitivity, Power, *Recall*, Probability of detection
FN/P	False negative rate, Miss rate
TP/P*	Positive predictive value, *Precision*
FP/P*	False discovery rate
TN/N*	Negative predictive value
FN/N*	False omission rate
P/*n*	Prevalence
(FN + FP)/*n*	*Misclassification rate*
(TN + TP)/*n*	Accuracy, 1 − misclassification rate
2TP/(P* + P)	F_1 *score*
$(1 + \beta^2)$TP/$((1 + \beta^2)$TP + β^2FN + FP)	F_β *score*

Binary classification - too many metrics?

Ratio	Name
FP/N	False positive rate, Fall-out, Probability of false alarm
TN/N	True negative rate, Specificity, Selectivity
TP/P	True positive rate, Sensitivity, Power, *Recall*, Probability of detection
FN/P	False negative rate, Miss rate
TP/P*	Positive predictive value, *Precision*
FP/P*	False discovery rate
TN/N*	Negative predictive value
FN/N*	False omission rate
P/*n*	Prevalence
(FN + FP)/*n*	*Misclassification rate*
(TN + TP)/*n*	Accuracy, 1 − misclassification rate
2TP/(P* + P)	F_1 *score*
$(1 + \beta^2)$TP/$((1 + \beta^2)$TP + β^2FN + FP)	F_β *score*

We could have a whole course on metrics!

Binary classification - too many metrics?

Ratio	Name
FP/N	False positive rate, Fall-out, Probability of false alarm
TN/N	True negative rate, Specificity, Selectivity
TP/P	True positive rate, Sensitivity, Power, *Recall*, Probability of detection
FN/P	False negative rate, Miss rate
TP/P*	Positive predictive value, *Precision*
FP/P*	False discovery rate
TN/N*	Negative predictive value
FN/N*	False omission rate
P/*n*	Prevalence
(FN + FP)/*n*	*Misclassification rate*
(TN + TP)/*n*	Accuracy, 1 − misclassification rate
2TP/(P* + P)	F_1 *score*
$(1 + \beta^2)$TP/$((1 + \beta^2)$TP + β^2FN + FP)	F_β *score*

We could have a whole course on metrics!

Need to consider down-stream applications, e.g. in medical domain, FNs are costly but FPs result in unnecessary further tests.

Binary classification - too many metrics?

Ratio	Name
FP/N	False positive rate, Fall-out, Probability of false alarm
TN/N	True negative rate, Specificity, Selectivity
TP/P	True positive rate, Sensitivity, Power, *Recall*, Probability of detection
FN/P	False negative rate, Miss rate
TP/P*	Positive predictive value, *Precision*
FP/P*	False discovery rate
TN/N*	Negative predictive value
FN/N*	False omission rate
P/*n*	Prevalence
(FN + FP)/*n*	*Misclassification rate*
(TN + TP)/*n*	Accuracy, 1 − misclassification rate
2TP/(P* + P)	F_1 *score*
$(1 + \beta^2)$TP/$((1 + \beta^2)$TP + β^2FN + FP)	F_β *score*

We could have a whole course on metrics!

Need to consider down-stream applications, e.g. in medical domain, FNs are costly but FPs result in unnecessary further tests.

Accuracy and error are not useful for imbalanced classes

Binary classification - too many metrics?

Ratio	Name
FP/N	False positive rate, Fall-out, Probability of false alarm
TN/N	True negative rate, Specificity, Selectivity
TP/P	True positive rate, Sensitivity, Power, *Recall*, Probability of detection
FN/P	False negative rate, Miss rate
TP/P*	Positive predictive value, *Precision*
FP/P*	False discovery rate
TN/N*	Negative predictive value
FN/N*	False omission rate
P/*n*	Prevalence
(FN + FP)/*n*	*Misclassification rate*
(TN + TP)/*n*	Accuracy, 1 − misclassification rate
2TP/(P* + P)	F_1 *score*
$(1 + \beta^2)$TP/$((1 + \beta^2)$TP + β^2FN + FP)	F_β *score*

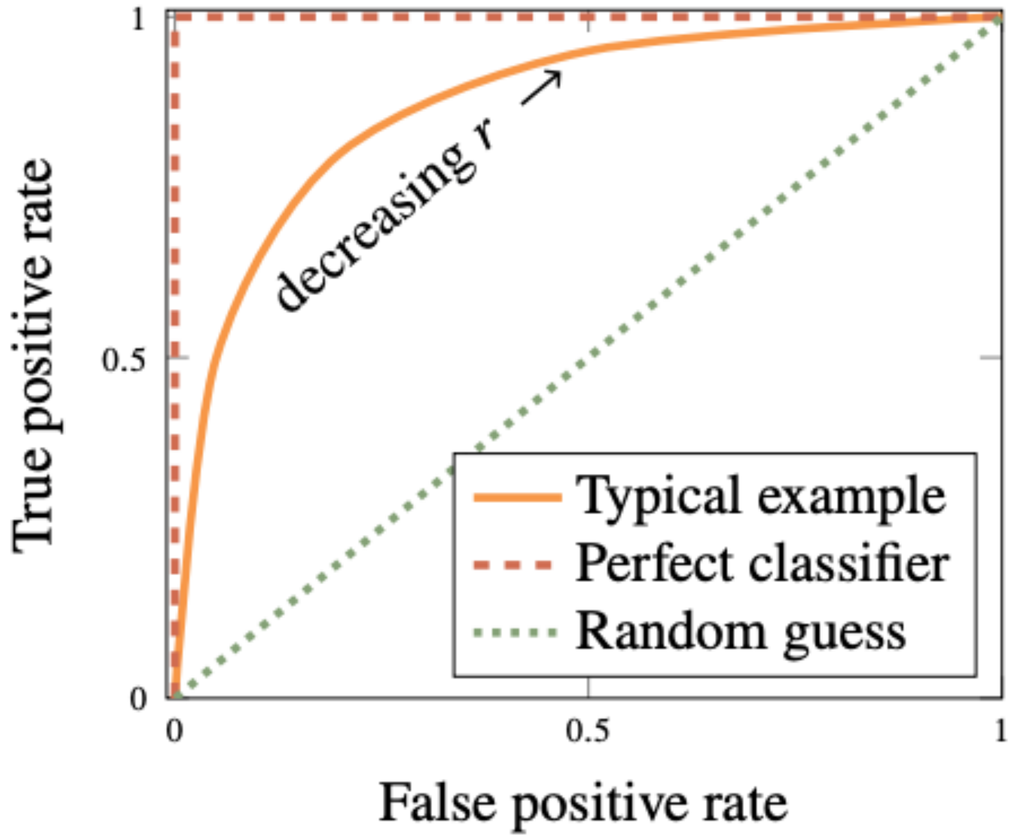
We could have a whole course on metrics!

Need to consider down-stream applications, e.g. in medical domain, FNs are costly but FPs result in unnecessary further tests.

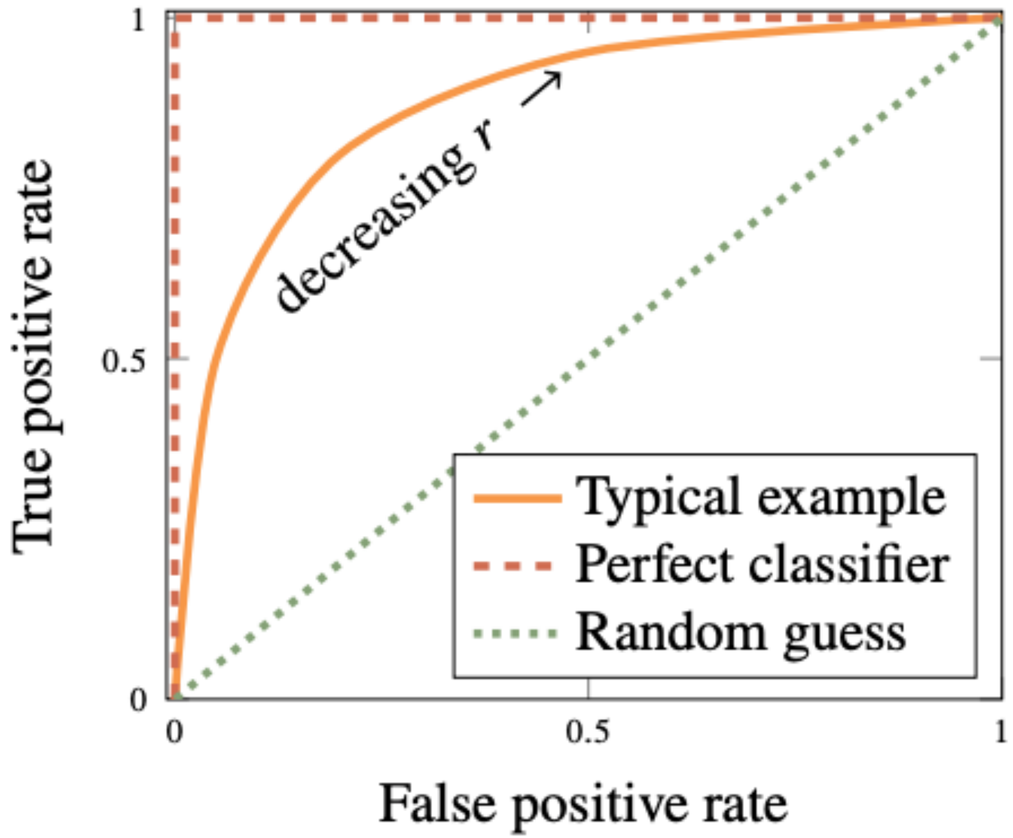
Accuracy and error are not useful for imbalanced classes

In this table: N is Nn, P is Np, n is N

Binary classification - Receiver Operating Characteristic



Binary classification - Receiver Operating Characteristic



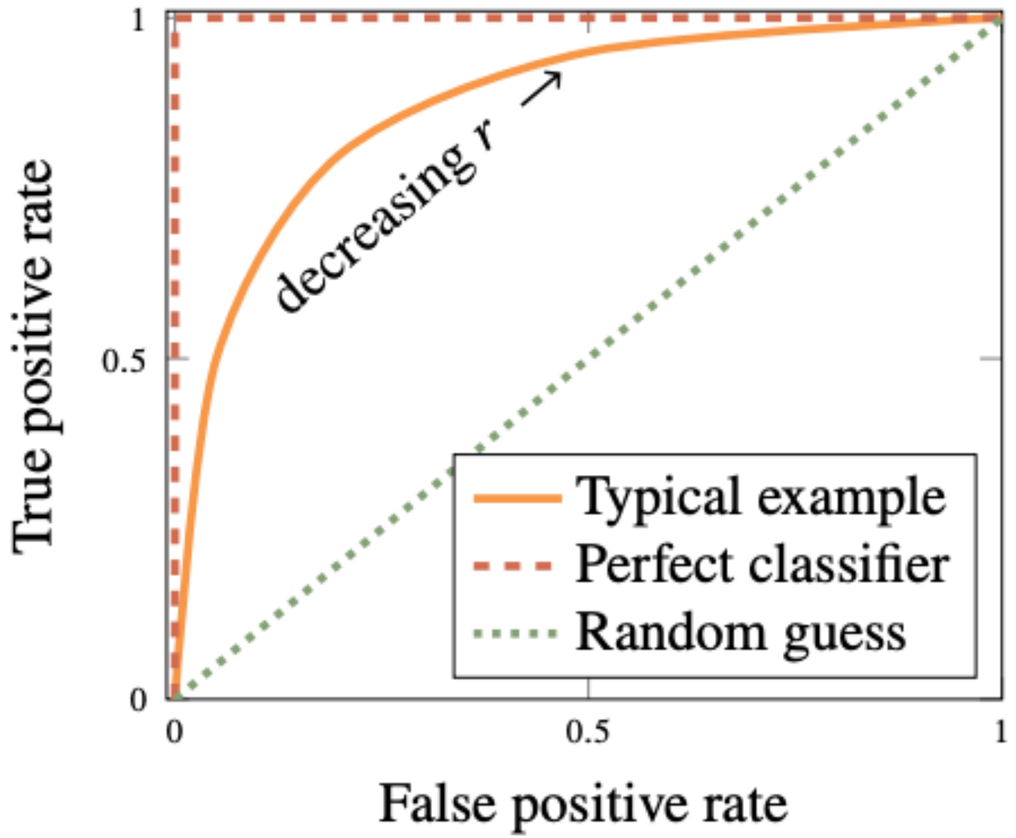
Consider a binary classification example, at test time:

$$p(y^* = 1 \,|\, x^*) = g(x^*) \text{ and } p(y^* = 0 \,|\, x^*) = 1 - g(x^*)$$

We choose a **decision threshold** r and decide:

$$y^* = \begin{cases} 1 & \text{if } g(x^*) > r \\ 0 & \text{otherwise} \end{cases}$$

Binary classification - Receiver Operating Characteristic



Consider a binary classification example, at test time:

$$p(y^* = 1 \,|\, x^*) = g(x^*) \text{ and } p(y^* = 0 \,|\, x^*) = 1 - g(x^*)$$

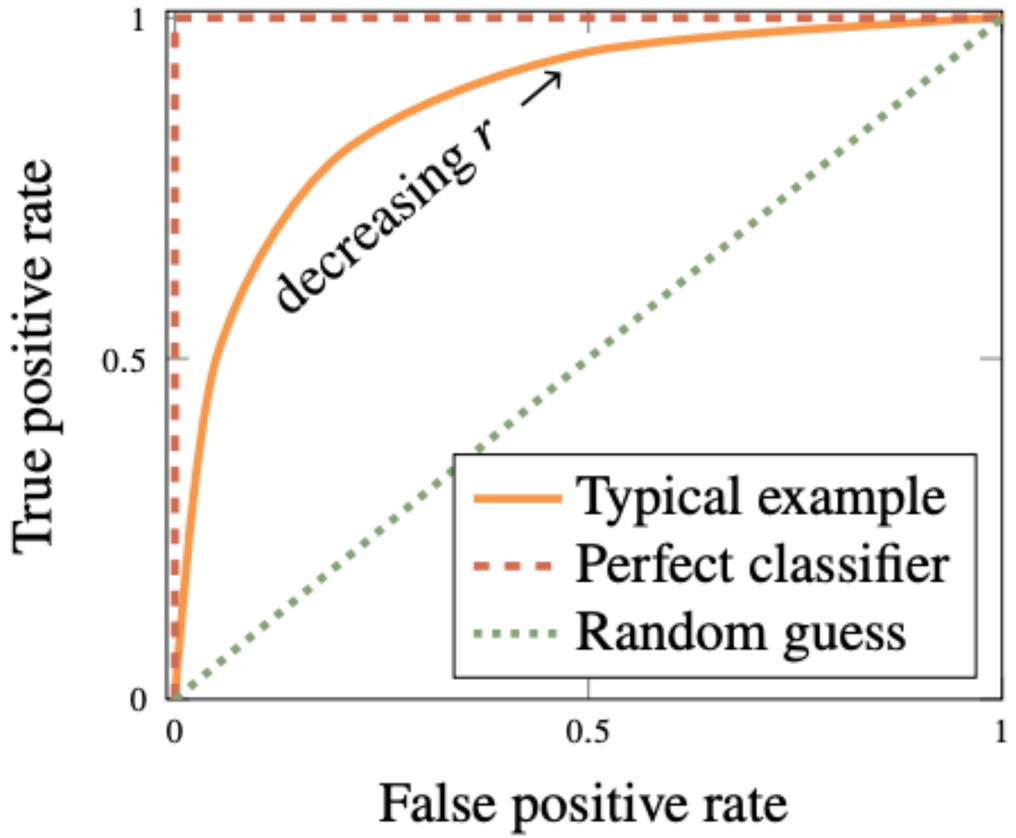
We choose a **decision threshold** r and decide:

$$y^* = \begin{cases} 1 & \text{if } g(x^*) > r \\ 0 & \text{otherwise} \end{cases}$$

For each threshold between 0.0 and 1.0, we make decisions for all test points, construct the confusion matrix, calculate TPR and FPR, and draw a corresponding point on the **ROC** chart.

Join the points for all thresholds to form the ROC curve.

Binary classification - Receiver Operating Characteristic



ROC curve is used to select an *operating point*, aka the threshold of the classifier.

We can compare multiple classifiers using their ROC curves. Better classifiers tend to have higher Area Under the Curve (AUC).

Consider a binary classification example, at test time:

$$p(y^* = 1 \,|\, x^*) = g(x^*) \text{ and } p(y^* = 0 \,|\, x^*) = 1 - g(x^*)$$

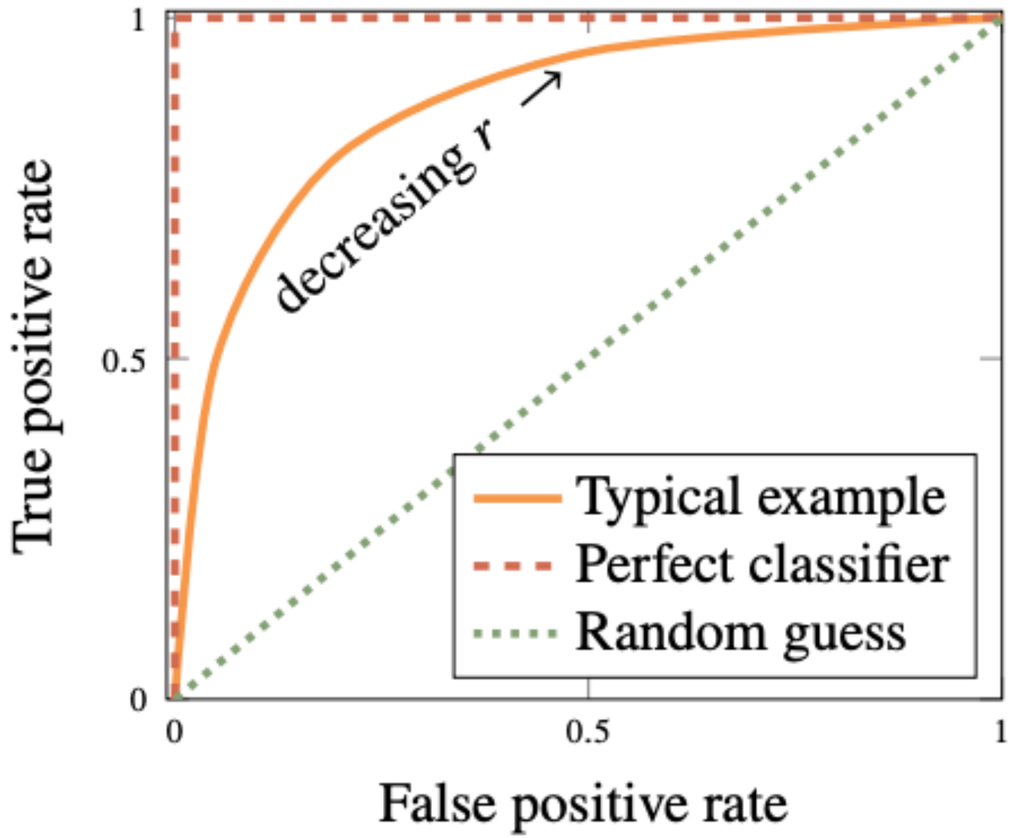
We choose a **decision threshold** r and decide:

$$y^* = \begin{cases} 1 & \text{if } g(x^*) > r \\ 0 & \text{otherwise} \end{cases}$$

For each threshold between 0.0 and 1.0, we make decisions for all test points, construct the confusion matrix, calculate TPR and FPR, and draw a corresponding point on the **ROC** chart.

Join the points for all thresholds to form the ROC curve.

Binary classification - Receiver Operating Characteristic



ROC curve is used to select an *operating point*, aka the threshold of the classifier.

We can compare multiple classifiers using their ROC curves. Better classifiers tend to have higher Area Under the Curve (AUC).

Consider a binary classification example, at test time:

$$p(y^* = 1 \,|\, x^*) = g(x^*) \text{ and } p(y^* = 0 \,|\, x^*) = 1 - g(x^*)$$

We choose a **decision threshold** r and decide:

$$y^* = \begin{cases} 1 & \text{if } g(x^*) > r \\ 0 & \text{otherwise} \end{cases}$$

For each threshold between 0.0 and 1.0, we make decisions for all test points, construct the confusion matrix, calculate TPR and FPR, and draw a corresponding point on the **ROC** chart.

Join the points for all thresholds to form the ROC curve.

For imbalanced data (with many negatives), we can use **precision-recall curve** instead.

Overview

1. Recap: Linear regression
2. From least squares to binary logistic classification
3. From binary to multi-class classification
4. Evaluation
5. **Linearly separable data - perceptron algorithm**

Rosenblatt

Frank Rosenblatt
1928–1969

Rosenblatt's perceptron played an important role in the history of machine learning. Initially, Rosenblatt simulated the perceptron on an IBM 704 computer at Cornell in 1957, but by the early 1960s he had built special-purpose hardware that provided a direct, parallel implementation of perceptron learning. Many of his ideas were encapsulated in "Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms" published in 1962. Rosenblatt's work was criticized by Marvin Minksy, whose objections were published in the book "Perceptrons", co-authored with Seymour Papert. This book was widely misinterpreted at the time as showing that neural networks were fatally flawed and could only learn solutions for linearly separable problems. In fact, it only proved such limitations in the case of single-layer networks such as the perceptron and merely conjectured (incorrectly) that they applied to more general network models. Unfortunately, however, this book contributed to the substantial decline in research funding for neural computing, a situation that was not reversed until the mid-1980s. Today, there are many hundreds, if not thousands, of applications of neural networks in widespread use, with examples in areas such as handwriting recognition and information retrieval being used routinely by millions of people.

Model - step `squashing` function and criterion

Model - step `squashing` function and criterion

Linear mapping:

$$f_\theta(\mathbf{x}) = \sum_{d=1}^{D} \theta_d x_d = \theta^\intercal \mathbf{x}, \theta \in \mathbb{R}^D$$

Model - step `squashing` function and criterion

Linear mapping:

$$f_\theta(\mathbf{x}) = \sum_{d=1}^{D} \theta_d x_d = \theta^\intercal \mathbf{x}, \theta \in \mathbb{R}^D$$

Reminder - logistic regression:

$$p(y_n \,|\, x_n, \theta) = \begin{cases} g_\theta(x_n), & \text{if } y = -1 \\ 1 - g_\theta(x_n), & \text{if } y = 1 \end{cases}$$

$g_\theta(x) = \sigma(f_\theta(x))$, σ is logistic sigmoid

Model - step `squashing` function and criterion

Linear mapping:

$$f_\theta(\mathbf{x}) = \sum_{d=1}^{D} \theta_d x_d = \theta^\intercal \mathbf{x}, \theta \in \mathbb{R}^D$$

Reminder - logistic regression:

$$p(y_n \,|\, x_n, \theta) = \begin{cases} g_\theta(x_n), & \text{if } y = -1 \\ 1 - g_\theta(x_n), & \text{if } y = 1 \end{cases}$$

$g_\theta(x) = \sigma(f_\theta(x))$, σ is logistic sigmoid

Perceptron model: $y_n = \begin{cases} 1, & \text{if } \theta^\intercal x_n \geq 0 \\ -1 & \text{if } \theta^\intercal x_n < 0 \end{cases}$

Model - step `squashing` function and criterion

Linear mapping:

$$f_\theta(\mathbf{x}) = \sum_{d=1}^{D} \theta_d x_d = \theta^\intercal \mathbf{x}, \theta \in \mathbb{R}^D$$

Reminder - logistic regression:

$$p(y_n \,|\, x_n, \theta) = \begin{cases} g_\theta(x_n), & \text{if } y = -1 \\ 1 - g_\theta(x_n), & \text{if } y = 1 \end{cases}$$

$g_\theta(x) = \sigma(f_\theta(x))$, σ is logistic sigmoid

Perceptron model: $y_n = \begin{cases} 1, & \text{if } \theta^\intercal x_n \geq 0 \\ -1 & \text{if } \theta^\intercal x_n < 0 \end{cases}$

Perceptron criterion: Want $y_n \theta^\intercal x_n > 0$ for all n, $L(\theta) = -\sum_{n \in \mathscr{M}} y_n \theta^T x_n$

$\mathscr{M}$: all mis-classified examples

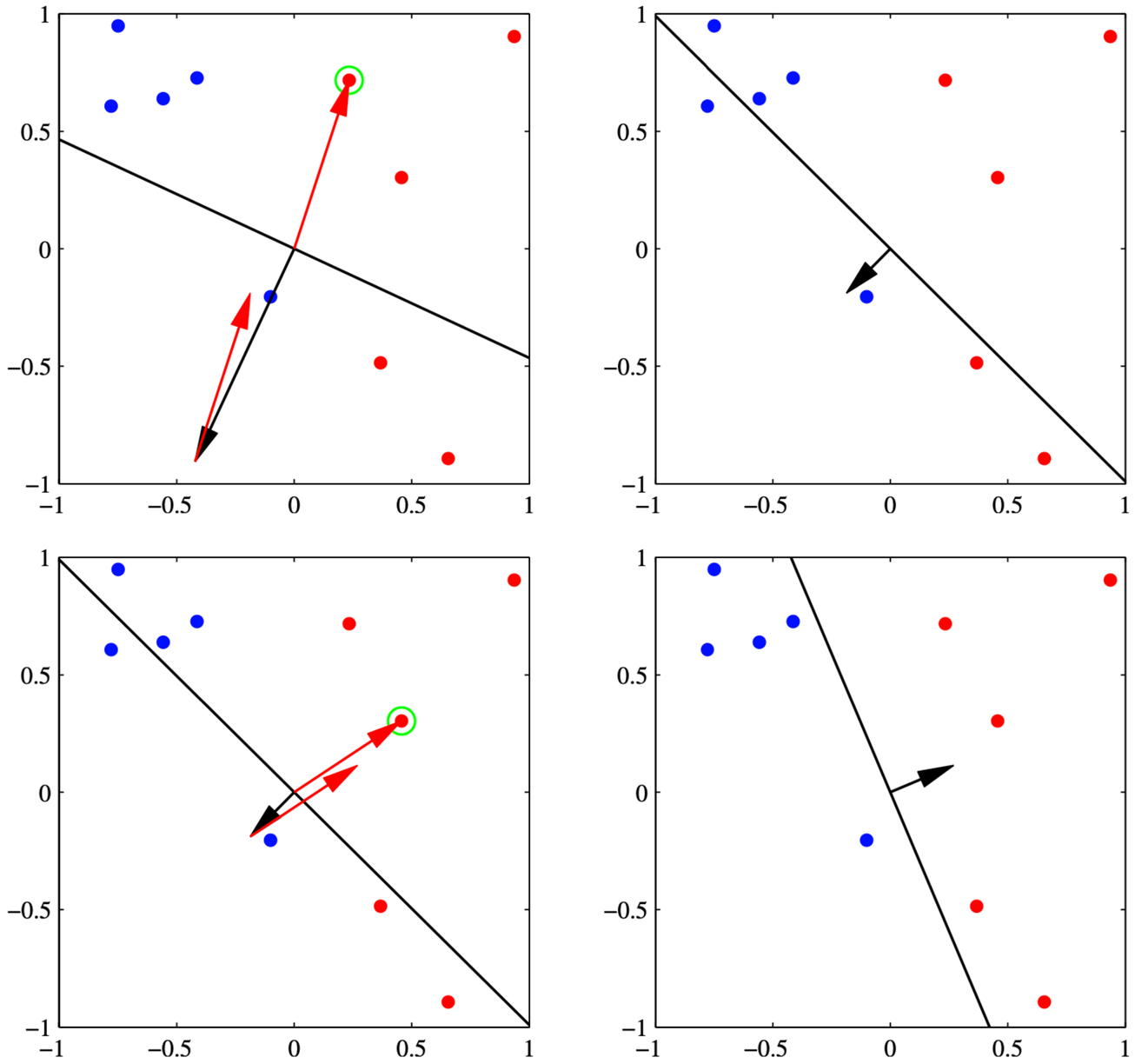
Perceptron - learning algorithm

Perceptron criterion: Want $y_n \theta^\intercal x_n > 0$ for all n, $L(\theta) = -\sum_{n \in \mathcal{M}} y_n \theta^T x_n$

$\mathcal{M}$: all mis-classified examples

SGD update: *for each misclassified example,* $\theta^{t+1} = \theta^t + x_n y_n$. Derive this!

Perceptron algorithm - an example



PRML Bishop

Perceptron - potential issues

Perceptron - potential issues

Perceptron convergence theorem (Rosenblatt 1962): if there exists an exact solution (aka. if the training data set is linearly separable), then the perceptron learning algorithm is guaranteed to find an exact solution in a finite number of steps.

Perceptron - potential issues

Perceptron convergence theorem (Rosenblatt 1962): if there exists an exact solution (aka. if the training data set is linearly separable), then the perceptron learning algorithm is guaranteed to find an exact solution in a finite number of steps.

But the number of steps required to achieve convergence could still be *substantial.*

Perceptron - potential issues

Perceptron convergence theorem (Rosenblatt 1962): if there exists an exact solution (aka. if the training data set is linearly separable), then the perceptron learning algorithm is guaranteed to find an exact solution in a finite number of steps.

But the number of steps required to achieve convergence could still be *substantial.*

Even when the data set is linearly separable, there may be *many* solutions, and which one is found will depend on the *initialisation* of the parameters and on the order of presentation of the data points. Furthermore, for data sets that are not linearly separable, the perceptron learning algorithm will **never** converge.

Perceptron - potential issues

Perceptron convergence theorem (Rosenblatt 1962): if there exists an exact solution (aka. if the training data set is linearly separable), then the perceptron learning algorithm is guaranteed to find an exact solution in a finite number of steps.

But the number of steps required to achieve convergence could still be *substantial.*

Even when the data set is linearly separable, there may be *many* solutions, and which one is found will depend on the *initialisation* of the parameters and on the order of presentation of the data points. Furthermore, for data sets that are not linearly separable, the perceptron learning algorithm will **never** converge.

The perceptron *does not* provide probabilistic outputs, nor does it generalise readily to $K > 2$ classes

Whew :) That's it! Well, not quite :(

- *Assignment 4* is now available on Wattle (due next week - W12 Monday)
- *Exam* timetable is available.
  - Check time + location
  - Past exam papers released on Wattle
- Guest lecture: W12 Monday October 23
  - Dr Zheng Yuan, King's College London
  - *Examinable*!
  - Please do show up!
- Review lecture next Wed