582631 — 5 credits

Introduction to Machine Learning

Lecturer: Teemu Roos
Assistant: Ville Hyvönen

Department of Computer Science
University of Helsinki

(based in part on material by Patrik Hoyer and Jyrki Kivinen)

November 1st–December 16th 2016

Lectures 3–4:
Linear models & Evaluating performance II

November 8 & 11, 2016

Linear models

- We consider the case $\mathbf{x} \in \mathbb{R}^p$ throughout this lecture
- Function $f : \mathbb{R}^p \to \mathbb{R}$ is *linear* if for some $\boldsymbol{\beta} \in \mathbb{R}^p$ it can be written as

$$f(\mathbf{x}) = \boldsymbol{\beta} \cdot \mathbf{x} = \sum_{j=1}^{p} \beta_j x_j$$

  and *affine* if for some $\boldsymbol{\beta} \in \mathbb{R}^p$ and $a \in \mathbb{R}$ we can write

$$f(\mathbf{x}) = \boldsymbol{\beta} \cdot \mathbf{x} + a$$

- $\boldsymbol{\beta}$ is called *coefficient vector* and a is called *intercept* (or particularly in machine learning literature, *weight vector* and *bias*)

Linear models (2)

- Linear model generally means using an affine function by itself for regression, or as scoring function for classification
- The learning problem is to determine the parameters β and a based on data
- Linear regression and classification have been extensively studies in statistics

Univariate linear regression

- As warm-up, we consider linear regression in one-dimensional case $p = 1$
- We use square error and want to minimise it on training set $(x_1, y_1), \ldots, (x_n, y_n)$
- Thus, we want to find $a, \beta \in \mathbb{R}$ that minimise

$$E(\beta, a) = \sum_{i=1}^{n} (y_i - (\beta x_i + a))^2$$

- This is known as *ordinary least squares* and can be motivated as maximum likelihood estimate for (β, a) if we assume

$$y_i = \beta x_i + a + \epsilon_i$$

where ϵ_i are i.i.d. Gaussian noise with zero mean

Univariate linear regression (2)

- We solve the minimisation problem by setting the partial derivatives to zero
- We denote the solution by $(\hat{\beta}, \hat{a})$
- We have

$$\frac{\partial E(\beta, a)}{\partial a} = -2 \sum_{i=1}^{n} (y_i - \beta x_i - a)$$

and setting this to zero gives

$$\hat{a} = \bar{y} - \beta \bar{x}$$

where $\bar{y} = (1/n) \sum_i y_i$ and $\bar{x} = (1/n) \sum_i x_i$

- This implies in particular that the point $(\bar{x}, \bar{y})$ is on the line $y = \hat{\beta} x + \hat{a}$

Univariate linear regression (3)

- Further,

$$\frac{\partial E(\beta, a)}{\partial \beta} = -2 \sum_{i=1}^{n} x_i (y_i - \beta x_i - a)$$

- Plugging in $a = \hat{a}$ and setting the derivative to zero gives us

$$\sum_{i=1}^{n} x_i (y_i - \beta x_i - \bar{y} + \beta \bar{x}) = 0$$

from which we can solve

$$\hat{\beta} = \frac{\sum_{i=1}^{N} x_i (y_i - \bar{y})}{\sum_{i=1}^{N} x_i (x_i - \bar{x})}$$

Univariate linear regression (4)

- Since
$$\sum_{i=1}^{n} \bar{x}(y_i - \bar{y}) = \bar{x}\left(\sum_{i=1}^{n} y_i - n\bar{y}\right) = 0$$
and
$$\sum_{i=1}^{n} \bar{x}(x_i - \bar{x}) = \bar{x}\left(\sum_{i=1}^{n} x_i - n\bar{x}\right) = 0$$
we can finally rewrite this as
$$\hat{\beta} = \frac{\sum_{i=1}^{N}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{N}(x_i - \bar{x})^2}$$

- Notice that we have $\hat{\beta} = \sigma_{xy}/\sigma_{xx}$ where σ_{pq} is sample covariance between p and q:
$$\sigma_{pq} = \frac{1}{n-1}\sum_{i=1}^{n}(p_i - \bar{p})(q_i - \bar{q})$$

Multivariate linear regression

- We now move to the general case of learning a linear function $\mathbb{R}^p \to \mathbb{R}$ for arbitrary p
- We use the squared error, which is by far the most commonly used loss for linear regression
- One potential problem with squared error is its sensitivity to *outliers*
  - one alternative is absolute loss $\left|y - \hat{f}(x)\right|$
  - computations become trickier with absolute loss

Multivariate linear regression (2)

- We assume that the matrix $\mathbf{X} \in \mathbb{R}^{n \times p}$ has n instances $\mathbf{x}_i$ as its rows and $\mathbf{y} \in \mathbb{R}^n$ contains the corresponding labels y_i
- Terminology: $\mathbf{X}$ is the **design matrix**; elements of $\mathbf{x}_i$ are **covariates**; y_i is the **response**
- We write

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$$

  where the *residual* $\epsilon_i = y_i - \mathbf{x}_i \cdot \boldsymbol{\beta}$ indicates error that coefficient vector $\boldsymbol{\beta}$ makes on data point $(\mathbf{x}_i, y_i)$
- Our goal is to find $\boldsymbol{\beta}$ which minimises the sum of squared residuals

$$\sum_{i=1}^{n} \epsilon_i^2 = \|\boldsymbol{\epsilon}\|_2^2$$

Multivariate linear regression (3)

- By an argument involving matrix derivatives (or alternatively, orthogonal projections), we obtain the least squares solution which can be conveniently expressed using matrix notation.
- With $\mathbf{A}^{-1}$ denoting the matrix inverse of a (square) matrix $\mathbf{A}$, the solution is given by

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y}$$

- In R:

```
library(MASS)
lm.fit = lm(medv ~ crim, data = Boston)
lm.fit = lm(medv ~ ., data = D) # all variables
summary(lm.fit)
```

Multivariate linear regression (4)

- If the columns $\mathbf{c}_j$ of $\mathbf{X}$ are linearly independent, the matrix $\mathbf{X}^{\mathrm{T}}\mathbf{X}$ is of full rank and has an inverse
- For $n \geq p$, this is true except for degenerate special cases
- FOr $n < p$, this is never true, and no unique solution exists (We'll talk about the "large p, small n" case later.)
- $\mathbf{X}^{\mathrm{T}}\mathbf{X}$ is a $p \times p$ matrix, and inverting it takes $O(p^3)$ time
- For very high dimensional problems the computation time may be prohibitive

Useful trick

- It would be simpler to learn just linear functions and not worry about the intercept term separately
- An easy trick for this is to replace each instance $\mathbf{x}_i = (x_{i1}, \ldots, x_{ip}) \in \mathbb{R}^p$ by $\mathbf{x}'_i = (1, x_{i1}, \ldots, x_{ip}) \in \mathbb{R}^{p+1}$
- Now an affine function $f(\mathbf{x}) = \boldsymbol{\beta} \cdot \mathbf{x} + a$ in $\mathbb{R}^p$ becomes linear function $g(\mathbf{x}') = \boldsymbol{\beta}' \cdot \mathbf{x}'$ where $\boldsymbol{\beta}' = (a, \beta_1, \ldots, \beta_p)$
- If we write the set of instances $\mathbf{x}_1, \ldots, \mathbf{x}_n$ as an $n \times p$ matrix, this means adding an extra column of ones

Useful trick (2)

- For most part we now present algorithms for learning linear functions (instead of affine)
- In practice, to run them on p-dimensional data, we add the column of ones and run the algorithm in $p + 1$ dimensions
- The first component of $\boldsymbol{\beta}$ then gives the intercept
- However sometimes we might still want to treat the intercept separately (for example in *regularisation*)

Nonlinear models by transforming the input

- *Linear* regression can also be used to fit models which are *nonlinear* functions of the input
- Example: For fitting a degree 5 polynomial

$$y_i = f(x_i) = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \beta_3 x_i^3 + \beta_4 x_i^4 + \beta_5 x_i^5$$

. . . create the input matrix

$$\mathbf{X} = \begin{pmatrix} 1 & x_1 & x_1^2 & x_1^3 & x_1^4 & x_1^5 \\ 1 & x_2 & x_2^2 & x_2^3 & x_2^4 & x_2^5 \\ 1 & x_3 & x_3^2 & x_3^3 & x_3^4 & x_3^5 \\ 1 & x_4 & x_4^2 & x_4^3 & x_4^4 & x_4^5 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \end{pmatrix}, \text{ and } \mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ \vdots \end{pmatrix}$$

Nonlinear predictors by transforming the input (2)

- We can also explicitly include some interaction terms, as in

$$y_i = f(\mathbf{x}_i) = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \beta_3 x_{i1} x_{i2}$$

  using the following input matrix:

$$\mathbf{X} = \begin{pmatrix} 1 & x_{11} & x_{12} & x_{11}x_{12} \\ 1 & x_{21} & x_{22} & x_{21}x_{22} \\ 1 & x_{31} & x_{32} & x_{31}x_{32} \\ 1 & x_{41} & x_{42} & x_{41}x_{42} \\ \vdots & \vdots & \vdots & \vdots \end{pmatrix}, \quad \text{and} \quad \mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ \vdots \end{pmatrix}$$

- See the book (page 87 onwards) for more on this.

Evaluating model performance

Evaluating models: Outline

- A fundamental issue in machine learning is that we build models based on training data, but really care about performance on new unseen test data
- *Generalisation* refers to the learned model's ability to work well also on unseen data
  - good generalisation: what we learned from training data also applies to test data
  - poor generalisation: what seemed to work well on training data is not so good on test data

Goals for this topic

- Familiarity with the basic ideas of evaluating generalisation performance of (supervised) learning system
- Ability to explain overfitting and underfitting with examples
- Ability to explain with examples the idea of model complexity and its relation to overfitting and underfitting
- Using separate training, validation and test sets and cross validation in practice

How good is my classifier?

- Apply the learned classifier to the training data?
  - a simple model will not be able to fit all the training data perfectly
  - the more complex the model, the better it typically fits
  - in particular, in *nested* model classes such as polynomials of increasing order, a more complex model always fits better than a simpler model
  - at the extreme case, we could fit a model that is flexible enough to fit *any* data *perfectly*

  $\Rightarrow$ does this suggest that a complex model is always better?

- Of course not... the goal of learning is to perform well on *new* (unseen) data. How can we test that?

- Note that we almost invariable make the basic assumption that future data comes from the same source as the training data. Otherwise we're doomed!

Statistical learning model

Setting the stage:

- We consider supervised learning: goal is to learn a function $\hat{f} : \mathcal{X} \to \mathcal{Y}$.
- During learning, we create $\hat{f}$ based on training set $\{(x_1, y_1), \ldots, (x_N, y_N)\}$ where $(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$
- Later we test $\hat{f}$ on unseen data points $\{(x_{N+1}, y_{N+1}), \ldots, (x_{N+M}, y_{N+M})\}$
- We have a **loss function** $L : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ and wish to minimise the average loss on unseen data

$$\frac{1}{M} \sum_{i=1}^{M} L(\hat{f}(x_{N+i}), y_{N+i})$$

Statistical learning model (2)

- Loss function $L(\hat{y}, y)$: How much does it "cost" us if we predict $\hat{y}$ when the outcome if y.
- We're already familiar with the **squared error** in regression:

$$L(\hat{y}, y) = (\hat{y} - y)^2$$

- In classification, the most straightforward loss function is the **zero–one loss**:

$$L(\hat{y}, y) = \begin{cases} 0, & \text{if } \hat{y} = y \\ 1, & \text{otherwise} \end{cases}$$

- Asymmetric loss functions can be more sensible in many situations:

$$L(\hat{y}, y) = \begin{cases} 0, & \text{if } \hat{y} = y \\ a, & \text{if } \hat{y} = 1, y = 0 \\ b, & \text{if } \hat{y} = 0, y = 1 \end{cases}$$

Statistical learning model (3)

(loss functions continued...)

- A classifier can also make probabilistic predictions and output a probability distribution $\hat{p}$ over the values of y.
- In the probabilistic case, an interesting loss function is the **logarithmic loss** (or *log-loss* for short):

$$L(\hat{p}, y) = -\log \hat{p}(y) \geq 0$$

- and many more...
- Furthermore, sometimes when minimizing the actual loss function is hard, we may use a *surrogate loss* function that is similar to the actual loss function but easier to manipulate — we'll return to this in connection to Support Vector Machines

Statistical learning model (4)

- Assume that there is a fixed but unknown probability distribution P over $\mathcal{X} \times \mathcal{Y}$ such that pairs are (x_i, y_i) are independent samples from it
- We say the data points are *independent and identically distibuted* (i.i.d.)
- We wish to minimise the *generalisation error* (also called *risk*) of $\hat{f}$, which is the expected loss

$$\mathrm{E}_{(x,y)\sim P}[L(\hat{f}(x), y)]$$

where $\mathrm{E}_{(x,y)\sim P}[\cdot]$ denotes expectation when a single data point (x, y) is drawn from P

Statistical learning model (5)

- If P were known, this would just be an optimization problem:

$$\min_{\hat{f}} E_{(x,y)\sim P}[L(\hat{f}(x), y)]$$

- (This problem could be very hard to solve, but it wouldn't be a statistical problem.)

- Since P is not known, *learning* comes to the picture

Statistical learning model (6)

- The key is that we have training data drawn from P, so that we can use it to make more or less accurate inferences about properties of P
- In particular, based on the law of large numbers (and as we have seen), the average loss is close to the expected loss with high probability:

$$\sum_{i=1}^{n} \frac{1}{n} L(\hat{f}(x_i), y_i) \approx E_{(x,y)\sim P}[L(\hat{f}(x), y)]$$

- For zero–one loss, the difference between the average and the expected loss can be bounded (with high probability) by Hoeffding's inequality; see Exercise 1.1
- ...but remember the problem when there are many models!

Overfitting

- *Overfitting* means creating models that follow too closely the specifics of the training data, resulting in poor performance on unseen data
- Overfitting often results from using too complex models with too little data
  - complex models allow high accuracy but require lots of data to train
  - simple models require less training data but are incapable of modelling complex phenomena accurately
- Choosing the right model complexity is a difficult problem for which there are many methods (incl. cross validation; Exercise 1.3)

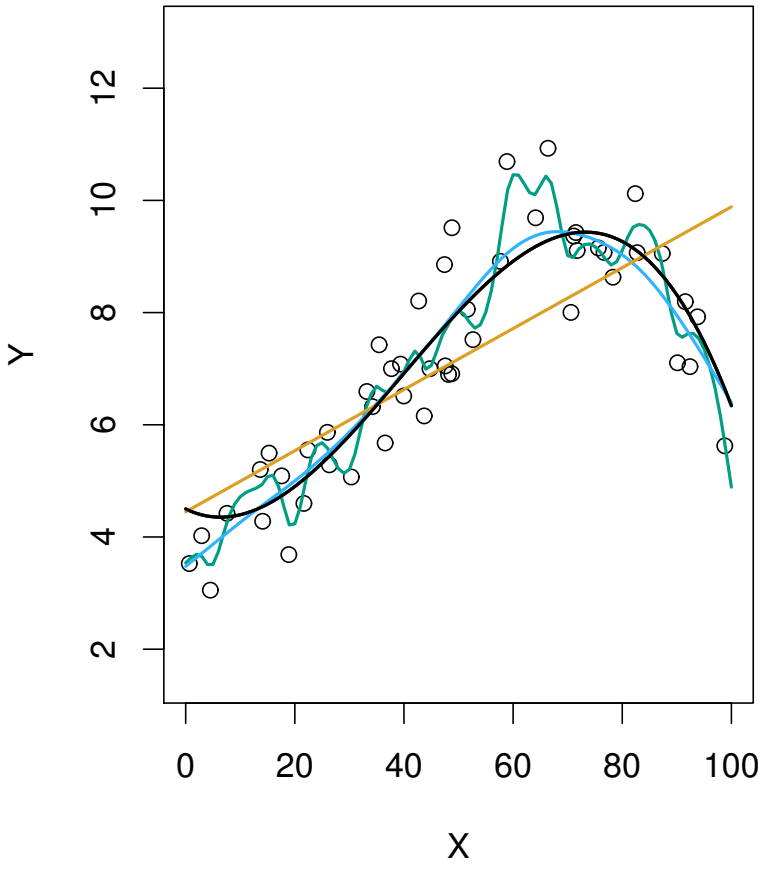
What is model complexity?

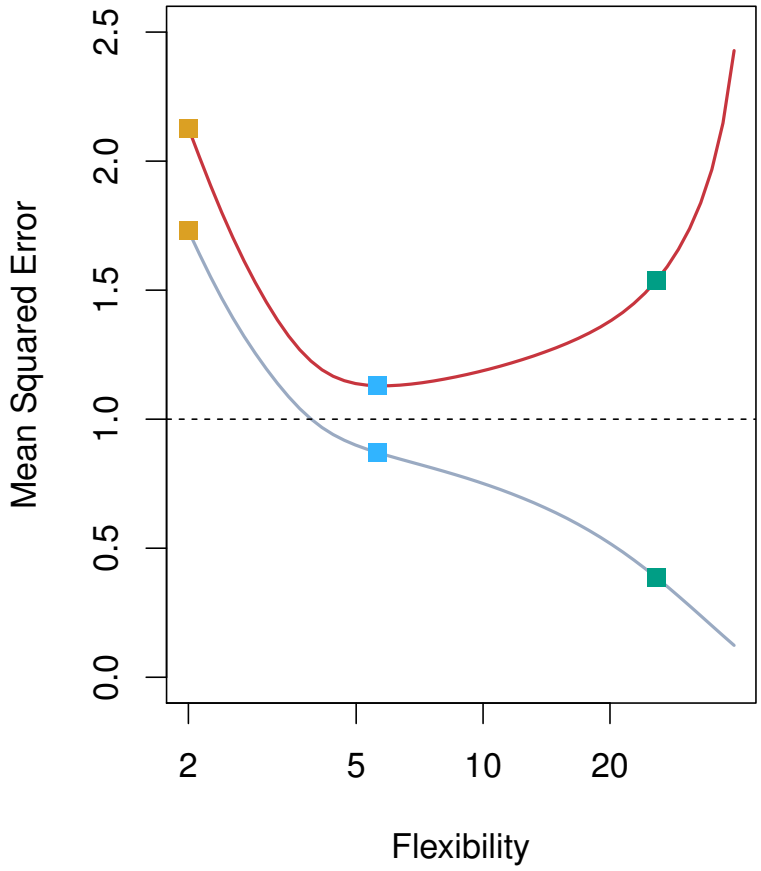
- The simplest case is the one where the number of models available is finite; see again Exercise 1.1
- For *parametric* models the number of parameters can be used to obtain a measure of complexity (e.g. linear model in p dimensions, degree k polynomial)
- Some non-parametric models also have intuitive complexity measures (e.g. based on the number of nodes in decision tree)
- There are also less obvious parameters that can be used to control overfitting (e.g. kernel width, parameter k in kNN, norm of coefficient vector in linear model)
- Mathematical study of various formal notions of complexity is a vast field; we'll scratch the surface

Error vs flexibility (train and test)

- *Left:* Data source (black line), data (circles), and three regression models of increasing complexity; *Right:* training and test errors (squared error) of the three models

(Figure 2.9 from the course textbook)

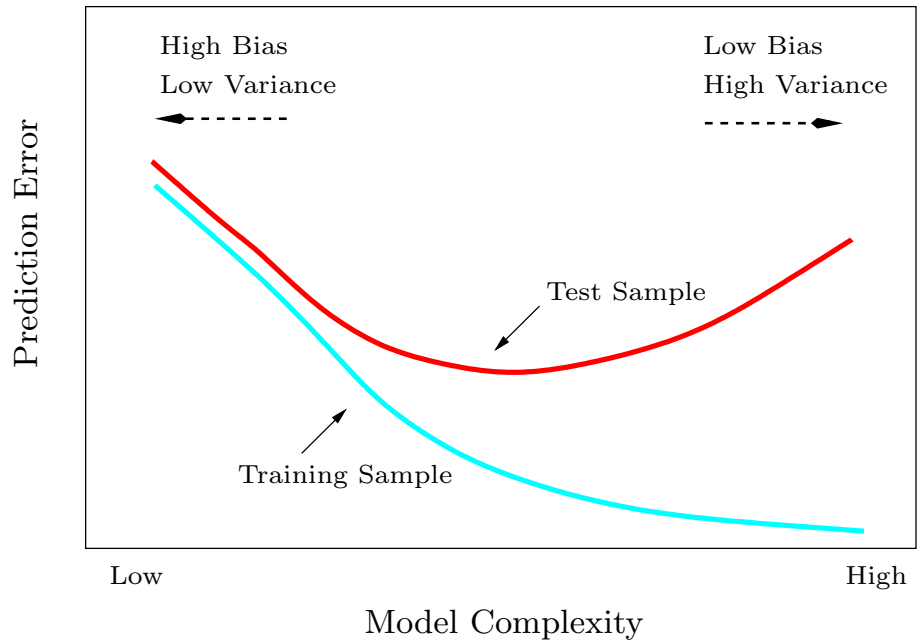




Error vs flexibility (train and test)

- Typical behaviour: The higher the model complexity (more flexible model) the lower the error on the training sample. However, the error curve for a test sample is U-shaped.

(figure from Hastie et al, 2009)



Bias-variance tradeoff

- Based on N training datapoints from the distribution, how close is the learned classifier to the optimal classifier?

  Consider multiple trials: repeatedly and independently drawing N training points from the underlying distribution.

  - *Bias:* how far the average model (over all trials) is from the real optimal classifier
  - *Variance:* how far a model (based on an individual training set) tends to be from the average model

- Goal: Low bias and low variance.

- High model complexity $\Rightarrow$ low bias and high variance
  Low model complexity $\Rightarrow$ high bias and low variance

Bias-variance for regression

- Bias and variance have a particular mathematical meaning in regression with square loss
- Let $\hat{f}_S : \mathcal{X} \to \mathbb{R}$ be the model our algorithm produces from training set S
- Let $f_*(x)$ be the prediction of some "target" function f_* (say, Bayes optimal)
- The loss of $\hat{f}$ with respect to the target on a given point x is

$$(f_*(x) - \hat{f}_S(x))^2$$

- Taking expectation over all possible training sets gives

$$\mathrm{E}_S[(f_*(x) - \hat{f}_S(x))^2]$$

Bias-variance for regression (2)

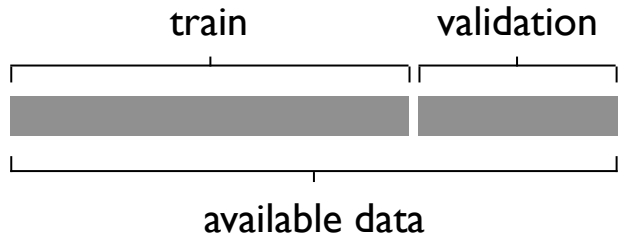
- Write $\bar{f}(x) = \mathrm{E}_S[\hat{f}(x)]$ for the average prediction of our algorithm on x
- A straightforward calculation gives the decomposition

$$\begin{aligned} &\mathrm{E}_S[(f_*(x) - \hat{f}_S(x))^2] \\ &\quad = \quad (f_*(x) - \bar{f}_S(x))^2 + \mathrm{E}_S[(\hat{f}_S(x) - \bar{f}(x))^2] \end{aligned}$$

- *bias* $(f_*(x) - \bar{f}_S(x))^2$ measures how much our "aiming point" $\bar{f}(x)$ is off the "target" $f_*(x)$
- *variance* $\mathrm{E}_S[(\hat{f}_S(x) - \bar{f}(x))^2]$ measures how much the actual prediction $\hat{f}_S(x)$ wanders around the "aiming point" due to random training set

Using 'validation' data to overcome overfitting

1. Split the data into 'train' and 'validation' subsets:



2. Fit models with varying complexity on 'training' data, e.g.
   - regression with different covariate subsets (feature selection)
   - decision trees with variable number of nodes
   - support vector machines with different regularization parameters

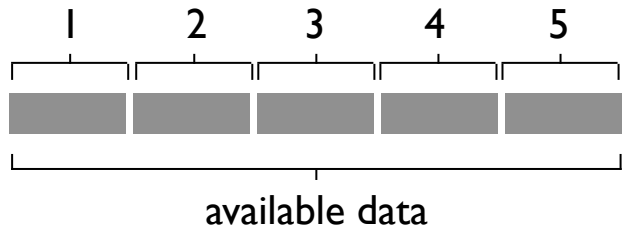
3. Choose the subset/number-of-nodes/regularization based on performance on the 'validation' set

(An issue: the amount of training data is not the same as in the original problem. Also: trade-off between the amount of training vs validation data)

Cross-validation

To get more reliable statistics than a single 'split' provides, use K-fold *cross-validation* (see Exercise 1.3.c):

1. Divide the data into K equal-sized subsets:



2. For j goes from 1 to K:
   - 2.1 Train the model(s) using all data except that of subset j
   - 2.2 Compute the resulting validation error on the subset j

3. Average the K results

When $K = N$ (i.e. each datapoint is a separate subset) this is known as *leave-one-out* cross-validation.