# Lecture 14
# Introduction to Machine Learning: Part II

Rob Gaizauskas

# Lecture Outline

- Review of Key Concepts
- A More Abstract View of  Supervised Learning
- A Second Simple Supervised Learning Algorithm: Linear Regression

- Reading: (Readings that begin with * are **mandatory**)
  - *Russell and Norvig (2010), Chapter 18 "Learning from Examples", sections 18.1-18.3, 18.6.1
  - T. Mitchell. *Machine Learning.* McGraw Hill, 1997.
  - I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations, 2nd edition*, Morgan Kaufmann, San Francisco, 2005.

# Review of Key Concepts: What is Machine Learning ?

- A possible definition:

  *The study of how to design computer programs whose performance at some task improves through experience.*

  Or, more precisely, (Mitchell, 1997):

  **Definition**: A computer program is said to **learn**
  – from experience *E*
  – with respect to some class of tasks *T* and
  – performance measure *P*
  if its performance at tasks in *T* as measured by *P* improves with experience *E*

- Important question:
  – Are we only interested in *performance* of learning program?
  – Or are we also interested in discovering *human-comprehensible descriptions of patterns* in data? (*knowledge discovery*)

# Review of Key Concepts: 3 Main Types of Learning

- Supervised learning
  - Agent observes some input-output pairs and learns a function that maps from input to output
- Unsupervised learning
  - Agent learns patterns in the input even though no explicit feedback is given
    - Common example is clustering – detecting potentially useful clusters in input examples
- Reinforcement learning
  - Agent learns from series of reinforcements – rewards or punishments – received after it has performed a sequence of actions
    - Challenge is to decide which actions prior to the reinforcement were responsible for it

# Supervised Learning: Learning Input-Output Functions

- In supervised learning we are typically trying to learn a function $f$ from examples
- $f$ is referred to as the target function
- $f$ takes a vector-valued input, an n-tuple $\mathbf{x} = (x_1, x_2, \ldots x_n)$
- $f$ yields a vector-valued $k$-tuple as output, though often $k = 1$, i.e. $f$ produces a single output value

# Learning Input-Output Functions (2)

- Job of the learner is to output a hypothesis *h* which is its guess or approximation of the target function *f*
- *h* is assumed to be drawn from a class of functions $H$, called the hypothesis space
- Thus, learning may be viewed as search in a hypothesis space
- Note that *f* may or may not be in $H$ and this may or may not be known

# Learning Input-Output Functions (3)

- The learner selects a hypothesis $h$ based on a set $\Xi$ of training examples.
- Each training example is an input-output pair consisting of
  - an n-tuple $\mathbf{x}$ drawn from the set of input vectors over which $f$ is defined
  - the value of $f$ at $\mathbf{x}$, i.e. $f(\mathbf{x})$

# Learning Input-Output Functions (4)

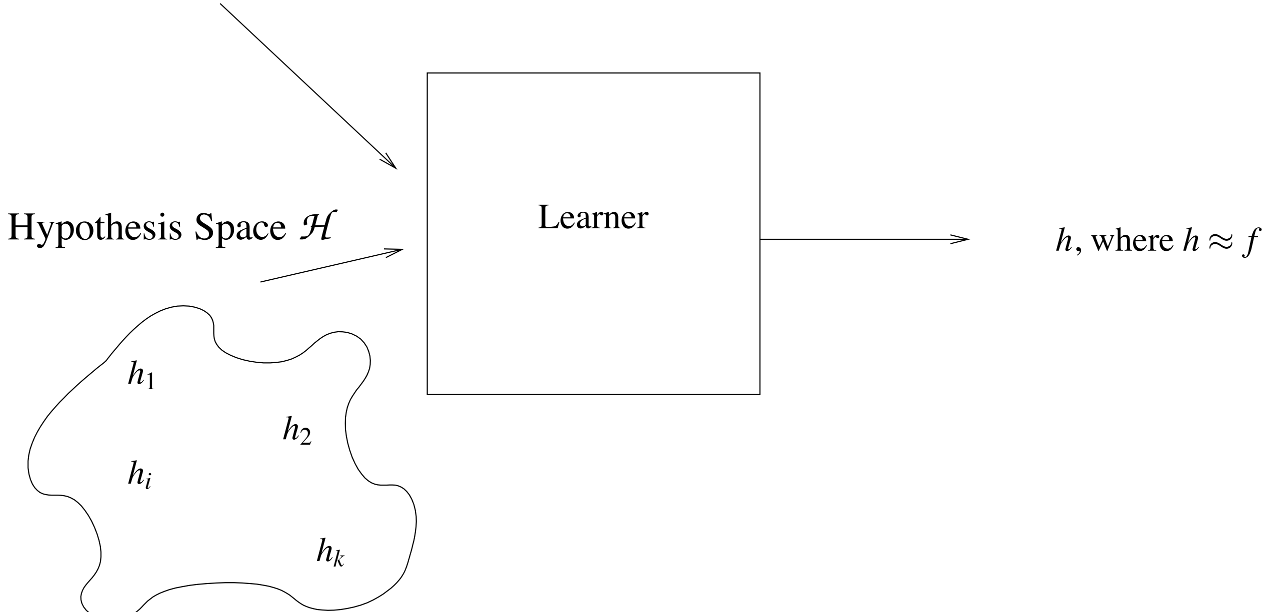
- The goal of the learner is to find an *h* that matches *f* as closely as possible
  - i.e. *h* should correctly predict the value of *f* for inputs not previously seen (not in the training set)
- If a hypothesis succeeds in correctly predicting the output value for unseen examples it is said to generalize well
- To evaluate a hypothesis the standard approach is to test it on a set of pairs of inputs and associated output values not used in training, referred to as the *test set*
  - The result of the evaluation is a measure of accuracy or, conversely, error associated with *h*

# General Setting

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \xrightarrow{\text{Target function } f} y \text{ or } \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_k \end{bmatrix}$$

Training Examples

$$\Xi = \{\mathbf{x}_1, \ldots \mathbf{x}_m\} + f(\mathbf{x}_i) \text{ for each } \mathbf{x}_i \in \Xi$$



# Input Vectors

- Input vectors go by a variety of names in the ML literature: input vector, pattern vector, feature vector, sample, example, instance
- Components $x_i$ of input vectors are variously called: features, attributes, input variables, components.
- Values of components generally of two major sorts:
  - numeric
    - also called ordinal and dividing into real-valued and discrete-valued numbers
  - nominal
    - also called categorical, enumerated or (confusingly) discrete

# Input Vectors: Example

- A loan applicant Fred might be an instance to a learner represented by the 6-tuple:

  (m, 27, 1, 3, u, 24376)

  where the attributes are *gender*, *age*, *in-work*, *years-at-current-job*, *education-code*, *salary*
  - *gender* is nominal (values *m* or *f*)
  - *age* is ordinal (discrete-valued number)
  - *in-work* is nominal – a special case of a boolean-valued attribute
  - *years-at-current-job* is ordinal (discrete-valued number)
  - *education-code* is nominal (some fixed set of code values, e.g. *u* = undergraduate degree
  - *salary* is ordinal (discrete-valued number)

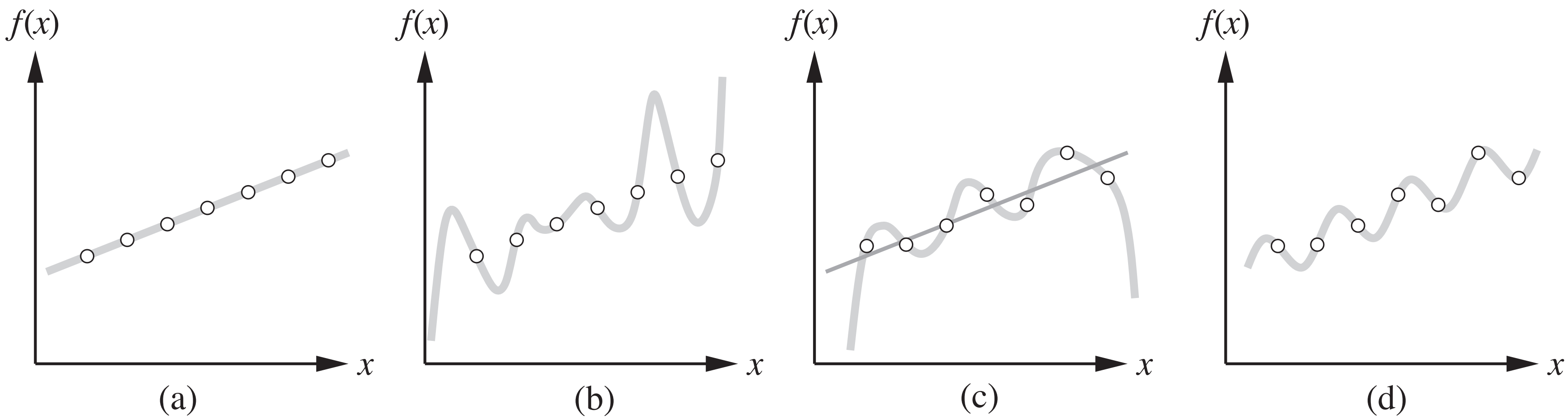
# Outputs

- If a learner produces a hypothesis *h* that outputs a real-number
  - The learning problem is called regression
  - The hypothesis is called a *function estimator*
  - Its output is called an *output value* or *estimate*
- If a learner produces a hypothesis *h* that outputs a categorical value
  - The learning problem is called classification
  - The hypothesis is called a *classifier*, *recognizer* or *categorizer*
  - Its output is called a *label*, *class*, *category* or *decision*
- Outputs may also be vector-valued with the components being numeric or nominal (or both)

# Outputs: Example

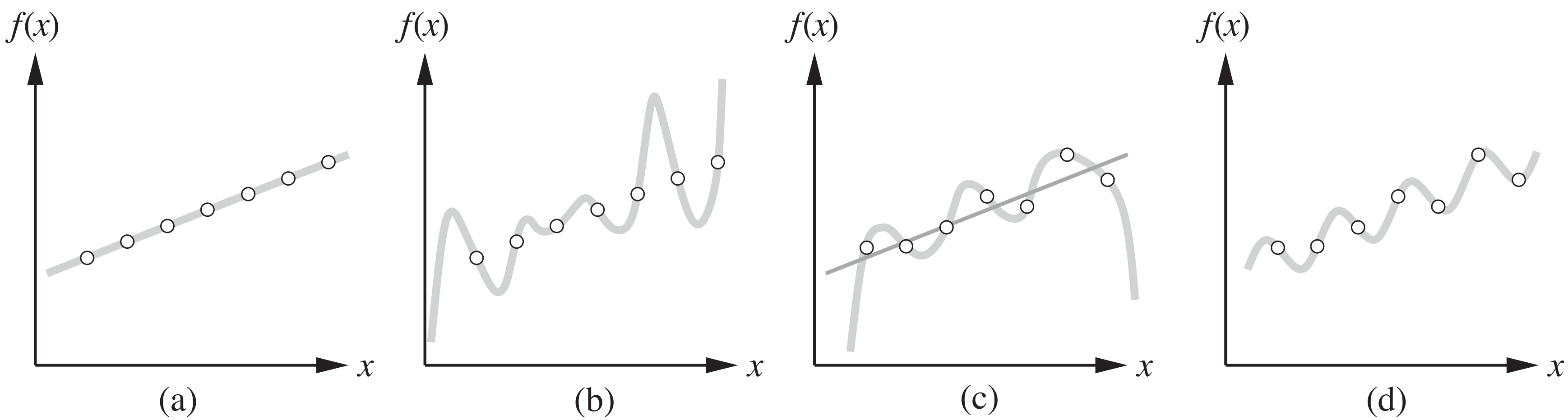
- In the loans applicant example, the learned hypothesis could be
  - a function estimator if the output is a real-number approximating the probability of Fred defaulting
  - a classifier if the output is a boolean 1 or 0 indicating whether Fred should be given a loan or not

# More Examples: Fitting a Function to Data



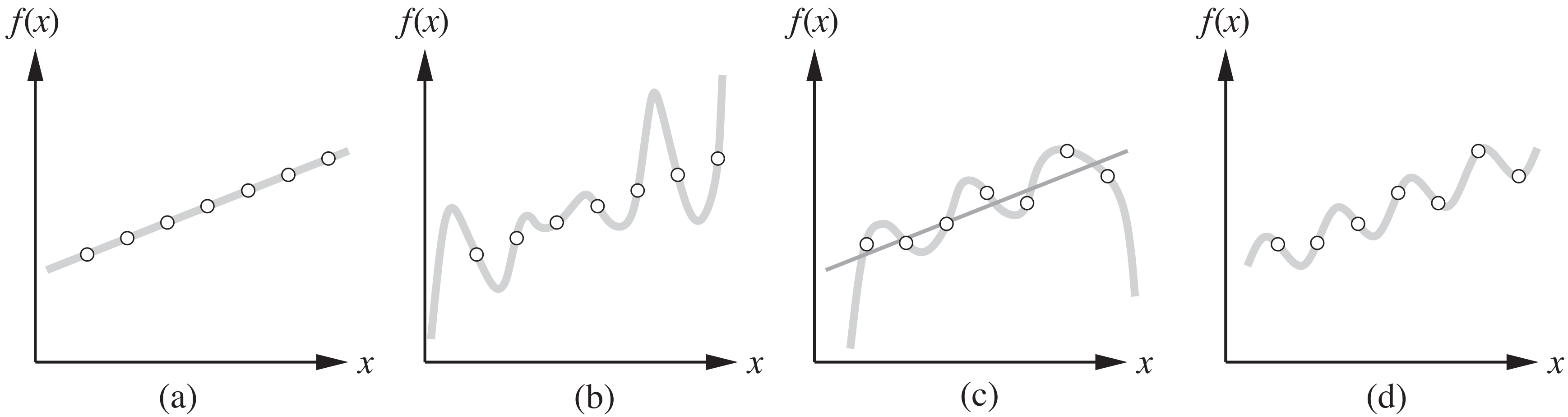

- Suppose hypothesis space $H$ is the set of polynomials, such as $x^5 + 3x^2 + 2$
- (a) and (b) show two polynomials consistent with (agree with all data points) the data
  - (a) is a linear hypothesis (0.4x + 3)
  - (b) is a degree 7 polynomial
  - How do we choose? – Ockham's razor – prefer the simplest

# More Examples: Fitting a Function to Data




- (c) shows a degree-6 polynomial that exactly fits the data (NB: different data set from (a) and (b))
  - Also shows a straight line inexact fit – might generalize better to new examples
  - In general tradeoff between complex hypotheses that fit well vs simpler hypotheses that may generalize better

# More Examples: Fitting a Function to Data




- (d) shows an exact fit where the function is a polynomial over sin(x), rather than just x, i.e. hypothesis is from a different hypothesis space
  - Shows importance of choice of hypothesis space
  - A learning problem is said to be realizable if the hypothesis space contains the true function
  - Cannot always tell if a learning problem is realizable as true function may not be known

# Expressiveness vs Complexity of $H$

- Why not always choose the set of all computable functions (equivalently, the set of all Java programs or Turing machines) as the hypothesis space $H$ ?
  - Would ensure that $H$ contains the true function if $f$ is computable – this is the best we can do
- Ignores the fact that in general

  There is a tradeoff between the expressiveness of a hypothesis space and the complexity of finding a good hypothesis in that space
- For example:
  - Fitting a straight line to data is an easy computation
  - Fitting a high-degree polynomial is harder
  - Fitting Turing machines is, in general, undecidable
- Also, difficulty of using *h* after learning needs to be considered:
  - When *h* is linear, computing *h(x)* is fast
  - If *h* is an arbitrary Turing machine, it is not even guaranteed to terminate

# A Second Simple Supervised Learning Algorithm: Linear Regression

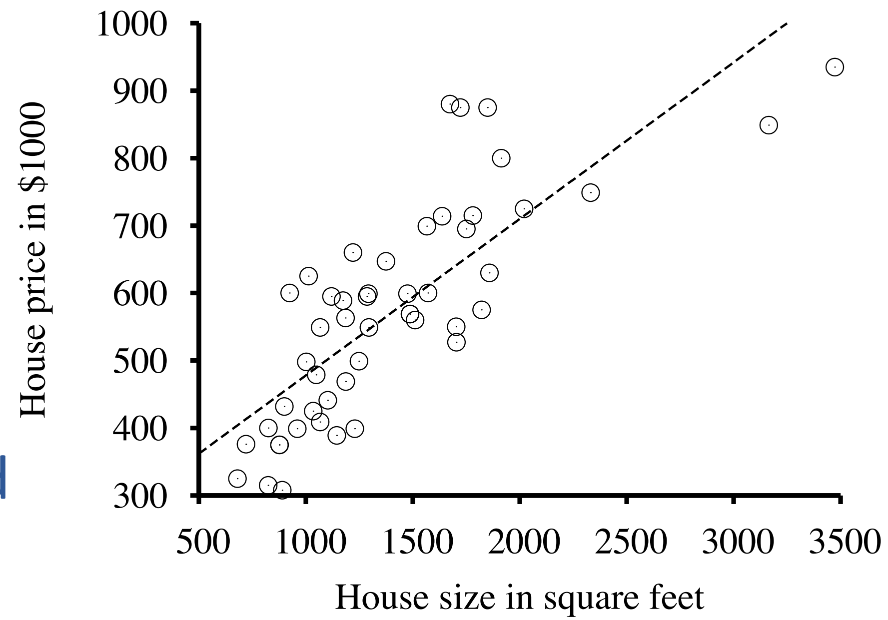
- Decision tree learners operate in a hypothesis space of decision trees
  - Typically used to learn classifiers which classify new instances into one amongst a finite set of discrete classes
- Linear regression learners operate over the hypothesis space of linear functions of continuous-valued inputs
- Here we consider just the case of regression with a univariate linear function, aka “fitting a straight line”

# Univariate Linear Regression

- Univariate linear functions have the form

  $$y = w_1 x + \ w_0$$

  where $w_0$ and $w_1$ are real-valued co-efficients to be learned (also called weights)




- Let **w** be the vector $[w_0, w_1]$ and define

  $$h_{\mathbf{w}}(x) = w_1 x + \ w_0$$

- Figure shows example of training set of *n* points in *x,y* plane, each point representing the [size,price] of a house
- The task of finding the best $h_{\mathbf{w}}$ that fits such data is called linear regression

# Squared Loss

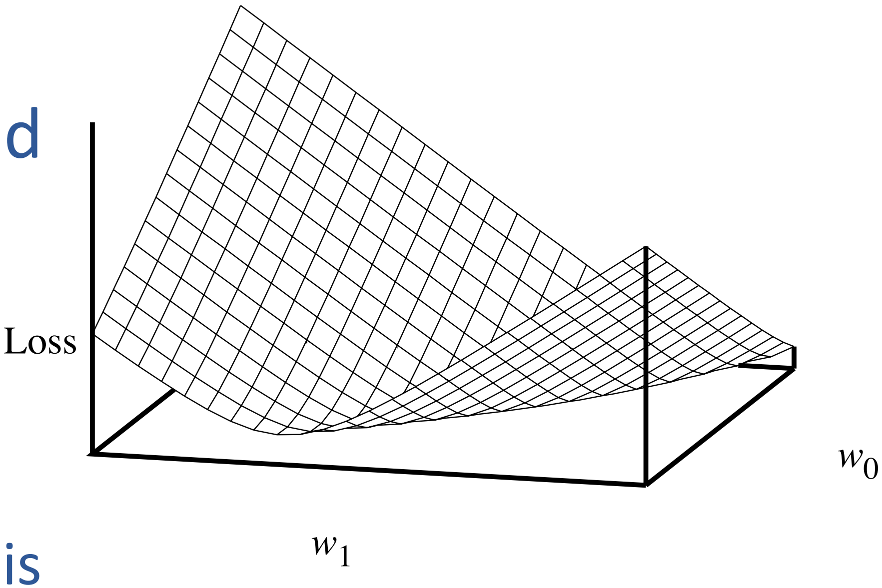
- Standard approach to fit a line to data is to find the values of the weights $[w_0, w_1]$ that minimise the loss over all training examples
- Loss is usually interpreted to mean the squared loss function, $L_2$, summed over all examples:

$$Loss(h_{\mathbf{w}}) = \sum_{j=1}^{N} L_2\,(y_j, h_{\mathbf{w}}(x_j)) = \sum_{j=1}^{N} (y_j - h_{\mathbf{w}}(x_j))^2 = \sum_{j=1}^{N} (y_j - (w_1 x_j + w_0))^2$$

- Think of this as minimising the sum of the squares of the distances of each point (training example) from the "hypothesis" line – a better line will have a smaller squared error

# Squared Loss (2)

- Loss is minimised when the partial derivatives wrt $w_0$ and $w_1$ are zero
- For univariate linear regression can plot loss function in 3D plot
  - Function is convex and there is a global minimum
- In this case can compute the minimum via a closed form solution (see Russell and Norvig p. 719, eqn. 18.3)



# Gradient Descent

- To go beyond linear models, equations defining minimum loss often have no closed-form for solution
- Standard approach is these cases is to use hill-climbing search that follows gradient of function to be optimized until a local minimum is found – called gradient descent
  - Can be used for linear regression as well
- Choose any point in weight space – in the case of univariate linear regression, a point in the $(w_{0,} w_1)$ plane
- Then move to a neighbouring point that is downhill and repeat until the process converges on a minimum possible loss:

**w** ← any point in parameter space
**loop** until convergence **do**
    **for each** $w_i$ **in w do**

$$w_i \leftarrow w_i - \alpha \frac{\delta}{\delta w_i} Loss(\mathbf{w})$$

Here α is a parameter called the learning rate

# Gradient Descent (2)

- Taking the partial derivatives of the loss function in the univariate linear regression case leads to the following update rules for the weights:

$$w_0 \leftarrow w_o - \alpha \sum_{j}^{N} (y_j - h_{\mathbf{w}}(x_j))$$

$$w_1 \leftarrow w_1 - \alpha \sum_{j}^{N} (y_j - h_{\mathbf{w}}(x_j)) \times x_j$$

(see Russell and Norvig for full derivation)

- These update rules make sense: if $h_w(x) > y$, i.e. output of hypothesis is too large
  - Reduce $w_0$ a bit
  - Reduce $w_1$ if $x$ was a positive input but increase $w_1$ if $x$ was a negative input

# Summary

- Machine learning is the study of how to design computer programs whose performance at some task improves through experience
- Any component of an agent can be improved by learning from data
- Three main types of learning are: unsupervised learning, reinforcement learning and supervised learning
- Supervised learning has been the most extensively studied and applied form of learning
- Supervised learning may be viewed as the learning of a target function from training examples
- Supervised learning algorithms can be divided into
  - Classification
  - Regression

  algorithms depending on whether their output is a categorical value or a real number
- Decision tree learning is an example of a classification algorithm
- Univariate linear regression is an example of a regression algorithm

# References

Mitchell, Tom (1997) *Machine Learning*. WCB/McGraw-Hill, Boston. See: http://www.cs.cmu.edu/˜tom/mlbook.html.

Russell, Stuart and Norvig, Peter (2010) *Artificial Intelligence: A Modern Introduction* (3rd ed). Pearson. Chapter 18.

# Learning Input-Output Functions: Performance Evaluation

- Important to know how well our learner is doing, i.e. how good the hypothesis it produces is
- Standard approach is to test hypothesis on a set of inputs and function values not used in training, referred to as the *test set*
- Common evaluation measures used are *mean-squared error* and *accuracy* (= proportion of instances misclassified)