Lecture 2

Training and Testing

University of Amsterdam

Training

Supervised learning:

- We *learn* from examples
  - Training data: inputs and outputs
  - Representation of the input
  - Representation of the output
- Find a function that maps inputs to outputs
  - That also applies to data we've never seen: **generalisation**

Assumption

Both training data and future data are sampled independently from the same distribution (i.i.d.)

  - We cannot consider all functions: **inductive bias**

Training

Once we've chosen our representation and hypothesis space, how do we find our hypothesis?

- Optimise an **objective function**
  - Deal with noise
- Evaluate the machine's performance

Classification: a first attempt

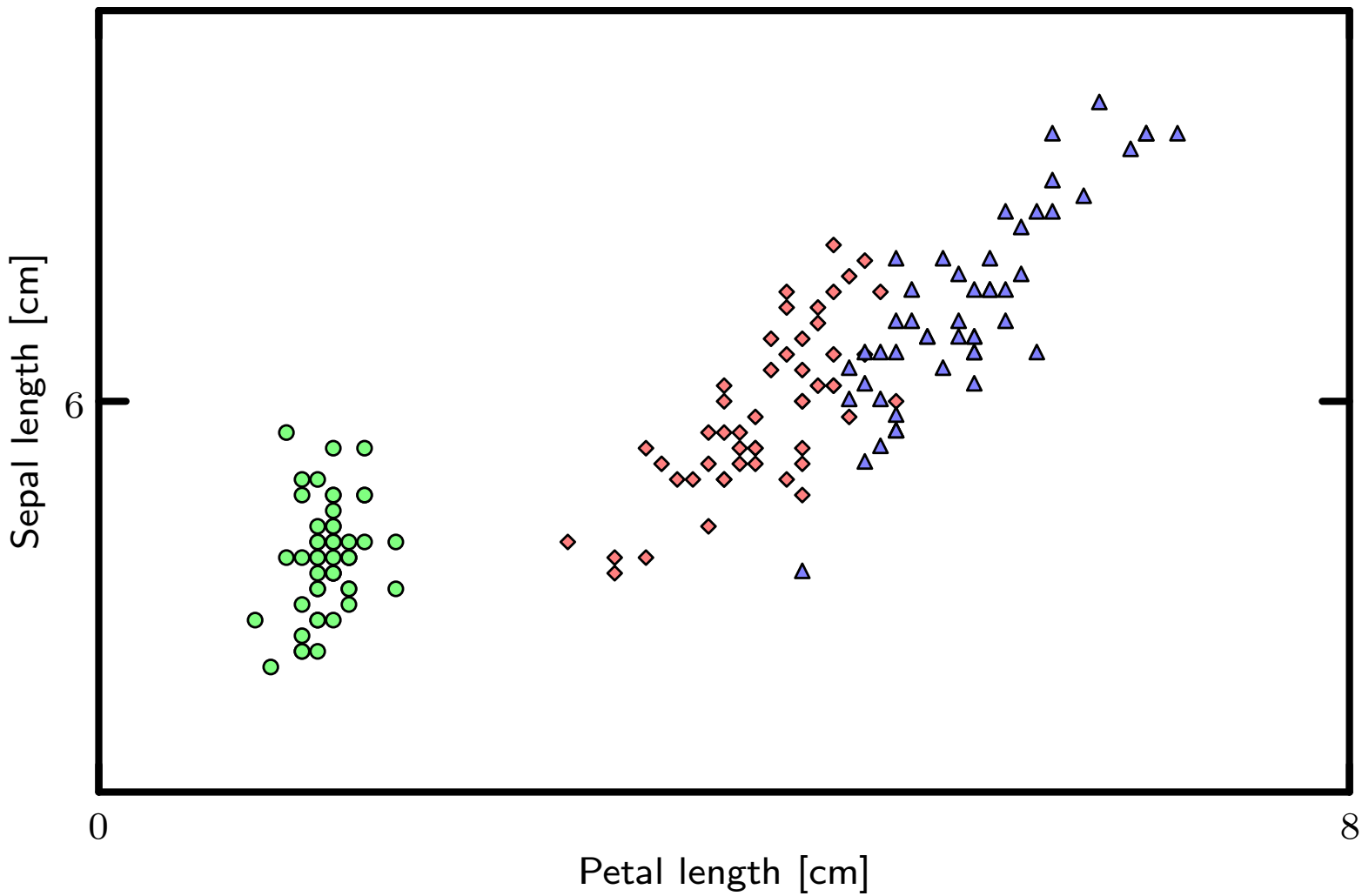
Problem:

- Our input space is vast
- We cannot store all possible input values and evaluate new examples by checking whether we've already seen them

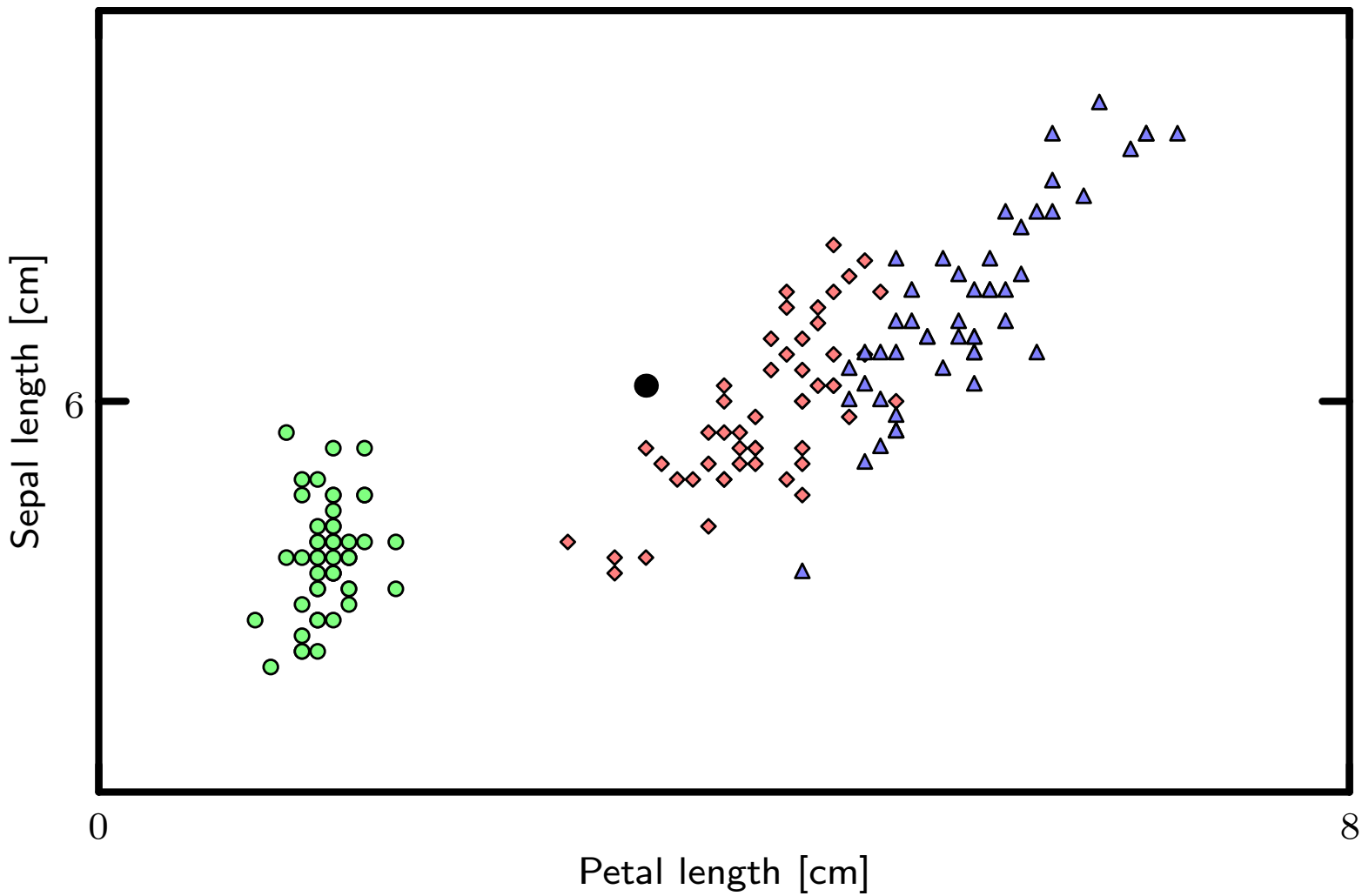
A simple solution: Nearest Neighbours

- Store the training data
- When evaluating new data, find the closest training element and classify according to that element's class
- Closest: Euclidean distance

k Nearest Neighbours



k Nearest Neighbours



About kNN

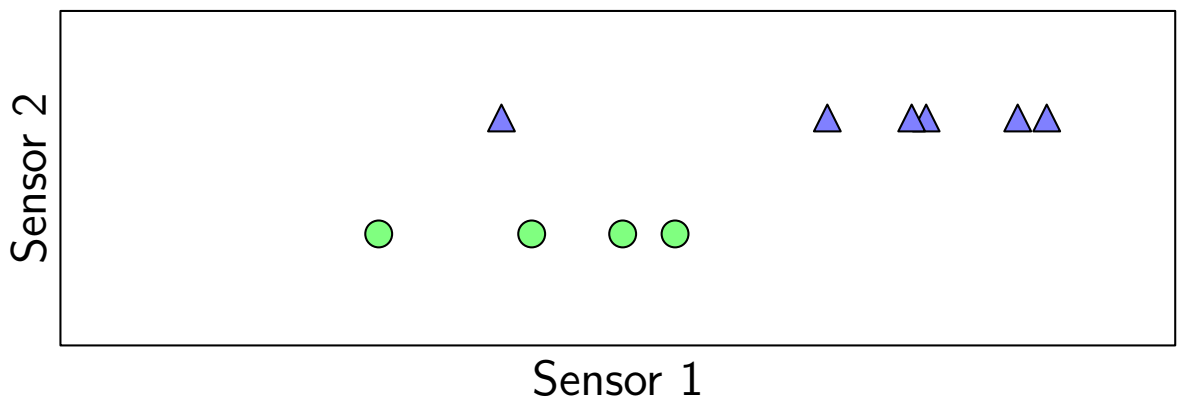
- The Good:
  - Simple and intuitive
  - Powerful — often performs well, even on "difficult" data
  - Provides an excellent baseline for more complicated techniques
- The Bad:
  - Requires storage of the complete training set
  - Requires expensive search
  - Problems with Euclidean distance

About kNN

- The Good:
  - Simple and intuitive
  - Powerful — often performs well, even on "difficult" data
  - Provides an excellent baseline for more complicated techniques
- The Bad:
  - Requires storage of the complete training set
  - Requires expensive search
  - Problems with Euclidean distance

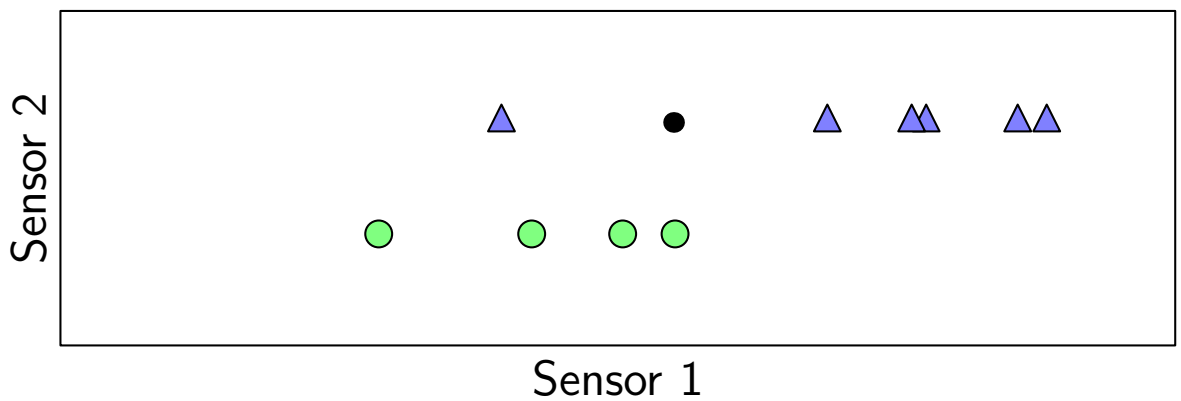
About kNN

- The Good:
  - Simple and intuitive
  - Powerful — often performs well, even on "difficult" data
  - Provides an excellent baseline for more complicated techniques
- The Bad:
  - Requires storage of the complete training set
  - Requires expensive search
  - Problems with Euclidean distance



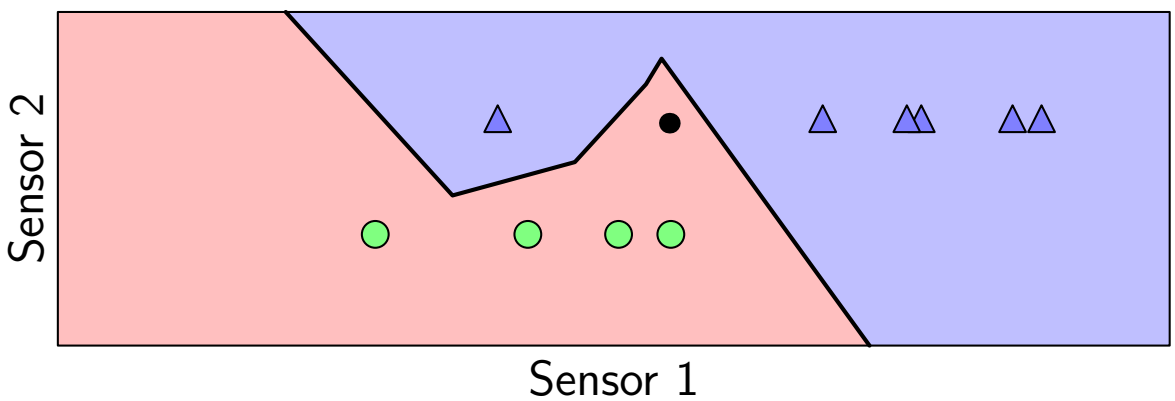
About kNN

- The Good:
  - Simple and intuitive
  - Powerful — often performs well, even on "difficult" data
  - Provides an excellent baseline for more complicated techniques
- The Bad:
  - Requires storage of the complete training set
  - Requires expensive search
  - Problems with Euclidean distance

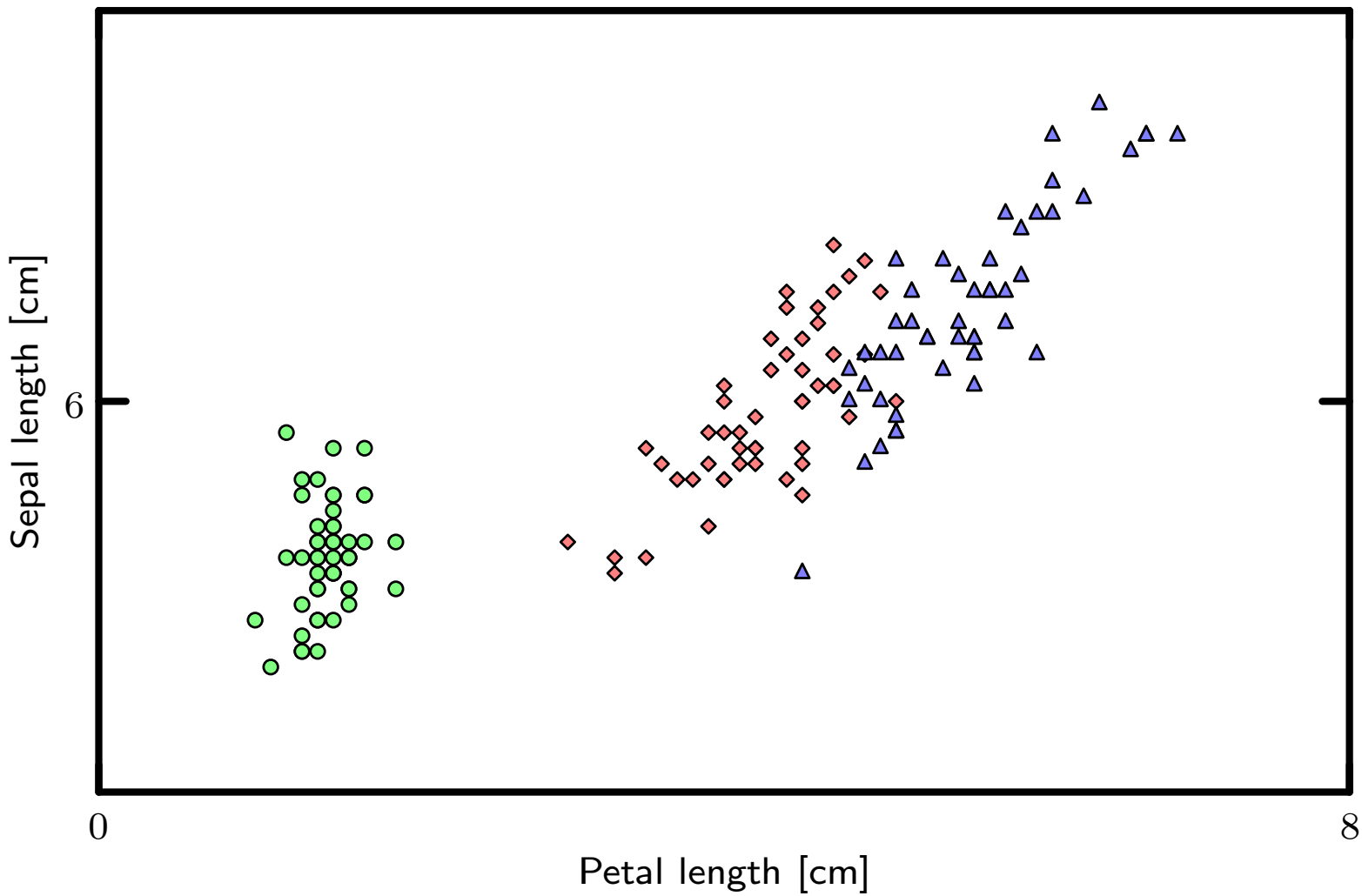


About kNN

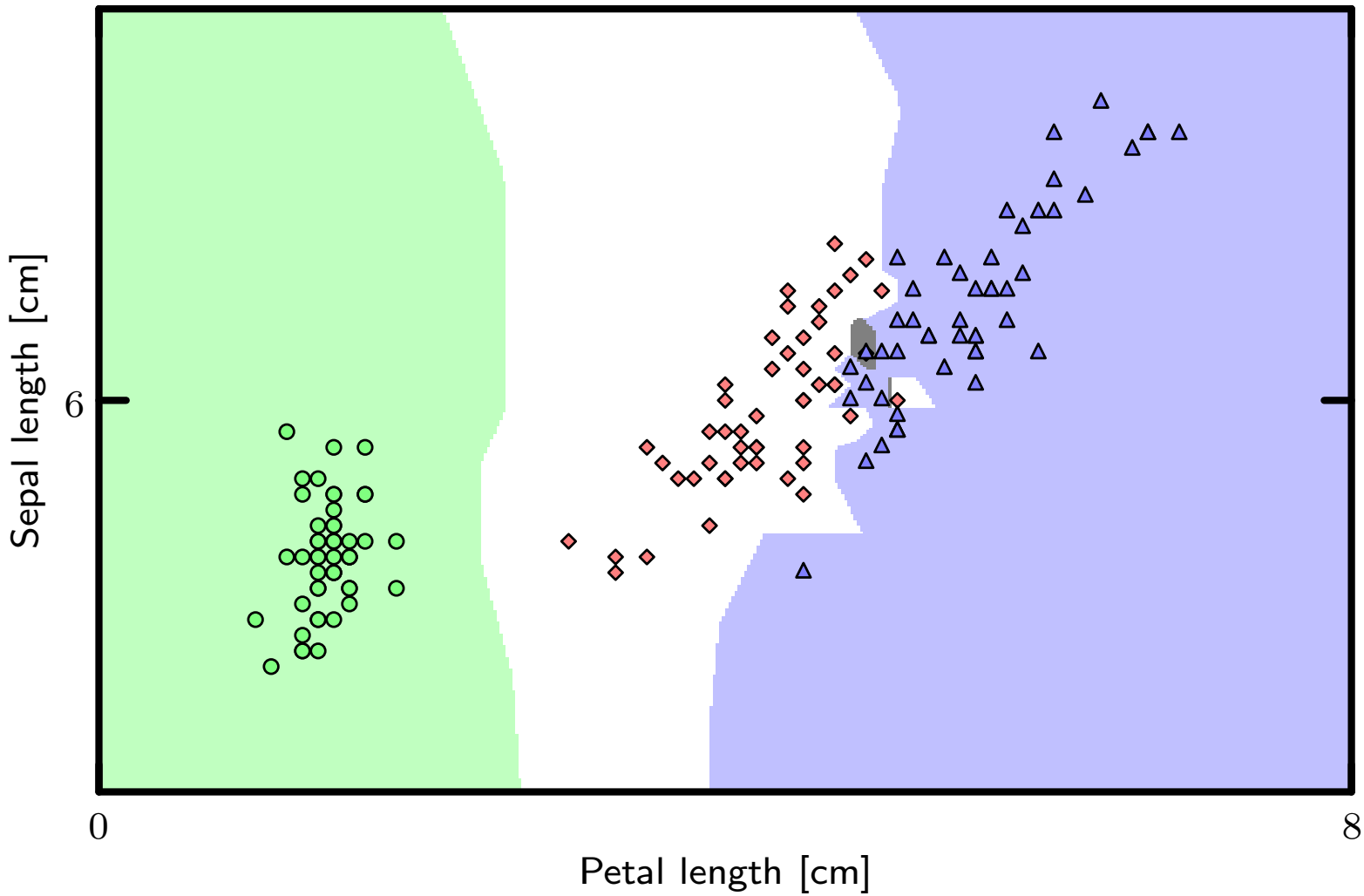
- The Good:
  - Simple and intuitive
  - Powerful — often performs well, even on "difficult" data
  - Provides an excellent baseline for more complicated techniques
- The Bad:
  - Requires storage of the complete training set
  - Requires expensive search
  - Problems with Euclidean distance



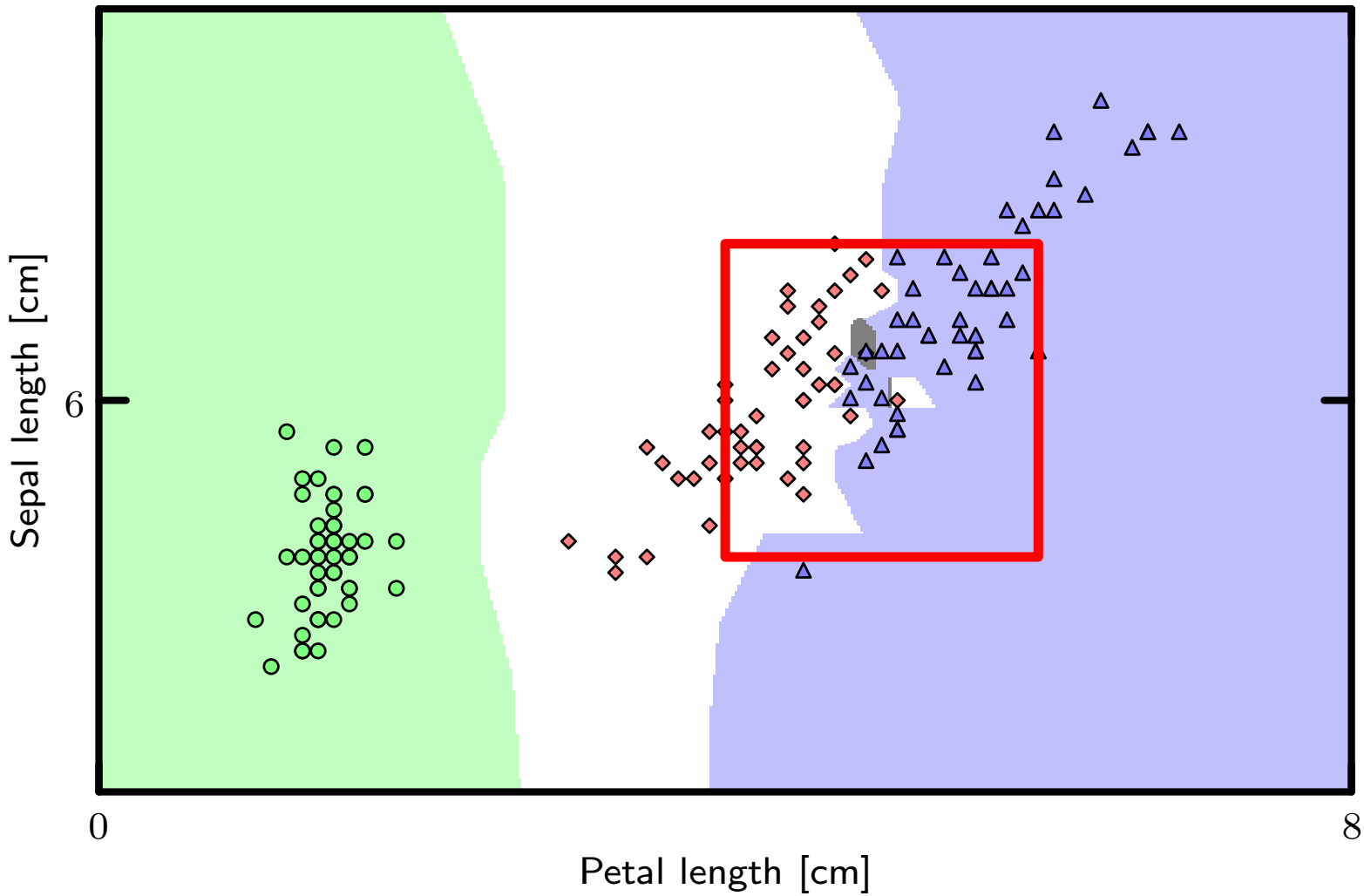
So what are we doing here?



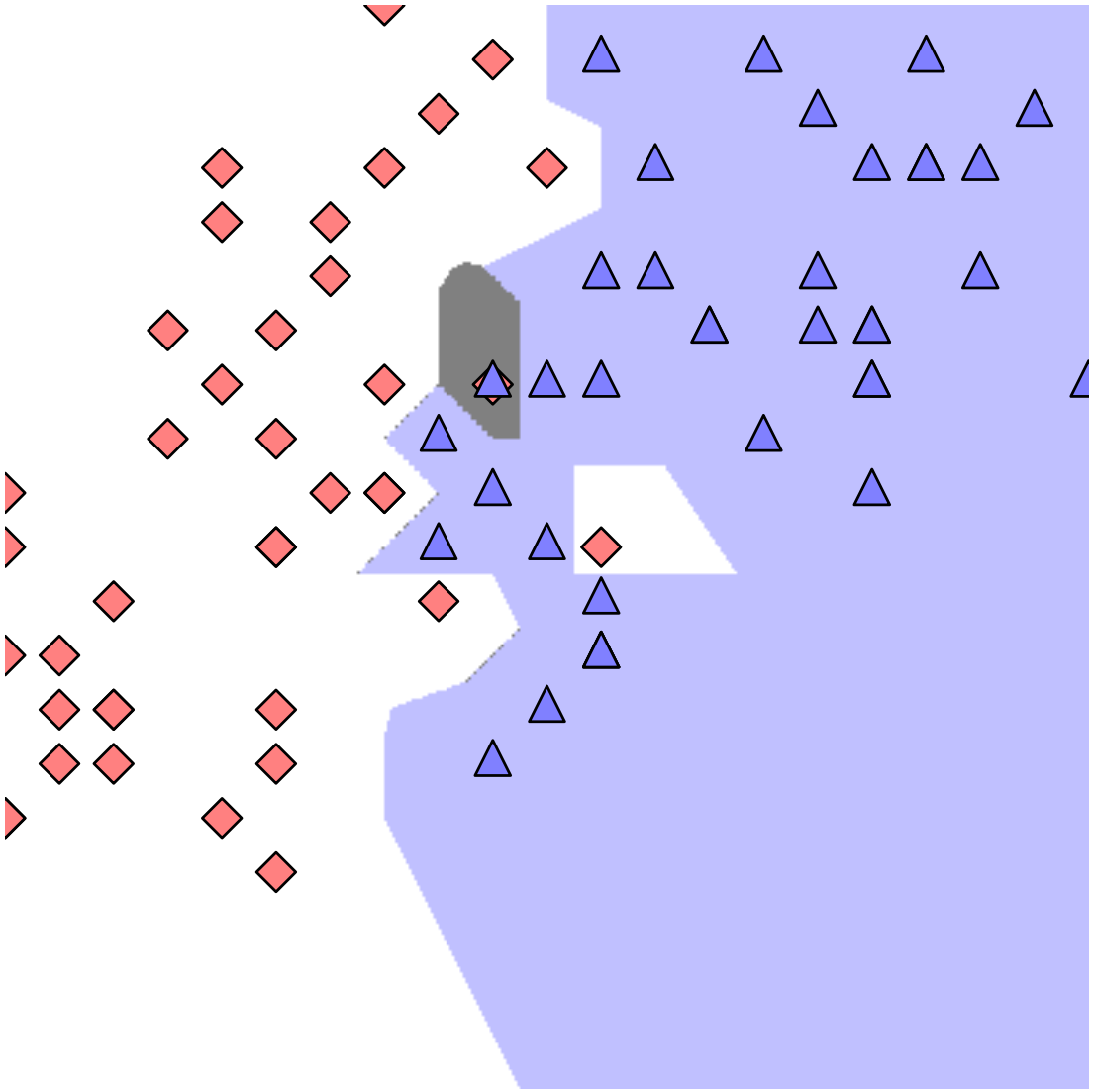
So what are we doing here?



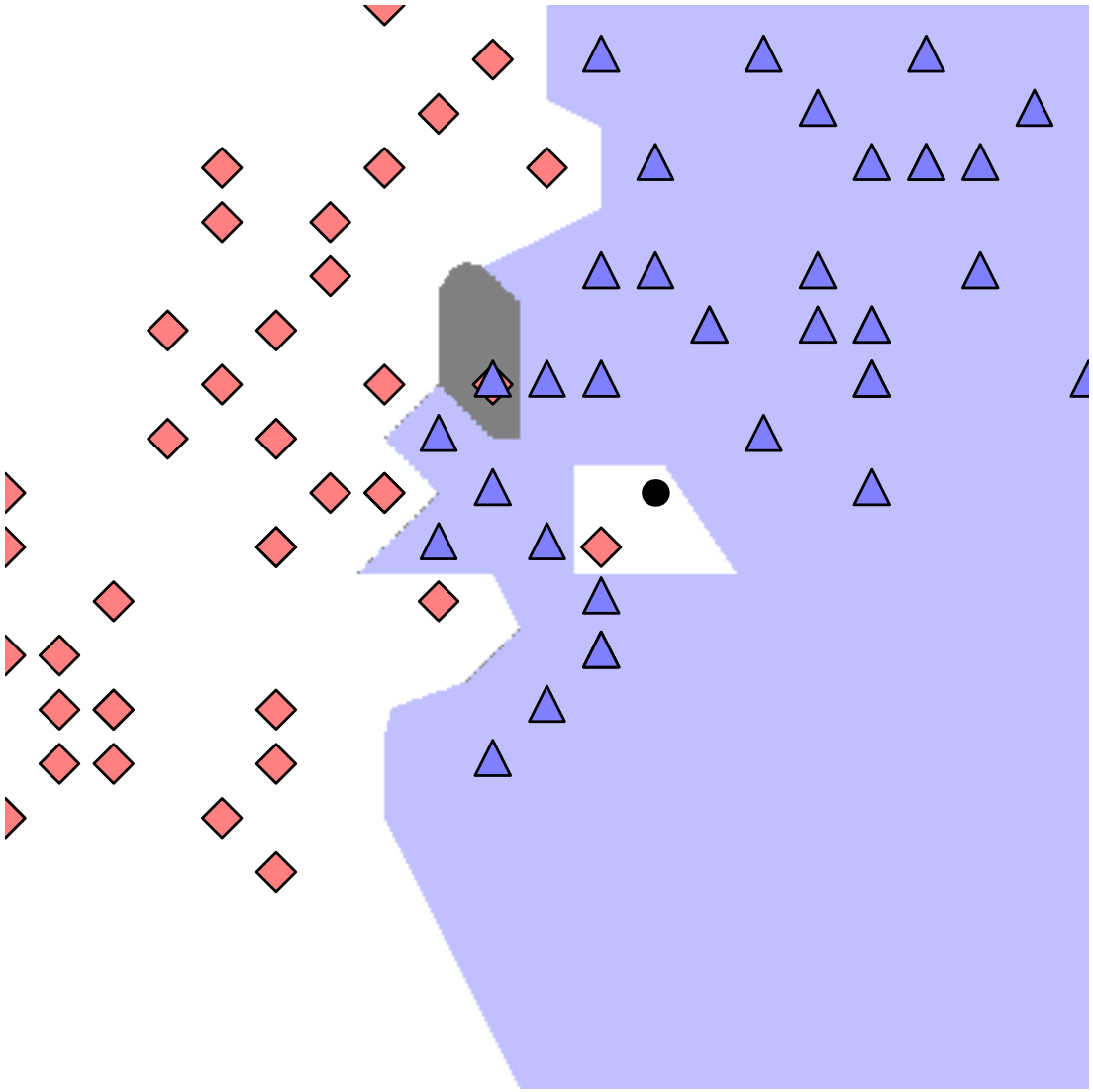
So what are we doing here?



So what are we doing here?



Overfitting



kNN: Why does it work (and why does it fail?)

We split up our space into regions, where we assume that the probability of finding datapoints of a given class is approximately constant.
Estimate the probability: count number of elements in region.
This can be done by:

1. Creating cells of equal volume
2. Growing regions to contain exactly k elements

kNN: Why does it work (and why does it fail?)

If our regions are sufficiently small, and if our number of elements in each region is sufficiently large, we can estimate the density within a region as:

$$p(\mathbf{x}) = \frac{k}{NV} \tag{1}$$

where

- k is the number of elements in the region,
- N is the total number of elements,
- V is the volume of the region

kNN: Why does it work (and why does it fail?)

The estimation of $p(\mathbf{x}) = \frac{k}{NV}$ is based on two contradictory requirements:

- $\mathcal{R}$ is small, so that $p(\mathbf{x})$ is approx. constant in the region
- $\mathcal{R}$ is large, so that k large enough.

In kNN, we estimate $p(\mathbf{x})$ by keeping k constant and growing V until it contains exactly k prototypes. Therefore:

- if k is too small, V will be too small and our estimate of $p(\mathbf{x})$ will be bad
- if V is too large, $p(\mathbf{x})$ will not be constant in the region, and our estimate will be bad.

Error measures: classification

- Confusion matrix (m classes)

$$\begin{array}{cc} & \text{Estimated class} \\ & \begin{array}{ccc} \mathcal{C}_1 & \dots & \mathcal{C}_m \end{array} \\ \text{True class} \begin{array}{c} \mathcal{C}_1 \\ \vdots \\ \mathcal{C}_m \end{array} & \begin{bmatrix} n_{11} & \dots & n_{1m} \\ \vdots & \ddots & \vdots \\ n_{m1} & \dots & n_{mm} \end{bmatrix} \end{array}$$

- Error measures:

$$\text{Accuracy} = \#_{correct}/\#_{datapoints} = \frac{\sum_i n_{ii}}{\sum_{ij} n_{ij}}$$

$$\text{Error rate} = 1 - \text{accuracy}$$

Two class problems

- Confusion Matrix

		Estimated Positive	Negative
True	Positive (P)	TP	FN
	Negative (N)	FP	TN

- Performance measures

$$\begin{aligned}
\text{Accuracy} \quad & A = \frac{TP+TN}{P+N} \\
\text{Error rate} \quad & = 1 - A = \frac{FP+FN}{P+N} \\
\text{Precision} \quad & = \frac{TP}{TP+FP} \\
\text{Recall} \quad & = \frac{TP}{TP+FN} \\
\text{f-measure} \quad & = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}
\end{aligned}$$

Decision Threshold

- Sometimes we don't want to "just" minimise the error rate

Example: Cancer diagnosis

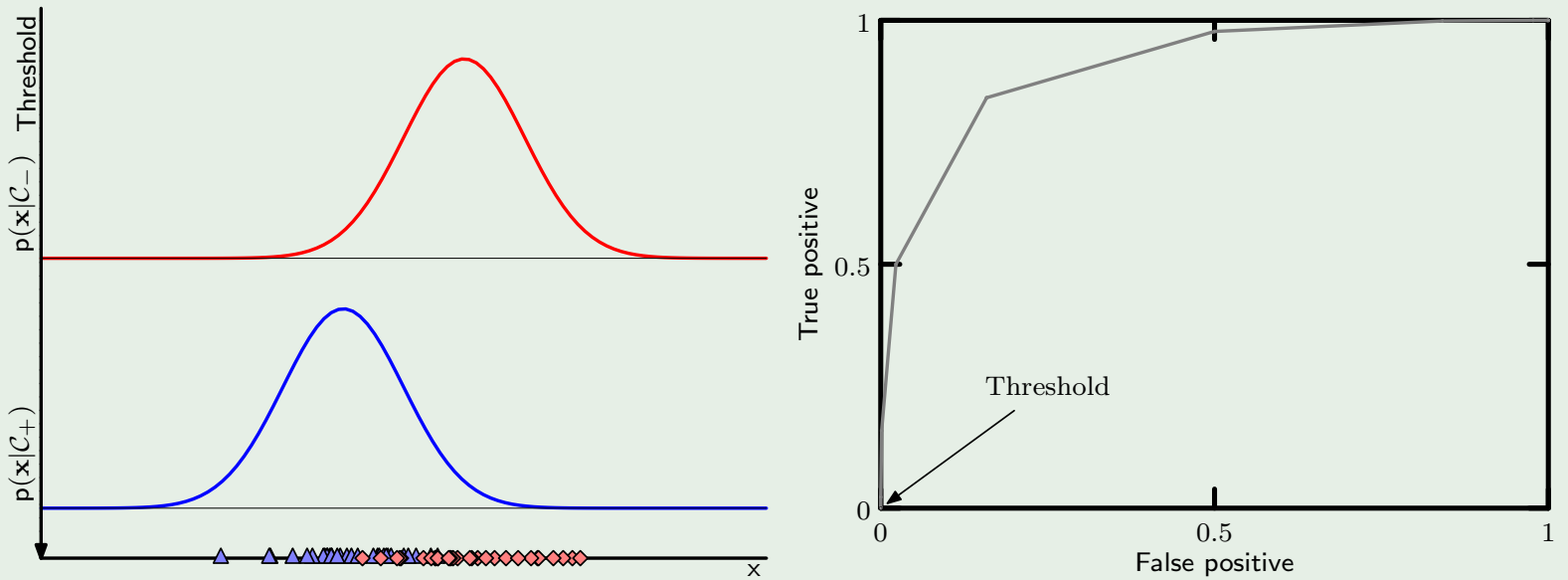
Misclassifying a diseased person as healthy (FN) results in death, while misclassifying a healthy person (FP) results in additional tests.

- Cost/Loss function: weight the errors according to type
  - Example: loss matrix $L = \begin{array}{c} \\ \text{Cancer} \\ \text{Normal} \end{array} \begin{array}{c} \begin{array}{cc} \text{Cancer} & \text{Normal} \end{array} \\ \begin{bmatrix} 0 & 1000 \\ 1 & 0 \end{bmatrix} \end{array}$
  - Minimise the expected loss: for each $\mathbf{x}$, assign the class j for which $\sum_k L_{kj} p(\mathcal{C}_k|\mathbf{x})$ is minimal

Receiver Operating Characteristic (ROC)

- Plot of True Positive Rate against False Positive Rate
- Each point of the curve corresponds to a different threshold

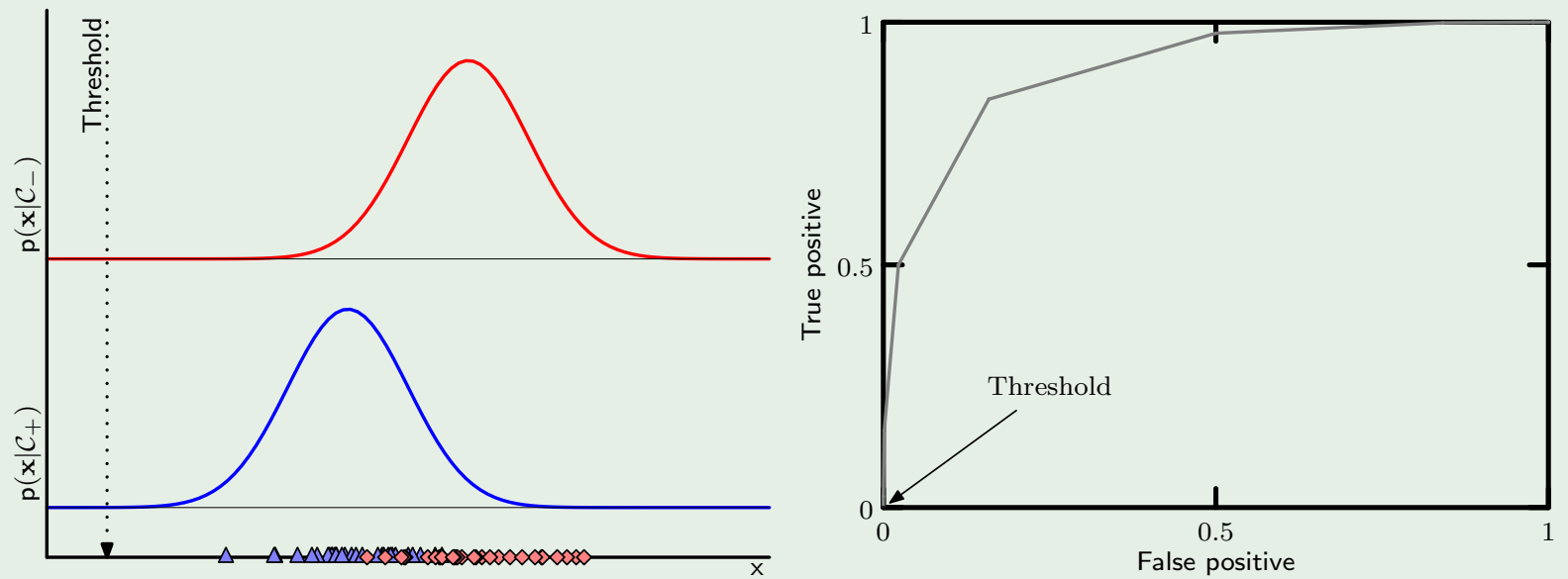
Example



Receiver Operating Characteristic (ROC)

- Plot of True Positive Rate against False Positive Rate
- Each point of the curve corresponds to a different threshold

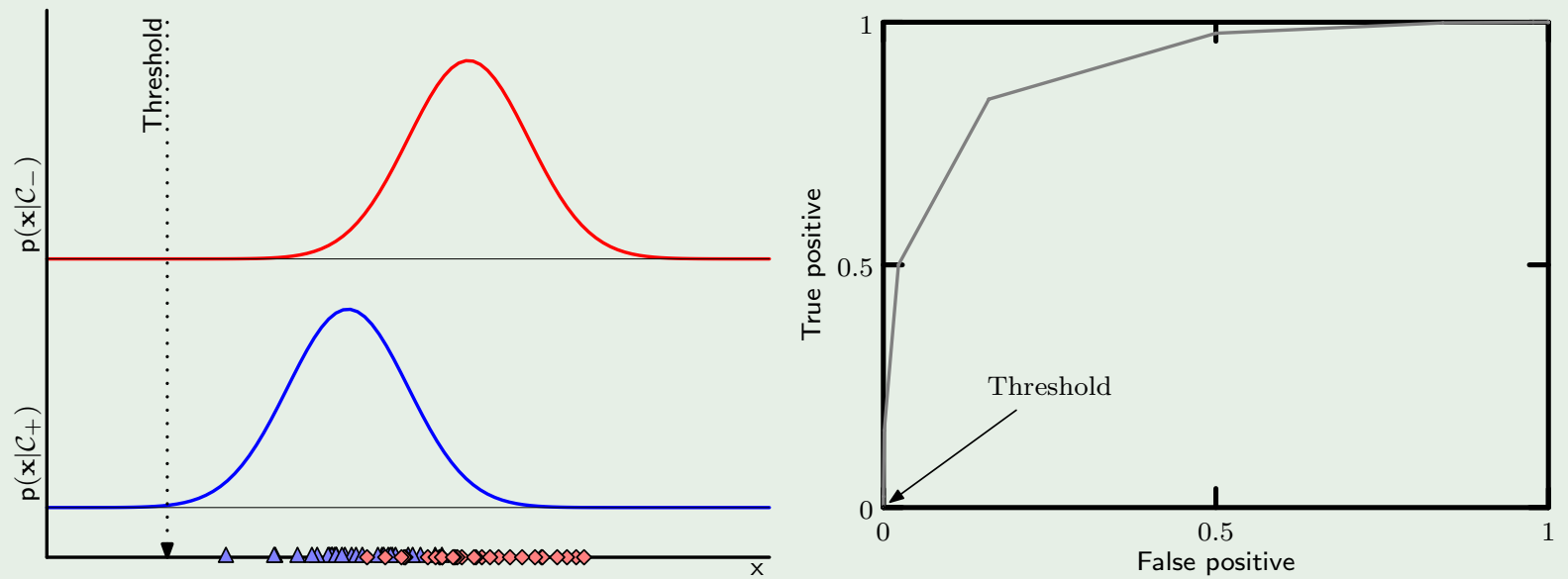
Example



Receiver Operating Characteristic (ROC)

- Plot of True Positive Rate against False Positive Rate
- Each point of the curve corresponds to a different threshold

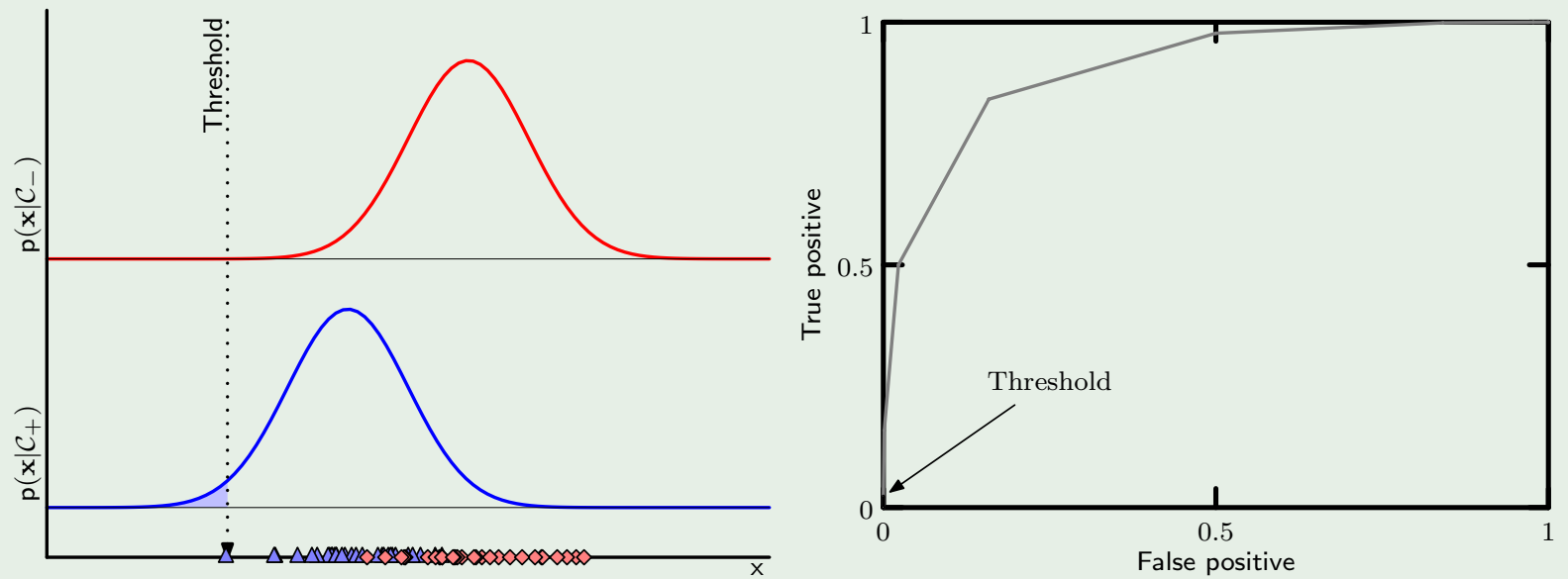
Example



Receiver Operating Characteristic (ROC)

- Plot of True Positive Rate against False Positive Rate
- Each point of the curve corresponds to a different threshold

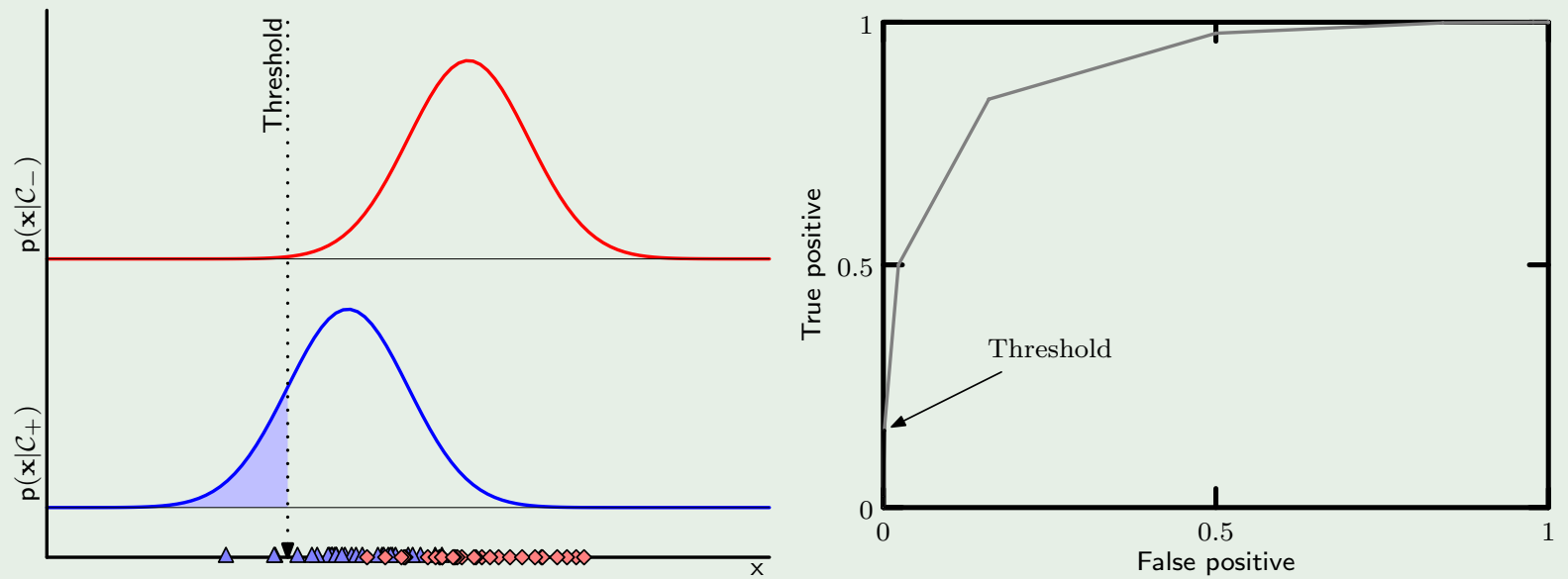
Example



Receiver Operating Characteristic (ROC)

- Plot of True Positive Rate against False Positive Rate
- Each point of the curve corresponds to a different threshold

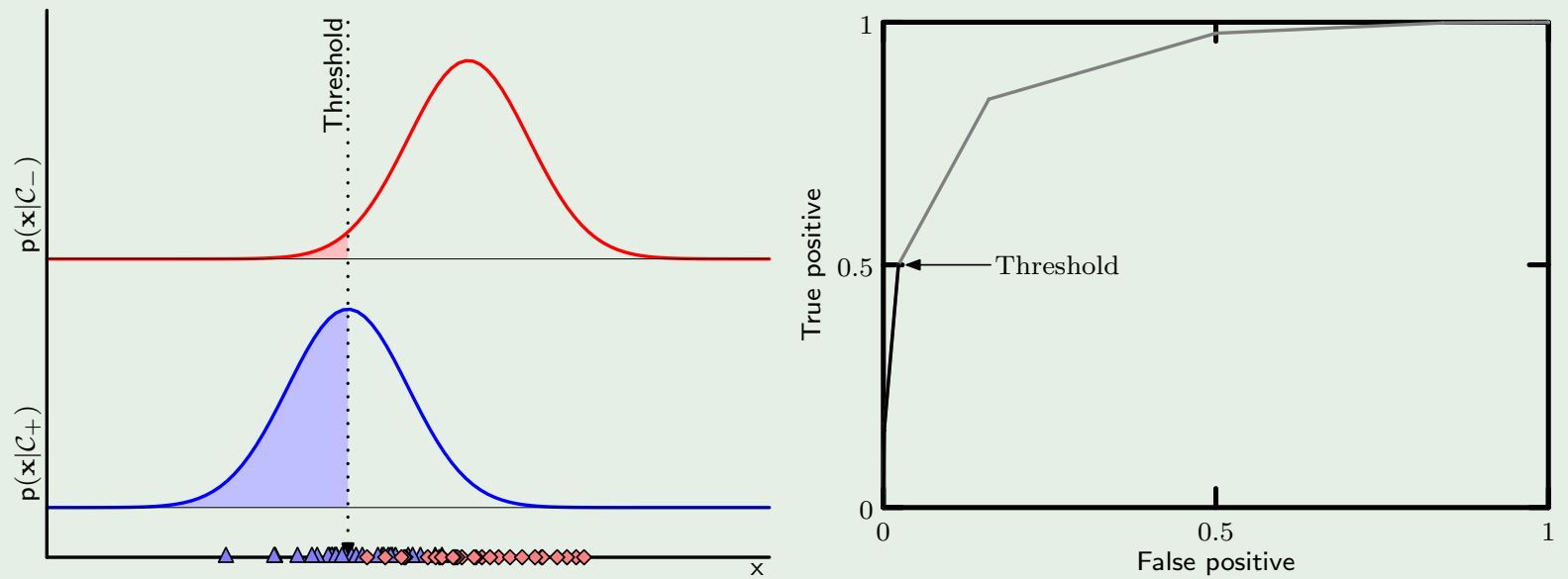
Example



Receiver Operating Characteristic (ROC)

- Plot of True Positive Rate against False Positive Rate
- Each point of the curve corresponds to a different threshold

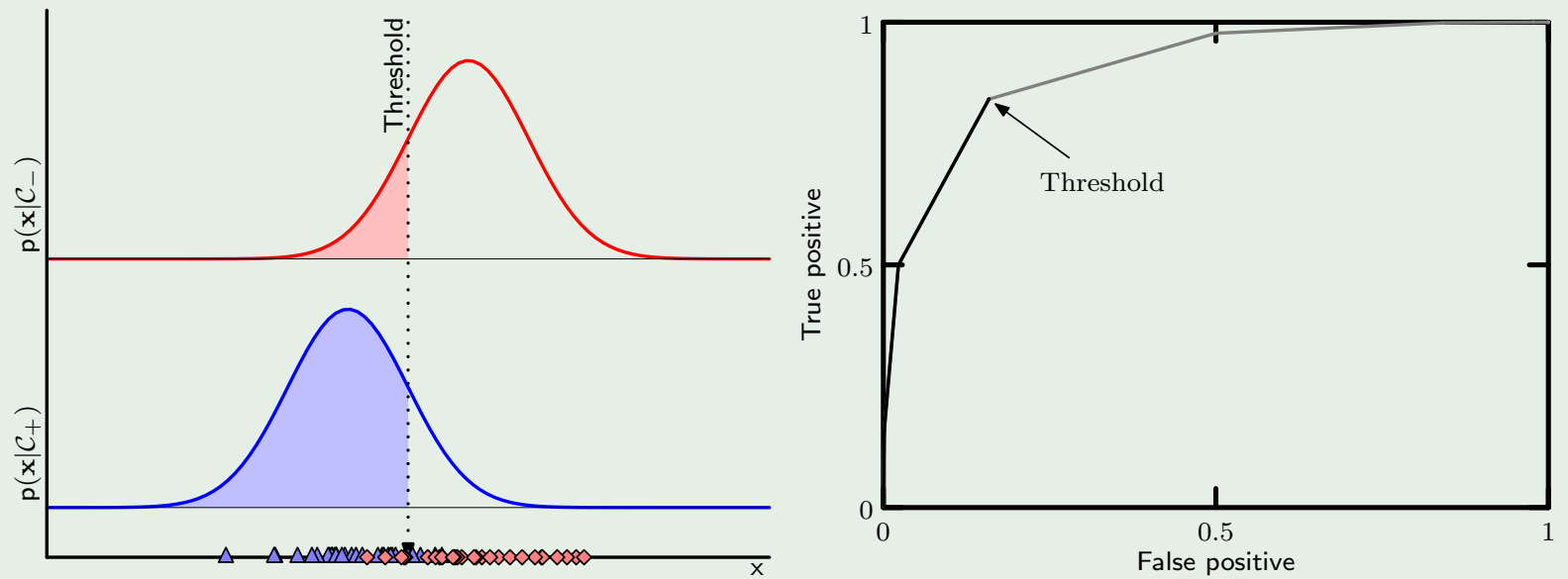
Example



Receiver Operating Characteristic (ROC)

- Plot of True Positive Rate against False Positive Rate
- Each point of the curve corresponds to a different threshold

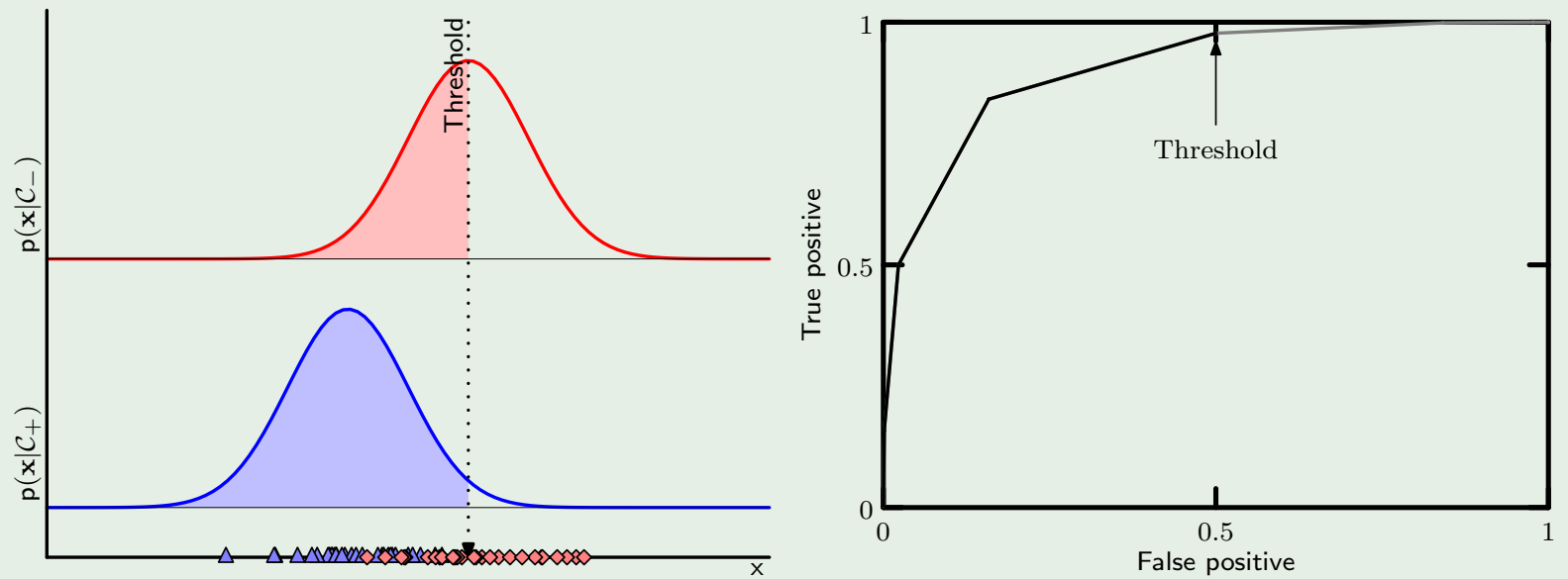
Example



Receiver Operating Characteristic (ROC)

- Plot of True Positive Rate against False Positive Rate
- Each point of the curve corresponds to a different threshold

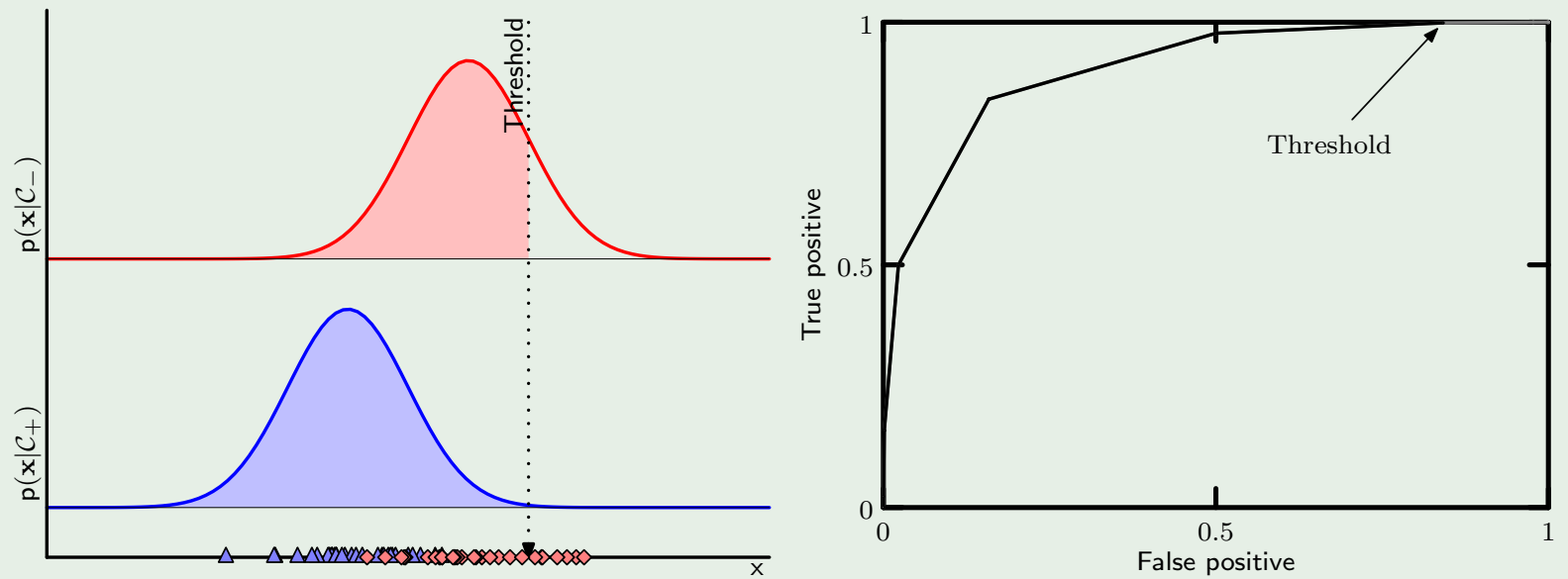
Example



Receiver Operating Characteristic (ROC)

- Plot of True Positive Rate against False Positive Rate
- Each point of the curve corresponds to a different threshold

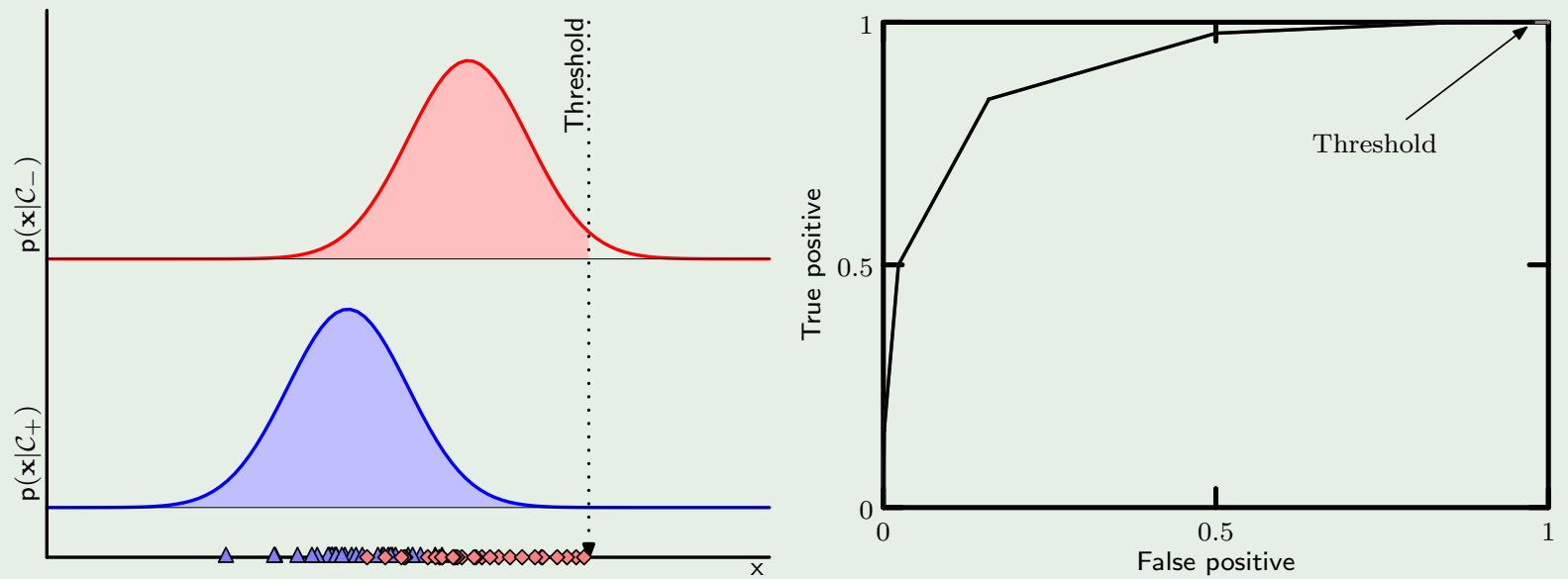
Example



Receiver Operating Characteristic (ROC)

- Plot of True Positive Rate against False Positive Rate
- Each point of the curve corresponds to a different threshold

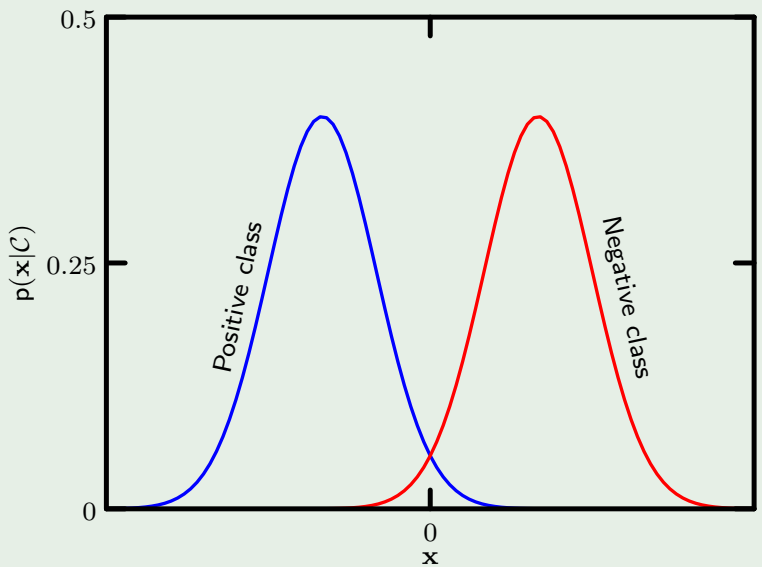
Example

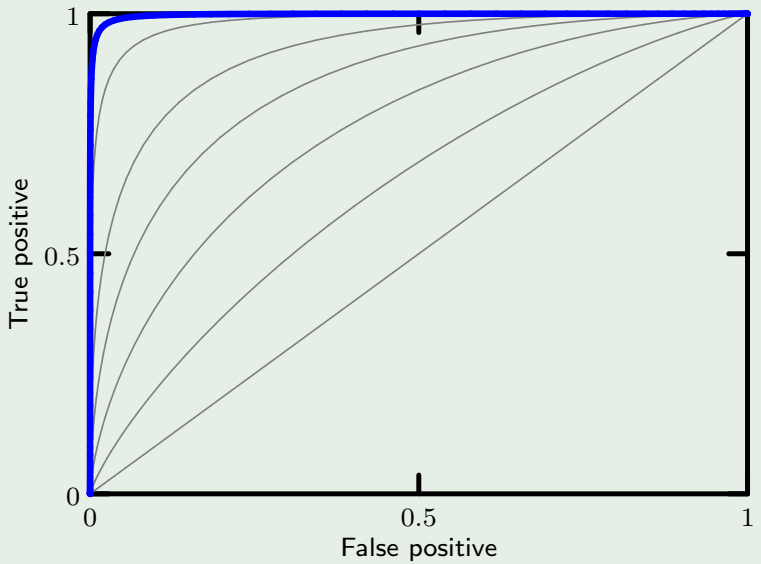


Area under the curve (AUC)

- The ROC gives a measure of the classification performance
- The Area Under the Curve reflects how well the classifier performs
  - Independently from the specific cost function used

Example

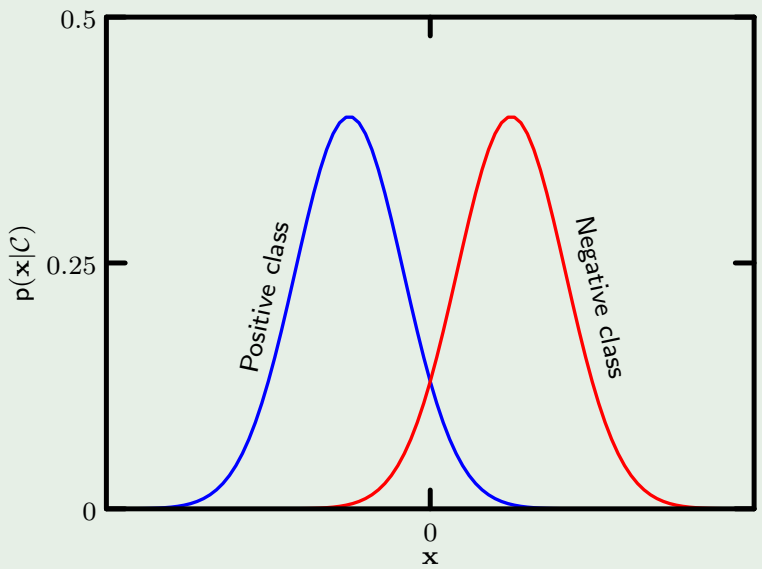


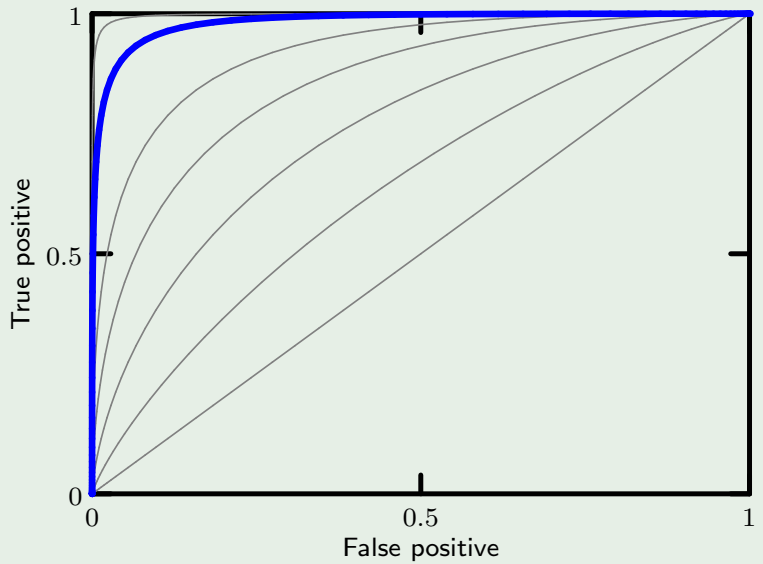


Area under the curve (AUC)

- The ROC gives a measure of the classification performance
- The Area Under the Curve reflects how well the classifier performs
  - Independently from the specific cost function used

Example

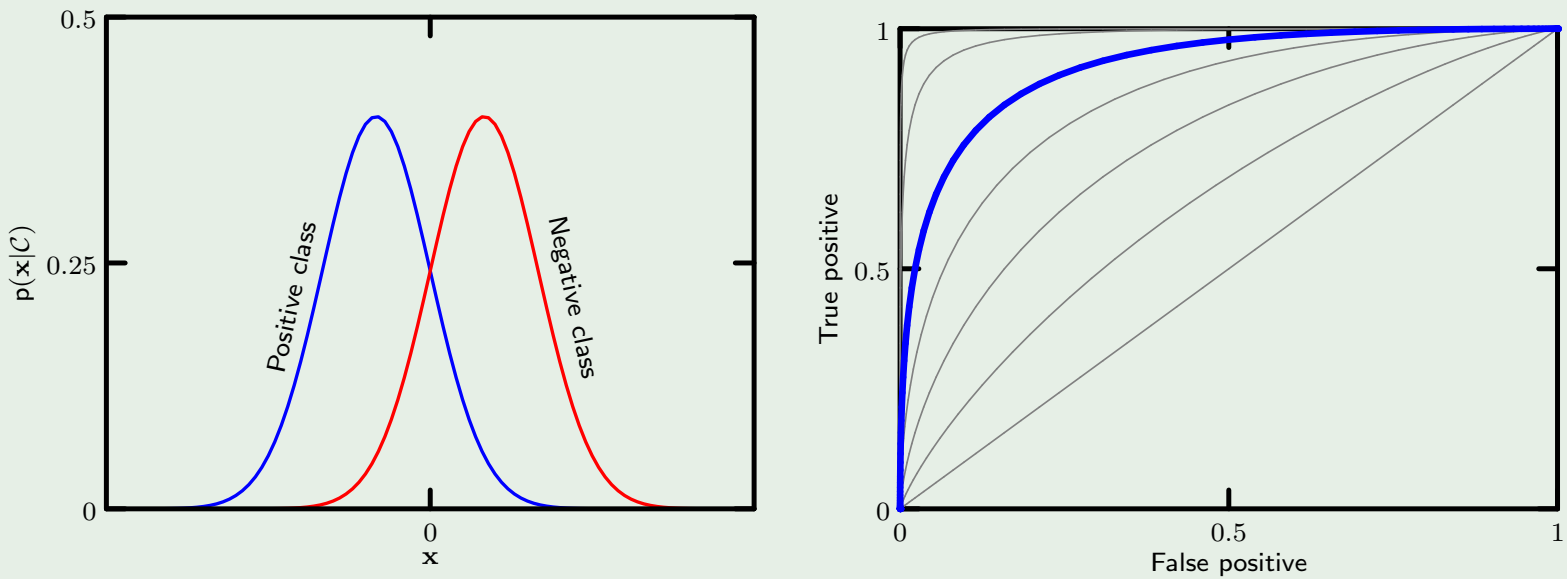




Area under the curve (AUC)

- The ROC gives a measure of the classification performance
- The Area Under the Curve reflects how well the classifier performs
  - Independently from the specific cost function used

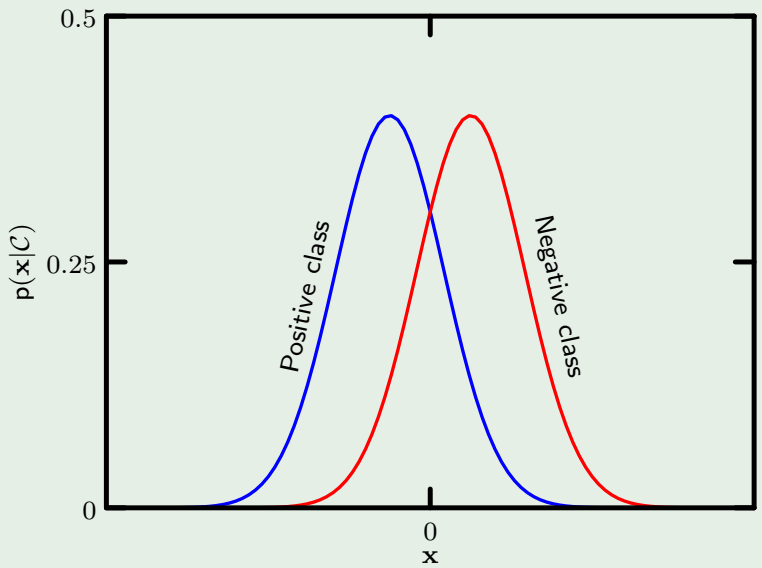
Example

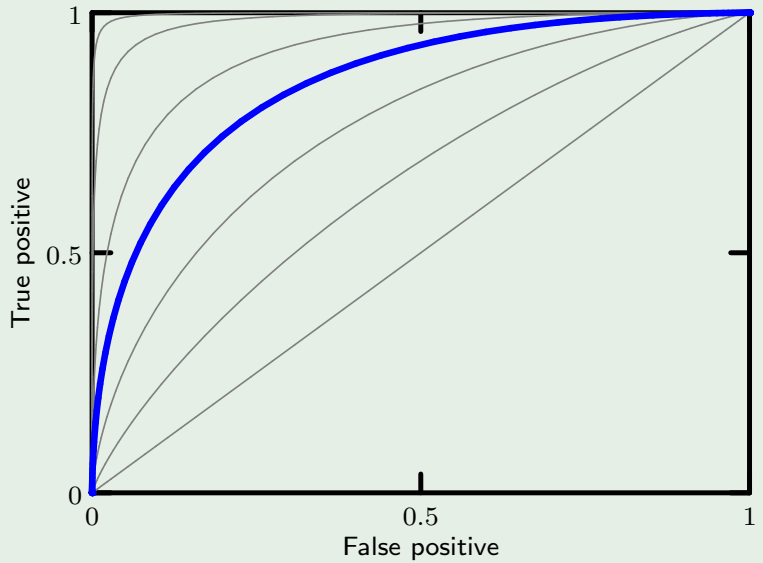


Area under the curve (AUC)

- The ROC gives a measure of the classification performance
- The Area Under the Curve reflects how well the classifier performs
  - Independently from the specific cost function used

Example

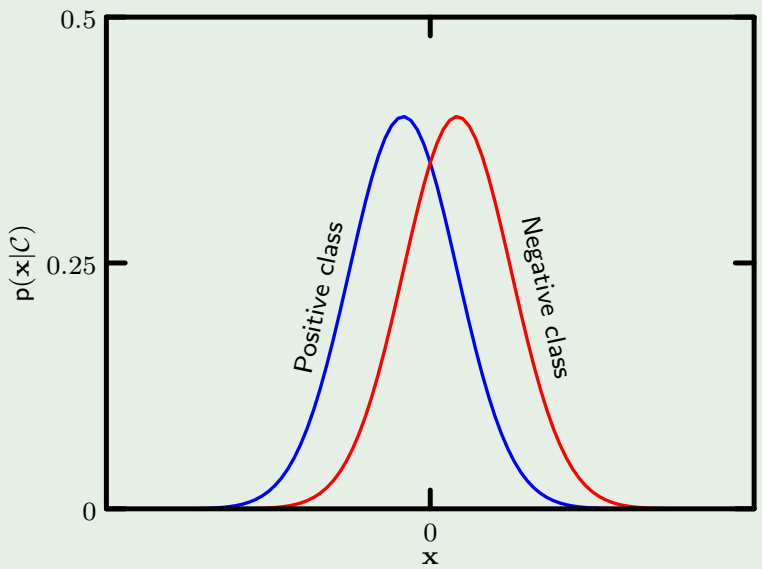


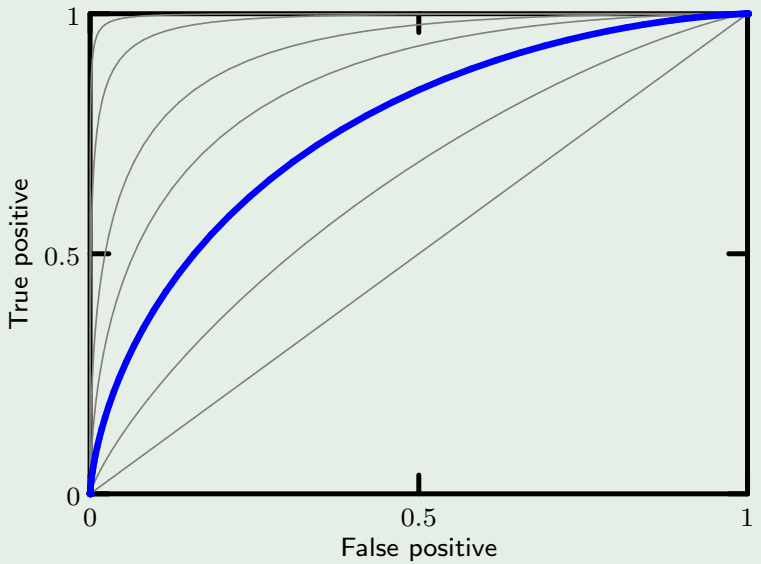


Area under the curve (AUC)

- The ROC gives a measure of the classification performance
- The Area Under the Curve reflects how well the classifier performs
  - Independently from the specific cost function used

Example

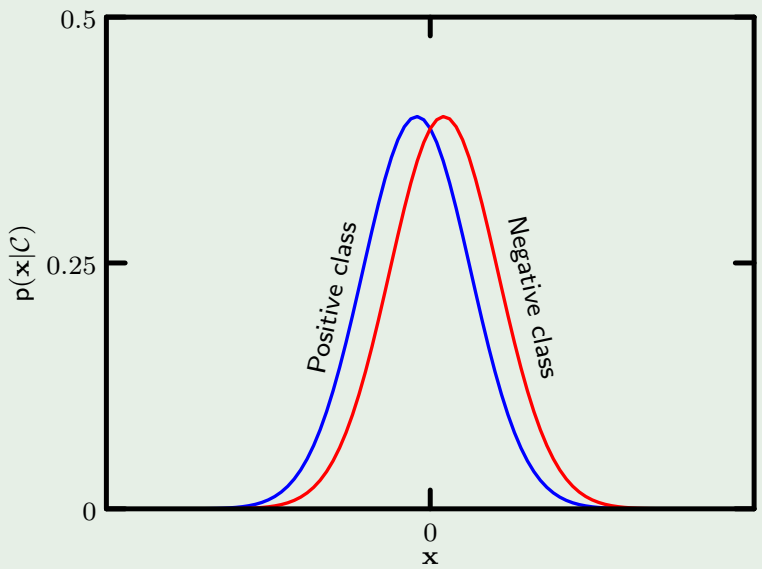


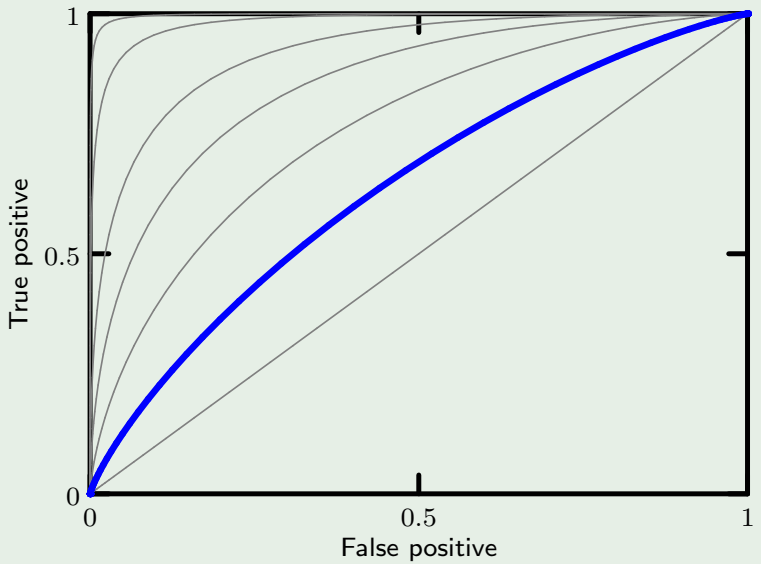


Area under the curve (AUC)

- The ROC gives a measure of the classification performance
- The Area Under the Curve reflects how well the classifier performs
  - Independently from the specific cost function used

Example

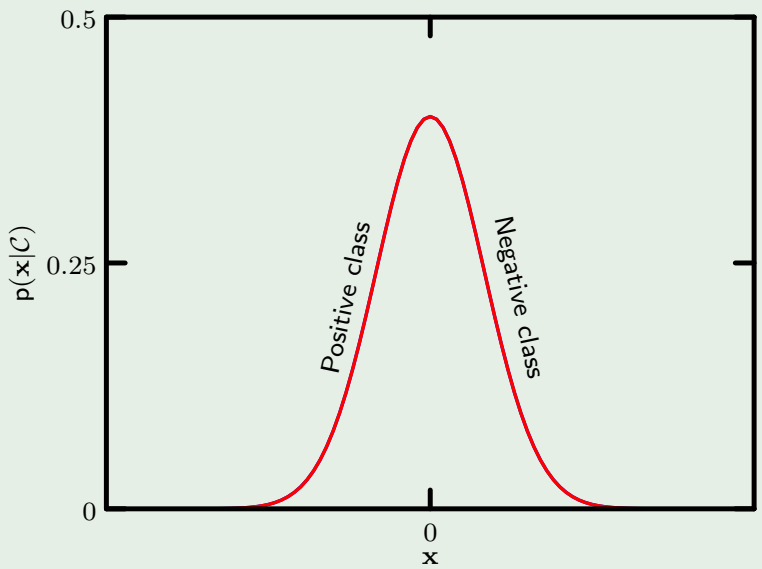


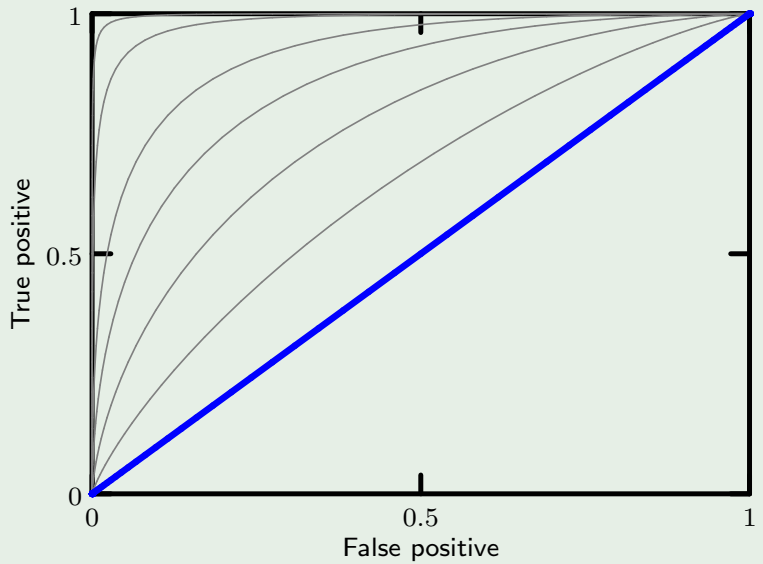


Area under the curve (AUC)

- The ROC gives a measure of the classification performance
- The Area Under the Curve reflects how well the classifier performs
  - Independently from the specific cost function used

Example





Error measures: regression

Typically: **sum-of-squares** error function:

$$E_{SSE}(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\left(y(x_n, \mathbf{w}) - t_n\right)^2 \tag{2}$$

- Minimising the SSE is equivalent with maximising the log-likelihood under the assumption of zero-mean Gaussian noise.

Sometimes more convenient: **root-mean-square** error:

$$E_{RMS}(\mathbf{w}) = \sqrt{2E(\mathbf{w})/N} \tag{3}$$

- Square root ensures that the error has same scale as target
- Division by N allows comparison over data sets of different size

Overfitting and Generalisation

- Generalisation: learn, from known examples, about unseen examples
- Overfitting: learn properties from the given examples which do not apply to unseen examples
- Evaluate on separate set

Example: polynomial regression

- Process: $y = \sin(2\pi x)$
- Observations: corrupted by Gaussian noise ▸ def:

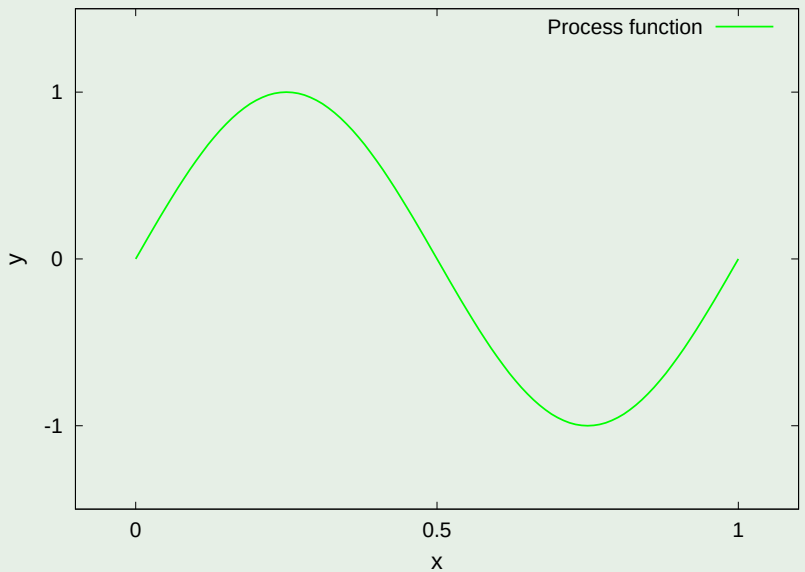
$$y = \sin(2\pi x) + \xi$$

with

$$\xi \sim \mathcal{N}(0, 0.3)$$

- Attempt to recover a description of the process, using a polynomial function

$$y = w_0 + w_1 x + w_2 x^2 + \cdots$$



Example: polynomial regression

- Process: $y = \sin(2\pi x)$
- Observations: corrupted by Gaussian noise ▸ def:

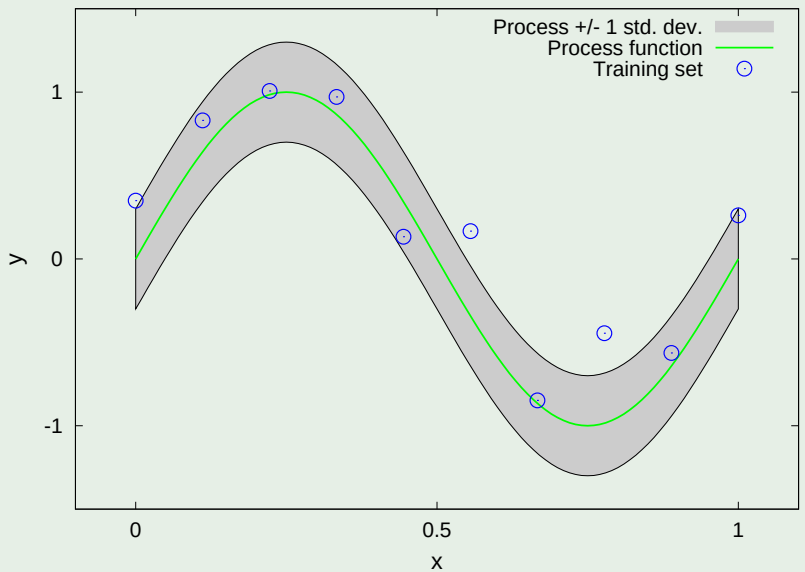
$$y = \sin(2\pi x) + \xi$$

with

$$\xi \sim \mathcal{N}(0, 0.3)$$

- Attempt to recover a description of the process, using a polynomial function

$$y = w_0 + w_1 x + w_2 x^2 + \cdots$$



Example: polynomial regression

- Process: $y = \sin(2\pi x)$
- Observations: corrupted by Gaussian noise ▸ def:

$$y = \sin(2\pi x) + \xi$$

with

$$\xi \sim \mathcal{N}(0, 0.3)$$

- Attempt to recover a description of the process, using a polynomial function

$$y = w_0 + w_1 x + w_2 x^2 + \cdots$$

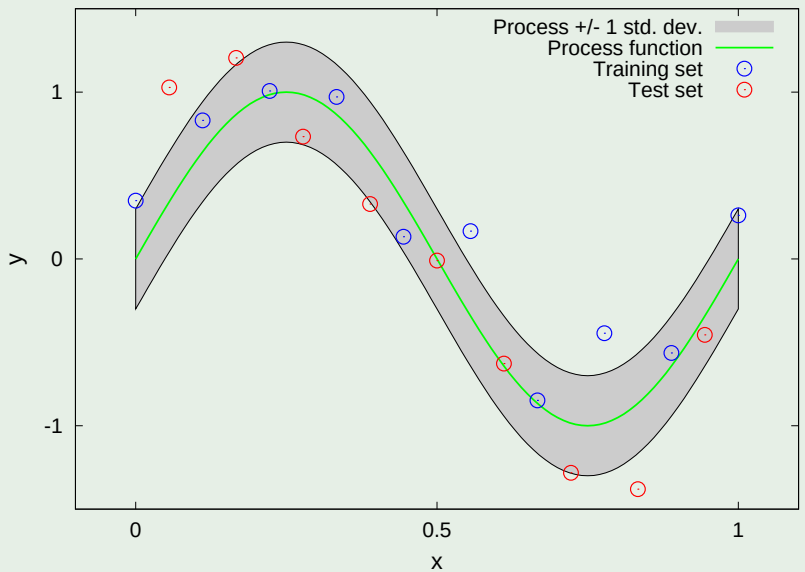
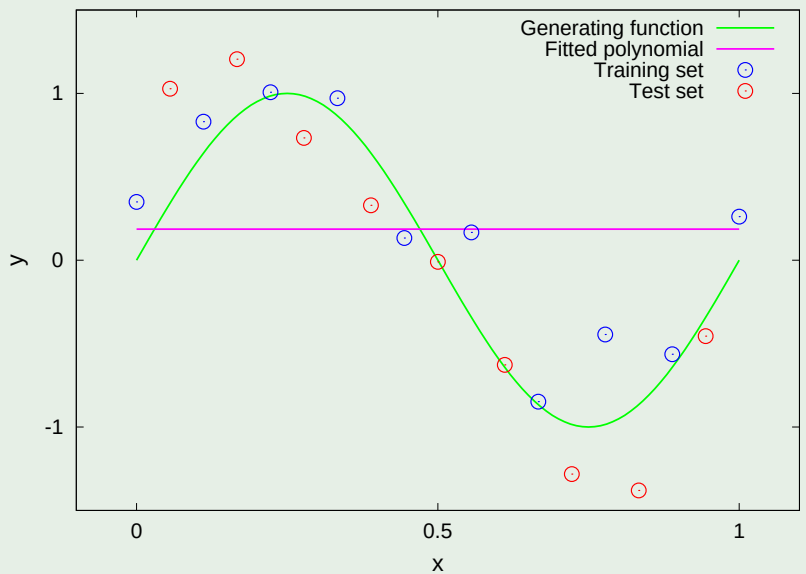
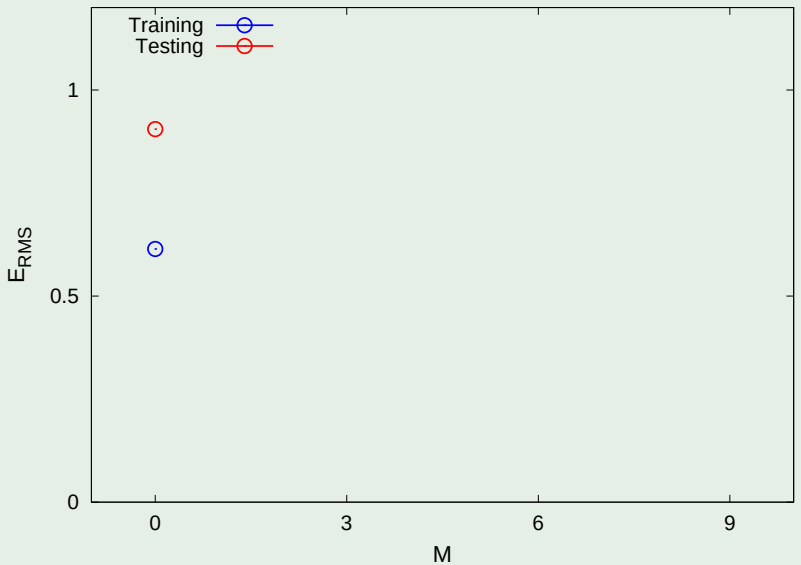


Illustration of overfitting

Example

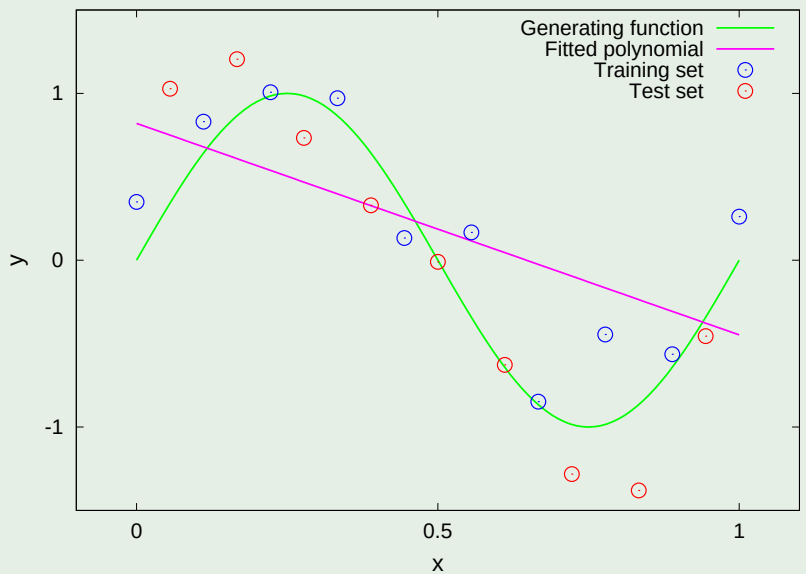


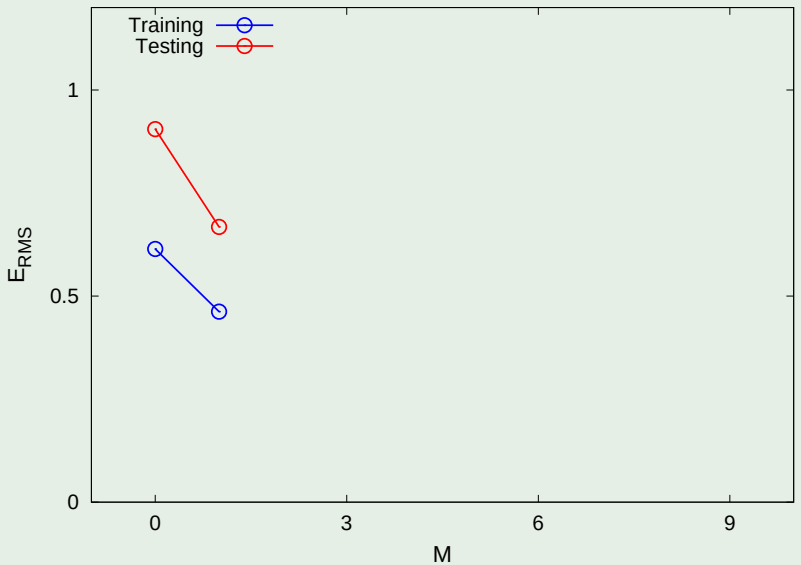


$M = 0$

Illustration of overfitting

Example

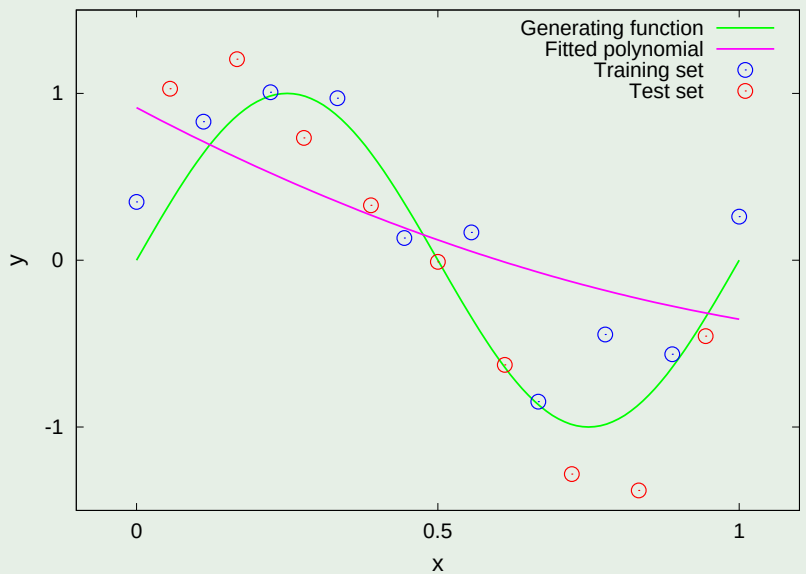


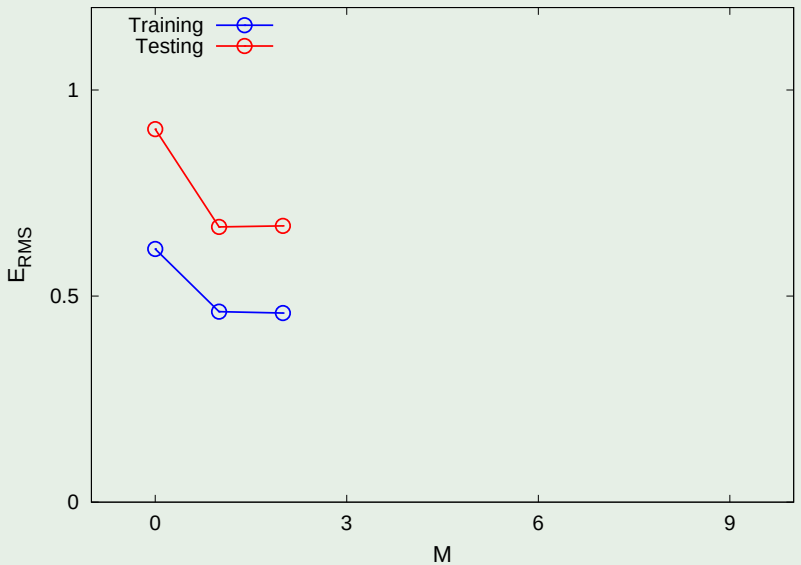


$M = 1$

Illustration of overfitting

Example

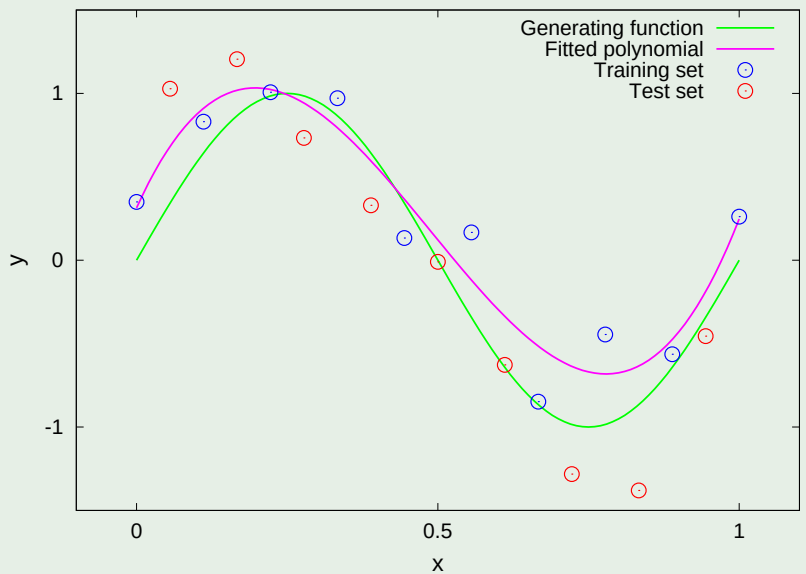


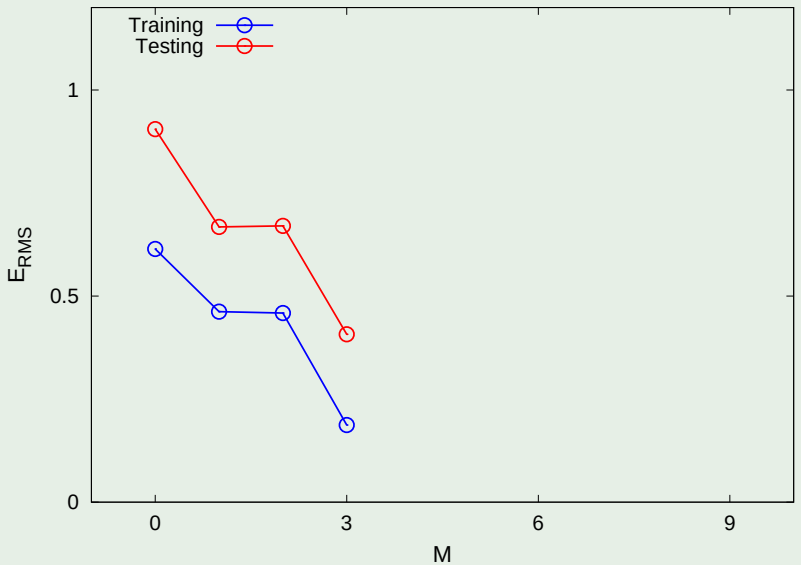


$M = 2$

Illustration of overfitting

Example

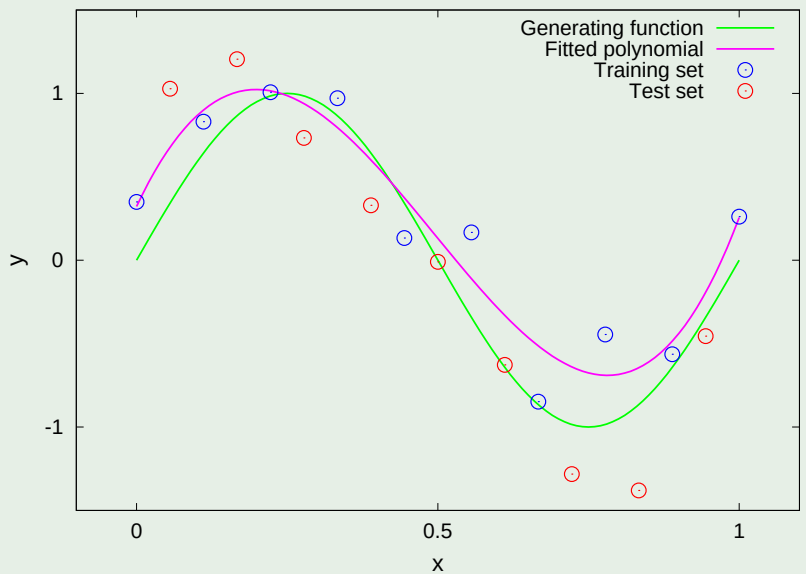


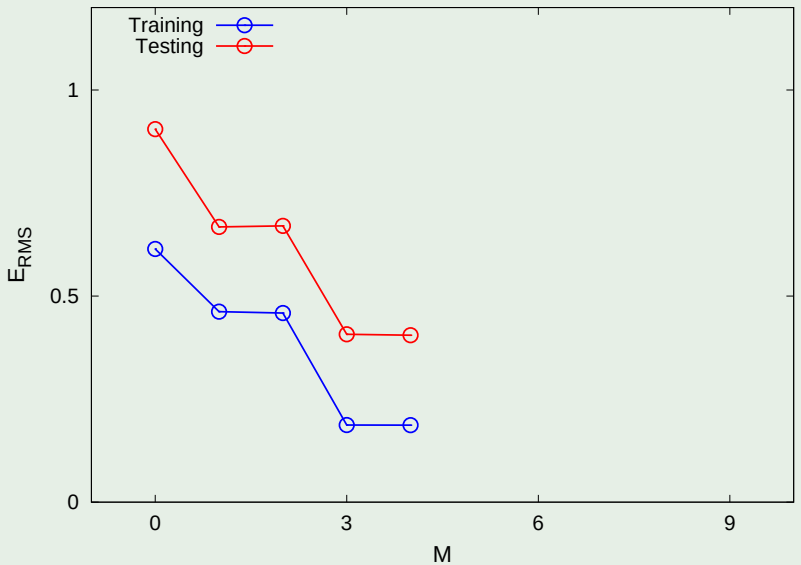


$M = 3$

Illustration of overfitting

Example

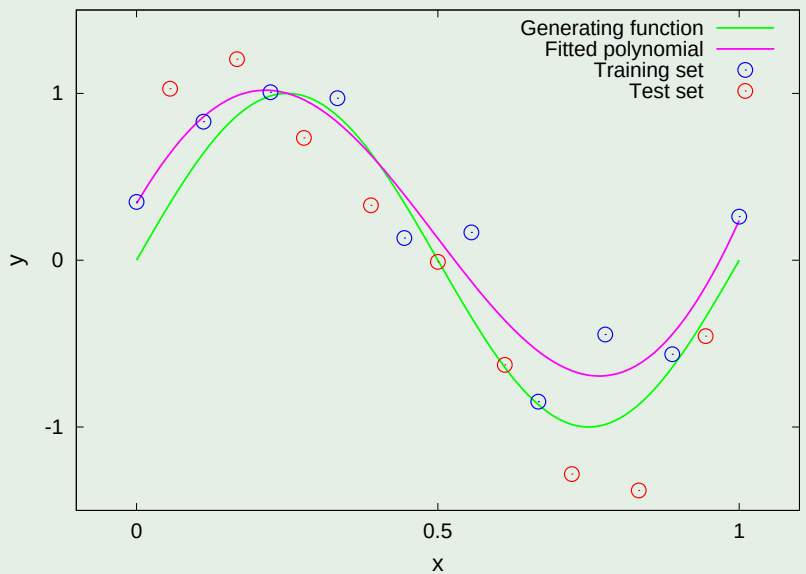


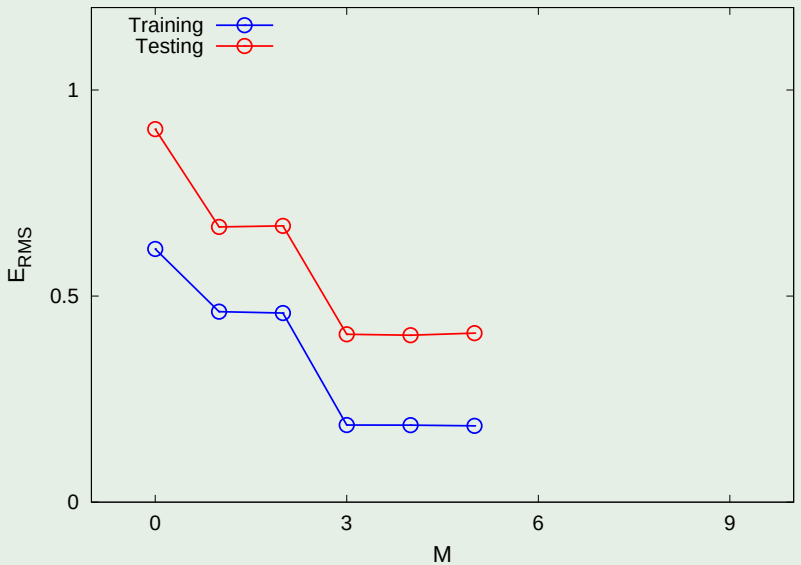


$M = 4$

Illustration of overfitting

Example

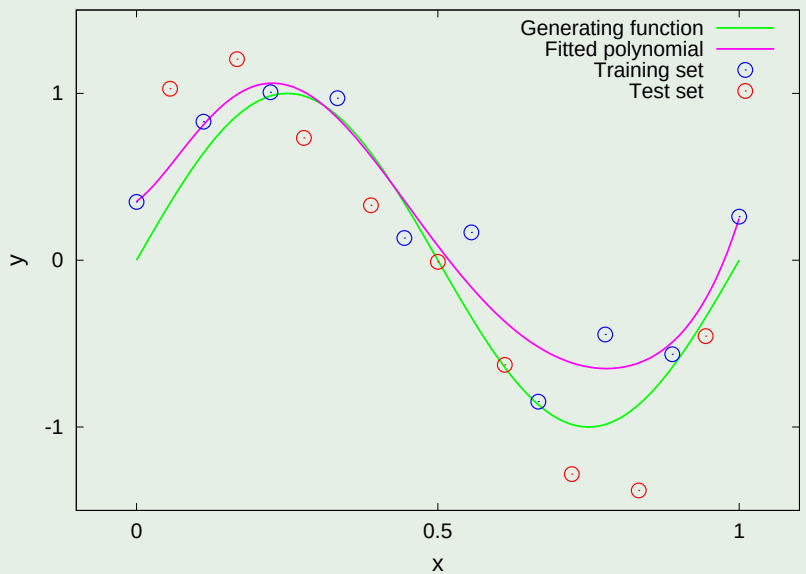


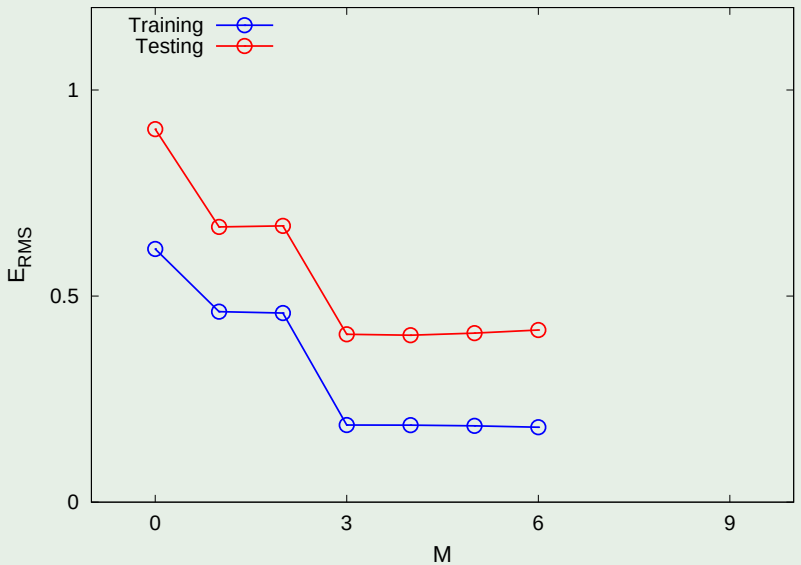


$M = 5$

Illustration of overfitting

Example

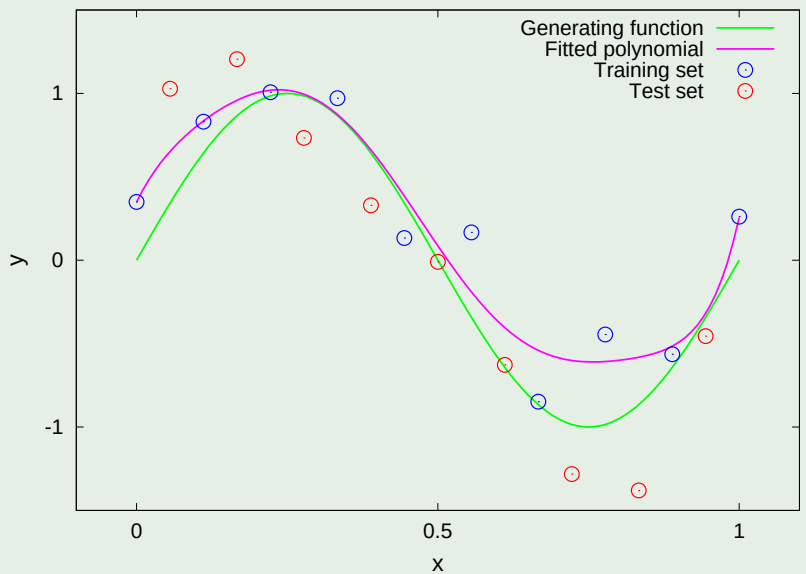


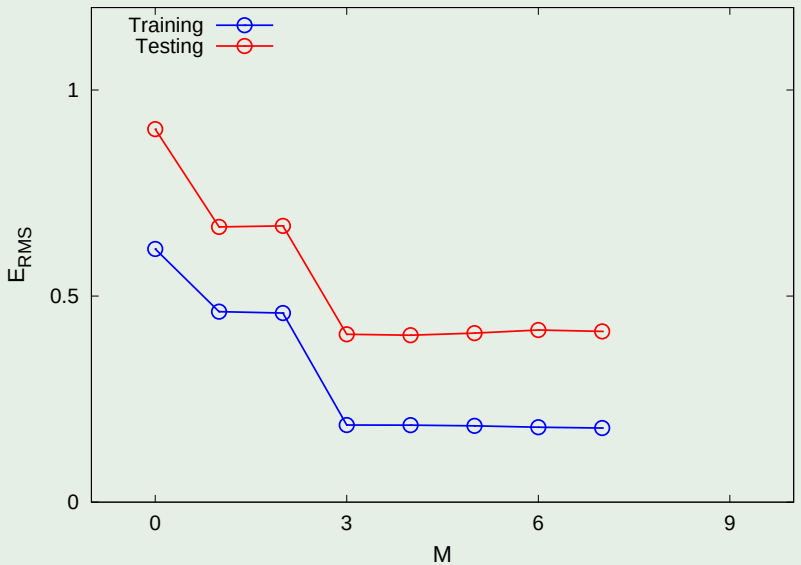


$M = 6$

Illustration of overfitting

Example

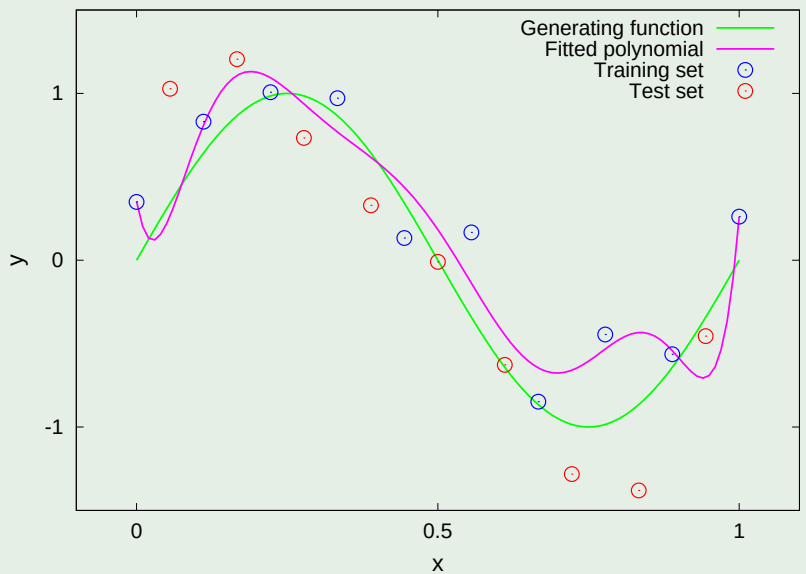


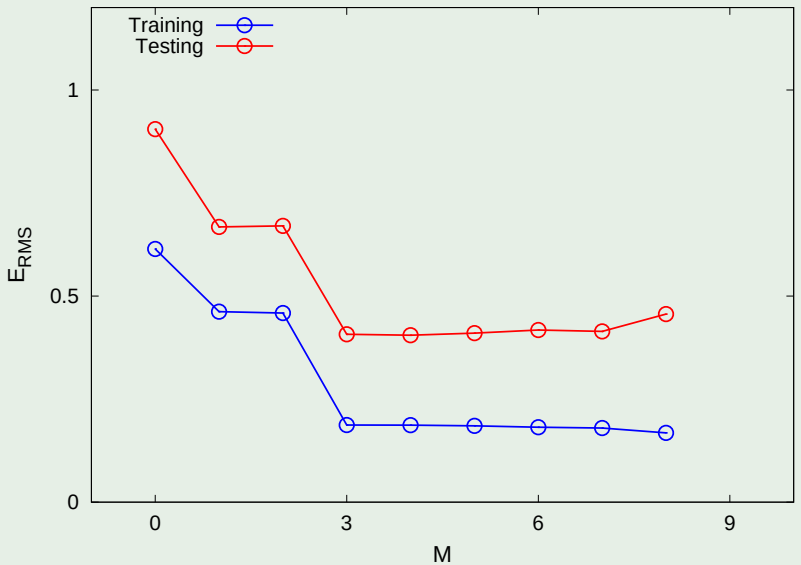


$M = 7$

Illustration of overfitting

Example

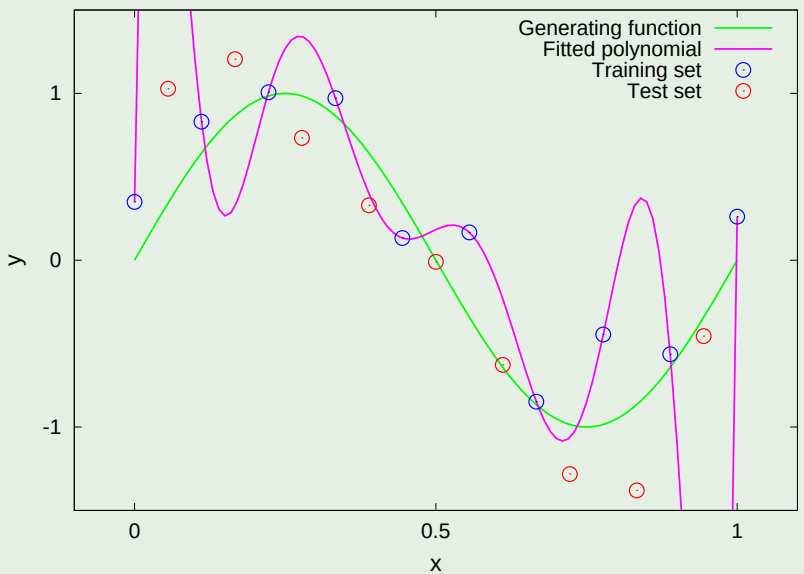


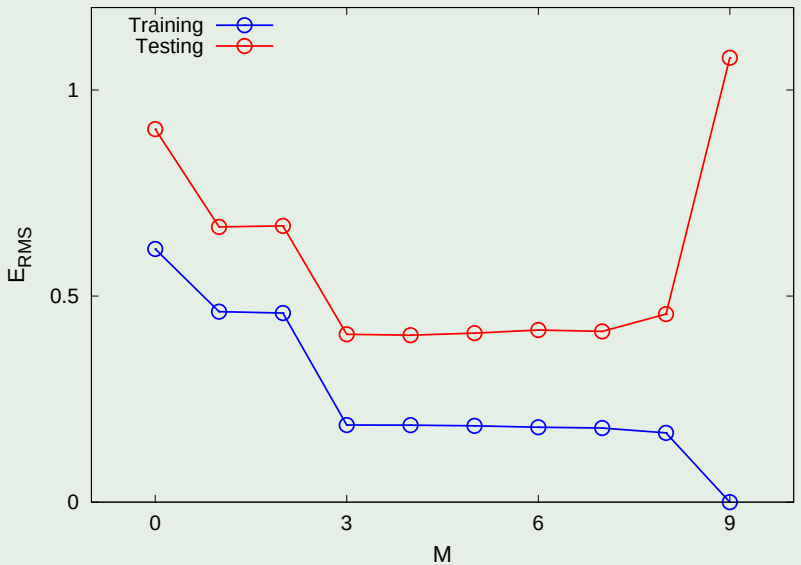


$M = 8$

Illustration of overfitting

Example



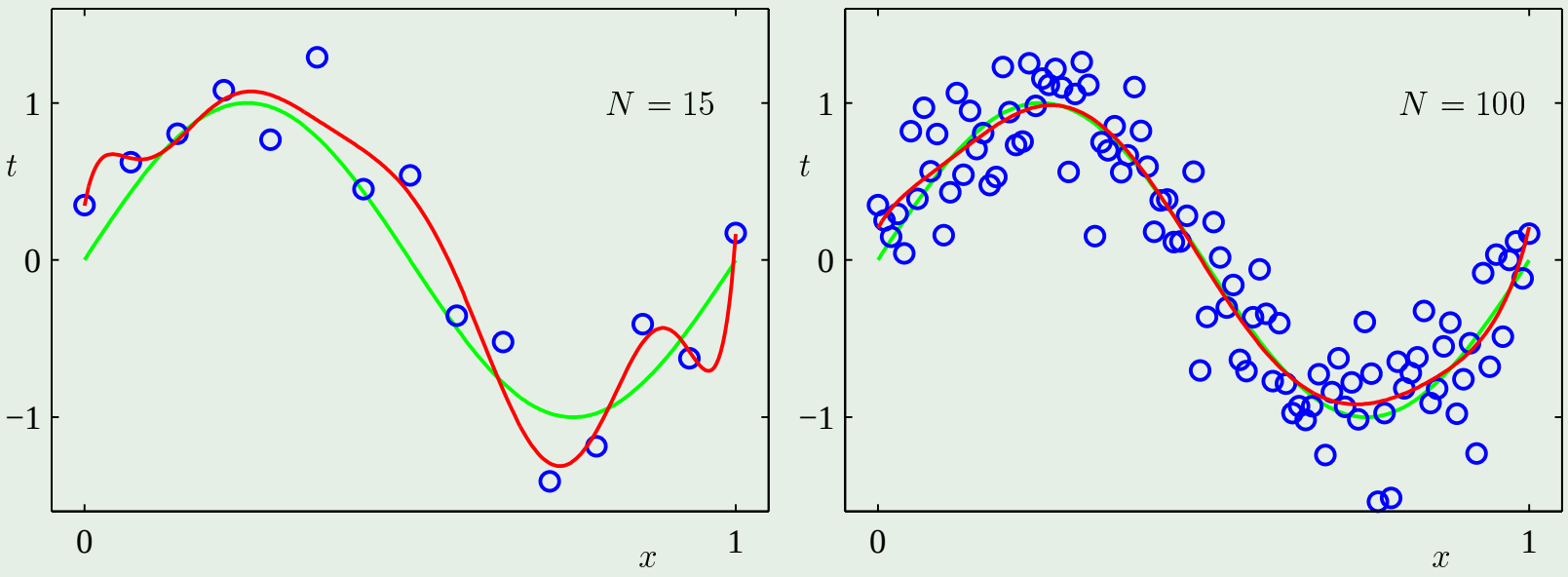


$M = 9$

More data

For fixed model complexity (in this case, $M = 9$), increasing the amount of training data reduces overfitting

Example



Weights and overfitting

	$M=0$	$M=1$	$M=3$	$M=6$	$M=9$
w_0	0.19	0.82	0.31	0.35	0.35
w_1		-1.27	7.99	2.62	232.37
w_2			-25.43	32.10	-5321.79
w_3			17.37	-206.27	48568.00
w_4				399.00	-231637.92
w_5				-332.71	640038.66
w_6				105.16	-1061794.80
w_7					1042394.73
w_8					-557680.13
w_9					125200.80

Parameter shrinkage

- Penalise overly flexible models
- Add a penalty term to the objective function
- Penalty term depends on model
  - Typically, penalise large parameter values

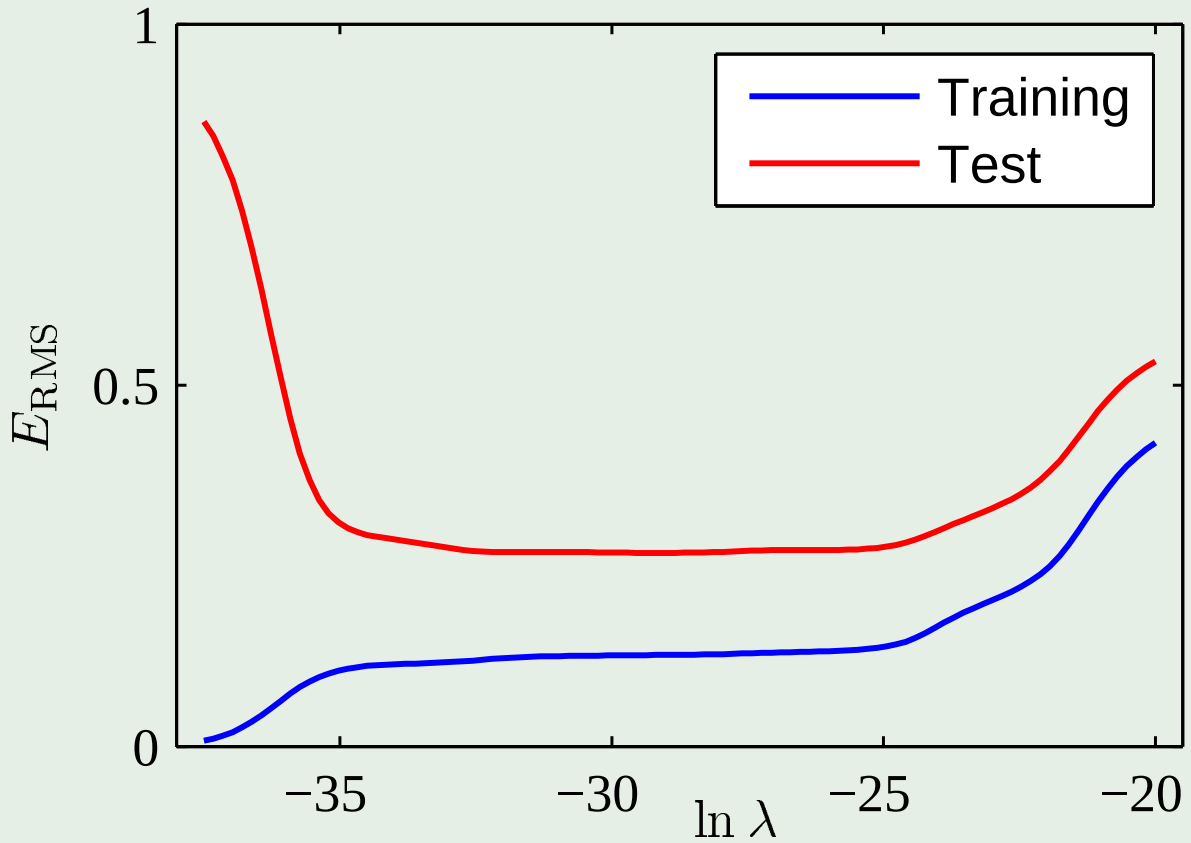
Example: Polynomial curve fitting

$$\hat{E}(\mathbf{w}) = \underbrace{\frac{1}{2}\sum_{n=1}^{N}(y(x_n, \mathbf{w}) - t_n)^2}_{\text{Objective function}} + \underbrace{\frac{\lambda}{2}||\mathbf{w}||^2}_{\text{Penalty}}$$

- Parameter λ controls regularisation
  - "How much do you trust the data"
  - Must be set independently

Parameter shrinkage

Example: Polynomial curve fitting

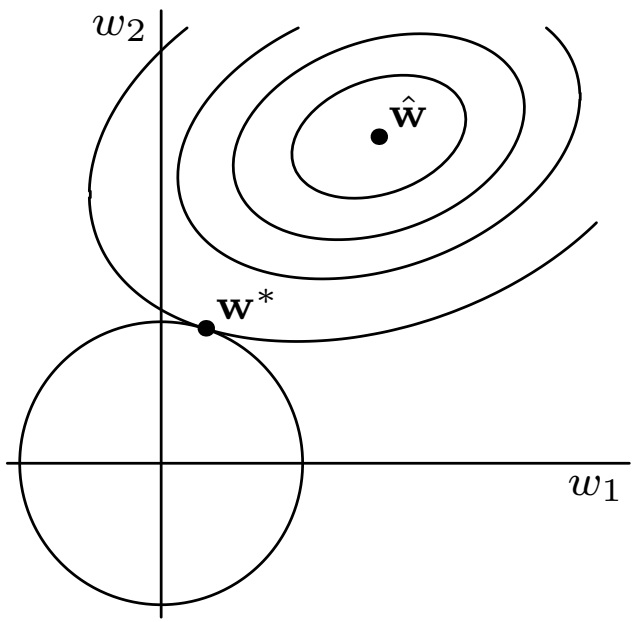


Notes about parameter shrinkage

1. **Leave w_0 out of the penalty term.**
   Shifting the data should not affect the model's performance
2. **Square penalty $\mathbf{w}^\top \mathbf{w}$**
   - Leads to simple optimisation
   - Called ridge regression (stats), weight decay (neural networks)
3. **L_1 norm $\sum_{i=1}^{M} |w_i|$**
   - Cannot be optimised in closed form
   - *Sparse* solutions (some $w_i = 0$)
   - LASSO: least absolute shrinkage and selection operator
4. **L_q norm $\sum_{i=1}^{M} |w_i|^q$**

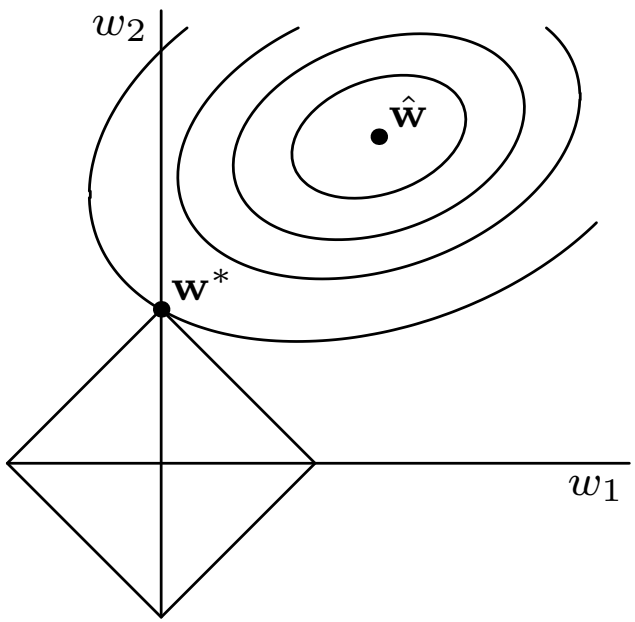
Notes about parameter shrinkage

1. **Leave w_0 out of the penalty term.**
   Shifting the data should not affect the model's performance
2. **Square penalty $\mathbf{w}^\top \mathbf{w}$**
   - Leads to simple optimisation
   - Called ridge regression (stats), weight decay (neural networks)
3. **L_1 norm $\sum_{i=1}^{M} |w_i|$**
   - Cannot be optimised in closed form
   - *Sparse* solutions (some $w_i = 0$)
   - LASSO: least absolute shrinkage and selection operator
4. **L_q norm $\sum_{i=1}^{M} |w_i|^q$**



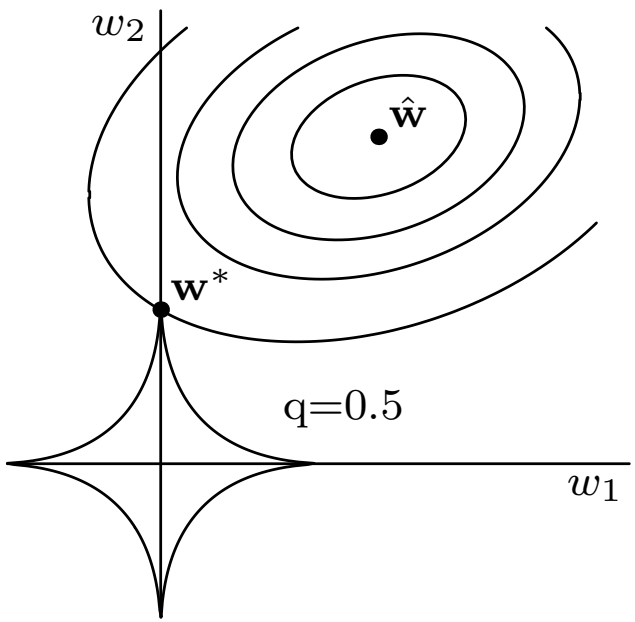
Notes about parameter shrinkage

1. **Leave w_0 out of the penalty term.** Shifting the data should not affect the model's performance
2. **Square penalty $\mathbf{w}^\top \mathbf{w}$**
   - Leads to simple optimisation
   - Called ridge regression (stats), weight decay (neural networks)
3. **L_1 norm** $\sum_{i=1}^{M} |w_i|$
   - Cannot be optimised in closed form
   - *Sparse* solutions (some $w_i = 0$)
   - LASSO: least absolute shrinkage and selection operator
4. **L_q norm** $\sum_{i=1}^{M} |w_i|^q$



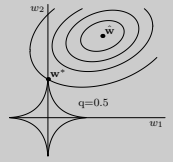
Notes about parameter shrinkage

1. **Leave w_0 out of the penalty term.** Shifting the data should not affect the model's performance
2. **Square penalty $\mathbf{w}^\top \mathbf{w}$**
   - Leads to simple optimisation
   - Called ridge regression (stats), weight decay (neural networks)
3. **L_1 norm** $\sum_{i=1}^{M} |w_i|$
   - Cannot be optimised in closed form
   - *Sparse* solutions (some $w_i = 0$)
   - LASSO: least absolute shrinkage and selection operator
4. **L_q norm** $\sum_{i=1}^{M} |w_i|^q$



Notes about parameter shrinkage

- **Leave w_0 out of the penalty term.** Shifting the data should not affect the model's performance
- **Square penalty $\mathbf{w}^\top\mathbf{w}$**
  - Leads to simple optimisation
  - Called ridge regression (stats), weight decay (neural networks)
- **L_1 norm** $\sum_{i=1}^{M} |w_i|$
  - Cannot be optimised in closed form
  - *Sparse* solutions (some $w_i = 0$)
  - LASSO: least absolute shrinkage and selection operator
- **L_q norm** $\sum_{i=1}^{M} |w_i|^q$



1. Called ridge regression, weight decay, other names?
2. Mention the elastic-net penalty $\lambda \sum \alpha\beta_i^2 + (1-\alpha)|\beta_i|$

Maximum A Posteriori (MAP) learning

Using Bayes' rule, we have:

$$p(\boldsymbol{\theta}|\mathbf{X}) = \frac{p(\mathbf{X}|\boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\mathbf{X})} \tag{4}$$

This requires us to place a prior over the parameter values

- Any prior is possible, choose prior to reflect prior knowledge
- If we use a Gaussian distribution with zero mean, this is equivalent to ML learning with parameter shrinkage
- The denominator $p(\mathbf{X}) = \int p(\mathbf{X}|\boldsymbol{\theta})p(\boldsymbol{\theta})\mathrm{d}\boldsymbol{\theta}$ is often intractable to compute but is constant, so that

$$\arg\max_{\boldsymbol{\theta}} \frac{p(\mathbf{X}|\boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\mathbf{X})} = \arg\max_{\boldsymbol{\theta}} p(\mathbf{X}|\boldsymbol{\theta})p(\boldsymbol{\theta}) \tag{5}$$

- As $|\mathbf{X}| \to \infty$, influence prior vanishes

Laplace smoothing

Conjugate prior:

- functional form of prior leads to posterior with same functional form
- allows on-line, iterative learning: posterior becomes prior for next datapoint
- prior can be interpreted as having seen a number of examples

Laplace smoothing

- Maximum likelihood learning
- Pretend n examples of each possible outcome have been observed before starting

Regularisation: Laplace smoothing

Example

I have a coin and want to evaluate $p(\text{head}) = \mu^{\text{head}}(1-\mu)^{\text{tail}}$. If I see a single observation, a head. What is the ML estimate of μ

$$\mu = \frac{1}{1+0} = 1 \tag{6}$$

Laplace smoothing: assume you have already seen a number of examples of every possible outcome before you start:

$$\mu = \frac{2}{2+1} = \frac{2}{3} \tag{7}$$

As the amount of observed data grows large, the influence of the smoothing vanishes.

Regularisation: Laplace smoothing

Example

I have a coin and want to evaluate $p(\text{head}) = \mu^{\text{head}}(1-\mu)^{\text{tail}}$. If I see a single observation, a head. What is the ML estimate of μ

$$\mu = \frac{1}{1+0} = 1 \tag{6}$$

Laplace smoothing: assume you have already seen a number of examples of every possible outcome before you start:

$$\mu = \frac{2}{2+1} = \frac{2}{3} \tag{7}$$

As the amount of observed data grows large, the influence of the smoothing vanishes.

The Bayesian approach

In fact, we're not really interested in knowing the original distribution that "generated" the data

- We'll never know that anyway

What we really want to do, is to use the knowledge that we have in an optimal way. That is, we want

$$p(t|\mathbf{x}, \mathbf{X}, \mathbf{t}) = \int p(t|\mathbf{x}, \theta) p(\theta|\mathbf{X}, \mathbf{t}) d\theta \tag{8}$$

In effect, we consider all the models (of the form that we have chosen beforehand) that could have generated the data, and weight them their prediction according to how probable they are.

How do we avoid overfitting?

More training data As more training data gets available, more complex models become warranted

Use a simple model The simpler the model, the more likely it will generalise

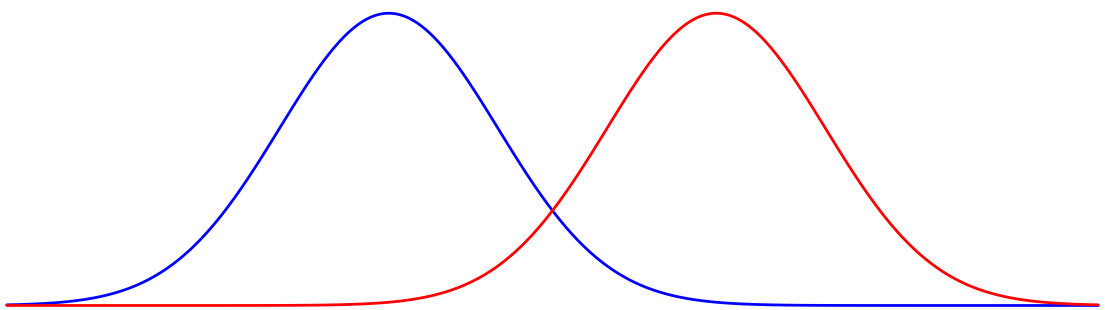
- Occam's razor
- By making the model too inflexible to fit the noise, we force it to "focus" on the process
- Inductive bias — Learn the *structure* of the data, not the data itself

Penalise complexity Penalise model parameters that make the model complex

Bayesian inference Do not use the best/most likely model: consider all possible models and weigh them according to their likelihood (See lecture 4)

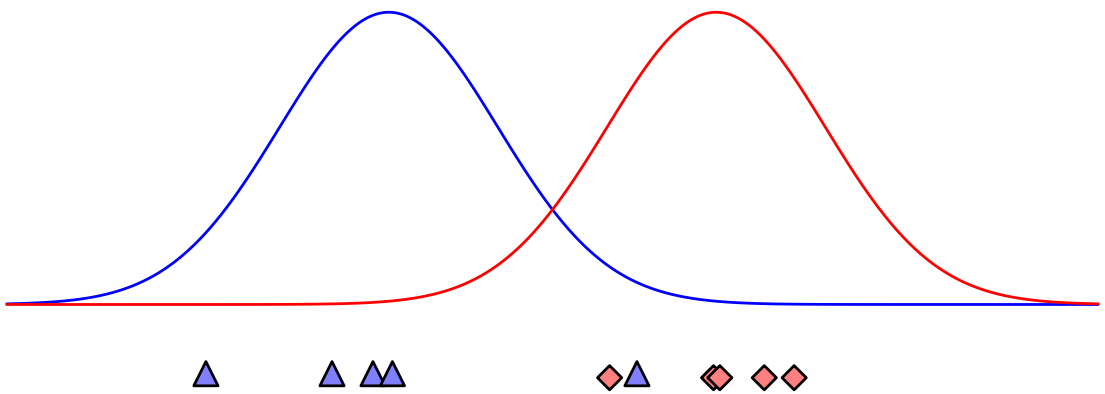
Measures of discriminability

- Apparent error rate E_A
  - Correct classification on training set
  - Too optimistic due to overfitting
  - Depends on: classifier,training set
- True error rate E_T
- Bayes error rate E_B



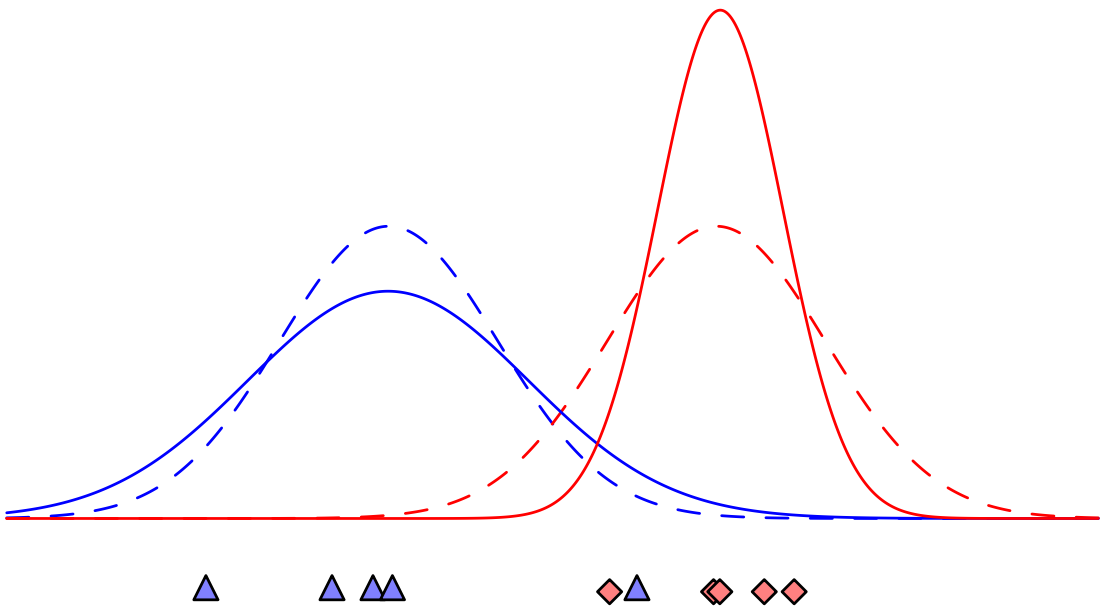
Measures of discriminability

- Apparent error rate E_A
  - Correct classification on training set
  - Too optimistic due to overfitting
  - Depends on: classifier,training set
- True error rate E_T
- Bayes error rate E_B



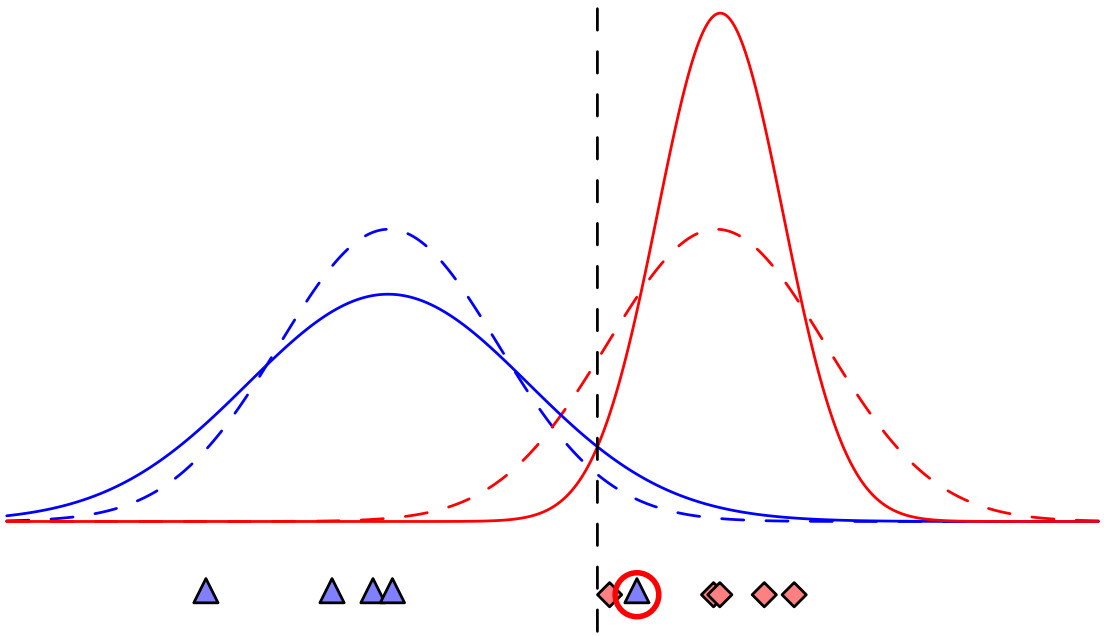
Measures of discriminability

- Apparent error rate E_A
  - Correct classification on training set
  - Too optimistic due to overfitting
  - Depends on: classifier,training set
- True error rate E_T
- Bayes error rate E_B



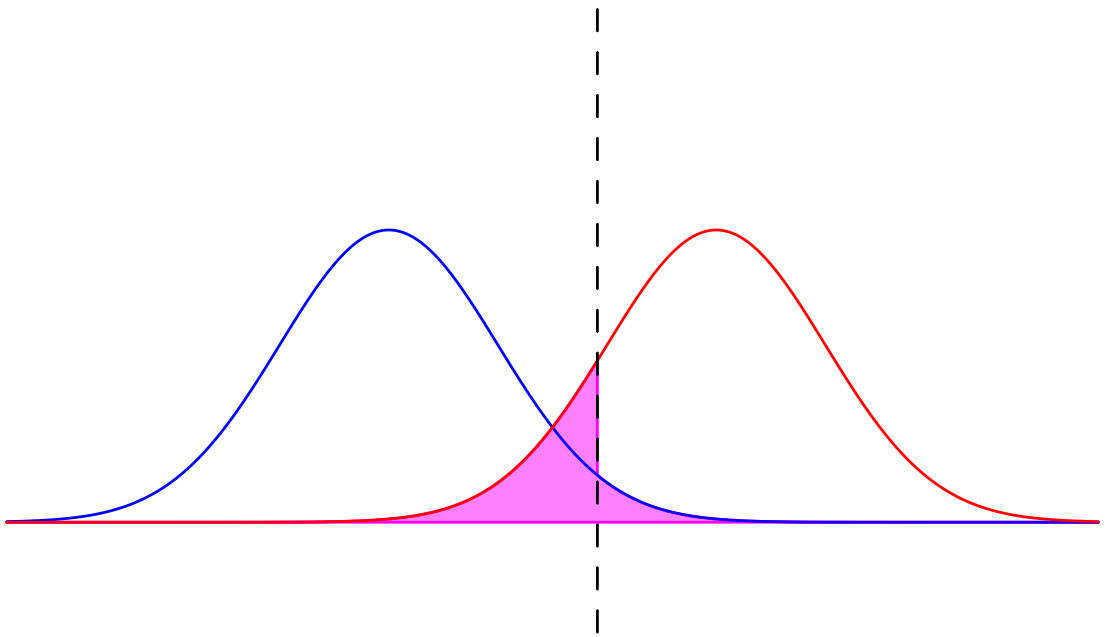
Measures of discriminability

- Apparent error rate E_A
  - Correct classification on training set
  - Too optimistic due to overfitting
  - Depends on: classifier,training set
- True error rate E_T
- Bayes error rate E_B



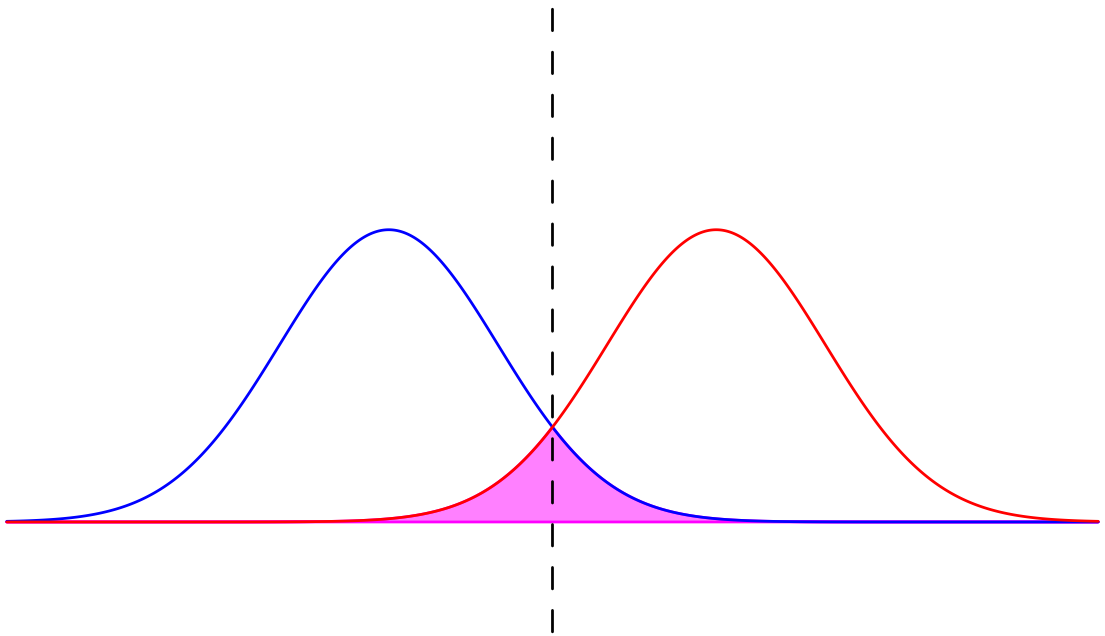
Measures of discriminability

- Apparent error rate E_A
- True error rate E_T
  - Classification error on infinite test set
  - Expected probability misclassifying randomly chosen pattern
  - Depends on: classifier, data distribution
- Bayes error rate E_B



Measures of discriminability

- Apparent error rate E_A
- True error rate E_T
- Bayes error rate E_B
  - Optimal error rate
  - Classification error when classifying based on the true class probabilities



Train and Test sets: Avoid overfitting

- Use a training set to train the machine
- Use a separate data set to avoid overfitting
- **However:** This biases the machine towards the separate set
  - Performance on this set is not an unbiased estimate of real-world performance
- **Solution:** Separate the data into three distinct sets

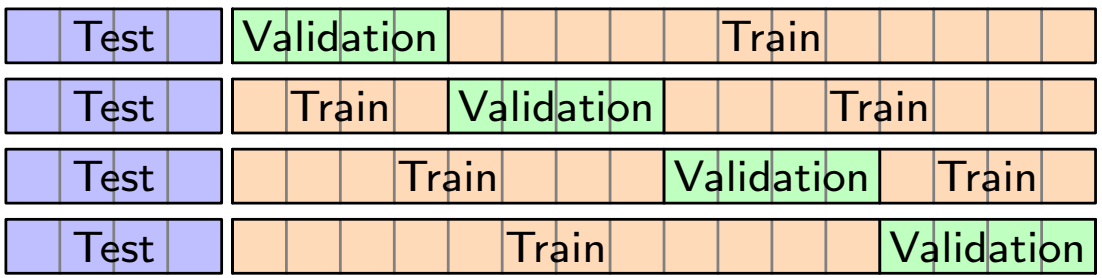
Train	Optimise the objective function
Validation	Model selection
Test	Estimate performance

Cross-Validation

In practice, available training data is often limited

- Splitting the data in sets further reduces this

- Solution: k-fold cross-validation
- Repeatedly split the data and average the results (here, $k = 4$)

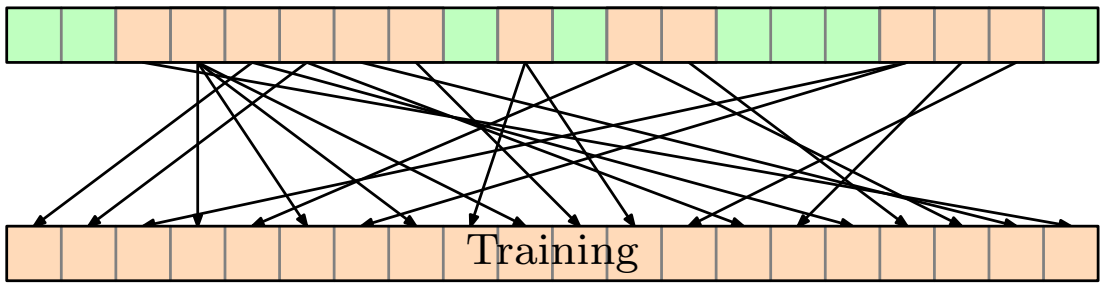


Leave-one-out and Bootstrapping

- In the limit, $k = N$ where N is the size of the dataset
  - "*Leave-one-out*": use a validation set of size 1
  - Maximises the use of the data
  - Minimises correlation between train and test sets
  - Computationally expensive
- Cross-validation is sometimes used for error bars:
  - Evaluate how the error varies on different data sets
  - Incorrect: the data sets are not independent
- Improvement: Bootstrap method

Bootstrapping

Sample N points at random from the data **with replacement**



The probability for not picking a data point is

$$p(\neg b) = (1 - 1/N)^N \approx 0.368 \tag{9}$$

The expected number of used data points is therefore

$$p(b) = 1 - p(\neg b) \approx 0.632N \tag{10}$$

Bootstrap accuracy

Estimate the accuracy as follows:
Repeat L times

- Estimate A_T by training on the sampled data, testing on the remaining data
- Estimate A_A by training and testing on the sampled data
- The "bootstrap accuracy" is then given by

$$\text{acc} = \frac{1}{L}\sum_{i=1}^{L}\left(0.632A_{T,i} + 0.368A_{A,i}\right) \tag{11}$$

The variance on the bootstrap samples is a good (low-variance) estimator of the variance on multiple data sets, but it can have a high bias.

Wrap up

In today's lecture, we have seen:

- A simple classifier, k-Nearest-Neighbours
- How to evaluate how well a machine performs
- The inherent difficulty of training from a given set of examples, overfitting
- How to avoid overfitting
- How to optimise the use of the data we have

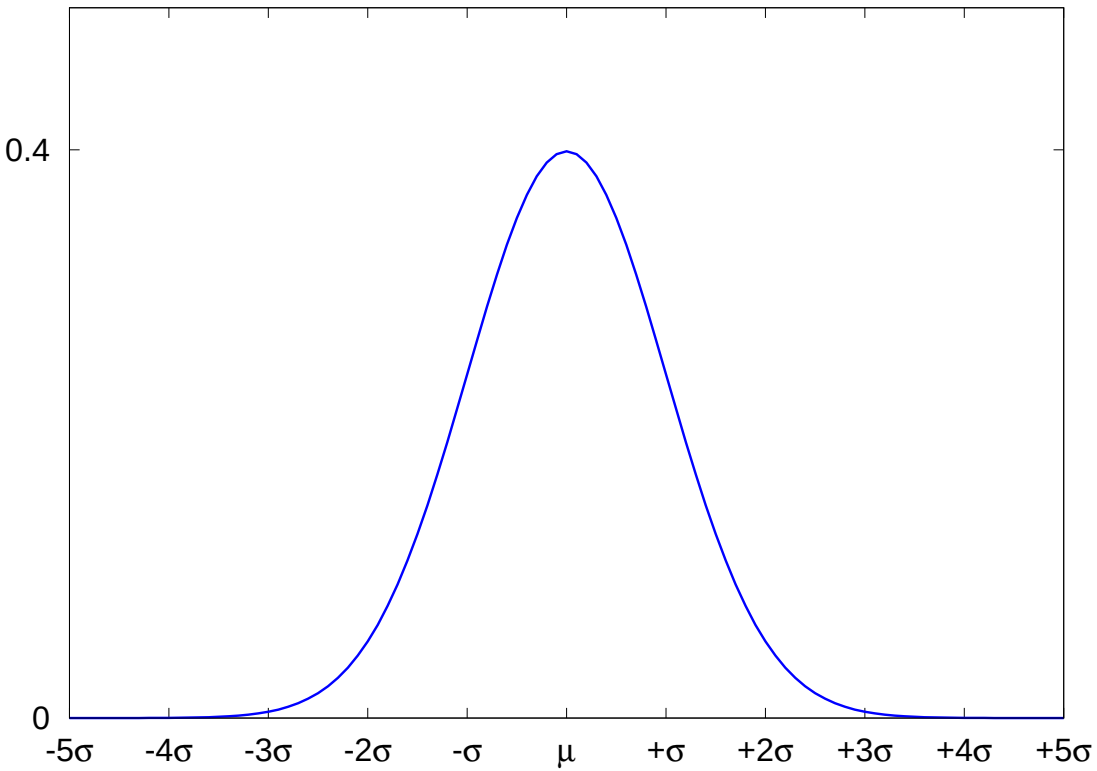
Coming up:

- **Exercise:** Derive ML and MAP parameter estimators for Gaussian distribution
- **Lab:** Introduction to Matlab
- **Next week:** we have a look at discriminant-based classifiers

6 Distributions

- Univariate distributions

The Gaussian (Normal) Distribution



back

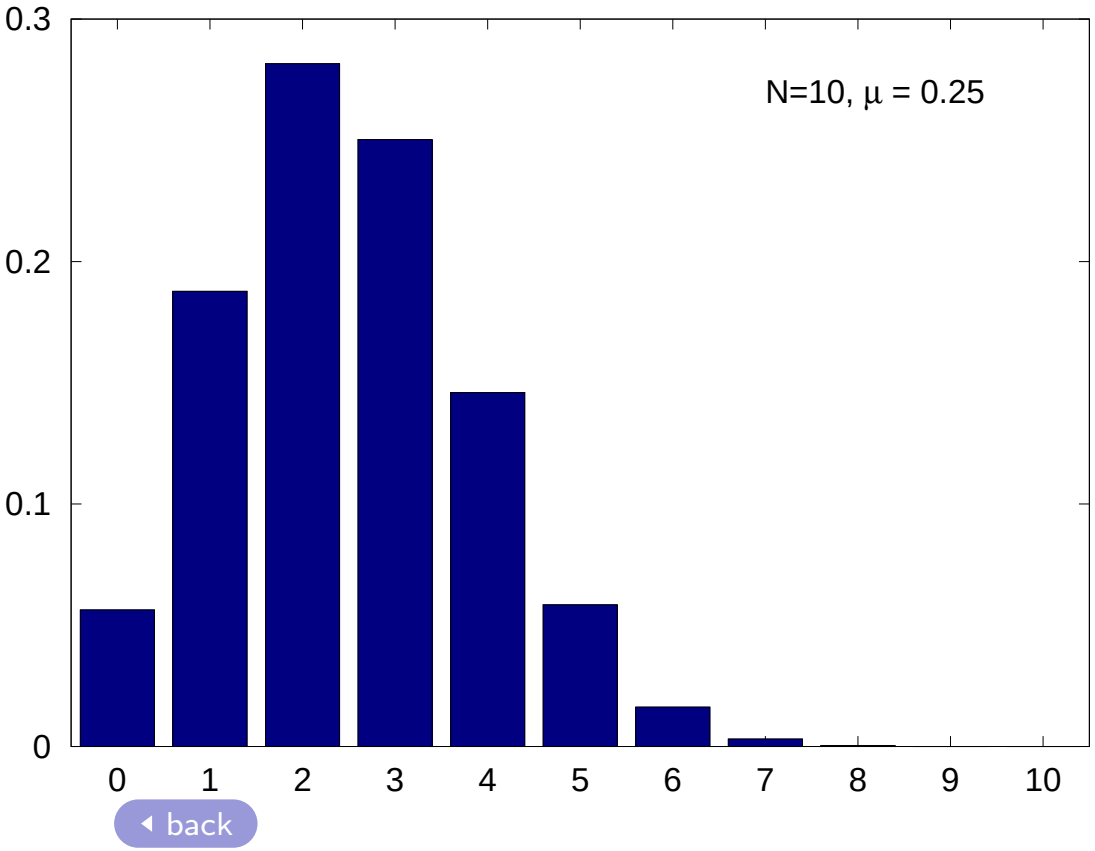
Notation

- $x \sim \mathcal{N}(\mu, \sigma)$

Properties

- $p(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp -\frac{(x-\mu)^2}{2\sigma^2}$
- $\mathbb{E}[x] = \mu$
- $\text{var}[x] = \sigma^2$

The Binomial Distribution



Notation

- $x \sim \text{Bin}(N, \mu)$

Properties

- $p(x) = \binom{N}{x} \mu^x (1-\mu)^{N-x}$

  where

$$\binom{N}{x} = \frac{x!}{x!(N-x!)}$$

- $\mathbb{E}[x] = N\mu$
- $\text{var}[x] = N\mu(1-\mu)$

The Bernoulli distribution

Notation

- $x \sim \text{Bern}(\mu)$

Properties

- $p(x) = \mu^x (1-\mu)^{1-x}$
- $\mathbb{E}[x] = \mu$
- $\text{var}[x] = \mu(1-\mu)$

[Image shamelessly copied from: http://governing.typepad.com]