Pattern recognition

Francesco Corona

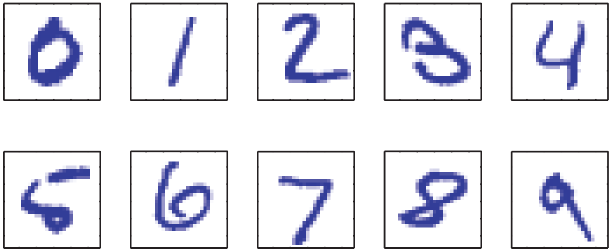
Outline

Pattern recognition
and also machine learning

Pattern recognition and machine learning

Pattern recognition is concerned with automatic discovery of *regularities* in data through computer algorithms and the use of such regularities to take actions

- Typical example is the classification of the data into different categories



A machine capable of recognising handwritten digits from images (28×28 pixels) represented as input vectors ($\mathbf{x}$, comprising $28 \times 28 = 784$ numbers)

- Produce the identity of the digit $(0, \ldots, 9)$
- Not a trivial task (variety of handwriting)

Pattern recognition and machine learning (cont.)

Learning regularities in data and searching for patterns is a central problem

- *Machine intelligence* has a long and somehow successful history

The *first generation* of machine learning is the good old *artificial intelligence*

- Expert systems using knowledge extracted from humans (rules)
- Combinatorial explosion (lots of rules and exceptions to rules)
- 70's and 80's

Pattern recognition and machine learning (cont.)

The *second generation* is basically *neural networks* and *support vector machines*

- Black-box statistical models focused on learning from data
- Prior knowledge is rather general assumptions (continuity)
- Hard to incorporate specific (complex) domain knowledge
- 80's and 90's

The *third generation* is integration of *domain knowledge* and *statistical learning*

- Probabilistic modelling and Bayesian framework
- Probabilistic graphical models and inference
- 00's

This course is located somewhere **between the second and third generation**

This course

The course in a nutshell - The core

- **Linear models for regression**: Linear basis function models, Bias-variance decomposition, Bayesian linear regression, (Bayesian model comparison, Evidence approximation)
- **Linear models for classification**: Discriminant functions, Probabilistic generative models, Probabilistic discriminative models, (Laplace approximation, Bayesian logistic regression)
- **Neural networks**: Feed-forward network functions, Network training, Error back-propagation (the Hessian matrix), (Regularisation, Mixture density models, Bayesian neural networks)
- **Kernel methods**: Dual representations, Building kernels, Radial basis function networks, Gaussian processes, Maximum margin classifiers, (Relevance vector machines)

The course in a nutshell - The maybes

- **Graphical models**: Bayesian networks, Conditional independence, (Random Markov fields, Inference)
- **Mixture models and Expectation maximisation**: K-means clustering, Mixtures of Gaussians, (EM)

The course in a nutshell - Can't do without

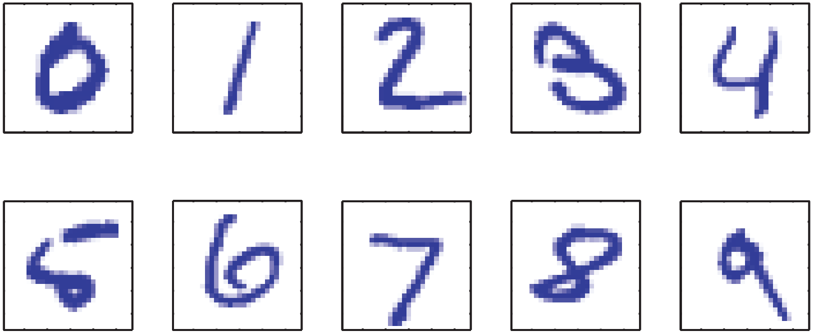
- **Probability theory**: Generalities, Probability densities, Expectations and covariances, Bayesian probabilities, the Gaussian distribution, Curve fitting
- **Decision theory**: Generalities, Minimising the misclassification rate, Minimising the expected loss, Reject option, Inference and decision, Loss functions for regression
- **Information theory**: Generalities, Relative entropy and mutual information

- **Probability distributions**: Binary variables and Multinomial distributions, the Gaussian distribution, (the Exponential family), Nonparametric density estimation

- **On modelling**: Model selection and the curse of dimensionality

Terminology

Terminology



We have a large set of N **input vectors** $\{\mathbf{x}_i\}_{i=1}^{N}$ (say, $\mathbf{x}_i$ is the 784-vector whose elements comprise the pixel values of an unrolled 28×28 image)

We have also the corresponding output or **target vectors** $\{\mathbf{t}_i\}_{i=1}^{N}$ (say, $\mathbf{t}_i$ is the binary 10-vector whose elements represent the identity of the digit)

Terminology (cont.)

The sets $\{\mathbf{x}_i\}$ and $\{\mathbf{t}_i\}$ together are the **training set**, it is used to tune a model

- Tuning a model means determining optimal values for its parameters

The result of running a machine learning model is expressed as a function $\mathbf{y}(\mathbf{x})$

- it takes a new input vector $\mathbf{x}$ (say, a new image), it processes it
- it generates an output vector $\mathbf{y}$, encoded as the target vector

The precise form of $\mathbf{y}(\mathbf{x})$ is determined during the **learning** or **training phase**

Once the model is trained, it can determine the identity of new input vectors

- the **test set**

The ability to categorise correctly test examples is known as **generalisation**

Terminology (cont.)

Applications in which the training data comprises examples of the input vectors along with their corresponding target vectors are **supervised learning** problems

- **Classification**: The aim is to assign each input vector to one of a finite number of discrete categories
- **Regression**: The aim is to assign each input vector to one or more continuous variables

Terminology (cont.)

Applications in which he training data consists of a set of input vectors **x** with out any corresponding target values **t** are **unsupervised learning** problems

- **Clustering**: Discover groups of similar examples within the data
- **Density estimation**: Determine the distribution of data in the input space
- **Visualisation**: Project the data from a high-dimensional space to 2 or $3D$

Terminology (cont.)

Vectors are denoted by lower case bold letters $\mathbf{x}$

- All vectors are assumed to be column vectors
- Superscript T denotes transposition
- $\mathbf{x}^T$ is a row vector
- $(x_1, \dots, x_D)$ is a row vector with D elements
- $\mathbf{x} = (x_1, \dots, x_D)^T$ is again a column vector

Matrices are denoted by upper case bold letters $\mathbf{M}$

- The $M \times M$ identity matrix is $\mathbf{I}_M$, or simply $\mathbf{I}$
- Elements I_{ij} of $\mathbf{I}$ equal 1 if $i = j$, and 0 if $i \neq j$

Terminology (cont.)

A functional is denoted $f[y]$ where $y(x)$ is some function of some variable x

The expectation of function $f(x, y)$ wrt to a random variable x is $\mathbb{E}_x[f(x, y)]$

- If the distribution of x is conditioned on another variable z the corresponding conditional expectation is denoted $\mathbb{E}_x[f(x, y)|z]$

The variance is denoted $\text{var}[f(x)]$, for vector variables the covariance is $\text{cov}[\mathbf{x}, \mathbf{y}]$

Terminology (cont.)

N values $\mathbf{x}_1, \ldots, \mathbf{x}_N$ of a D-dim vector $\mathbf{x} = (x_1, \ldots, x_D)^T$ can be combined

- Into a matrix $\mathbf{M}$, whose n-th row is the row vector $\mathbf{x}_n^T$
- The (n, j) element of $\mathbf{M}$ is the i-th element of the n-th value $\mathbf{x}_n$

1-D variables a matrix are denoted by typewriter letters $\mathtt{x}$: $\mathtt{x}$ is a column vector whose n-th element is x_n, it has dimensionality N ($\mathbf{x}$ has dimensionality D)

An example

Polynomial curve fitting

We consider a simple regression problem to motive a number of key concepts

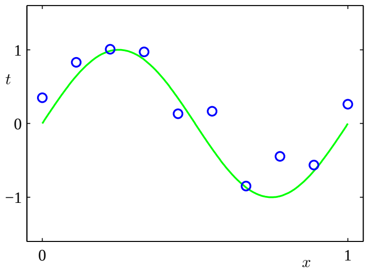
Suppose we observe a real-valued input variable x, and we wish to use this observation to predict the value of a real-valued target variable t

- The data for this example is generated from the function $\sin(2\pi x)$ with *random noise* included in the target values

Now suppose that we are given a training set comprising

- N observations of x, $\mathbf{x} \equiv (x_1, \dots, x_N)^T$
- N observations of t, $\mathbf{t} \equiv (t_1, \dots, t_N)^T$

Polynomial curve fitting (cont.)



Training data of $N = 10$ points, blue circles

- each comprising an observation of the **input variable** x along with the corresponding **target variable** t

The **unknown function** $\sin(2\pi x)$ is used to generate the data, green curve

- Goal: Predict the value of t for some new value of x
- without knowledge of the green curve

The **input training data** $\mathtt{x}$ was generated by choosing values of x_n, for $n = 1, \ldots, N$, that are spaced uniformly in the range $[0, 1]$

The **target training data** $\mathtt{t}$ was obtained by computing values $\sin(2\pi x_n)$ of the function and adding a small level of Gaussian noise

Polynomial curve fitting (cont.)

Goal: Exploit the training set to make predictions of the value t of the target variable, for some new value x of the input variable

- Implicitly, try to discover the underlying function $\sin(2\pi x)$

This is intrinsically a difficult problem:

- We have to generalise from a finite data set
- The observed data are corrupted with noise

For a given x there is uncertainty as to the appropriate value for t

- **Probability theory** provides a framework for expressing such uncertainty in a precise and quantitative manner
- **Decision theory** allows us to exploit this probabilistic representation in order to make predictions that are optimal, according to an appropriate criteria

Polynomial curve fitting (cont.)

For the moment, however, we shall proceed informally

We consider a simple approach based on curve fitting

- We shall fit the data using a polynomial function of the form

$$y(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \cdots + w_M x^M = \sum_{j=0}^{M} w_j x^j \qquad (1)$$

- M is the order of the polynomial, x^j denotes x raised to the power of j
- The polynomial coefficients $w_0, \ldots, w_M$ are collected in the vector $\mathbf{w}$

The polynomial function $y(x, \mathbf{w})$

- Nonlinear function of the inputs x
- Linear function of the coefficients $\mathbf{w}$

Functions that are linear in the unknown parameters are called **linear models**

Polynomial curve fitting (cont.)

The coefficients values are obtained by fitting the polynomial to training data

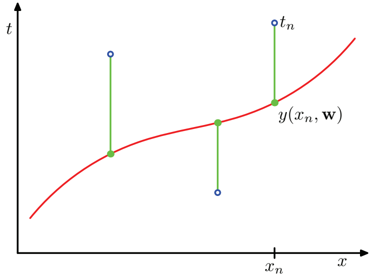
- By minimising an **error function**, a measure of misfit between function $y(x, \mathbf{w})$, for any given value of $\mathbf{w}$, and the training set data points
- A choice of error function is the sum of the squares of the errors between predictions $y(x_n, \mathbf{w})$ for each point x_n and corresponding target values t_n

$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} \Big(y(x_n, \mathbf{w}) - t_n \Big)^2 \qquad (2)$$

- It is a nonnegative quantity that would be zero if and only if function $y(x, \mathbf{w})$ were to pass through each training data point

Polynomial curve fitting (cont.)

Error function $E(\mathbf{w})$ corresponds to (one half of) the sum of the squares of the displacements (vertical green bars) of each data point t_n from function $y(x, \mathbf{w})$



$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\Big(y(x_n, \mathbf{w}) - t_n\Big)^2$$

To solve the curve fitting problem

- Choose the value of $\mathbf{w}$ for which $E(\mathbf{w})$ is as small as possible

($\star$) Because the error function is a quadratic function of the coefficients $\mathbf{w}$, its derivatives with respect to the coefficients will be linear in the elements of $\mathbf{w}$

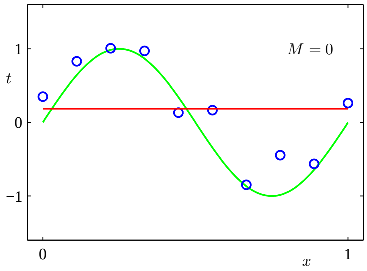
- The minimisation of $E(\mathbf{w})$ has a unique solution, $\mathbf{w}^\star$, found in closed form
- The resulting polynomial is given by the function $y(x, \mathbf{w}^\star)$

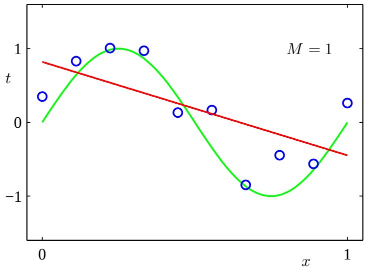
Polynomial curve fitting (cont.)

There still remains the problem of choosing the order M of the polynomial

- **model comparison** or **model selection**

Examples of the results of fitting polynomials of different order to the data

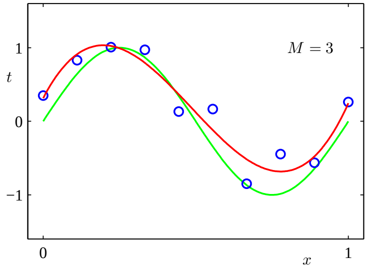




The constant ($M = 0$) and first order ($M = 1$) polynomials give poor fits to the data and consequently a rather poor representation of the function $\sin(2\pi x)$

Polynomial curve fitting (cont.)

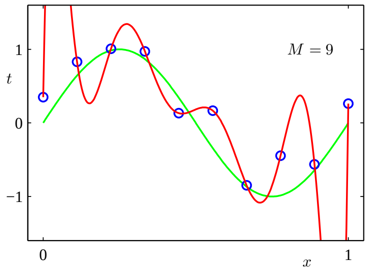
The third order ($M = 3$) polynomial gives a good fit to the function $\sin(2\pi x)$



Polynomial curve fitting (cont.)

A higher order polynomial ($M = 9$) gives an excellent fit to the training data

- The polynomial passes exactly through each data point and $E(\mathbf{w}^\star) = 0$



However, the fitted curve oscillates wildly and gives a very poor representation of the function $\sin(2\pi x)$, this is a behaviour that is known as **over-fitting**

Polynomial curve fitting (cont.)

How to get good generalisation by making accurate predictions for new data?

We can obtain some quantitative insight into the dependence of the generalisation performance on M by considering a **separate test set**

- 100 new points generated using the same procedure used to generate the training set points, but with new choices for the random noise values

For each choice of M, we evaluate the residual value of $E(\mathbf{w}^\star)$ for the training data, and we also evaluate the residual value of $E(\mathbf{w}^\star)$ for the test data set

It is sometimes more convenient to use the root-mean-square (RMS) error

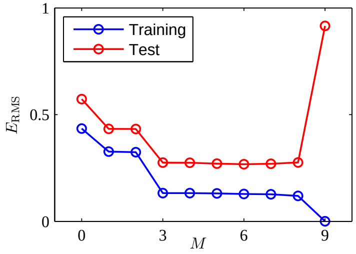
$$E_{RMS} = \sqrt{2E(\mathbf{w}^\star)/N} \tag{3}$$

The division by N allows a consistent comparison of datasets of different sizes and the square root ensures that E_{RMS} is measured on the same scale/units as t

Polynomial curve fitting (cont.)

Graphs of the training and test set RMS errors for various values of M

- The test error is a a measure of how well we are doing in predicting the values of t for new data observations of x

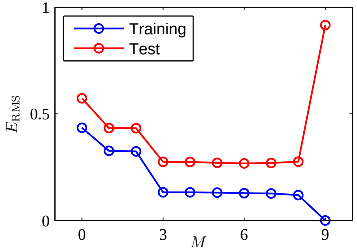


- Small values of M (0, 1 and 2) give relatively large values of test set error
- Values of M in the range $3 \leq M \leq 8$ give small values for the test set error

Low-order polynomials are rather inflexible and are incapable of capturing the oscillations in the function $\sin(2\pi x)$, whereas high-order polynomials are too flexible and capable of capturing also the oscillations in the additive noise

Polynomial curve fitting (cont.)

For $M = 9$, the training set error goes to zero (intuitively, as this polynomial contains 10 degrees of freedom corresponding to the 10 coefficients $w_0, \dots, w_9$)



- It can be tuned exactly to the 10 data points in the training set
- The test set error has become large
- The corresponding function $y(x, \mathbf{w}^\star)$ exhibits wild oscillations

Paradoxical: A polynomial of given order contains all lower order polynomials

- The $M = 9$ polynomial is at least as good as the $M = 3$ polynomial

Supposing that the best predictor would be the generating function $\sin(2\pi x)$

- A power series expansion of $\sin(2\pi x)$ contains terms of all orders
- The results should improve monotonically as we increase M

Polynomial curve fitting (cont.)

We can gain some insight into the problem by examining the values of the coefficients $\mathbf{w}^\star$ that are obtained by fitting polynomials of various order

- As M increases, the magnitude of the coefficients typically gets larger

For the $M = 9$ polynomial, the coefficients became finely tuned to the data

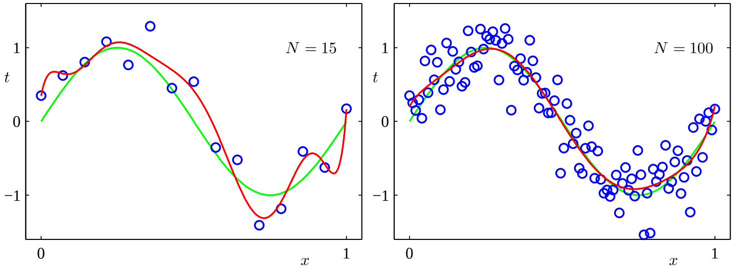
	$M=0$	$M=1$	$M=6$	$M=9$
$w_0^\star$	0.19	0.82	0.31	0.35
$w_1^\star$		-1.27	7.99	232.37
$w_2^\star$			-25.43	-5321.83
$w_3^\star$			17.37	48568.31
$w_4^\star$				-231639.30
$w_5^\star$				640042.26
$w_6^\star$				-1061800.52
$w_7^\star$				1042400.18
$w_8^\star$				-557682.99
$w_9^\star$				125201.43

- Large positive/negative values, so the polynomial matches each of the data points exactly
- Between points the function exhibits the large oscillations

Intuitively, what is happening is that the more flexible polynomials with larger values of M are becoming increasingly tuned to the random noise on the target

Polynomial curve fitting (cont.)

The behaviour of a model of given complexity ($M = 9$) as data size is varied



Over-fitting problem become less severe as the data size increases

- The larger the data set, the more complex (more flexible) the model that we can afford to fit to the data
- One heuristic is that the number of data points should be no less than some multiple (5 or 10) of the number of adaptive parameters in the model

Polynomial curve fitting (cont.)

Number of parameters is not necessarily the best measure of model complexity

- Why limit the number of parameters in a model according to the size of the available training set?
- It would seem more reasonable to choose model complexity according to the problem complexity

We shall see that the least squares approach to finding the model parameters represents a specific case of **maximum likelihood**, and that the over-fitting problem can be understood as a general property of maximum likelihood

- By adopting a **Bayesian approach**, the over-fitting problem can be avoided

We shall see that there is no difficulty from a Bayesian perspective in employing models for which the number of parameters greatly exceeds the data size

- In a Bayesian model the **effective number of parameters** adapts automatically to the size of the data set

Polynomial curve fitting (cont.)

We continue with the current approach and consider how we can apply it to data sets of limited size where we may wish to use relatively complex models

One technique that is often used to control over-fitting is **regularisation**

- It involves adding a penalty term to the error function $E(\mathbf{w})$, in order to discourage the coefficients from reaching large values
- The simplest such penalty term takes the form of a sum of squares of all of the coefficients, leading to a modified error function of the form

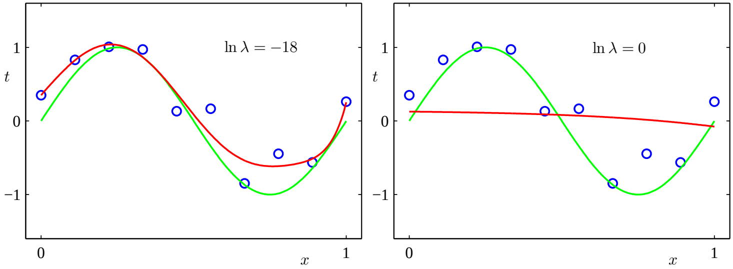
$$\tilde{E}(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\big(y(x_n, \mathbf{w}) - t_n\big)^2 + \frac{\lambda}{2}||\mathbf{w}||^2 \qquad (4)$$

- where $||\mathbf{w}||^2 = \mathbf{w}^T\mathbf{w} = w_0^2 + w_1^2 + \cdots + w_M^2$
- Coefficient λ governs the relative importance of the regularisation term, compared with the standard sum-of-squares error term

Again, the error function above can be minimised exactly in closed form ($\star$)

Polynomial curve fitting (cont.)

Fitting the polynomial of order $M = 9$ to the data using a regularised error



- For $\ln \lambda = -18$ (it's a small value for λ), over-fitting is suppressed
- For $\ln \lambda = 0$ (it's a large value for λ), we obtain again a poor fit

Polynomial curve fitting (cont.)

Coefficients $\mathbf{w}^\star$ for $M = 9$ polynomials with varying regularisation parameter λ

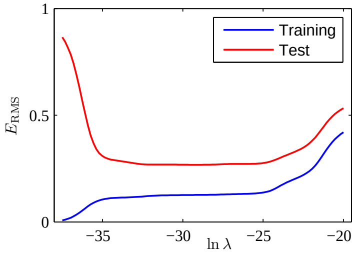
	$\ln\lambda = -\infty$	$\ln\lambda = -18$	$\ln\lambda = 0$
$w_0^\star$	0.35	0.35	0.13
$w_1^\star$	232.37	4.74	-0.05
$w_2^\star$	-5321.83	-0.77	-0.06
$w_3^\star$	48568.31	-31.97	-0.05
$w_4^\star$	-231639.30	-3.89	-0.03
$w_5^\star$	640042.26	55.28	-0.02
$w_6^\star$	-1061800.52	41.32	-0.01
$w_7^\star$	1042400.18	-45.95	-0.00
$w_8^\star$	-557682.99	-91.53	0.00
$w_9^\star$	125201.43	72.68	0.01

- $\ln\lambda = -\infty$ corresponds to an unregularised model ($\lambda = 0$)
- As λ increases, the magnitude of the coefficients gets smaller

Regularisation has the effect of reducing the magnitude of the coeffcients

Polynomial curve fitting (cont.)

The impact of the regularisation term on the generalisation error can be seen by plotting the RMS error (Equation 3) for both training and test sets against $\ln \lambda$



λ controls the effective model complexity and sets the degree of over-fitting

Polynomial curve fitting (cont.)

By minimising an error function, we have to determine a suitable value of model complexity, whether this is polynomial order M or regularisation coefficient λ

A simple way of achieving is to take the available data and then partition it into

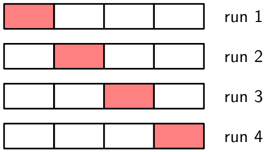
- A training set, to determine $\mathbf{w}$ for different settings of model complexity
- A separate **validation set**, to find an optimal value of model complexity

Seeing the inherent data scarcity, this is too wasteful of valuable training data

One solution to this is to use a technique commonly known as **cross-validation**

Polynomial curve fitting (cont.)

S-**fold cross-validation** allows a proportion $(S-1)/S$ of the available data to be used for training while making use of all of the data to asses performance



For the $S = 4$ case, we take all of the available data and the we partition it into S groups

- in the simplest case these are of equal size

$S-1$ of the groups (white blocks) are used to train a set of models, the remaining group (red block) is used for evaluating their performance

This procedure is then repeated for all S possible choices for the held-out group, the red blocks, and the performance scores from the S runs are then averaged

When data is particularly scarce, it is appropriate to consider the case $S = N$, where N is the total number of data points, **leave-one-out cross-validation**