

An introduction to Machine Learning

Pierre Geurts

p.geurts@ulg.ac.be

Last update: 3/10/2017

Institut Montefiore
University of Liège



Outline

- Introduction
- Supervised Learning
- Other learning protocols/frameworks

Machine Learning: definition

Two definitions:

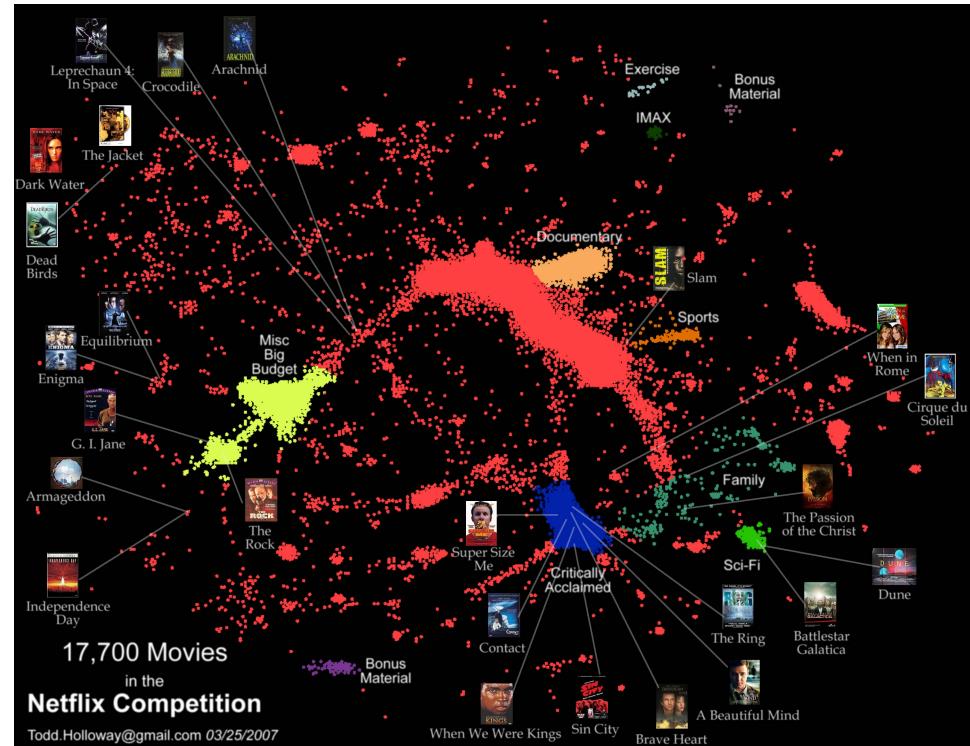
- Machine Learning is concerned with the design, the analysis, and the application of algorithms that allow computers to learn
 - A computer learns if it improves its performance at some task with experience (i.e., by collecting **data**)
- Machine Learning is concerned with the design, the analysis, and the application of algorithms to extract a model of a system from the sole observation (or the simulation) of this system in some situations (i.e., by collecting **data**).
 - A model can be any relationship between the variables used to describe the system.
 - Two main goals: make predictions and better understand the system

Machine learning: when ?

- Learning is useful when:
 - Human expertise does not exist
navigating on Mars
 - Humans are unable to explain their expertise
speech recognition
 - Solution changes in time
routing on a computer network
 - Solution needs to be adapted to particular cases
user biometrics
- Example:
 - It is easier to write a program that learns to play checkers or backgammon well by self-play rather than converting the expertise of a master player to a program.

Applications: recommendation system

- Netflix prize: predict how much someone is going to love a movie based on his movies preferences
- Data: over 100 million ratings that over 480,000 users gave to nearly 18,000 movies
- Reward: \$1,000,000 dollars if 10% improvement with respect to Netflix's system in 2006 (two teams succeeded in 2009)



<http://www.netflixprize.com>

Applications: robotics

Machine learning is a core technology within robotics:

- To better sense the environment (computer vision, sound recognition...)
- To decide on which sequence of actions to take to perform a given task (reinforcement learning, imitation learning...)



Applications: credit risk analysis

- Data:

Customer103: (time=t0)

Years of credit: 9
Loan balance: \$2,400
Income: \$52k
Own House: Yes
Other delinquent accts: 2
Max billing cycles late: 3
Profitable customer?: ?
...

Customer103: (time=t1)

Years of credit: 9
Loan balance: \$3,250
Income: ?
Own House: Yes
Other delinquent accts: 2
Max billing cycles late: 4
Profitable customer?: ?
...

Customer103: (time=tn)

Years of credit: 9
Loan balance: \$4,500
Income: ?
Own House: Yes
Other delinquent accts: 3
Max billing cycles late: 6
Profitable customer?: No
...

- Logical rules automatically learned from data:

```
If      Other-Delinquent-Accounts > 2, and
      Number-Delinquent-Billing-Cycles > 1
Then   Profitable-Customer? = No
        [Deny Credit Card application]
If      Other-Delinquent-Accounts = 0, and
        (Income > $30k) OR (Years-of-Credit > 3)
Then   Profitable-Customer? = Yes
        [Accept Credit Card application]
```

Some applications at ULg

Wide area control of power systems

(ULg, PEPITe, Hydro-Québec)



Problem

- ▶ Improve emergency control scheme
 - ▶ Churchill-Falls power plant
 - ▶ Reduce probability of blackout

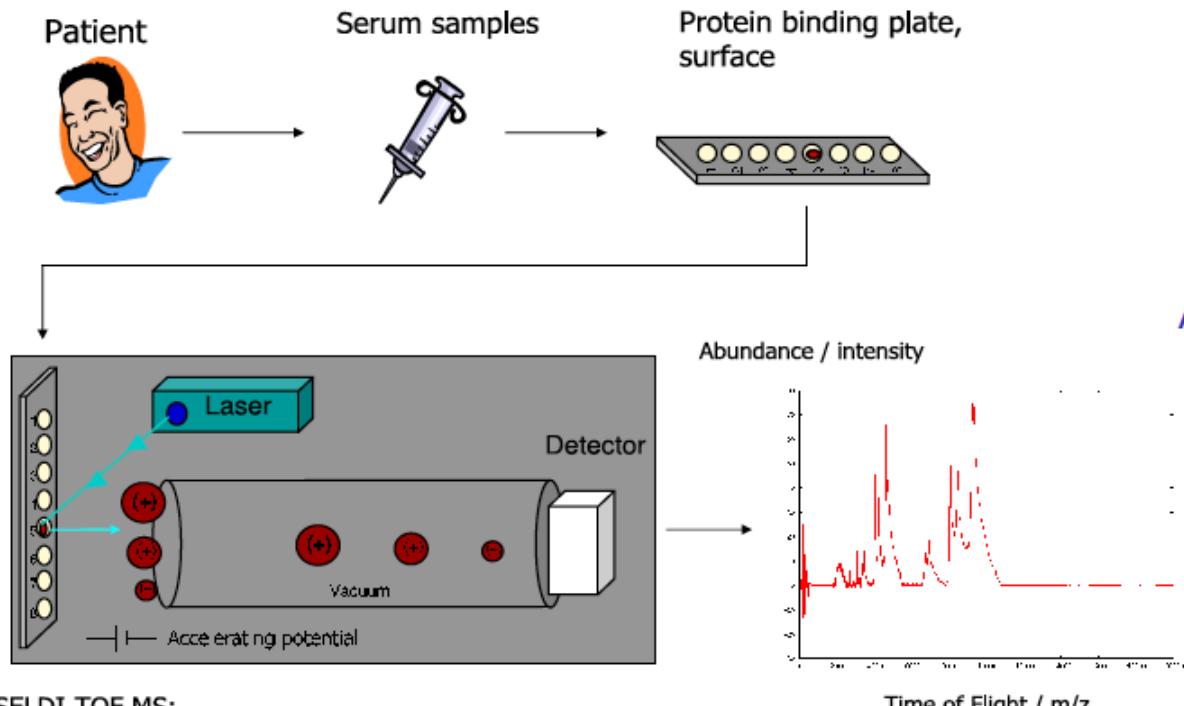
Approach

- ▶ 10,000 real-time snapshots sampled (several years)
- ▶ Massive time-domain simulations
- ▶ Automatically learn decision rules to determine optimal amount of generation and load to trip
- ▶ Implement rules in real-time
- ▶ **New rules enhance security**

Some applications at ULg

Medical diagnosis

(CBIG/GIGA collaboration)



SELDI-TOF MS:

Surface Enhanced Laser Desorption/ Ionisation Time of Flight Mass Spectrometry

Problem

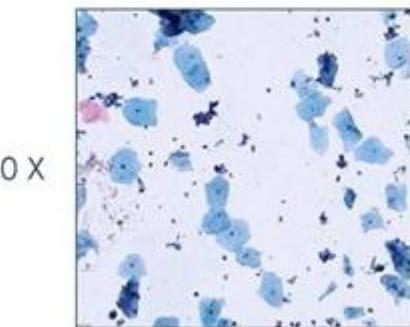
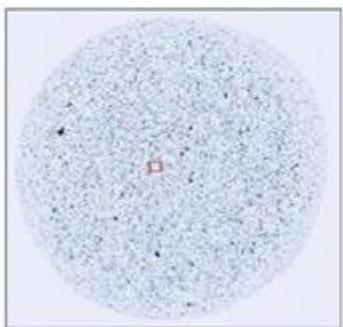
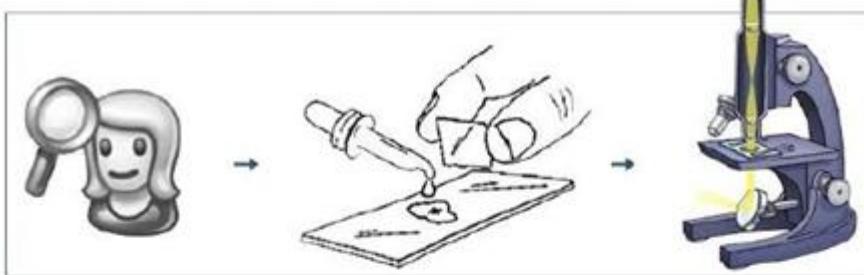
- ▶ Diagnosis of Rhumatoid Arthritis and other inflammatory diseases

Approach [GFd⁺04]

- ▶ Proteomic analysis of serum samples
- ▶ Automatic learning to
 - ▶ identify biomarkers (protein fragments) specific of disease
 - ▶ derive classifier for medical diagnosis

Some applications at ULg

Computer-aided cytology



Problem

- Early diagnosis of disease based on cytological tests
- Improve detection of rare abnormal cells among tens of thousands of normal cells

Approach

- Pathologists collect a database of normal and abnormal cells
- Automatically learn a cell classification model
- Scan whole-slide digital images ($\geq 40000 \times 40000$ pixels) and rank most suspicious ones for expert review

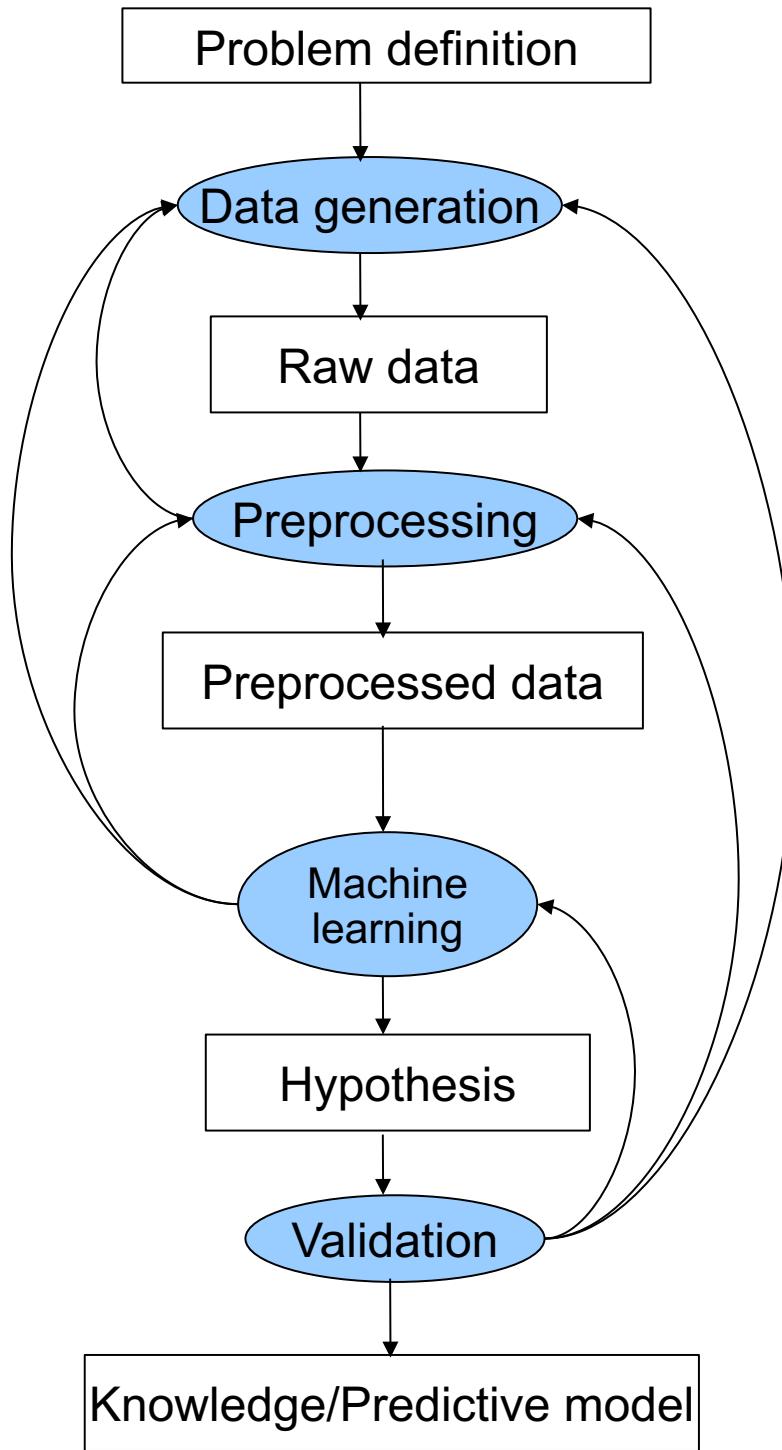
(Collaboration with US company CellSolutions and PEPITe)

Other applications

- Machine learning has a wide spectrum of applications including:
 - Retail: Market basket analysis, Customer relationship management (CRM)
 - Finance: Credit scoring, fraud detection
 - Manufacturing: Optimization, troubleshooting
 - Power systems: monitoring, control
 - Medicine: Medical diagnosis
 - Telecommunications: Quality of service optimization, routing
 - Bioinformatics: Motifs, alignment, network inference
 - Web mining: Search engines
 - ...

Related fields

- Artificial Intelligence: smart algorithms
- Statistics: inference from a sample
- Computer Science: efficient algorithms and complex models
- Systems and control: analysis, modeling, and control of dynamical systems
- Data Mining/data science: searching through large volumes of data



One part of the data mining process

- Each step generates many questions:
 - Data generation: **data types, sample size, online/offline...**
 - Preprocessing: **normalization, missing values, feature selection/extraction...**
 - Machine learning: **hypothesis, choice of learning paradigm/algorithm...**
 - Hypothesis validation: **cross-validation, model deployment...**

Glossary

- Data=a table (dataset, database, sample)

Variables (attributes, features) =
measurements made on objects

	VAR 1	VAR 2	VAR 3	VAR 4	VAR 5	VAR 6	VAR 7	VAR 8	VAR 9	VAR 10	VAR 11	...
Object 1	0	1	2	0	1	1	2	1	0	2	0	...
Object 2	2	1	2	0	1	1	0	2	1	0	2	...
Object 3	0	0	1	0	1	1	2	0	2	1	2	...
Object 4	1	1	2	2	0	0	0	1	2	1	1	...
Object 5	0	1	0	2	1	0	2	1	1	0	1	...
Object 6	0	1	2	1	1	1	1	1	1	1	1	...
Object 7	2	1	0	1	1	2	2	2	1	1	1	...
Object 8	2	2	1	0	0	0	1	1	1	1	2	...
Object 9	1	1	0	1	0	0	0	0	1	2	1	...
Object 10	1	2	2	0	1	0	1	2	1	0	1	...
...

Objects (samples, observations,
individuals, examples, patterns)

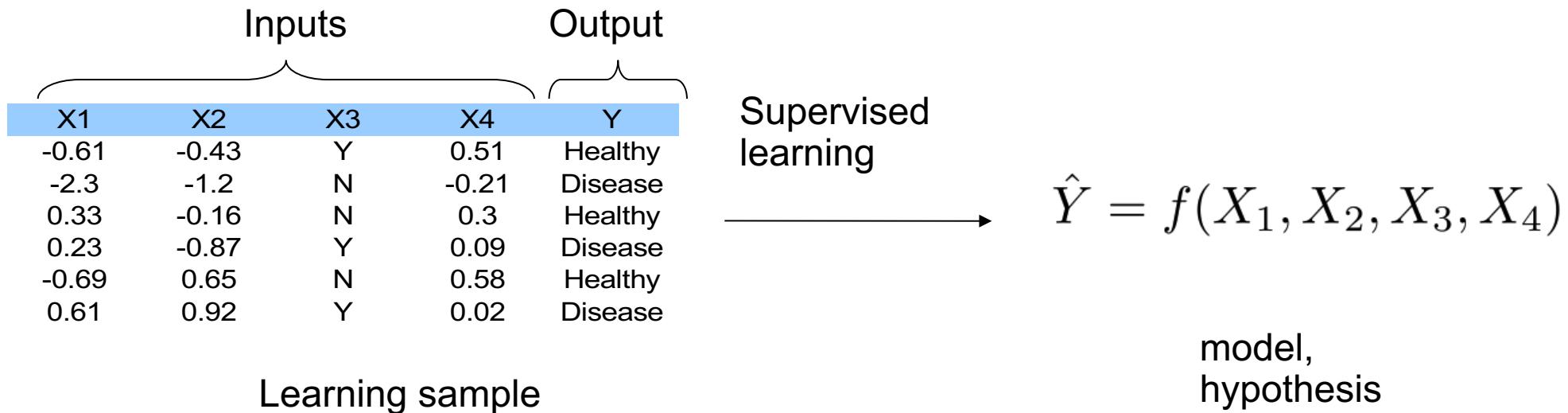
Dimension=number of variables
Size=number of objects

- Objects: samples, patients, documents, images...
- Variables: genes, proteins, words, pixels...

Outline

- Introduction
- Supervised Learning
 - Introduction
 - Model selection, cross-validation, overfitting
 - Some supervised learning algorithms
 - Beyond classification and regression
- Other learning protocols/frameworks

Supervised learning



- Goal: from the database (learning sample), find a function f of the inputs that approximates **at best** the output
- Formally:

From a learning sample $\{(x_i, y_i) | i = 1, \dots, N\}$ with $x_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$, find a function $f : \mathcal{X} \rightarrow \mathcal{Y}$ that minimizes the expectation of some loss function $\ell : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}$ over the joint distribution of input/output pairs:

$$E_{x,y}\{\ell(f(x), y)\}$$

- Symbolic output \Rightarrow *classification*, Numerical output \Rightarrow *regression*

Two main goals

- Predictive:

Make predictions for a ***new*** object described by its attributes

X1	X2	X3	X4	Y
-0.71	-0.27	T	-0.72	Healthy
-2.3	-1.2	F	-0.92	Disease
0.42	0.26	F	-0.06	Healthy
0.84	-0.78	T	-0.3	Disease
-0.55	-0.63	F	-0.02	Healthy
0.07	0.24	T	0.4	Disease
0.75	0.49	F	-0.88	?

- Informative:

Help to understand the relationship between the inputs and the output

$$\hat{Y} = \text{disease if } X_3 = F \text{ and } X_2 < 0.3$$

Find the most relevant inputs

Example of applications

- Biomedical domain: medical diagnosis, differentiation of diseases, prediction of the response to a treatment...

Gene expression, Metabolite concentrations...

Patients	X1	X2	...	X4	Y
	-0.1	0.02	...	0.01	Healthy
	-2.3	-1.2	...	0.88	Disease
	0	0.65	...	-0.69	Healthy
	0.71	0.85	...	-0.03	Disease
	-0.18	0.14	...	0.84	Healthy
	-0.64	0.15	...	0.03	Disease

Example of applications

- Perceptual tasks: handwritten character recognition, speech recognition...

7 2 1 0 4 1 4 9 5 9
0 6 9 0 1 5 9 7 3 4
7 6 6 5 4 0 7 4 0 1
3 1 3 4 7 2 7 1 2 1
1 7 4 2 3 5 1 2 4 4
6 3 5 5 6 0 4 1 9 5
7 8 9 3 7 4 6 4 3 0
7 0 2 9 1 7 3 2 9 7
1 6 2 7 8 4 7 3 6 1
3 6 9 3 1 4 1 7 6 9

- Inputs:

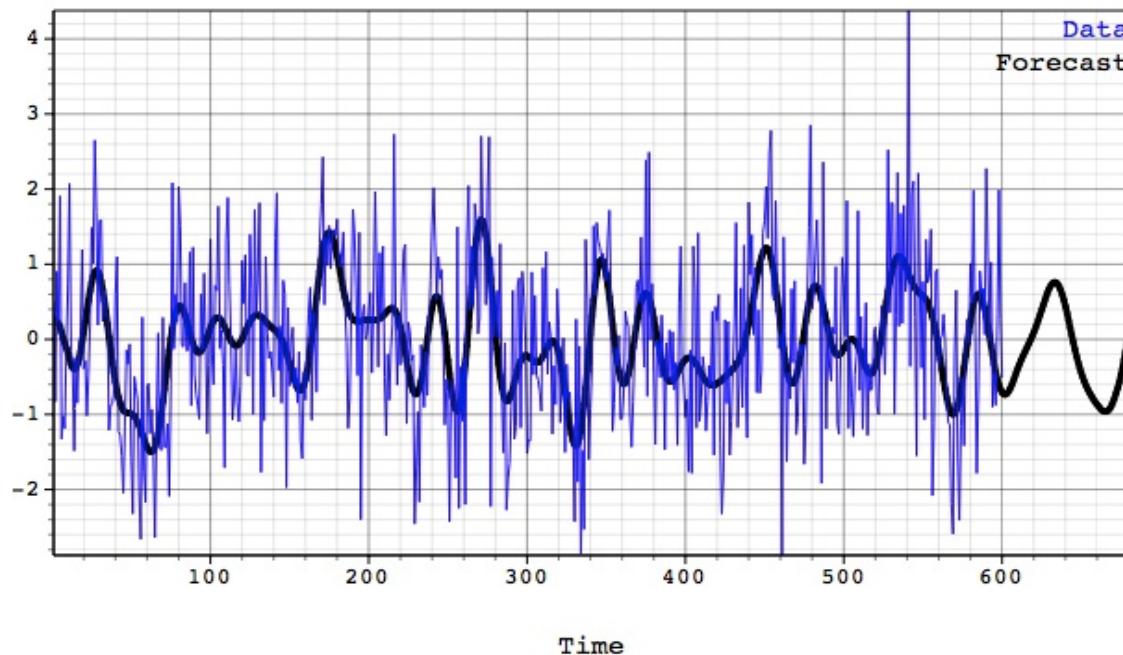
- a grey intensity [0,255] for each pixel
- each image is represented by a vector of pixel intensities
- eg.: $32 \times 32 = 1024$ dimensions

- Output:

- 9 discrete values
- $Y = \{0, 1, 2, \dots, 9\}$

Example of applications

- Time series prediction: predicting electricity load, network usage, stock market prices...



Past course projects

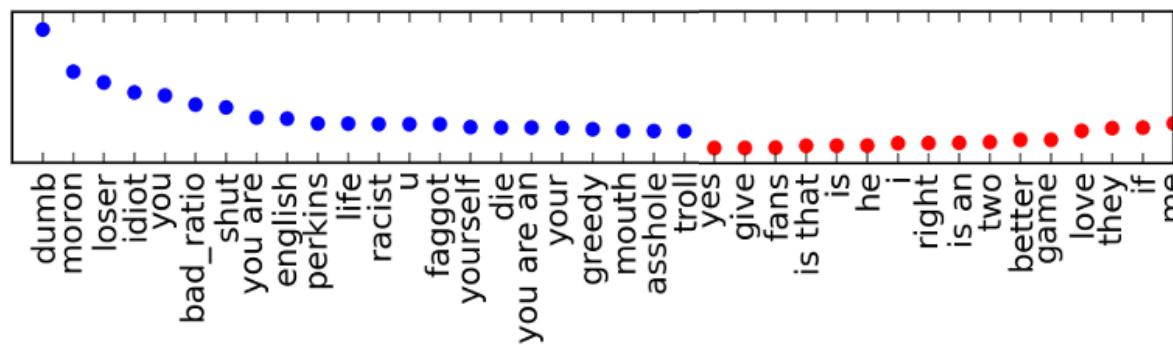
- Develop a system to detect North Atlantic right whales calls from audio recordings (to prevent collisions with shipping traffic)



- Develop a system to recognize (German) traffic signs (for autonomous driving)



- Detect insulting messages in social commentary



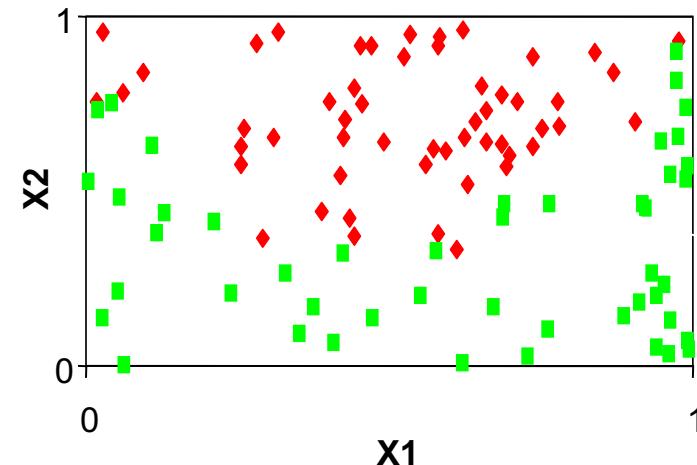
Outline

- Introduction
- Supervised Learning
 - Introduction
 - Model selection, cross-validation, overfitting
 - Some supervised learning algorithms
 - Beyond classification and regression
- Other learning protocols/frameworks

Illustrative problem

- Medical diagnosis from two measurements (eg., weights and temperature)

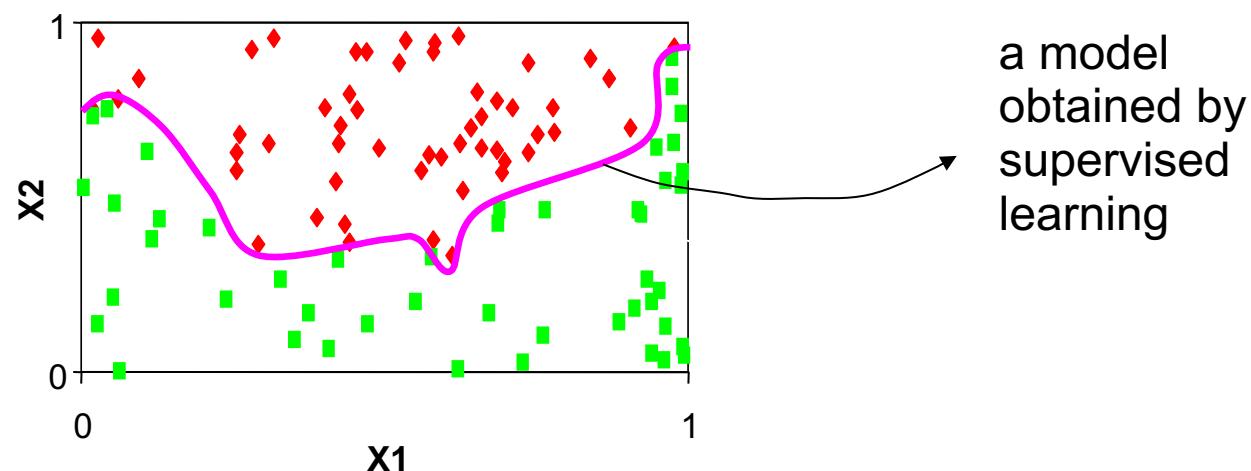
X1	X2	Y
0.93	0.9	Healthy
0.44	0.85	Disease
0.53	0.31	Healthy
0.19	0.28	Disease
...
0.57	0.09	Disease
0.12	0.47	Healthy



- Goal: find a model that classifies at best **new** cases for which X_1 and X_2 are known

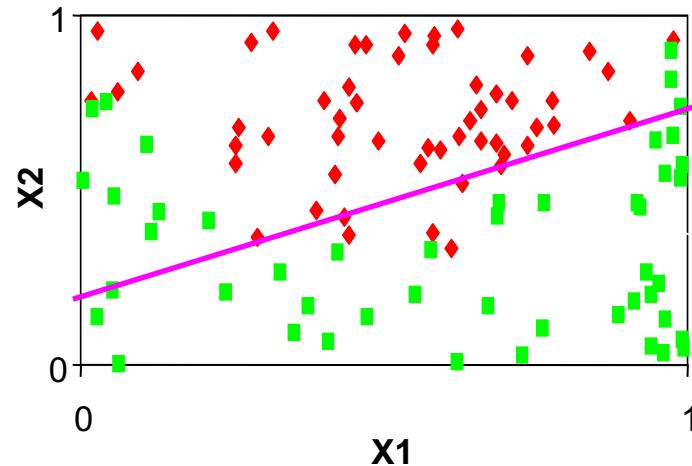
Learning algorithm

- A learning algorithm is defined by:
 - a family of candidate models (=**hypothesis space H**)
 - a quality measure for a model
 - an optimization strategy
- It takes as input a learning sample and outputs a function h in H of maximum quality



Linear model

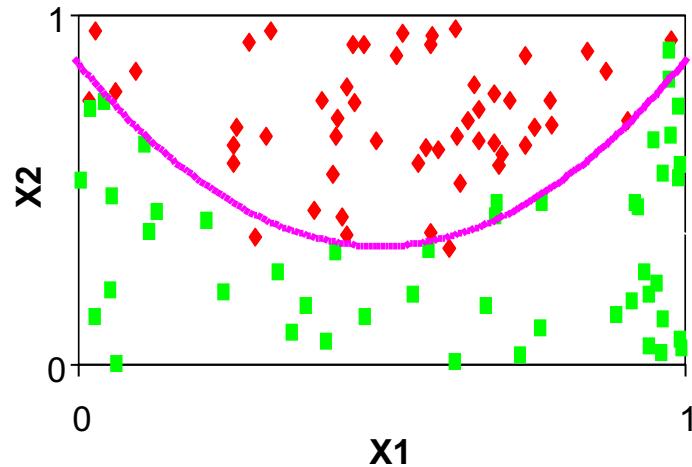
$$h(X_1, X_2) = \begin{cases} \text{Disease if } w_0 + w_1 * X_1 + w_2 * X_2 > 0 \\ \text{Normal otherwise} \end{cases}$$



- Learning phase: from the learning sample, find the best values for w_0 , w_1 and w_2
- Many alternatives even for this simple model (LDA, Perceptron, SVM...)

Quadratic model

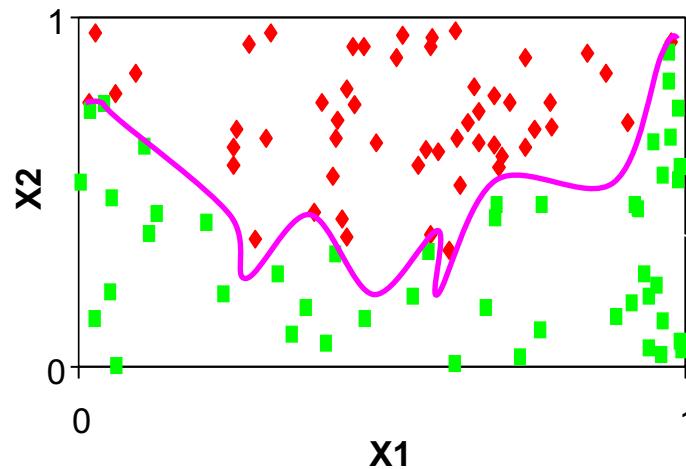
$$h(X_1, X_2) = \begin{cases} \text{Disease if } w_0 + w_1 * X_1 + w_2 * X_2 + w_3 * X_1^2 + w_4 * X_2^2 > 0 \\ \text{Normal otherwise} \end{cases}$$



- Learning phase: from the learning sample, find the best values for w_0, w_1, w_2, w_3 and w_4
- Many alternatives even for this simple model (LDA, Perceptron, SVM...)

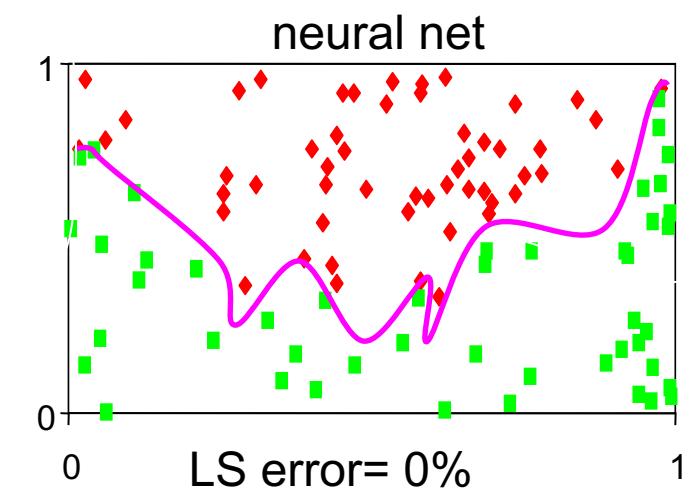
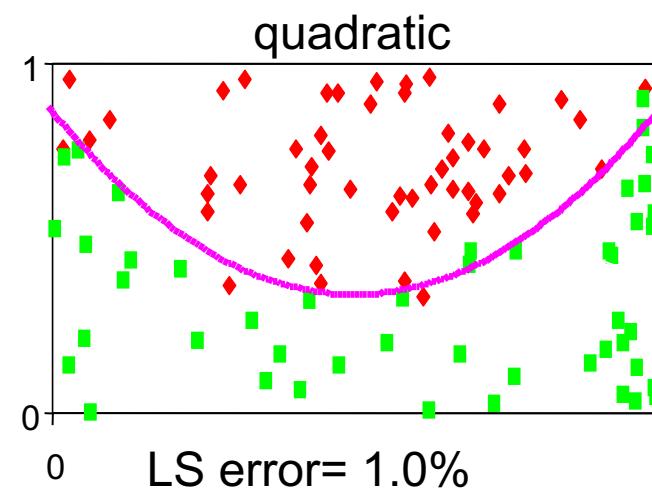
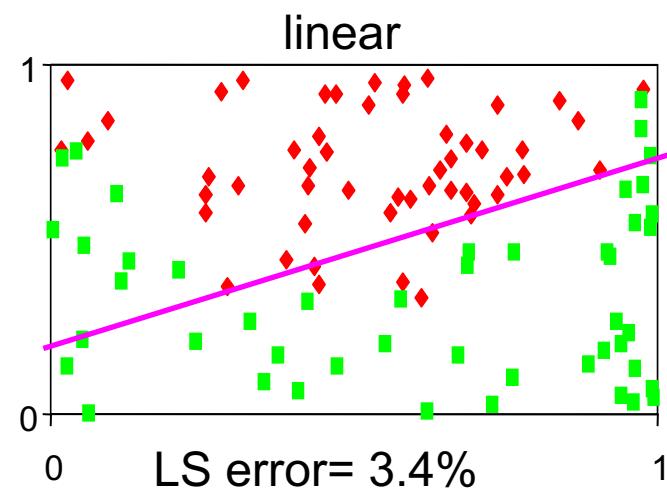
Artificial neural network

$$h(X_1, X_2) = \begin{cases} \text{Disease if } \textcolor{red}{\text{some very complex function of } X_1, X_2 > 0} \\ \text{Normal otherwise} \end{cases}$$

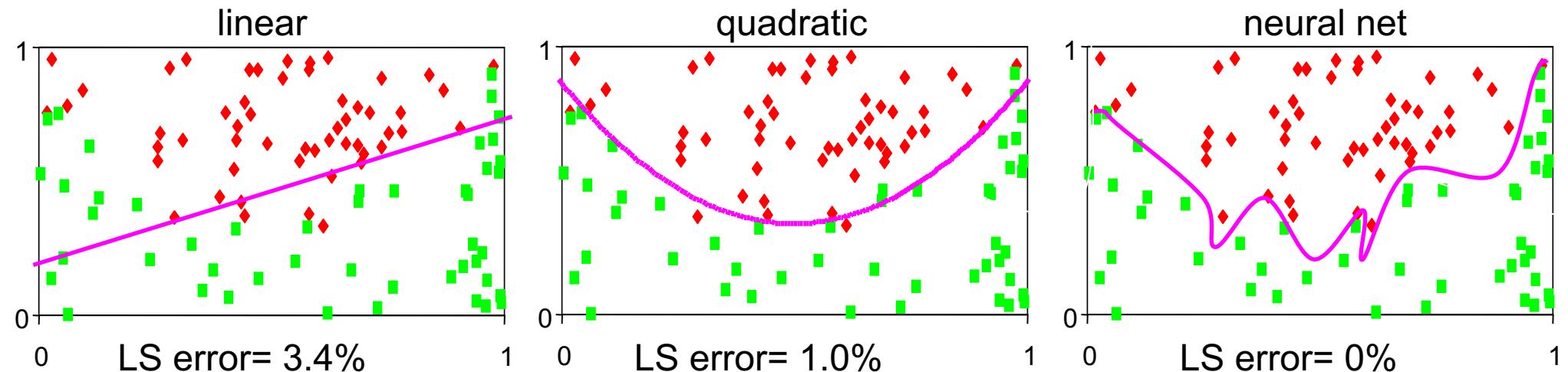


- Learning phase: from the learning sample, find the numerous parameters of the very complex function

Which model is the best?

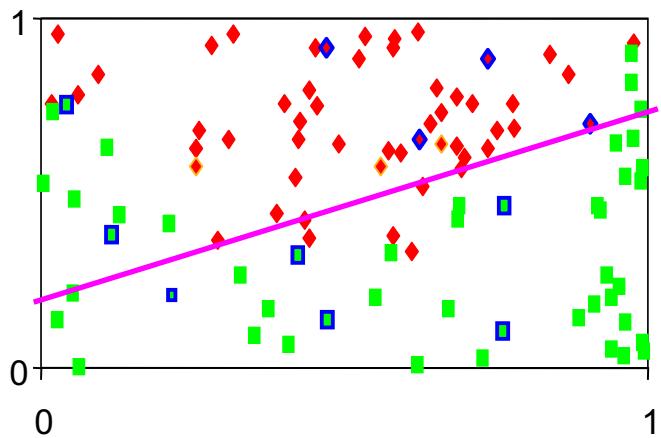


Which model is the best?



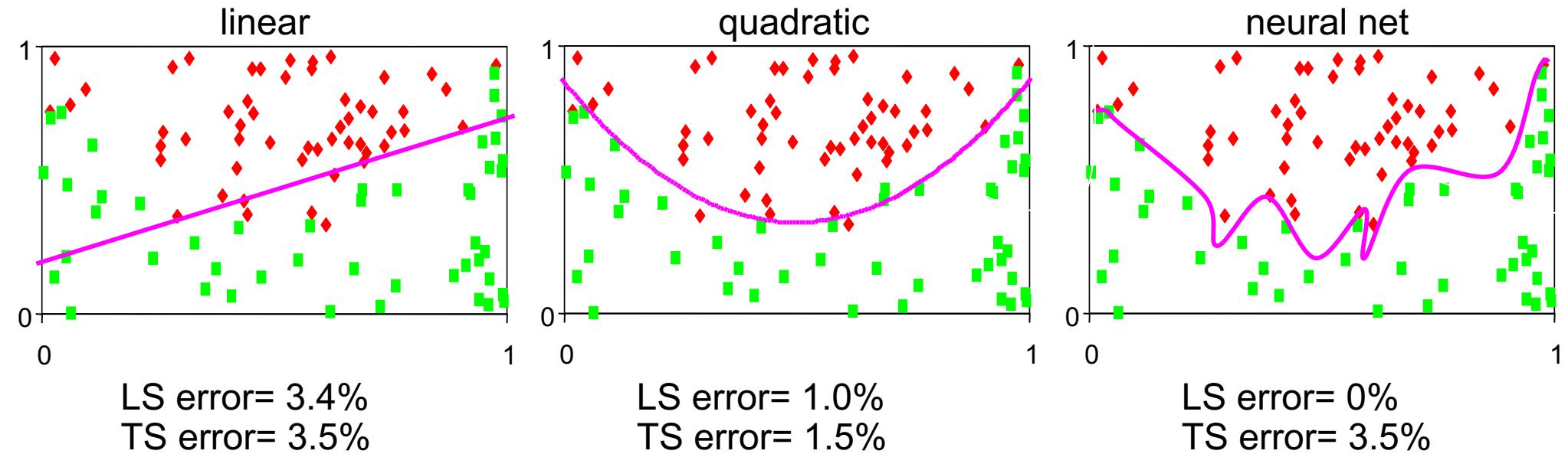
- Why not choose the model that minimises the error rate on the learning sample? (also called *re-substitution error*)
- How well are you going to predict future data drawn from the same distribution? (*generalisation error*)

The test set method



1. Randomly choose 30% of the data to be in a test sample
2. The remainder is a learning sample
3. Learn the model from the learning sample
4. Estimate its future performance on the test sample

Which model is the best?



- We say that the neural network **overfits** the data
- **Overfitting** occurs when the learning algorithm starts fitting noise.
- (by opposition, the linear model underfits the data)

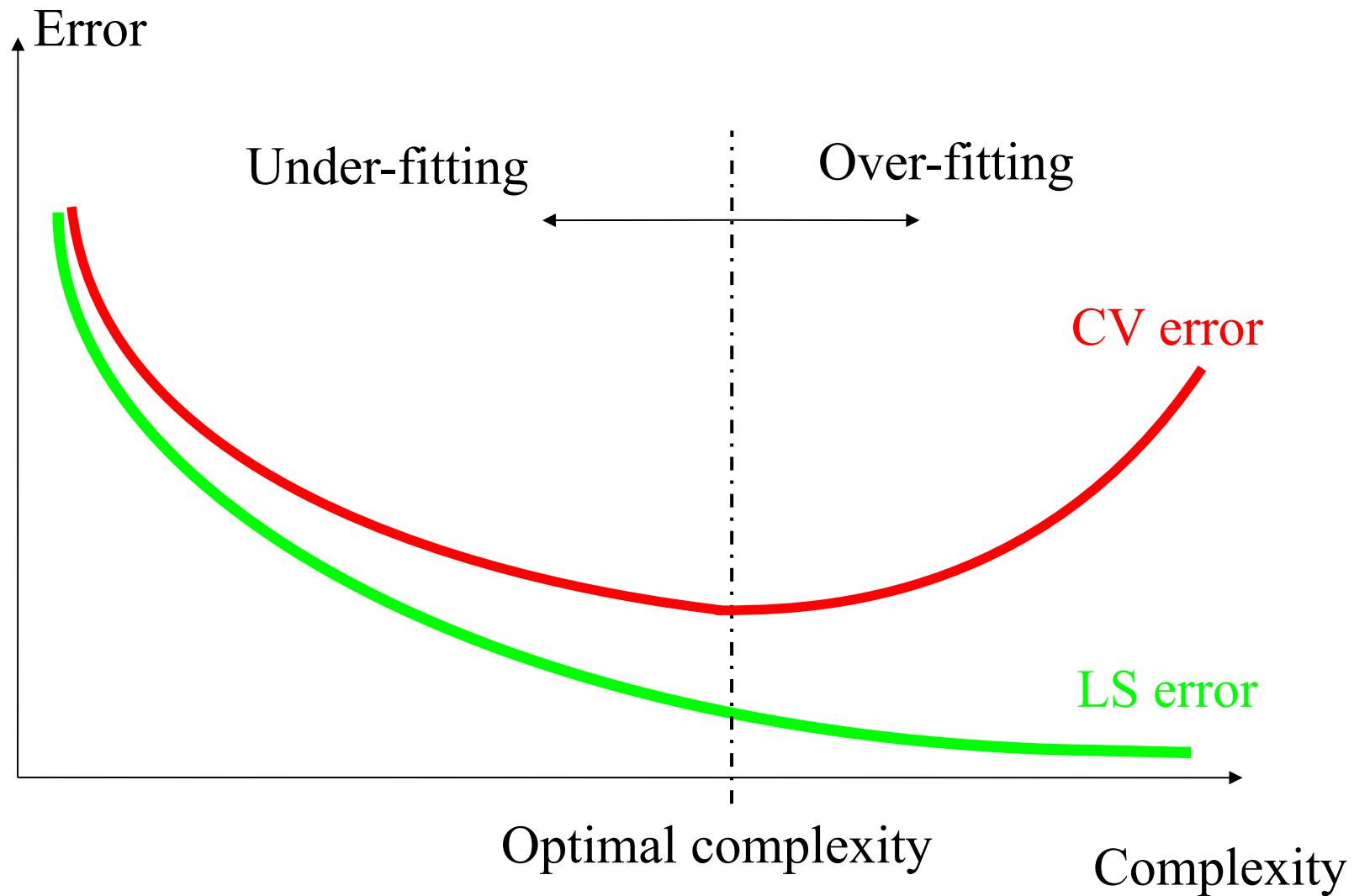
k -fold Cross Validation

- Randomly partition the dataset into k subsets (for example 10)



- For each subset:
 - learn the model on the objects that are not in the subset
 - compute the error rate on the points in the subset
- Report the mean error rate over the k subsets
- When $k=\text{the number of objects}$ \Rightarrow leave-one-out cross validation

CV-based complexity control



Complexity

- Controlling complexity is called **regularization** or **smoothing**
- Complexity can be controlled in several ways
 - The size of the hypothesis space: number of candidate models, range of the parameters...
 - The performance criterion: learning set performance versus parameter range, eg. minimizes
$$\text{Err(LS)} + \lambda C(\text{model})$$
 - The optimization algorithms: number of iterations, nature of the optimization problem (one global optimum versus several local optima)...

Outline

- Introduction
- Model selection, cross-validation, overfitting
- Some supervised learning algorithms
 - k-NN
 - Linear methods
 - Artificial neural networks
 - Support vector machines
 - Decision trees
 - Ensemble methods
- Beyond classification and regression

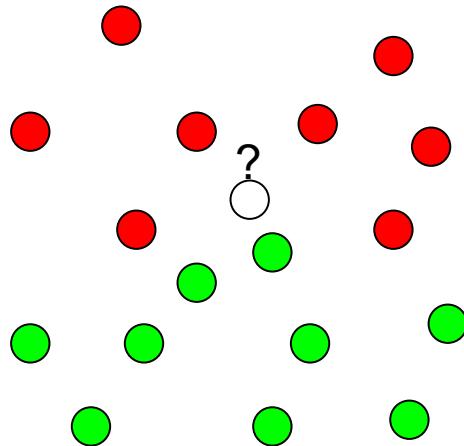
Comparison of learning algorithms

- Three main criteria:
 - Accuracy:
 - Measured by the generalization error (estimated by CV)
 - Efficiency:
 - Computing times and scalability for learning and testing
 - Interpretability:
 - Comprehension brought by the model about the input-output relationship
- Unfortunately, there is usually a trade-off between these criteria

1-Nearest Neighbor (1-NN)

(prototype based method, instance based learning, non-parametric method)

- One of the simplest learning algorithm:
 - outputs as a prediction the output associated to the sample which is the closest to the test object



	M1	M2	Y
1	0.32	0.81	Healthy
2	0.15	0.38	Disease
3	0.39	0.34	Healthy
4	0.62	0.11	Disease
5	0.92	0.43	?

$$d(5,1) = \sqrt{(0.32 - 0.92)^2 + (0.81 - 0.43)^2} = 0.71$$

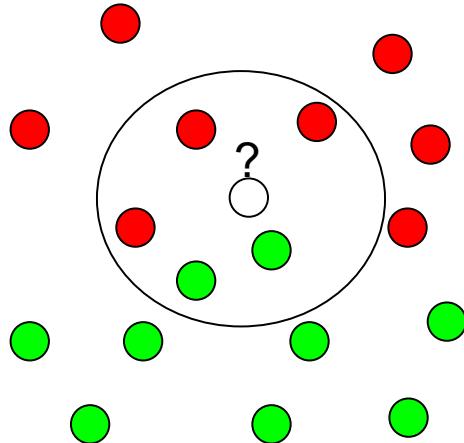
$$d(5,2) = \sqrt{(0.15 - 0.92)^2 + (0.38 - 0.43)^2} = 0.77$$

$$d(5,3) = \sqrt{(0.39 - 0.92)^2 + (0.34 - 0.43)^2} = 0.71$$

$$d(5,4) = \sqrt{(0.62 - 0.92)^2 + (0.43 - 0.43)^2} = 0.44$$

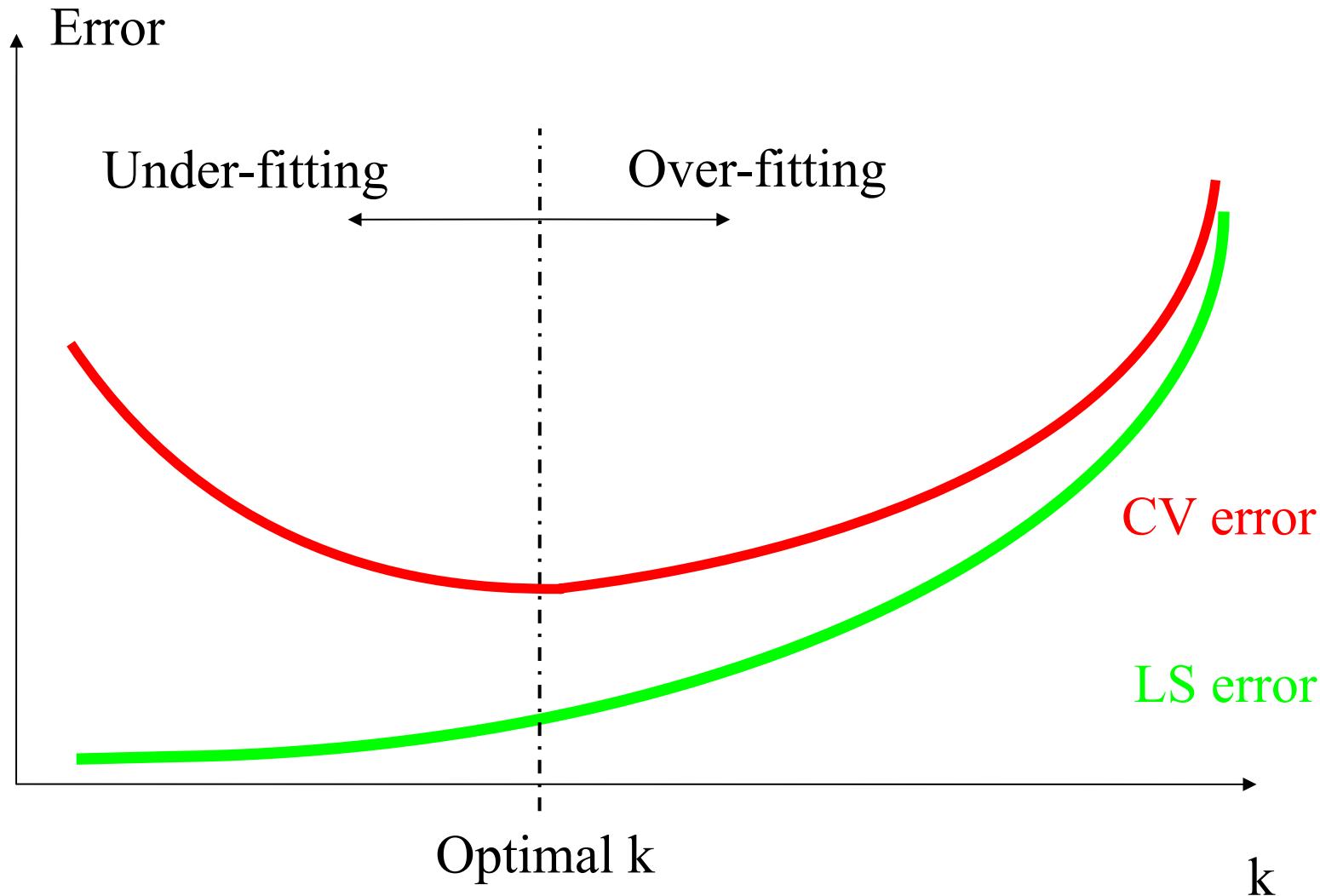
- closest=usually of minimal Euclidian distance

Obvious extension: k-NN



- Find the k nearest neighbors (instead of only the first one) with respect to Euclidian distance
- Output the most frequent class (classification) or the average outputs (regression) among the k neighbors.

Effect of k on the error

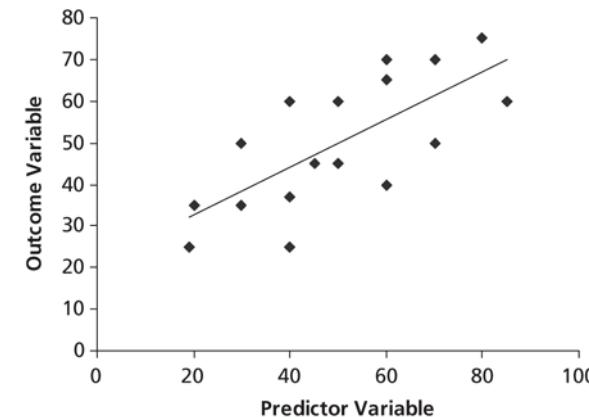
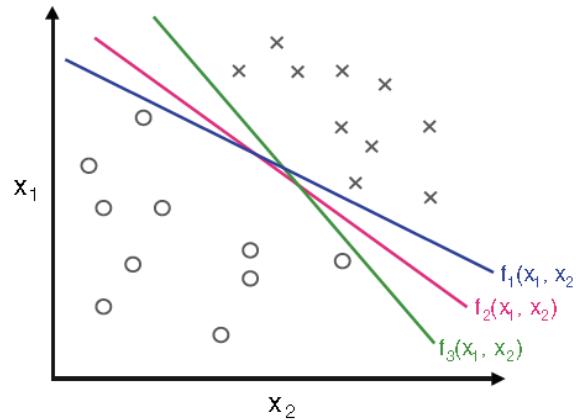


k-NN

- Advantages:
 - very simple
 - can be adapted to any data type by changing the distance measure
- Drawbacks:
 - choosing a good distance measure is a hard problem
 - very sensitive to the presence of noisy variables
 - slow for testing

Linear methods

- Find a model which is a linear combination of the inputs
 - Regression: $y = w_0 + w_1 x_1 + w_2 x_2 + \dots + w_n x_n$
 - Classification: $y = c_1$ if $w_0 + w_1 x_1 + \dots + w_n x_n > 0$, c_2 otherwise

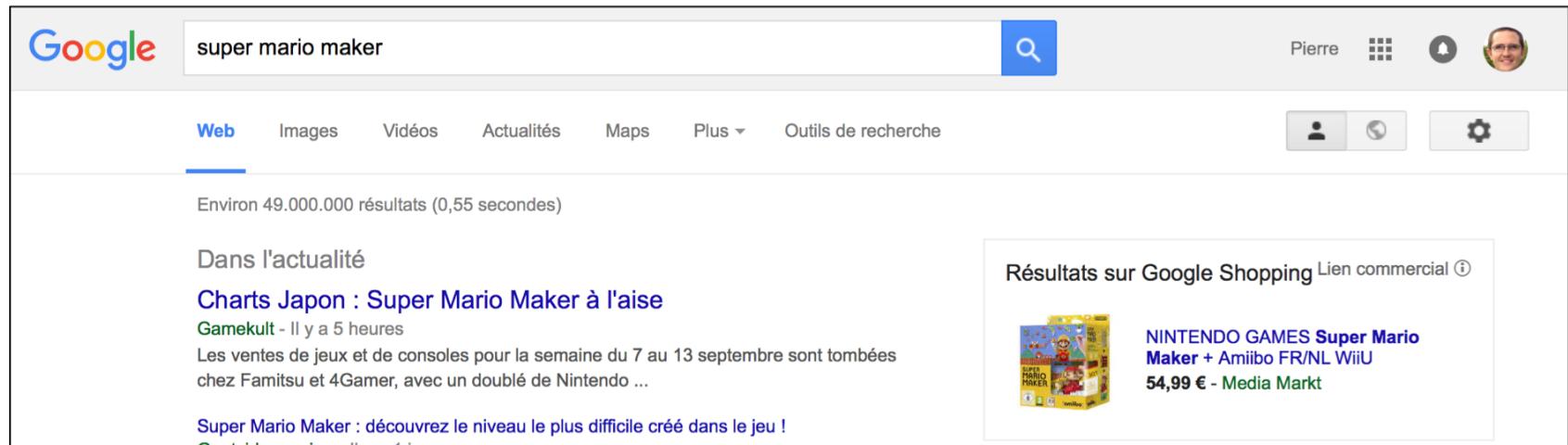


- Several methods exist to find coefficients $w_0, w_1 \dots$ corresponding to different objective functions, optimization algorithms, eg.:
 - Regression: least-square regression, ridge regression, partial least square, support vector regression, LASSO...
 - Classification: linear discriminant analysis, logistic regression, PLS-discriminant analysis, support vector machines...

Linear methods

- Advantages:
 - simple
 - there exist fast and scalable variants
 - provide interpretable models through variable weights (magnitude and sign)
- Drawbacks:
 - often not as accurate as other (non-linear) methods

Application: Click-through rate prediction



- **Goal:** given an ad and a query, predict the probability that the user will click on the ad
- Crucial to decide which ads to show, in which order and at what price
- One of the most profitable applications of ML methods
- Most used method in this domain is ***logistic regression***, a linear classification method that is well adapted to:
 - Massive datasets: billions of samples and billions of features
 - Very sparse features: very few non-zero features per sample

Non-linear extensions

- Generalization of linear methods:

$$y = w_0 + w_1 \phi_1(x) + w_2 \phi_2(x_2) + \dots + w_n \phi_n(x)$$

- Any linear method can be applied (but regularization becomes more important)
- Artificial neural networks (with a single hidden layer):

$$y = g\left(\sum_j W_j g\left(\sum_i w_{i,j} x_i\right)\right)$$

where g is a non linear function (eg. sigmoid)

- (a non linear function of a linear combination of non linear functions of linear combinations of inputs)
- Kernel methods:

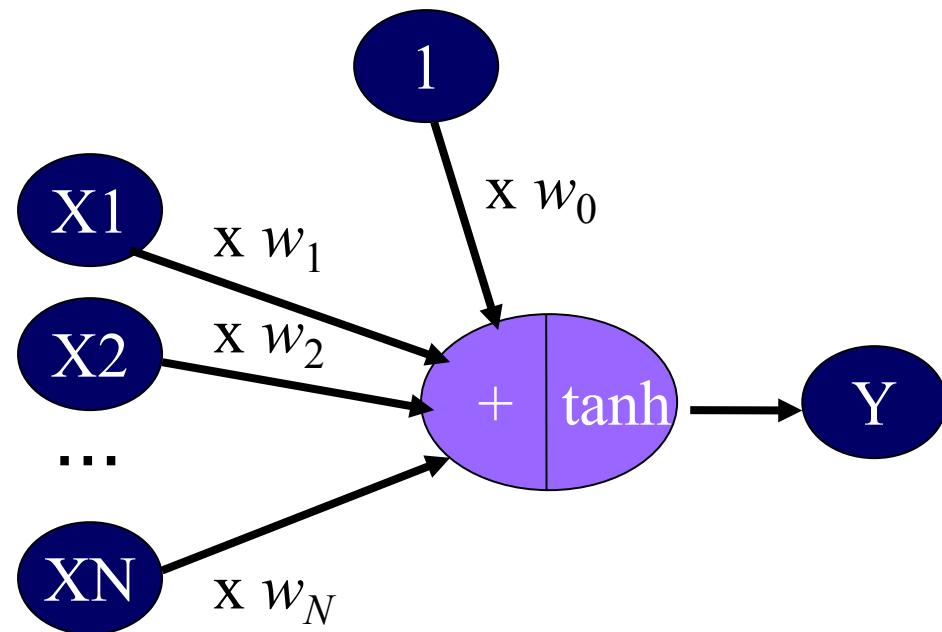
$$y = \sum_i w_i \phi_i(x) \Leftrightarrow y = \sum_j \alpha_j k(x_j, x)$$

where $k(x, x') = \langle \phi(x), \phi(x') \rangle$ is the dot-product in the feature space and j indexes training examples

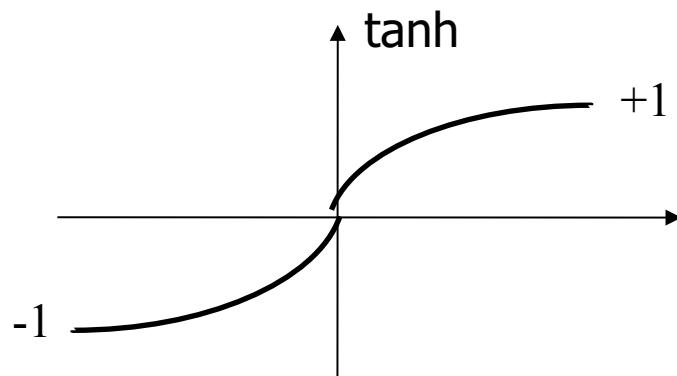
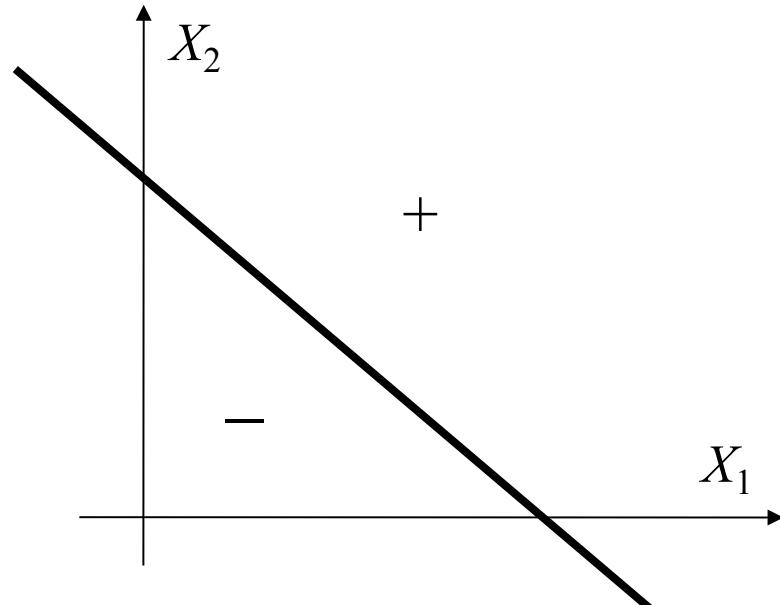
Artificial neural networks

- Supervised learning method initially inspired by the behavior of the human brain
- Consists of the inter-connection of several small units
- Essentially numerical but can handle classification and discrete inputs with appropriate coding
- Introduced in the late 50s, very popular in the 90s, much less popular in the 2000s, recent come-back (new branding: *deep learning*)

Hypothesis space: a single neuron

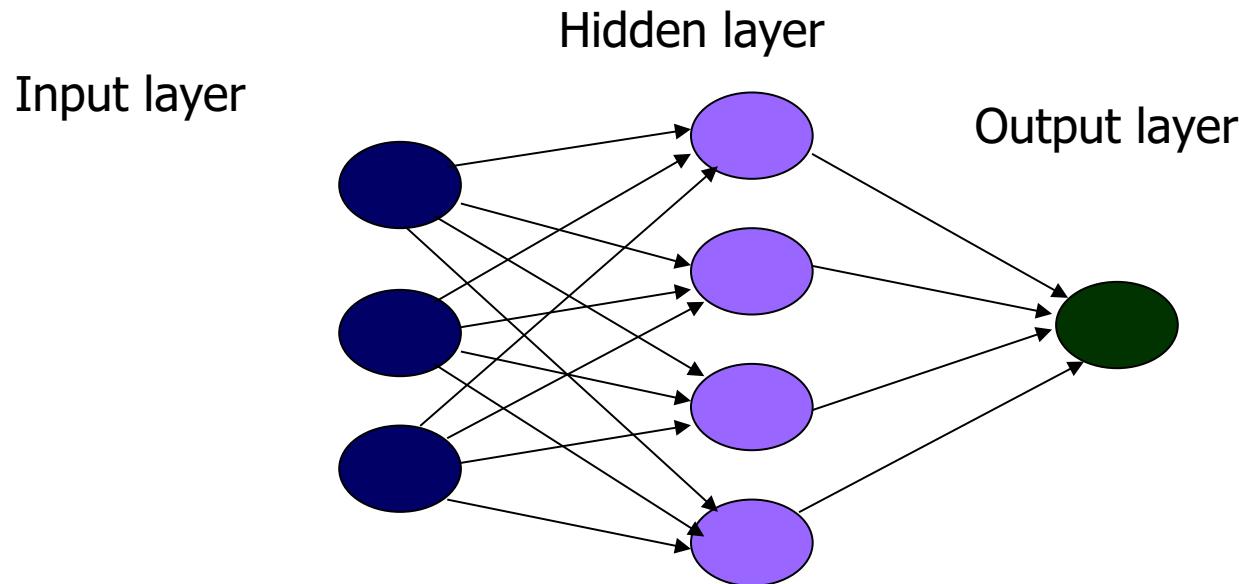


$$Y = \tanh(w_1 * X_1 + w_2 * X_2 + \dots + w_N * X_N + w_0)$$



Hypothesis space: Multi-layers Perceptron

- Inter-connection of several neurons (just like in the human brain)



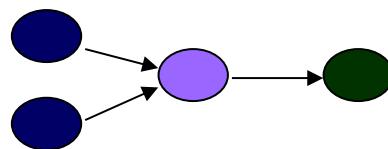
- With a sufficient number of neurons and a sufficient number of layers, a neural network can model any function of the inputs.

Learning

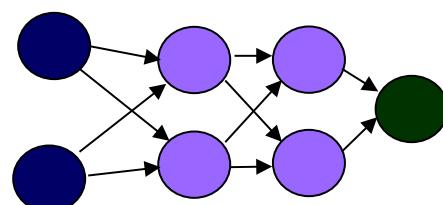
- Choose a structure
- Tune the value of the parameters (connections between neurons) so as to minimize the learning sample error.
 - Non-linear optimization by the back-propagation algorithm.
In practice, quite slow.
- Repeat for different structures
- Select the structure that minimizes CV error

Illustrative example

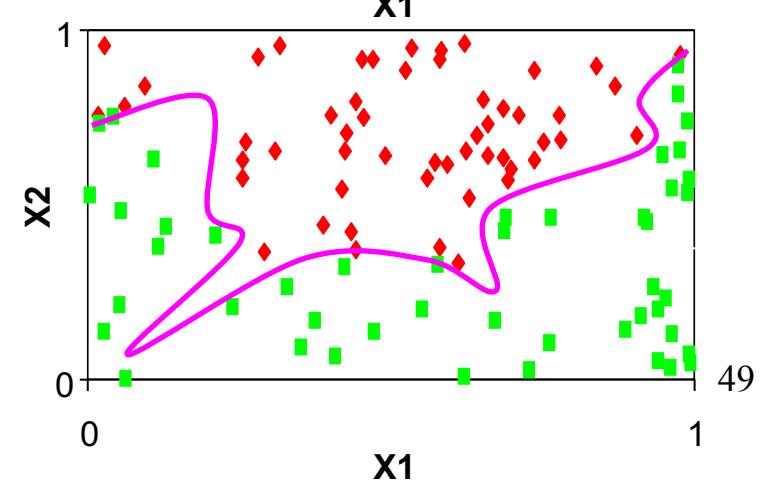
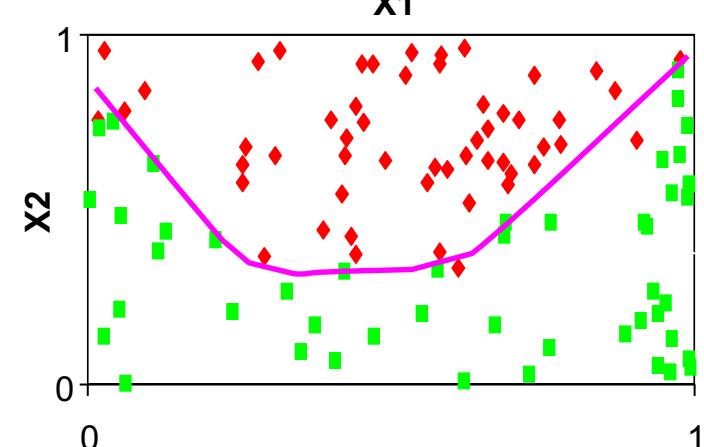
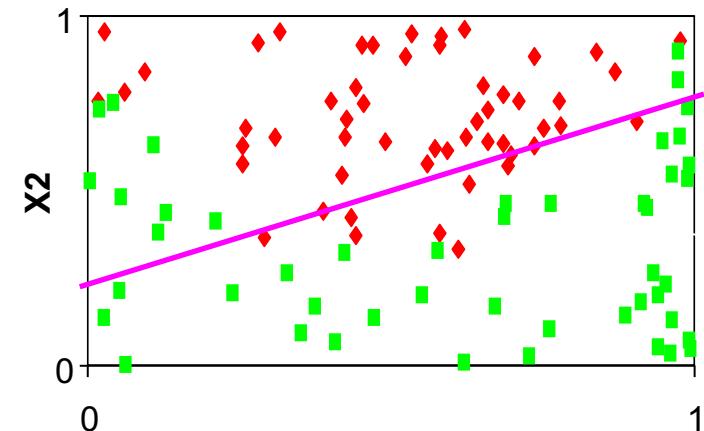
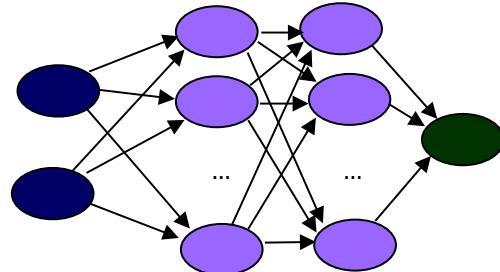
1 neuron



2 + 2 neurons



10 + 10 neurons

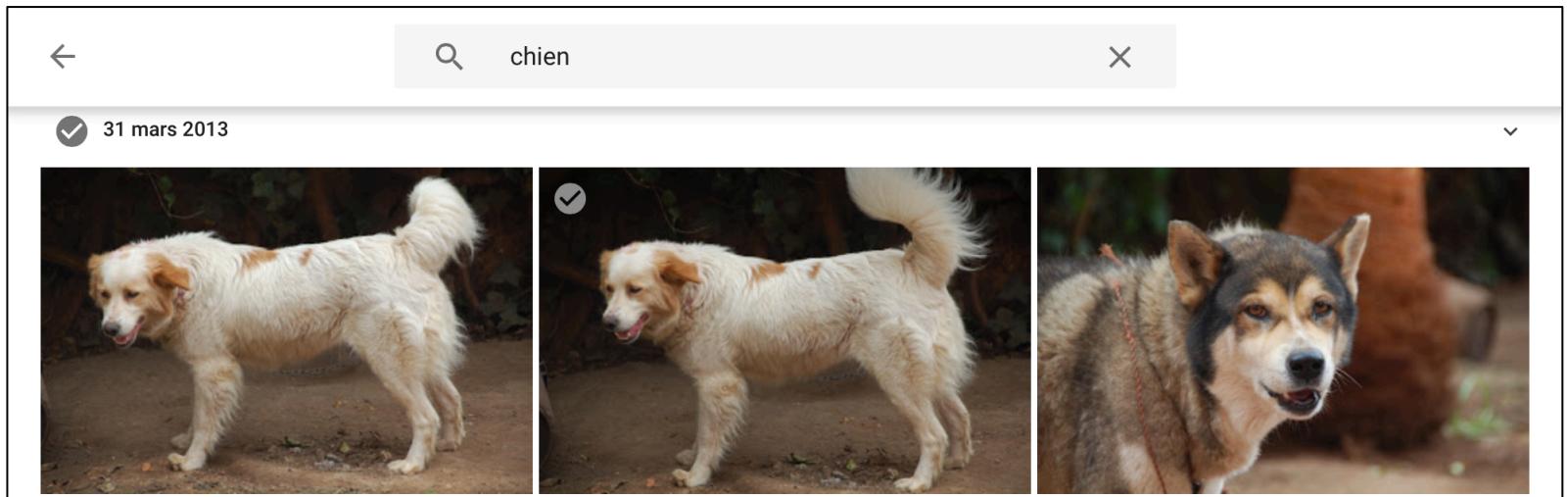


Artificial neural networks

- Advantages:
 - Universal approximators
 - May be very accurate (if the method is well used)
- Drawbacks:
 - The learning phase may be very slow
 - Black-box models, very difficult to interpret
 - Scalability

Application: image labeling

- Goal: Label images with info about their content
- Eg.: Google photos' search engine:



- Most successful methods in this domain are based on huge (*deep*) neural networks. E.g., GoogLeNet:
 - 100 layers (22 with tuned parameters), more than 4M parameters
 - Trained from 1.2M images, with 1000 image categories
 - 6.67% top-5 error rate

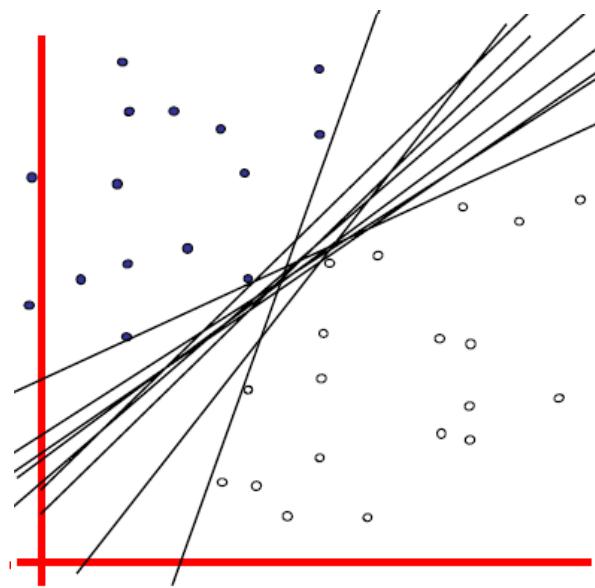
<http://arxiv.org/pdf/1409.4842.pdf>

Support vector machines

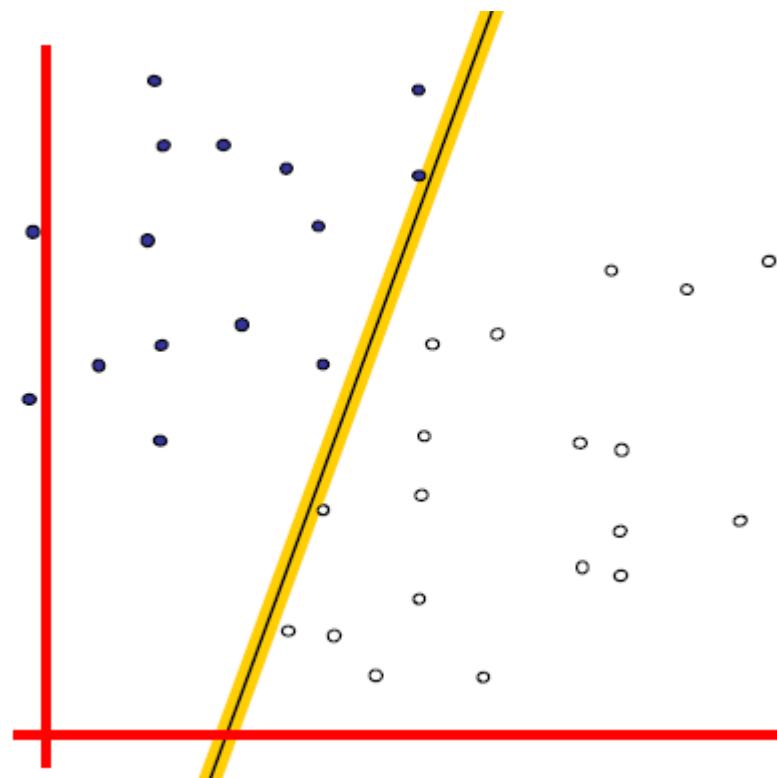
- Recent (mid-90's) and very successful method
- Based on two smart ideas:
 - large margin classifier
 - kernelized input space

Linear classifier

- Where would you place a linear classifier?

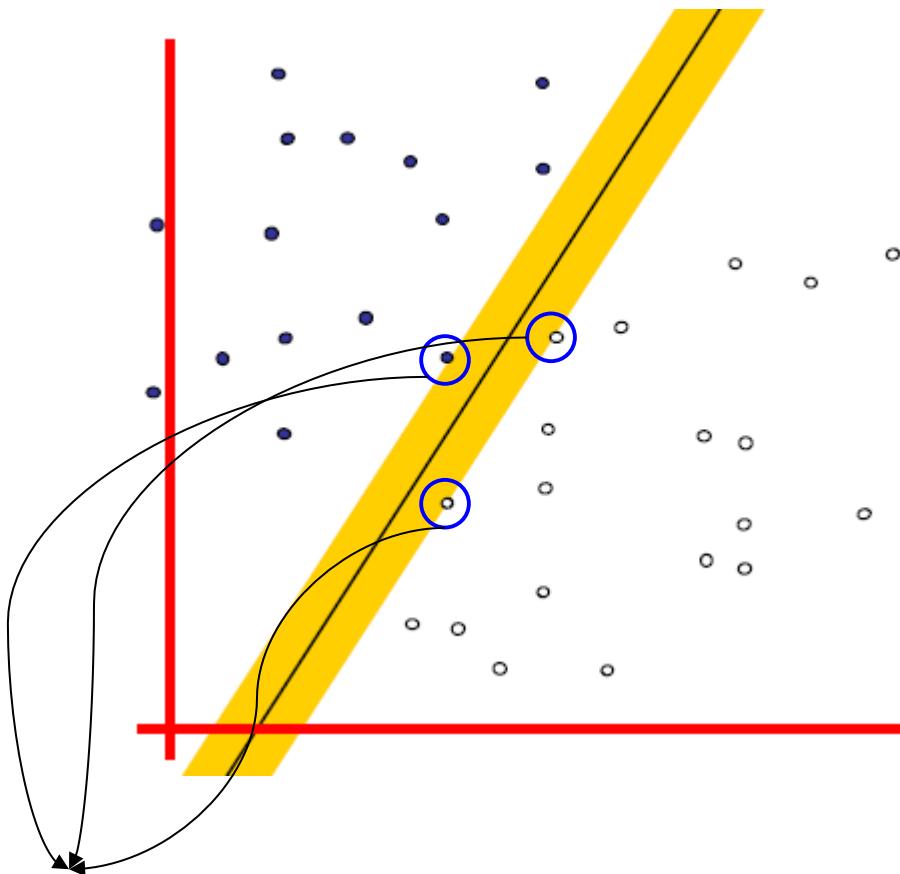


Margin of a linear classifier



- The margin = the width that the boundary could be increased by before hitting a datapoint.

Maximum-margin linear classifier

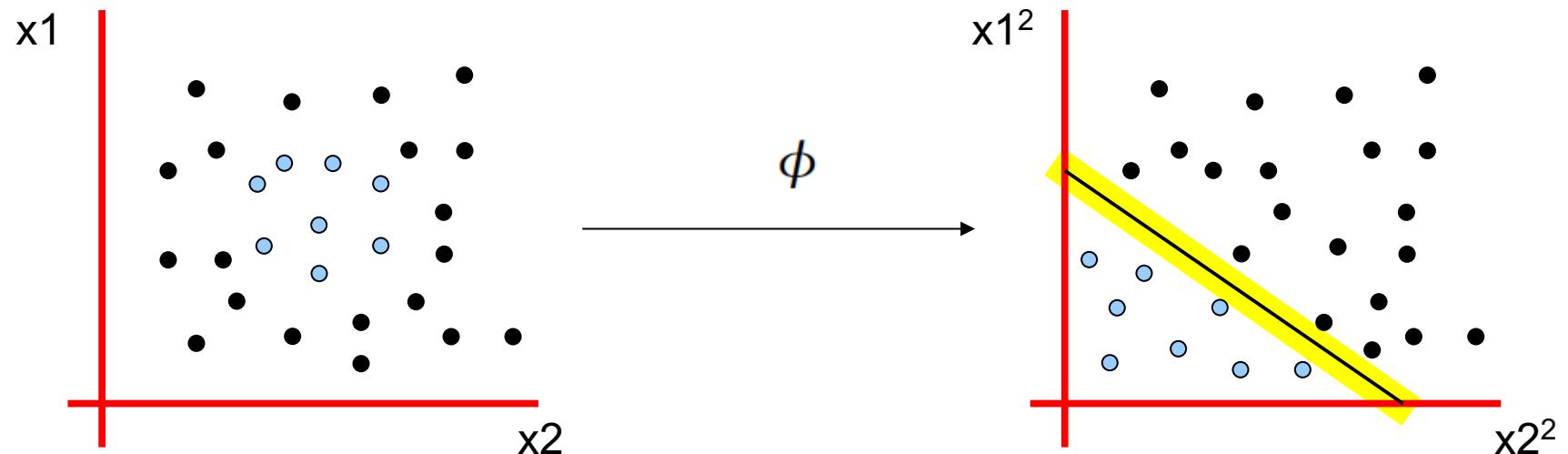


Support vectors: the samples the closest to the hyperplane

- The linear classifier with the maximum margin (= Linear SVM)
- Why ?
 - Intuitively, safest
 - Works very well
 - Theoretical bounds:
 $E(TS) < O(1/\text{margin})$
 - Kernel trick

Non-linear boundary

- What about this problem?



- Solution:
 - map the data into a new feature space where the boundary is linear
 - Find the maximum margin model in this new space

The kernel trick

- Intuitively:
 - You don't need to compute explicitly the mapping φ
 - All you need is a (special) similarity measure between objects (like for the kNN)
 - This similarity measure is called a **kernel**
- Mathematically:
 - The maximum-margin classifier in some feature space can be written only in terms of dot-products in that feature space:

$$k(x, x') = \langle \varphi(x), \varphi(x') \rangle$$

Support vector machines

	X1	X2	Y
1	0.49	0.94	C1
2	0.86	0.59	C2
3	0.6	0.79	C2
4	0.83	0.66	C1
5	0.63	0.27	C1
6	-0.76	0.47	C2

kernel matrix

	1	2	3	4	5	6
1	1	0.14	0.96	0.17	0.01	0.24
2	0.14	1	0.02	0.17	0.22	0.67
3	0.96	0.02	1	0.15	0.27	0.07
4	0.17	0.7	0.15	1	0.37	0.55
5	0.01	0.22	0.27	0.37	1	-0.25
6	0.24	0.67	0.07	0.55	-0.25	1

	X1	Y
1	ACGCTCTATAG	C1
2	ACTCGCTTAGA	C2
3	GTCTCTGAGAG	C2
4	CGCTAGCGTCG	C1
5	CGATCAGCAGC	C1
6	GCTCGCGCTCG	C2

Class labels

	Y
1	C1
2	C2
3	C2
4	C1
5	C1
6	C2

SVM
algorithm

Classification
model

Support vector machines

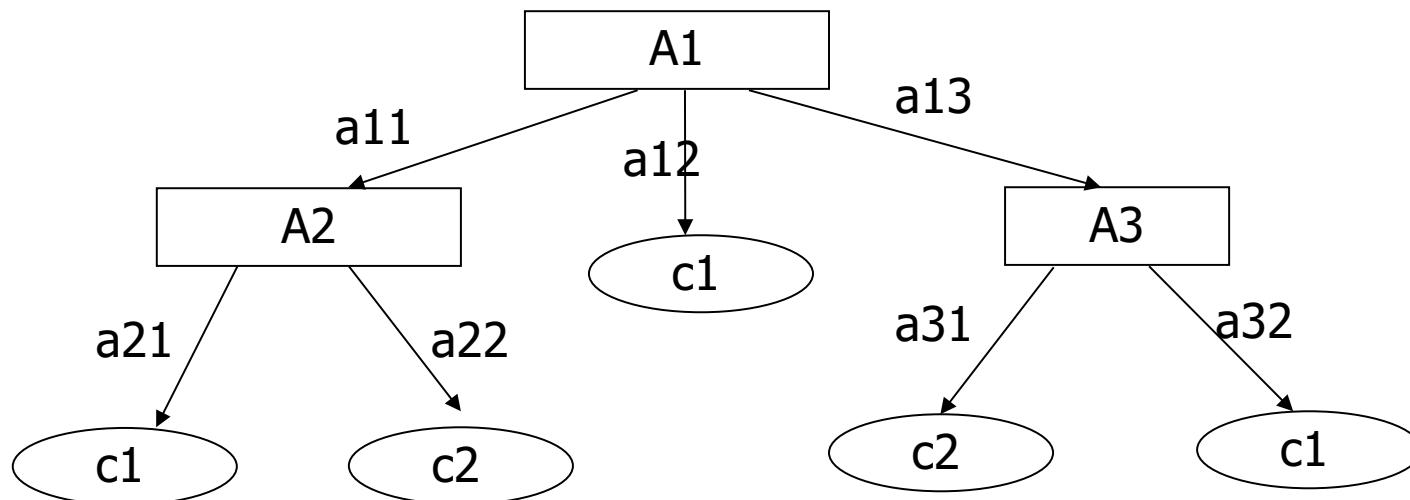
- Advantages:
 - State-of-the-art accuracy on many problems
 - Can handle any data types by changing the kernel (many applications on sequences, texts, graphs...)
- Drawbacks:
 - Tuning the method parameter is very crucial to get good results and somewhat tricky
 - Black-box models, not easy to interpret

Decision (classification) trees

- A learning algorithm that can handle:
 - Classification problems (binary or multi-valued)
 - Attributes may be discrete (binary or multi-valued) or continuous.
- Classification trees were invented at least twice (in mid-80's):
 - By statisticians: CART (Breiman et al.)
 - By the AI community: ID3, C4.5 (Quinlan et al.)

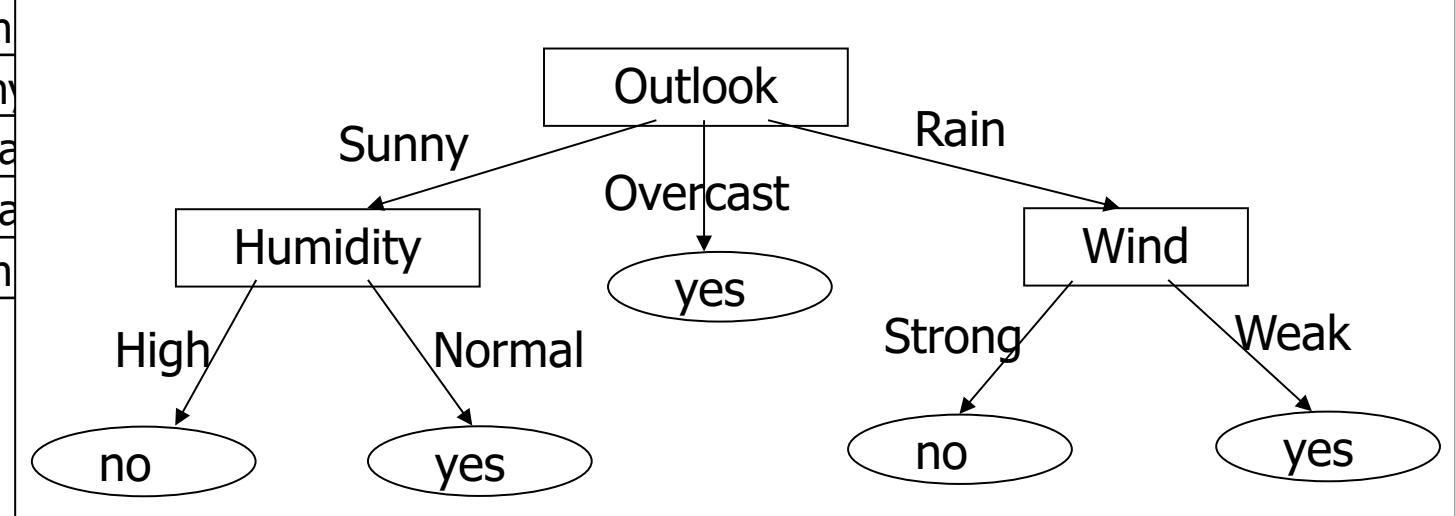
Decision trees

- A decision tree is a tree where:
 - Each interior node tests an attribute
 - Each branch corresponds to an attribute value
 - Each leaf node is labeled with a class



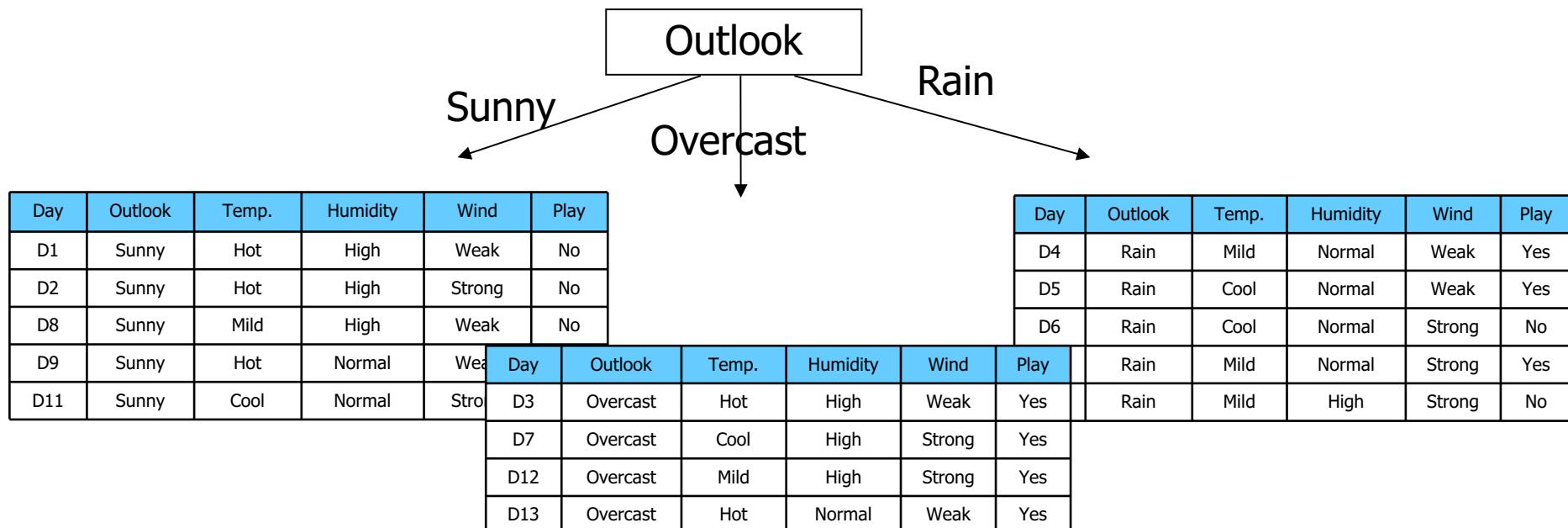
A simple database: playtennis

Day	Outlook	Temperature	Humidity	Wind	Play Tennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	Normal	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	High	Strong	Yes
D8	Sunny	Mild	Normal	Weak	No
D9	Sunny	Hot	Normal	Weak	Yes
D10	Rain				
D11	Sunny				
D12	Overcast				
D13	Overcast				
D14	Rain				

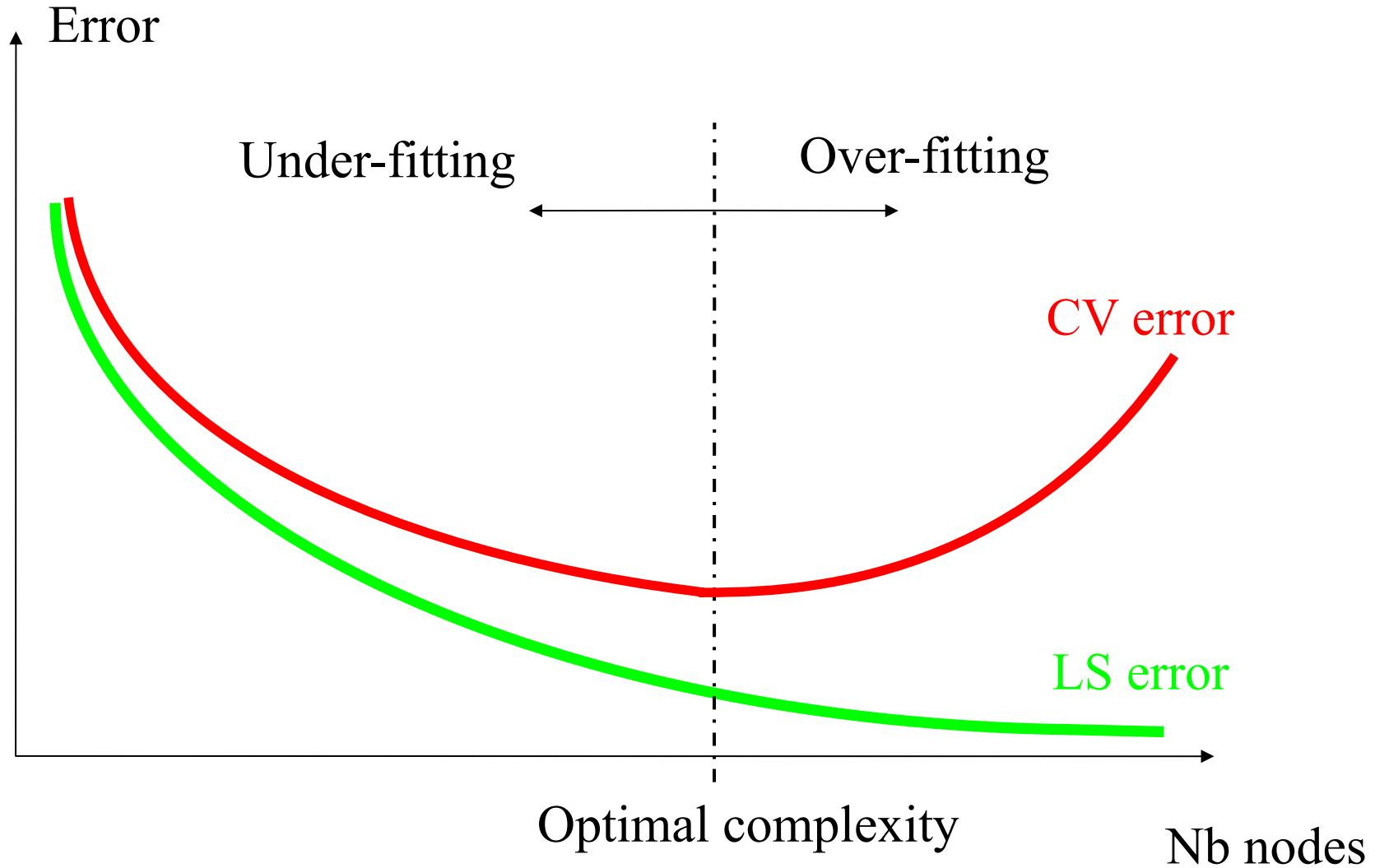


Top-down induction of DTs

- Choose « best » attribute
- Split the learning sample
- Proceed recursively until each object is correctly classified



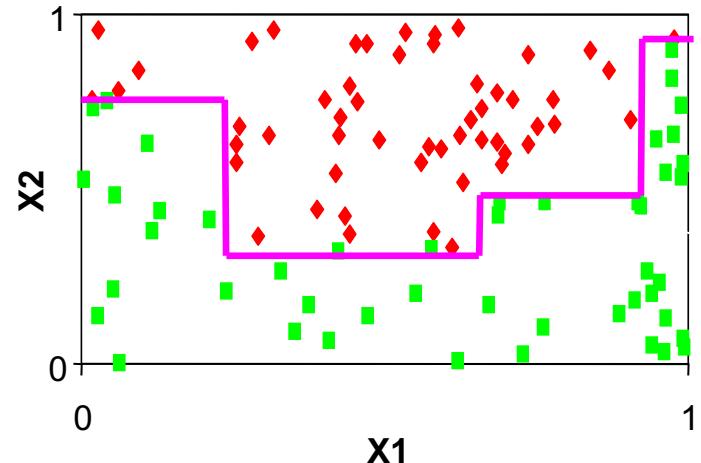
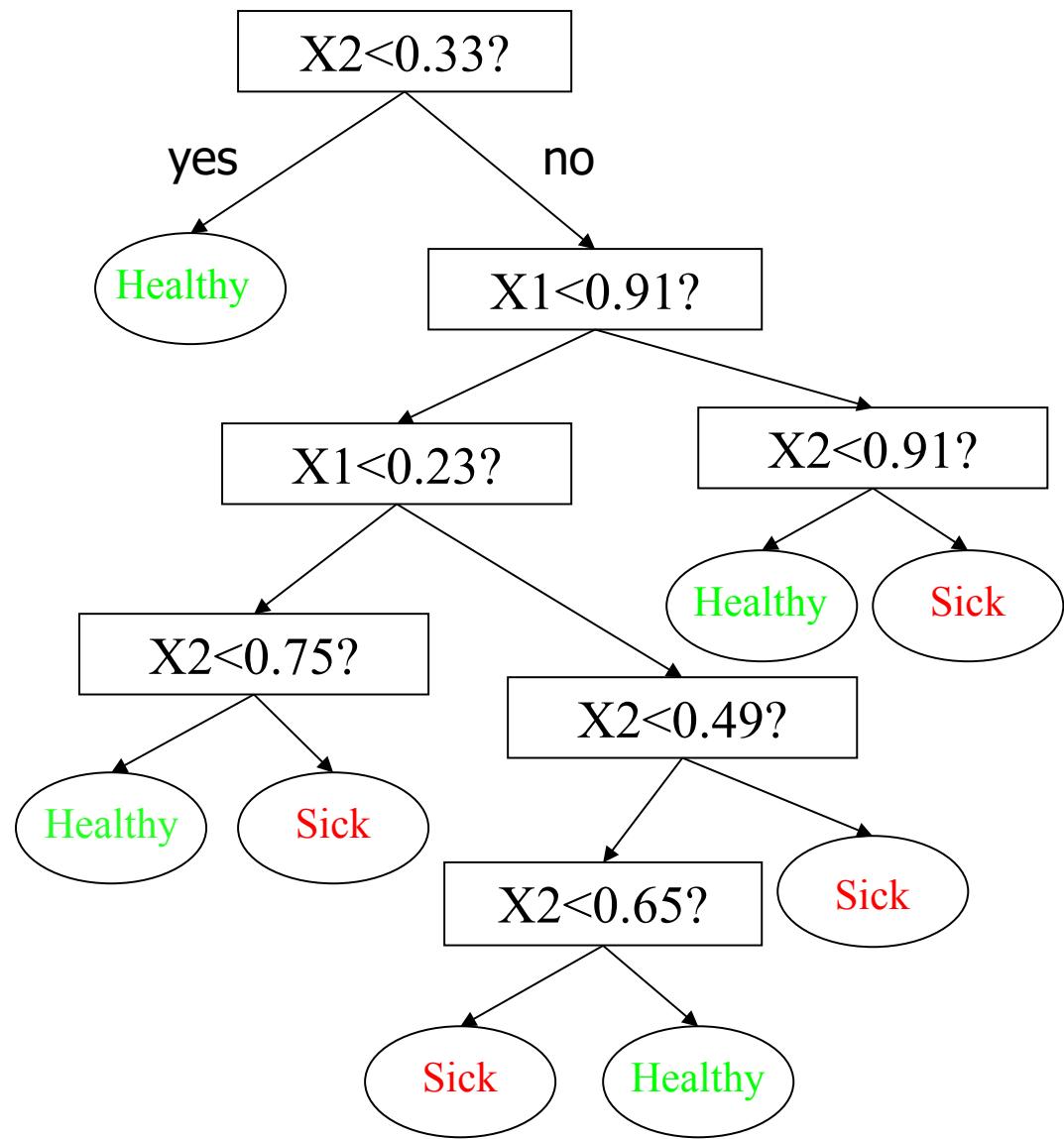
Effect of number of nodes on error



How can we avoid overfitting?

- Pre-pruning: stop growing the tree earlier, before it reaches the point where it perfectly classifies the learning sample
- Post-pruning: allow the tree to overfit and then post-prune the tree
- Ensemble methods (later)

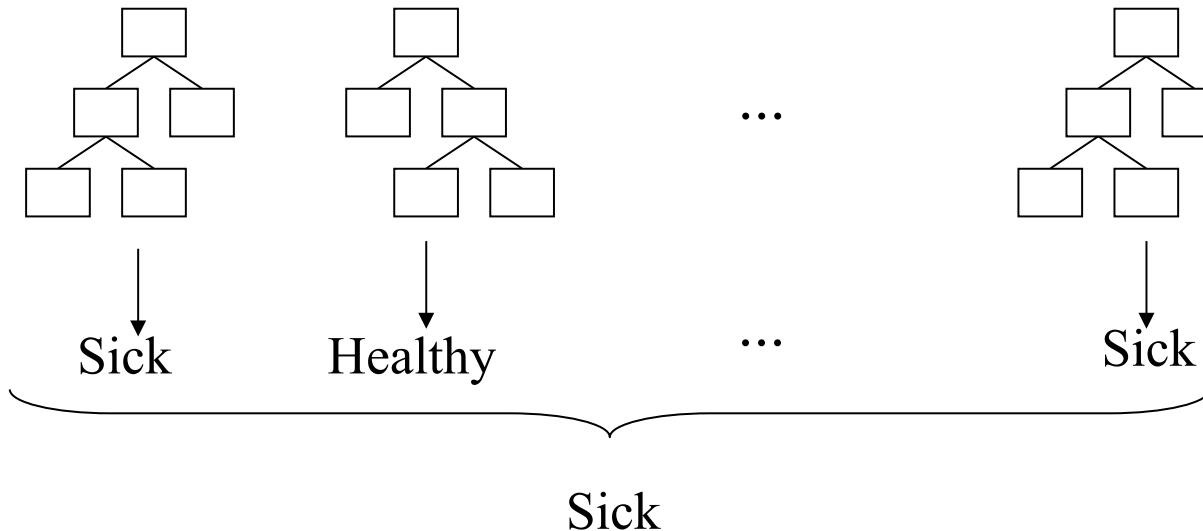
Illustrative example



Decision and regression trees

- Advantages:
 - very fast and scalable method (able to handle a very large number of inputs and objects)
 - provide directly interpretable models and give an idea of the relevance of attributes
- Drawbacks:
 - high variance (more on this later)
 - often not as accurate as other methods

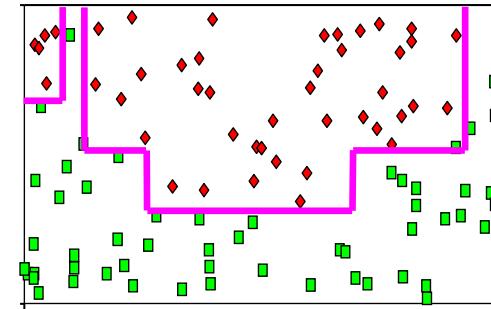
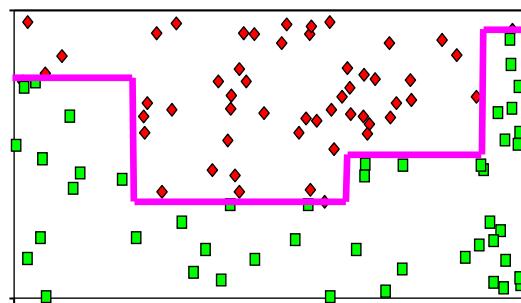
Ensemble methods



- Combine the predictions of several models built with a learning algorithm. Often improve very much accuracy.
- Often used in combination with decision trees for efficiency reasons
- Examples of algorithms: Bagging, Random Forests, Boosting...

Bagging: motivation

- Different learning samples yield different models, especially when the learning algorithm overfits the data



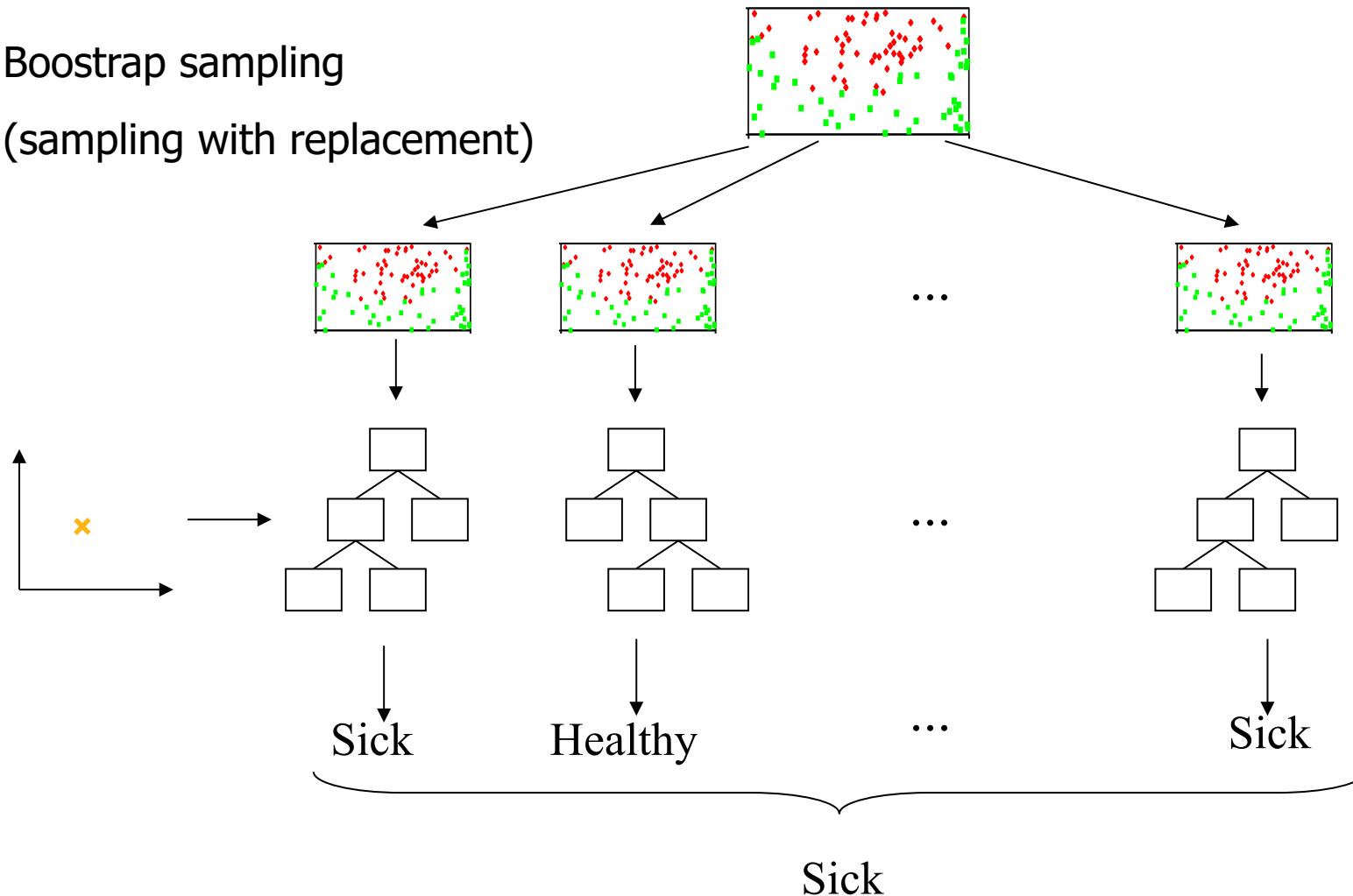
As there is only one optimal model, this *variance* is source of error

- Solution: aggregate several models to obtain a more stable one



Bagging: bootstrap aggregating

Bootstrap sampling
(sampling with replacement)



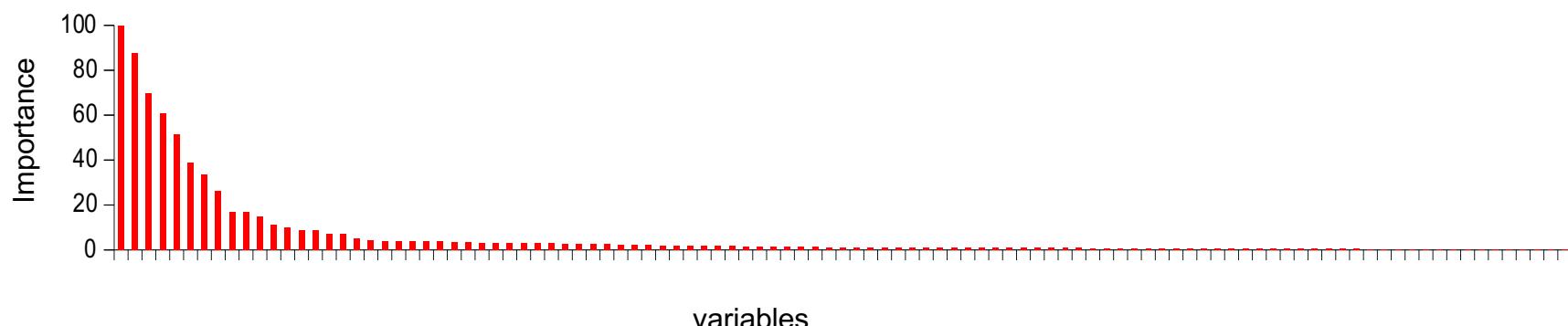
Note: the more models, the better.

Example on microarray data

- 72 patients, 7129 gene expressions, 2 classes of Leukemia (ALL and AML) (Golub et al., Science, 1999)
- Leave-one-out error with several variants

Method	Error
1 decision tree	22.2% (16/72)
Random forests ($k=85, T=500$)	9.7% (7/72)
Extra-trees ($s_{th}=0.5, T=500$)	5.5% (4/72)
Adaboost (1 test node, $T=500$)	1.4% (1/72)

- Variable importance with boosting



Application: Kinect

- Ensemble of randomized decision trees are used in Microsoft's Xbox Kinect for body part labeling:



- Each sample corresponds to a single pixel and is described by depth differences between neighbor pixels
- Final model is implemented on GPU to get very fast predictions (200 frames per second)

Method comparison

Method	Accuracy	Efficiency	Interpretability	Ease of use
kNN	++	+	++	++
DT	+	+++	+++	+++
Linear	++	++++	++	+++
Ensemble	+++	+++	++	+++
ANN	++++	++	+	++
SVM	+++	+	+	+

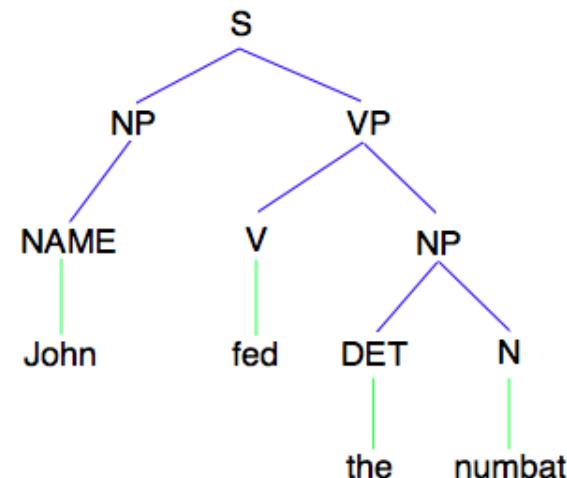
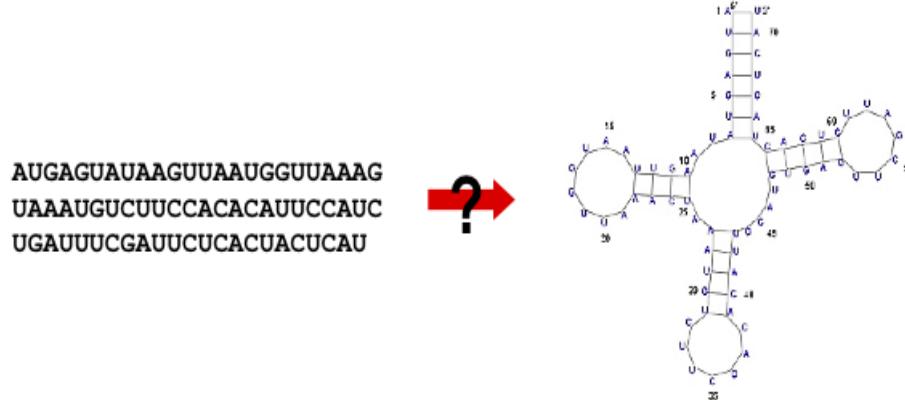
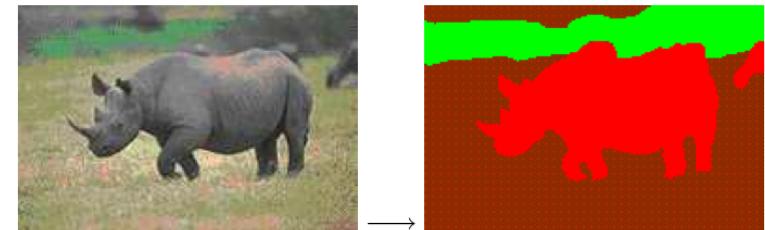
- Note:
 - The relative importance of the criteria depends on the specific application
 - These are only general trends. Eg., in terms of accuracy, no algorithm is always better or worse than all others.

Outline

- Introduction
- Supervised Learning
 - Introduction
 - Model selection, cross-validation, overfitting
 - Some supervised learning algorithms
 - Beyond classification and regression
- Other learning protocols/frameworks

Beyond classification and regression

- All supervised learning problems can not be turned into standard classification or regression problems
- Examples:
 - Graph predictions
 - Sequence labeling
 - image segmentation



Structured output approaches

- Decomposition:
 - Reduce the problem to several simpler classification or regression problems by decomposing the output
 - Not always possible and does not take into account dependencies between sub-outputs
- Multiple/kernel output methods:
 - Extend regression methods to handle an output space endowed with a kernel
 - This can be done with regression trees or ridge regression for example
- Large margin methods
 - Use SVM-based approaches to learn a model that scores directly input-output pairs:

$$y = \arg \max_{y'} \sum_i w_i \phi_i(x, y')$$

Outline

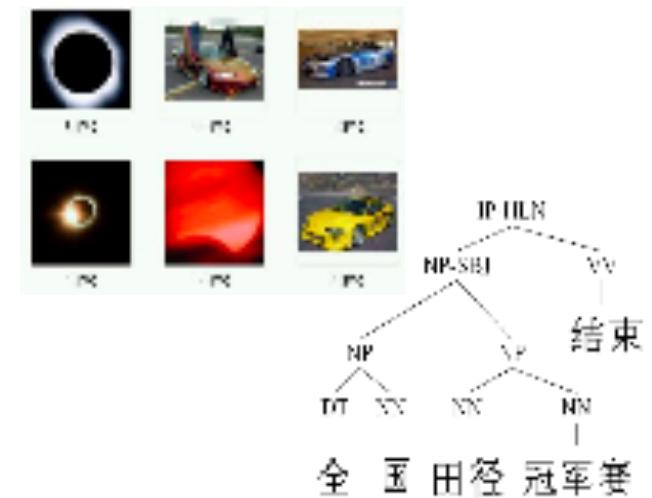
- Introduction
- Supervised learning
- Other learning protocols/frameworks
 - On-line versus batch learning
 - Semi-supervised learning
 - Transductive learning
 - Active learning
 - Reinforcement learning
 - Unsupervised learning

On-line versus batch learning

- **Batch learning:** all learning examples are supposed to be accessible (in memory) for training the model
- **On-line learning:** examples are treated one by one (mini-batch learning: examples are treated in blocs).
- Applications:
 - Big data: training set is too large to be stored in memory
 - Concept drift: allows to continuously adapt the model to dynamical change of the system
 - Learning with streaming data
- There exist online implementations of most machine learning methods

Labeled versus unlabeled data

- **Unlabeled data**=input-output pairs without output value
- In many settings, unlabeled data is cheap but labeled data can be hard to get
 - labels may require human experts
 - human annotation is expensive, slow, unreliable
 - labels may require special devices
- Examples:
 - Biomedical domain
 - Speech analysis
 - Natural language parsing
 - Image categorization/segmentation
 - Network measurement

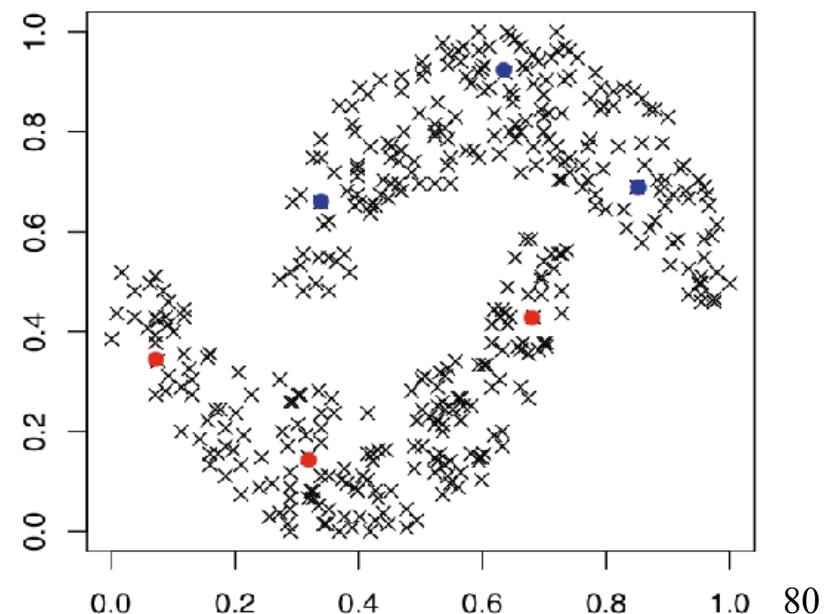
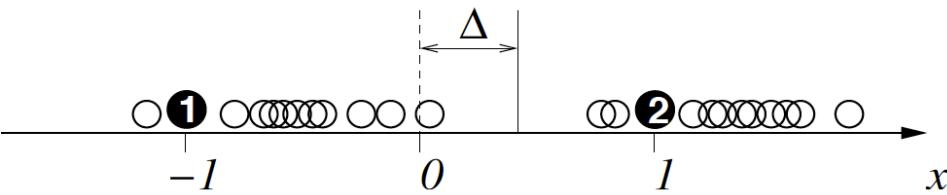
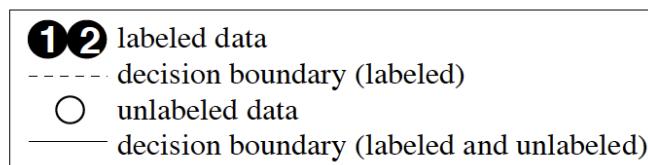


Semi-supervised learning

- Goal: exploit both labeled and unlabeled data to build better models than using each one alone

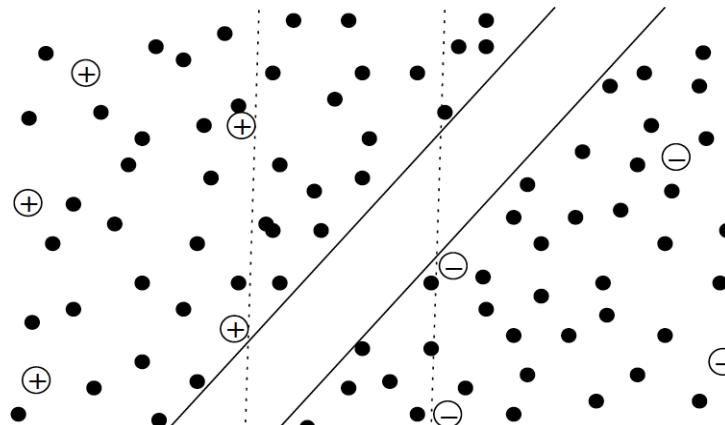
	A1	A2	A3	A4	Y
labeled data	0.01	0.37	T	0.54	Healthy
	-2.3	-1.2	F	0.37	Disease
	0.69	-0.78	F	0.63	Healthy
unlabeled data	-0.56	-0.89	T	-0.42	
	-0.85	0.62	F	-0.05	
	-0.17	0.09	T	0.29	
test data	-0.09	0.3	F	0.17	?

- Why would it improve?



Some approaches

- Self-training
 - Iteratively label some unlabeled examples with a model learned from the previously labeled examples
- Semi-supervised SVM (S3VM)
 - Enumerate all possible labeling of the unlabeled examples
 - Learn an SVM for each labeling
 - Pick the one with the largest margin

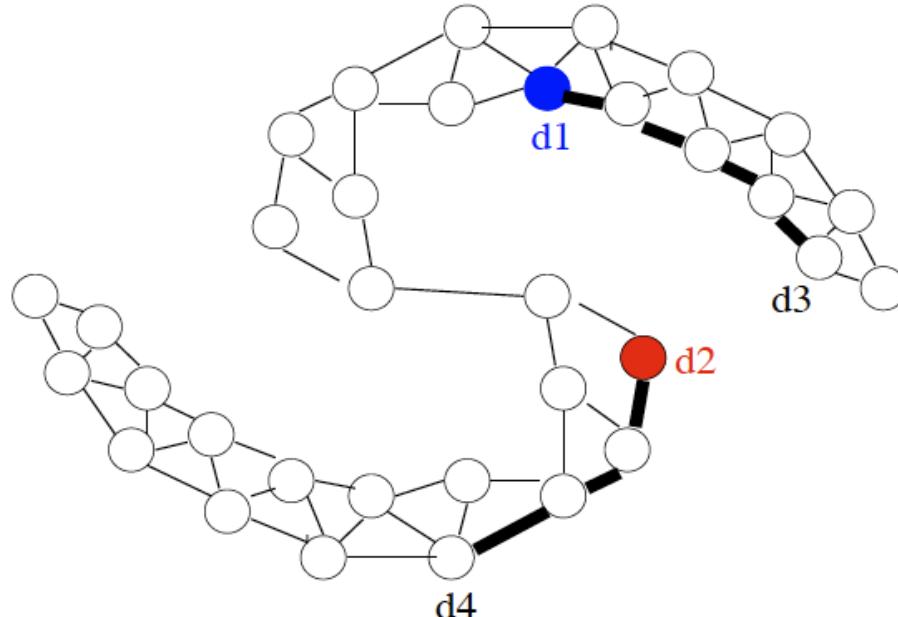


Transductive learning

- Like supervised learning but we have access to the test data from the beginning and we want to exploit it
- We don't want a model, only compute predictions for the unlabeled data
- Simple solution:
 - Apply semi-supervised learning techniques using the test data as unlabeled data to get a model
 - Use the resulting model to make predictions on the test data
- There exist also specific algorithms that avoid building a model

Some approaches

- Graph-based algorithms
 - Build a graph over the (labeled and unlabeled) examples (from the inputs)
 - Learn a model that predicts well labeled examples and is smooth over the graph

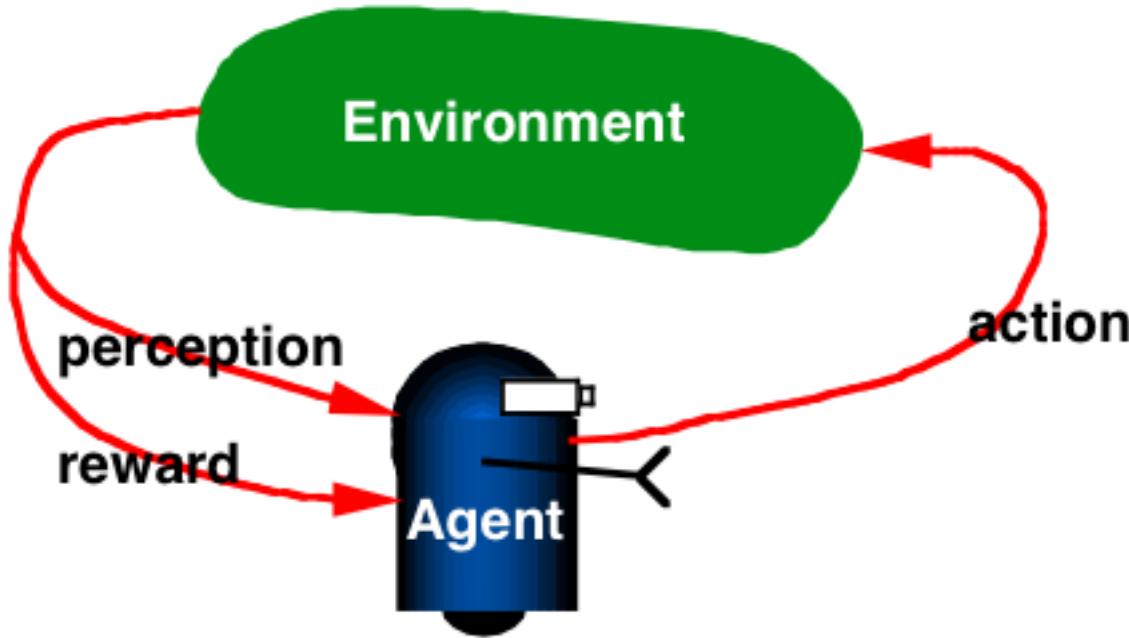


Active learning

- Goal:
 - Given unlabeled data, find (adaptively) the examples to label in order to learn an accurate model
 - The hope is to reduce the number of labeled instances with respect to the standard batch SL
- Usually, in an online setting:
 - Choose the k “*best*” unlabeled examples
 - Determine their labels
 - Update the model and iterate
- Algorithms differ in the way the “*best*” unlabeled examples are selected
 - Example: choose the k examples for which the model predictions are the most uncertain

Reinforcement learning (see INFO8003)

Learning from interactions



$$s_0 \xrightarrow{a_0} s_1 \xrightarrow{r_0} s_2 \xrightarrow{a_1} s_3 \xrightarrow{r_1} s_4 \xrightarrow{a_2} \dots$$

Goal: learn to choose sequence of actions (= policy) that maximizes $r_0 + \gamma r_1 + \gamma^2 r_2 + \dots$, where $0 \leq \gamma < 1$

RL approaches

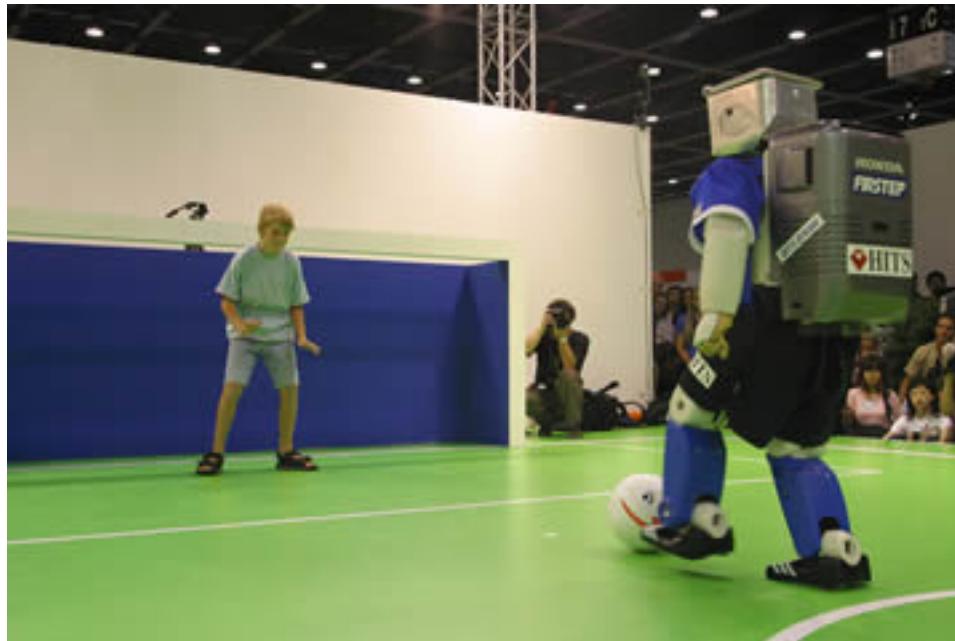
- System is usually modeled by
 - state transition probabilities $P(s_{t+1}|s_t, a_t)$
 - reward probabilities $P(r_{t+1}|s_t, a_t)$
(= Markov Decision Process)
- Model of the dynamics and reward is known \Rightarrow try to compute optimal policy by dynamic programming
- Model is unknown
 - Model-based approaches \Rightarrow first learn a model of the dynamics and then derive an optimal policy from it (DP)
 - Model-free approaches \Rightarrow learn directly a policy from the observed system trajectories

Examples of applications

- Robocup Soccer Teams (Stone & Veloso, Riedmiller et al.)
- Inventory Management (Van Roy, Bertsekas, Lee & Tsitsiklis)
- Dynamic Channel Assignment, Routing (Singh & Bertsekas, Nie & Haykin, Boyan & Littman)
- Elevator Control (Crites & Barto)
- Many Robots: navigation, bi-pedal walking, grasping, switching between skills...
- Games: TD-Gammon and Jellyfish (Tesauro, Dahl), GO, video games...

Robocup

- Goal: by the year 2050, develop a team of fully autonomous humanoid robots that can win against the human world soccer champion team.



<http://www.robocup.org>

<http://www.youtube.com/watch?v=xqWELifR2P0>

Autonomous helicopter



<http://heli.stanford.edu/>

AlphaGo (Google DeepMind)



- Supervised learning from strong amateur games using (deep) neural networks (to predict next move and game winner)
- Improvement using reinforcement learning by playing against versions of itself

Unsupervised learning

- Unsupervised learning tries to find any regularities in the data without guidance about inputs and outputs

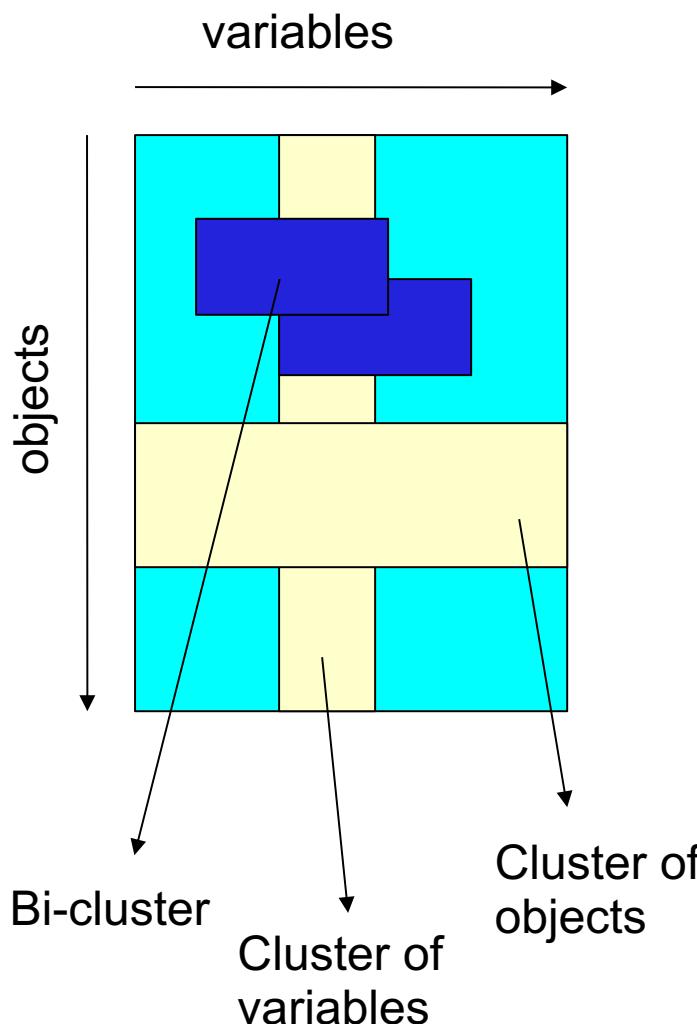
A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15	A16	A17	A18	A19
-0.27	-0.15	-0.14	0.91	-0.17	0.26	-0.48	-0.1	-0.53	-0.65	0.23	0.22	0.98	0.57	0.02	-0.55	-0.32	0.28	-0.33
-2.3	-1.2	-4.5	-0.01	-0.83	0.66	0.55	0.27	-0.65	0.39	-1.3	-0.2	-3.5	0.4	0.21	-0.87	0.64	0.6	-0.29
0.41	0.77	-0.44	0	0.03	-0.82	0.17	0.54	-0.04	0.6	0.41	0.66	-0.27	-0.86	-0.92	0	0.48	0.74	0.49
0.28	-0.71	-0.82	0.27	-0.21	-0.9	0.61	-0.57	0.44	0.21	0.97	-0.27	0.74	0.2	-0.16	0.7	0.79	0.59	-0.33
-0.28	0.48	0.79	-0.14	0.8	0.28	0.75	0.26	0.3	-0.78	-0.72	0.94	-0.78	0.48	0.26	0.83	-0.88	-0.59	0.71
0.01	0.36	0.03	0.03	0.59	-0.5	0.4	-0.88	-0.53	0.95	0.15	0.31	0.06	0.37	0.66	-0.34	0.79	-0.12	0.49
-0.53	-0.8	-0.64	-0.93	-0.51	0.28	0.25	0.01	-0.94	0.96	0.25	-0.12	0.27	-0.72	-0.77	-0.31	0.44	0.58	-0.86
0.04	0.94	-0.92	-0.38	-0.07	0.98	0.1	0.19	-0.57	-0.69	-0.23	0.05	0.13	-0.28	0.98	-0.08	-0.3	-0.84	0.47
-0.88	-0.73	-0.4	0.58	0.24	0.08	-0.2	0.42	-0.61	-0.13	-0.47	-0.36	-0.37	0.95	-0.31	0.25	0.55	0.52	-0.66
-0.56	0.97	-0.93	0.91	0.36	-0.14	-0.9	0.65	0.41	-0.12	0.35	0.21	0.22	0.73	0.68	-0.65	-0.4	0.91	-0.64

- Are there interesting groups of variables or samples? outliers? What are the dependencies between variables?

Unsupervised learning methods

- Many families of tasks/problems exist, among which:
 - Clustering: try to find natural groups of samples/variables
 - eg: k-means, hierarchical clustering
 - Dimensionality reduction: project the data from a high-dimensional space down to a small number of dimensions
 - eg: principal/independent component analysis, MDS
 - Density estimation: determine the distribution of data within the input space
 - eg: bayesian networks, mixture models.

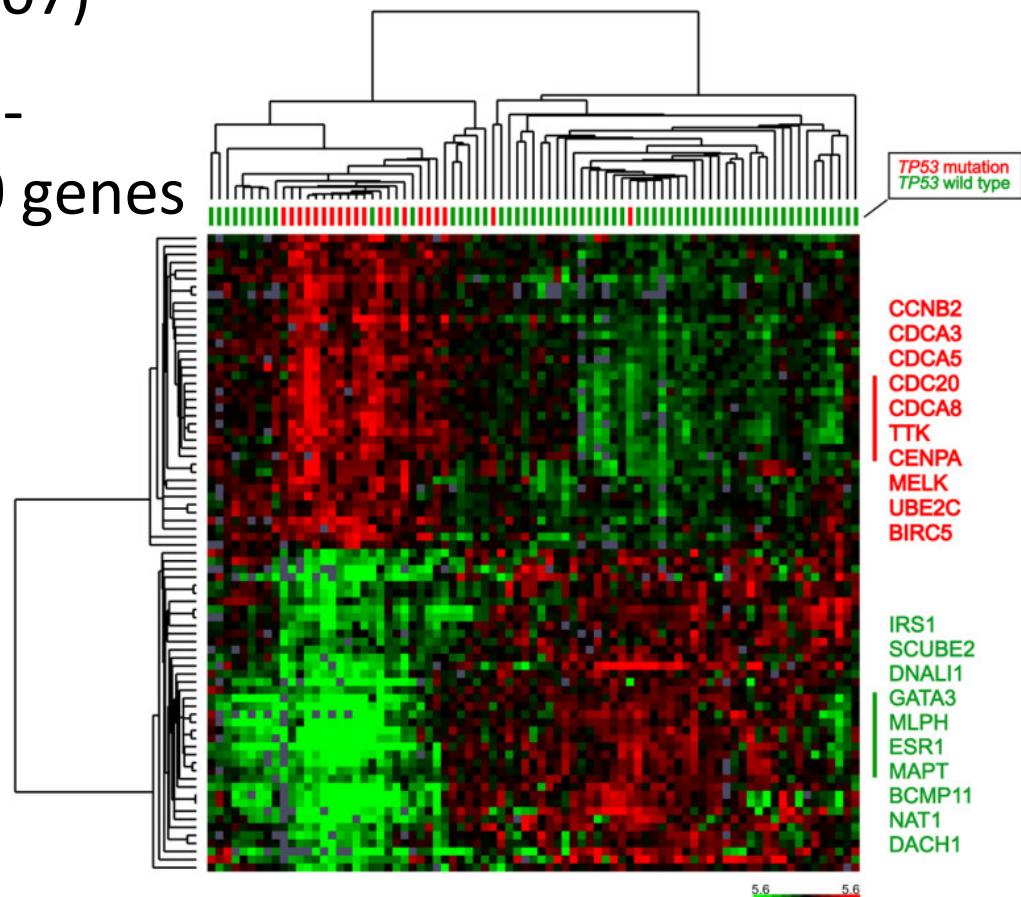
Clustering



- Clustering rows
grouping similar objects
- Clustering columns
grouping similar variables across samples
- Bi-Clustering/Two-way clustering
grouping objects that are similar across a subset of variables

Illustrations (1)

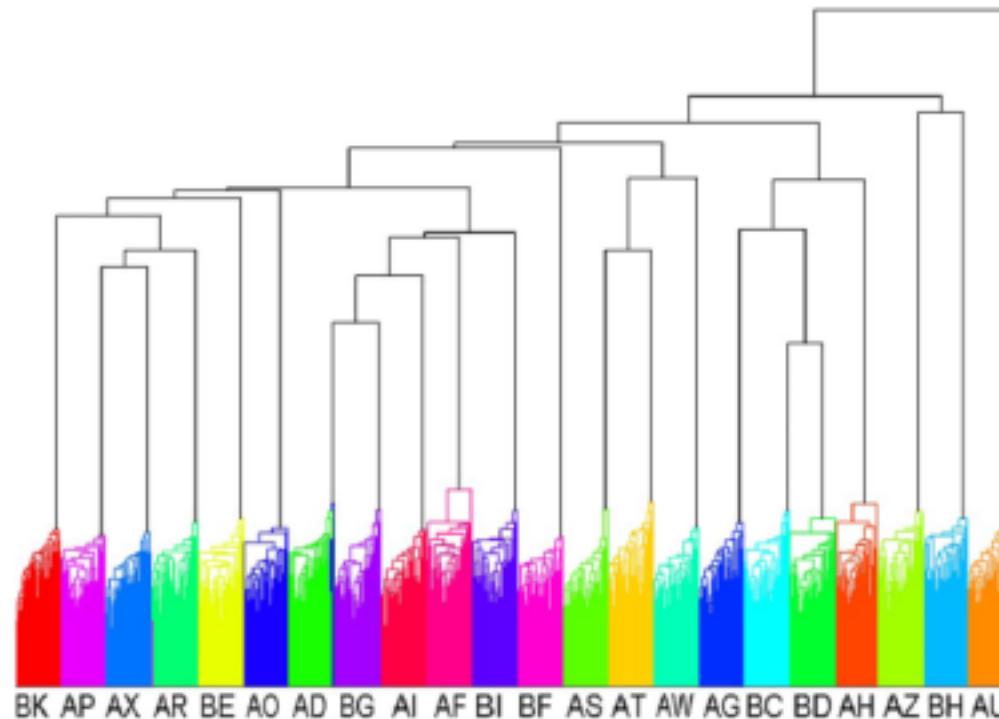
- Breast cancer data (Langerød et al., Breast cancer, 2007)
- 80 tumor samples (wild-type, TP53 mutated), 80 genes



Illustrations (2)

Assfalg et al., *PNAS*, Jan 2008

- Evidence of different metabolic phenotypes in humans
- Urine samples of 22 volunteers over 3 months, NMR spectra analysed by HCA



Application: vector quantization



FIGURE 14.9. Sir Ronald A. Fisher (1890-1962) was one of the founders of modern day statistics, to whom we owe maximum-likelihood, sufficiency, and many other fundamental concepts. The image on the left is a 1024×1024 grayscale image at 8 bits per pixel. The center image is the result of 2×2 block VQ, using 200 code vectors, with a compression rate of 1.9 bits/pixel. The right image uses only four code vectors, with a compression rate of 0.50 bits/pixel

Principal Component Analysis

- An exploratory technique used to reduce the dimensionality of the data set to a smaller space (2D, 3D)

A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	PC1	PC2
0.25	0.93	0.04	-0.78	-0.53	0.57	0.19	0.29	0.37	-0.22	0.36	0.1
-2.3	-1.2	-4.5	-0.51	-0.76	0.07	0.81	0.95	0.99	0.26	-2.3	-1.2
-0.29	-1	0.73	-0.33	0.52	0.13	0.13	0.53	-0.5	-0.48	0.27	-0.89
-0.16	-0.17	-0.26	0.32	-0.08	-0.38	-0.48	0.99	-0.95	0.34	-0.19	0.7
0.07	-0.87	0.39	0.5	-0.63	-0.53	0.79	0.88	0.74	-0.14	-0.77	-0.7
0.61	0.15	0.68	-0.94	0.5	0.06	-0.56	0.49	0	-0.77	-0.65	-0.99

- Transform some large number of variables into a smaller number of uncorrelated variables called principal components (PCs)

Objectives of PCA

- Reduce dimensionality (pre-processing for other methods)
- Choose the most useful (informative) variables
- Compress the data
- Visualize multidimensional data
 - to identify groups of objects
 - to identify outliers

Illustration (1)

Holmes et al., *Nature*, Vol. 453, No. 15, May 2008

- Investigation of metabolic phenotype variation across and within four human populations (17 cities from 4 countries: China, Japan, UK, USA)
- ^1H NMR spectra of urine specimens from 4630 participants
- PCA plots of median spectra per population (city) and gender

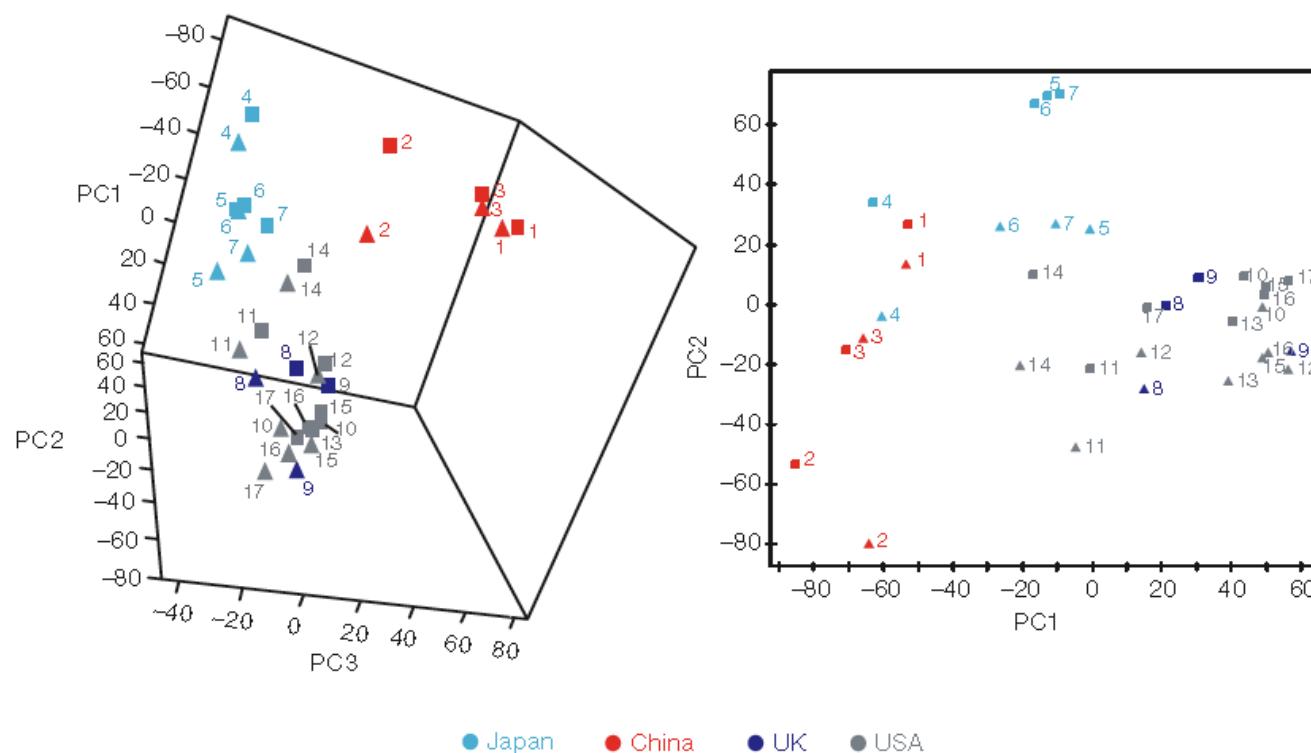
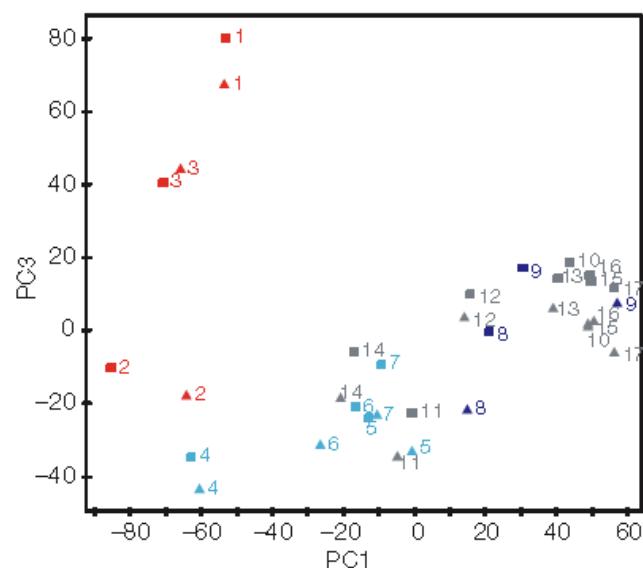
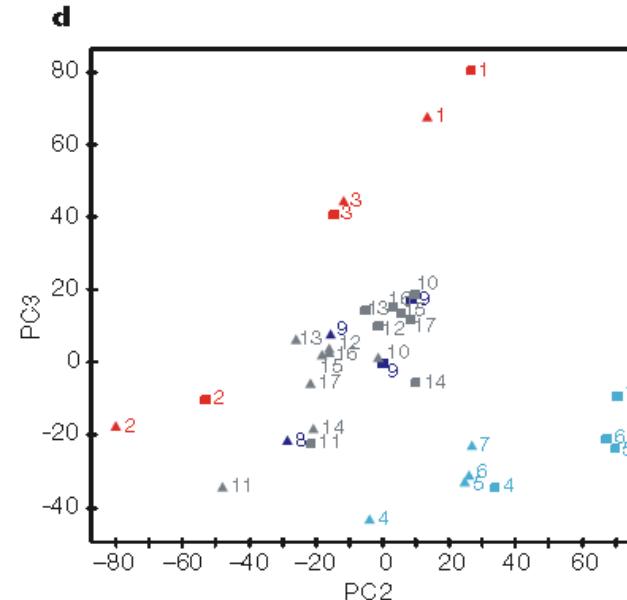
a**c****d**

Illustration (2)

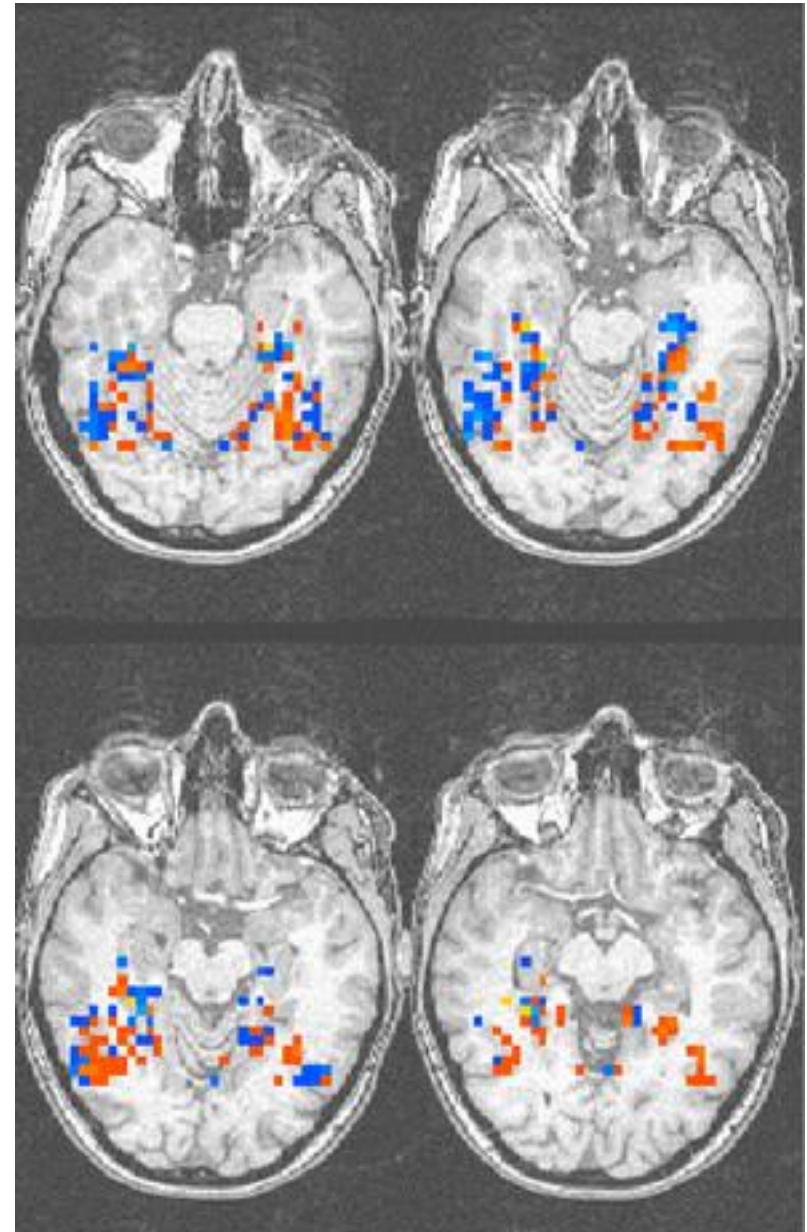
Neuroimaging

L voxels (brain regions)

→ N patients/brain maps

A1 A2 A3 A4 A5 ... A7 A8

A1	A2	A3	A4	A5	...	A7	A8
-0.91	0.74	0.74	0.97	-0.06	...	-0.04	-0.73
-2.3	-1.2	-4.5	0.47	0.13	...	0.16	0.26
-0.98	-0.46	0.98	0.77	-0.14	...	0.44	-0.12
0.97	-0.64	-0.3	-0.14	-0.29	...	-0.43	0.27
-0.64	-0.34	0.21	-0.57	-0.39	...	0.02	-0.61
0.41	-0.95	0.21	-0.17	-0.68	...	0.11	0.49



Books

- Reference book for the course:
 - *The elements of statistical learning: data mining, inference, and prediction.* T. Hastie et al, Springer, 2001 (second edition in 2009)
 - Freely downloadable here:
 - <http://statweb.stanford.edu/~tibs/ElemStatLearn/>
- Other textbooks
 - *Machine Learning.* Tom Mitchell, McGraw Hill, 1997.
 - *Pattern classification* (2nd edition). R.Duda, P.Hart, D.Stork, Wiley Interscience, 2000
 - *Pattern Recognition and Machine Learning (Information Science and Statistics).* C.M.Bishop, Springer, 2004
 - *Introduction to Machine Learning.* Ethan Alpaydin, MIT Press, 2004.
 - *Machine Learning: The Art and Science of Algorithms that Make Sense of Data.* Peter Flach. Cambridge University Press, 2012.
 - *Machine Learning: a probabilistic perspective.* Kevin P. Murphy. MIT Press, 2012.

Books

- More advanced/specific topics
 - *kernel methods for pattern analysis.* J. Shawe-Taylor and N. Cristianini. Cambridge University Press, 2004
 - *Reinforcement Learning: An Introduction.* R.S. Sutton and A.G. Barto. MIT Press, 1998
 - *Neuro-Dynamic Programming.* D.P Bertsekas and J.N. Tsitsiklis. Athena Scientific, 1996
 - *Semi-supervised learning.* Chapelle et al., MIT Press, 2006
 - *Predicting structured data.* G. Bakir et al., MIT Press, 2007
 - *Deep learning.* Goodfellow, Bengio, Courville, MIT Press, 2016

Generic ML toolboxes

- scikit-learn
 - scikit-learn.org
 - Open source machine learning toolbox (in Python)
- WEKA
 - <http://www.cs.waikato.ac.nz/ml/weka/>
 - Open source machine learning toolbox (in Java)
- Many R and Matlab packages
 - <http://www.kyb.mpg.de/bs/people/spider/>
 - <http://www.cs.ubc.ca/~murphyk/Software/BNT/bnt.html>
 - ...

Journals

- Journal of Machine Learning Research
- Machine Learning
- IEEE Transactions on Pattern Analysis and Machine Intelligence
- Journal of Artificial Intelligence Research
- Neural computation
- Annals of Statistics
- IEEE Transactions on Neural Networks
- Data Mining and Knowledge Discovery
- ...

Conferences

- International Conference on Machine Learning (ICML)
- European Conference on Machine Learning (ECML)
- Neural Information Processing Systems (NIPS)
- Uncertainty in Artificial Intelligence (UAI)
- International Joint Conference on Artificial Intelligence (IJCAI)
- International Conference on Artificial Neural Networks (ICANN)
- Computational Learning Theory (COLT)
- Knowledge Discovery and Data mining (KDD)
- ...

Classification and regression trees

Pierre Geurts & Louis Wehenkel

Institut Montefiore, University of Liège, Belgium



ELEN062-1
Introduction to Machine Learning
September 2022

Acknowledgment: These slides have been reformatted by Yann Claes in July 2020.

Outline

- ① Supervised learning
- ② Principles of decision trees
- ③ Extensions
- ④ Regression trees
- ⑤ By-products
- ⑥ Conclusions, research and further reading

Outline

- ① Supervised learning
- ② Principles of decision trees
- ③ Extensions
- ④ Regression trees
- ⑤ By-products
- ⑥ Conclusions, research and further reading

Database/Dataset/Sample

A database/dataset/sample is a **collection of objects/observations** (rows) described by **attributes/features/variables** (columns).

checkingaccount	duration	purpose	amount	savings	yearsemployed	age	good or bad
0<=...<200 DM	48	radiotv	5951	...<100 DM	1<...<4	22	bad
...<0 DM	6	radiotv	1169	unknown	...>7	67	good
no	12	education	2096	...<100 DM	4<...<7	49	good
...<0 DM	42	furniture	7882	...<100 DM	4<...<7	45	good
...<0 DM	24	newcar	4870	...<100 DM	1<...<4	53	bad
no	36	education	9055	unknown	1<...<4	35	good
no	24	furniture	2835	500<...<1000 DM	...>7	53	good
0<=...<200 DM	36	usedcar	6948	...<100 DM	1<...<4	35	good
no	12	radiotv	3059	...>1000 DM	4<...<7	61	good
0<=...<200 DM	30	newcar	5234	...<100 DM	unemployed	28	bad
0<=...<200 DM	12	newcar	1295	...<100 DM	...<1	25	bad
...<0 DM	48	business	4308	...<100 DM	...<1	24	bad
0<=...<200 DM	12	radiotv	1567	...<100 DM	1<...<4	22	good

Supervised learning

Inputs				Output
A1	A2	...	A1000	Y
-0.86	0.17	...	0	C2
-2.3	-1.2	...	-0.42	C1
-0.37	-0.11	...	-0.64	C1
0.41	0.67	...	-0.8	C2
-0.51	-0.59	...	0.98	C2
-0.25	-0.27	...	-0.68	C1
-0.52	0.23	...	0.11	C1
-1.3	-0.2	...	0.14	C1
0.93	-0.78	...	-0.01	C2
-0.25	-0.29	...	0.69	C2
-0.6	0.92	...	-0.64	C1
0.22	-0.8	...	-0.5	C2
-0.62	0.2	...	0.08	C1
-0.3	0.8	...	0.02	C2
-0.91	0.44	...	-0.57	C1
0.76	0.65	...	-0.08	C1

$$\longrightarrow \hat{Y} = f(A_1, \dots, A_{1000})$$

Automatic learning

Goal: from the database, find a function $f(\cdot)$ of the inputs that approximates at best the output.

2 cases:

- **Discrete** output \rightarrow **classification** problem
- **Continuous** output \rightarrow **regression** problem

Application (i)

- ▶ Predict whether or not a bank client will be a good or a bad debtor.
- ▶ Image classification:
 - Face recognition
 - Handwritten characters recognition

3 → 3

5 → 5

Application (ii)

- ▶ Classification of cancer types from gene expression profiles

No. patient	Gene 1	Gene 2	...	Gene 7129	Leucemia
1	-134	28	...	123	AML
2	-123	0	...	17	AML
3	56	-123	...	-23	ALL
...
72	89	-123	...	12	ALL

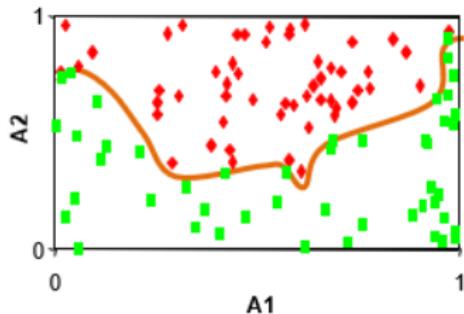
Source: [1]

Learning algorithm

A learning algorithm receives as input a **learning sample** and returns a **function** $h(\cdot)$.

It is defined by:

- A hypothesis space H , which is a set of candidate models.
- A quality measure for a model.
- An optimisation strategy.



A model $h(\cdot) \in H$ obtained by automatic learning.

Outline

- ① Supervised learning
- ② Principles of decision trees
 - Tree representation
 - Tree learning
- ③ Extensions
- ④ Regression trees
- ⑤ By-products
- ⑥ Conclusions, research and further reading

Classification trees (aka Decision trees)

A supervised learning algorithm that can handle:

- **Classification problems**, that can be binary or multi-valued.
- **Discrete** (binary or multi-valued) or **continuous** attributes.

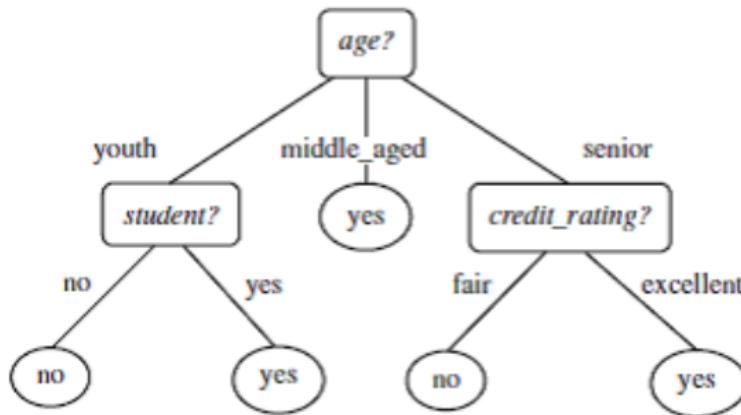
Classification trees were invented several times:

- ▶ By statisticians: e.g. CART (Breiman et al.)
- ▶ By the AI community: e.g. ID3, C4.5 (Quinlan et al.)

Hypothesis space

A decision tree is a tree where:

- Each interior node tests an attribute
- Each branch corresponds to an attribute value
- Each leaf is labelled with a class



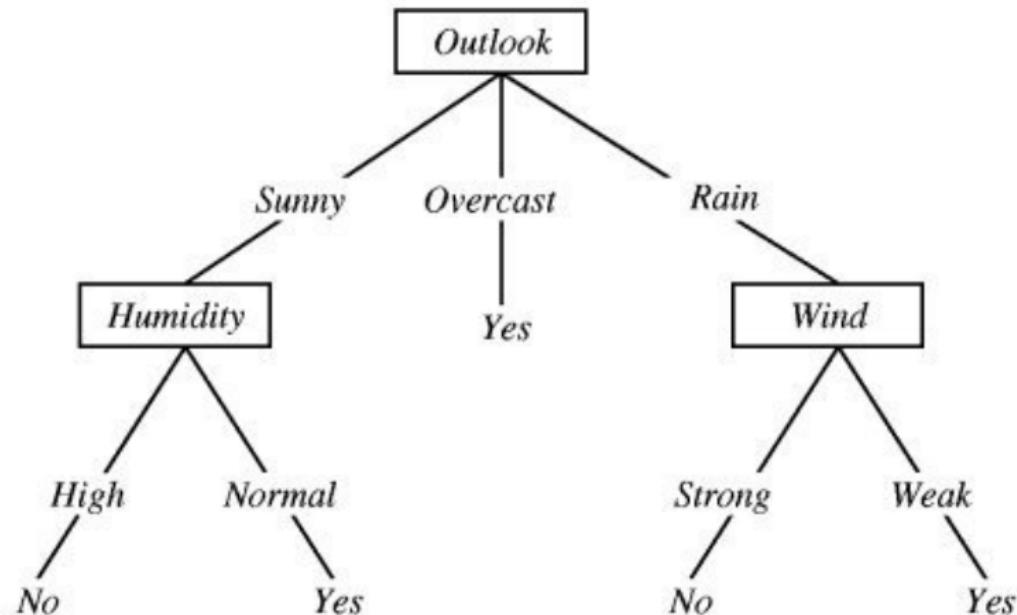
A decision tree allowing one to predict whether a customer will buy a given kind of computer. Source: [2]

Illustration - Should I play tennis? (i)

Day	Outlook	Temperature	Humidity	Wind	Play Tennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	Normal	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	High	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Hot	Normal	Weak	Yes
D10	Rain	Mild	Normal	Strong	Yes
D11	Sunny	Cool	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Adapted from [3]. (NB: first column is 'dummy'; output $Y \equiv$ 'Play Tennis')

Illustration - Should I play tennis? (ii)

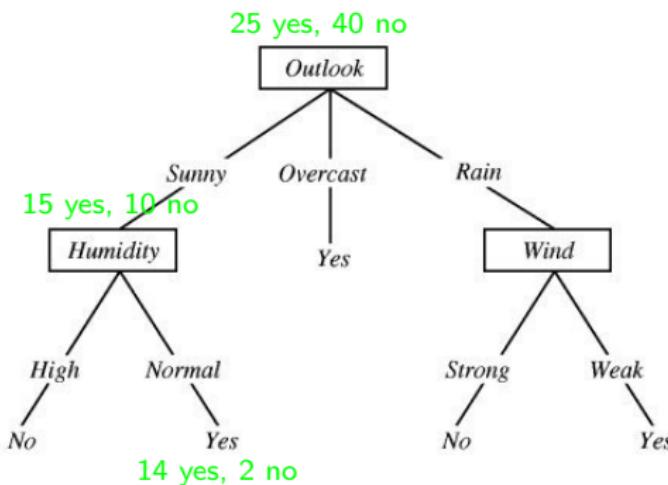


Source: [3]

Tree learning

Tree learning problem consists in choosing the **tree structure** and determining **the predictions** at leaf nodes.

For each leaf, **the prediction (or label)** is chosen such that the misclassification error in the part of the *LS* reaching that leaf is **minimized**: the **majority class** in the part of the *LS* reaching the leaf.



How to generate trees? (i)

What properties do we want a decision tree to have?

- ▶ It should be consistent with the learning sample (for the moment):
 - Trivial algorithm: construct a decision tree that has one path to a leaf for each example.
Problem: it does not capture useful information from the database.

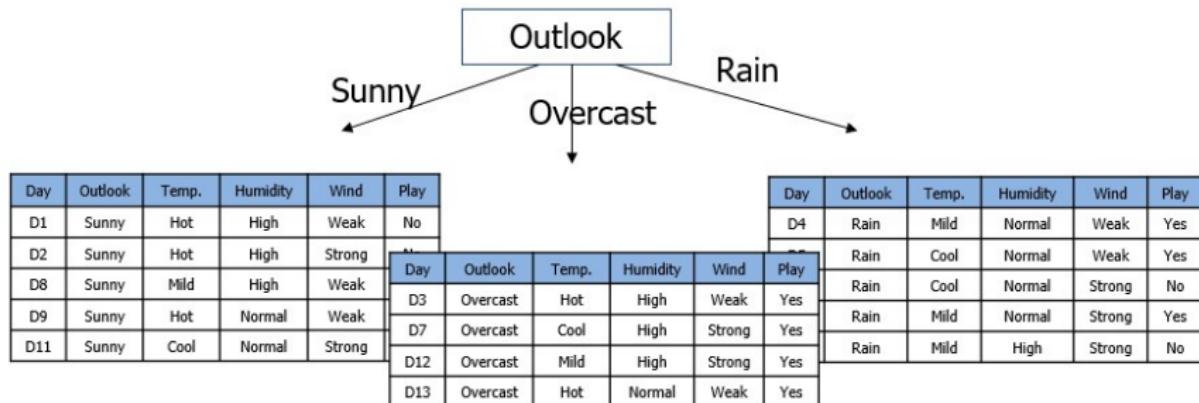
How to generate trees (ii)

What properties do we want decision trees to have?

- ▶ It should be at the same time as simple as possible.
 - Trivial algorithm: generate all trees and pick the simplest one that is consistent with the learning sample.
 - Problem:** there are too many trees.

Top-down induction of decision trees (i)

Idea: Choose the best attribute, split the learning sample accordingly and proceed recursively until each object is correctly classified.



Top-down induction of decision trees (ii)

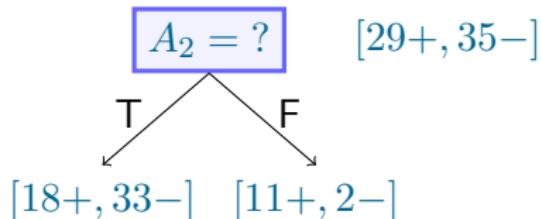
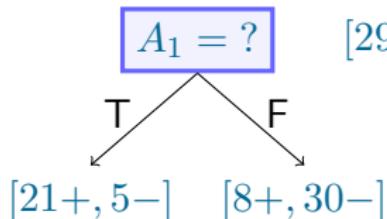
Algorithm 1: learn_dt(LS)

```
if all objects from  $LS$  have the same class or if all objects have the
same values for every attribute then
    Create a leaf with a label corresponding to the majority class of
    the objects of  $LS$ ;
end if
else
    Use  $LS$  to find the best splitting attribute  $A^*$  ;
    Create a test node for that attribute ;
    forall different values  $a$  of  $A^*$  do
        Build  $LS_a = \{o \in LS \mid A^*(o) \text{ is } a\}$  ;
        Use learn_dt( $LS_a$ ) to grow a subtree from  $LS_a$ 
    end forall
end if
```

Properties of the top-down approach

- ▶ Hill-climbing algorithm in the space of possible decision trees:
 - It adds a sub-tree to the current tree and continues its search
 - It does not backtrack
- ▶ Highly dependent upon the criterion for selecting attributes to test
(what we called above the “best splitting attribute A^* ”)
- ▶ **Sub-optimal (heuristic)** but very fast

Which attribute is the best splitting attribute?



We want a small tree. Therefore, we should **maximize** the class separation at each step, i.e make successors as **pure** as possible.
⇒ it will favour short paths in trees.

Impurity measures

Let:

- LS be a sample of objects
- p_j ($\forall j = 1, \dots, J$) the proportion of objects in the LS belonging to output-class j .

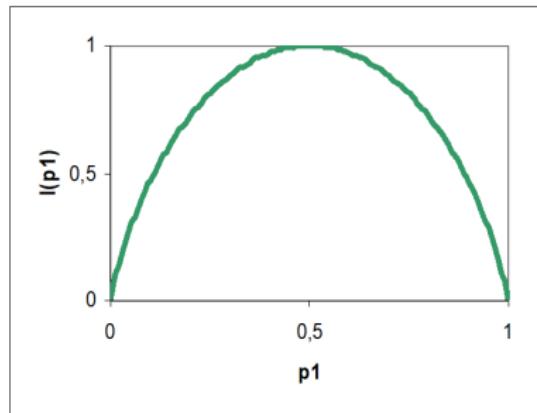
An **impurity** measure $I(LS)$ should satisfy the following props:

- $I(LS)$ is minimum only when $\exists i$ s.t. $p_i = 1$ and $p_j = 0$ for $j \neq i$ (pure sample);
- $I(LS)$ is maximum only when $\forall j : p_j = \frac{1}{J}$ (uniform number of objects among classes);
- $I(LS)$ is a symmetric function of its arguments p_1, \dots, p_J .

Shannon entropy as an impurity measure (i)

Definition of Shannon entropy:

$$\begin{aligned} H(LS) &\triangleq -\sum_{j=1}^J p_j \log_2 p_j \\ &\triangleq I_{Sh}(LS) \end{aligned}$$



If there are only two classes we have (since $p_2 = 1 - p_1$)

$$H(LS) = -p_1 \log_2 p_1 - (1 - p_1) \log_2 (1 - p_1).$$

Notice that I_{Sh} satisfies the above props (see graphic).

(NB: Shannon entropy is the basis of information theory.)

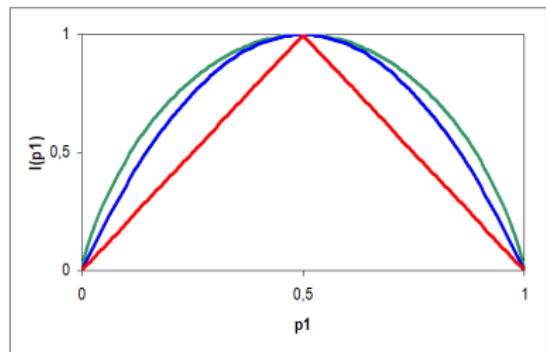
Other examples of impurity measures (ii)

- ▶ Gini index:

$$I_{\text{Gi}}(LS) \stackrel{\Delta}{=} \sum_{j=1}^J p_j(1 - p_j).$$

- ▶ (Misclassification) Error rate:

$$I_{\text{ER}}(LS) \stackrel{\Delta}{=} 1 - \max_j p_j.$$



Two-class case. Respectively, the **Shannon entropy**, the **Gini index** and the **Error rate**, normalized between 0 and 1.

Reduction of impurity achieved by a split

For a given impurity measure, the best splitting attribute is the one which **maximizes** the expected **reduction** of impurity defined by

$$\Delta I(LS, A) = I(LS) - \sum_{a \in A(LS)} \frac{|LS_a|}{|LS|} I(LS_a),$$

where LS_a is the subset of objects o from LS such that $A(o) = a$, and where $A(LS)$ is the set of different values of A observed in LS .

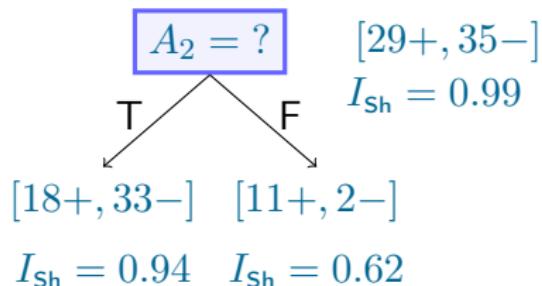
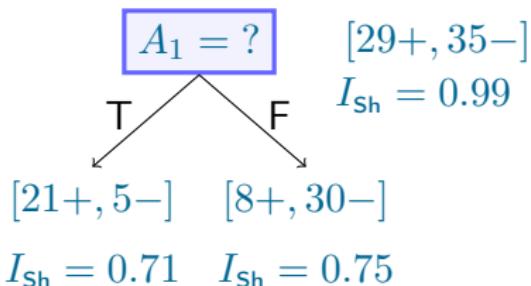
ΔI is also called a **score measure** or a splitting criterion.

NB: There are other ways to define a splitting criterion that do not rely on an impurity measure.

NB: The reduction of Shannon entropy is called the **information gain**.

Illustration (with Shannon entropy as impurity measure)

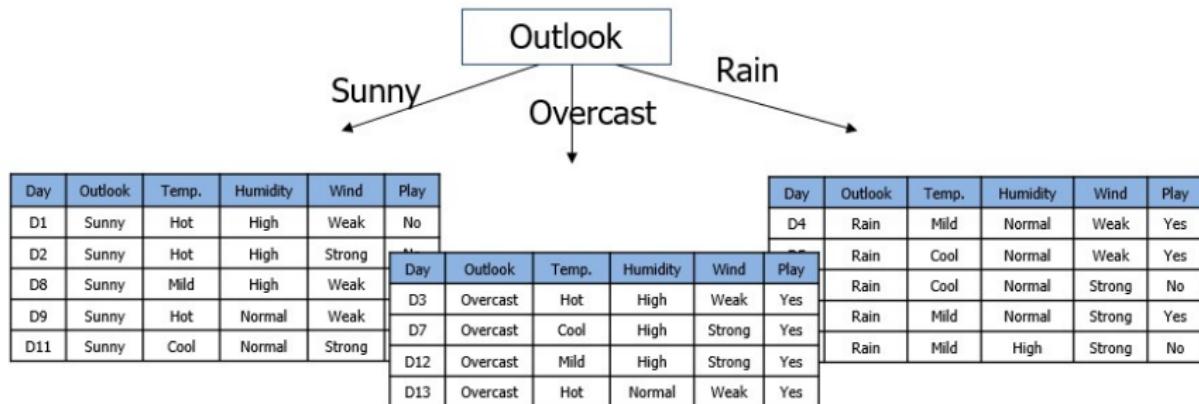
Which attribute is the best one to split?



$$\rightarrow \Delta I_{Sh}(LS, A_1) = 0.99 - \frac{26}{64} \times 0.71 - \frac{38}{64} \times 0.75 = 0.25$$

$$\rightarrow \Delta I_{Sh}(LS, A_2) = 0.99 - \frac{51}{64} \times 0.94 - \frac{13}{64} \times 0.62 = 0.12$$

Application to the tennis problem



Which attribute should be tested here?

- $\Delta I_{Sh}(LS, \text{Temp.}) = 0.970 - \frac{3}{5} \times 0.918 - \frac{1}{5} \times 0.0 - \frac{1}{5} \times 0.0 = 0.419$
- $\Delta I_{Sh}(LS, \text{Hum.}) = 0.970 - \frac{3}{5} \times 0.0 - \frac{2}{5} \times 0.0 = 0.970$
- $\Delta I_{Sh}(LS, \text{Wind}) = 0.970 - \frac{2}{5} \times 1.0 - \frac{3}{5} \times 0.918 = 0.019$

The best attribute is thus *humidity*.

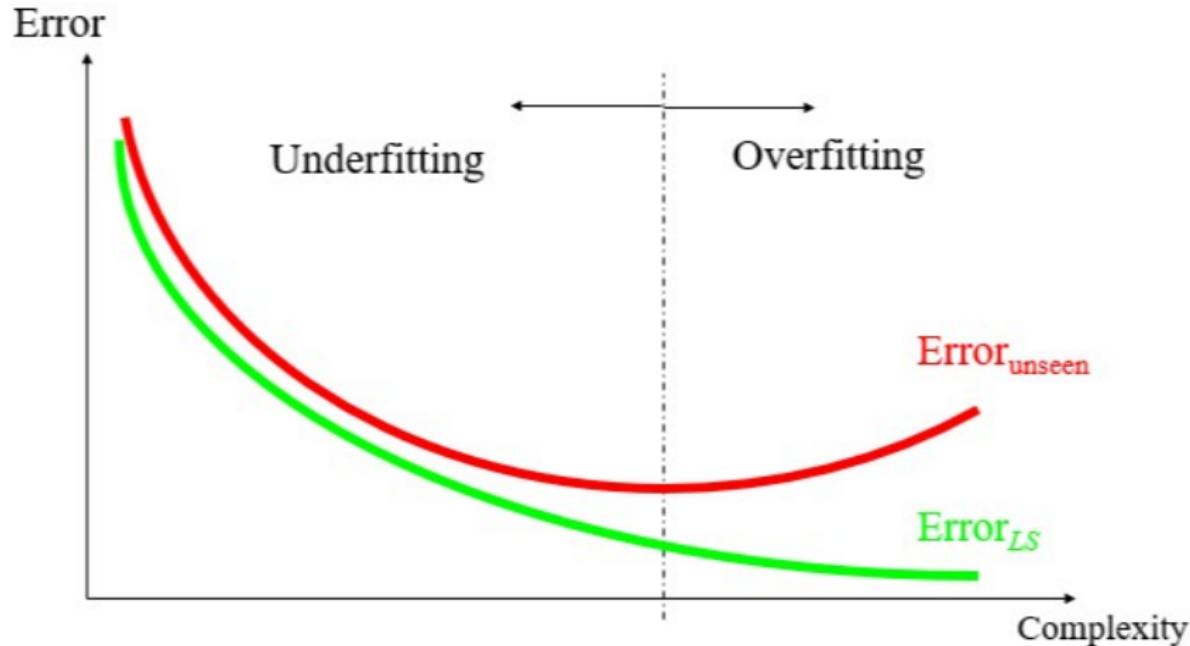
Overfitting (i)

For now, trees are perfectly consistent with the learning sample. However, often, we would like them to be good at predicting classes of unseen data from the same distribution, which is called **generalization**.

A tree T overfits the learning sample if and only if $\exists T'$ such that:

1. $\text{Error}_{LS}(T) < \text{Error}_{LS}(T')$
2. $\text{Error}_{unseen}(T) > \text{Error}_{unseen}(T')$

Overfitting (ii)

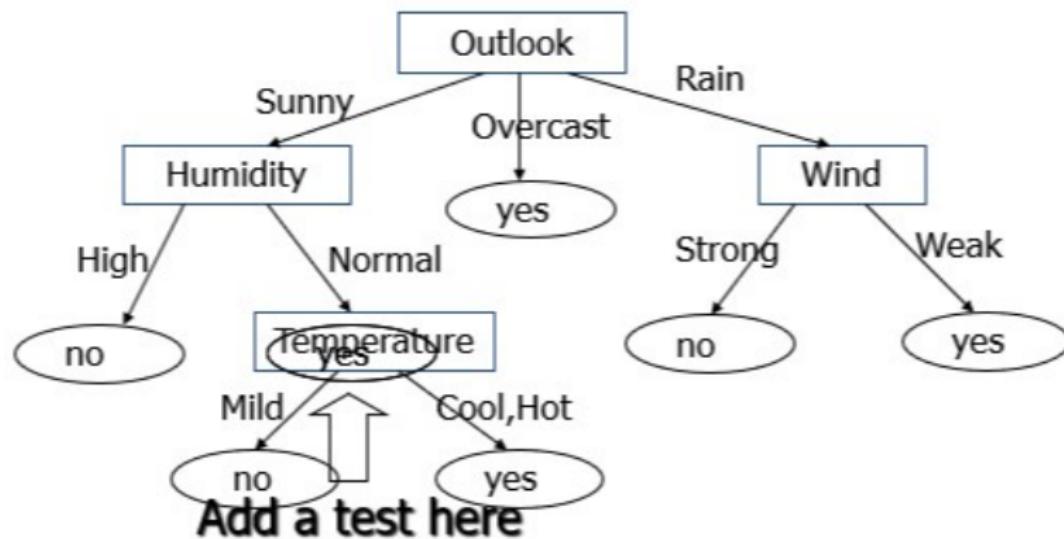


In practice, $\text{Error}_{\text{unseen}}(T)$ is estimated from a separate test sample.

Why do trees overfit the learning sample? (i)

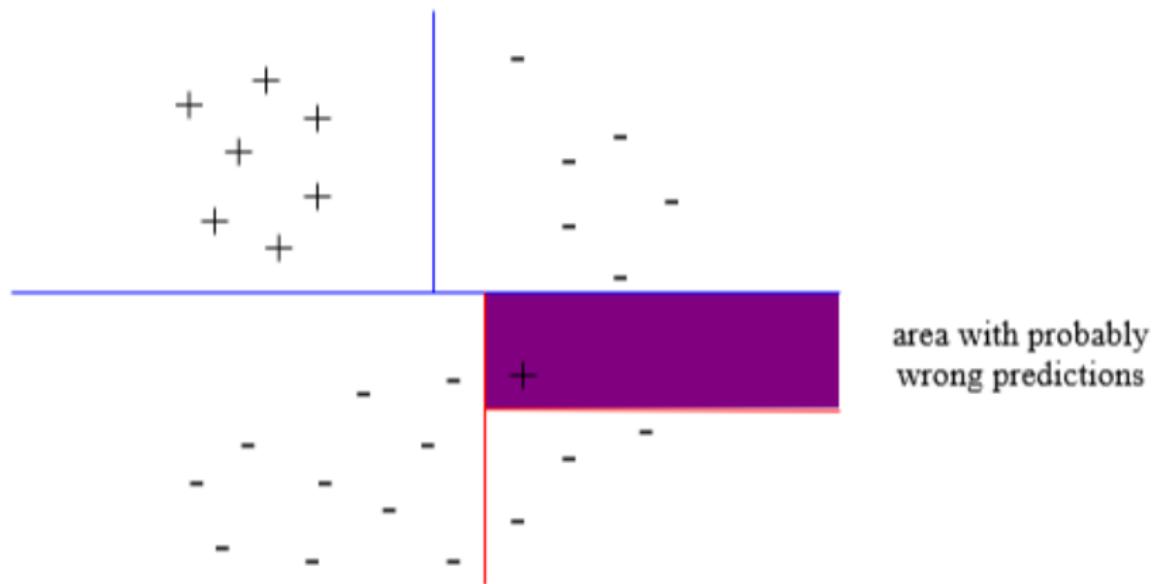
Data is noisy or attributes do not completely predict the outcome.

Day	Outlook	Temperature	Humidity	Wind	Play Tennis
D15	Sunny	Mild	Normal	Strong	No



Reasons for overfitting (ii)

Data is incomplete, i.e. all cases are not covered.



We do not have enough data in some part of the learning sample to make a good decision.

How to avoid overfitting?

- ▶ Pre-pruning: stop growing the tree earlier, before it reaches the point where it perfectly classifies the learning sample.
- ▶ Post-pruning: allow the tree to overfit, then post-prune it.
- ▶ Ensemble methods: these will be covered later in the course.

Pre-pruning

Idea: stop splitting a node if either:

- the local sample size is $< N_{\min}$
- the local sample impurity $< I_{\text{th}}$
- the impurity reduction ΔI of the best test is not large enough, according to some statistical hypothesis test at level α .

Caveats:

- for criteria [a,b,c] suitable values of the meta-parameters, N_{\min} , I_{th} , α , are often problem dependent;
- criterion [c] may recommend stop-splitting too early.

Post-pruning (i)

Idea: split the learning sample into two sets:

- a growing sample GS to build the tree
- a validation sample VS to evaluate its generalization error

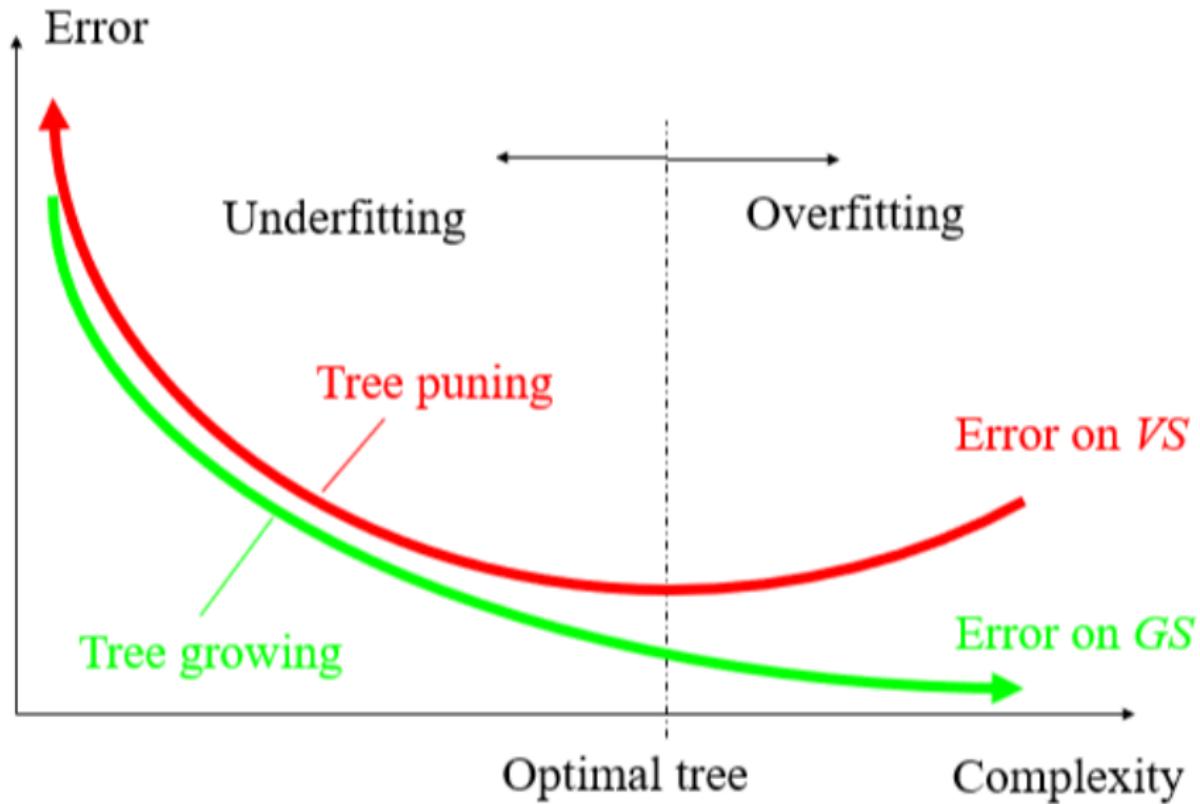
Then, build a complete tree from GS .

Next, compute a sequence of trees $\{T_1, T_2, \dots\}$ where:

- T_1 is the complete tree
- T_i is obtained by removing some test nodes from T_{i-1}

Finally, select the tree T_i^* from the sequence that **minimizes** the error on the validation sample VS .

Post-pruning (ii)



Post-pruning (iii)

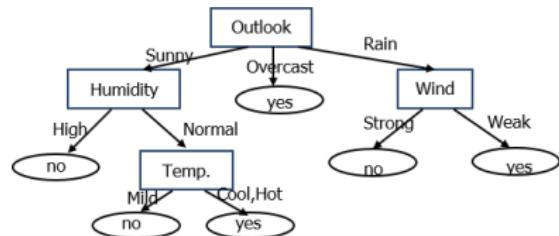
How to build the sequence of trees?

- ▶ Reduced error pruning: at each step, remove the node that most decreases the error on the validation sample.
- ▶ Cost-complexity pruning: define a cost-complexity criterion

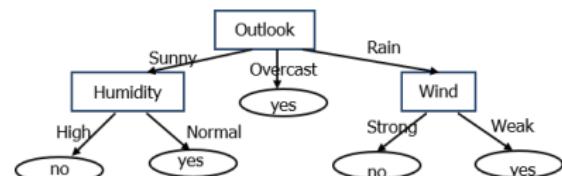
$$\text{Error}_{GS}(T) + \beta \text{ Complexity}(T)$$

and build the sequence of trees that minimizes this criterion, for increasing values of β .

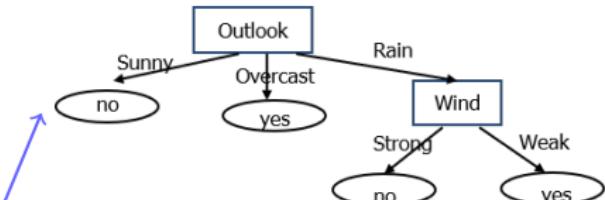
Post-pruning (iv)



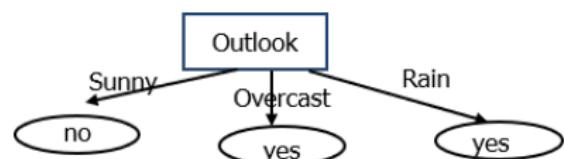
Error_{GS} = 0%, Error_{VS} = 10%



Error_{GS} = 6%, Error_{VS} = 8%



Error_{GS} = 13%, Error_{VS} = 15%



Error_{GS} = 27%, Error_{VS} = 25%

Error_{GS} = 33%, Error_{VS} = 35%

Post-pruning (v)

Problem: it requires to dedicate a part of the learning sample as a validation set, which may be a problem in the case of a small database.

⇒ **Solution:** K -fold cross-validation

- Split the training set into K parts, often 10
- Generate K trees, each one leaving out one part among K
- Make a prediction for each learning object with the only tree built without this case
- Estimate the error of this prediction

K -fold cross-validation may be combined with pruning.

How to use decision trees?

► Large data sets (ideal case):

- Split the data set into three parts: GS , VS , TS
- Grow a tree from GS
- Post-prune it from VS
- Test it on TS

► Small data sets (often):

- Grow a tree from the whole database
- Pre-prune it with default parameters (risky)/post-prune it by 10-fold cross-validation (costly)
- Estimate its accuracy by 10-fold cross-validation

Outline

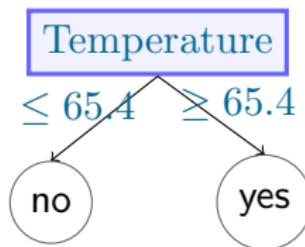
- ① Supervised learning
- ② Principles of decision trees
- ③ Extensions
 - Continuous attributes
 - Attributes with many values
 - Missing values
- ④ Regression trees
- ⑤ By-products
- ⑥ Conclusions, research and further reading

Continuous attributes (i)

Example: temperatures as a number instead of a discrete value.

There are two solutions:

- Pre-discretize: *cold* if the temperature is below 70, *mild* if between 70 and 75, *hot* if above 75.
- Discretize during tree growing:



How to find the cut-point?

Continuous attributes (ii)

Temp.	Play
80	No
85	No
83	Yes
75	Yes
68	Yes
65	No
64	Yes
72	No
75	Yes
70	Yes
69	Yes
72	Yes
81	Yes
71	No

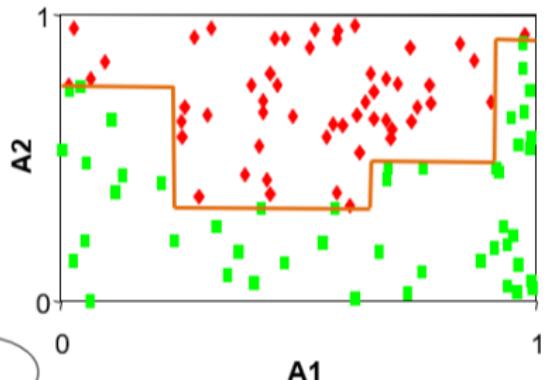
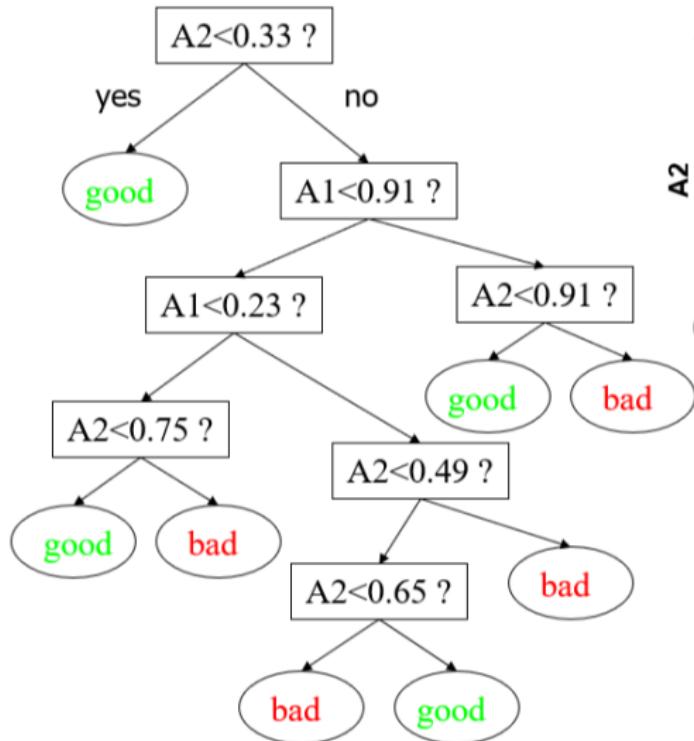
Sort →

Temp.	Play		
64	Yes	→ Temp. < 64.5	$\Delta I = 0.048$
65	No	→ Temp. < 66.5	$\Delta I = 0.010$
68	Yes	→ Temp. < 68.5	$\Delta I = 0.000$
69	Yes	→ Temp. < 69.5	$\Delta I = 0.015$
70	Yes	→ Temp. < 70.5	$\Delta I = 0.045$
71	No	→ Temp. < 71.5	$\Delta I = 0.001$
72	No		
72	Yes	→ Temp. < 73.5	$\Delta I = 0.001$
75	Yes		
75	Yes	→ Temp. < 77.5	$\Delta I = 0.025$
80	No	→ Temp. < 80.5	$\Delta I = 0.000$
81	Yes	→ Temp. < 82.0	$\Delta I = 0.010$
83	Yes	→ Temp. < 84.0	$\Delta I = 0.113$
85	No		

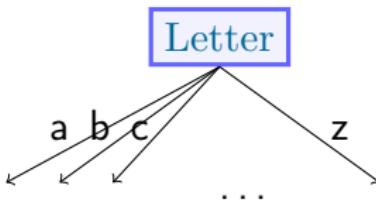
Continuous attributes (iii)

Number	A1	A2	Colour
1	0.58	0.75	Red
2	0.78	0.65	Red
3	0.89	0.23	Green
4	0.12	0.98	Red
5	0.17	0.26	Green
6	0.50	0.48	Red
7	0.45	0.16	Green
8	0.80	0.75	Green
...
100	0.75	0.13	Green

Continuous attributes (iv)



Attributes with many values (i)



Problems:

- Not good splits: they fragment the data too quickly, leaving unsufficient data for the next level.
- The reduction of impurity of such tests is often high (e.g. splitting on the object ID)

There are two solutions:

- Change the splitting criterion to penalize attributes with many values.
- Consider only binary splits (preferable)

Attributes with many values (ii)

Modified splitting criteria:

- $\text{GainRatio}(LS, A) = \frac{\Delta I_{\text{Sh}}(LS, A)}{\text{SplitInformation}(LS, A)}$
- $\text{SplitInformation}(LS, A) = - \sum_a \frac{|LS_a|}{|LS|} \log_2 \left(\frac{|LS_a|}{|LS|} \right)$
The split information is high when there are many values.

Example: outlook in the tennis problem.

- $\Delta I_{\text{Sh}}(LS, \text{outlook}) = 0.246$
- $\text{SplitInformation}(LS, \text{outlook}) = 1.577$
- $\text{GainRatio}(LS, \text{outlook}) = \frac{0.246}{1.577} = 0.156 < 0.246$

Problem: the gain ratio favours unbalanced tests.

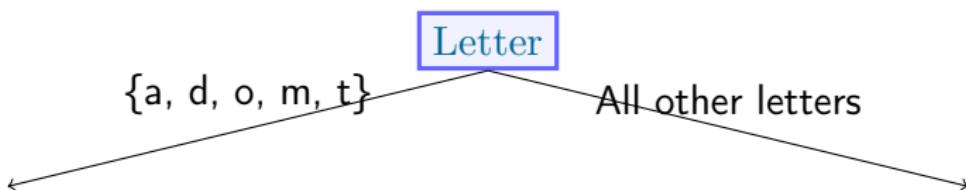
Attributes with many values (iii)

Outlook	Temperature	Humidity	Windy	PlayTennis
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Rainy	Mild	Normal	False	Yes
Sunny	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rainy	Mild	High	True	No

Source: [3]

Attributes with many values (iv)

Allow binary tests only:



There are $2^{N-1} - 1$ non-trivial binary partitions for N values. If N is small, we can use enumeration.

However, if N is large, a **heuristic** is needed.

Example: Greedy approach

Missing attribute values

Not all attribute values are known for every object during learning or testing.

Day	Outlook	Temperature	Humidity	Wind	Play
D15	Sunny	Hot	?	Strong	No

There are three strategies:

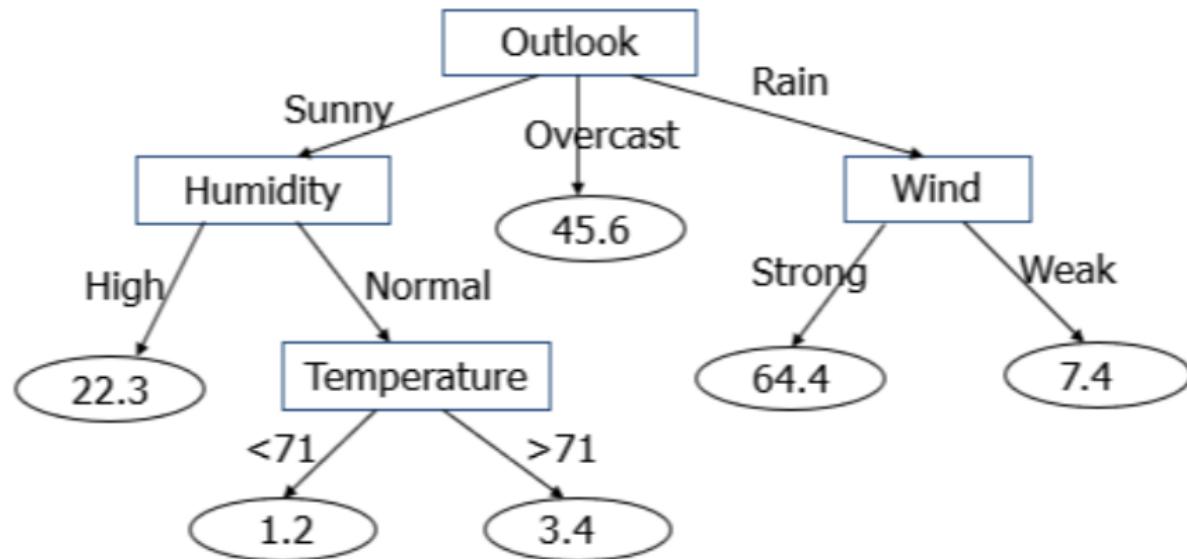
- Assign the most common value in the learning sample
- Assign the most common value in the tree
- Assign a probability to each possible value

Outline

- ① Supervised learning
- ② Principles of decision trees
- ③ Extensions
- ④ Regression trees
- ⑤ By-products
- ⑥ Conclusions, research and further reading

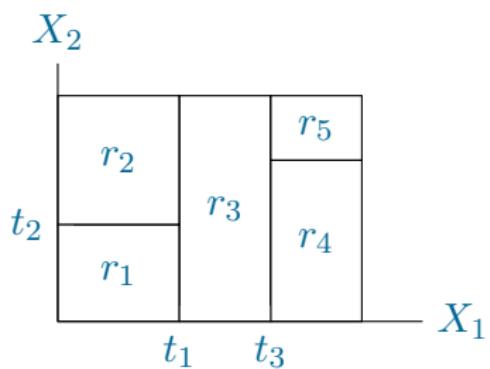
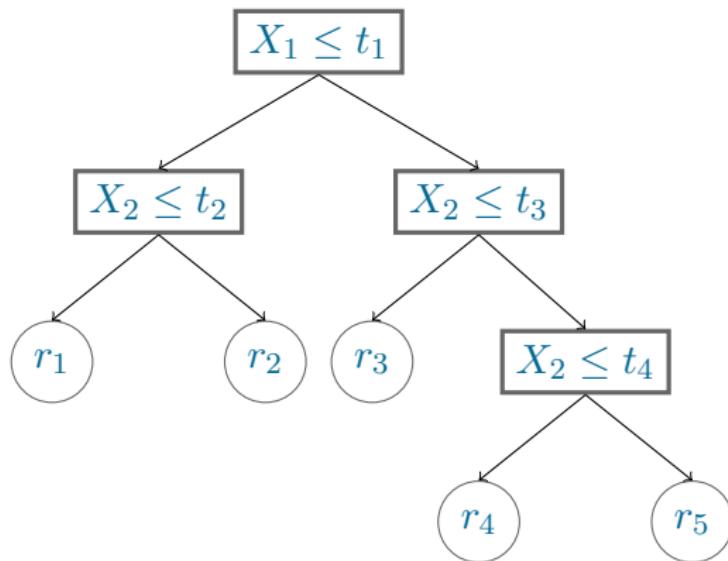
Regression trees (i)

A regression tree is exactly the same model as a decision tree but with a **number** in each **leaf** instead of a class.



Regression trees (ii)

A regression tree is a piecewise constant function of the input attributes.



Regression tree growing

To minimize the square error on the learning sample, the prediction at a leaf is the **average output** of the learning cases reaching that leaf.

The impurity of a sample is defined by the variance of the output in that sample:

$$I(LS) = \text{var}_{y|LS}\{y\} = E_{y|LS} \left\{ (y - E_{y|LS}\{y\})^2 \right\}$$

where $E_{y|LS}\{f(y)\}$ denotes the average of $f(y)$ in the sample LS .

The best split is the one that reduces the most variance:

$$\Delta I(LS, A) = \text{var}_{y|LS}\{y\} - \sum_a \frac{|LS_a|}{|LS|} \text{var}_{y|LS_a}\{y\}$$

Regression tree pruning

The methods are exactly the same: pre-pruning and post-pruning.

In post-pruning, the tree that minimizes the squared error on VS is selected.

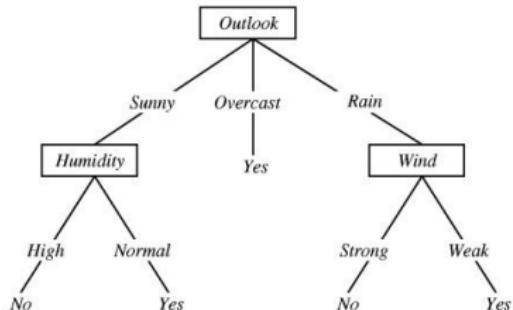
In practice, pruning is more important in regression problems because full trees are much more complex: often, all objects have a different output value and hence the full tree has as many leaves as objects in the learning sample.

Outline

- ① Supervised learning
- ② Principles of decision trees
- ③ Extensions
- ④ Regression trees
- ⑤ By-products
 - Interpretability
 - Variable selection
 - Variable importance
- ⑥ Conclusions, research and further reading

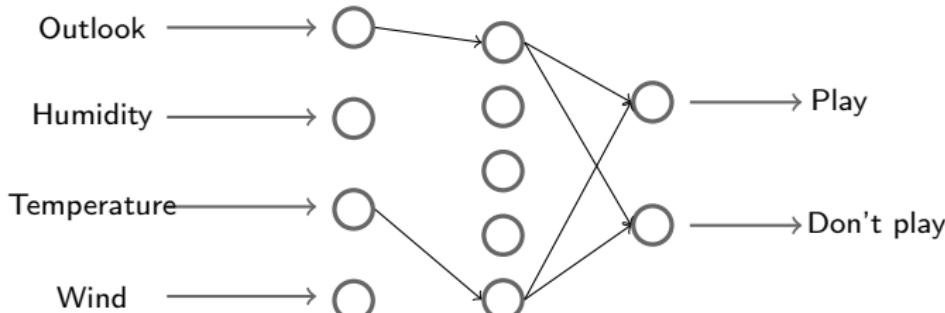
Interpretability (i)

With decision trees,
interpretability is obvious.

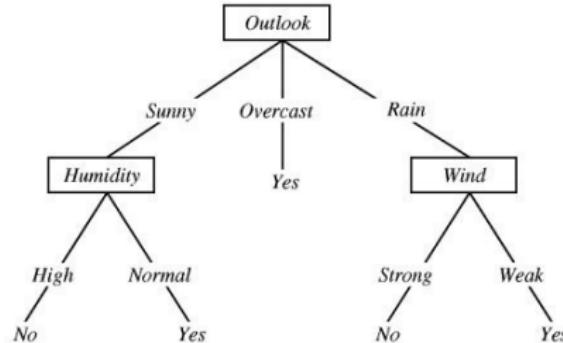


Source: [3]

With neural networks, it is more complex.



Interpretability (ii)



Source: [3]

A tree may be converted into a set of rules:

- ▶ If (Outlook = *Sunny*) and (Humidity = *High*) then
PlayTennis = *No*
- ▶ If (Outlook = *Sunny*) and (Humidity = *Normal*) then
PlayTennis = *Yes*
- ▶ If Outlook = *Overcast* then PlayTennis = *Yes*
- ▶ ...

Attribute selection

If some attributes are not useful for classification, they will not be selected in the pruned tree.

Attribute selection is of practical importance, if measuring the value of an attribute is costly.

Example: Medical diagnosis

Decision trees are often used as a pre-processing step for other learning algorithms that suffer more when there are irrelevant variables.

Variable importance

In many applications, all variables do **not** contribute equally in predicting the output.

We can evaluate variable importance by computing the total reduction of impurity brought by each variable:

$$I(A) = \sum_{\text{Nodes where } A \text{ is tested}} |LS_{node}| \Delta I(LS_{node}, A)$$



Outline

- ① Supervised learning
- ② Principles of decision trees
- ③ Extensions
- ④ Regression trees
- ⑤ By-products
- ⑥ Conclusions, research and further reading

Decision trees - Conclusions

Advantages:

- They are very fast: they can handle very large datasets with many attributes.
→ Complexity: $\mathcal{O}(nN \log N)$ where n is the number of attributes and N the number of samples.
- They are flexible: they can handle several attribute types, classification and regression problems, missing values, ...
- They have a good interpretability: they provide rules and attribute importance.

Disadvantages:

- They are quite unstable, due to their high variance.
- They are not always competitive with other algorithms in terms of accuracy.

Research

- ▶ Cost and un-balanced learning sample.
- ▶ Oblique trees (test like $\sum \alpha_i A_i < a_{th}$).
- ▶ Using predictive models in leaves, e.g. using linear regression.
- ▶ Induction graphs.
- ▶ Fuzzy decision trees (from a crisp partition to a fuzzy partition of the learning sample).

Further reading

- ▶ Hastie et al., *The elements of statistical learning: data mining, inference, and prediction*, [4]:
 - chapter 9, Section 9.2
- ▶ Louppe Gilles, *Understanding random forests : from theory to practice* [5].
- ▶ L. Breiman et al., *Classification and regression trees*, [6]
- ▶ J.R. Quinlan, *C4.5: programs for machine learning*, [7]
- ▶ D.Zighed and R.Rakotomalala, *Graphes d'induction: apprentissage et data mining*, [8]
- ▶ Supplementary slides are also available on the course website.

Software

- ▶ scikit-learn: <http://scikit-learn.org>
- ▶ Weka: <https://www.cs.waikato.ac.nz/ml/weka/>
- ▶ R: packages *tree* and *rpart*

References |

-  Todd R Golub, Donna K Slonim, Pablo Tamayo, Christine Huard, Michelle Gaasenbeek, Jill P Mesirov, Hilary Coller, Mignon L Loh, James R Downing, Mark A Caligiuri, et al.
Molecular classification of cancer: class discovery and class prediction by gene expression monitoring.
science, 286(5439):531–537, 1999.
-  Decision tree and entropy algorithm.
[https://zhengtianyu.wordpress.com/2013/12/13/
decision-trees-and-entropy-algorithm/](https://zhengtianyu.wordpress.com/2013/12/13/decision-trees-and-entropy-algorithm/).
Accessed: 2020-07-29.
-  A tutorial to understand decision tree id3 learning algorithm.
[https://nulpointerexception.com/2017/12/16/
a-tutorial-to-understand-decision-tree-id3-learning-algorithm/](https://nulpointerexception.com/2017/12/16/a-tutorial-to-understand-decision-tree-id3-learning-algorithm/).
Accessed: 2020-07-29.
-  Trevor Hastie, Robert Tibshirani, and Jerome Friedman.
The elements of statistical learning: data mining, inference, and prediction.
Springer Science & Business Media, 2009.
-  Gilles Louppe.
Understanding random forests : from theory to practice.
Université de Liège, 2014.

References II

-  Leo Breiman, Jerome Friedman, Richard Olshen, and Charles Stone.
Classification and regression trees. wadsworth int.
Group, 37(15):237–251, 1984.
-  J Ross Quinlan.
C4. 5: programs for machine learning.
Elsevier, 2014.
-  Djamel A Zighed and Ricco Rakotomalala.
Graphes d'induction: apprentissage et data mining.
Hermes Science Publications Paris, 2000.

Frequently asked questions

- ▶ What is the computational complexity of the learning algorithm ?
- ▶ How do we handle (continuous) numerical input variables ?
- ▶ Explain the optimal splitting algo.
- ▶ What are the 2-3 main changes to make to implement regression tree learning with respect to decision tree learning ?
- ▶ How to adapt this approach to general loss-functions $\ell(\cdot, \cdot)$?
- ▶ Discuss theoretical asymptotic ($N \rightarrow \infty$) properties ?
- ▶ Discuss computational complexity and interpretability.

Linear regression

Louis Wehenkel & Pierre Geurts

Institut Montefiore, University of Liège, Belgium



ELEN062-1
Introduction to Machine Learning
October 12, 2021

Outline

① Batch-mode Supervised Learning

② Linear regression

Least mean square error solution

Regularization and algorithmics

Residual fitting

- ▶ Objects (or observations): $LS = \{o_1, \dots, o_N\}$
- ▶ Attribute vector: $a^i = (a_1(o_i), \dots, a_n(o_i))^T, \quad \forall i = 1, \dots, N.$
- ▶ Attribute values: $a_j = (a_j(o_1), \dots, a_j(o_N))^T \quad \forall j = 1, \dots, n.$
- ▶ Outputs: $y^i = y(o_i)$ or $c^i = c(o_i), \quad \forall i = 1, \dots, N.$
- ▶ LS Table

o	$a_1(o)$	$a_2(o)$	\dots	$a_n(o)$	$y(o)$
1	a_1^1	a_2^1	\dots	a_n^1	y^1
2	a_1^2	a_2^2	\dots	a_n^2	y^2
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots
N	a_1^N	a_2^N	\dots	a_n^N	y^N

- ▶ LS attribute matrix: $A = (a^1, \dots, a^N)$ (n lines, N columns)
- ▶ LS output column: $\mathbf{y} = (y^1, \dots, y^N)^T$

Linear regression models

- ▶ Output is numerical scalar
- ▶ All inputs are numerical scalars
- ▶ Linear regression tries to approximate output by

$$\hat{y}(o) = w_0 + \sum_{i=1}^n w_i a_i(o)$$

- ▶ Supervised learning problem:

Choose the parameters w_0, w_1, \dots, w_n so as to fit well LS and have good generalization to unseen objects

Linear regression models

Linear in the parameters, not necessarily in the original inputs.

$$\hat{y}(o) = w_0 + \sum_{i=1}^k w_i \phi_i(a(o))$$

Inputs can come from different sources:

- ▶ quantitative measurements
- ▶ transformations of quantitative measurements (log, square-root, etc.)
- ▶ basis expansions, such as $a_2(o) = a_1^2(o)$, $a_2(o) = a_1^3(o)$, etc.
- ▶ numeric or “dummy” coding of qualitative inputs

Least mean square error solution

Posing, $a_0(o) = 1, \forall o$ and denoting by

1. $\mathbf{a}'(o_i) = (a_0(o_i), a_1(o_i), \dots, a_n(o_i))^T$, and
2. $\mathbf{w}' = (w_0, w_1, \dots, w_n)^T$, square error (SE) at o_i is defined by

$$SE(o_i, \mathbf{w}') = (y(o_i) - \hat{y}(o_i))^2 = (y(o_i) - \mathbf{w}'^T \mathbf{a}'(o_i))^2$$

and the total squared error (TSE) by

$$TSE(LS, \mathbf{w}') = \sum_{i=1}^N (y(o_i) - \mathbf{w}'^T \mathbf{a}'(o_i))^2$$

or in vector notation (denoting by $A' = (\mathbf{a}'^1, \dots, \mathbf{a}'^N)$)

$$TSE(LS, \mathbf{w}') = (\mathbf{y} - A'^T \mathbf{w}')^T (\mathbf{y} - A'^T \mathbf{w}')$$

Least mean square error solution

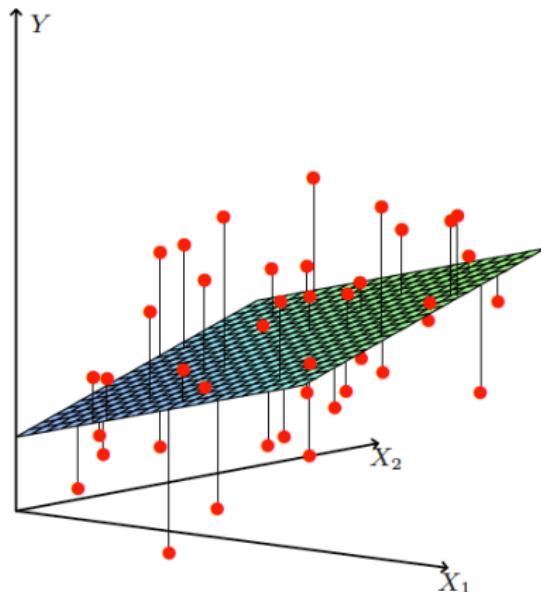


FIGURE 3.1. Linear least squares fitting with $X \in \mathbb{R}^2$. We seek the linear function of X that minimizes the sum of squared residuals from Y .

Least mean square error solution: one dimension

Assuming only one input, the solution is computed as:

$$(w_0^*, w_1^*) = \arg \min_{w_0, w_1} \sum_{i=1}^N (y(o_i) - w_0 - w_1 a_1(o_i))^2$$

Cancelling the derivative with respect to w_0 and w_1 , one gets:

$$\begin{aligned} w_1^* &= \frac{\sum_{i=1}^N (a_1(o_i) - \bar{a}_1)(y(o_i) - \bar{y})}{\sum_{i=1}^N (a_1(o_i) - \bar{a}_1)^2} = \frac{\text{cov}(a_1, y)}{\sigma_{a_1}^2} \\ w_0^* &= \bar{y} - w_1^* \bar{a}_1 \end{aligned}$$

where $\bar{a}_1 = N^{-1} \sum_{k=1}^N a_1(o_k)$ and $\bar{y} = N^{-1} \sum_{k=1}^N y(o_k)$.

Substituting the above into $\hat{y}(o) = w_0^* + w_1^* a_1(o)$:

$$\frac{\hat{y}(o) - \bar{y}}{\sigma_y} = \rho_{a_1, y} \frac{a_1(o) - \bar{a}_1}{\sigma_{a_1}},$$

with $\rho_{a_1, y}$ the correlation between a_1 and y , and σ_y, σ_{a_1} the standard deviations of y and a_1 , all computed on the LS.

Least mean square error solution: multidimensional case

Choose \mathbf{w}' to minimize

$$TSE(LS, \mathbf{w}') = (\mathbf{y} - A'^T \mathbf{w}')^T (\mathbf{y} - A'^T \mathbf{w}').$$

Differentiating w.r.t. \mathbf{w}' (gradient)

$$\nabla_{\mathbf{w}'} TSE(LS, \mathbf{w}') = -2A'(\mathbf{y} - A'^T \mathbf{w}')$$

and solving for $\nabla_{\mathbf{w}'} TSE(LS, \mathbf{w}'^*) = 0$ we obtain

$$\mathbf{w}'^* = (A' A'^T)^{-1} A' \mathbf{y}$$

Note that $\nabla_{\mathbf{w}'}^2 TSE(LS, \mathbf{w}') = 2A' A'^T$ is symmetric positive (semi-) definite.

Least mean square error solution (...)

Shift invariance: suppose we define new attribute vector by $\mathbf{a}_c(o) = \mathbf{a}(o) + \mathbf{c}$ where \mathbf{c} is a constant vector (i.e. independant of object).

Let (w_0, \mathbf{w}) be the optimal solution in the original attribute space. Then it is easy to see that $(w_0 - \mathbf{w}^T \mathbf{c}, \mathbf{w})$ is optimal in the new space.

Indeed, we have

$$\hat{y}_c(o) = w_0 - \mathbf{w}^T \mathbf{c} + \mathbf{w}^T \mathbf{a}_c(o) = w_0 + \mathbf{w}^T \mathbf{a}(o) = \hat{y}(o).$$

Hence, if $(w_0 - \mathbf{w}^T \mathbf{c}, \mathbf{w})$ is not optimal in the new space, (w_0, \mathbf{w}) couldn't be optimal in the original space.

Least mean square error solution

(...)

Let us discuss the meaning of the table $(A'A'^T)$: element i, j is obtained by the scalar product of line i and line j of matrix A' . Thus we have

$$A'A'^T = N \left(\begin{array}{c|cccc} 1 & \bar{a}_1 & \dots & \bar{a}_n \\ \hline \bar{a}_1 & g_{1,1} & \dots & g_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ \bar{a}_n & g_{n,1} & \dots & g_{n,n} \end{array} \right)$$

where $\bar{a}_i = N^{-1} \sum_{k=1}^N a_i(o_k)$ and $g_{i,j} = N^{-1} \sum_{k=1}^N a_i(o_k) a_j(o_k)$

Assuming that the attributes have all a zero mean ($\bar{a}_i = 0$) we have $g_{i,j} = cov(a_i, a_j)$

Least mean square error solution

In the sequel we will use the notation Σ to denote the covariance matrix.

Thus if all the attributes are centered, we have

$$\mathbf{w}'^* = \begin{pmatrix} N^{-1} & \mathbf{0} \\ \mathbf{0} & N^{-1}\Sigma^{-1} \end{pmatrix} \begin{pmatrix} \mathbf{1} \\ A \end{pmatrix} \mathbf{y}.$$

In particular, $\mathbf{w}_0^* = N^{-1} \sum_{k=1}^N y^k = N^{-1} \sum_{k=1}^N y(o_k) = \bar{y}$.

In other words, if both a_i and y are centered, $\mathbf{w}_0^* = 0$.

Least mean square error solution

Assuming that the attributes have zero mean and unit variance ($g_{i,i} = 1$), we have

$$A'A'^T = N \left(\begin{array}{c|cccc} 1 & 0 & \dots & 0 \\ \hline 0 & \rho_{1,1} & \dots & \rho_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \rho_{n,1} & \dots & \rho_{n,n} \end{array} \right)$$

Note that $\rho_{i,i} = 1; \forall i = 1, \dots, n.$

- In this case the correlation and covariance matrices are identical.
- Pre-whiten the attributes before solving the linear system.
- Below, we assume attributes are pre-whitened and drop suffix $'$.

Least mean square error solution

Let us take a non-singular $n \times n$ matrix B and define the transformed attribute vector by $\mathbf{a}_B(o) = B\mathbf{a}(o)$.

For the transformed attributes, matrix A becomes matrix BA , and solution becomes:

$$\mathbf{w}_B = ((BA)(BA)^T)^{-1}BA\mathbf{y} = (B^T)^{-1}(AA^T)^{-1}B^{-1}BA\mathbf{y} = B^{T-1}\mathbf{w}$$

In other words,

$$\hat{\mathbf{y}}_B = \mathbf{w}_B^T \mathbf{a}_b = (B^{T-1}\mathbf{w})^T B\mathbf{a} = \mathbf{w}^T B^{-1} B\mathbf{a} = \mathbf{w}^T \mathbf{a}.$$

⇒ Invariance with respect to (non-singular) linear transformation

Least mean square error solution

Discussion of matrix $N\Sigma = AA^T$: computation, singularity, inversion.

1. It is easy to see that $N\Sigma = \sum_{i=1}^N \mathbf{a}(o_i)\mathbf{a}^T(o_i)$.
2. Therefore, rank of Σ is at most N .
3. Thus, if $n > N$, Σ is rank deficient (and hence singular).
4. If Σ is singular, unicity of optimal solution is lost, but existence is preserved.
5. Need to impose other criteria to find unique solution, i.e. to build algorithm.
6. Several such solutions are discussed in the reference book, in particular **regularization**.

Regularization of least mean square error solution

Instead of choosing \mathbf{w} to minimize

$$TSE(LS, \mathbf{w}) = (\mathbf{y} - \mathbf{A}^T \mathbf{w})^T (\mathbf{y} - \mathbf{A}^T \mathbf{w}).$$

Let us minimize w.r.t. \mathbf{w} and for given $\lambda > 0$

$$TSE_R(LS, \lambda, \mathbf{w}) = (\mathbf{y} - \mathbf{A}^T \mathbf{w})^T (\mathbf{y} - \mathbf{A}^T \mathbf{w}) + \lambda \mathbf{w}^T \mathbf{w}$$

Differentiating w.r.t. \mathbf{w} yields (I denotes the $n \times n$ identity matrix)

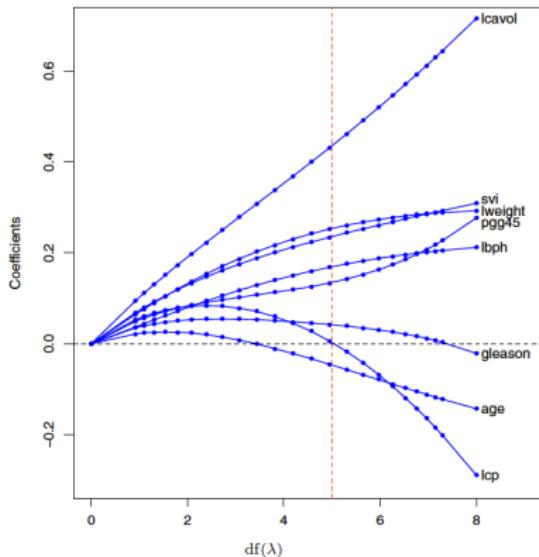
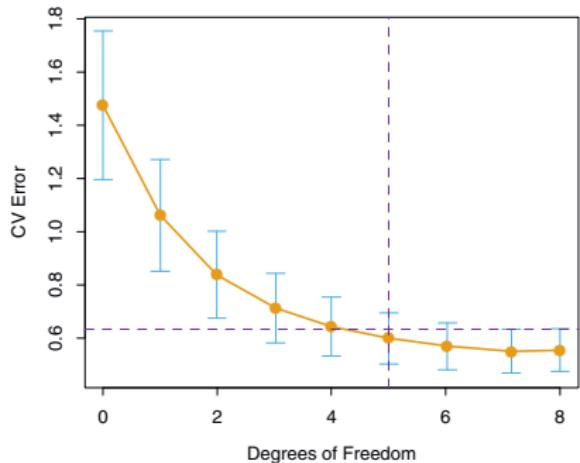
$$\nabla_{\mathbf{w}} TSE_R(LS, \mathbf{w}, \lambda) = -2\mathbf{A}(\mathbf{y} - \mathbf{A}^T \mathbf{w}) + 2\lambda I \mathbf{w}$$

in other words

$$\mathbf{w}^*(\lambda) = (A A^T + \lambda I)^{-1} A \mathbf{y}$$

which has a unique solution, $\forall \lambda > 0$!

Illustration: effect of λ on CV error and optimal weights



(See Figures 3.7 and 3.8 in reference book)

$df(\lambda) = n$ when $\lambda = 0$ and $df(\lambda) \rightarrow 0$ when $\lambda \rightarrow \infty$

Computational complexity:

- ▶ Building the covariance matrix: in the order of Nn^2 operations
- ▶ Solving the system for w^* : in the order of n^3 operations

Various alternative techniques exist to solve system.

Some will be discussed in the sequel.

Other regularizations

- ▶ The above regularization method is called *Ridge Regression*. It belongs to the family of *shrinkage methods*.
- ▶ Other regularization for linear regression models:
 - ▶ LASSO: a shrinkage method replacing $\sum_i w_i^2 < t$ by $\sum_i |w_i| < t$ (discussed later in the course).
 - ▶ Subset selection: select an optimal subset of input attributes on which to regress. Various heuristics exist to determine the subset.

Residual fitting (a.k.a. Forward-Stagewise Regression)

Residual fitting: alternative algorithm, of general interest

- ▶ Start by computing w_0 for the no-variable case: $w_0 = \bar{y}$
- ▶ Introduce attributes (**assumed of zero mean, unit variance**) progressively, one at the time
 - ▶ Define residual at step k by
$$\Delta_k y(o) = y(o) - w_0 - \sum_{i=1}^{k-1} w_i a_i(o)$$
 - ▶ Find best fit of residual with only attribute a_k :

$$w_k = \rho_{a_k, \Delta_k y} \sigma_{\Delta_k y}.$$

(since residuals have zero mean, and attributes are pre-whitened)

Note that this algorithm is in general suboptimal w.r.t. to the direct solution given previously, but it is linear in the number of attributes.

References

Chapter 3 from the reference book (Hastie *et al.*, 2009):

- ▶ Section 3.2: Linear regression models and least squares
- ▶ Section 3.4.1: Ridge regression
- ▶ Section 3.3.3: Forward stagewise regression

Frequently asked questions

- ▶ How to choose a value for λ ?
- ▶ Asymptotic ($N \rightarrow \infty$) properties of LR and Ridge-regression
- ▶ Discuss LASSO vs Ridge regression
- ▶ Discuss computational complexity and interpretability

Nearest neighbor methods

Louis Wehenkel & Pierre Geurts

Institut Montefiore, University of Liège, Belgium



ELEN062-1
Introduction to Machine Learning
September 1, 2020

Outline

- ① Batch-mode Supervised Learning
- ② Nearest neighbor and kernel-based methods
 - Properties of the NN method
 - Refinements of the NN method
- ③ Relation between tree-based and kernel-based methods
- ④ Relation between kernel-based and linear methods

- ▶ Objects (or observations): $LS = \{o_1, \dots, o_N\}$
- ▶ Attribute vector: $a^i = (a_1(o_i), \dots, a_n(o_i))^T, \quad \forall i = 1, \dots, N.$
- ▶ Outputs: $y^i = y(o_i)$ or $c^i = c(o_i), \quad \forall i = 1, \dots, N.$
- ▶ LS Table

o	$a_1(o)$	$a_2(o)$	\dots	$a_n(o)$	$y(o)$
1	a_1^1	a_2^1	\dots	a_n^1	y^1
2	a_1^2	a_2^2	\dots	a_n^2	y^2
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots
N	a_1^N	a_2^N	\dots	a_n^N	y^N

Nearest neighbor methods

Intuition: **similar objects should have similar output values.**

- ▶ NB: all inputs are numerical scalars
- ▶ Define distance measure in the input space:

$$d_a(o, o') = (\mathbf{a}(o) - \mathbf{a}(o'))^T (\mathbf{a}(o) - \mathbf{a}(o')) = \sum_{i=1}^n (a_i(o) - a_i(o'))^2$$

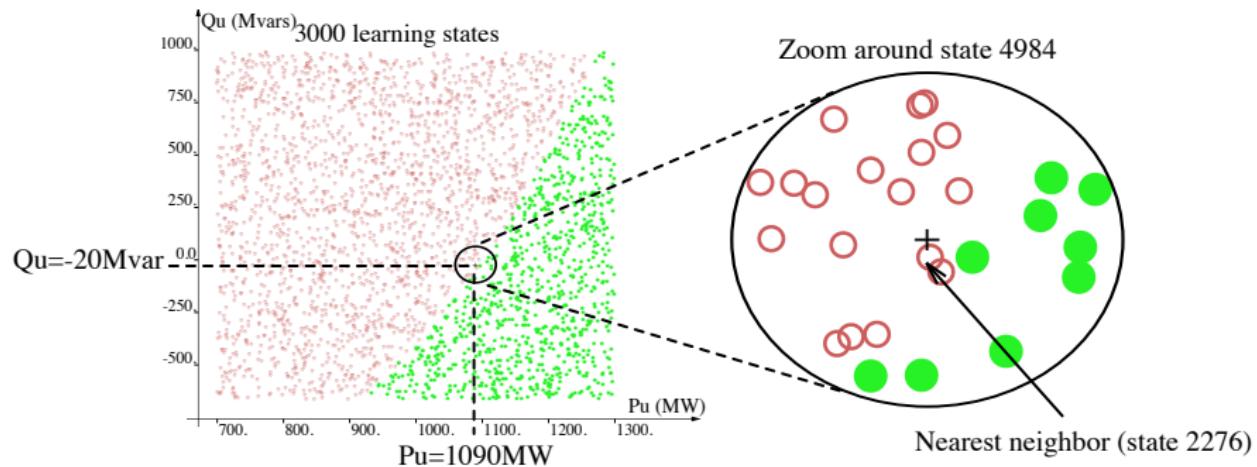
- ▶ Nearest neighbor:

$$NN_a(o, LS) = \arg \min_{o' \in LS} d_a(o, o')$$

- ▶ Extrapolate output from nearest neighbor:

$$\hat{y}_{NN}(o) = y(NN_a(o, LS))$$

Nearest neighbor methods: illustration



Properties of the NN method

Computational

- ▶ Training: storage of the LS ($n \times N$)
- ▶ Testing: N distance computations $\Rightarrow N \times n$ computations

Accuracy

- ▶ Asymptotically ($N \rightarrow \infty$): suboptimal
(except if problem is deterministic)
- ▶ Strong dependence on choice of attributes
 \Rightarrow weighting of attributes

$$d_a^{\textcolor{red}{w}}(o, o') = \sum_{i=1}^n \textcolor{red}{w}_i (a_i(o) - a_i(o'))^2$$

or attribute selection...

Refinements of the NN method

1. The k -NN method:

- ▶ Instead of using only the nearest neighbor, one uses the k (a number to be determined) nearest neighbors:

$$kNN_a(o, LS) = \text{First}(k, \text{Sort}(LS, d_a(o, \cdot)))$$

- ▶ Extrapolate from k nearest neighbors, e.g. for regression

$$\hat{y}_{kNN}(o) = k^{-1} \sum_{o' \in kNN_a(o, LS)} y(o')$$

and majority class for classification.

- ▶ k allows to control **overfitting** (like pruning of trees).
- ▶ Asymptotically ($N \rightarrow \infty$): $k(N) \rightarrow \infty$ and $\frac{k(N)}{N} \rightarrow 0 \Rightarrow$ optimal method (minimum error)

Refinements of the NN method

2. Condensing and editing of the LS :

- ▶ Condensing: remove ‘useless’ objects LS
- ▶ Editing: remove ‘outliers’ from LS
- ▶ Apply first editing then condensing (see notes)

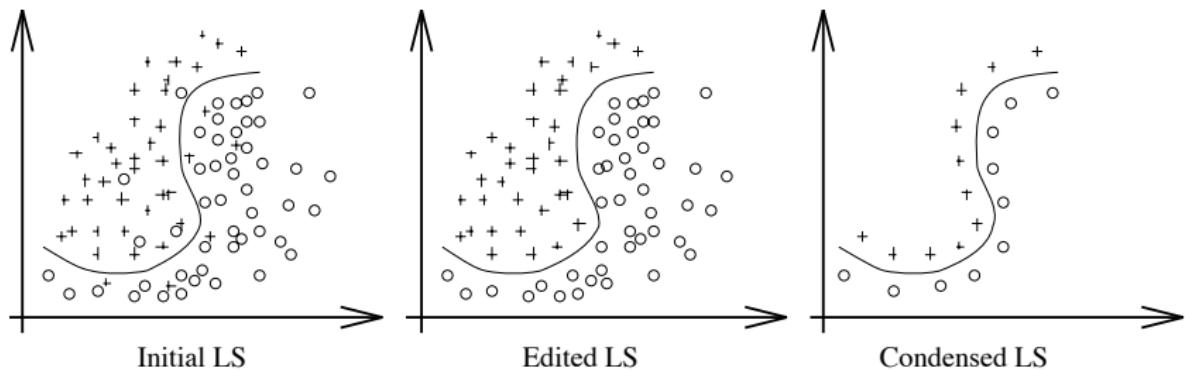
3. Automatic tuning of the weight vector w ...

4. Parzen windows and/or kernel methods:

$$\hat{y}_K(o) = \sum_{o' \in LS} y(o') K(o, o')$$

where $K(o, o')$ is a measure of similarity

Nearest neighbor, editing and condensing



Relation between tree-based and kernel-based methods

Kernel defined by a regression tree:

- ▶ Let $\mathcal{L}_i, i = 1, \dots, |\mathcal{T}|$ denote the leaves of \mathcal{T} .
- ▶ Let N_i denote the number of objects in the sub-LS of \mathcal{L}_i .
- ▶ Let $K_{\mathcal{T}}(o, o')$ be equal to N_i^{-1} if o and o' reach same leaf \mathcal{L}_i , and 0 otherwise.
- ▶ Then the approximation of the regression tree may be written as

$$\hat{y}_{\mathcal{T}}(o) = \sum_{o' \in LS} y(o') K_{\mathcal{T}}(o, o').$$

Scalar product representation of tree kernels

Kernel defined by a regression tree:

- ▶ Let $\mathcal{L}_i, i = 1, \dots, |\mathcal{T}|$ denote the leaves of \mathcal{T} .
- ▶ Let N_i denote the number of objects in the sub-LS of \mathcal{L}_i .
- ▶ For each leaf, define a function attribute $a_{\mathcal{L}_i}(o)$ by
 $a_{\mathcal{L}_i}(o) = N_i^{-1/2}$ if o reaches \mathcal{L}_i , and zero otherwise.
- ▶ Let $\mathbf{a}_{\mathcal{T}}(o) = (a_{\mathcal{L}_1}(o), \dots, a_{\mathcal{L}_{|\mathcal{T}|}}(o))^T$
- ▶ Then we have that

$$K_{\mathcal{T}}(o, o') = \mathbf{a}_{\mathcal{T}}^T(o) \mathbf{a}_{\mathcal{T}}(o')$$

- ▶ and

$$\hat{y}_{\mathcal{T}}(o) = \sum_{o' \in LS} y(o') \mathbf{a}_{\mathcal{T}}^T(o) \mathbf{a}_{\mathcal{T}}(o').$$

Relation between kernel-based and linear methods

Let us consider a two-class classification problem, and define $y(o) = 1$ if $c(o) = c_1$ and $y(o) = -1$ if $c(o) = c_2$.

Let us construct a simple classifier:

- ▶ Center of class 1: $\mathbf{c}_+ = N_+^{-1} \sum_{o' \in LS_+} \mathbf{a}(o')$
- ▶ Center of class 2: $\mathbf{c}_- = N_-^{-1} \sum_{o' \in LS_-} \mathbf{a}(o')$
- ▶ Classifier: $\hat{y}(o) = 1$ if $d(\mathbf{c}_+, \mathbf{a}(o)) < d(\mathbf{c}_-, \mathbf{a}(o))$.
- ▶ Define $\mathbf{c} = \frac{\mathbf{c}_+ + \mathbf{c}_-}{2}$ and $\Delta\mathbf{c} = \mathbf{c}_+ - \mathbf{c}_-$
- ▶ With these notations we have $\hat{y}(o) = \text{sgn}((\mathbf{a}(o) - \mathbf{c})^T \Delta\mathbf{c})$
- ▶ In other words:

$$\hat{y}(o) = \text{sgn} \left(N_+^{-1} \sum_{o' \in LS_+} \mathbf{a}^T(o') \mathbf{a}(o) - N_-^{-1} \sum_{o' \in LS_-} \mathbf{a}^T(o') \mathbf{a}(o) + b \right)$$

$$\text{where } b = \frac{1}{2} (\|\mathbf{c}_-\|^2 - \|\mathbf{c}_+\|^2)$$

Further reading

Reference book (*Hastie et al.*, 2009):

- ▶ Section 2.3
- ▶ Section 13.3

Frequently asked questions

- ▶ How to choose a value for k ?
- ▶ Asymptotic ($N \rightarrow \infty$) properties of 1-NN vs k -NN
 - ▶ Under which conditions is 1-NN asymptotically consistent?
 - ▶ Explain why: $k(N) \rightarrow \infty$ and $\frac{k(N)}{N} \rightarrow 0 \Rightarrow$ optimal method (minimum error)
- ▶ Discuss computational complexity and interpretability

(Deep) Neural Networks

Louis Wehenkel & Pierre Geurts

Institut Montefiore, University of Liège, Belgium



ELEN062-1
Introduction to Machine Learning
October 19, 2022

Outline

- ① Introduction
- ② Single neuron models
- ③ Multilayer perceptron
- ④ Other neural network models
- ⑤ Conclusion

Batch-mode vs Online-mode Supervised Learning (Notations)

- | Objects (or observations): $LS = \{o_1, \dots, o_N\}$
- | Attribute vector: $a^i = (a_1(o_i), \dots, a_n(o_i))^T$, $i = 1, \dots, N$
- | Outputs: $y^i = y(o_i)$ or $c^i = c(o_i)$, $i = 1, \dots, N$
- | LS Table

o	$a_1(o)$	$a_2(o)$	\dots	$a_n(o)$	$y(o)$
1	a_1^1	a_2^1	\vdots	a_n^1	y^1
2	a_1^2	a_2^2	\vdots	a_n^2	y^2
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots
N	a_1^N	a_2^N	\vdots	a_n^N	y^N

Focus for this lecture on numerical inputs, and numerical outputs (classes will be encoded numerically if needed).

Batch-mode vs online mode learning

- | In batch-mode
 - | Samples provided and processed **together** to **construct** model
 - | Need to store **samples** (not the model)
 - | Classical approach for data mining
- | In online-mode
 - | Samples provided and processed **one by one** to **update** model
 - | Need to store the model (not the samples)
 - | Classical approach for adaptive systems
- | But both approaches can be adapted to handle both contexts
 - | Samples available together can be exploited one by one
 - | Samples provided one by one can be stored and then exploited together

Motivations for Artificial Neural Networks

Intuition: biological brain can learn, so let's try to be inspired by it to build learning algorithms.

- | Starting point: single neuron models
 - | perceptron, LTU and STU for linear supervised learning
 - | online (biologically plausible) learning algorithms
- | Complexify: multilayer perceptrons
 - | flexible models for non-linear supervised learning
 - | universal approximation property
 - | iterative training algorithms based on non-linear optimization
- | ...other neural network models of importance

Outline

1 Introduction

2 Single neuron models

Hard threshold unit (LTU) and the perceptron

Soft threshold unit (STU) and gradient descent

Theoretical properties

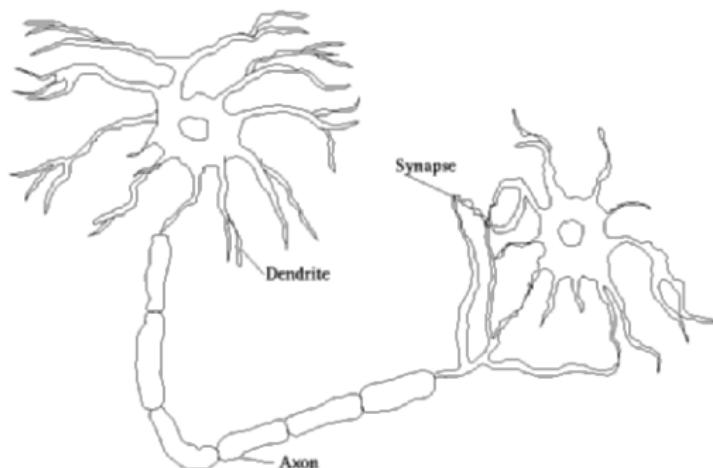
3 Multilayer perceptron

4 Other neural network models

5 Conclusion

Single neuron models

The biological neuron:



Human brain: 10^{11} neurons, each with 10^4 synapses
Memory (knowledge): stored in the synapses



—

—

—

—

— — —

— — —

— — —

—

—

—





Bias/variance trade-off, model assessment and model selection

Pierre Geurts and Louis Wehenkel

Institut Montefiore, University of Liège, Belgium



ELEN062-1
Introduction to Machine Learning
July 2020

Supervised learning (i)

Let us assume that one is given:

- ▶ A learning sample of N input-output pairs

$$LS = \{(x_i, y_i) | i = 1, \dots, N\}, \quad x_i \in \mathcal{X}, y_i \in \mathcal{Y}$$

independently and identically drawn (*i.i.d.*) from an unknown distribution $p(x, y)$.

- ▶ A loss function

$$L : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}$$

measuring the discrepancy between its arguments.

One wants to find a function $f : \mathcal{X} \rightarrow \mathcal{Y}$ that minimizes the following expectation (*generalization error*):

$$E_{x,y} \{L(y, f(x))\}$$

Supervised learning (ii)

There are two types of problems:

- Classification:

$$L(y, y') = \mathbb{1}(y \neq y') \quad (\text{error rate})$$

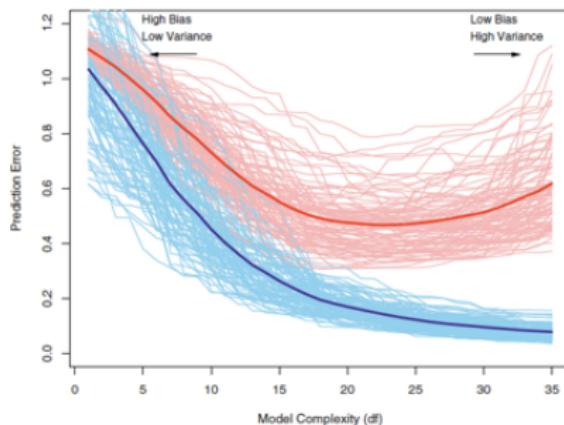
- Regression:

$$L(y, y') = (y - y')^2 \quad (\text{square error})$$

Learning set randomness

Let us denote by \hat{f}_{LS} the function learned from a learning sample LS by a given learning algorithm.

The function \hat{f}_{LS} (its prediction at some point x) is a random variable.



Error on the **learning sample** and **test sample** for 100 different samples.

Source: [1]

Two quantities of interest

Given a model \hat{f}_{LS} built from some learning sample, its **generalization** error is given by:

$$Err_{LS} = E_{x,y} \left\{ L \left(y, \hat{f}_{LS}(x) \right) \right\}$$

⇒ useful for model assessment and model selection.

Given a learning algorithm, its **expected generalization** error over random learning samples of size N is given by:

$$E_{LS} \{ Err_{LS} \} = E_{LS} \left\{ E_{x,y} \left\{ L \left(y, \hat{f}_{LS}(x) \right) \right\} \right\}$$

⇒ useful to characterize a learning algorithm.

Outline

- ▶ Bias/variance trade-off: decomposition of the expected error $E_{LS}\{Err_{LS}\}$ that helps to understand overfitting.
- ▶ Performance evaluation: procedures to estimate Err_{LS} or $E_{LS}\{Err_{LS}\}$.
- ▶ Performance measures: common loss functions L for classification and regression.

Outline

- ① Bias/variance trade-off
- ② Performance evaluation
- ③ Performance measures
- ④ Further reading

Outline

① Bias/variance trade-off

Bias and variance definitions

Parameters that influence bias and variance

Bias and variance reduction techniques

② Performance evaluation

③ Performance measures

④ Further reading

Bias and variance definitions - Outline

- ▶ Bias/variance decomposition for a simple regression problem with no input.
- ▶ Extension to regression problems with inputs
- ▶ Bias/variance trade-off in classification problems

A simple regression problem with no input (i)

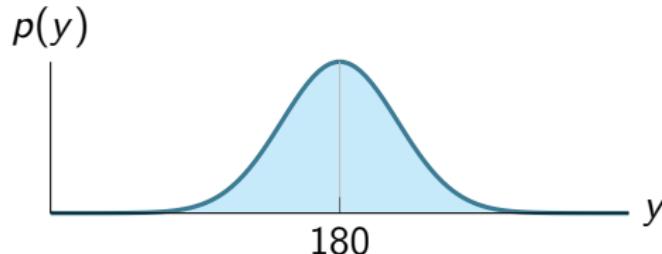
Goal: predict as well as possible the height of a Belgian male adult.

More formally:

- Choose an error measure (e.g. square error).
- Find an estimation \hat{y} such that

$$E_y \left\{ (y - \hat{y})^2 \right\}$$

over the whole population of Belgian male adults is minimized.



A simple regression problem with no input (ii)

The estimation that minimizes the error can be computed by cancelling its derivative:

$$\begin{aligned}\frac{\partial}{\partial y'} E_y \left\{ (y - y')^2 \right\} &= 0 \\ \Leftrightarrow E_y \left\{ -2(y - y') \right\} &= 0 \\ \Leftrightarrow E_y \{y\} - E_y \{y'\} &= 0 \\ \Leftrightarrow y' &= E_y \{y\}\end{aligned}$$

Hence, the estimation that minimizes the error is $E_y \{y\}$, which is called the **Bayes model**.

However, in practice, it is impossible to compute the exact value of $E_y \{y\}$, as this would imply to measure the height of every Belgian male adult.

Learning algorithms (i)

As $p(y)$ is unknown, one needs to find an estimation \hat{y} from a sample of individuals i.i.d. from the Belgian male adult population:

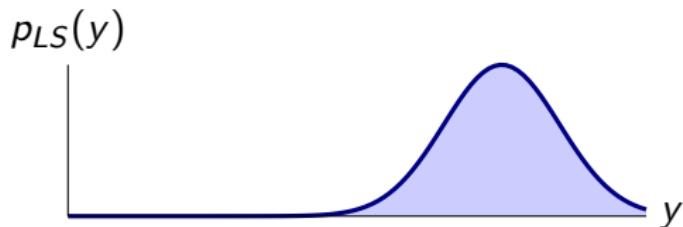
$$LS = \{y_1, \dots, y_N\}$$

Examples:

- $\hat{y}_1 = \frac{1}{N} \sum_{i=1}^N y_i$
- $\hat{y}_2 = \frac{\lambda 180 + \sum_{i=1}^N y_i}{\lambda + N}, \quad \lambda \in [0, +\infty[$ (if it is known that the height is close to 180)

Learning algorithms (ii)

As learning samples LS are randomly drawn, the prediction \hat{y} will also be a random variable:



A good learning algorithm should not be **good** only on a single learning sample but **on average** over all learning samples of size N . One thus seeks to minimize

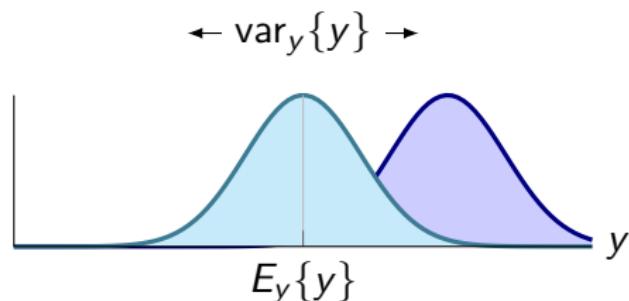
$$E = E_{LS} \left\{ E_y \left\{ (y - \hat{y})^2 \right\} \right\}$$

Bias/variance decomposition (i)

Let us analyse this error in more details:

$$\begin{aligned} & E_{LS} \{ E_y \{ (y - \hat{y})^2 \} \} \\ = & E_{LS} \left\{ E_y \left\{ (y - E_y\{y\} + E_y\{y\} - \hat{y})^2 \right\} \right\} \\ = & E_{LS} \left\{ E_y \left\{ (y - E_y\{y\})^2 \right\} \right\} + E_{LS} \left\{ E_y \left\{ (E_y\{y\} - \hat{y})^2 \right\} \right\} \\ + & E_{LS} \{ E_y \{ 2(y - E_y\{y\})(E_y\{y\} - \hat{y}) \} \} \\ = & E_y \left\{ (y - E_y\{y\})^2 \right\} + E_{LS} \left\{ (E_y\{y\} - \hat{y})^2 \right\} \\ + & E_{LS} \{ 2(E_y\{y\} - E_y\{y\})(E_y\{y\} - \hat{y}) \} \\ = & E_y \left\{ (y - E_y\{y\})^2 \right\} + E_{LS} \left\{ (E_y\{y\} - \hat{y})^2 \right\} \end{aligned}$$

Bias/variance decomposition (ii)



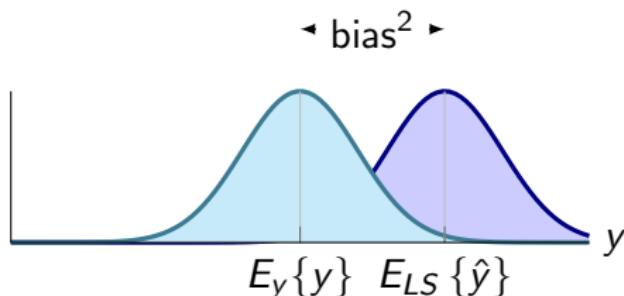
$$E = \underbrace{E_y \left\{ (y - E_y\{y\})^2 \right\}}_{= \text{residual error} = \text{var}_y\{y\}} + E_{LS} \left\{ (E_y\{y\} - \hat{y})^2 \right\}$$

The residual error is the **minimal attainable** error.

Bias/variance decomposition (iii)

$$\begin{aligned} & E_{LS} \left\{ (E_y\{y\} - \hat{y})^2 \right\} \\ = & E_{LS} \left\{ (E_y\{y\} - E_{LS}\{\hat{y}\} + E_{LS}\{\hat{y}\} - \hat{y})^2 \right\} \\ = & E_{LS} \left\{ (E_y\{y\} - E_{LS}\{\hat{y}\})^2 \right\} + E_{LS} \left\{ (E_{LS}\{\hat{y}\} - \hat{y})^2 \right\} \\ + & E_{LS} \{ 2(E_y\{y\} - E_{LS}\{\hat{y}\})(E_{LS}\{\hat{y}\} - \hat{y}) \} \\ = & (E_y\{y\} - E_{LS}\{\hat{y}\})^2 + E_{LS} \left\{ (\hat{y} - E_{LS}\{\hat{y}\})^2 \right\} \\ + & 2(E_y\{y\} - E_{LS}\{\hat{y}\})(E_{LS}\{\hat{y}\} - E_{LS}\{\hat{y}\}) \\ = & (E_y\{y\} - E_{LS}\{\hat{y}\})^2 + E_{LS} \left\{ (\hat{y} - E_{LS}\{\hat{y}\})^2 \right\} \end{aligned}$$

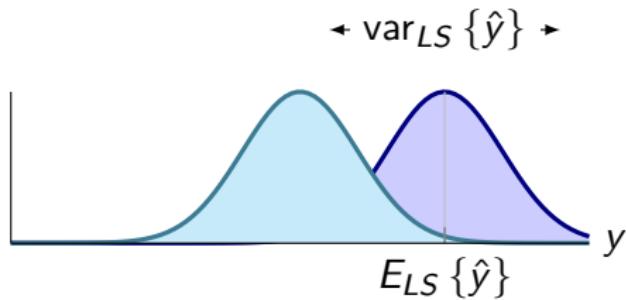
Bias/variance decomposition (iv)



$$E = \text{var}_y\{y\} + \underbrace{(E_y\{y\} - \overbrace{E_{LS}\{\hat{y}\}}^{\text{= average model}})^2}_{\text{= bias}^2} + \dots$$

where the bias is the **difference** between the average model and the Bayes model.

Bias/variance decomposition (v)



$$E = \text{var}_y\{y\} + \text{bias}^2 + \underbrace{E_{LS} \left\{ (\hat{y} - E_{LS} \{\hat{y}\})^2 \right\}}_{= \text{estimation variance} = \text{var}_{LS} \{\hat{y}\}}$$

where $\text{var}_{LS} \{\hat{y}\}$ is a consequence of **overfitting**.

Bias/variance decomposition (vi)

The **expected generalization** error can thus be rewritten as:

$$E = \text{var}_y\{y\} + \text{bias}^2 + \text{var}_{LS}\{\hat{y}\}$$

Application to a simple example (i)

Let us first consider a model \hat{y}_1 computing the average height over the learning sample:

$$\hat{y}_1 = \frac{1}{N} \sum_{i=1}^N y_i$$

One can easily compute:

$$\text{bias}^2 = (E_y\{y\} - E_{LS}\{\hat{y}_1\})^2 = 0$$

$$\text{var}_{LS}\{\hat{y}_1\} = \frac{1}{N} \text{var}_y\{y\}$$

From statistics, \hat{y}_1 is the best estimate with zero bias.

Application to a simple example (ii)

Let us now consider a model where it is known that the height is close to 180:

$$\hat{y}_2 = \frac{\lambda 180 + \sum y_i}{\lambda + N}$$

One can compute:

$$\text{bias}^2 = \left(\frac{\lambda}{\lambda + N} \right)^2 (E_y\{y\} - 180)^2$$
$$\text{var}_{LS}\{\hat{y}_2\} = \frac{N}{(\lambda + N)^2} \text{var}_y\{y\}$$

From these, it can be observed that \hat{y}_1 may not be the best estimator because of its variance. There is a **bias/variance trade-off** with respect to λ .

Bayesian approach (i)

Let us assume that:

- The average height is close to 180cm:

$$P(\bar{y}) = A \exp\left(-\frac{(\bar{y} - 180)^2}{2\sigma_y^2}\right)$$

- The height of one individual is Gaussian around the mean:

$$P(y_i | \bar{y}) = B \exp\left(-\frac{(y_i - \bar{y})^2}{2\sigma_y^2}\right)$$

What is the most probable value of \bar{y} after having seen the learning sample?

$$\hat{y} = \arg \max_{\bar{y}} P(\bar{y} | LS)$$

Bayesian approach (ii)

$$\begin{aligned}\hat{y} &= \arg \max_{\bar{y}} P(\bar{y} \mid LS) \\&= \arg \max_{\bar{y}} P(LS \mid \bar{y})P(\bar{y}) \quad (\text{Bayes theorem and } P(LS) \text{ is constant}) \\&= \arg \max_{\bar{y}} P(y_1, \dots, y_N \mid \bar{y}) P(\bar{y}) \\&= \arg \max_{\bar{y}} \prod_{i=1}^N P(y_i \mid \bar{y}) P(\bar{y}) \quad (\text{independence of learning cases}) \\&= \arg \min_{\bar{y}} - \sum_{i=1}^N \log(P(y_i \mid \bar{y})) - \log(P(\bar{y}))\end{aligned}$$

Bayesian approach (iii)

$$= \arg \min_{\bar{y}} \sum_{i=1}^N \frac{(y_i - \bar{y})^2}{2\sigma_y^2} + \frac{(\bar{y} - 180)^2}{2\sigma_{\bar{y}}^2}$$

= ...

$$= \frac{\lambda 180 + \sum_i y_i}{\lambda + N} \text{ with } \lambda = \frac{\sigma_y^2}{\sigma_{\bar{y}}^2}$$

Extension to a problem with inputs (i)

In this case, $\hat{y}(\underline{x})$ is a function of several inputs \Rightarrow average over the whole input space.

The error becomes:

$$E_{\underline{x},y} \left\{ (y - \hat{y}(\underline{x}))^2 \right\}$$

When averaging over all learning sets:

$$\begin{aligned} E &= E_{LS} \left\{ E_{\underline{x},y} \left\{ (y - \hat{y}(\underline{x}))^2 \right\} \right\} \\ &= E_{\underline{x}} \left\{ E_{LS} \left\{ E_{y|\underline{x}} \left\{ (y - \hat{y}(\underline{x}))^2 \right\} \right\} \right\} \\ &= E_{\underline{x}} \left\{ \text{var}_{y|\underline{x}} \{y\} \right\} + E_{\underline{x}} \left\{ \text{bias}^2(\underline{x}) \right\} + E_{\underline{x}} \left\{ \text{var}_{LS} \{\hat{y}(\underline{x})\} \right\} \end{aligned}$$

Extension to a problem with inputs (ii)

$$E_{LS} \{ E_{y|\underline{x}} \{ (y - \hat{y}(\underline{x}))^2 \} \} = \text{noise}(\underline{x}) + \text{bias}^2(\underline{x}) + \text{variance}(\underline{x})$$

- ▶ $\text{noise}(\underline{x}) = E_{y|\underline{x}} \{ (y - h_B(\underline{x}))^2 \}$:
Quantifies how much y varies from $h_B(\underline{x}) = E_{y|\underline{x}}\{y\}$ (*Bayes model*).
- ▶ $\text{bias}^2(\underline{x}) = (h_B(\underline{x}) - E_{LS} \{ \hat{y}(\underline{x}) \})^2$:
Measures the error between the Bayes model and the average model.
- ▶ $\text{variance}(\underline{x}) = E_{LS} \{ (\hat{y}(\underline{x}) - E_{LS} \{ \hat{y}(\underline{x}) \})^2 \}$:
Quantifies how much $\hat{y}(\underline{x})$ varies from one learning sample to another.

Illustration (i)

Let us consider the following problem:

- A single input x , which is a uniform random variable drawn in $[0, 1]$.
- $y = h(x) + \varepsilon$ where $\varepsilon \sim \mathcal{N}(0, 1)$.

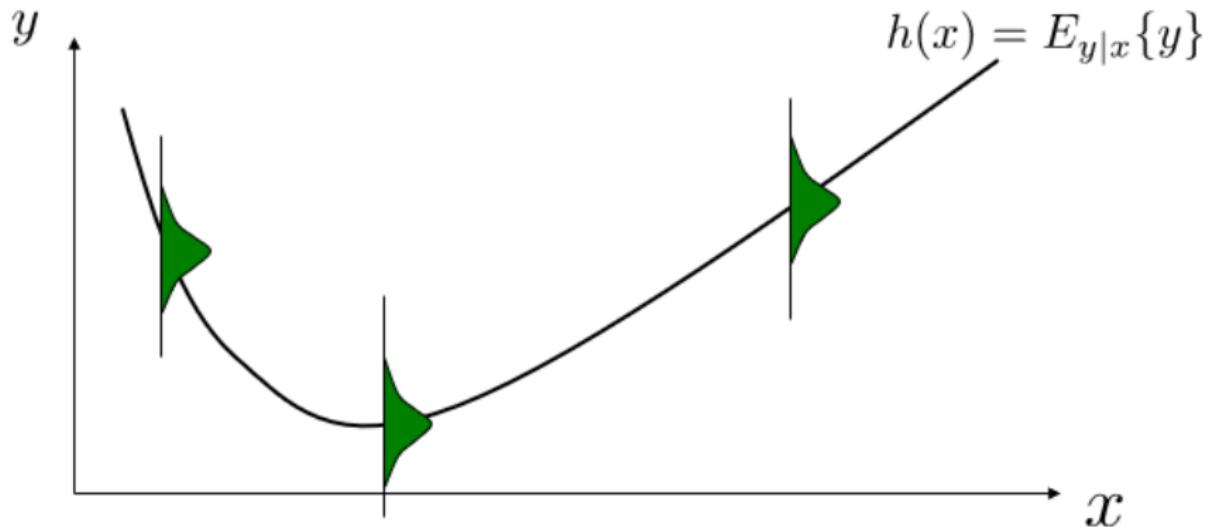


Illustration (ii)

A **low variance** and **high bias** method leads to **underfitting**.

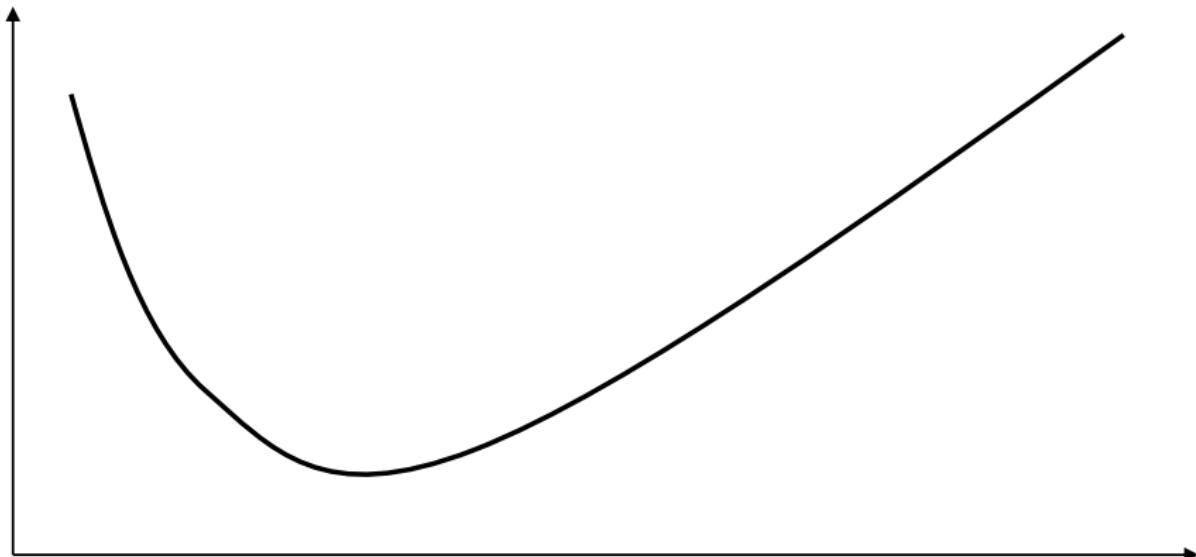


Illustration (ii)

A **low variance** and **high bias** method leads to **underfitting**.

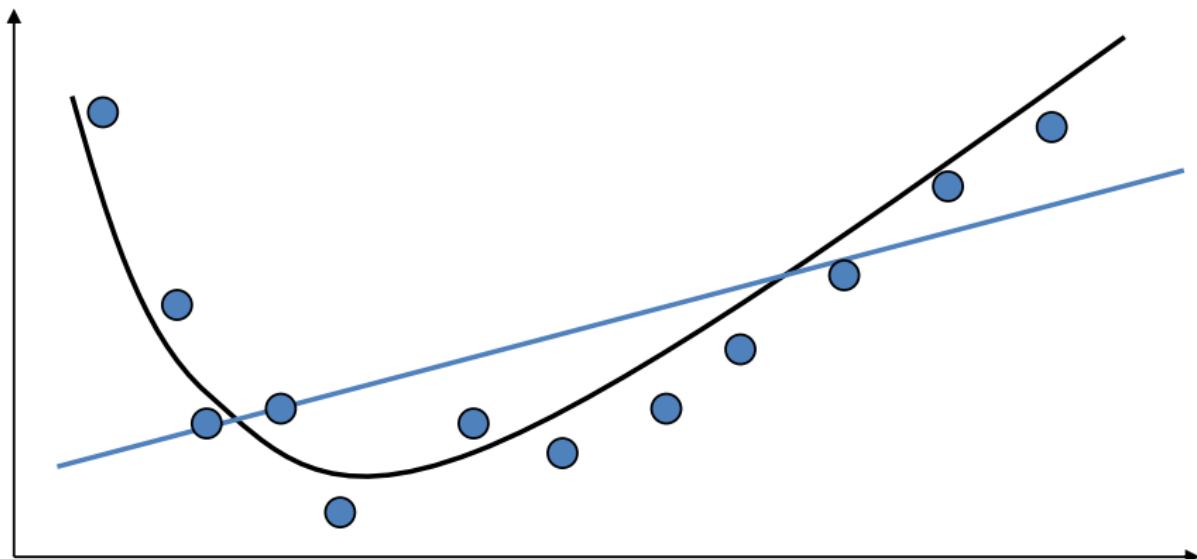


Illustration (ii)

A **low variance** and **high bias** method leads to **underfitting**.

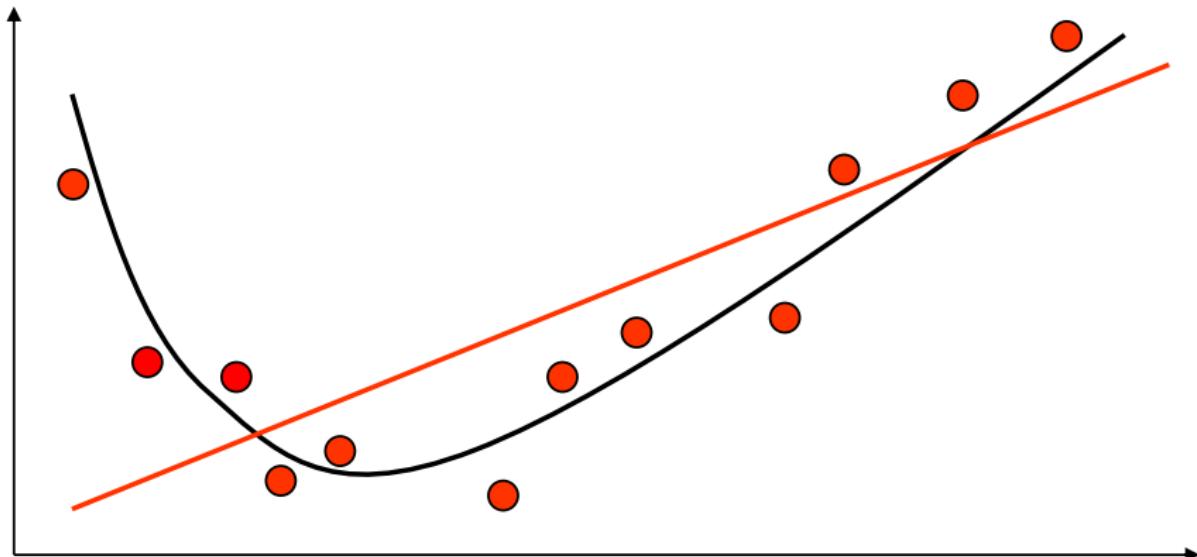


Illustration (ii)

A **low variance** and **high bias** method leads to **underfitting**.

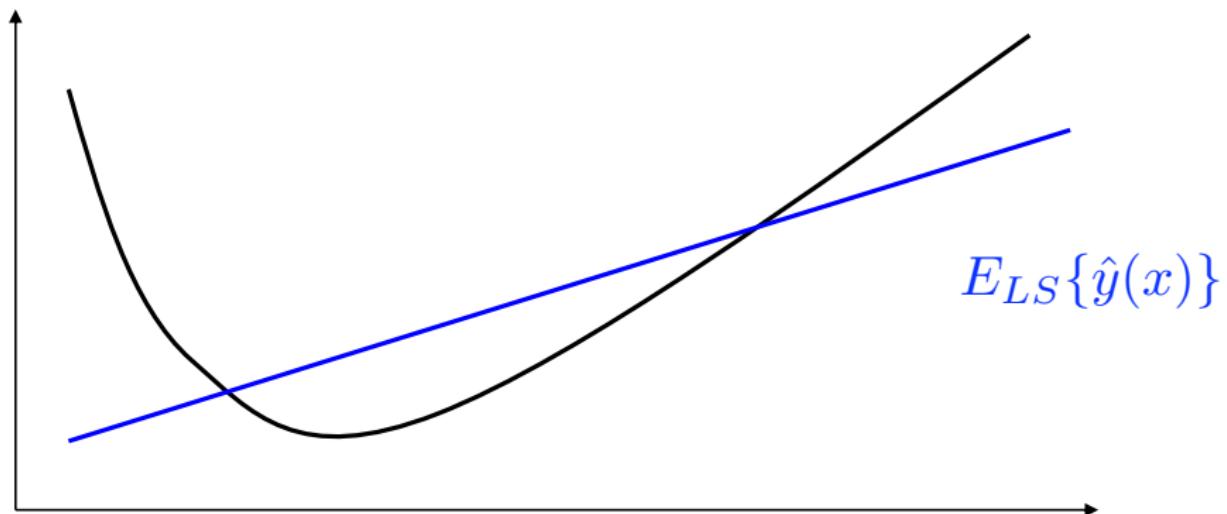


Illustration (ii)

A **low variance** and **high bias** method leads to **underfitting**.

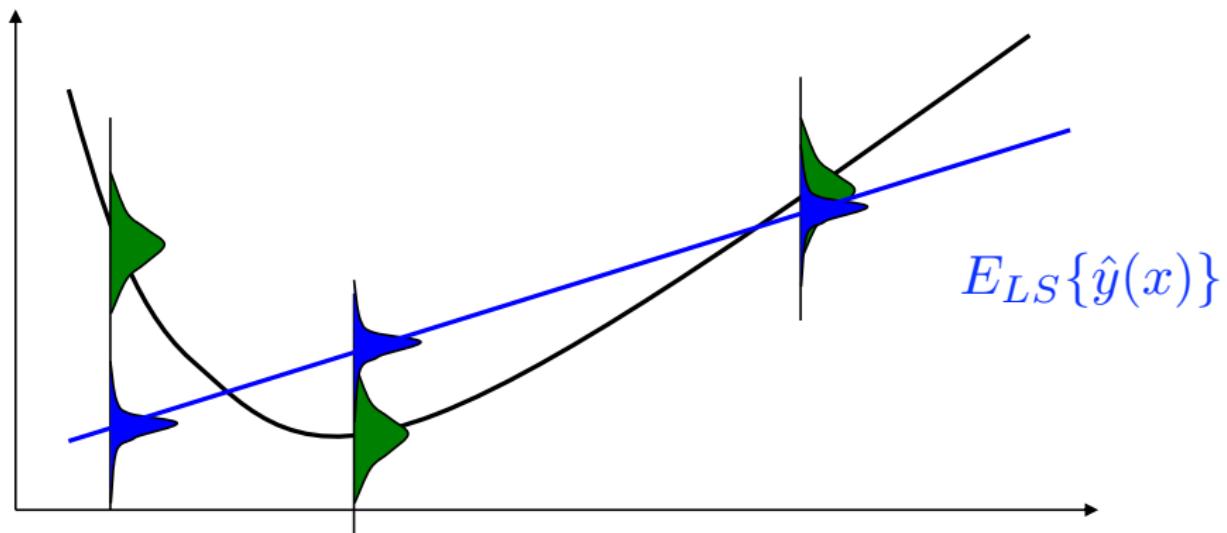


Illustration (iii)

A **low bias** and **high variance** method leads to **overfitting**.

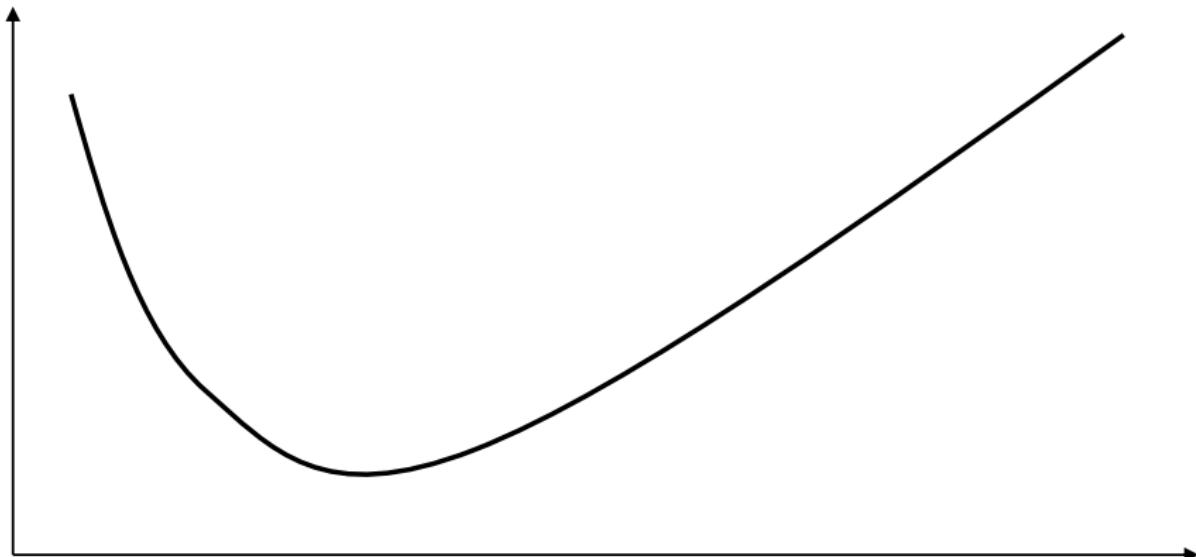


Illustration (iii)

A **low bias** and **high variance** method leads to **overfitting**.

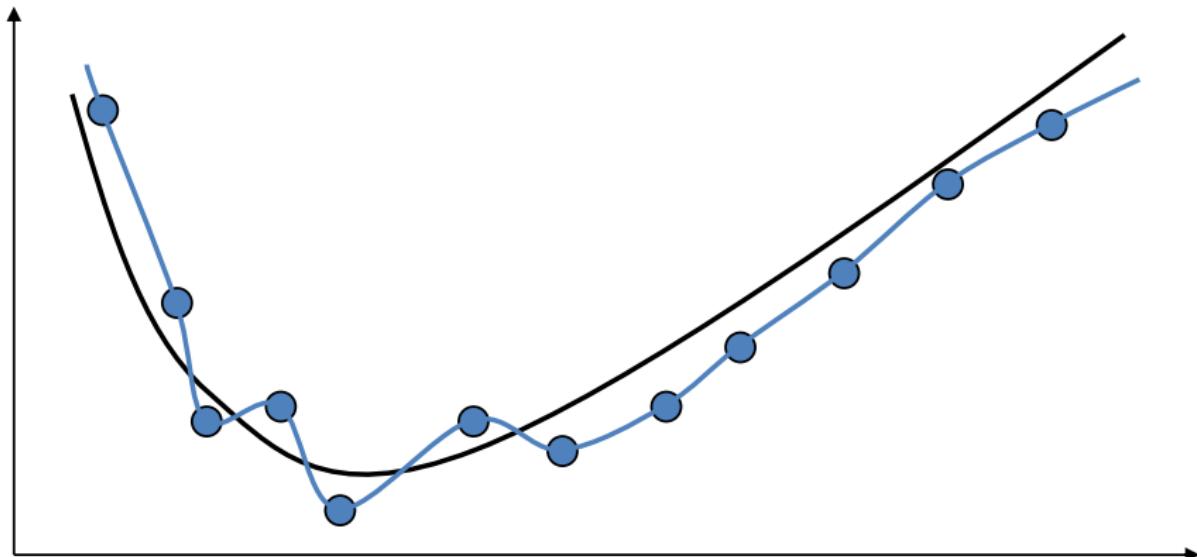


Illustration (iii)

A **low bias** and **high variance** method leads to **overfitting**.

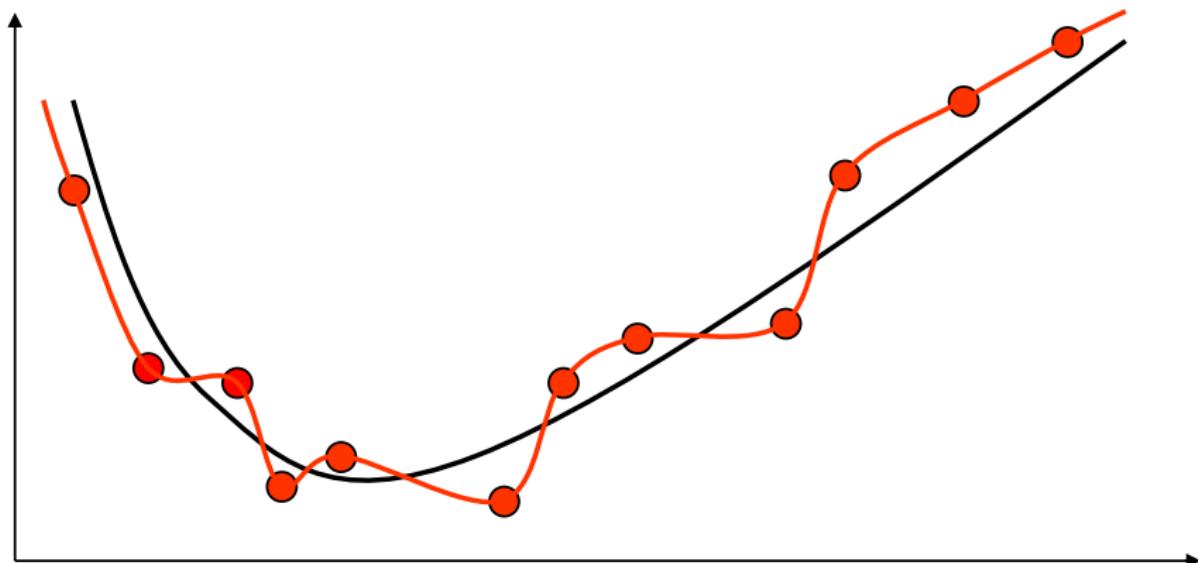


Illustration (iii)

A **low bias** and **high variance** method leads to **overfitting**.

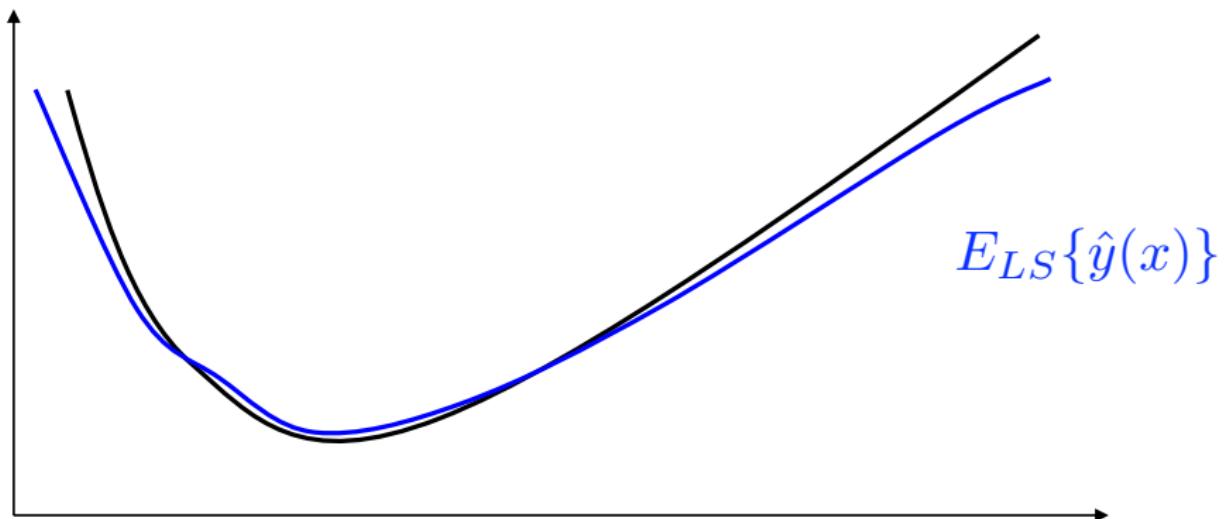


Illustration (iii)

A **low bias** and **high variance** method leads to **overfitting**.

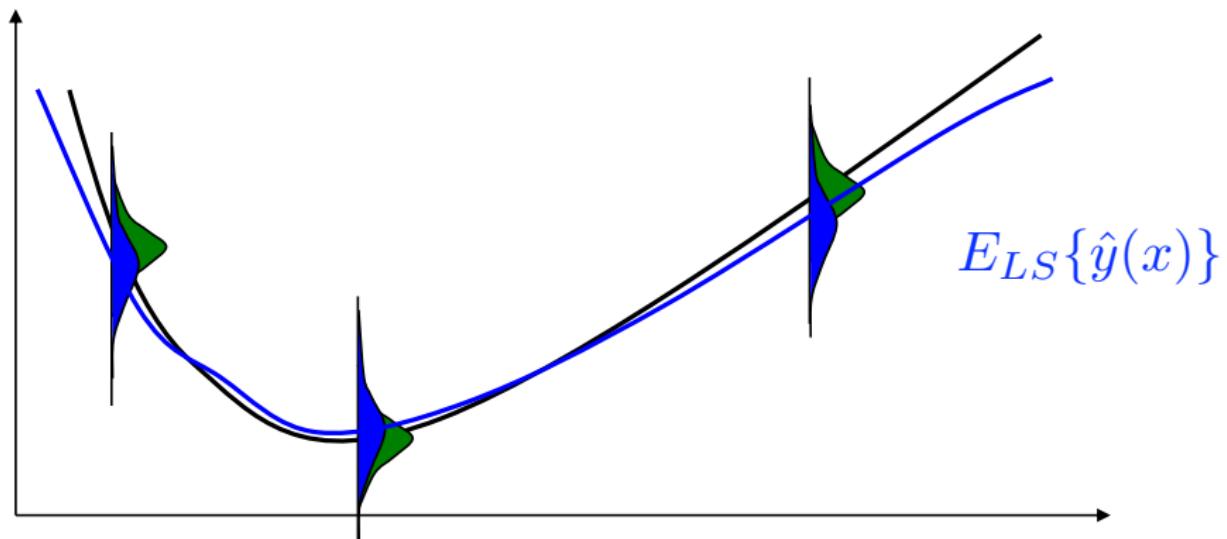


Illustration (iv)

No noise doesn't imply no variance, rather less variance.

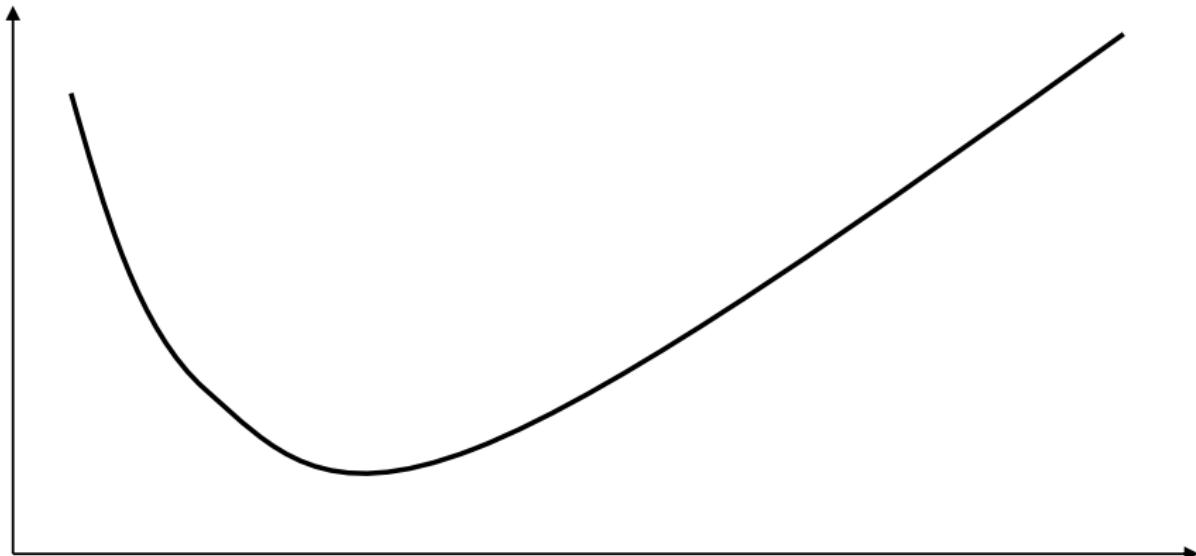


Illustration (iv)

No noise doesn't imply no variance, rather less variance.

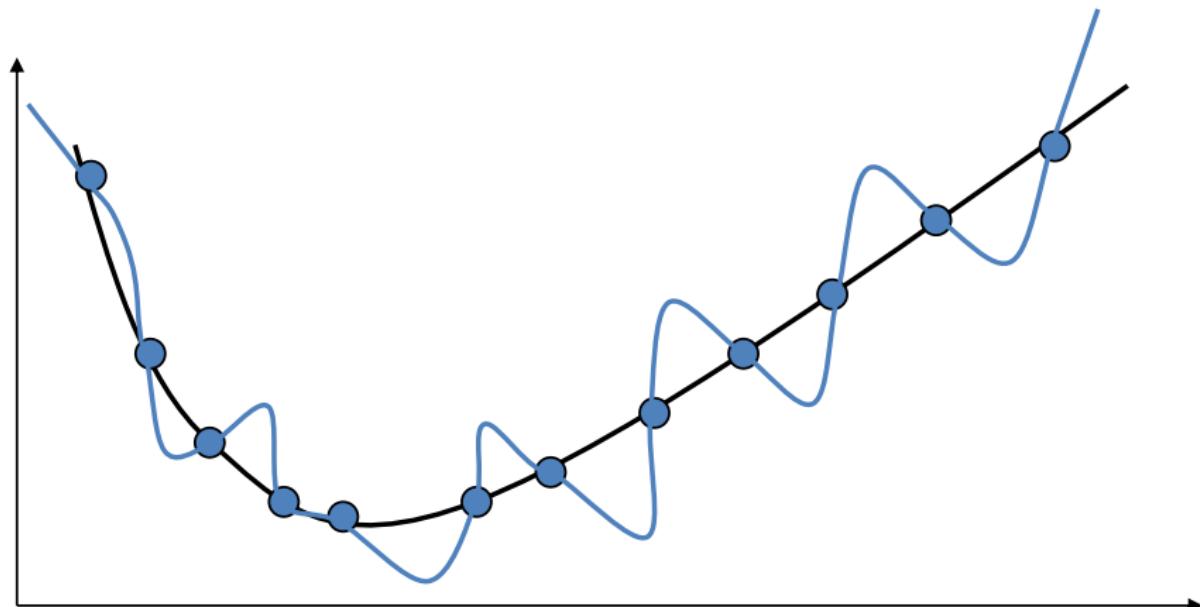
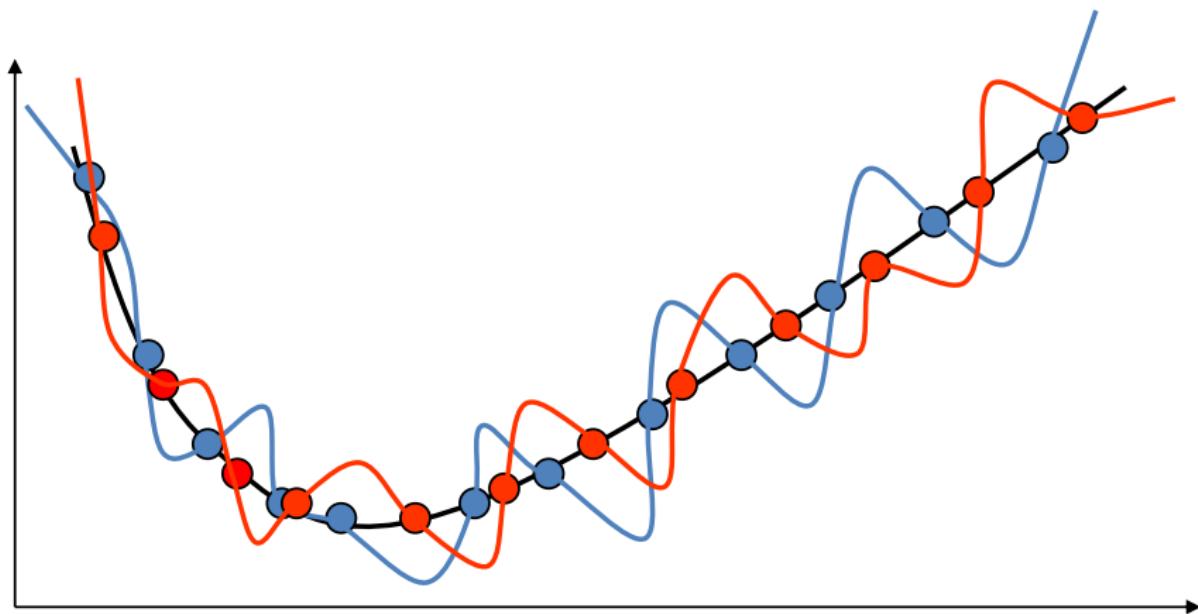


Illustration (iv)

No noise doesn't imply no variance, rather less variance.



Bias/variance trade-off in classification problems (i)

The mean misclassification error corresponds to:

$$E = E_{LS} \{ E_{\underline{x},y} \{ 1(y \neq \hat{y}(\underline{x})) \} \}$$

The best possible model is the **Bayes model**:

$$h_B(\underline{x}) = \arg \max_c P(y = c | \underline{x})$$

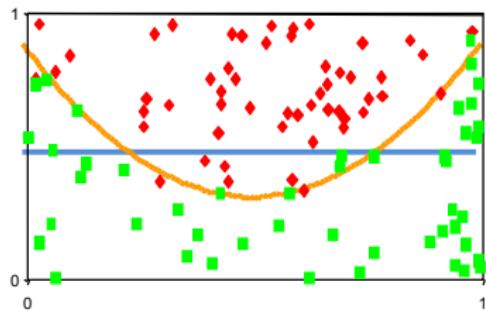
The “average” model is:

$$\arg \max_c P(\hat{y}(\underline{x} = c | \underline{x}))$$

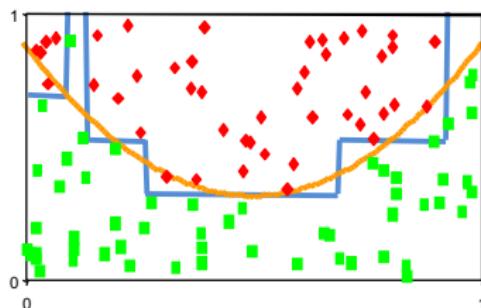
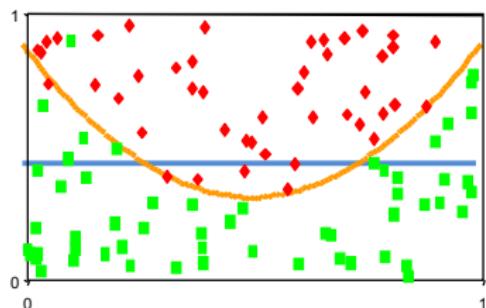
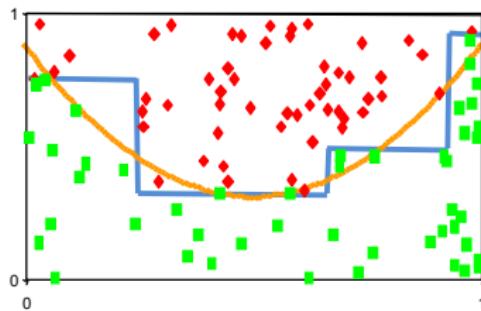
Unfortunately, there is no such decomposition of the mean misclassification error into a bias and variance terms. Nevertheless, the same phenomena can be observed.

Bias/variance trade-off in classification problems (ii)

Single test node

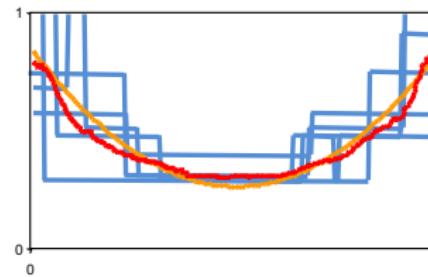
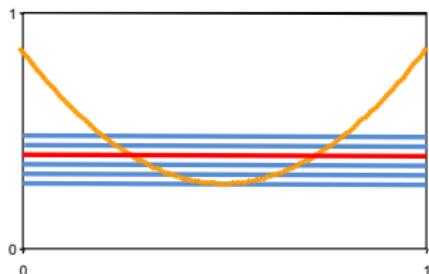


Fully grown tree



Decision tree classifiers on two different data sets (top and bottom).

Bias/variance trade-off in classification problems (iii)



The **bias** is a **systematic** error component, which is independent of the learning sample. Rather, it depends on the **learning algorithm**.

The **variance** is the error due to the **variability** of the model with respect to the **learning sample** randomness.

There are errors due to bias and errors due to variance.

Parameters that influence bias and variance - Outline

- ▶ Complexity of the model
- ▶ Complexity of the Bayes model
- ▶ Noise
- ▶ Learning sample size
- ▶ Learning algorithm

Illustrative problem

Let us consider the following artificial problem:

- 10 inputs, all uniform random variables drawn in $[0, 1]$.
- The true function depends only on 5 inputs:

$$y(\underline{x}) = 10 \sin(\pi x_1 x_2) + 20(x_3 - 0.5)^2 + 10x_4 + 6x_5 + \varepsilon$$

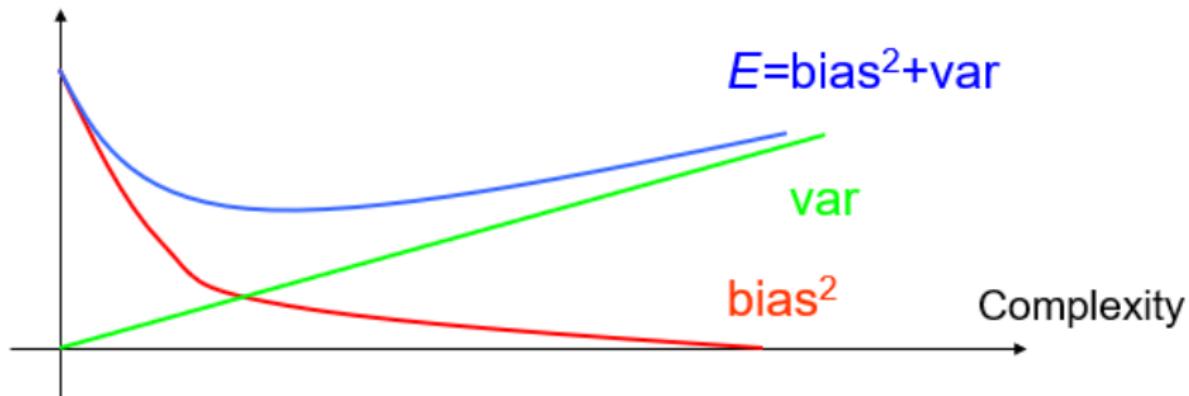
where ε is a random variable drawn in $\mathcal{N}(0, 1)$.

The following experimentations are conducted:

1. $E_{LS} \rightarrow$ Average over 50 learning sets of size 500.
2. $E_{\underline{x},y} \rightarrow$ Average over 2000 cases

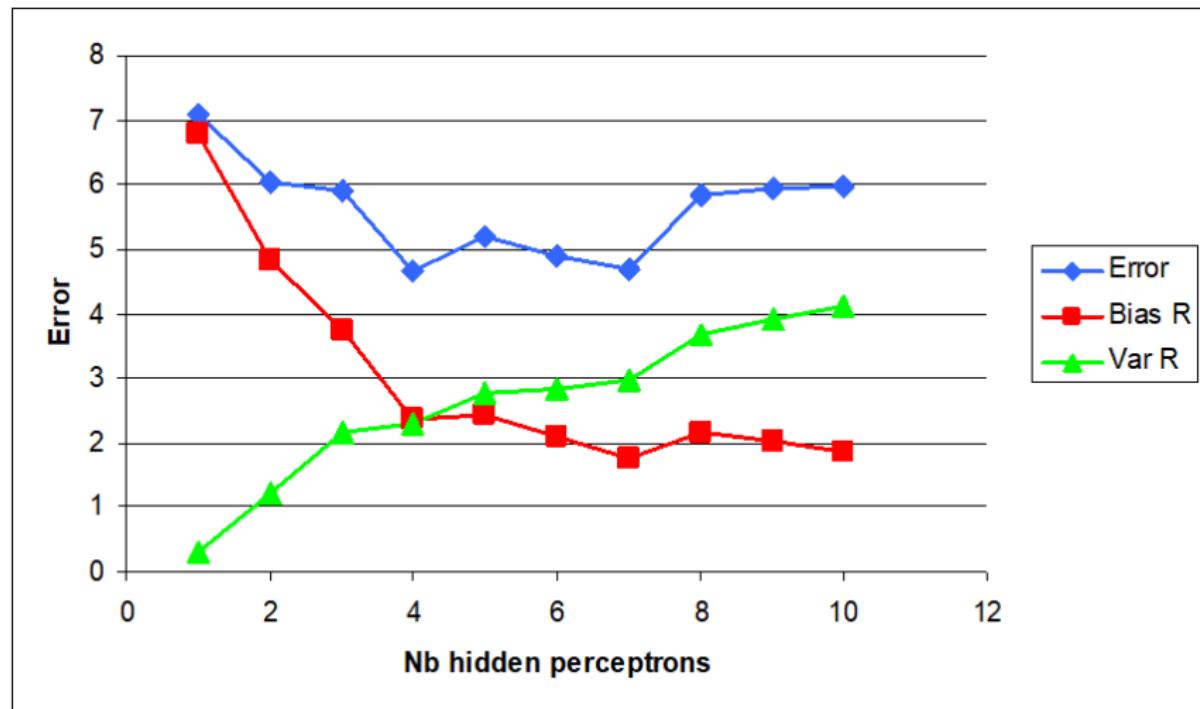
\Rightarrow Estimate bias and variance, as well as the residual error.

Complexity of the model



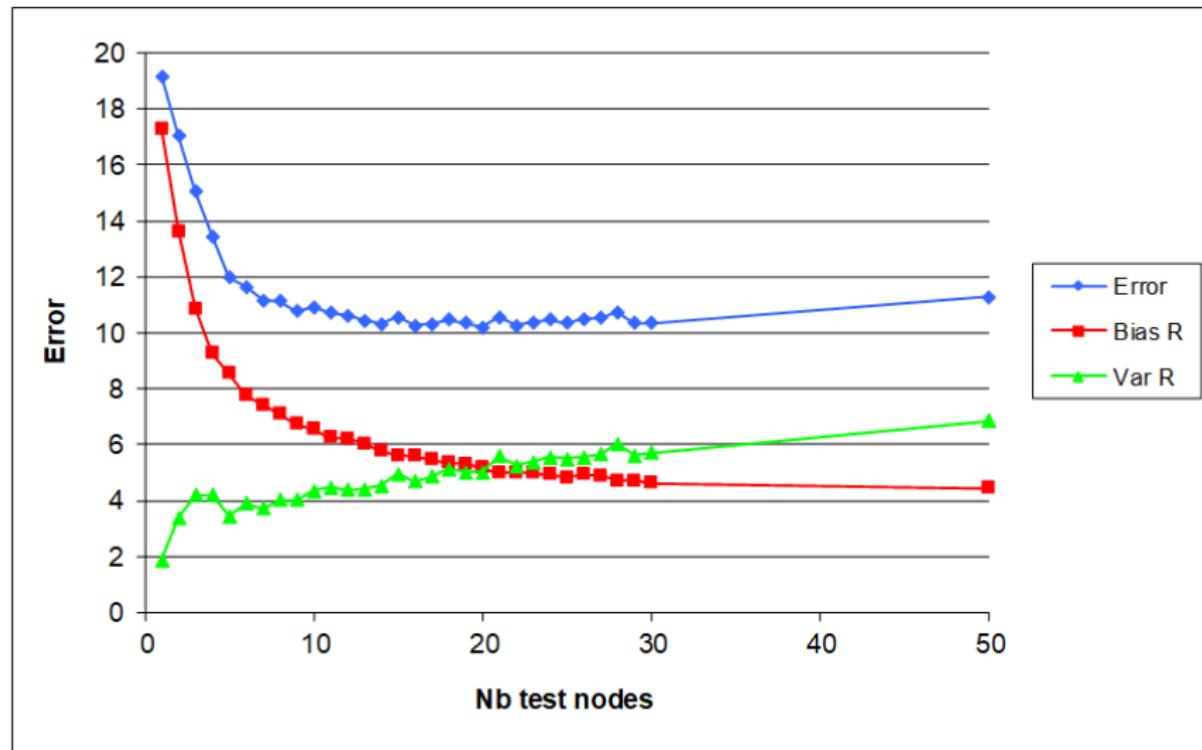
Usually, the bias is a decreasing function of the complexity, while the variance is an increasing function of the complexity.
Note: the residual error is included in the bias term.

Complexity of the model - Neural networks



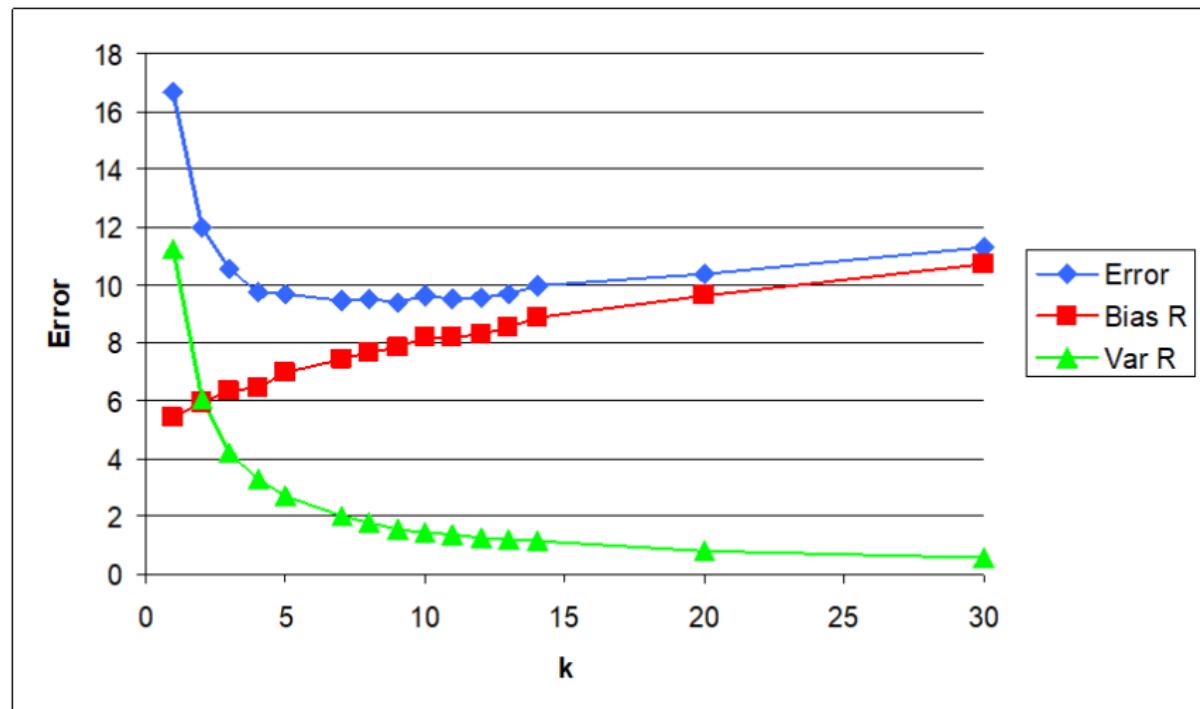
Error, bias and variance w.r.t. the number of neurons in the hidden layer.

Complexity of the model - Regression trees



Error, bias and variance w.r.t. the number of test nodes.

Complexity of the model - kNN



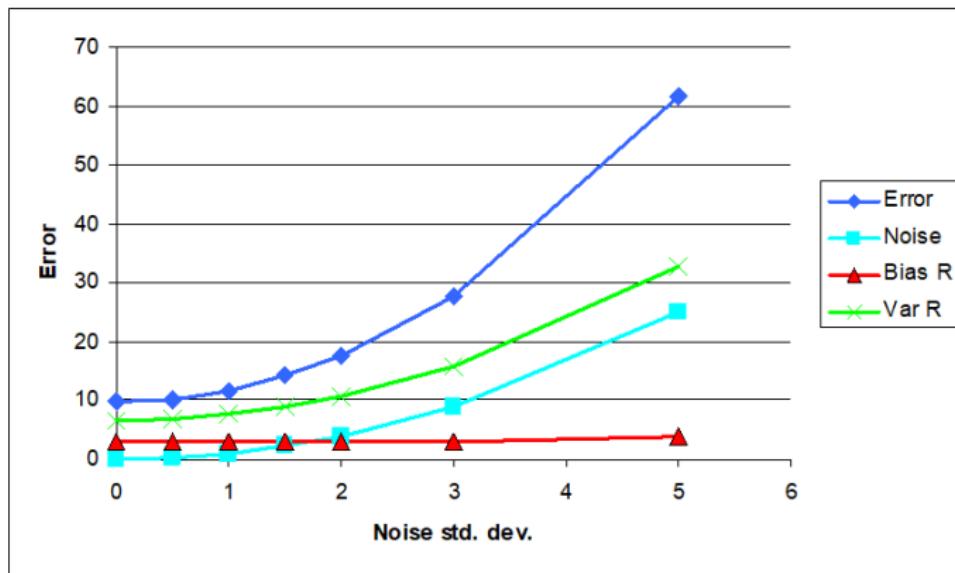
Error, bias and variance w.r.t. the number of neighbors.

Complexity of the Bayes model

At fixed model complexity, the bias **increases** with the complexity of the Bayes model. Nevertheless, the effect on variance is difficult to predict.

Noise

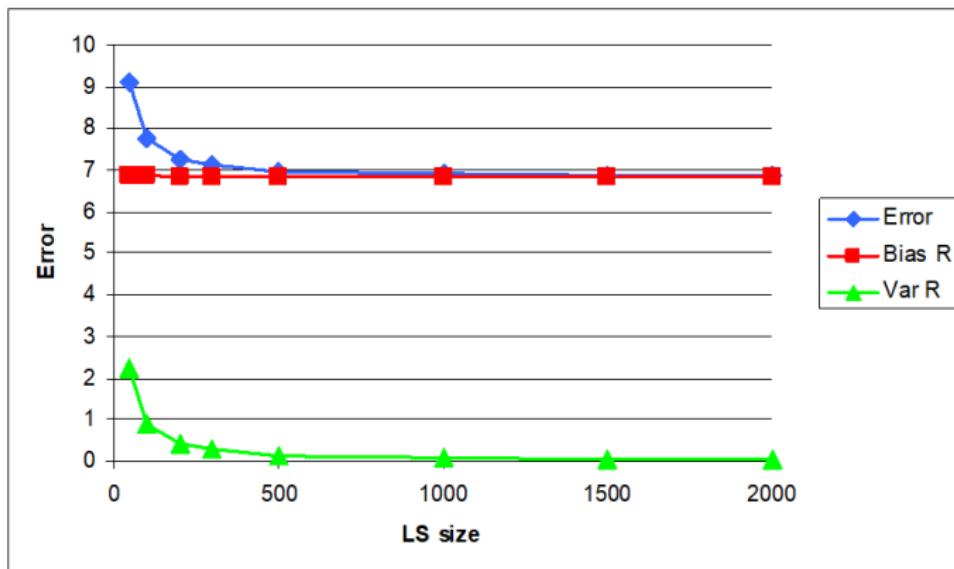
Variance **increases** with noise while bias remains mainly unaffected.



Influence of noise on full regression trees.

Learning sample size (i)

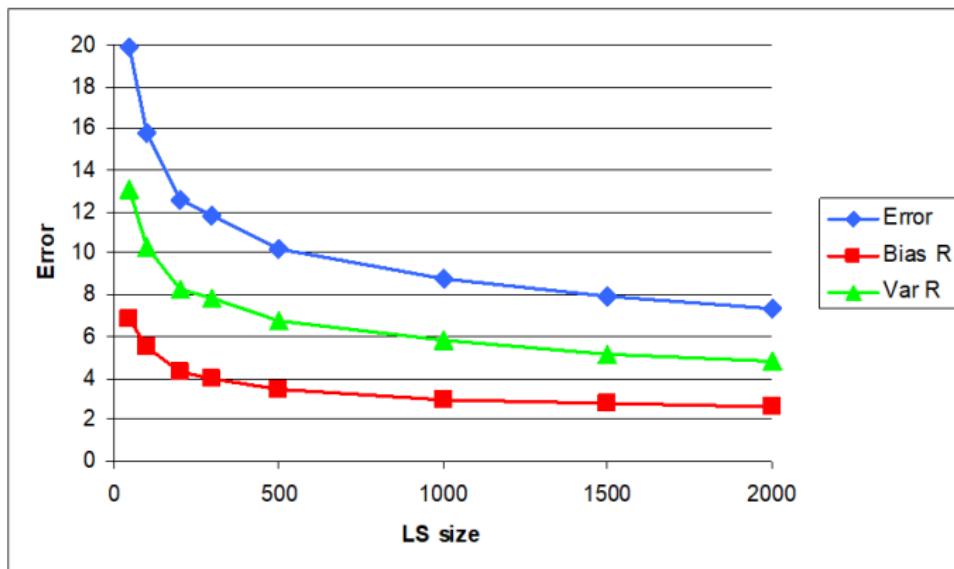
At fixed model complexity, bias remains **constant** and variance **decreases** with the learning sample size.



Influence of the learning sample size on linear regression.

Learning sample size (ii)

When the complexity of the model is dependent on the learning sample size, **both** bias and variance **decrease** with the learning sample size.



Influence of the learning sample size on regression trees.

Learning algorithms - Linear regression

Method	E^2	bias 2 + noise	var
Linear regression	7.0	6.8	0.2
kNN ($k = 1$)	15.4	5.0	10.4
kNN ($k = 10$)	8.5	7.2	1.3
MLP (10)	2.0	1.2	0.8
MLP (10 - 10)	4.6	1.4	3.2
Regression tree	10.2	3.5	6.7

Linear regression has few parameters. Therefore, it has a small variance. However, since the true function is non-linear, it has a high bias.

Learning algorithms - kNN

Method	E^2	bias ² + noise	var
Linear regression	7.0	6.8	0.2
kNN ($k = 1$)	15.4	5.0	10.4
kNN ($k = 10$)	8.5	7.2	1.3
MLP (10)	2.0	1.2	0.8
MLP (10 - 10)	4.6	1.4	3.2
Regression tree	10.2	3.5	6.7

For small values of k , there is a high variance and a moderate bias. For high values of k , the variance decreases but bias increases.

Learning algorithms - MLP

Method	E^2	bias ² + noise	var
Linear regression	7.0	6.8	0.2
kNN ($k = 1$)	15.4	5.0	10.4
kNN ($k = 10$)	8.5	7.2	1.3
MLP (10)	2.0	1.2	0.8
MLP (10 - 10)	4.6	1.4	3.2
Regression tree	10.2	3.5	6.7

In both cases, the bias is low. However, variance increases with the model complexity.

Learning algorithms - Regression tree

Method	E^2	bias 2 + noise	var
Linear regression	7.0	6.8	0.2
kNN ($k = 1$)	15.4	5.0	10.4
kNN ($k = 10$)	8.5	7.2	1.3
MLP (10)	2.0	1.2	0.8
MLP (10 - 10)	4.6	1.4	3.2
Regression tree	10.2	3.5	6.7

Regression trees have a small bias. Indeed, a complex enough tree can approximate any non-linear function. However, it has a high variance.

Bias and variance reduction techniques - Outline

- ▶ Introduction
- ▶ Dealing with the bias/variance trade-off of one algorithm
- ▶ Ensemble methods

Bias and variance reduction techniques

In the context of a given method, adapting the learning algorithm to find the best trade-off between bias and variance is a solution, but not a panacea.

Examples: pruning, weight decay.

Ensemble methods change the bias/variance trade-off, but destroy some features of the original method.

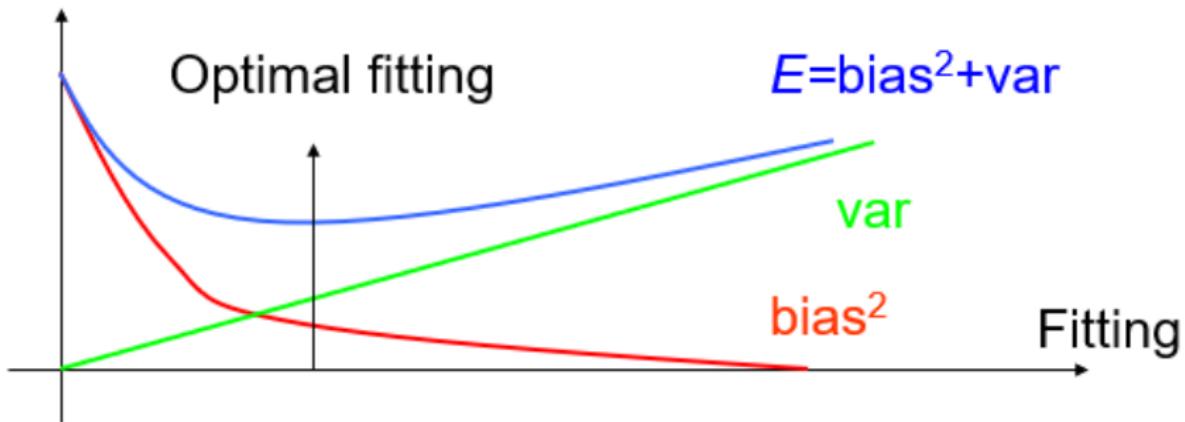
Examples: bagging, boosting.

Variance reduction for a given model (i)

Idea: reduce the ability of the learning algorithm to fit the learning sample.

- ▶ Pruning: reduces the model complexity explicitly
- ▶ Early stopping: reduces the amount of search
- ▶ Regularization: reduces the size of the hypothesis space.
Example: weight decay in neural networks consists in penalizing high weight values.

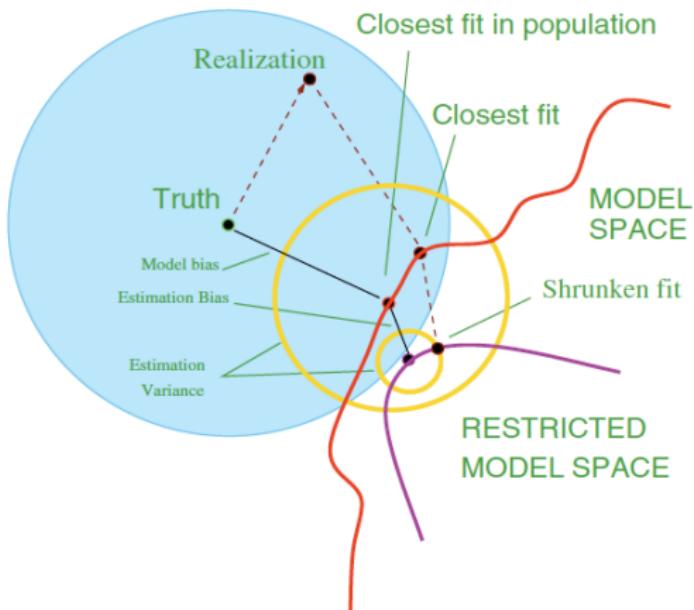
Variance reduction for a given model (ii)



The selection of the optimal level of fitting can be done:

- a priori (**not optimal**)
- by cross-validation (less efficient):
 - $\text{bias}^2 \approx$ error on the learning set.
 - $E \approx$ error on an independent test set.

Variance reduction for a given model (iii)



Source: [1]

Variance reduction for a given model (iv)

Examples:

- Post-pruning of regression trees.
- Early stopping of MLP by cross-validation.

Method	E	Bias	Variance
Full regression tree (250)	10.2	3.5	6.7
Pruned regression tree (45)	9.1	4.3	4.8
Full learned MLP	4.6	1.4	3.2
Early stopped MLP	3.8	1.5	2.3

As expected, variance decreases but bias increases.

Ensemble methods (i)

Ensemble methods combine the predictions of several models built with a learning algorithm in order to improve with respect to the use of a single model.

There are two main families:

1. **Averaging techniques**: they grow several models independently and average their predictions. They mainly decrease **variance**.
Examples: bagging, random forests.
2. **Boosting type algorithms**: they grow several models sequentially. They mainly decrease **bias**.
Examples: adaboost, MART.

Ensemble methods (ii)

Examples: bagging, boosting, random forests.

Method	E	Bias	Variance
Full regression tree	10.2	3.5	6.7
Bagging	5.3	3.8	1.5
Random forests	4.9	4.0	0.9
Boosting	5.0	3.1	1.9

Discussion

The notions of bias and variance are very useful to predict how changing the (learning and problem) parameters will affect accuracy. This explains why very simple methods can work much better than more complex ones on very difficult tasks.

Variance reduction is a very important topic: reducing bias is easy, while keeping variance low is not as easy. It is especially important when machine learning is applied in domains that require complex models: time series analysis, computer vision, natural language processing, . . .

All learning algorithms are **not** equal in terms of variance. Trees are among the worse methods according to this criterion.

Outline

- ① Bias/variance trade-off
- ② Performance evaluation
 - Model assessment and selection
 - Cross-validation
 - Bootstrap
 - CV-based model selection
- ③ Performance measures
- ④ Further reading

Estimating the performance of a model

Given a model learned from some data set of size N , how to estimate its performance from this data set?

What for?

- ▶ Model selection: choosing the best model among several models.
Example: determining the right complexity of a model or choosing between different learning algorithms.
- ▶ Model assessment: having chosen a final model, it consists in estimating its performance on new data.

Large data sets: test set method

Idea: randomly divide the data set into two parts: a **learning set** and a **test set**.

Example: 70% – 30%



Method:

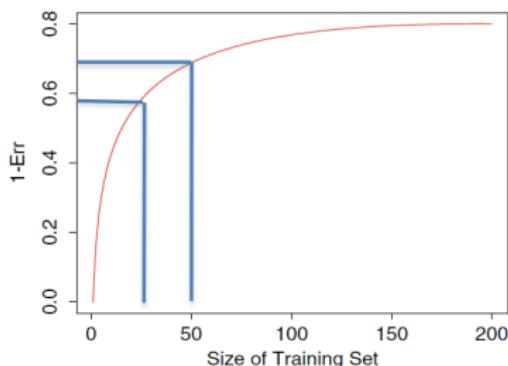
1. Fit the model on the learning set
2. Test it on the test set

The resulting estimate is an estimate of the error of a model learned on the **whole** data set.

Small data sets

In this case, the test set error is **unreliable** because it is based on a small sample: 30% of an already small data set.

It is also pessimistically biased as an estimate of the error of a model built on the whole data set: for small sample sizes, a model learned on 70% of the data is significantly less good than a model learned on the whole data.



Learning curve: performance vs learning set size

k-fold cross-validation

Idea: randomly divide the data set into k subsets (e.g. $k = 10$).



Method:

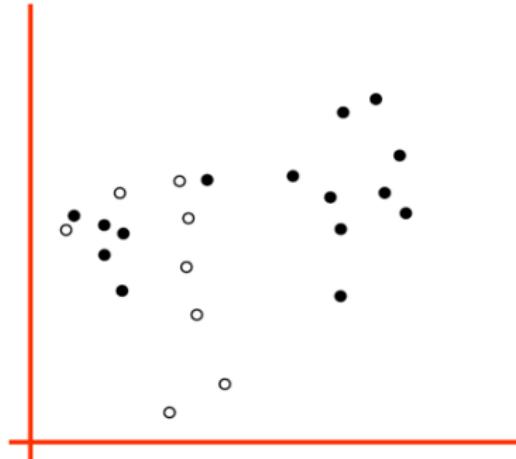
- For each subset:
 1. Learn the model on the objects that are not in the subset.
 2. Compute a prediction with this model for the points in the subset.
- Report the mean error over these predictions.

When $k = N$, the method is called **leave-one-out** cross-validation.

Choosing a value for k

- ▶ $k = N$:
 - Unbiased: removing one object does not change much the size of the learning sample.
 - High variance: highly data set dependent.
 - Slow: requires to train N models.
- ▶ $k = 5, 10$:
 - Lower variance and faster: only 5 – 10 models on fewer data.
 - Potentially biased: see learning curve.

Exercise



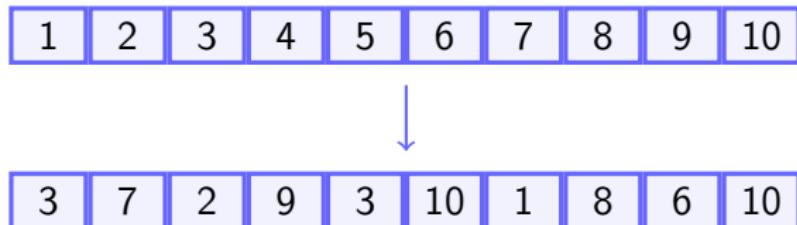
Source: [2]

In this classification problem
with two inputs:

- ▶ What is the resubstitution error (LS error) of 1-NN?
- ▶ What is the LOO error of 1-NN?
- ▶ What is the LOO error of 3-NN?
- ▶ What is the LOO error of 22-NN?

Bootstrap (i)

Bootstrap sampling consists in sampling **with replacement**.



Some objects do not appear and some others appear several times:

$$P(o_i \in \text{bootstrap}) = 1 - (1 - \frac{1}{N})^N \approx 1 - \frac{1}{e} = 0.632$$

Bootstrap (ii)

Let us define the **bootstrap error estimate** method:

- For $i = 1$ to B :
 1. Take a bootstrap sample B_i from the data set.
 2. Learn a model f_i on it.
- For each object, compute the expected error of all models that were built without it (about 30%).
- Average over all objects.

Some improvements:

- ▶ “.632 bootstrap”, which corrects for the learning curve.
- ▶ “.632+ bootstrap”, which corrects for overfitting.

Conditional vs Expected test errors

Conditional test error (for a given model \hat{f}_{LS}):

$$\text{Err}_{LS} = E_{x,y} \left\{ L \left(y, \hat{f}_{LS}(x) \right) \right\}$$

Expected test error:

$$E_{LS} \{ \text{Err}_{LS} \} = E_{LS} \left\{ E_{x,y} \left\{ L \left(y, \hat{f}_{LS}(x) \right) \right\} \right\}$$

Only the test set method estimates the first error. Cross-validation estimates the second one (even leave-one-out).

Model selection: typical scenario

Given a data set of N objects (input-output pairs), how to best exploit this data set to obtain:

- ▶ The best possible model (e.g. among regression trees and k-NN)
→ **model selection**
- ▶ An estimate of its prediction error → **model assessment**

Again, the solution depends on the size N of the data set.

Large data sets: test set method

Idea: randomly divide the data set into 3 parts:

1. A learning set LS
2. A validation set VS
3. A test set TS

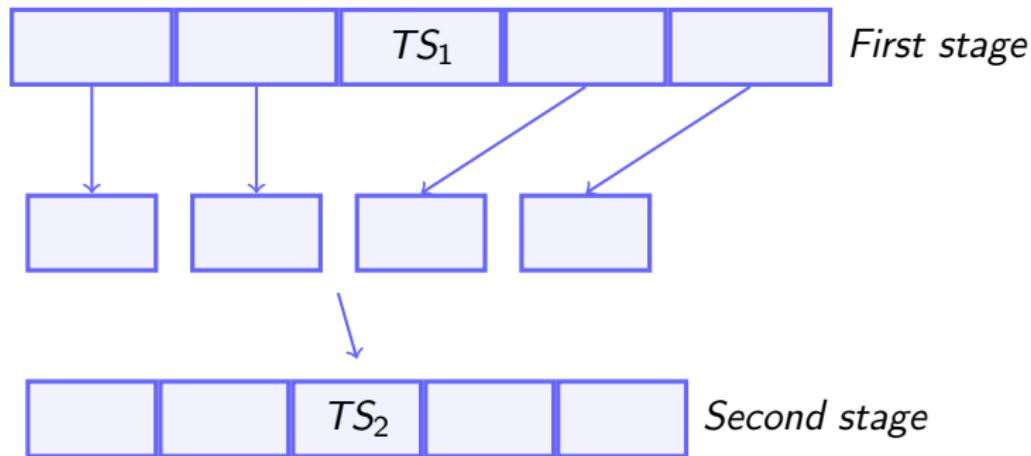
Example: 50% – 25% – 25%



1. Fit the models to compare on the learning set, using different algorithms or different complexity values.
2. Select the best one based on its performance on the validation set.
3. Retrain this model on $LS + VS$.
4. Test it on the test set → **performance estimate**.
5. Retrain this model on $LS + VS + TS$. This yields the finally chosen model.

Small data sets: cross-validation

Idea: use two stages of k -fold cross-validation.



The **first** stage is used for the **assessment** of the final model, while the **second** one is used for **model selection**.

Note: we could also combine test set and cross-validation.

Why do we need a validation set/second stage? (i)

How well does the second stage (resp. validation set) error estimate the true error?

When comparing many complex models, the probability of finding a good only one by chance is high.

Therefore, this estimated error is expected to be **overly optimistic**.

Why do we need a validation set/second stage? (ii)

Let us consider the following example:

- $N = 50$
- 1 000 input variables
- Input variables are unrelated to the class. Their values are i.i.d. from $\mathcal{N}(0, 1)$.

⇒ any model should have a 50% generalization error rate.

Let us now compare 1 000 learning algorithms: the i -th algorithm learns a decision tree on the i -th feature only. For each of them, the 10-fold cross-validation error is computed.

- 10-fold cross-validation error of the best model $\approx 16\%$
- Its error on a test sample of 5 000 cases $\approx 48\%$

Why do we need a validation set/second stage? (iii)

This is the idea behind **selection bias**.

General rule: any choice made using the output should be inside a cross-validation loop.

Let us consider another example on the same data set:

- ▶ Select the 10 attributes that are the most correlated with the output in the learning set.
- ▶ Estimate the error rate of a tree built with these 10 attributes using 10-fold cross-validation on the same learning set: 20%
- ▶ Estimate the error of this model on a sample of 5 000 cases: 51%

Analytical methods for model selection

Idea: find the model that minimizes a criterion, typically of the form:

$$\text{Err(LS)} + G(\text{Complexity})$$

where G is a monotonically increasing function.

The criterion is derived from theoretical arguments.

Example: the minimum description length approach is motivated from coding theory.

Advantage:

- Cheap: no need for retraining

Drawbacks:

- OK for model selection but not for model assessment.
- May miss the true optimum in the finite sample case.

Outline

- ① Bias/variance trade-off
- ② Performance evaluation
- ③ Performance measures
 - Classification
 - Regression
 - Loss functions for learning
- ④ Further reading

Performance criteria

True class	Model 1	Model 2
1 Negative	Positive	Negative
2 Negative	Negative	Negative
3 Negative	Positive	Positive
4 Negative	Positive	Negative
5 Negative	Negative	Negative
6 Negative	Negative	Negative
7 Negative	Negative	Positive
8 Negative	Negative	Negative
9 Negative	Negative	Negative
10 Positive	Positive	Positive
11 Positive	Positive	Negative
12 Positive	Positive	Positive
13 Positive	Positive	Positive
14 Positive	Negative	Negative
15 Positive	Positive	Negative

Which of these two models is the best? The choice of an error or quality measure is highly **application-dependent**.

Binary classification (i)

Results can be summarized in a **contingency table**, also called confusion matrix.

Actual class	Predicted class		Total
	Positive	Negative	
Positive	True Positive	False Negative	P
Negative	False Positive	True Negative	N

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

$$\rightarrow \text{Accuracy} = \frac{TP+TN}{P+N} = 1 - \text{Error rate}$$

Binary classification (ii)

The simplest criterion to compare model performances is the error rate or the accuracy:

Example:

Actual class	Predicted class		Total	Actual class	Predicted class		Total
	Positive	Negative			Positive	Negative	
Positive	5	1	6	Positive	3	3	6
Negative	3	6	9	Negative	2	7	9

Model 1

$$\rightarrow \text{Error rate} = \frac{4}{15} = 27\%$$
$$\rightarrow \text{Accuracy} = \frac{11}{15} = 73\%$$

Model 2

$$\rightarrow \text{Error rate} = \frac{5}{15} = 33\%$$
$$\rightarrow \text{Accuracy} = \frac{10}{15} = 66\%$$

Limitations of the error rate

Actual class	Predicted class		Total
	Positive	Negative	
Positive	0	10	P
Negative	0	90	N

Model 1

Error rate = 10%

Actual class	Predicted class		Total
	Positive	Negative	
Positive	10	0	P
Negative	10	80	N

Model 2

Error rate = 10%

Actual class	Predicted class		Total
	Positive	Negative	
Positive	0	50	P
Negative	0	50	N

Model 3

Error rate = 50%

This criterion does not convey any information about the error distribution across classes: the first two models have the same error rates but different error distributions. It is also sensitive to changes in the class distribution in the test sample.

Sensitivity and specificity (i)

For medical diagnosis, more appropriate measures are:

- Sensitivity (or recall) = $\frac{TP}{P}$
- Specificity = $\frac{TN}{TN+FP} = 1 - \frac{FP}{N}$

Sensitivity and specificity (ii)

Actual class	Predicted class		Total
	Positive	Negative	
Positive	0	10	10
Negative	0	90	90

Model 1

Actual class	Predicted class		Total
	Positive	Negative	
Positive	10	0	10
Negative	10	80	90

Model 2

$$\begin{aligned}\rightarrow \text{Error rate} &= 10\% \\ \rightarrow \text{Sensitivity} &= \frac{0}{10} = 0\% \\ \rightarrow \text{Specificity} &= \frac{90}{90} = 100\%\end{aligned}$$

$$\begin{aligned}\rightarrow \text{Error rate} &= 10\% \\ \rightarrow \text{Sensitivity} &= \frac{10}{10} = 100\% \\ \rightarrow \text{Specificity} &= \frac{80}{90} = 89\%\end{aligned}$$

Actual class	Predicted class		Total
	Positive	Negative	
Positive	0	50	P
Negative	0	50	N

Model 3

$$\begin{aligned}\rightarrow \text{Error rate} &= 50\% \\ \rightarrow \text{Sensitivity} &= \frac{0}{50} = 0\% \\ \rightarrow \text{Specificity} &= \frac{50}{50} = 100\%\end{aligned}$$

Sensitivity and specificity (iii)

True class	Model 1	Model 2
1 Negative	Positive	Negative
2 Negative	Negative	Negative
3 Negative	Positive	Positive
4 Negative	Positive	Negative
5 Negative	Negative	Negative
6 Negative	Negative	Negative
7 Negative	Negative	Positive
8 Negative	Negative	Negative
9 Negative	Negative	Negative
10 Positive	Positive	Positive
11 Positive	Positive	Negative
12 Positive	Positive	Positive
13 Positive	Positive	Positive
14 Positive	Negative	Negative
15 Positive	Positive	Negative

Actual class	Predicted class		Total
	Positive	Negative	
Positive	5	1	6
Negative	3	6	9

Model 1

$$\rightarrow \text{Sensitivity} = \frac{5}{6} = 83\%$$

$$\rightarrow \text{Specificity} = \frac{6}{9} = 66\%$$

Actual class	Predicted class		Total
	Positive	Negative	
Positive	3	3	6
Negative	2	7	9

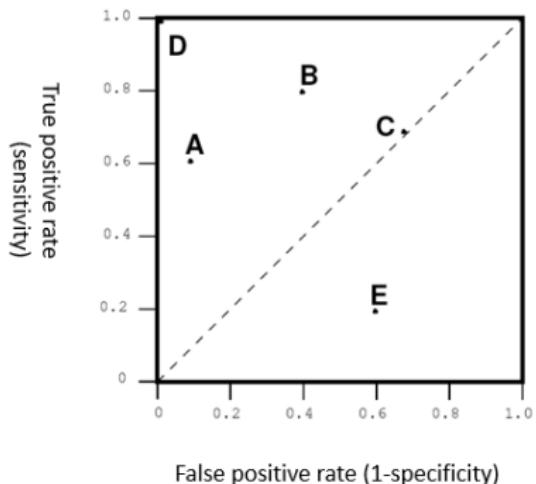
Model 2

$$\rightarrow \text{Sensitivity} = \frac{3}{6} = 50\%$$

$$\rightarrow \text{Specificity} = \frac{7}{9} = 78\%$$

Determining which model is the best one depends on the application.

Receiver Operating Characteristic (ROC) Curve (i)



Where are:

- The best classifier?
- A classifier that always says positive?
- A classifier that always says negative?
- A classifier that randomly guesses the class?

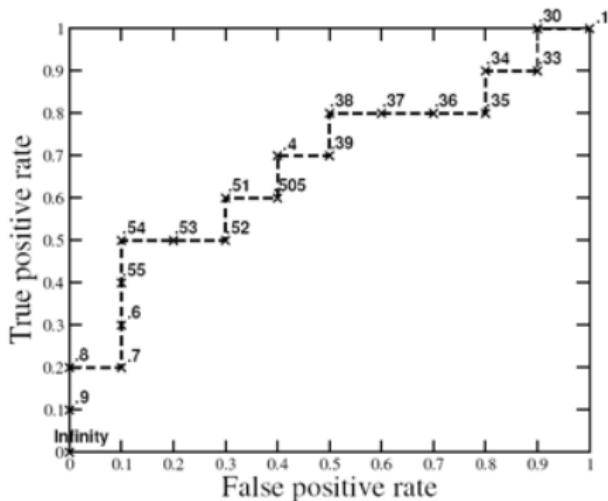
ROC Curve (ii)

Often, the output of a classification algorithm is a number, e.g. a class probability. In this case, a threshold may be chosen in order to balance sensitivity and specificity.

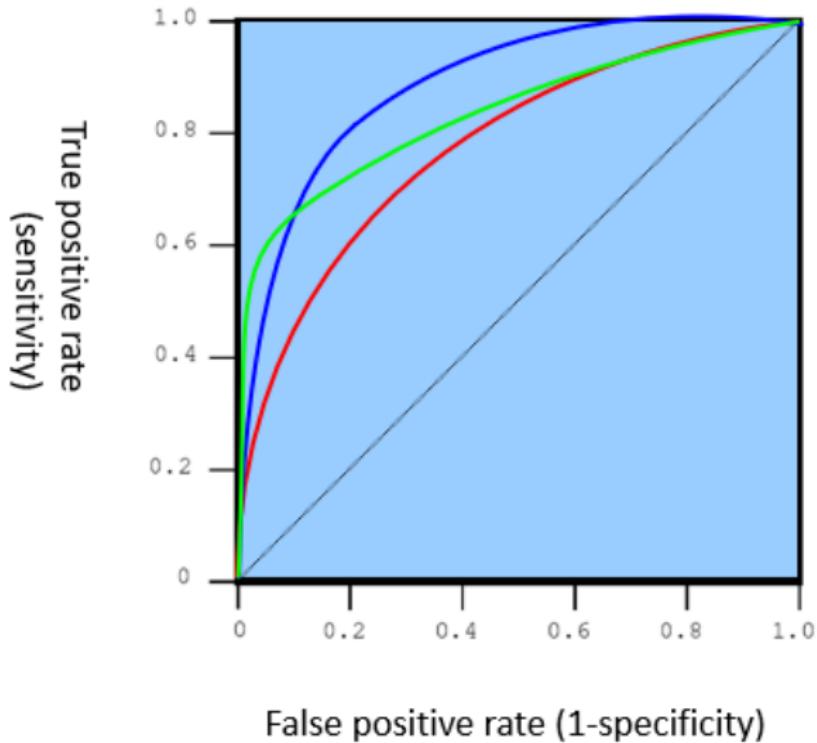
A ROC curve plots sensitivity versus $1 - \text{specificity}$ values for different thresholds.

ROC Curve (iii)

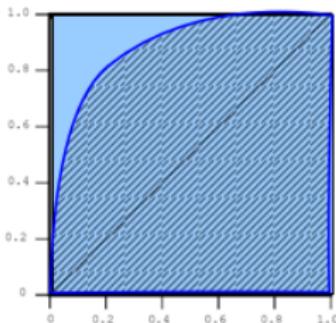
Inst#	Class	Score	Inst#	Class	Score
1	p	.9	11	p	.4
2	p	.8	12	n	.39
3	n	.7	13	p	.38
4	p	.6	14	n	.37
5	p	.55	15	n	.36
6	p	.54	16	n	.35
7	n	.53	17	p	.34
8	n	.52	18	n	.33
9	p	.51	19	p	.30
10	n	.505	20	n	.1



ROC Curve (iv)



Area under the ROC curve



The area under the ROC curve summarizes a ROC curve by a single number. It can be interpreted as the **probability** that two objects randomly drawn from the sample are well ordered by the model, *i.e.* the positive has a higher score than the negative.

However, it does not tell the whole story.

Precision and recall (i)

Other frequently used measures are:

- ▶ Precision = $\frac{TP}{TP+FP}$ = proportion of good predictions among all the positive predictions
- ▶ Recall = $\frac{TP}{TP+FN}$ = proportion of positives that are detected
- ▶ F-measure = $2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$

Precision and recall (ii)

Actual class	Predicted class		Total
	Positive	Negative	
Positive	10	0	10
Negative	50	950	1000

Model 1

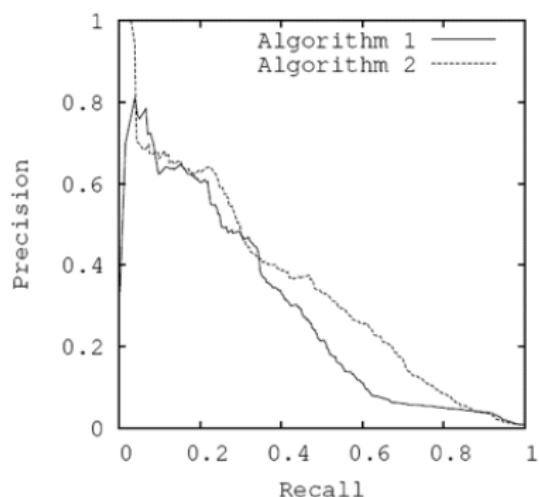
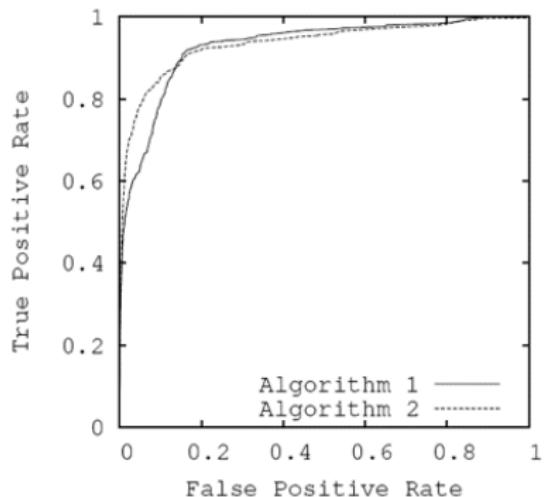
- Sensitivity = $\frac{10}{10} = 100\%$
- Specificity = $\frac{950}{1000} = 95\%$
- Precision = $\frac{10}{60} = 17\%$
- Recall = $\frac{10}{10} = 100\%$
- F-measure = 29%

Actual class	Predicted class		Total
	Positive	Negative	
Positive	10	0	10
Negative	10	990	1000

Model 2

- Sensitivity = $\frac{10}{10} = 100\%$
- Specificity = $\frac{990}{1000} = 99\%$
- Precision = $\frac{10}{20} = 50\%$
- Recall = $\frac{10}{10} = 100\%$
- F-measure = 66%

Precision/recall vs ROC curve



Regression performance (i)

	True Y	Model 1	Model 2
1	59.43	63.08	78.12
2	33.15	36.66	37.28
3	11.8	13.28	19.62
4	77.11	78.38	90.01
5	40.6	42.92	60.19
6	81.25	85	86.13
7	83.01	86.37	102.32
8	0.55	1.67	16.6
9	76.72	80.92	94.74
10	29.18	33.28	52.54
11	20.37	22.43	36.94
12	95.13	98.86	104.24
13	11.66	15.97	28.16
14	42.95	46.12	65.82
15	94.55	94.75	104.02

► Mean squared error

$$\frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2$$

$$\rightarrow MSE_1 = 53.38$$

$$\rightarrow MSE_2 = 249.6$$

► Mean absolute error

$$\frac{1}{N} \sum_{i=1}^N |y_i - \hat{y}_i|$$

$$\rightarrow MAE_1 = 4.3$$

$$\rightarrow MAE_2 = 14.6$$

Regression performance (ii)

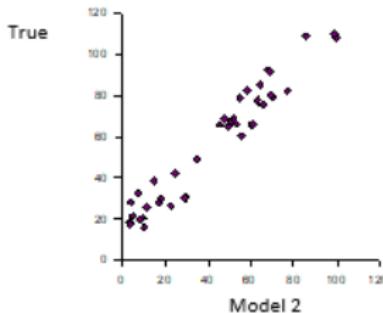
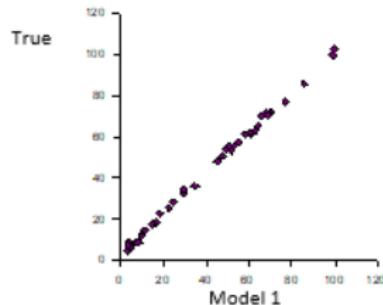
- ▶ Pearson correlation

$$\frac{\sum_i (y_i - \frac{1}{N} \sum_j y_j) (\hat{y}_i - \frac{1}{N} \sum_j \hat{y}_j)}{(N-1)s_y s_{\hat{y}}}$$

- ▶ Spearman rank correlation

$$1 - \frac{6 \sum_i d_i^2}{N(N^2 - 1)}$$

with d_i the difference of rank of y_i and \hat{y}_i



Performance measures for training

Performance measures for training can be different from performance measures for testing. There are several reasons for that:

► Algorithmic:

- A differentiable measure is amenable to gradient optimization.
- A decomposable measure is amenable to online training.

Examples: the error rate and MAE are not derivable, the AUC is not decomposable.

► Overfitting:

- For training, the loss function often incorporates a penalty term for model complexity, which is irrelevant at test time.
- Some measures are less prone to overfitting (e.g. margin).

Outline

- ① Bias/variance trade-off
- ② Performance evaluation
- ③ Performance measures
- ④ Further reading

Further reading

- Hastie et al., chapter 2 & 7, [1]:
 - Bias/variance trade-off (2.5, 2.9, 7.2, 7.3)
 - Model assessment and selection (7.1, 7.2, 7.3, 7.10, 7.11)

References |

-  Trevor Hastie, Robert Tibshirani, and Jerome Friedman.
The elements of statistical learning: data mining, inference, and prediction.
Springer Science & Business Media, 2009.
-  Cross-validation for detecting and preventing overfitting.
<https://www.autonlab.org/resources/tutorials>.
Accessed: 2020-10-21.

Support vector machines and kernel-based methods

Louis Wehenkel & Pierre Geurts

Institut Montefiore, University of Liège, Belgium



ELEN062-1
Introduction to Machine Learning
September 1, 2020

Outline

- ① Linear support vector machines
- ② The Kernel trick
- ③ Kernel ridge regression
- ④ Conclusion and references

Outline

① Linear support vector machines

Linear classification model

Maximal Margin Hyperplane

Optimisation problem formulation

A brief introduction to constrained optimization

Dual problem formulation

Support vectors

Soft margin SVM

② The Kernel trick

③ Kernel ridge regression

④ Conclusion and references

Linear classification model

- ▶ Given a $LS = \{(x_k, y_k)\}_{k=1}^N$, where $y_k \in \{-1, 1\}$, and $x_k \in \mathbb{R}^n$.
- ▶ Find a classifier in the form

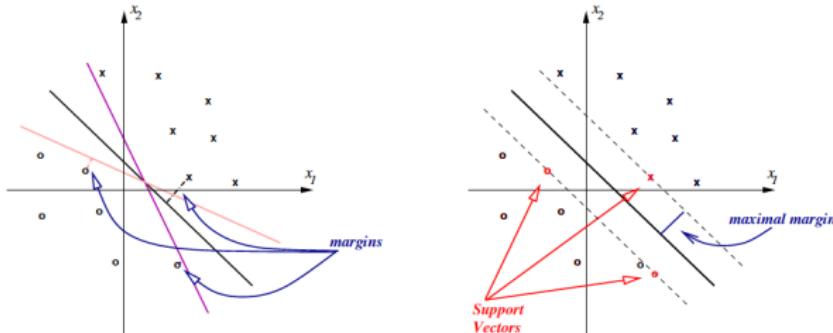
$$\hat{y}(x) = \text{sgn}(w^T x + b),$$

which classifies the LS correctly, i.e. that minimizes

$$\sum_{k=1}^N 1(y_k \neq \hat{y}(x_k))$$

- ▶ Several methods to find one such classifier: perceptron, linear discriminant analysis, naive bayes...

Maximal margin hyperplane



- ▶ When the data is linearly separable in the feature space, the separating hyperplane is not unique
- ▶ SVM maximizes the distance from the hyperplane to the nearest points in LS , i.e.

$$\max_{w,b} \min \{ \|x - x_k\| : w^T x + b = 0, k = 1, \dots, N \}.$$

Why maximizing the margin?

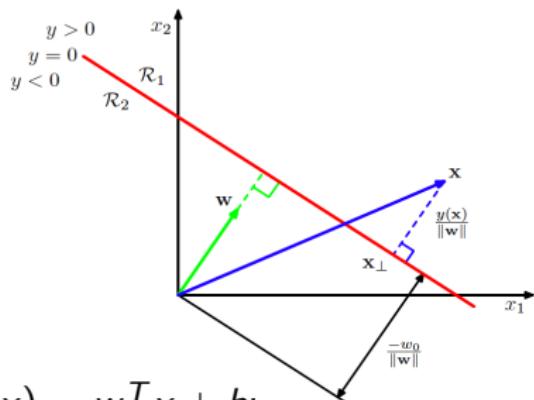
- ▶ Intuitively, it feels safest
- ▶ There exist theoretical bounds on the generalization error that depend on the margin

$$\text{Err}(\mathcal{T}S) < O(1/\gamma),$$

where γ is the margin. But these bounds are often loose.

- ▶ It works very well in practice.
- ▶ It yields a convex optimization problem whose solution can be written in terms of dot-products only. But this is the case with other criterion as well.

Some geometry



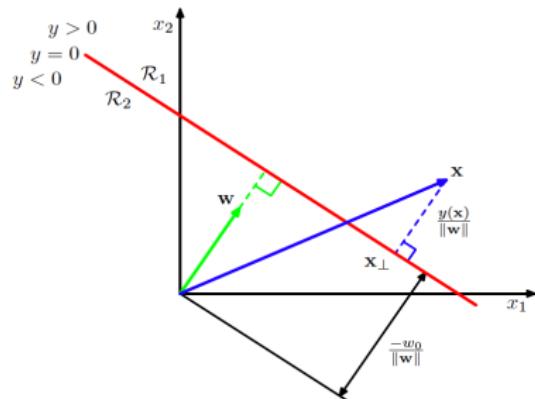
- \$w\$ is perpendicular to the line \$y(x) = w^T x + b\$:

$$y(x_a) = 0 = y(x_b) \Rightarrow w^T(x_A - x_B) = 0$$

- Let \$x\$ such that \$y(x) = 0\$. The distance from the origin to the line is:

$$\|x\| \cos(w, x) = \|x\| \frac{w^T x}{\|w\| \|x\|} = \frac{w^T x}{\|w\|} = \frac{-b}{\|w\|}$$

Some geometry



- Any point x can be written as:

$$x = x_{\perp} + r \frac{w}{\|w\|},$$

where $|r|$ is the distance from x to the line.

- Multiplying both sides by w^T and adding b , one gets:

$$w^T x + b = w^T x_{\perp} + b + r \frac{w^T w}{\|w\|} = 0 + r \cdot \|w\| \Rightarrow r = \frac{y(x)}{\|w\|}$$

Optimisation problem formulation

- The optimisation problem can be written:

$$\arg \max_{w,b} \left\{ \frac{1}{\|w\|} \min_n [y_n \cdot (w^T x_n + b)] \right\}.$$

- The solution is not unique as the hyperplane is unchanged if we multiply w and b by a constant $c > 0$.
- To impose unicity, one typically chooses $|w^T x + b| = 1$ for the point x that is closest to the surface (support vector).
- The problem is then equivalent to maximizing $\frac{1}{\|w\|}$ (or minimizing $\|w\|$) with the constraints:

$$y_k(w^T x_k + b) \geq 1, \forall k = 1, \dots, N.$$

(Show that $|w^T x_k + b| = 1$ for at least two points x_k at the solution)

Optimisation problem formulation

The SVM problem is equivalent to

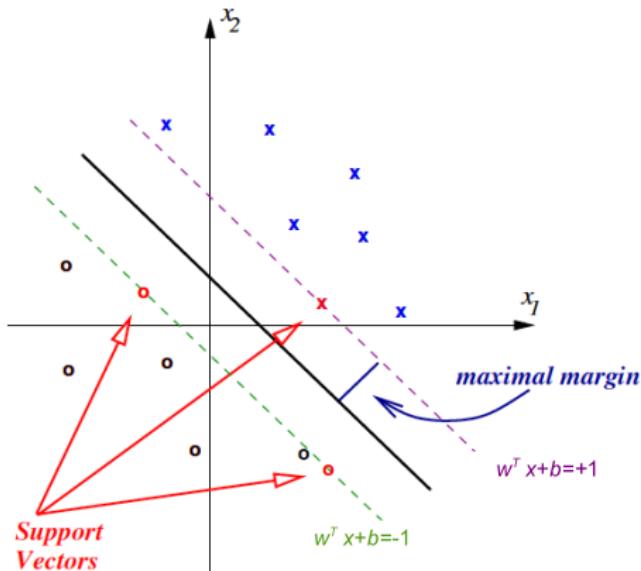
$$\min_{w,b} \mathcal{E}(w, b) = \frac{1}{2} \|w\|^2$$

subject to the N inequality constraints

$$y_k(w^T x_k + b) \geq 1, \forall k = 1, \dots, N.$$

NB

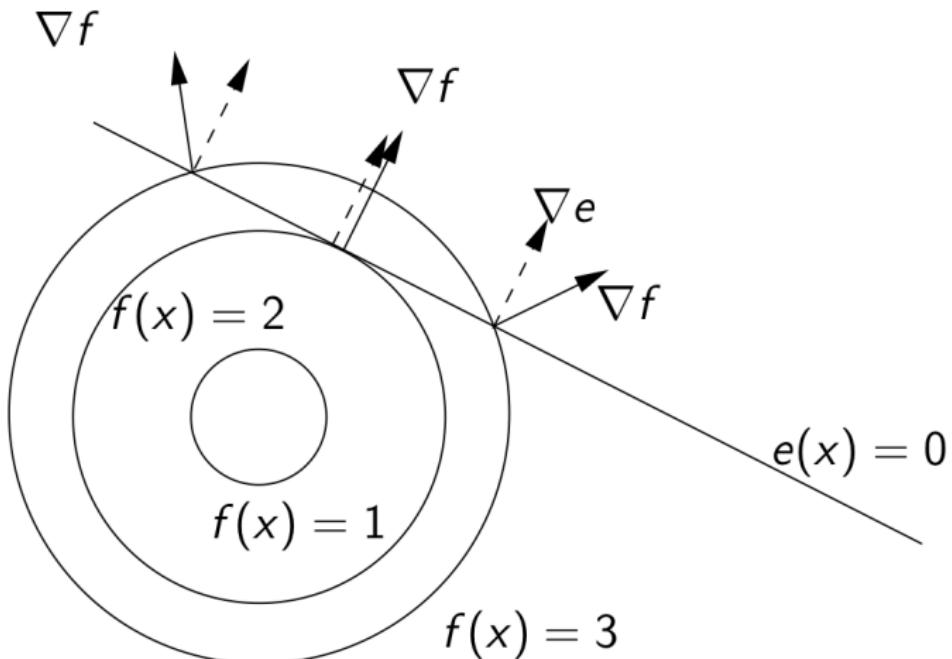
- ▶ $\|w\| \rightarrow \frac{1}{2} \|w\|^2$ for mathematical convenience
- ▶ This is a quadratic programming problem
- ▶ A solution exists only when the data is linearly separable



A brief introduction to constrained optimisation

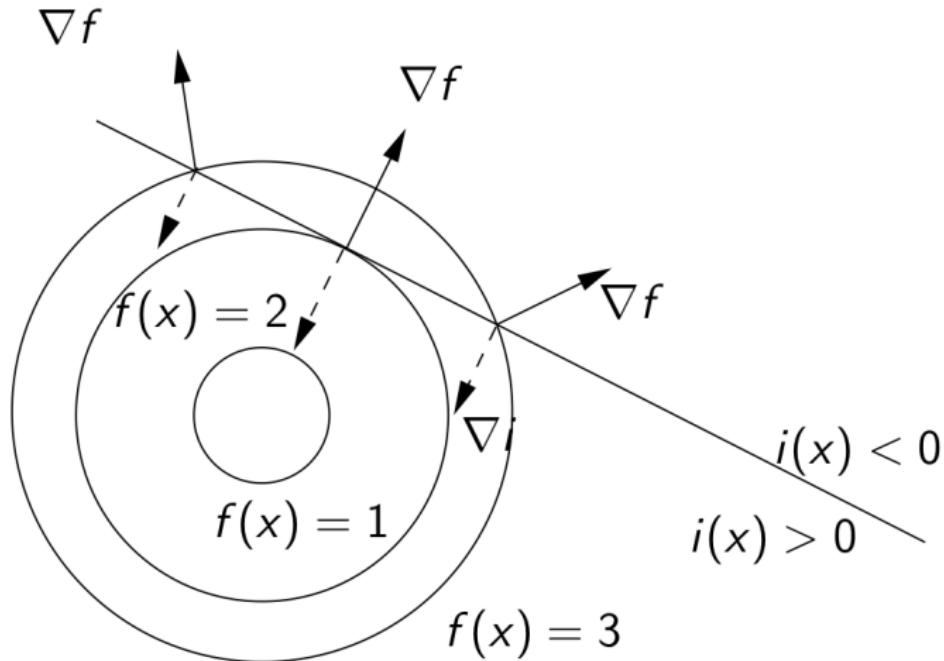
- ▶ Equality constrained optimisation problem
 - ▶ Minimise (or maximise) $f(x) \in \mathbb{R}$, with $x \in \mathbb{R}^n$
 - ▶ subject to p equality constraints $e_i(x) = 0, i = 1, \dots, p$
- ▶ Inequality constrained optimisation problem
 - ▶ Minimise (or maximise) $f(x) \in \mathbb{R}$, with $x \in \mathbb{R}^n$
 - ▶ subject to p equality constraints $e_i(x) = 0, i = 1, \dots, p$
 - ▶ subject to r inequality constraints $i_j(x) \leq 0, j = 1, \dots, r$
- ▶ Feasibility: existence of at least one x satisfying all equality and inequality constraints

Pictorial view: equality constraints



At the optimum, $\nabla f(x) + \alpha \nabla e(x) = 0$ and $e(x) = 0$.

Pictorial view: active inequality constraint



At the optimum, $\nabla f(x) + \alpha \nabla i(x) = 0$, $i(x) = 0$ and $\alpha > 0$.

Karush-Kuhn-Tucker conditions

- To minimise $f(x) \in \mathbb{R}$, with $x \in \mathbb{R}^n$, subject to equality constraints $e_i(x) = 0, i = 1, \dots, p$ and inequality constraints $i_j(x) \leq 0, j = 1, \dots, r$, define the Lagrangian

$$\mathcal{L}(x, \alpha, \beta) = f(x) + \alpha^T e(x) + \beta^T i(x) \quad (\alpha \in \mathbb{R}^p, \beta \in \mathbb{R}^r).$$

- At the optimum, there must exist some $\alpha \in \mathbb{R}^p$ and $\beta \in \mathbb{R}^r$ such that the following conditions hold:

$$\nabla_x \mathcal{L} = 0 \rightarrow \nabla_x f(x) + \alpha^T \nabla_x e(x) + \beta^T \nabla_x i(x) = 0$$

$$\nabla_\alpha \mathcal{L} = 0 \rightarrow e_i(x) = 0, \quad \forall i = 1, \dots, p$$

$$\nabla_\beta \mathcal{L} \leq 0 \rightarrow i_j(x) \leq 0, \quad \forall j = 1, \dots, r$$

$\beta_j i_j(x) = 0$, (complementary slackness conditions)

$$\beta_j \geq 0.$$

Convex optimization problems

If the optimization problem is convex (ie. f and $i_j, j = 1, \dots, r$ are convex and $e_i, i = 1, \dots, p$ are affine (linear) functions) and some conditions on the constraints are satisfied, we have (roughly):

- ▶ KKT conditions are necessary and sufficient conditions for optimality \Rightarrow in some cases, an optimum x^* may be obtained analytically from these conditions
- ▶ Let $\mathcal{W}(\alpha, \beta) = \min_x \mathcal{L}(x, \alpha, \beta)$ and let α^* and β^* be the solution of the (Lagrange) dual problem:

$$\begin{aligned} & \max_{\alpha, \beta} \mathcal{W}(\alpha, \beta) \\ & \text{subject to } \beta_j \geq 0, \forall j = 1, \dots, r, \end{aligned}$$

then the minimizer x^* of $\mathcal{L}(x, \alpha^*, \beta^*)$ is the solution of the original optimization problem.

Back to the optimisation problem formulation

- The SVM optimization problem is formulated as:

$$\min_{w,b} \mathcal{E}(w, b) = \frac{1}{2} \|w\|^2$$

subject to the N inequality constraints

$$y_k(w^T x_k + b) \geq 1, \forall k = 1, \dots, N.$$

- Let $\alpha_k \geq 0, k = 1, \dots, N$ and let us construct the Lagrangian

$$\mathcal{L}(w, b, \alpha) = \frac{1}{2} \|w\|^2 - \sum_{k=1}^N \alpha_k (y_k(w^T x_k + b) - 1)$$

This function must be minimized w.r.t. w and b , and maximized w.r.t. α .

Lagrangian equations

By deriving the Lagrangian

$$\mathcal{L}(w, b, \alpha) = \frac{1}{2}||w||^2 - \sum_{k=1}^N \alpha_k (y_k(w^T x_k + b) - 1)$$

according to the primal variables w and b , we obtain the following optimality conditions:

$$\begin{cases} \frac{\partial \mathcal{L}}{\partial w} = 0 & \rightarrow w = \sum_{j=1}^N \alpha_j y_j x_j \\ \frac{\partial \mathcal{L}}{\partial b} = 0 & \rightarrow \sum_{k=1}^N \alpha_k y_k = 0 \end{cases}$$

(with $\alpha_k \geq 0$).

Dual problem

Substituting in the Lagrangian

$$\mathcal{L}(w, b, \alpha) = \frac{1}{2}||w||^2 - \sum_{k=1}^N \alpha_k(y_k(w^T x_k + b) - 1)$$

the expressions $w = \sum_{j=1}^N \alpha_j y_j x_j$ and $\sum_{k=1}^N \alpha_k y_k = 0$ yields the dual (quadratic) maximisation problem

$$\max_{\alpha} \mathcal{W}(\alpha) = \sum_{k=1}^N \alpha_k - \frac{1}{2} \sum_{i,j=1}^N \alpha_i \alpha_j y_i y_j x_i^T x_j$$

subject to the N inequality constraints

$$\alpha_k \geq 0, \forall k = 1, \dots, N.$$

and one equality constraint

$$\sum_{i=1}^N \alpha_i y_i = 0$$

Support vectors

Back to the primal problem:

$$\mathcal{L}(w, b, \alpha) = \frac{1}{2} ||w||^2 - \sum_{k=1}^N \alpha_k (y_k (w^T x_k + b) - 1)$$

According to the KKT complementary conditions, the solution vector w is such that:

$$\alpha_k (y_k (w^T x_k + b) - 1) = 0, \forall k = 1, \dots, N$$

- ▶ $\alpha_k = 0$ if the constraint is satisfied as a strict inequality $y_k (w^T x_k + b) > 1$ because this is the way to maximize \mathcal{L}
- ▶ $\alpha_k > 0$ if the constraint is satisfied as an equality $y_k (w^T x_k + b) = 1$, in which case x_k is a **support vector**

Final model

Once the optimal values of α have been determined, the final model may be written as

$$\hat{y}(x) = \text{sgn} \left(\sum_{i=1}^N y_i \alpha_i x_i^T x + b \right),$$

where the α_k values that are different from zero (i.e. strictly positive) are corresponding to the **support vectors**.

NB: b is computed by exploiting the fact that for any $\alpha_k > 0$ we have necessarily $y_k(w^T x_k + b) - 1 = 0$.

Leave-one-out bound

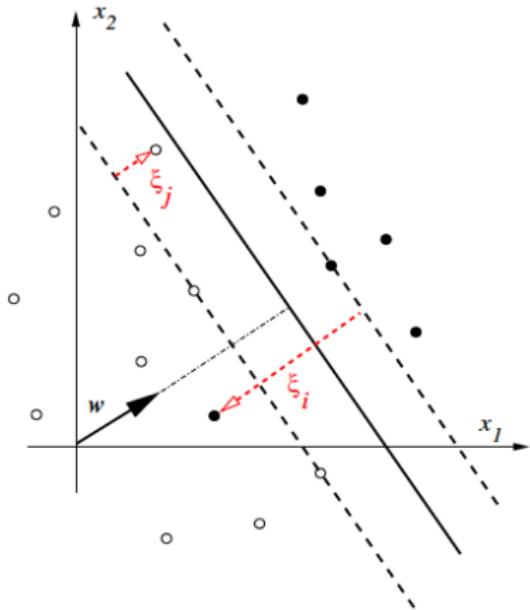
- ▶ Having a small set of support vectors is computationally efficient as only those vectors x_k and their weights α_k and class labels y_k need to be stored for classifying new examples

$$f(x) = \operatorname{sgn} \left(\sum_{i|\alpha_i > 0} y_i \alpha_i x_i^T x + b \right),$$

- ▶ Moreover, if a non-support vector x' is removed from the learning sample or moved around freely (outside the margin region), the solution would be unchanged.
- ▶ The proportion of support vectors in the learning sample gives a bound on the **leave-one-out error**:

$$\text{Err}_{\text{loo}} \leq \frac{|\{k | \alpha_k > 0\}|}{N}$$

Soft margin



- ▶ Due to noise or outliers, the samples may not be linearly separable in the feature space
- ▶ Discrepancies with respect to the margin is measured by slack variables $\xi_i \geq 0$ with the associated relaxed constraints $y_i(w^T x_i + b) \geq 1 - \xi_i$
- ▶ By making ξ_i large enough, the constraints can always be met
 - ▶ if $0 < \xi_i < 1$ the margin is not satisfied but x_i is still correctly classified
 - ▶ if $\xi_i > 1$ then x_i is misclassified

1-Norm soft margin optimization problem

Primal problem:

$$\min_{w, \xi} \frac{1}{2} \|w\|^2 + C \sum_{i=1}^N \xi_i$$

subject to

$$y_i(w^T x_i + b) \geq 1 - \xi_i, \xi_i \geq 0, \forall i = 1, \dots, N$$

where C is a positive constant balancing the objective of maximizing the margin and minimizing the margin error

Dual problem:

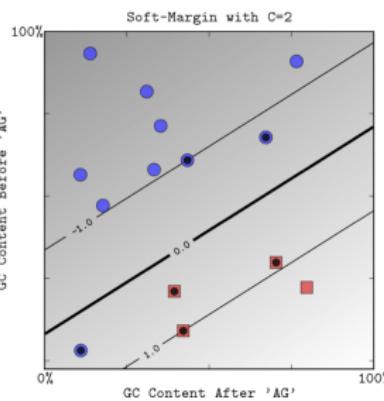
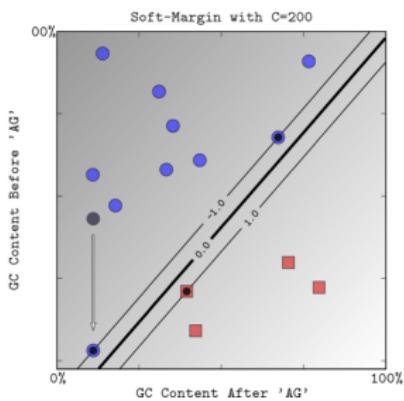
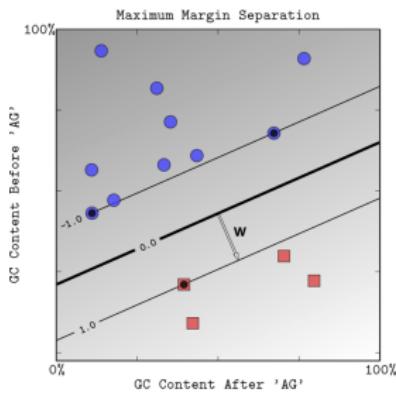
$$\max_{\alpha} \mathcal{W}(\alpha) = \sum_{k=1}^N \alpha_k - \frac{1}{2} \sum_{i,j=1}^N \alpha_i \alpha_j y_i y_j x_i^T x_j$$

subject to

$$0 \leq \alpha_k \leq C, \forall k = 1, \dots, N. \quad (\text{box constraints})$$

$$\sum_{i=1}^N \alpha_i y_i = 0$$

Effect of C



Outline

① Linear support vector machines

② The Kernel trick

Kernel based SVMs

Formal definition of Kernel function

Examples of Kernels

Kernel methods

③ Kernel ridge regression

④ Conclusion and references

Dot products

The quadratic optimization problem (hard and soft margin):

$$\max_{\alpha} \mathcal{W}(\alpha) = \sum_{k=1}^N \alpha_k - \frac{1}{2} \sum_{i,j=1}^N \alpha_i \alpha_j y_i y_j x_i^T x_j$$

and the decision function:

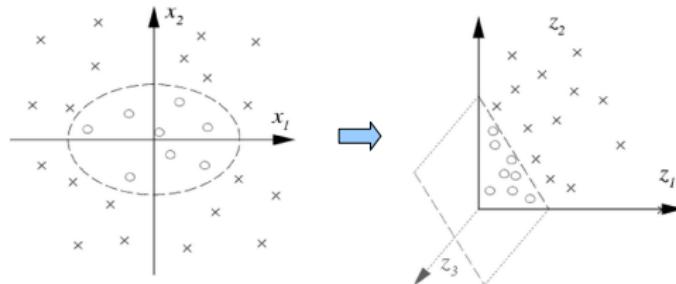
$$\hat{y}(x) = \text{sgn} \left(\sum_{i=1}^N y_i \alpha_i x_i^T x + b \right)$$

make use of the learning samples x_i only through dot products.

Moreover the number of parameters to be estimated only depends on the number of training samples and is thus independent of the dimension of the input space (number of attributes).

Non-linear support vector machines

- ▶ The training samples may not be linearly separable in the input space (the above optimization problem does not have a solution)
- ▶ Consider a non-linear mapping ϕ to a new feature space
 $\phi(x) = [z_1, z_2, z_3]^T = [x_1^2, x_2^2, \sqrt{2}x_1x_2]^T$



- ▶ The dual problem becomes

$$\max_{\alpha} \mathcal{W}(\alpha) = \sum_{k=1}^N \alpha_k - \frac{1}{2} \sum_{i,j=1}^N \alpha_i \alpha_j y_i y_j \phi(x_i)^T \phi(x_j)$$

The kernel trick

- ▶ Instead of explicitly defining a mapping ϕ , we can directly specify the dot product $\phi^T(x)\phi(x')$ and leave the mapping implicit
- ▶ It is possible to characterise mathematically the functions $K(x, x')$ defined on pairs of objects that correspond to the dot product $\phi^T(x)\phi(x')$ for some implicit mapping ϕ (see next slides)
⇒ Such function k is called a **(positive or Mercer) kernel**
- ▶ It can be thought of as a similarity measure between x and x'
- ▶ The **kernel trick**: Any learning algorithm that uses the data only via dot products can rely on this implicit mapping by replacing $x^T x'$ with $K(x, x')$

Kernel-based SVM classification

Assuming that we use a kernel $K(x, x')$ corresponding to the vectorisation map $\phi(x)$, we obtain straightforwardly

$$\hat{y}(x) = \text{sgn} \left(\sum_{k=1}^N y_k \alpha_k \phi(x_k)^T \phi(x) + b \right) = \text{sgn} \left(\sum_{k=1}^N y_k \alpha_k K(x, x_k) + b \right),$$

where the α_k can be determined by solving the following quadratic maximisation problem:

$$\max_{\alpha} \mathcal{W}(\alpha) = \sum_{k=1}^N \alpha_k - \frac{1}{2} \sum_{i,j=1}^N \alpha_i \alpha_j y_i y_j K(x_i, x_j)$$

subject to the N inequality constraints

$$\alpha_k \geq 0, \forall k = 1, \dots, N.$$

and one equality constraint

$$\sum_{i=1}^N \alpha_i y_i = 0$$

- ▶ Let U be a nonempty set of objects, then a function $K(\cdot, \cdot)$,

$$K(\cdot, \cdot) : U \times U \mapsto \mathbb{R}$$

such that for all $N \in \mathbb{N}$ and all $o_1, \dots, o_N \in U$ the $N \times N$ matrix

$$K : K_{i,j} = K(o_i, o_j)$$

is symmetric and positive (semi)definite, is called a **positive kernel**.

- ▶ Example: let $\phi(\cdot)$ be a function defined on U with values in \mathbb{R}^m (for some fixed m). Then

$$K_\phi(o, o') = \phi^T(o)\phi(o')$$

is a positive kernel. **NB:** $\phi(o)$ is a vector representation of o .

- The general result is as follows:

For any positive kernel K defined on U , there exists a scalar product space \mathcal{V} and a function $\phi(\cdot) : U \mapsto \mathcal{V}$, such that

$$K(o, o') = \phi(o) \times \phi(o'),$$

where the operator \times denotes the scalar product in \mathcal{V} .

- In general, the space \mathcal{V} is not necessarily of finite dimension.
- The kernel defines a “scalar product”, hence a “norm” and a “distance” measure over U , which are inherited from \mathcal{V} :

$$\begin{aligned} d_U^2(o, o') &= d_{\mathcal{V}}^2(\phi(o), \phi(o')) = (\phi(o) - \phi(o'))^T (\phi(o) - \phi(o')) \\ &= \phi(o) \times \phi(o) + \phi(o') \times \phi(o') - 2\phi(o) \times \phi(o') \\ &= K(o, o) + K(o', o') - 2K(o, o'). \end{aligned}$$

Examples of Kernels

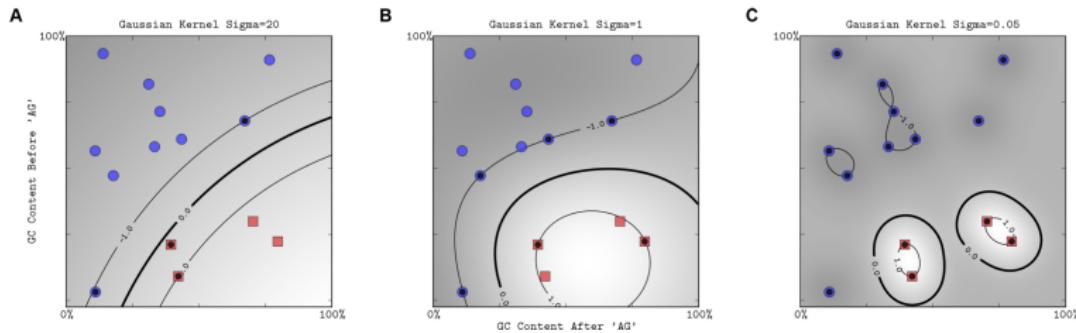
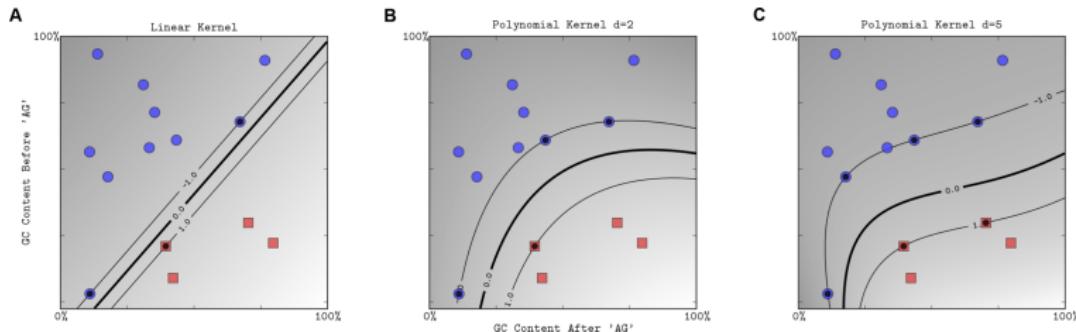
- ▶ A trivial kernel: the constant kernel
 - ▶ $K(o, o') \equiv 1$
- ▶ Linear kernel defined on numerical attributes
 - ▶ $K(o, o') = \mathbf{a}^T(o)\mathbf{a}(o')$
- ▶ Hamming kernel for discrete attributes:
 - ▶ $K(o, o') = \sum_{i=1}^m \delta_{a_i(o), a_i(o')}$
- ▶ Text kernel
 - ▶ number of common substrings in o and o'
(infinite dimension if text size is not a priori bounded)
- ▶ Combinations of kernels:
 - ▶ The sum of several (positive) kernels is also a (positive) kernel
 - ▶ The product of several kernels is also a kernel
 - ▶ Polynomial kernels: $\sum_{i=0}^n a_i(K(x, x'))^i$ if $\forall i : a_i \geq 0$

Examples of Kernels

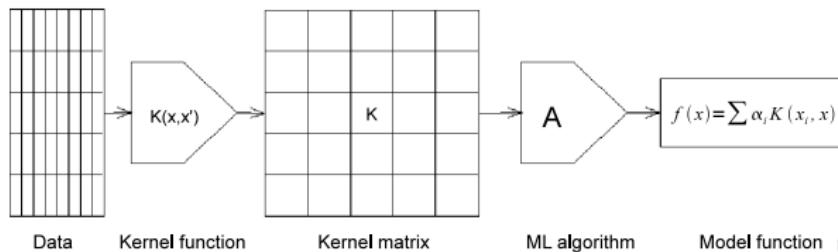
(for numerical attributes)

- ▶ Consider the kernel $K(x, x') = (x^T x')^2$.
 - ▶ We have in dimension 2, posing $x = (x_1, x_2)$ and $x' = (x'_1, x'_2)$:
 - ▶ $x^T x' = x_1 x'_1 + x_2 x'_2$, and thus
 - ▶ $(x^T x')^2 = (x_1 x'_1 + x_2 x'_2)^2 = x_1^2 x'_1^2 + 2x_1 x_2 x'_1 x'_2 + x_2^2 x'_2^2$.
 - ▶ Hence, posing $\phi(x) = (x_1^2, \sqrt{2}x_1 x_2, x_2^2)^T$
 - ▶ we have $(x^T x')^2 = \phi(x)^T \phi(x')$,
 - ▶ $K(x, x')$ corresponds to the scalar product in the space of degree 2 products of the original features.
- ▶ In general $K(x, x') = (x^T x')^d$ corresponds to the scalar product in the space of all degree d product features.
- ▶ Another useful kernel
 - ▶ Gaussian kernel: $K(x, x') = \exp\left(\frac{-(x-x')^T(x-x')}{2\sigma^2}\right)$

Effect of the kernel



Kernel methods



- ▶ Modular approach: decouple the algorithm from the representation
 - ▶ Many algorithms can be “kernelized”: ridge regression, fischer discriminant, PCA, k-means, etc.
 - ▶ Kernels have been defined for many data types: time series, images, graphs, sequences
- ▶ Main interests of kernel methods:
 - ▶ We can work in very high dimensional spaces (potentially infinite) efficiently as soon as the inner product is cheap to compute
 - ▶ Can apply classical algorithms on general, non-vectorial, data types (e.g. to build a linear model on graphs)

Outline

① Linear support vector machines

② The Kernel trick

③ Kernel ridge regression

④ Conclusion and references

Ridge regression problem

- ▶ Let us assume that we are given a learning set

$$LS = \{(x_k, y_k)\}_{k=1}^N,$$

and have chosen a kernel $K(x, x')$ defined on the input space, corresponding to a vectorisation mapping

$$\phi(x) \in \mathbb{R}^n \quad \text{i.e.} \quad \phi(x)^T \phi(x') = K(x, x').$$

- ▶ Least squares ridge regression consists in building a regression model in the form $\hat{y}(x) = w^T \phi(x) + b$, by minimising w.r.t. $w \in \mathbb{R}^n$ and $b \in \mathbb{R}$ the error function ($\gamma > 0$):

$$ERR(w, b) = \frac{1}{2} w^T w + \frac{1}{2} \gamma \sum_{k=1}^N (y_k - (w^T \phi(x_k) + b))^2.$$

Equality-constrained formulation of ridge regression

- We can restate the unconstrained minimisation problem

$$\min_{w,b} ERR(w, b) = \frac{1}{2} w^T w + \frac{1}{2} \gamma \sum_{k=1}^N (y_k - (w^T \phi(x_k) + b))^2,$$

as an (equality) constrained minimisation problem, namely

$$\min_{w,b,e} \mathcal{E}(w, b, e) = \frac{1}{2} w^T w + \frac{1}{2} \gamma \sum_{k=1}^N e_k^2$$

subject to N equality constraints

$$y_k - (w^T \phi(x_k) + b) = e_k, k = 1, \dots, N$$

Lagrangian formulation of kernel-based ridge regression

The optimisation problem to be solved:

$$\min_{w, b, e} \mathcal{E}(w, b, e) = \frac{1}{2} w^T w + \frac{1}{2} \gamma \sum_{k=1}^N e_k^2$$

subject to the N equality constraints

$$y_k = w^T \phi(x_k) + b + e_k, k = 1, \dots, N.$$

This problem can be solved by defining the Lagrangian

$$\mathcal{L}(w, b, e; \alpha) = \frac{1}{2} w^T w + \frac{1}{2} \gamma \sum_{k=1}^N e_k^2 - \sum_{k=1}^N \alpha_k (w^T \phi(x_k) + b + e_k - y_k)$$

and stating the optimality conditions

$$\frac{\partial \mathcal{L}}{\partial w} = 0; \frac{\partial \mathcal{L}}{\partial b} = 0; \frac{\partial \mathcal{L}}{\partial e_k} = 0; \frac{\partial \mathcal{L}}{\partial \alpha_k} = 0.$$

Lagrangian equations

In other words, we have

$$\mathcal{L}(w, b, e; \alpha) = \frac{1}{2} \|w\|^2 + \frac{1}{2}\gamma \sum_{k=1}^N e_k^2 - \sum_{k=1}^N \alpha_k (w^T \phi(x_k) + b + e_k - y_k)$$

yielding the optimality conditions

$$\begin{cases} \frac{\partial \mathcal{L}}{\partial w} = 0 & \rightarrow w = \sum_{j=1}^N \alpha_j \phi(x_j) \\ \frac{\partial \mathcal{L}}{\partial b} = 0 & \rightarrow \sum_{k=1}^N \alpha_k = 0 \\ \frac{\partial \mathcal{L}}{\partial e_k} = 0 & \rightarrow \alpha_k = \gamma e_k, \quad \forall k = 1, \dots, N \\ \frac{\partial \mathcal{L}}{\partial \alpha_k} = 0 & \rightarrow w^T \phi(x_k) + b + e_k - y_k = 0, \quad \forall k = 1, \dots, N \end{cases}$$

Lagrangian equations

Solution:

$$\begin{cases} \frac{\partial \mathcal{L}}{\partial w} = 0 & \rightarrow w = \sum_{j=1}^N \alpha_j \phi(x_j) \\ \frac{\partial \mathcal{L}}{\partial b} = 0 & \rightarrow \sum_{k=1}^N \alpha_k = 0 \\ \frac{\partial \mathcal{L}}{\partial e_k} = 0 & \rightarrow \alpha_k = \gamma e_k, \quad \forall k = 1, \dots, N \\ \frac{\partial \mathcal{L}}{\partial \alpha_k} = 0 & \rightarrow w^T \phi(x_k) + b + e_k - y_k = 0, \quad \forall k = 1, \dots, N \end{cases}$$

Substituting w from the first equation in the last one we can express e_k as

$$e_k = y_k - b - \sum_{j=1}^N \alpha_j \phi^T(x_j) \phi(x_k) = y_k - b - \sum_{j=1}^N \alpha_j K(x_j, x_k),$$

which allows us to eliminate e_k from the third equation.

Matrix equation for the unknowns α and b

We have to solve the following equations for $\alpha_1, \dots, \alpha_N$ and b

$$\sum_{k=1}^N \alpha_k = 0$$

$$\alpha_k = \gamma e_k \text{ i.e. } \alpha_k = \gamma \left(y_k - b - \sum_{j=1}^N \alpha_j K(x_j, x_k) \right)$$

in other words

$$\left(\begin{array}{c|c} 0 & 1 \cdots 1 \\ \hline 1 & \\ \vdots & K + \gamma^{-1} I \\ 1 & \end{array} \right) \begin{pmatrix} b \\ \alpha_1 \\ \vdots \\ \alpha_N \end{pmatrix} = \begin{pmatrix} 0 \\ y_1 \\ \vdots \\ y_N \end{pmatrix},$$

where the matrix K is defined by $K_{i,j} = K(x_i, x_j)$.

Kernel based ridge regression: discussion

- The model

$$\hat{y}(x) = b + \sum_{i=1}^N \alpha_i K(x, x_i)$$

is expressed purely in terms of the Kernel function.

- Learning algorithm uses only the values of $K(x_i, x_j)$ and y_k to determine α and b .
- We don't need to know $\phi(x)$.
- Order N^3 operations for learning, and order N operations for prediction.
- Refinements:
 - Sparse approximations
 - Prune away the training points which correspond to small values of α .

Outline

- ① Linear support vector machines
- ② The Kernel trick
- ③ Kernel ridge regression
- ④ Conclusion and references

Conclusion

- ▶ Support vector machines are based on two very smart ideas:
 - ▶ Margin maximization (to improve generalization)
 - ▶ The kernel trick (to handle very high dimensional or complex input spaces)
- ▶ SVMs are quite well motivated theoretically
- ▶ SVMs are among the most successful techniques for classification
- ▶ There exist very efficient implementations that can handle very large problems
- ▶ Drawbacks:
 - ▶ Essentially black-box models
 - ▶ The choice of a good kernel is difficult and critical to reach good accuracy

Conclusion

- ▶ Kernel methods are very useful tools in machine learning
 - ▶ Several generic frameworks have been proposed to define kernels for various data types
 - ▶ Many algorithms have been kernelized
 - ▶ Especially interesting in complex application domains like bioinformatics
- ▶ Convex optimization is a very useful tool in machine learning
 - ▶ Many machine learning problems can be formulated as convex optimization problems
 - ▶ Often, a unique solution implies a more stable model
 - ▶ Can benefit from the huge work made on optimization algorithms to focus on designing good objective functions

Further reading

- ▶ Bishop: Chapter 7 (7.1 and 7.1.1), Appendix E
- ▶ Hastie et al.: 12.2, 12.3 (12.3.1)
- ▶ Ben-Hur, A., Soon Ong, C., Sonnenburg, S., Schölkopf, B., Rätsch, G. (2008). *Support vector machines and kernels for computational biology*. PLOS Computational biology 4(10).
<http://www.ploscompbiol.org/>

References

- ▶ Schölkopf, B. and Smola, A. (2002). *Learning with kernels: support vector machines, regularization, optimization and beyond.* MIT Press, Cambridge, MA.
- ▶ Shawe-Taylor, J. and Cristianini, N. (2004). *Kernel methods for pattern analysis.* Cambridge University Press.
- ▶ Vapnik, V. (2000). *The nature of statistical learning theory.* Springer.
- ▶ Boyd, S. and Vandenberghe, L (2004). *Convex optimization.* Cambridge University Press.
- ▶ Some slides borrowed from Pierre Dupont (UCL)

Softwares

- ▶ LIBSVM & LIBLINEAR

<http://www.csie.ntu.edu.tw/~cjlin/libsvm>

- ▶ SVM^{light}

<http://svmlight.joachims.org/>

- ▶ SHOGUN

<http://www.shogun-toolbox.org/>

Ensemble methods and Feature selection

Pierre Geurts & Louis Wehenkel

Institut Montefiore, University of Liège, Belgium



ELEN062-1
Introduction to Machine Learning
November 15, 2023

Acknowledgment: These slides have been reformatted by Yann Claes in July 2020.

Part I

Ensemble methods

Outline

- ① Averaging techniques
- ② Boosting techniques
- ③ Other ensemble approaches
- ④ Conclusions on ensemble methods

Reminder: bias/variance decomposition

$$E_{LS} \left\{ E_{y|\underline{x}} \left\{ (y - \hat{y}(\underline{x}))^2 \right\} \right\} = \text{noise}(\underline{x}) + \text{bias}^2(\underline{x}) + \text{variance}(\underline{x})$$

- $\text{noise}(\underline{x}) = E_{y|\underline{x}} \left\{ (y - h_B(\underline{x}))^2 \right\}$: quantifies how much y varies from $h_B(\underline{x}) = E_{y|\underline{x}}\{y\}$ (the Bayes model).
- $\text{bias}^2(\underline{x}) = (h_B(\underline{x}) - E_{LS}\{\hat{y}(\underline{x})\})^2$: measures the error between the Bayes model and the average model.
- $\text{variance}(\underline{x}) = E_{LS} \left\{ (\hat{y} - E_{LS}\{\hat{y}(\underline{x})\})^2 \right\}$: quantifies how much $\hat{y}(\underline{x})$ varies from one learning sample to another.

Reminder: bias and variance reduction techniques

- ▶ In the context of a given method:
 - Adapt the learning algorithm to find the best trade-off between bias and variance.
 - Not a panacea but the least we can do.
Example: pruning, weight decay.
- ▶ Ensemble methods:
 - Change the bias/variance trade-off.
 - Universal but destroys some features of the initial method.
Example: bagging, boosting.

Ensemble methods

They combine the predictions of several models built with a learning algorithm in order to improve with respect to the use of a single model.

There exist two main families:

- ▶ **Averaging** techniques: they grow several models independently and simply average their predictions. They mainly decrease **variance**.
Example: bagging, random forests.
- ▶ **Boosting** techniques: they grow several models sequentially. They mainly decrease **bias**.
Example: adaboost, gradient boosting.

Outline

① Averaging techniques

Bagging

Random Forests

Ambiguity decomposition

② Boosting techniques

③ Other ensemble approaches

④ Conclusions on ensemble methods

Bagging (i)

Suppose that we can generate several learning samples LS_i from the original data distribution $P(\underline{x}, y)$.

Let us study the following algorithm:

- Draw T learning samples $\{LS_1, LS_2, \dots, LS_T\}$
- Learn a model \hat{y}_{LS_i} from each LS_i
- Compute the average model $\hat{y}_{ens}(\underline{x}) = \frac{1}{T} \sum_{i=1}^T \hat{y}_{LS_i}(\underline{x})$

How do the bias and variance of this algorithm relate to that of the original algorithm?

Bagging (ii)

The bias/variance decomposition is given by:

$$\begin{aligned} E_{LS}\{\text{Err}(\underline{x})\} &= E_{y|\underline{x}} \left\{ (y - h_B(\underline{x}))^2 \right\} + (h_B(\underline{x}) - E_{LS}\{\hat{y}(\underline{x})\})^2 \\ &\quad + E_{LS} \left\{ (\hat{y}(\underline{x}) - E_{LS}\{\hat{y}(\underline{x})\})^2 \right\} \end{aligned}$$

Its **bias** is the **same** as the original algorithm:

$$\begin{aligned} E_{LS_1, \dots, LS_T} \{\hat{y}_{ens}(\underline{x})\} &= \frac{1}{T} \sum_i E_{LS_i} \{\hat{y}_{LS_i}(\underline{x})\} \\ &= E_{LS} \{\hat{y}_{LS}(\underline{x})\} \end{aligned}$$

Its **variance** is divided by T :

$$\begin{aligned} E_{LS_1, \dots, LS_T} \{(\hat{y}_{ens}(\underline{x}) - E_{LS_1, \dots, LS_T} \{\hat{y}_{ens}(\underline{x})\})^2\} \\ = \frac{1}{T} E_{LS} \left\{ (\hat{y}(\underline{x}) - E_{LS} \{\hat{y}(\underline{x})\})^2 \right\} \end{aligned}$$

⇒ The mean square error **decreases**.

Bagging (iii)

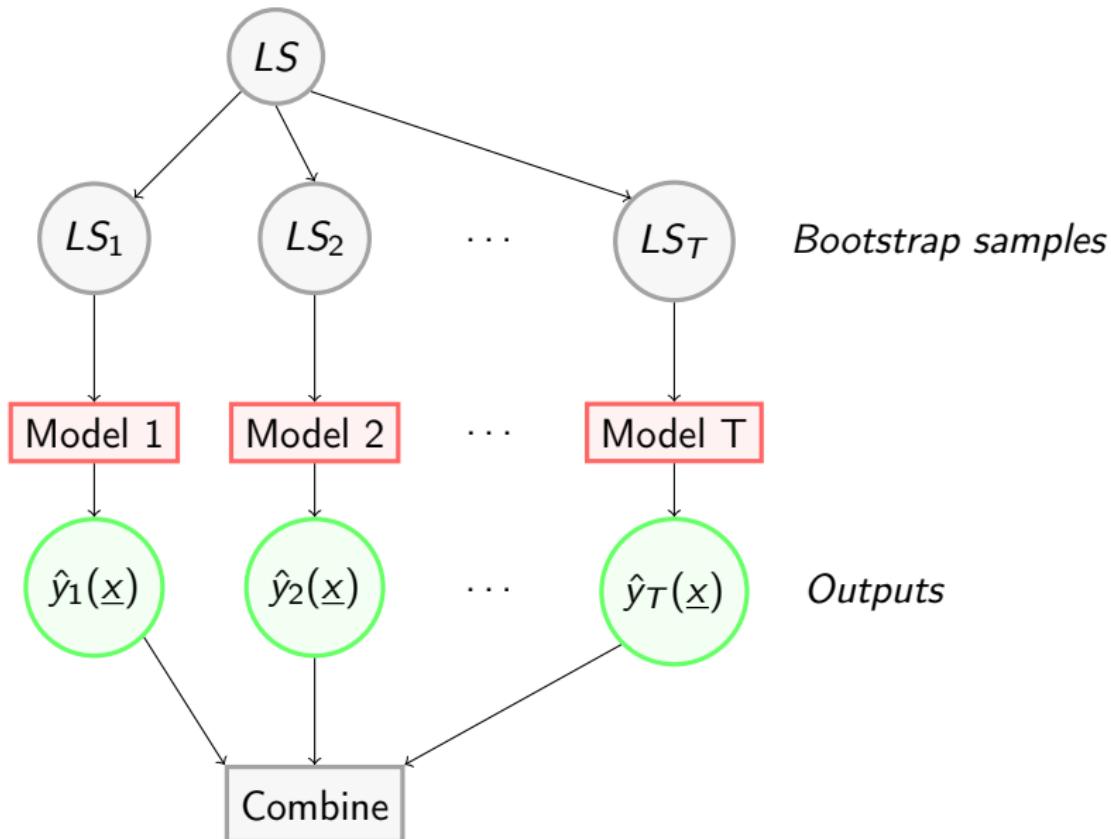
In practice, one cannot draw several LS_i : $P(\underline{x}, y)$ is **unknown**.
⇒ **Idea:** use bootstrap sampling to generate several learning samples.

This is the idea behind **bagging** (**bootstrap aggregating**):

- Draw T bootstrap samples $\{B_1, B_2, \dots, B_T\}$ from LS
- Learn a model \hat{y}_{B_i} from each B_i
- Build the average model $\hat{y}_{ens}(\underline{x}) = \frac{1}{T} \sum_{i=1}^T \hat{y}_{B_i}(\underline{x})$

Variance is **reduced** but bias **increases** a bit (because the effective size of a bootstrap sample is about 30% smaller than the original LS).

Bagging (iv)



Bagging (v)

For a regression problem:

$$\hat{y}_{ens}(\underline{x}) = \frac{1}{T}(\hat{y}_1(\underline{x}) + \hat{y}_2(\underline{x}) + \dots + \hat{y}_T(\underline{x}))$$

For a classification problem, instead, take the majority class in

$$\{\hat{y}_1(\underline{x}), \hat{y}_2(\underline{x}), \dots, \hat{y}_T(\underline{x})\}$$

Bagging (vi)

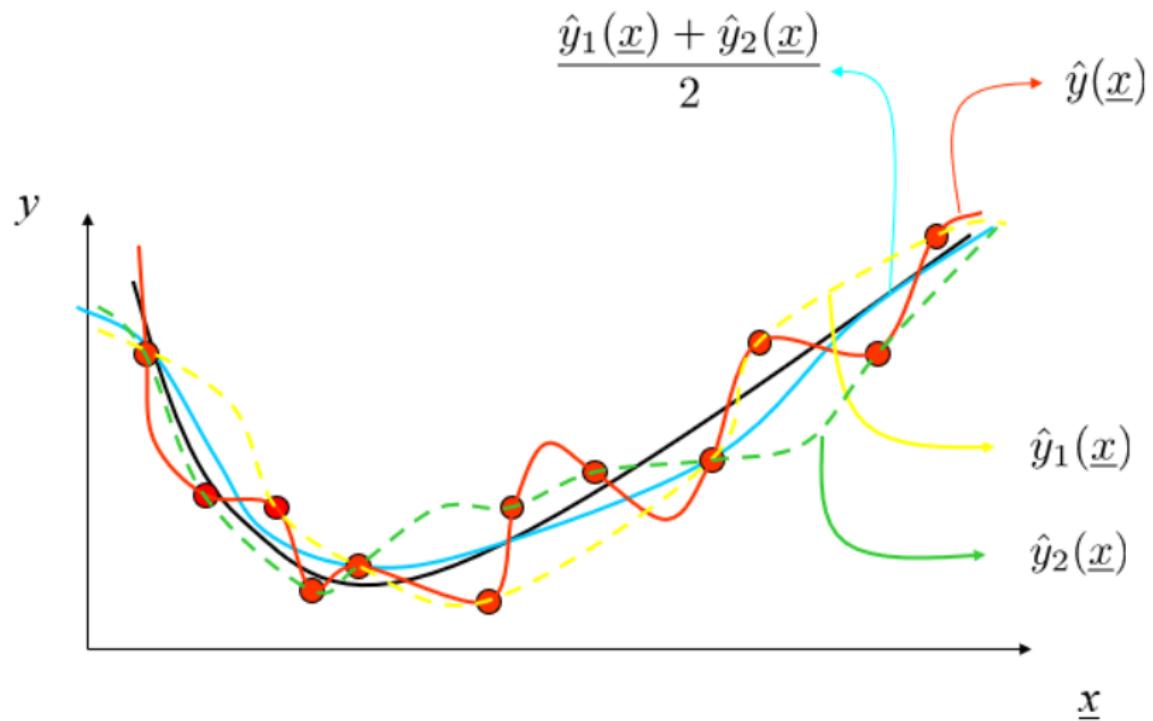
Usually, bagging reduces very much the variance without increasing too much the bias.

Method	E	Bias	Variance
3 Test regression Tree	14.8	11.1	3.7
Bagged ($T=25$)	11.7	10.7	1.0
Full regression Tree	10.2	3.5	6.7
Bagged ($T=25$)	5.3	3.8	1.5

Application of bagging to regression trees (on Friedman's problem).

⇒ Strong variance reduction without increasing the bias (although the model is much more complex than a single tree).

Bagging (vii)



Other averaging techniques

Another approach is the *perturb and combine* paradigm:

1. Perturb the data/learning algorithm to obtain several models that are good on the learning sample
2. Combine the predictions of these models

Usually, these methods decrease the variance (because of **averaging**) but slightly increase the bias (because of the **perturbation**).

Examples:

- Bagging perturbs the learning sample
- Learn several neural networks with random initial weights

Method	E	Bias	Variance
MLP (10-10)	4.6	1.4	3.2
Average of 10 MLPs	2.0	1.4	0.6

- Random forests

Random forests (i)

Random forests is a type of *perturb and combine* algorithm specifically designed for trees.

It combines **bagging** and **random attribute subset selection**:

- Build the tree from a bootstrap sample
- Instead of choosing the best split among all attributes, select the best split among a **random** subset of k attributes.

Note: It is equivalent to bagging when k is equal to the number of attributes.

There is a bias/variance trade-off with k : the smaller the k , the greater the reduction of variance but the higher the increase of bias.

Random forests (ii)

Method	E	Bias	Variance
Full regression Tree	10.2	3.5	6.7
Bagging ($k = 10$)	5.3	3.8	1.5
Random forests ($k = 7$)	4.8	3.8	1.0
Random forests ($k = 5$)	4.9	4.0	0.9
Random forests ($k = 3$)	5.6	4.7	0.9

Application to our illustrative problem.

Another advantage is that it decreases **computing times** with respect to bagging since only a subset of all attributes is considered when splitting a node.

Ambiguity decomposition (i)

Let us assume T models $\{\hat{y}_1, \hat{y}_2, \dots, \hat{y}_T\}$ and their average

$$\hat{y}_{ens}(\underline{x}) = \frac{1}{T} \sum_i \hat{y}_i(\underline{x})$$

We have the following decomposition:

$$\begin{aligned} \frac{1}{T} \sum_i E_{y|\underline{x}} \left\{ (y - \hat{y}_i(\underline{x}))^2 \right\} &= E_{y|\underline{x}} \left\{ (y - \hat{y}_{ens}(\underline{x}))^2 \right\} \\ &\quad + \frac{1}{T} \sum_i (y_i(\underline{x}) - \hat{y}_{ens}(\underline{x}))^2 \end{aligned}$$

meaning that

$$\begin{aligned} E_{y|\underline{x}} \left\{ (y - \hat{y}_{ens}(\underline{x}))^2 \right\} &= \frac{1}{T} \sum_i E_{y|\underline{x}} \left\{ (y - \hat{y}_i(\underline{x}))^2 \right\} \\ &\quad - \frac{1}{T} \sum_i (y_i(\underline{x}) - \hat{y}_{ens}(\underline{x}))^2 \end{aligned}$$

Ambiguity decomposition (ii)

Hence, the average model is **always** better than the individual models in the mean.

However, this is **not** true in classification!

Application: Kinect

An ensemble of randomized decision trees is used in Microsoft's Xbox Kinect for body part labelling.



Source: [1]

Each sample corresponds to a single pixel and is described by depth differences between neighbouring pixels.

The final model is implemented on GPU to get very fast predictions (200 frames per second).

Outline

- ① Averaging techniques
- ② Boosting techniques
 - Adaboost
 - Residual fitting
 - Gradient boosting
 - Bias/variance trade-off
- ③ Other ensemble approaches
- ④ Conclusions on ensemble methods

Motivation

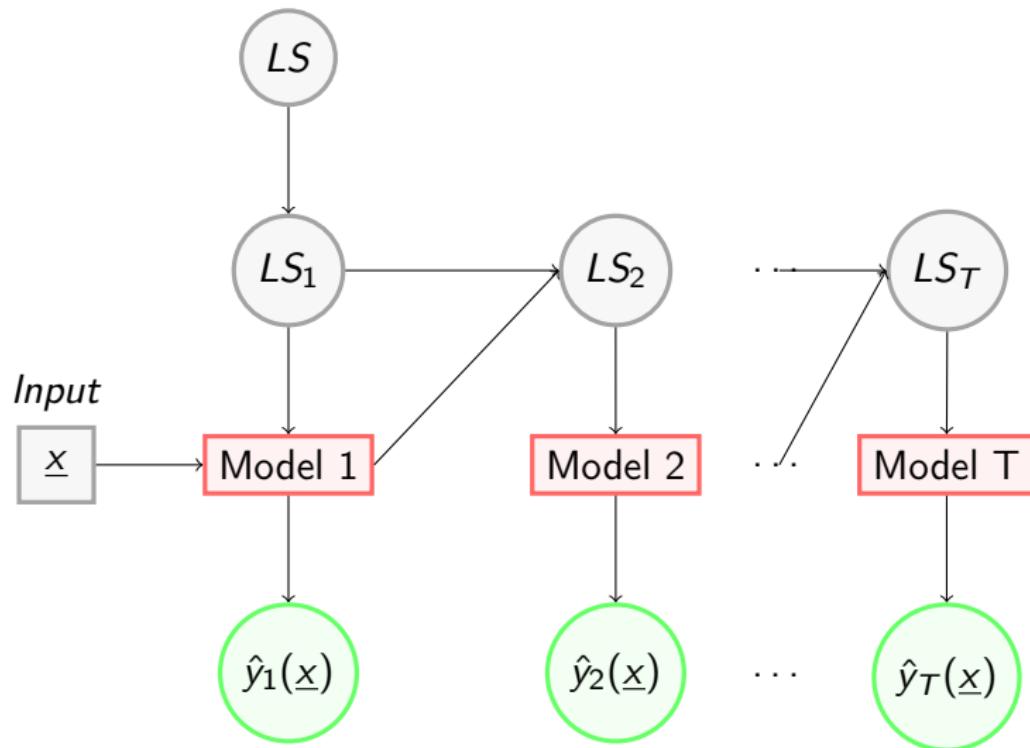
The motivation of boosting is to combine the outputs of many “weak” models to produce a powerful ensemble of models.

Note: A “weak” model is a model that has a high bias (strictly, in classification, a model slightly better than random guessing).

Differences with previous ensemble methods:

- Models are built **sequentially** on modified versions of the data.
- The predictions of the models are combined through a **weighted** sum/vote.

Boosting methods (i)



Boosting methods (ii)

Regression:

$$\hat{y}(\underline{x}) = \beta_1 \hat{y}_1(\underline{x}) + \beta_2 \hat{y}_2(\underline{x}) + \dots + \beta_T \hat{y}_T(\underline{x})$$

Classification:

$\hat{y}(\underline{x})$ = majority class in $\{\hat{y}_1(\underline{x}), \dots, \hat{y}_T(\underline{x})\}$ according to $\{\beta_1, \dots, \beta_T\}$.

Boosting methods (iii)

Many boosting methods exist.

First invented by computer scientists (Freund and Schapire, 1995) and then generalized by statisticians (e.g., Friedman, 2001).

In this lecture:

- ▶ Adaboost: specific method for classification
- ▶ Residual fitting: specific method for regression
- ▶ Gradient boosting: a generic method that can accommodate any loss function.

Adaboost: main principle

Assume that the learning algorithm accepts weighted objects

$$\{(\underline{x}_1, y_1, w_1), (\underline{x}_2, y_2, w_2), \dots, (\underline{x}_N, y_N, w_N)\}$$

Note: This is the case of many learning algorithms:

- With trees, take the weights into account when counting objects
- In neural networks, minimize the weighted squared error

At each step, Adaboost **increases** the weights of cases from the learning sample being **misclassified** by the last model. Therefore, the algorithm focuses on the difficult cases from the learning sample.

In the weighted majority vote, Adaboost gives higher influence to the more accurate models.

Adaboost: algorithm

- ▶ Input: a learning algorithm and a learning sample

$$\{(\underline{x}_i, y_i) : i = 1, \dots, N\}$$

- ▶ Initialize the weights

$$w_i = \frac{1}{N}, i = 1, \dots, N$$

- ▶ For $t = 1$ to T :

1. Build a model $\hat{y}_t(\underline{x})$ with the learning algorithm using weights w_i
2. Compute the weighted error

$$err_t = \frac{\sum_i w_i I(y_i \neq \hat{y}_t(\underline{x}_i))}{\sum_i w_i}$$

3. Compute $\beta_t = \log(\frac{1 - err_t}{err_t})$
4. Change the weights according to

$$w_i \leftarrow w_i \exp [\beta_t I(y_i \neq \hat{y}_t(\underline{x}_i))]$$

5. Normalize the weights such that $\sum_i w_i = 1$

Adaboost: illustration

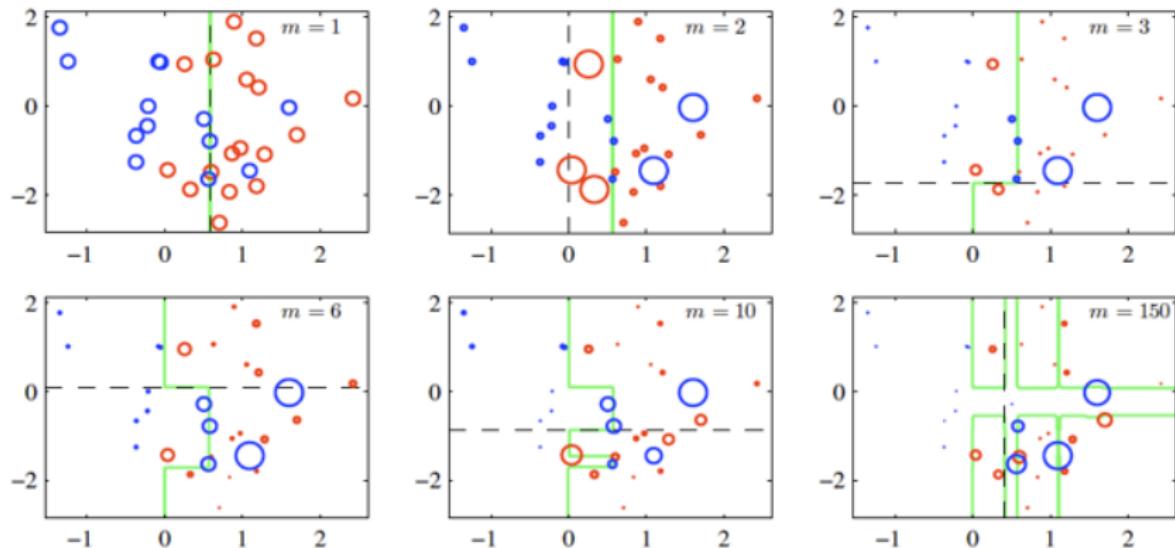


Figure 14.2 Illustration of boosting in which the base learners consist of simple thresholds applied to one or other of the axes. Each figure shows the number m of base learners trained so far, along with the decision boundary of the most recent base learner (dashed black line) and the combined decision boundary of the ensemble (solid green line). Each data point is depicted by a circle whose radius indicates the weight assigned to that data point when training the most recently added base learner. Thus, for instance, we see that points that are misclassified by the $m = 1$ base learner are given greater weight when training the $m = 2$ base learner.

Source: [2]

Residual fitting: algorithm

(a specific boosting algorithm for least-square regression)

- ▶ Input: a learning sample $\{(\underline{x}_i, y_i) : i = 1, \dots, N\}$
- ▶ Initialize the model and residuals

$$\hat{y}(\underline{x}) = \hat{y}_0(\underline{x}) = \frac{1}{N} \sum_i y_i$$

$$r_i = y_i \quad (i = 1, \dots, N)$$

- ▶ For $t = 1$ to T :
 1. For $i = 1$ to N , compute the residuals w.r.t the current \hat{y} :

$$r_i \leftarrow r_i - \hat{y}(\underline{x}_i)$$

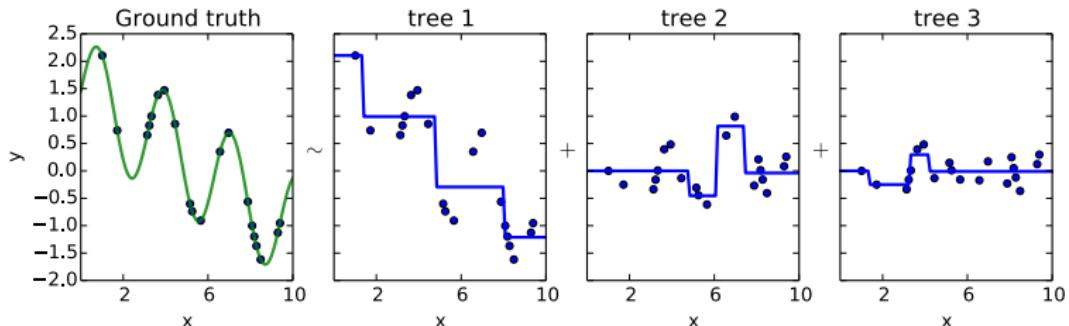
2. Build a regression model \hat{y}_t from the new learning sample

$$\{(\underline{x}_i, r_i) : i = 1, \dots, N\}$$

3. Update the model: $\hat{y}(\underline{x}) \leftarrow \hat{y}(\underline{x}) + \hat{y}_t(\underline{x})$
- ▶ Return the model $\hat{y}(\underline{x}) = \hat{y}_0(\underline{x}) + \hat{y}_1(\underline{x}) + \dots + \hat{y}_T(\underline{x})$

Residual fitting: illustration

With regression trees:



[Source: Louppe, 2014]

A generic boosting algorithm

Goal: Find $\hat{y}(\underline{x}) = \beta_1 \hat{y}_1(\underline{x}) + \beta_2 \hat{y}_2(\underline{x}) + \dots + \beta_T \hat{y}_T(\underline{x})$ that minimizes $\sum_{i=1}^N L(y_i, \hat{y}(\underline{x}_i))$.

Forward **stage-wise additive** modeling:

1. Initialize $\hat{y}(\underline{x}) = 0$
2. For $t = 1$ to T :
 - 2.1 Compute $(\beta_t, \hat{y}_t) = \arg \min_{\beta, \hat{y}'} \sum_i L(y_i, \hat{y}(\underline{x}_i) + \beta \hat{y}'(\underline{x}_i))$
 - 2.2 Set $\hat{y}(\underline{x}) \leftarrow \hat{y}(\underline{x}) + \beta_t \hat{y}_t(\underline{x})$

Examples:

- $L(y, y') = (y - y')^2 \Rightarrow$ Least-squares boosting
- $L(y, y') = \exp(-yy') \Rightarrow$ Adaboost (*try to prove it*)

But step 2.1 is not easy to solve in general \Rightarrow replace by a **gradient** step.

Gradient boosting

(Friedman, 2001)

General algorithm:

1. $\hat{y}(\underline{x}) \leftarrow \arg \min_{\gamma} \sum_i L(y_i, \gamma)$

2. For $t = 1$ to T :

- 2.1 For all (\underline{x}_i, y_i) , compute:

$$r(\underline{x}_i, y_i) = - \left[\frac{\partial L(y_i, f(\underline{x}_i))}{\partial f(\underline{x}_i)} \right]_{f=\hat{y}}$$

- 2.2 $\hat{y}_t \leftarrow \arg \min_{\hat{y}'} \sum_i (\hat{y}'(\underline{x}_i) - r(\underline{x}_i, y_i))^2$

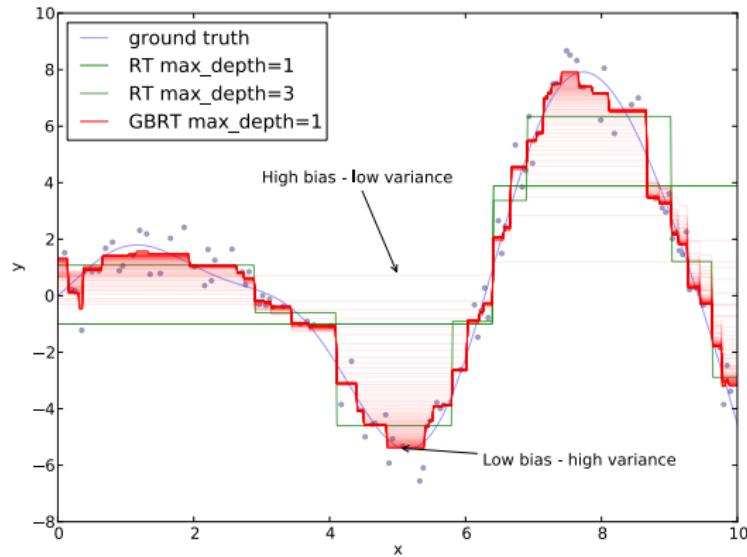
- 2.3 $\hat{y}(x) \leftarrow \hat{y}(x) + \mu \hat{y}_t(\underline{x})$, with $\mu \in [0, 1]$ a learning rate.

Step 2.2 can be solved with **any** least-square regressor.

Regression trees are the most commonly used base learners (because of low computational cost and good predictive performances).

Illustration (with regression trees)

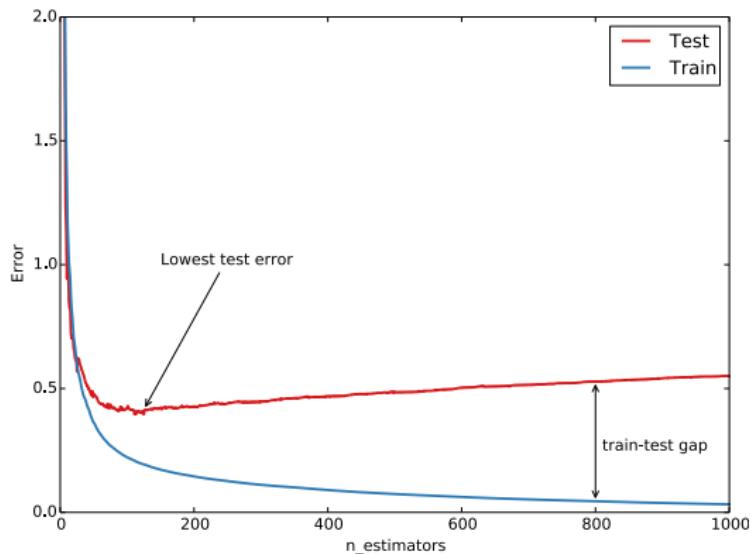
$$Y = \sin(X) + \sin(2 * X) + \epsilon, \text{ avec } X \sim U([0, 10]) \text{ et } \epsilon \sim N(0, 2)$$



Source: <https://github.com/pprett/pydata-gbdt-tutorial>

Illustration: impact of the number of trees

Unlike Bagging or Random forests, Gradient boosting can overfit with the number of trees



Source: <https://github.com/pprett/pydata-gbdt-tutorial>

Regularization

Many techniques exist to avoid overfitting:

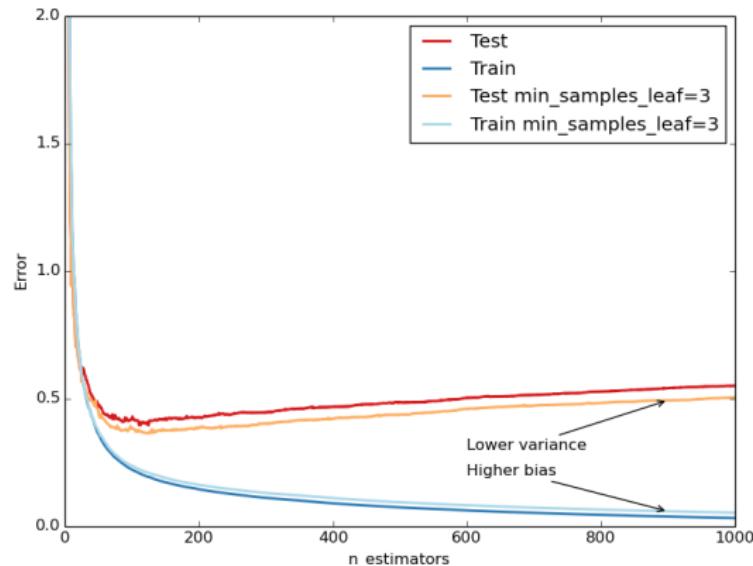
- ▶ **Shrinkage:** use a learning rate $\mu < 1$.
- ▶ **Reduce tree complexity:** constrain depth, minimum number of samples per leaf, or number of (test) nodes.
- ▶ **Stochastic gradient boosting:** introduce randomization à la Bagging/Random forests (e.g., feature and sample subsampling)

Each technique introduces a hyper-parameter that regulates some bias/variance trade-off.

They interact with each other and tuning them requires special care.

Illustration

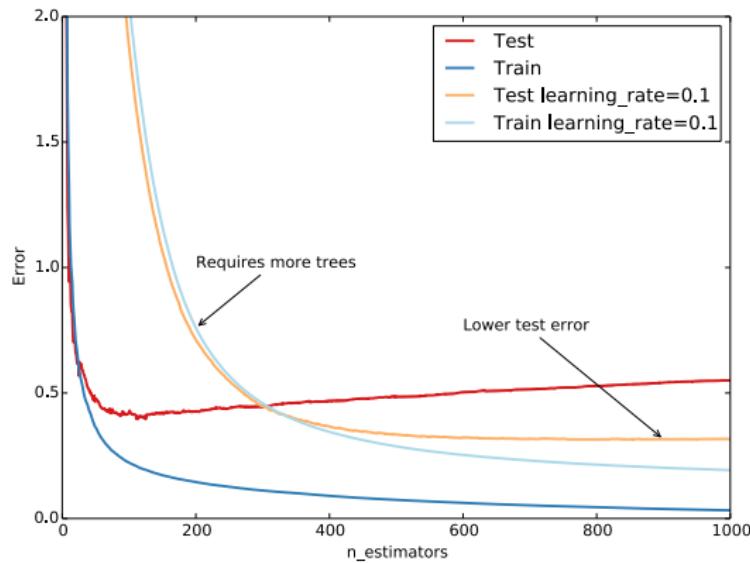
Impact of constraining leaf sizes



Source: <https://github.com/pprett/pydata-gbrt-tutorial>

Illustration

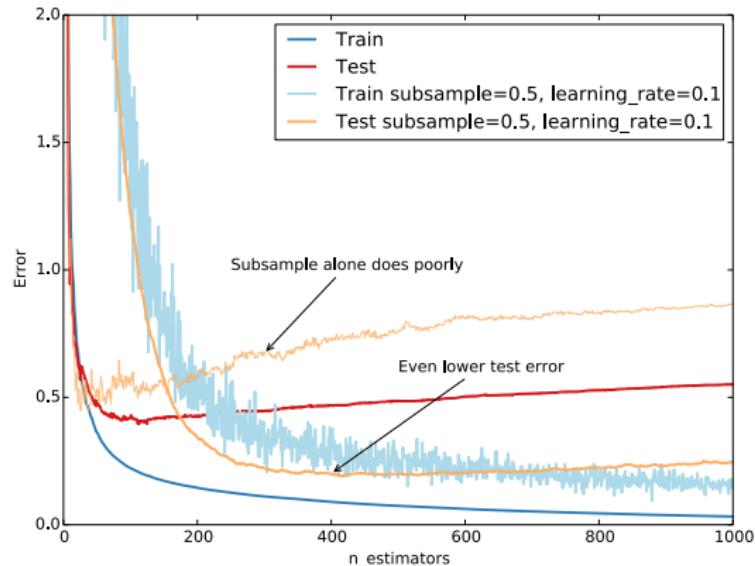
Impact of learning rate:



Lower learning rates requires more trees.

Illustration

Impact of subsampling



Runtime is furthermore reduced by half.

Source: <https://github.com/pprett/pydata-gbrt-tutorial>

Random forests (RF) vs gradient boosting with trees (GBT)

GBT uses ensembling for reducing **bias** (mostly), while RF uses ensembling for reducing **variance** (only).

GBT's benefits:

- ▶ Often yields better predictive performance than RF.
- ▶ Can be adapted more easily to any (differentiable) loss.
- ▶ Builds smaller trees/ensembles.

RF's benefits:

- ▶ Requires much less hyperparameter tuning. Works out of the box.
- ▶ Embarrassingly parallel.
- ▶ Requires less trick to be implemented efficiently.
- ▶ Better understood theoretically.

Bias/variance trade-off

Method	E	Bias	Variance
Full regression tree	10.2	3.5	6.7
Regression tree with 1 test	18.9	17.8	1.1
+ GBT ($T = 50$)	5.0	3.1	1.9
+ Bagging ($T = 50$)	17.9	17.3	0.6
Regression tree with 5 tests	11.7	8.8	2.9
+ GBT ($T = 50$)	6.4	1.7	4.7
+ Bagging ($T = 50$)	9.1	8.7	0.4

Application to our illustrative problem.

Boosting **reduces** the bias but **increases** the variance. However, with respect to full trees, it decreases both bias and variance.

A popular **implementation** of Gradient boosting with trees.

Some specificities:

- ▶ Instead of fitting the gradient, it fits a **second order** taylor expansion of the loss.
- ▶ Instead of using standard regression trees, the impurity measure is **adapted to the loss**.
- ▶ The loss (and thus the impurity) includes a term that penalises for complexity, which results in **automatic tree pruning**:

$$\Omega(\mathcal{T}) = \gamma N_{leaf} + \frac{\lambda}{2} \sum_{l=1}^{N_{leaf}} w_l^2$$

- ▶ Numerical inputs are potentially **binned** and only splits between bins are scored.
- ▶ Support GPU acceleration.

Outline

- ① Averaging techniques
- ② Boosting techniques
- ③ Other ensemble approaches
- ④ Conclusions on ensemble methods

Other ensemble approaches (i)

Bayesian model averaging:

$$P(y \mid \underline{x}, LS) = \sum_{h \in \mathcal{H}} P(y \mid h, LS)P(h \mid LS)$$

with

$$\begin{aligned} P(h \mid LS) &\propto P(h)P(LS \mid h) \\ &\propto P(h) \sum_{\theta} P(LS \mid \theta, h)P(\theta \mid h) \end{aligned}$$

where:

- $P(h)$ is the prior knowledge about models (e.g. simple models are more probable)
- $P(LS \mid h)$ is the quality of the fit

Other ensemble approaches (ii)

Stacking:

Learn a model to combine the models

- Let $LS = \{(\underline{x}_i, y_i) \mid i = 1, \dots, N\}$
- Let $A^t, t = 0, \dots, T$ be $T + 1$ learning algorithms
- For $t = 1, \dots, T$:
 1. Construct a model: $\hat{y}^t = A^t(LS)$
 2. Compute predictions: $\hat{y}_i^t = \hat{y}^t(\underline{x}_i)$
- Set $LS^0 = \{(\underline{x}_i^0, y_i)\}$ with $\underline{x}_i^0 = (\hat{y}_i^t)_{t=1}^T$
- Return $\hat{y} = A^0(LS^0)$

Outline

- ① Averaging techniques
- ② Boosting techniques
- ③ Other ensemble approaches
- ④ Conclusions on ensemble methods
 - Interpretability and efficiency
 - Conclusion

Interpretability and efficiency

Since we average several models, we lose the interpretability of the combined models and some efficiency.

However:

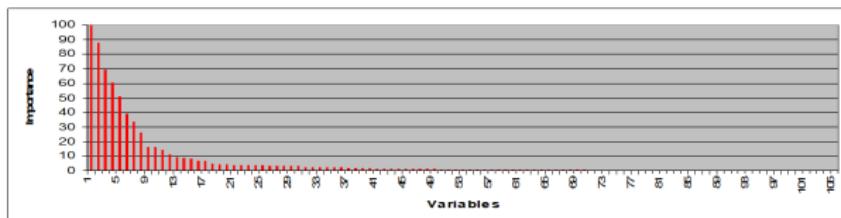
- With trees, we can still use the ensembles to compute variable importance by averaging over all trees. Actually, this even stabilizes the estimates.
- Averaging techniques can be parallelized and boosting type algorithms use smaller trees. Hence, the increase of computing times is not so detrimental.

Experiments on Golub's microarray data

- ▶ Input: 72 objects, 7129 numerical attributes (gene expressions), 2 classes (ALL and AL)
- ▶ Leave-one-out error with several variants

Method	Error
1 decision tree	22.2%(16/72)
Random forests ($k = 85$, $T = 500$)	9.7%(7/72)
Extra-trees ($s_{th} = 0.5$, $T = 500$)	5.5%(4/72)
Adaboost (1 test node, $T = 500$)	1.4%(1/72)

- ▶ Variable importance with boosting



Conclusion

Ensemble methods are very effective techniques to **reduce** bias and/or variance. They can transform a not-so-good method to a competitive method in terms of accuracy.

Random forests and gradient boosting with trees are two very competitive methods, with the first easier to use and the second often more accurate.

Interpretability of the model and efficiency of the method are **difficult** to preserve if we want to reduce variance significantly.

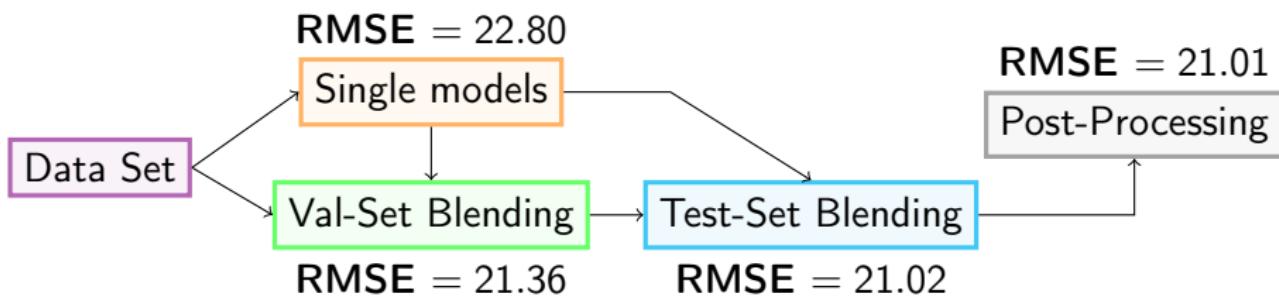
There are other ways to tackle the variance/overfitting problem:

- Bayesian approaches (\sim averaging techniques)
- Support vector machines, which maintain a low variance by maximizing the classification margin

Machine learning challenges

Machine learning challenges are commonly won by ensemble solutions:

- ▶ Netflix prize (\$1M reward): best solution combines 107 models obtained from different methods (*stacking*)
- ▶ Yahoo! 2011 KDD Cup (\$5 000 reward): best solution uses two levels of stacking



Further reading and software

Further reading:

- ▶ Hastie et al.: 8.7, 8.8, 10.1-4, 15 (not in detail)

Software:

- ▶ Many ensemble methods are available in common ML libraries (scikit-learn, R)
- ▶ Boosting:
 - XGBoost: <http://xgboost.readthedocs.io/>
 - Light-GBM: <https://github.com/Microsoft/LightGBM>
- ▶ Demos:
 - http://arogozhnikov.github.io/2016/07/05/gradient_boosting_playground.html
 - <https://cs.stanford.edu/~karpathy/svmjs/demo/demoforest.html>

Part II

Feature selection

Outline

① Motivation and formalization

② Filter techniques

③ Embedded techniques

④ Wrapper methods

⑤ Selection bias

⑥ Further reading

Motivation

Feature selection is a technique to **reduce** the number of features used by the learning algorithm.

It has several advantages:

- Avoid overfitting and improve model performance
- Improve interpretability
- Provide faster and more cost-effective models
- Reduce overall computing times, if the feature selection technique is fast

Feature selection vs Ranking

Feature selection:

Find a small (or the smallest) subset of features that maximizes accuracy.

Feature ranking:

Sort the variables according to their relevance at predicting the output.

There exist techniques in both families.

Note: Feature selection can be obtained from feature ranking, e.g. by selecting the top k features in a ranking.

Formalization

Let Y denote the class variable and $V = \{X_1, \dots, X_p\}$ the set of input variables.

A feature X_i is:

- **strongly relevant** iff $P(Y | X_i, V \setminus X_i) \neq P(Y | V \setminus X_i)$
- **weakly relevant** iff it is not strongly relevant and $P(Y | X_i, S) \neq P(Y | S)$ for some subset $S \subset V$
- **irrelevant** otherwise

A subset $M \subseteq V$ of variables is a **Markov boundary** for Y if it is minimal and $P(Y | M, V \setminus M) = P(Y | M)$: features in M are either weakly or strongly relevant. They are all strongly relevant when the distribution P is strictly positive.

Feature selection is often formulated as finding a Markov boundary M for Y .

Note: All variables in a Markov boundary do not necessarily appear in the Bayes model (depending on the loss function).

Feature selection

There are three main approaches:

- ▶ **Filter techniques**: a priori selection of the variables, *i.e.* independently of the supervised learning algorithm.
- ▶ **Embedded techniques**: feature selection is embedded in the learning algorithm.
- ▶ **Wrapper methods**: use cross-validation to find the optimal set of features for a given algorithm.

Filter techniques

Idea: associate a relevance score to each feature and remove the low-scoring ones.

Univariate scoring (often used):

- ▶ Any score measures used in decision trees
- ▶ Statistical test (t-test, chi-square, etc.)

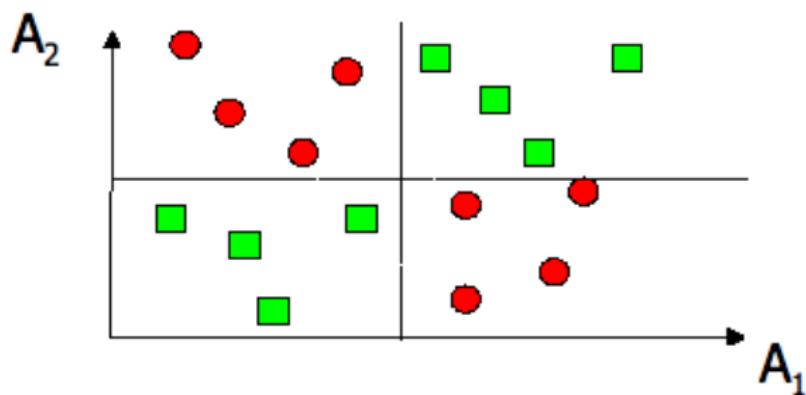
Multivariate scoring:

- ▶ Relief
- ▶ Markov blanket filter
- ▶ Decision trees
- ▶ ...

The optimal number of features can be determined by cross-validation.

Univariate scoring vs Multivariate scoring

Each feature is useless alone (low univariate scoring) but together they perfectly explain the classification.



Conclusions

Advantages:

- ▶ Univariate scoring is fast and scalable.
- ▶ Independent of the supervised learning algorithm.

Drawbacks:

- ▶ It ignores the supervised learning algorithm.
- ▶ Univariate scoring ignores feature dependencies.
- ▶ Multivariate scoring is slower than univariate scoring.

Embedded techniques

Some supervised learning methods embed feature selection. The search for an optimal subset of features is built into the learning algorithm.

Examples:

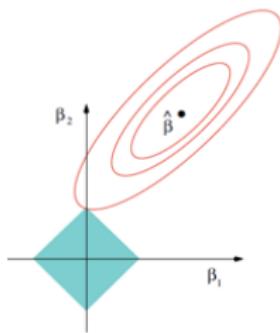
- Decision tree node splitting is a feature selection technique.
- Tree ensemble variable importance measures.
- Absolute weights in a linear SVM model:

$$\hat{y}(\underline{x}) = \text{sgn}\left(\sum_i w_i x_i + b\right)$$

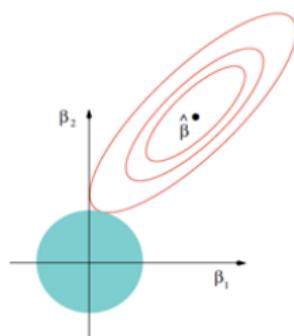
LASSO

Linear model learned with L1 penalization.

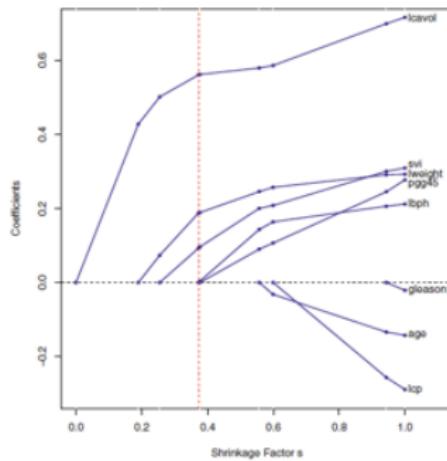
$$\min_{\beta} \sum_{i=1}^N (y_i - (\beta_0 + \sum_j \beta_j x_j))^2 + \lambda \sum_j |\beta_j|$$



LASSO



Ridge



Influence of λ .

Conclusions

Advantages:

- ▶ Usually computationally efficient.
- ▶ Well integrated within the learning algorithm.
- ▶ Multivariate.

Drawbacks:

- ▶ Specific to the learning algorithm.

Wrapper methods

Idea: Try to find a subset of features that maximizes the quality of the model induced by the learning algorithm. The quality is assessed by cross-validation.

As the number of subsets of p features is 2^P , all subsets can usually **not** be evaluated, and heuristics are necessary.

Many approaches exist:

- Forward/Backward selection: add (resp. remove) the variable that most decreases (resp. less increases) the error.
- Optimization by genetic algorithms.

Recursive feature elimination (i)

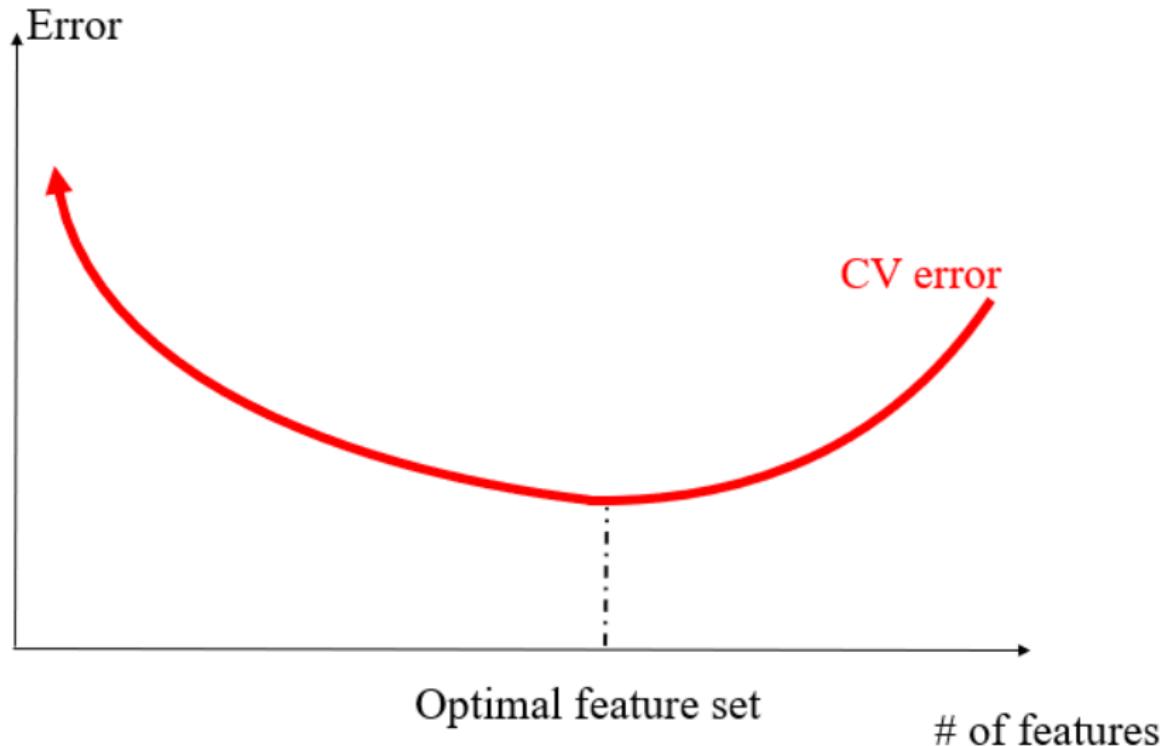
It is a popular wrapper, especially in bioinformatics.

Assume that we have a learning algorithm that can **rank** the features (e.g. linear SVM, decision trees).

Iterate:

1. Learn a model from the current feature set.
 2. Rank the features with the model.
 3. Remove the feature with the smallest ranking.
- Keep the feature set that yields the lowest cross-validation error.

Recursive feature elimination (ii)



Conclusions

Advantages:

- ▶ Custom-tailored to the learning algorithm.
- ▶ Find interactions and remove redundant variables.

Drawbacks:

- ▶ Prone to **overfitting**: it is often easy to find a small subset of noisy features that discriminates perfectly the classes.
- ▶ Expensive: we have to build a model for each subset of variables.

Selection bias (i)

We often see this experiment:

- Select the N top variables using some filter on the full dataset.
- Evaluate an algorithm that uses these N variables as inputs by cross-validation (e.g. LOO) on the dataset.

What is wrong with this protocol?

Selection bias (ii)

A1	A2	...	A1000	Y
-0.86	0.17	...	0	C2
-2.3	-1.2	...	-0.42	C1
-0.37	-0.11	...	-0.64	C1
0.41	0.67	...	-0.8	C2
-0.51	-0.59	...	0.98	C2
-0.25	-0.27	...	-0.68	C1
-0.52	0.23	...	0.11	C1
-1.3	-0.2	...	0.14	C1
0.93	-0.78	...	-0.01	C2
-0.25	-0.29	...	0.69	C2
-0.6	0.92	...	-0.64	C1
0.22	-0.8	...	-0.5	C2
-0.62	0.2	...	0.08	C1
-0.3	0.8	...	0.02	C2
-0.91	0.44	...	-0.57	C1
0.76	0.65	...	-0.08	C1

Dataset composed of 250 objects. Each object has 1 000 features and a single class. Each feature is randomly drawn from $\mathcal{N}(0, 1)$, and the class is selected at random.

Selection bias (iii)

Let us consider two trials:

- Tree bagging without feature selection → 10-fold cross-validation error $\approx 52\%$
- Tree bagging with 20 top features (t-test) → 10-fold cross-validation error $\approx 35\%$

One could conclude that there are 20 interesting variables, and that, from them, one can classify better than at random.

However, on a new set of 250 samples, the error is 52%.

Selection bias (iv)

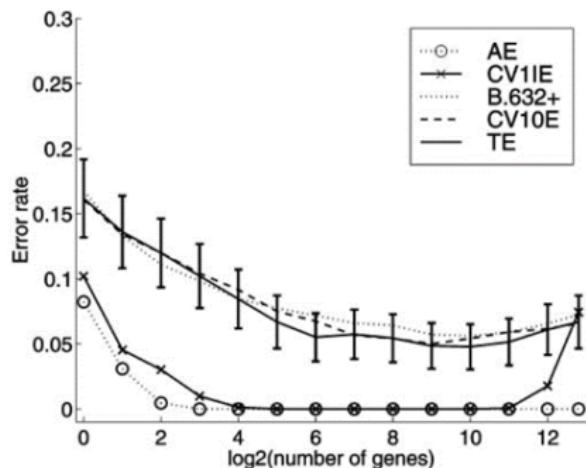
We have both selected the variables **and** tested the model on the basis of the whole training set \Rightarrow the CV error estimate is too optimistic (overfitting).

A correct protocol to assess the model would be:

- Divide the learning sample into 10 folds.
- For $i = 1$ to 10:
 1. Remove the i -th fold from the learning sample.
 2. Select the top 20 variables from the remaining objects.
 3. Learn the model using the 20 variables and the remaining objects.
 4. Test the model on the i -th fold.

Selection bias (v)

- ▶ AE = error on the learning set
- ▶ CV1IE = internal LOO
- ▶ CV10E = external 10-fold cross-validation
- ▶ TE = error on an independent test set
- ▶ B.632+ = another unbiased error estimate



Source: [3]

Outline

① Motivation and formalization

② Filter techniques

③ Embedded techniques

④ Wrapper methods

⑤ Selection bias

⑥ Further reading

Further reading and software

Further reading:

- Hastie et al.: chapter 18.
- Guyon and Elisseeff, An introduction to variable and feature selection.
<http://www.jmlr.org/papers/volume3/guyon03a/guyon03a.pdf>
<http://clopinet.com/fextract-book/IntroFS.pdf>
- Saeys et al. A review of feature selection techniques in bioinformatics.
<http://dx.doi.org/10.1093/bioinformatics/btm344>

Software:

- Many feature selection methods (filters, embedded or wrappers) are implemented in common ML libraries (scikit-learn, R).
- For example: https://scikit-learn.org/stable/modules/feature_selection.html.

References

-  Jamie Shotton, Andrew Fitzgibbon, Andrew Blake, Alex Kipman, Mark Finocchio, Bob Moore, and Toby Sharp.
Real-time human pose recognition in parts from a single depth image.
In *CVPR*. IEEE, June 2011.
Best Paper Award.
-  Christopher M Bishop.
Pattern recognition and machine learning.
springer, 2006.
-  Christophe Ambroise and Geoffrey J McLachlan.
Selection bias in gene extraction on the basis of microarray gene-expression data.
Proceedings of the national academy of sciences,
99(10):6562–6566, 2002.

Unsupervised learning

Pierre Geurts & Louis Wehenkel

Institut Montefiore, University of Liège, Belgium



ELEN062-1
Introduction to Machine Learning
November 2020

Acknowledgment: These slides have been reformatted by Yann Claes in July 2020.

Outline

- ① Motivation
- ② Clustering
- ③ Dimensionality reduction
- ④ Conclusions and further readings

Outline

- ① Motivation
- ② Clustering
- ③ Dimensionality reduction
- ④ Conclusions and further readings

Motivation

Unsupervised learning tries to find any **regularities** in the data **without** guidance about inputs and outputs.

- ▶ Are there interesting groups of variables or samples?
- ▶ What are the dependencies between variables?

Many families of problems exist, among which:

- ▶ Clustering: try to find natural groups of samples/variables.
Examples: hierarchical clustering, k-means.
- ▶ Dimensionality reduction: project the data from a high-dimensional space down to a small number of dimensions.
Examples: principal/independent component analysis, MDS.
- ▶ Density estimation: determine the distribution of data within the input space.
Examples: Bayesian networks, mixture models.

Outline

① Motivation

② Clustering

Principles

Hierarchical clustering

Combinatorial clustering and k-means clustering

Self-organizing maps

How many clusters?

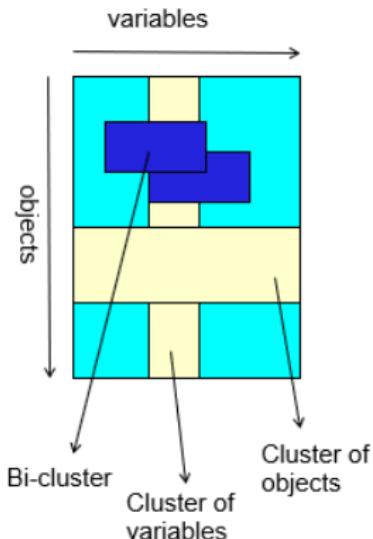
Feature selection and selection bias

③ Dimensionality reduction

④ Conclusions and further readings

Principles of clustering

Goal: group a collection of objects into subsets (*clusters*) such that those within each cluster are more closely related to one another than to objects from different clusters.



- ▶ Clustering **rows**: grouping similar objects.
- ▶ Clustering **variables**: grouping similar variables across samples.
- ▶ **Bi-clustering**: grouping objects that are similar across a subset of variables.

Applications

There are applications in many domains:

- ▶ Marketing: finding groups of customers with similar behavior given a large database of customer data containing their properties and past buying records.
- ▶ Biology: classification of plants and animals given their features.
- ▶ Insurance: identifying groups of motor insurance policy holders with a high average claim cost; identifying frauds.
- ▶ City-planning: identifying groups of houses according to their house type, value and geographical location.
- ▶ Earthquake studies: clustering observed earthquake epicenters to identify dangerous zones.
- ▶ WWW: document classification; clustering weblog data to discover groups of similar access patterns.

Main components of clustering

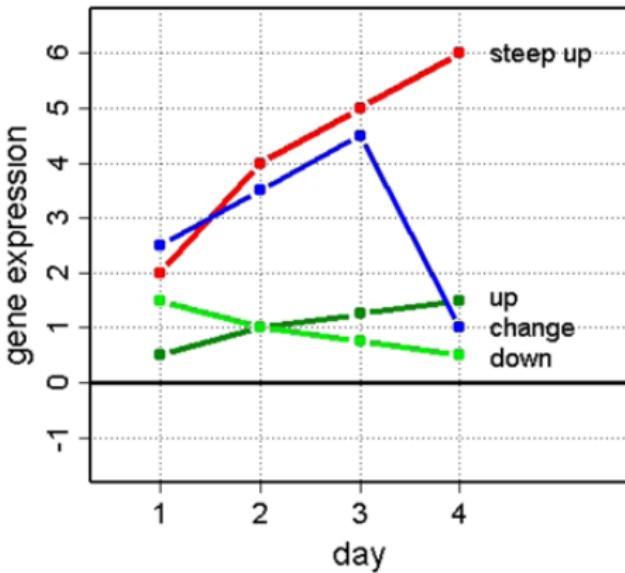
Two components are essential in cluster analysis:

- **Distance measure**: a notion of distance/similarity of two objects
→ When are two objects close to each other?
- **Cluster algorithm**: a procedure to minimize distances of objects within groups and/or maximize distances between groups.

Example

Let us assume that we have access to the measurement of gene expression on 4 consecutive days. Every gene is coded by a vector of length 4.

- **Steep up:** $x_1 = (2, 4, 5, 6)$
- **Up:** $x_2 = \left(\frac{2}{4}, \frac{4}{4}, \frac{5}{4}, \frac{6}{4}\right)$
- **Down:** $x_3 = \left(\frac{6}{4}, \frac{4}{4}, \frac{3}{4}, \frac{2}{4}\right)$
- **Change:** $x_4 = \left(\frac{5}{2}, \frac{7}{2}, \frac{9}{2}, \frac{2}{2}\right)$



Example - Euclidean distance

The distance between two vectors is the square root of the sum of the squared differences over all coordinates.

Example:

$$d_E(x_1, x_2) = \sqrt{(2 - \frac{2}{4})^2 + (4 - \frac{4}{4})^2 + (5 - \frac{5}{4})^2 + (6 - \frac{6}{4})^2} = 6.75$$

$$d = \begin{bmatrix} 0 & 6.75 & 7.59 & 5.07 \\ 6.75 & 0 & 1.5 & 4.59 \\ 7.59 & 1.5 & 0 & 4.64 \\ 5.07 & 4.59 & 4.64 & 0 \end{bmatrix}$$

Example - Manhattan distance

The distance between two vectors is the sum of the absolute (non-squared) differences over all coordinates.

Example:

$$d_M(x_1, x_2) = \left|2 - \frac{2}{4}\right| + \left|4 - \frac{4}{4}\right| + \left|5 - \frac{5}{4}\right| + \left|6 - \frac{6}{4}\right| = 12.75$$

$$d = \begin{bmatrix} 0 & 12.75 & 13.25 & 6.5 \\ 12.75 & 0 & 2.5 & 8.25 \\ 13.25 & 2.5 & 0 & 7.75 \\ 6.5 & 8.25 & 7.75 & 0 \end{bmatrix}$$

Example - Correlation distance

The distance between two vectors is $1 - \rho$, where ρ is the Pearson correlation of the two vectors.

Example: $d_c(x1, x2) =$

$$\frac{\left(2 - \frac{17}{4}\right)\left(\frac{2}{4} - \frac{17}{16}\right) + \left(4 - \frac{17}{4}\right)\left(\frac{4}{4} - \frac{17}{16}\right) + \left(5 - \frac{17}{4}\right)\left(\frac{5}{4} - \frac{17}{16}\right) + \left(6 - \frac{17}{4}\right)\left(\frac{6}{4} - \frac{17}{16}\right)}{\sqrt{\left(2 - \frac{17}{4}\right)^2 + \left(4 - \frac{17}{4}\right)^2 + \left(5 - \frac{17}{4}\right)^2 + \left(6 - \frac{17}{4}\right)^2} \sqrt{\left(\frac{2}{4} - \frac{17}{16}\right)^2 + \left(\frac{4}{4} - \frac{17}{16}\right)^2 + \left(\frac{5}{4} - \frac{17}{16}\right)^2 + \left(\frac{6}{4} - \frac{17}{16}\right)^2}}$$

$$d = \begin{bmatrix} 0 & 0 & 2 & 1.18 \\ 0 & 0 & 2 & 1.18 \\ 2 & 2 & 0 & 0.82 \\ 1.18 & 1.18 & 0.82 & 0 \end{bmatrix}$$

Example - Comparison of distances

Euclidian

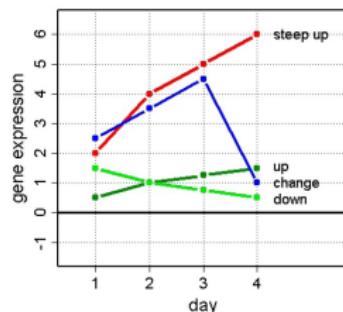
0	6.75	7.59	5.07
6.75	0	1.5	4.59
7.59	1.5	0	4.64
5.07	4.59	4.64	0

Manhattan

0	12.75	13.25	6.5
12.75	0	2.5	8.25
13.25	2.5	0	7.75
6.5	8.25	7.75	0

Correlation

0	0	2	1.18
0	0	2	1.18
2	2	0	0.82
1.18	1.18	0.82	0



	steep up	up	down	change
steep up	0 0 0	9 9 0	10 10 10	7 4 5
up	9 9 0	0 0 0	2 1 10	6 6 5
down	10 10 10	2 1 10	0 0 0	6 5 4
change	7 4 5	6 6 5	6 5 4	0 0 0

All distances are normalized to the interval [0,10] and then rounded

Clustering algorithms

There exist many clustering algorithms:

- ▶ Hierarchical clustering
- ▶ k-means
- ▶ Self-organizing maps (SOM)
- ▶ Autoclass, mixture models, ...

Hierarchical clustering allows the choice of the dissimilarity matrix.

k-means and SOMs take the original data directly as input. Attributes are assumed to live in the Euclidean space.

Hierarchical clustering

There exist two strategies for hierarchical clustering: **agglomerative** clustering and **divisive** clustering. We will cover only agglomerative clustering:

1. Each object is assigned to its own cluster
2. Iteratively:
 - The two most similar clusters are joined and replaced by a new one
 - The distance matrix d between clusters is updated with this new cluster, replacing the two joined ones.

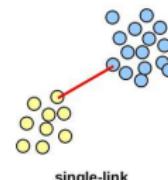
Note: Divisive clustering would start from a big cluster. *Why is it less popular?*

Distance between two clusters

There exist several ways of measuring the distance between two clusters:

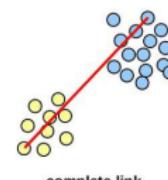
- **Single linkage**, that uses the smallest distance:

$$d_S(G, H) = \min_{i \in G, j \in H} d_{ij}$$



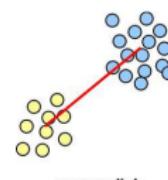
- **Complete linkage**, that uses the largest distance:

$$d_C(G, H) = \max_{i \in G, j \in H} d_{ij}$$



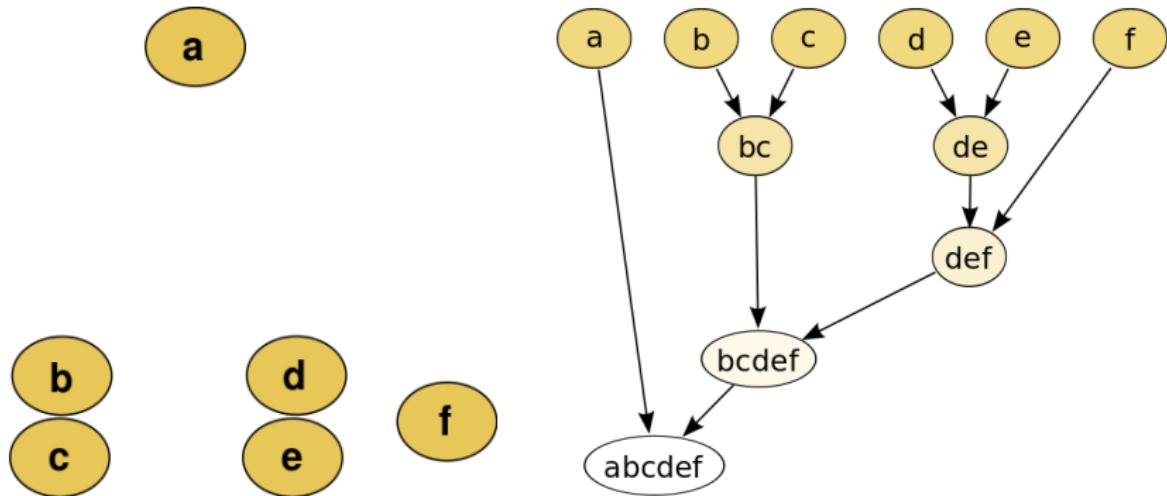
- **Average linkage**, that uses the average distance:

$$d_A(G, H) = \frac{1}{N_G N_H} \sum_{i \in G} \sum_{j \in H} d_{ij}$$



Source: [1]

Hierarchical clustering - Visualization



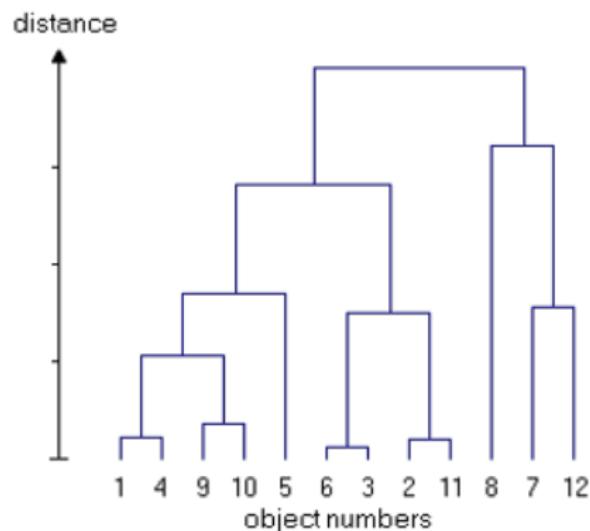
Source: [2]

Dendograms

A useful tool to visualize hierarchical clustering is **dendograms**.

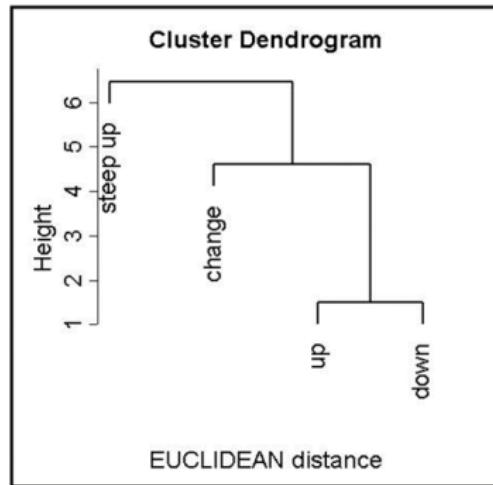
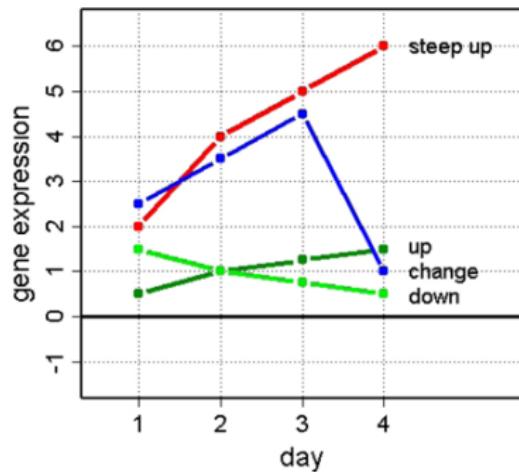
Clusters that are joined are combined by a line; the height of the line corresponds to the distance between these clusters.

Dendograms can be used to determine visually the number of clusters.



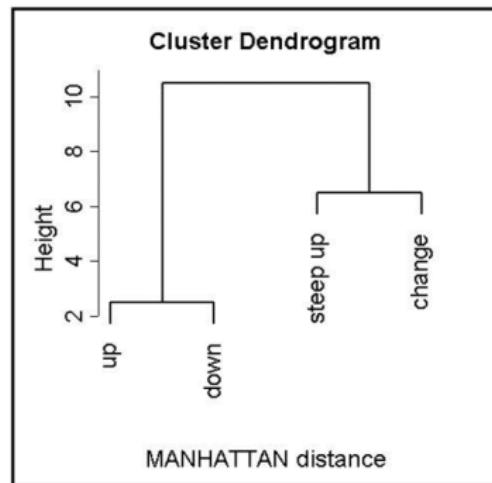
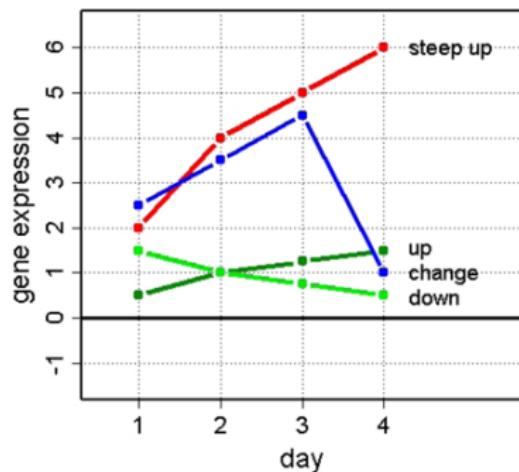
Dendrograms - Time series example (i)

Euclidean distance: similar values are clustered together.



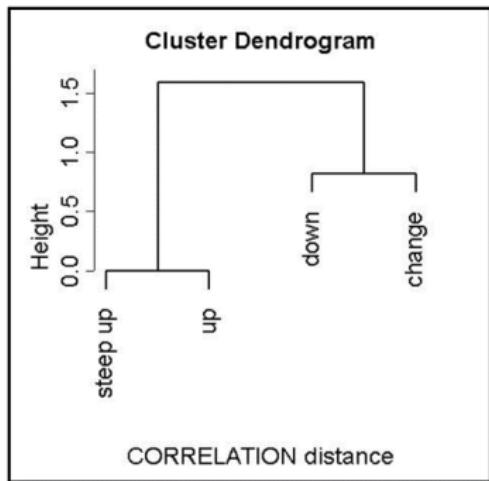
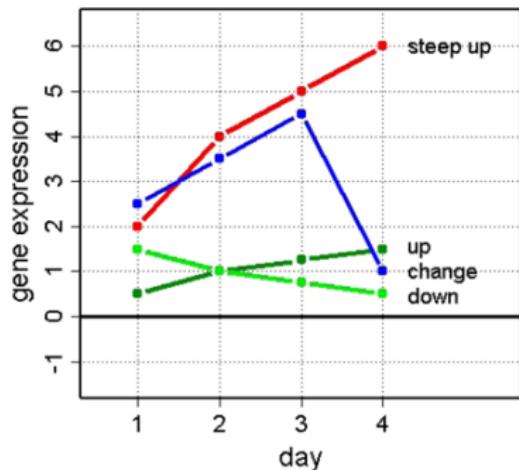
Dendrograms - Time series example (ii)

Manhattan distance: similar values are clustered together (robust).



Time series example (iii)

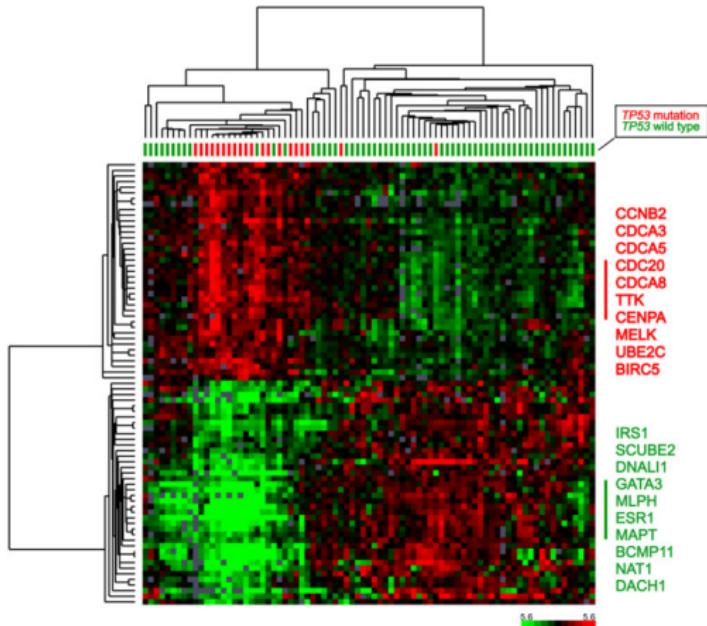
Correlation distance: correlated values are clustered together.



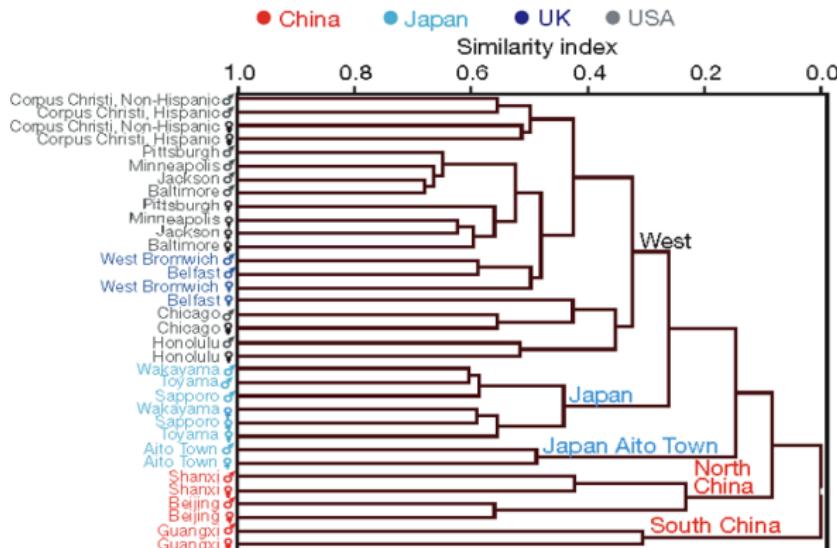
Illustrations of hierarchical clustering (i)

Hierarchical clustering applied to identify (and explain) subgroups of breast cancer patients [3].

It was conducted on 80 tumor samples (wild-type, TP53 mutated) and 80 genes.



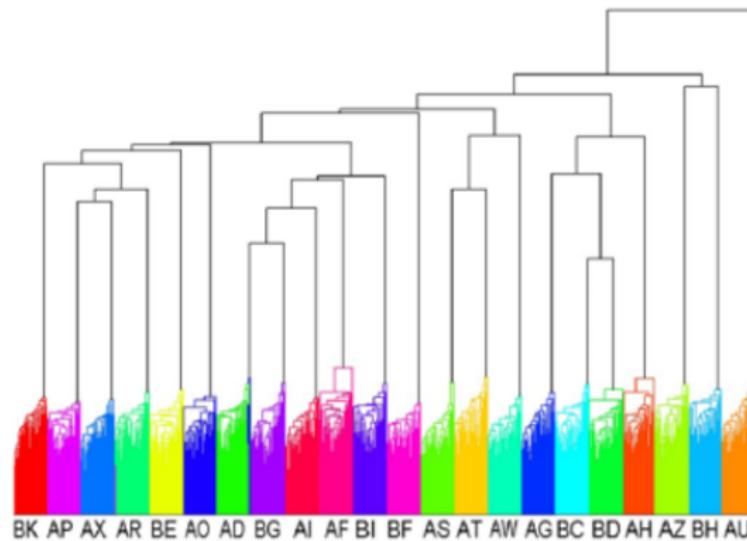
Illustrations of hierarchical clustering (ii)



Source: [4]

In this example, the authors use the Euclidean distance and the average linkage metric.

Illustrations of hierarchical clustering (iii)



Source: [5]

This example underlines the evidence of different metabolic phenotypes in humans using urine samples of 22 volunteers over 3 months. NMR spectra were analyzed by HCA.

Hierarchical clustering - Conclusions

Strengths:

- ▶ No need to assume any particular number of clusters.
- ▶ Can use any distance metric.
- ▶ Can sometimes find a meaningful taxonomy.

Limitations:

- ▶ Can find a taxonomy even if it does not exist.
- ▶ Once a decision is made to combine two clusters, it cannot be undone.
- ▶ Not well motivated from a theoretical point of view.

Outline

① Motivation

② Clustering

Principles

Hierarchical clustering

Combinatorial clustering and k-means clustering

Self-organizing maps

How many clusters?

Feature selection and selection bias

③ Dimensionality reduction

④ Conclusions and further readings

Combinatorial clustering algorithm

Given a number of clusters $K < N$ and an encoder C that assigns the i -th observation to a cluster $C(i)$, clustering aims at finding C^* that **minimizes** some loss function, which measures the degree to which the clustering goal is not met.

Example of loss function: the within-cluster scatter

$$W(C) = \frac{1}{2} \sum_{k=1}^K \frac{1}{N_k} \sum_{C(i)=k} \sum_{C(i')=k} d(x_i, x_{i'})$$

$$\text{with } N_k = \sum_{i=1}^N I(C(i) = k)$$

However, the number of possible assignments is too high for enumeration:

$$S(N, K) = \frac{1}{K!} \sum_{k=1}^K (-1)^{K-k} \binom{K}{k} k^N$$

Examples: $S(10, 4) = 34105$, $S(19, 4) = 10^{10}$

k-means clustering (i)

k-means clustering is a partitioning algorithm with a **prefixed** number of k clusters.

It uses the **Euclidean** distance between objects

$$d(x_i, x_{i'}) = \sum_{j=1}^p (x_{ij} - x_{i'j})^2 = \|x_i - x_{i'}\|^2$$

k-means clustering (ii)

It tries to **minimize** the sum of intra-cluster variances:

$$\begin{aligned} W(C) &= \frac{1}{2} \sum_{k=1}^K \frac{1}{N_k} \sum_{C(i)=k} \sum_{C(i')=k} \|x_i - x_{i'}\|^2 \\ &= \sum_{k=1}^K \sum_{C(i)=k} \|x_i - \bar{x}_k\|^2 \end{aligned}$$

where $\bar{x}_k = (\bar{x}_{1k}, \dots, \bar{x}_{pk})$ is the center of cluster k and N_k is the number of points in cluster k :

$$N_k = \sum_{i=1}^N I(C(i) = k)$$

It is **equivalent** to solve:

$$\min_{C, \{m_k\}_1^K} \sum_{k=1}^K \sum_{C(i)=k} \|x_i - m_k\|^2$$

k-means clustering (iii)

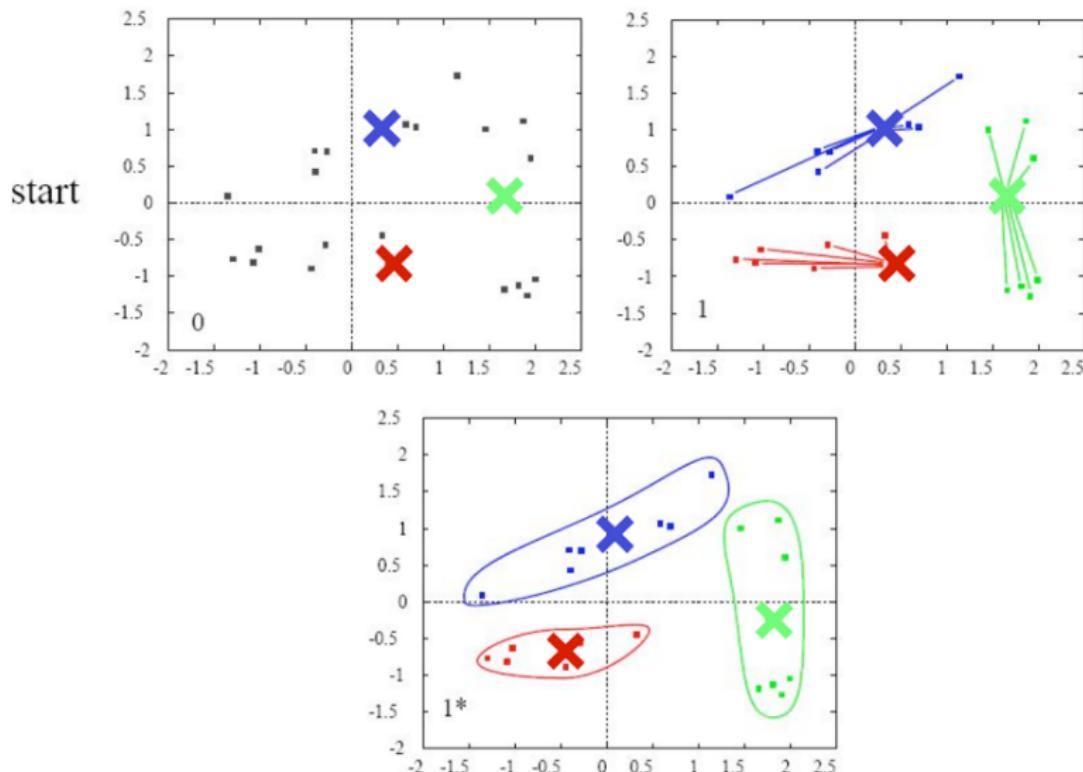
The algorithm is the following:

1. Randomly assign each point to a cluster.
2. Iterate:
 - Given the current cluster assignment, compute the cluster means $\{m_1, \dots, m_K\}$
 - Given the current cluster means, assign each observation to the closest cluster mean

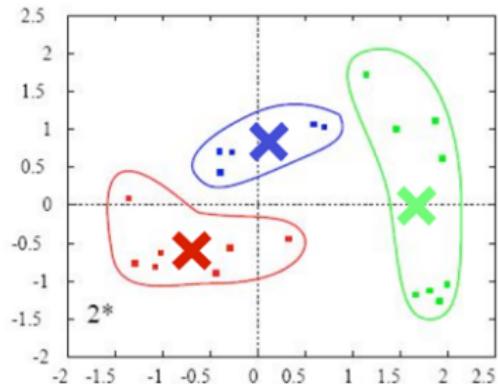
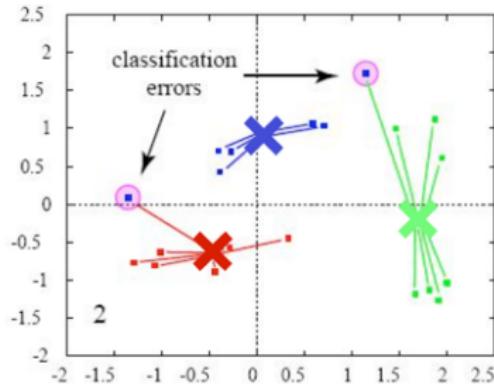
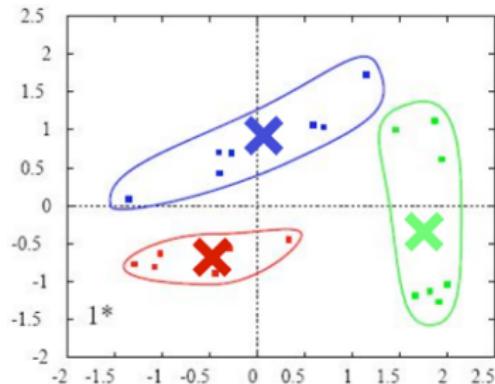
$$C(i) = \operatorname{argmin}_{1 \leq k \leq K} \|x_i - m_k\|^2$$

3. Stop when the assignment does not change anymore.

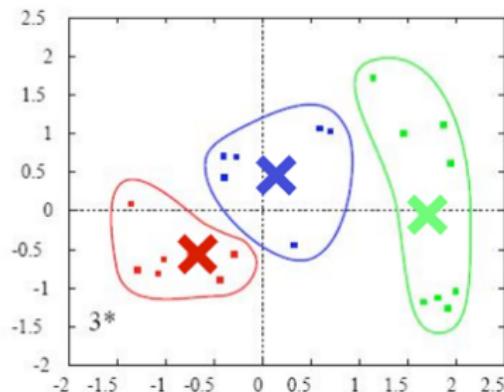
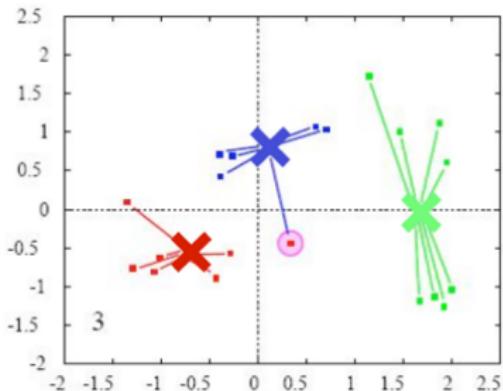
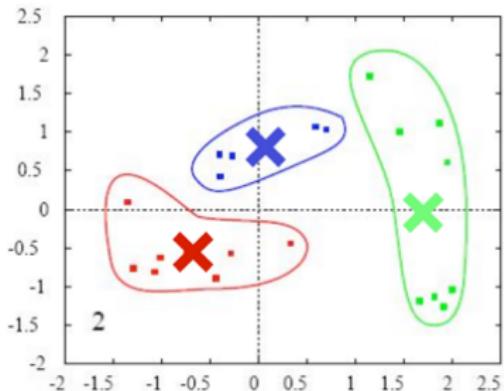
k-means clustering - Example (i)



k-means clustering - Example (ii)



k-means clustering - Example (iii)

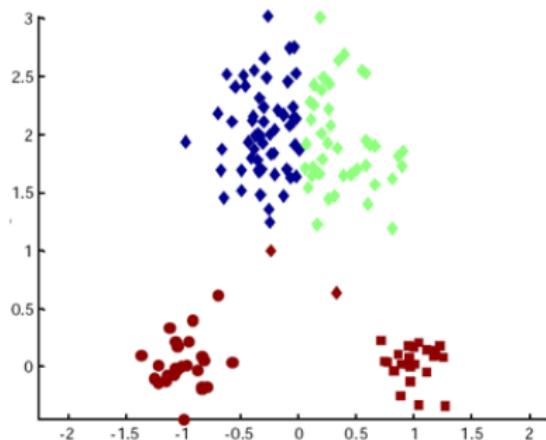
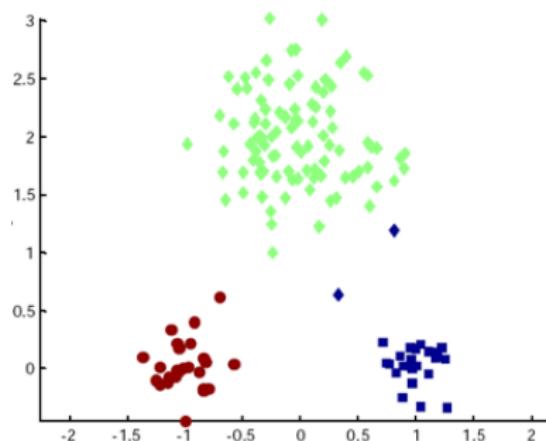


end

k-means clustering - Convergence

Each step reduces the within-cluster scatter (*can you prove it?*).
Hence, convergence is ensured but towards a local optimum only.

One could obtain any of these clusterings from a random start of k-means:



⇒ **Solution:** restart the algorithm several times and pick the clustering that minimizes $W(C)$.

k-means clustering - Application

k-means clustering can be applied for vector quantization:

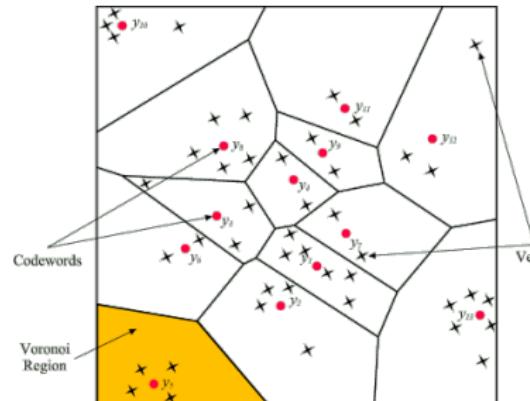
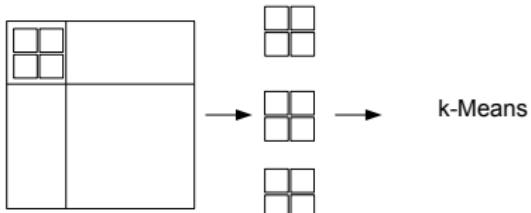


FIGURE 14.9. Sir Ronald A. Fisher (1890-1962) was one of the founders of modern day statistics, to whom we owe maximum-likelihood, sufficiency, and many other fundamental concepts. The image on the left is a 1024×1024 grayscale image at 8 bits per pixel. The center image is the result of 2×2 block VQ, using 200 code vectors, with a compression rate of 1.9 bits/pixel. The right image uses only four code vectors, with a compression rate of 0.50 bits/pixel

Source: [6]



k-medoids

k-medoids is an extension of k-means designed to handle any distance measure.

Algorithm 14.2 *K-medoids Clustering.*

1. For a given cluster assignment C find the observation in the cluster minimizing total distance to other points in that cluster:

$$i_k^* = \operatorname{argmin}_{\substack{\{i: C(i)=k\} \\ C(i')=k}} \sum D(x_i, x_{i'}). \quad (14.35)$$

Then $m_k = x_{i_k^*}$, $k = 1, 2, \dots, K$ are the current estimates of the cluster centers.

2. Given a current set of cluster centers $\{m_1, \dots, m_K\}$, minimize the total error by assigning each observation to the closest (current) cluster center:

$$C(i) = \operatorname{argmin}_{1 \leq k \leq K} D(x_i, m_k). \quad (14.36)$$

3. Iterate steps 1 and 2 until the assignments do not change.
-

It is **much slower** than k-means (*why?*).

k-means clustering - Conclusions

Strengths:

- ▶ Simple and understandable.
- ▶ Can cluster any new point (\neq hierarchical clustering).
- ▶ Well motivated theoretically

Limitations:

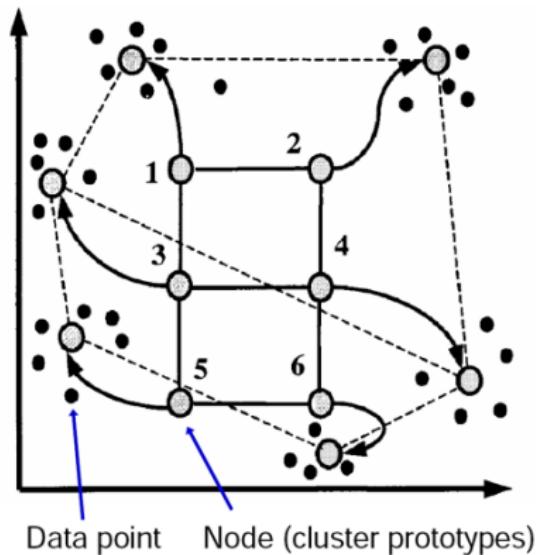
- ▶ Must fix the number of clusters beforehand.
- ▶ Sensitive to the initial choice of cluster centers.
- ▶ Sensitive to outliers.

Self-organizing maps

SOMs are similar to k-means clustering but with additional constraints. It consists in a **mapping** of the data space onto a one/two-dimensional array of k nodes.

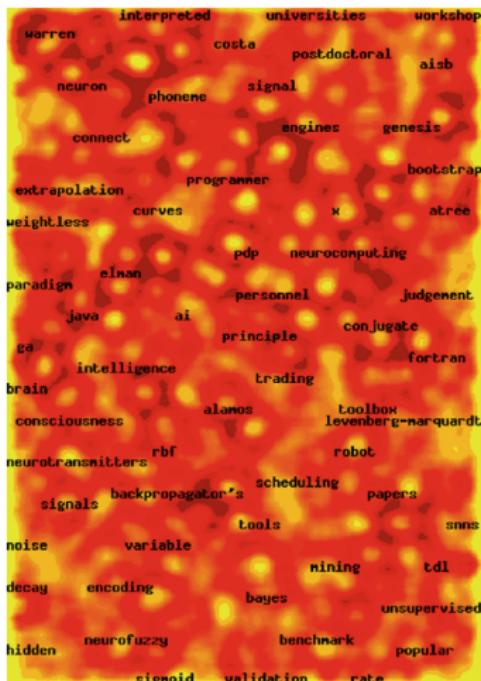
Iteration steps:

1. Pick a data point P at random.
2. Move all nodes in the direction of P : the closer (resp. further) a node, the more (resp. less) one moves it.
3. Decrease the amount of movement with the iteration steps.



Self-organizing maps - Application

One can apply self-organizing maps to document organization.



Source: [7]

How many clusters? (i)

Where should one stop hierarchical clustering? How should one choose k for k-means clustering and self-organizing maps?

These are very difficult and open questions.

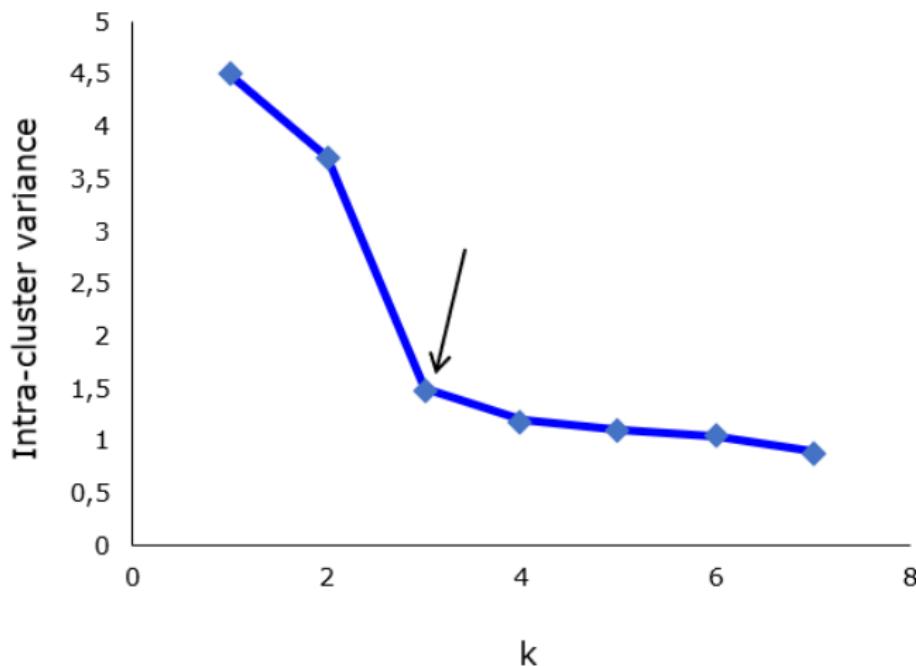
It is similar to overfitting in supervised learning:

- Too many clusters: overfit the data → find non-existing clusters in the data (noise).
- Too few clusters: underfit the data → miss some truly existing clusters.

However, there is no possibility to cross-validate.

How many clusters? (ii)

A criterion to choose the number of clusters is to locate the **knee** in the intra-cluster variance curve.



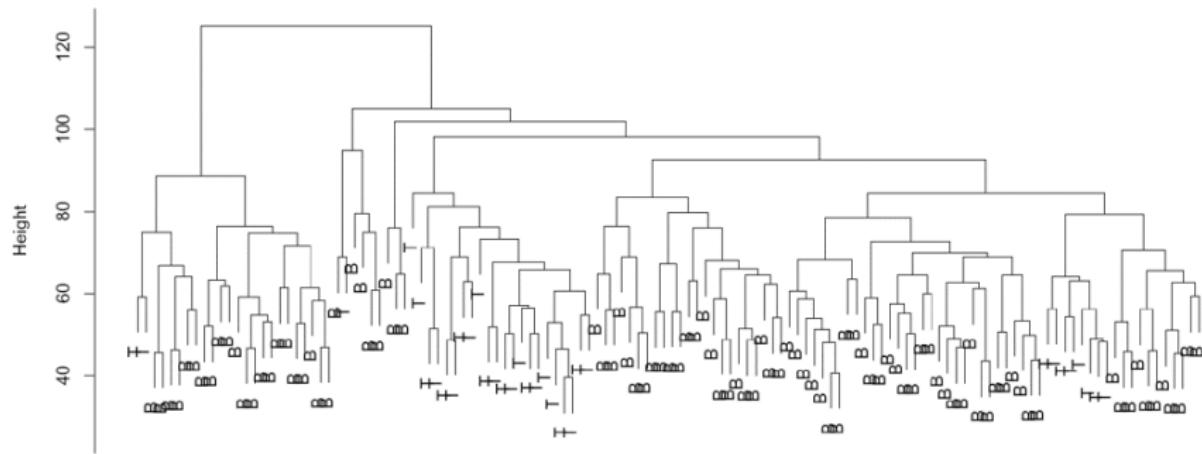
How many clusters? (iii)

There exist other criteria:

- Internal indices: a statistics based on within- and between-clusters distances. One should select k that minimizes/maximizes such index.
- Gap statistic: a resampling method that compares some internal index with what would be obtained from random data. One should search for the value of k that maximizes this difference [8].
- Stability: one should select k that leads to the most stable clusters. It is computed by a bootstrap analysis [9].

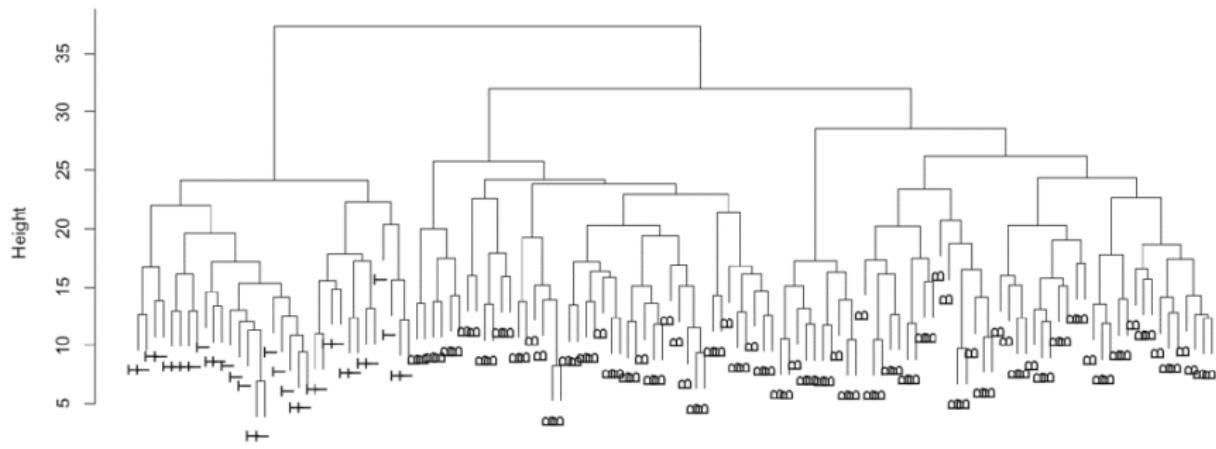
Feature selection for clustering (i)

Feature selection can also improve clustering by decreasing both noise and computing times.



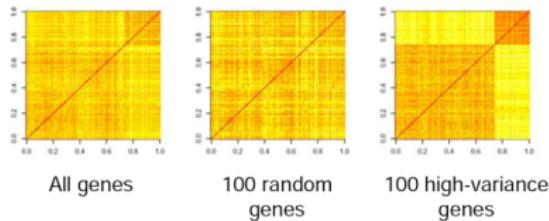
Example on Leukemia patients (**without** gene selection). Source: [10]

Feature selection for clustering (ii)

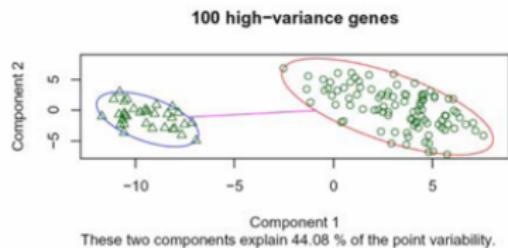
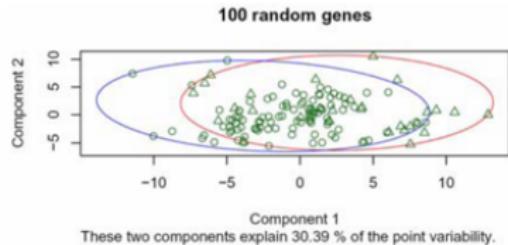


Example on Leukemia patients (with 100 top variance genes). Source: [10]

Feature selection for clustering (iii)



Distance matrices. Source: [10]

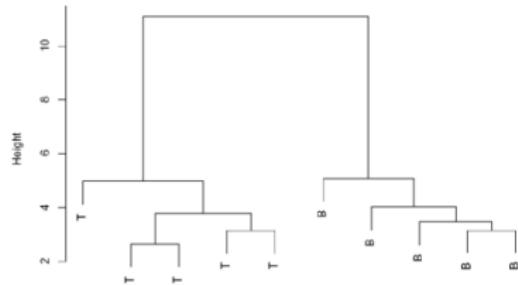


Sample types in the first two principal components. Source: [10]

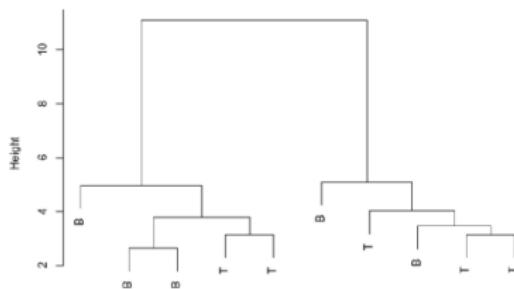
Selection bias in clustering

⚠ Clustering after **supervised** feature selection should be avoided! ⚠

Indeed, one will always retrieve the classification since this is the criterion one used to select the variables.



Dendrogram obtained by random assignment of labels, selection of the best discriminating genes and clustering using the selected genes.



Dendrogram with the original labels.

Outline

1 Motivation

2 Clustering

3 Dimensionality reduction

Principles

Principal Component Analysis

Other techniques

4 Conclusions and further readings

Principles (i)

Goal: reduce the dimensionality of the data set to a smaller space (2D, 3D).

X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	X'1	X'2
0.86	-0.48	-0.18	0.37	0.98	-0.97	0.84	-0.06	-0.35	0.56	0.11	-0.21
-2.3	-1.2	-4.5	-0.13	0.02	0.09	-0.71	0.88	0.78	0.7	-2.3	-1.2
0.26	-0.41	0.02	0.33	0.39	-0.46	0.92	0.15	-0.06	-0.26	0.76	-0.46
0.21	-0.13	-0.33	-0.5	0.82	-0.19	0.08	0.48	0.64	-0.38	-0.03	-0.45
0.25	0.11	-0.94	-0.04	0.45	-0.15	-0.85	0.45	0.42	0.29	0.23	-0.02
0.34	0.25	0.83	-0.24	-0.46	0.94	0.12	-0.02	-0.49	0.71	-0.65	-0.91

→ Feature selection: find a subset of the original variables

$$X'_i = X_j \quad \text{for some } j$$

→ Feature extraction: transform the original space into a space of fewer dimensions

$$X'_i = f(X_1, \dots, X_p)$$

→ Linear methods:

$$f(X_1, \dots, X_p) = w_0 + w_1 X_1 + \dots + w_p X_p$$

Principles (ii)

Objectives:

- Reduce the dimensionality of the data set (*pre-processing for other methods*).
- Choose the most useful/informative variables.
- Compress the data.
- Visualize multi-dimensional data in order to:
 - ▶ identify groups of objects
 - ▶ identify outliers

Principles of PCA (i)

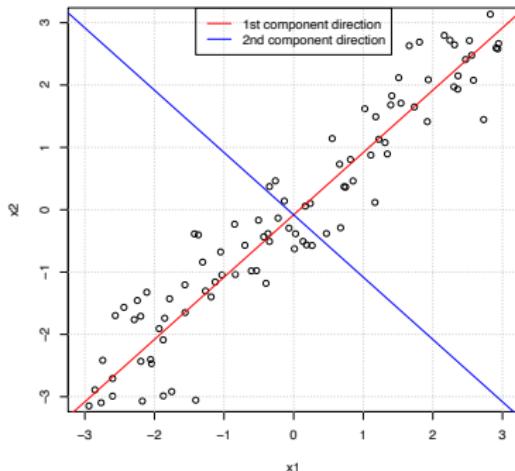
Principal Component Analysis is a *linear feature extraction* technique.

X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	PC1	PC2
0.86	-0.48	-0.18	0.37	0.98	-0.97	0.84	-0.06	-0.35	0.56	0.67	0.48
-2.3	-1.2	-4.5	-0.13	0.02	0.09	-0.71	0.88	0.78	0.7	-2.3	-1.2
0.26	-0.41	0.02	0.33	0.39	-0.46	0.92	0.15	-0.06	-0.26	-0.03	0.67
0.21	-0.13	-0.33	-0.5	0.82	-0.19	0.08	0.48	0.64	-0.38	-0.75	-0.79
0.25	0.11	-0.94	-0.04	0.45	-0.15	-0.85	0.45	0.42	0.29	-0.1	-0.04
0.34	0.25	0.83	-0.24	-0.46	0.94	0.12	-0.02	-0.49	0.71	0.47	0.84

Idea: transform some large number of variables into a smaller number of **uncorrelated** variables called *principal components*.

Principles of PCA (ii)

Goal: map data points into a few dimensions while trying to **preserve** the **variance** of the data as much as possible.



It is particularly efficient when there is a lot of correlation between variables (correlation = redundancy).

PCA - Mathematically

There are two formulations:

- **Maximum variance**: find the directions that maximize the variance of the projected data.
- **Minimum-error formulation**: minimize the reconstruction error of the projected data.

PCA - Maximum variance (i)

Consider a set of observations $\{x_n\}$, $n = 1, \dots, N$ with x_n a vector of dimension D .

We want to find the unit direction u_1 that **maximizes** the **variance** of the projection:

$$\arg \max_{u_1} \frac{1}{N} \sum_{n=1}^N \|u_1^T x_n - u_1^T \bar{x}\|^2 = u_1^T C u_1$$

with

$$\|u_1\| = u_1^T u_1 = 1$$

$$C = \frac{1}{N} \sum_{n=1}^N (x_n - \bar{x})(x_n - \bar{x})^T$$

PCA - Maximum variance (ii)

Introducing the Lagrange multiplier:

$$u_1^T C u_1 + \lambda_1 (1 - u_1^T u_1)$$

Setting the derivative with respect to u_1 to zero:

$$C u_1 = \lambda_1 u_1$$

$\Rightarrow u_1$ must be an eigenvector of C .

The variance is given by:

$$u_1^T C u_1 = \lambda_1$$

$\Rightarrow u_1$ must be the eigenvector corresponding to the highest eigenvalue λ_1 .

PCA - Maximum variance (iii)

The $(M + 1)$ -th component is obtained by maximizing

$$u_{M+1}^T C u_{M+1}$$

with the constraints:

$$u_{M+1}^T u_{M+1} = 1$$

$$u_{M+1}^T u_i = 0 \quad \forall i = 1, \dots, M + 1$$

Using the Lagrangian multiplier:

$$u_{M+1}^T C u_{M+1} + \lambda_{M+1} (1 - u_{M+1}^T u_{M+1}) + \sum_{i=1}^M \eta_i u_{M+1}^T u_i$$

PCA - Maximum variance (iv)

At the **optimum**:

$$0 = 2Cu_{M+1} - 2\lambda_{M+1}u_{M+1} + \sum_{i=1}^M \eta_i u_i$$

Multiplying by u_i^T in the left-hand side, one gets $\eta_i = 0$ and thus

$$Cu_{M+1} = \lambda_{M+1}u_{M+1}$$

$\Rightarrow u_{M+1}^T$ is the eigenvector with the $M + 1$ -th largest eigenvalue.

PCA - Maximum variance (v)

The i -th principal component for an object x_j is computed by
 $x'_{ji} = u_i^T x_j$.

The reconstructed input is thus

$$\hat{x}_j = \sum_{i=1}^M x'_{ji} u_i = \sum_{i=1}^M (u_i^T x_j) u_i$$

PCA also minimizes the reconstruction error:

$$\arg \max_{u_1, \dots, u_M} \frac{1}{N} \sum_{i=1}^N \|x_j - \hat{x}_j\|^2$$

PCA - Algorithm

Algorithm 1

Recover basis: Calculate $XX^\top = \sum_{i=1}^t x_i x_i^\top$ and let U = eigenvectors of XX^\top corresponding to the top d eigenvalues.

Encode training data: $Y = U^\top X$ where Y is a $d \times t$ matrix of encodings of the original data.

Reconstruct training data: $\hat{X} = UY = UU^\top X$.

Encode test example: $y = U^\top x$ where y is a d -dimensional encoding of x .

Reconstruct test example: $\hat{x} = Uy = UU^\top x$.

Table 1.1: *Direct PCA Algorithm*

PCA - Components (i)

Each component is a **linear combination** of the original variables.

X1	X2	X3	X4	X5	X6	X7	X8	X9	X10
0.86	-0.48	-0.18	0.37	0.98	-0.97	0.84	-0.06	-0.35	0.56
-2.3	-1.2	-4.5	-0.13	0.02	0.09	-0.71	0.88	0.78	0.7
0.26	-0.41	0.02	0.33	0.39	-0.46	0.92	0.15	-0.06	-0.26
0.21	-0.13	-0.33	-0.5	0.82	-0.19	0.08	0.48	0.64	-0.38
0.25	0.11	-0.94	-0.04	0.45	-0.15	-0.85	0.45	0.42	0.29
0.34	0.25	0.83	-0.24	-0.46	0.94	0.12	-0.02	-0.49	0.71

PC1	PC2
0.67	0.48
-2.3	-1.2
-0.03	0.67
-0.75	-0.79
-0.1	-0.04
0.47	0.84

Loadings of a component:

$$PC_1 = 0.2X_1 + 3.4X_2 - 4.5X_3$$

$$PC_2 = 0.4X_4 + 5.6X_5 + 2.3X_7$$

...

Loadings give an idea of the importance of a variable in the corresponding component. It can be used for feature selection.

PCA - Components (ii)

For each component, we have a measure of the percentage of **variance** of the initial data that it contains:

$$\text{var}(PC_1) = 4.5 \rightarrow 45\%$$

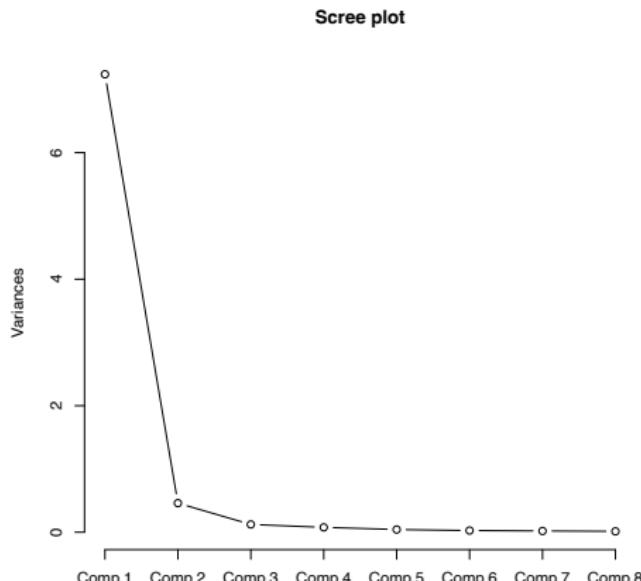
$$\text{var}(PC_2) = 3.3 \rightarrow 33\%$$

...

PCA - How many components? (i)

To determine the number of principal components to consider, there exist several ways:

- Scree plot: plot the eigenvalues (variance) of each component in decreasing order



PCA - How many components? (ii)

- ▶ Rules of thumb:

- Remove the components that have an eigenvalue smaller than 1.
- Select k at the *knee* of the curve, where the scree starts to accumulate.

PCA - Application 1 (i)

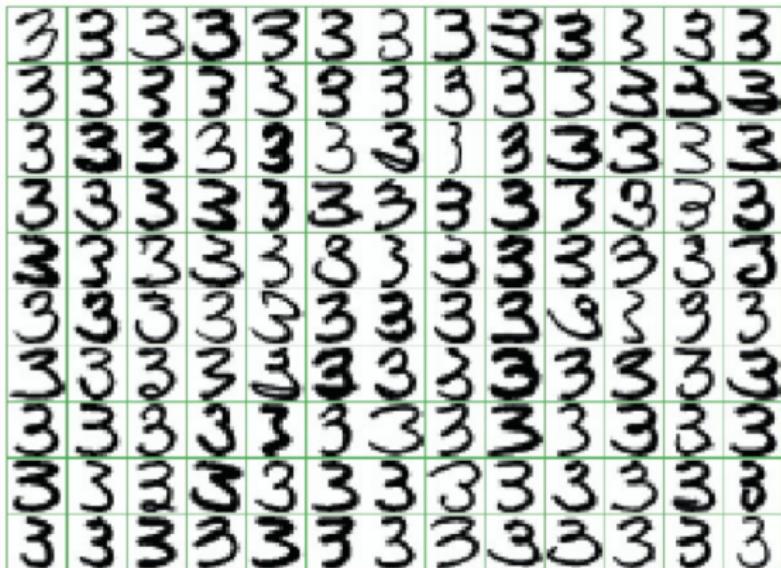


FIGURE 14.22. A sample of 130 handwritten 3's shows a variety of writing styles.

Source: [11]

PCA - Application 1 (ii)

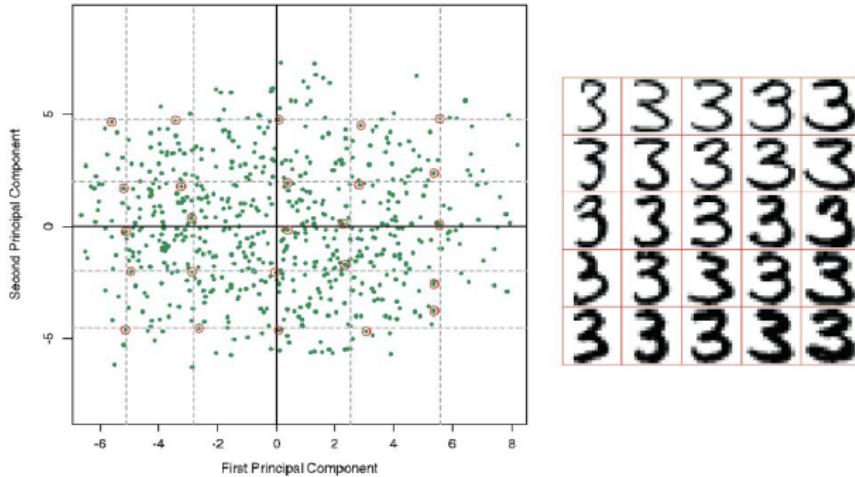
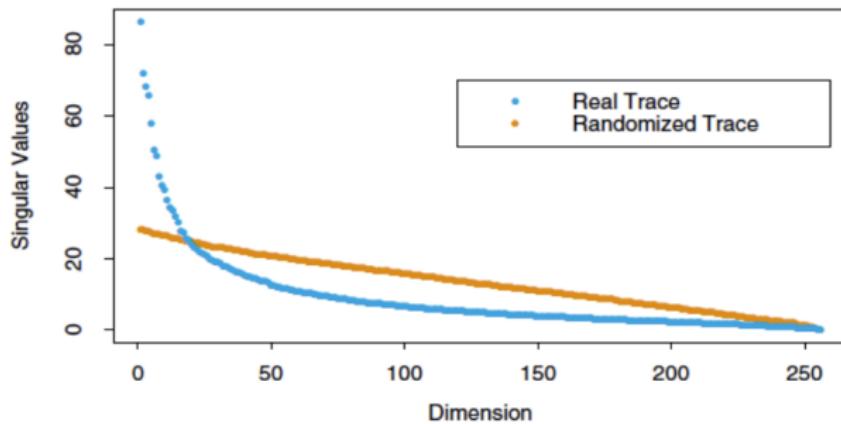


FIGURE 14.23. (Left panel:) the first two principal components of the handwritten threes. The circled points are the closest projected images to the vertices of a grid, defined by the marginal quantiles of the principal components. (Right panel:) The images corresponding to the circled points. These show the nature of the first two principal components.

Source: [11]

PCA - Application 1 (iii)

$$\begin{aligned}\hat{f}(\lambda) &= \bar{x} + \lambda_1 v_1 + \lambda_2 v_2 \\ &= \boxed{3} + \lambda_1 \cdot \boxed{3} + \lambda_2 \cdot \boxed{3}.\end{aligned}$$



Source: [11]

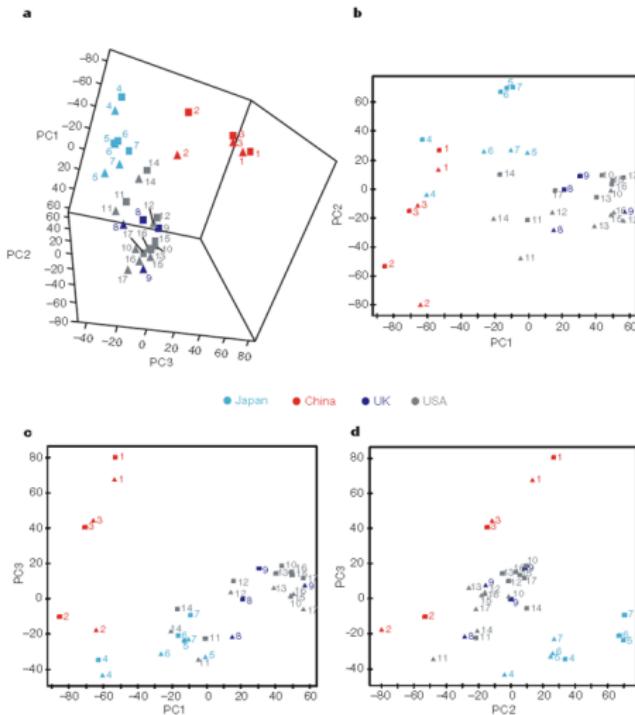
PCA - Application 2 (i)

Investigation of metabolic phenotype variation across and within four human populations: 17 cities from 4 countries (China, Japan, UK, USA) [4].

^1H NMR spectra of urine specimens from 4630 participants.

The authors draw PCA plots of median spectra per population (city) and gender.

PCA - Application 2 (ii)



Source: [4]

PCA - Application 3

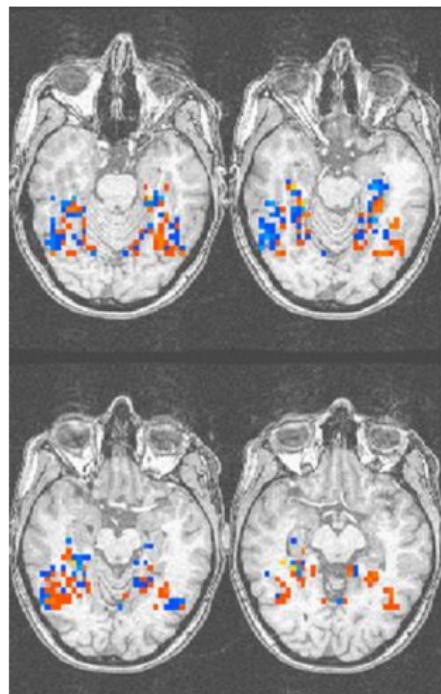
Neuroimaging:

L voxels (brain regions)

→ *N* patients/brain maps

A1 A2 A3 A4 A5 ... A7 A8

-0.91	0.74	0.74	0.97	-0.06	...	-0.04	-0.73
-2.3	-1.2	-4.5	0.47	0.13	...	0.16	0.26
-0.98	-0.46	0.98	0.77	-0.14	...	0.44	-0.12
0.97	-0.64	-0.3	-0.14	-0.29	...	-0.43	0.27
-0.64	-0.34	0.21	-0.57	-0.39	...	0.02	-0.61
0.41	-0.95	0.21	-0.17	-0.68	...	0.11	0.49



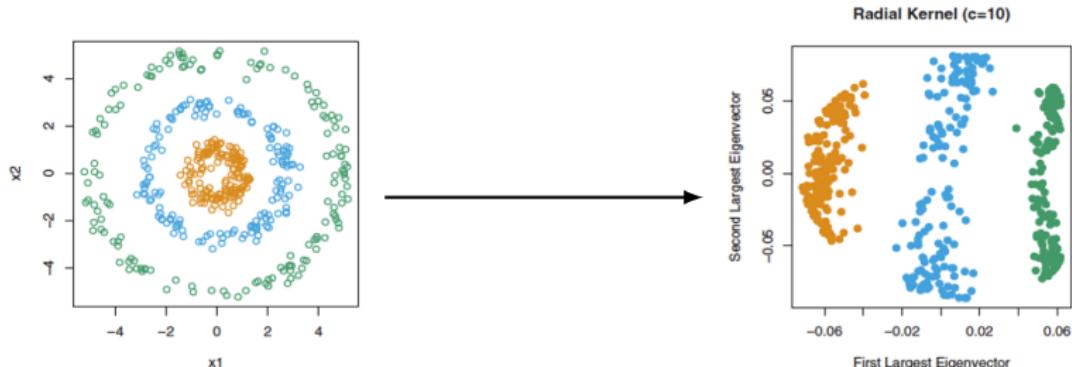
PCA - Limitations

1. PCA may be used to retrieve (visually) a priori determined groups, **however**:
 - If PCA fails at recovering known groups, one cannot conclude anything.
 - PCA is indirect with respect to clustering methods.
2. PCA may be used for feature selection, **however**:
 - First components may not be related at all to the output.
 - Feature selection is better addressed by supervised feature selection techniques.

PCA should only be considered as an **explanatory tool**.

Extensions of PCA

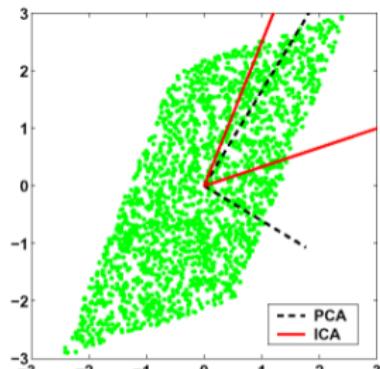
- ▶ Kernel PCA: a non-linear feature extraction technique based on a kernelization of PCA.



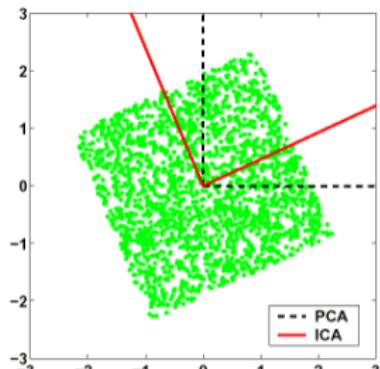
- ▶ Sparse PCA: a method to find components with sparse loadings (few components with non-zero weights) \rightarrow L1 penalization (LASSO).

Independent Component Analysis

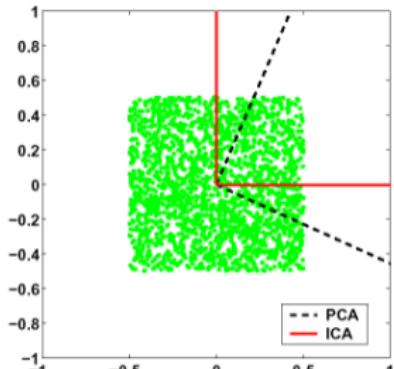
Goal: find independent components, rather than orthogonal components.



(a) Original



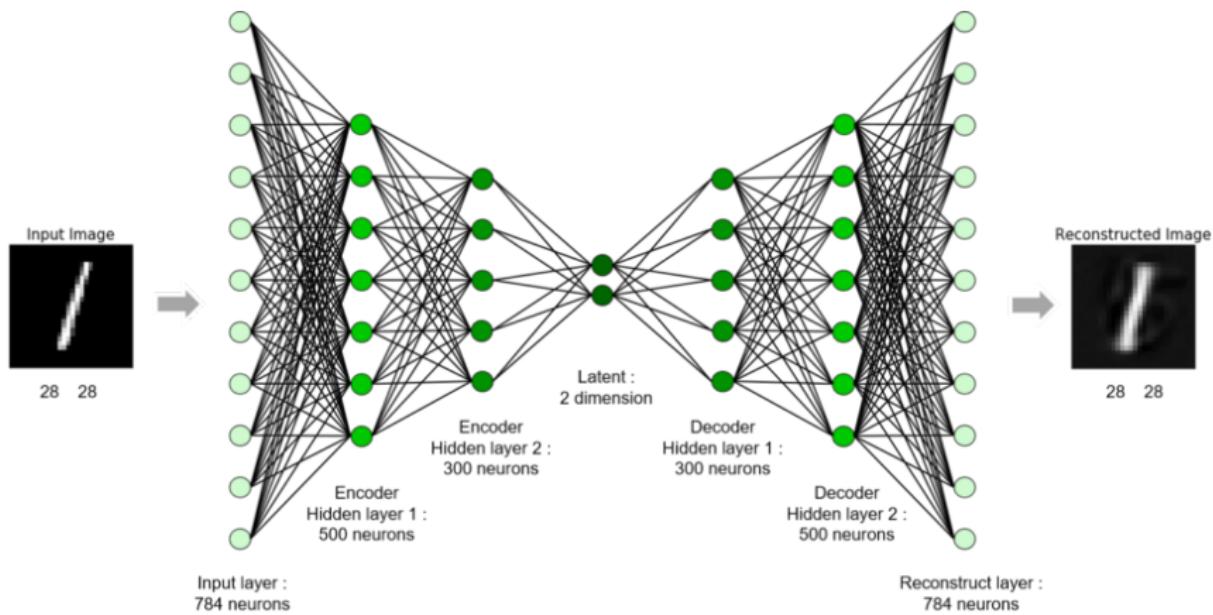
(b) After PCA pre-whitening



(c) After ICA projection

Auto-encoder with neural networks

This method is a **non-linear** embedding.



Source: [12]

Multi-dimensional scaling

Idea: find new coordinates such that some distances are respected (in the least-square sense).

Formally, find $z_1, \dots, z_N \in \mathbb{R}^k$ that minimize

$$S_M(z_1, \dots, z_N) = \sum_{i \neq i'} (d_{ii'} - \|z_i - z_{i'}\|)^2$$

Outline

- ① Motivation
- ② Clustering
- ③ Dimensionality reduction
- ④ Conclusions and further readings

Other unsupervised methods

- ▶ Association rules
- ▶ Density estimation:
 - Mixture models
 - Bayesian networks

Further reading

- ▶ Acknowledgments:
 - Several slides borrowed from Jorg Rahnenfuehrer.
<http://www.statistik.tu-dortmund.de/rahnenfuehrer.html>
- ▶ Further reading:
 - Hastie et al., chapter 14: clustering (14.3), PCA (14.5.1) [11]

References |

-  Pamela Beatriz Guevara Alvez.
Inference of a human brain fiber bundle atlas from high angular resolution diffusion imaging.
PhD thesis, Universite Paris Sud, 2011.
-  Wikipedia contributors.
Hierarchical clustering — Wikipedia, the free encyclopedia, 2020.
[Online; accessed 17-July-2020].
-  Anita Langerød, Hongjuan Zhao, Ørnulf Borgan, Jahn M Nesland, Ida RK Bukholm, Tone Ikdahl, Rolf Kåresen, Anne-Lise Børresen-Dale, and Stefanie S Jeffrey.
Tp53mutation status and gene expression profiles are powerful prognostic markers of breast cancer.
Breast cancer research, 9(3):R30, 2007.
-  Elaine Holmes, Ruey Leng Loo, Jeremiah Stamler, Magda Bictash, Ivan KS Yap, Queenie Chan, Tim Ebbels, Maria De Iorio, Ian J Brown, Kirill A Veselkov, et al.
Human metabolic phenotype diversity and its association with diet and blood pressure.
Nature, 453(7193):396–400, 2008.
-  Michael Assfalg, Ivano Bertini, Donato Colangiuli, Claudio Luchinat, Hartmut Schäfer, Birk Schütz, and Manfred Spraul.
Evidence of different metabolic phenotypes in humans.
Proceedings of the National Academy of Sciences, 105(5):1420–1424, 2008.
-  Mohamed Qasem.
Vector quantization, 2020.
[Online; accessed 17-July-2020].

References II

-  Dimitris Petrilis and Constantin Halatsis.
Combining SOMs and Ontologies for Effective Web Site Mining.
01 2011.
-  Robert Tibshirani, Guenther Walther, and Trevor Hastie.
Estimating the number of clusters in a data set via the gap statistic.
Journal of the Royal Statistical Society: Series B (Statistical Methodology), 63(2):411–423, 2001.
-  Asa Ben-Hur, Andre Elisseeff, and Isabelle Guyon.
A stability based method for discovering structure in clustered data.
In *Biocomputing 2002*, pages 6–17. World Scientific, 2001.
-  Sabina Chiaretti, Xiaochun Li, Robert Gentleman, Antonella Vitale, Marco Vignetti, Franco Mandelli, Jerome Ritz, and Robin Foa.
Gene expression profile of adult t-cell acute lymphocytic leukemia identifies distinct subsets of patients with different response to therapy and survival.
Blood, 103(7):2771–2778, 2004.
-  Trevor Hastie, Robert Tibshirani, and Jerome Friedman.
The elements of statistical learning: data mining, inference, and prediction.
Springer Science & Business Media, 2009.
-  Encode Box.
Autoencoder in biology (review and perspectives), Oct 2019.