K-nn

Alessio Micheli

micheli@di.unipi.it

Dipartimento di Informatica
Università di Pisa - Italy

**Computational Intelligence &
Machine Learning Group**

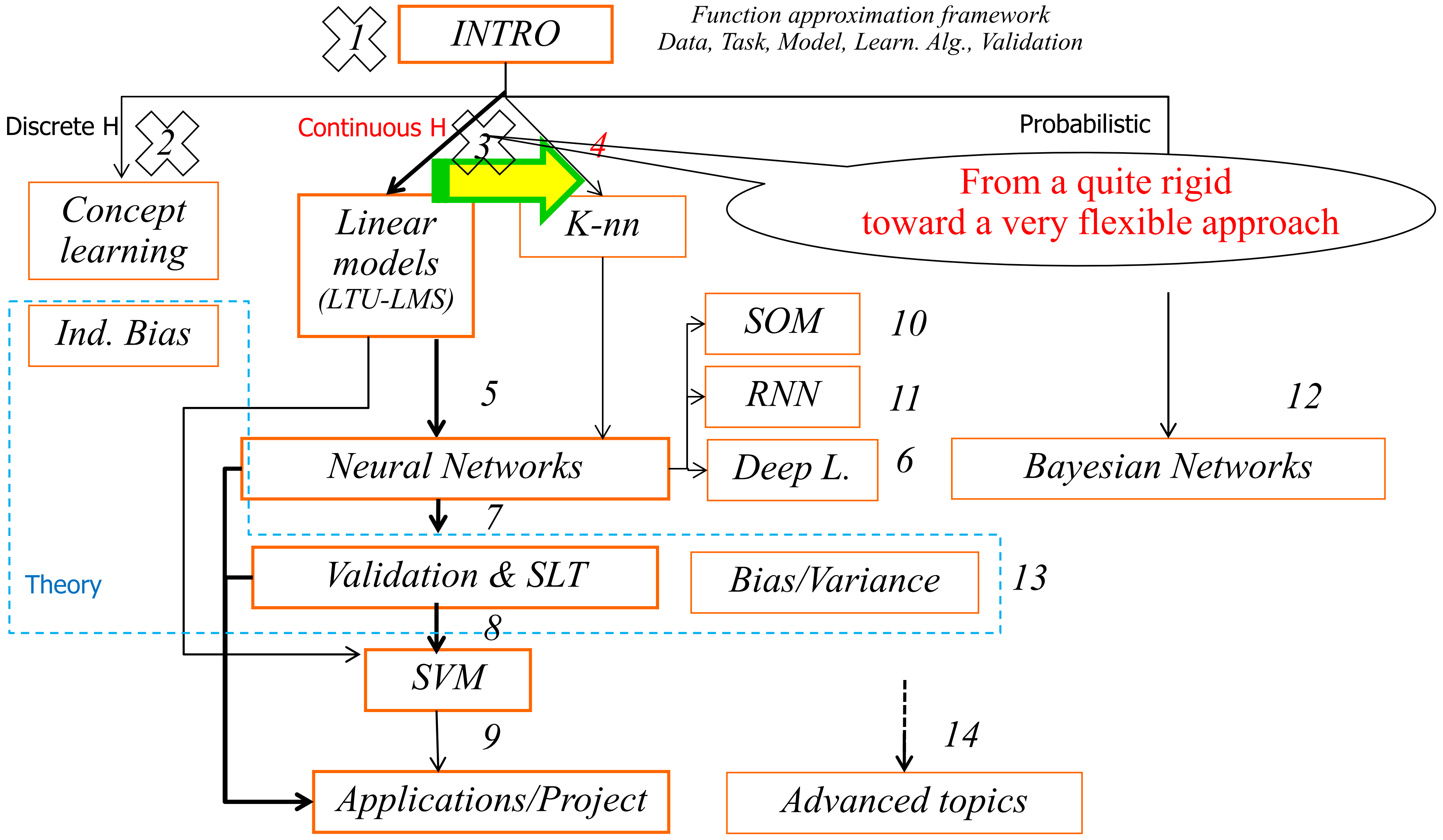
Sept, 2025

K-nn

Still simple but flexible (and local)

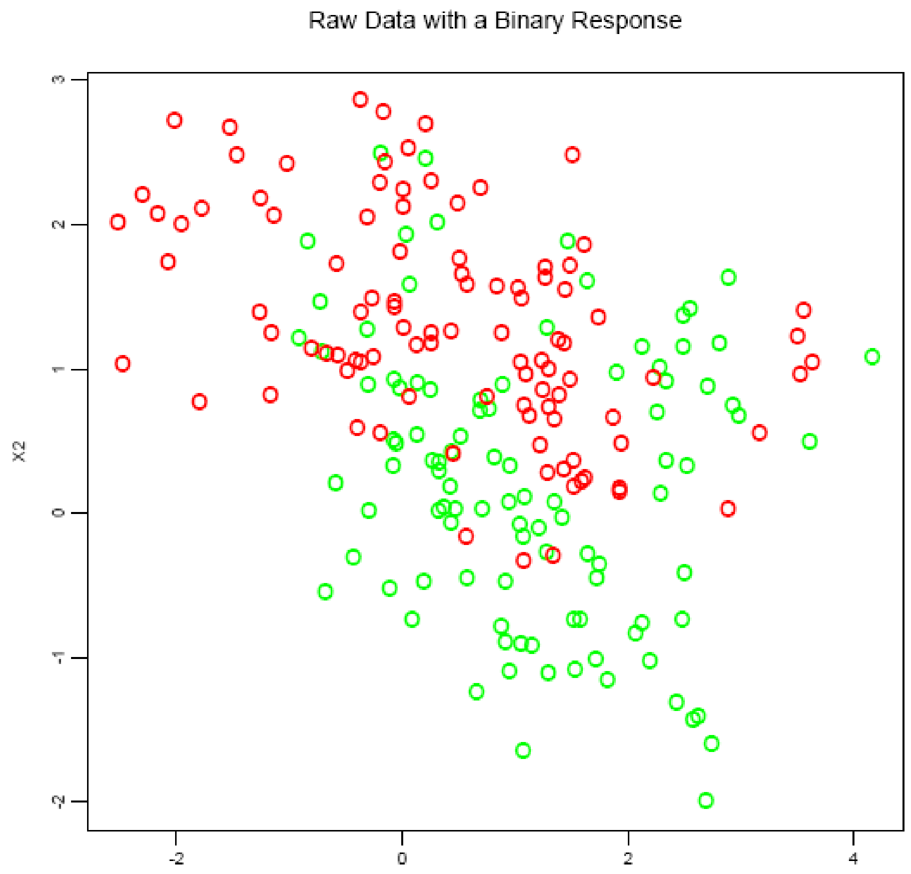
ML Course structure
Where we go

Dip. Informatica
University of Pisa



Repetita: Problem: example

Dip. Informatica
University of Pisa



200 points generated in IR^2 from an unknown distribution; 100 in each of two classes.

Can we build a rule to predict the color of future points?

Data may be generated by gaussian distribution (for each class) with different means

or by a mixture of different low variance gaussian distributions.

Repetita: Find by LS a line to separate

Dip. Informatica
University of Pisa

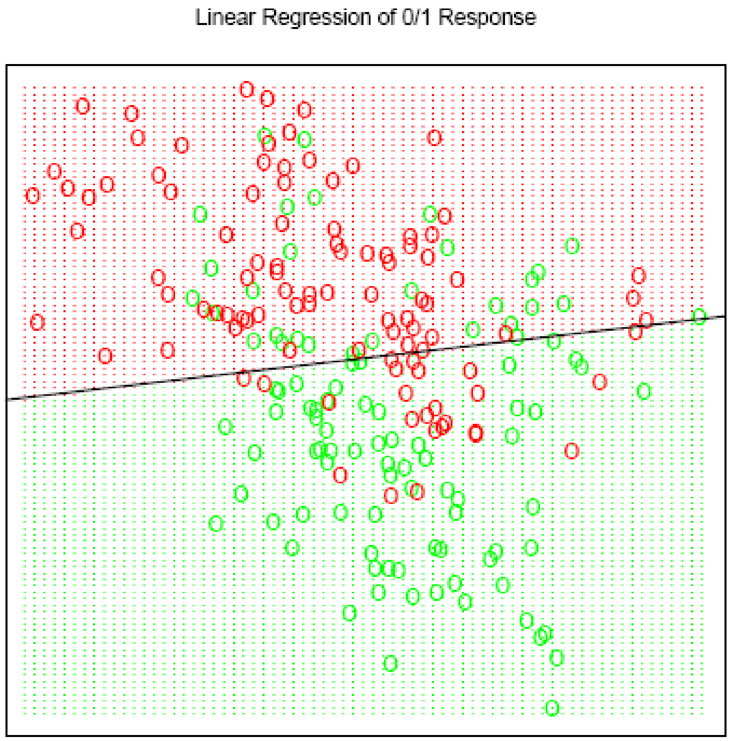


Figure 2.1 (© HTF 2001):

A classification example in two dimensions. The classes are coded as a binary variable GREEN = 0, RED = 1 *and then fit by linear regression. The line is the decision boundary defined by* $x^T w = 0.5$. *The red shaded region denotes that part of input space classified as* RED, *while the green region is classified as* GREEN.

$$\boxed{h(\boldsymbol{x}) = \begin{cases} 1 & \_if \quad \boldsymbol{x}^T \boldsymbol{w} > 0.5 \\ 0 & \_\_\_\_\_\_ otherwise \end{cases}}$$

Linear threshold unit

The decision boundary is $\{\boldsymbol{x} \mid \boldsymbol{x}^T w = 0.5\}$ is linear (and seems to make many errors on the training data). Is it true?

Learning... timing

- LEARNING ALG.

- Timing.
  - **Eager:** Analyze the training data and construct an explicit hypothesis.
  - **Lazy**: Store the training data and wait until a test data point is presented, then construct an ad hoc hypothesis to classify that one data point.

1-Nearest Neighbor (1-nn)

The algorithm:

- Simply store the training data $<\boldsymbol{x}_p, y_p>$ $p=1 \ldots l$
- Given an input $\boldsymbol{x}$ (with dimension n)
- Find the nearest training example $\boldsymbol{x}_i$
  - Find i s.t. we have min $d(\boldsymbol{x},\boldsymbol{x}_i)$ → $i(\boldsymbol{x}) = \arg\min_{p} \, d(\boldsymbol{x}, \boldsymbol{x}_p)$
  - E.g. Euclidian distance :

$$d(\boldsymbol{x}, \boldsymbol{x}_p) = \sqrt{\sum_{t=1}^{n} (x_t - \underbrace{x_{p,t}}_{\text{Pattern } \boldsymbol{x}_p \text{, component } t})^2} = ||\boldsymbol{x} - \boldsymbol{x}_p||$$

- Then output y_i

1-nn

1-Nearest Neighbor Classifier

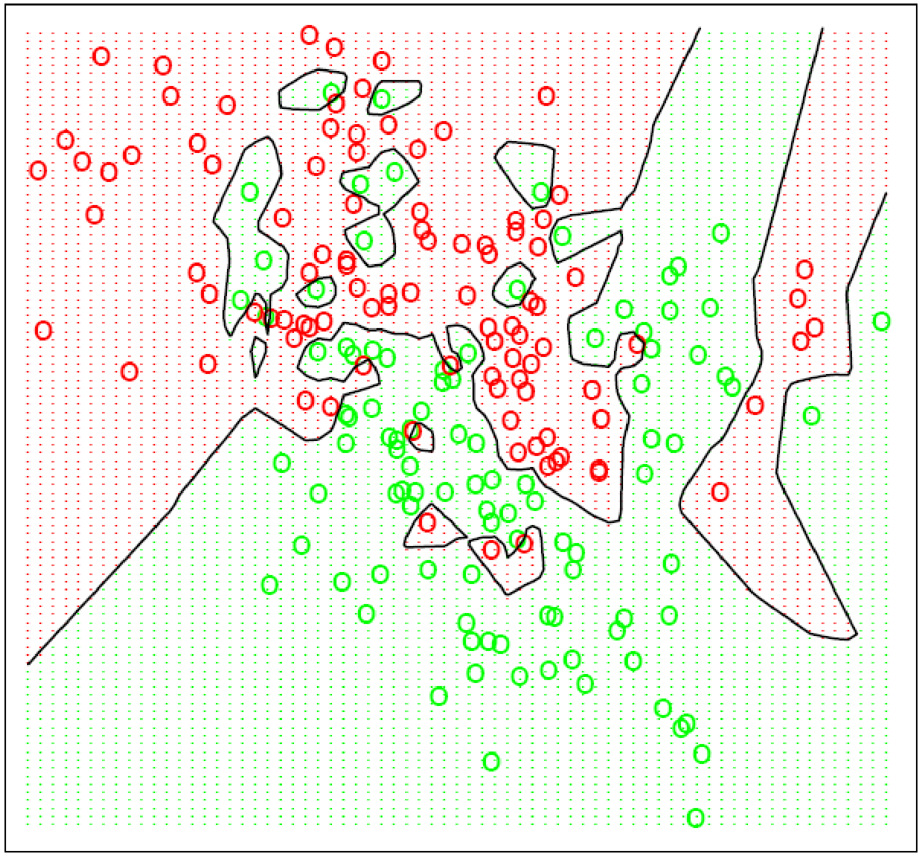
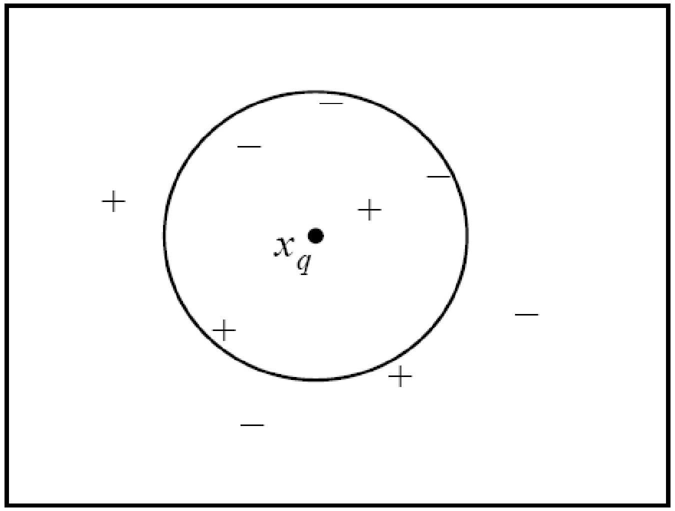


Figure 2.3:
The same classification example in two dimensions as in Figure 2.1.
GREEN = 0, RED = 1

- Very flexible !
- No misclassifications in TR data: 0 training error: what for test data ?
- Decision boundaries is *not* linear: it is quite *irregular*
- May be unnecessary noisy (e.g. for scenario 1)

1-nn vs 5-nn example

- 1-nn return + for x_q
- 5-nn return – for x_q



Smoothing over a set of neighbors for noisy data

K-Nearest Neighbors

- A natural way to classify a new point is to have a look at its neighbors,
- and take an average: $$avg_k(\boldsymbol{x}) = 1/k \sum_{\boldsymbol{x}_i \in N_k(\boldsymbol{x})} y_i$$
- where $N_k(\boldsymbol{x})$ is a neighborhood of $\boldsymbol{x}$ that contains exactly k neighbors (closest patterns according to *distance d*):
  - k-nearest neighborhood: **K-nn.**

- If there is a clear dominance of one of the classes in the neighborhood of an observation ***x***, then it is likely that the observation itself would belong to that class, too. Thus the classification rule is the **majority voting** among the members of $N_k(\boldsymbol{x})$. As before,

$$\boxed{h(\boldsymbol{x}) = \begin{cases} 1 & \_if \quad avg_k(\boldsymbol{x}) > 0.5 \\ 0 & \_\_\_\_\_\_ otherwise \end{cases}} \quad \text{for targets } y \text{ in } \{0,1\}$$

- For regression task: use directly the avg: mean over K-nn

15-nn

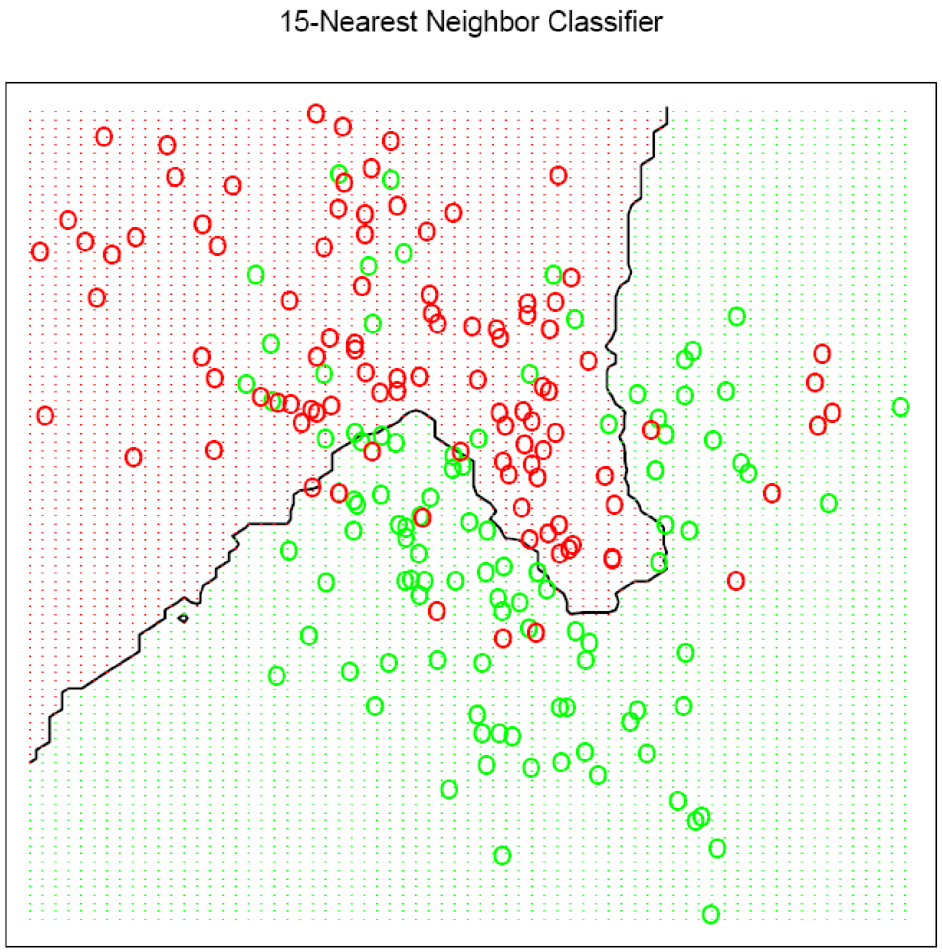
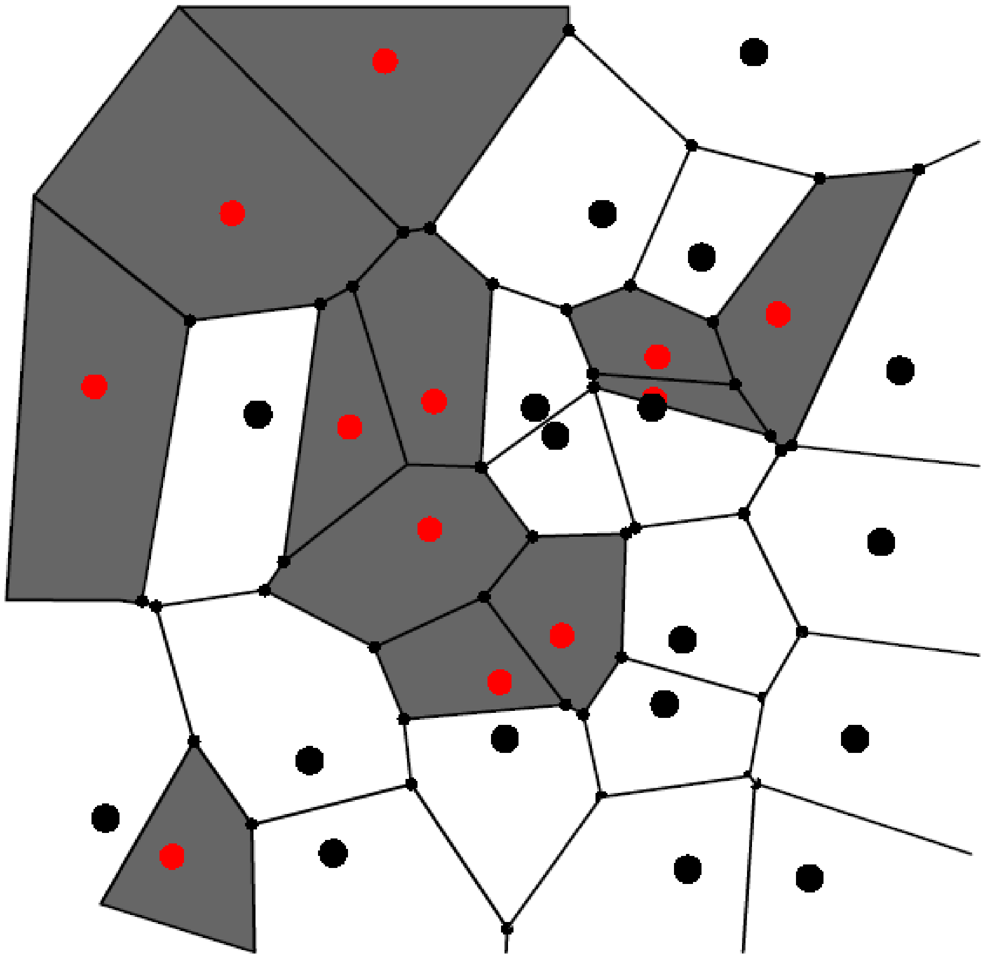


Figure 2.2:
The same classification example in two dimensions as in Figure 2.1.
GREEN = 0, RED = 1
The predicted class is hence chosen by majority vote amongst the 15*-nearest neighbors.*

- Still very flexible !
- Some misclassifications in TR data
- Decision boundaries is *not* linear: it is still quite, although less, *irregular*
- Decision boundary adapts to the local densities of the classes

Voronoi diagram



Each cell consisting of all points closer to ***x*** than to any other patterns.

The segments of the Voronoi diagram are all the points in the plane that are equidistant to two patterns.

Implicitly used by K-nn

K-nn for multi-class

- Return the class most common amongst its *k* nearest neighbors

$$h(\boldsymbol{x}) = \arg\max_{v} \sum_{\boldsymbol{x}_i \in N_k(\boldsymbol{x})} \boldsymbol{1}_{v,y_i}$$

$$\boldsymbol{1}_{v,y_i} = \begin{cases} 1 & if \quad v = y_i \\ 0 & otherwise \end{cases}$$

We count the classes in the neighbor (by the $\boldsymbol{1}_{v,y}$ for each v)
taking the most frequent class ($\arg\max$)

K-nn variants: Weighted distance

- It can be useful to weight the contributions of the neighbors, so that the nearer neighbors contribute more to the average than the more distant ones.

$$h(\boldsymbol{x}) = \arg\max_{v} \sum_{\boldsymbol{x}_i \in N_k(\boldsymbol{x})} \boldsymbol{1}_{v,y_i} \bullet \frac{1}{d(\boldsymbol{x}, \boldsymbol{x}_i)^2}$$

If $d = 0$ for a i , return y_i

$$\boldsymbol{1}_{v,y_i} = \begin{cases} 1 & if \quad v = y_i \\ 0 & otherwise \end{cases}$$

K-nn: An extreme

- Not a global hypothesis for all the instances → no model to be fit
  - We need to memorize the input examples
- Local estimations (by locally constant functions) vs global linear approximation/estimation of the target function (over the instance space)
- Lazy, memory based, instance-based, **distance-based methods**

K-nn versus Linear

Discussion on linear versus k-nn models

Two extremes of the ML panorama:

- Rigid (low variance) versus flexible (high variance):
  - In K-nn with small k few points can change the decision boundary
  - We may pay a price for this flexibility
- Eager versus lazy
- Parametric versus instance-based

- Linear regression uses 3 parameters to describe its fit in the example of Fig. 2.1, w_0, w_1, w_2 (or $n+1$ in general)
  Does K-nn use 1 (the value of k here)?
- More realistically, K-nn uses l /k "effective number of parameters" (Hastie-Tibshirani-Friedman 2001)*
  - where l is used the number of data (*N in the book)*

LS linear vs K-nn with various k values

Figure 2.4: *Misclassification curves for the simulation example used in Fig. 2.1, 2.2 and 2.3.*

Training set of size 200

Test set of size 10.000.

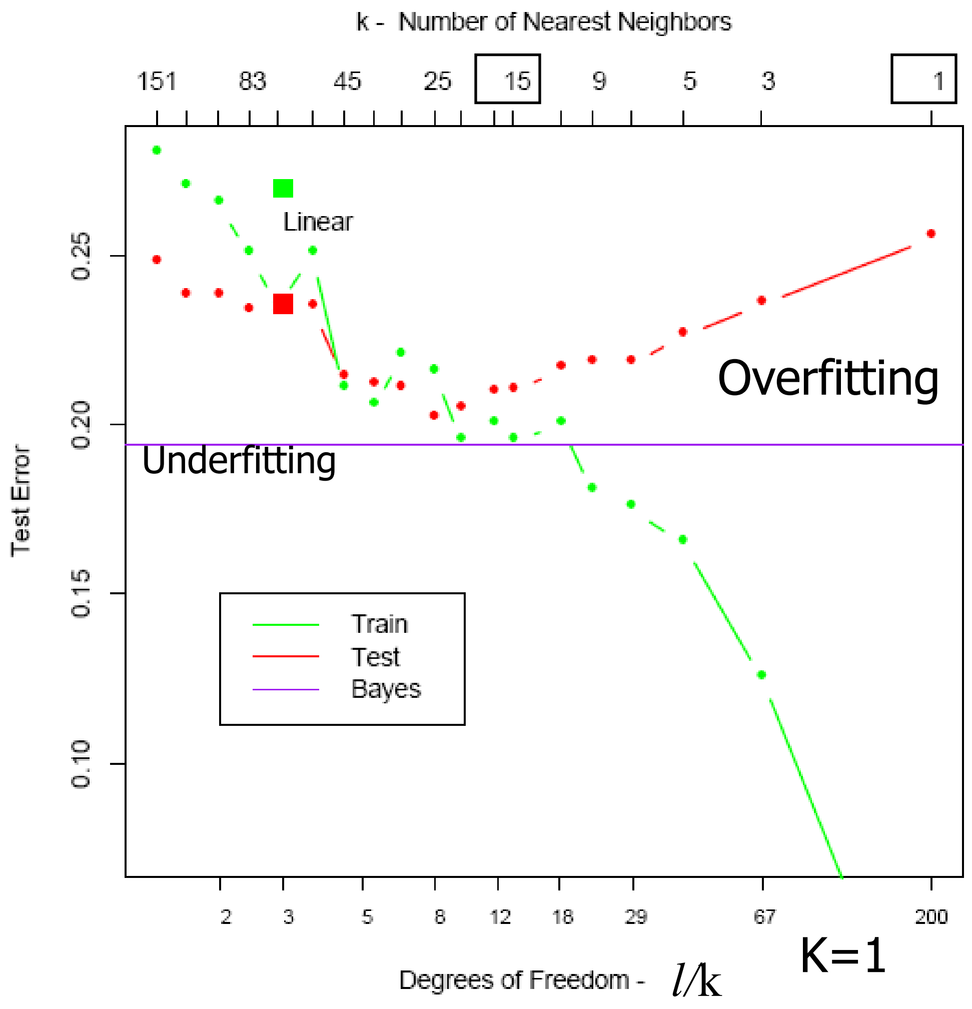
The red curves are test and

the green are training error for k-nn classification (changing K).

The results for linear regression are the bigger green and red dots at three degrees of freedom.

The purple line is the optimal Bayes Error Rate (See next slides)

Note how we move from underfitting to overfitting moving the values of k (i.e. the rate *l* /k): more flexibility allows to find the best result if we control "complexity" by K



Bayes error rate

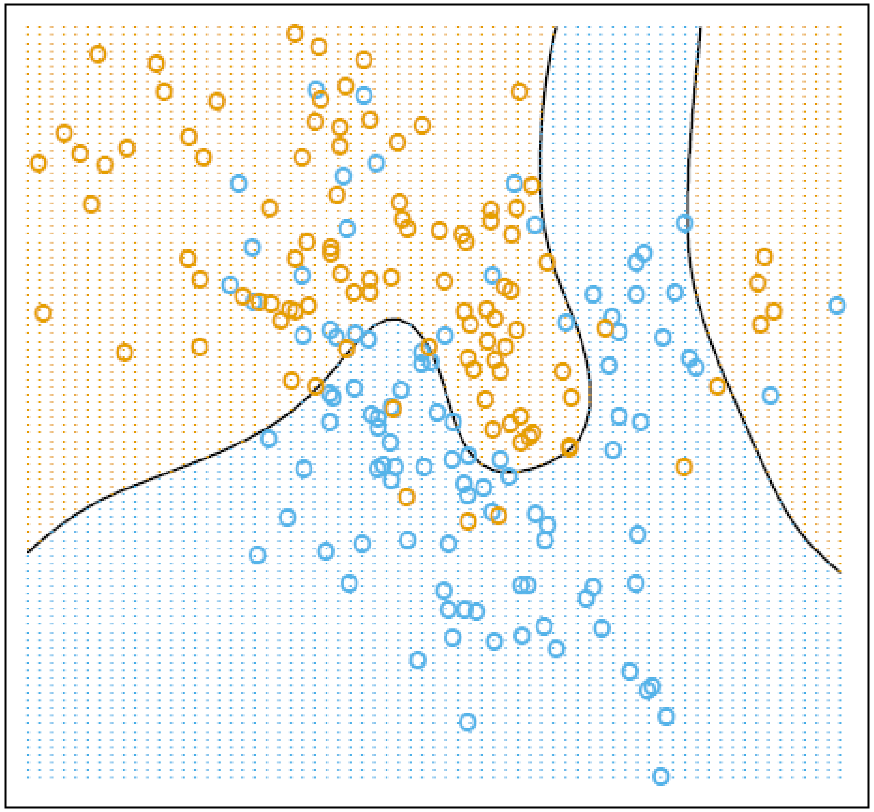
Bayes optimal solution (called Bayes classifier):

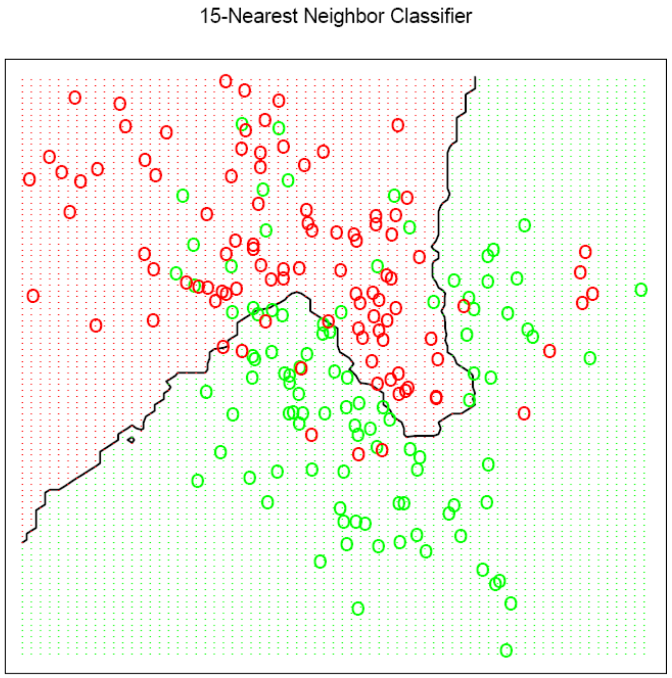
- If we know the density $P(\mathbf{x},y)$ we classify to the most probable class, using the conditional (discrete) distribution as:
  - Output the class v s.t. is max $P(v|\mathbf{x})$ (v *in* $\{C_1, C_2, C_3 \ldots . C_K\}$)

- The error rate of the (optimal) Bayes classifier is called the Bayes rate.
- I.e. the minimum achievable error rate given the distribution of the data (assuming that the generating density is known !!!)

- Note on K-nn: we see that the k-nn classifier directly *approximates this solution* (a majority vote in a nearest neighborhood amounts to exactly this), except that
  - conditional probability at a point is relaxed to conditional probability within a neighborhood of a point (local approximation), and
  - probabilities are estimated by training-sample proportions.

Bayes Optimal Classifier: results





- The optimal Bayes decision boundary for the simulation example.
- Since the generating density is known for each class, this boundary can be calculated exactly
- 15-nn was *close to this* (indeed it shown a min. test err in the plot a couple of slides ago)

Inductive Bias of k-nn

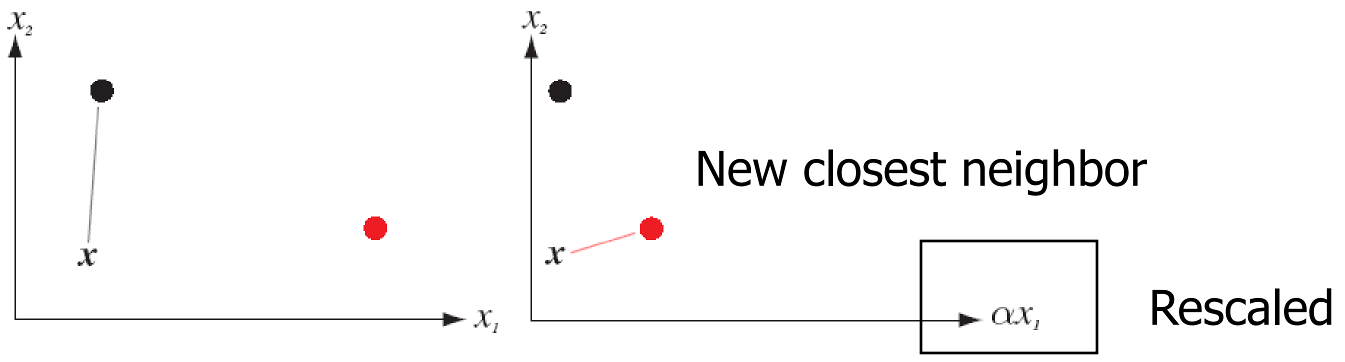
1. The assumed **distance** tells us which are the most similar examples
   - The classification is assumed similar to the classification of the neighbors according to the assumed metric
     - E.g. we can use other metric than the Euclidian
     - Symbolic data require "ad-hoc" metrics (e.g. Hamming distance between two strings of equal length is the number of positions for which the corresponding symbols are different).
2. **Locality**: local approximation (local smoothness prior): see before, and later in the course (vs Deep Learning bias)

- Next slides:
  - criticism and limitations

1. Scale changes and 2. other metrics

- **Scale changes**: Domain knowledge dependent choice
- **If** variables should contribute equally:
  - Pay attention to disparity in the ranges of each variables
  - Rescale data to equalize inputs ranges (→ change the metric)!
    E.g. mean zero and variance 1 normalization

- Variable scaling can have a high impact (i.e. k-nn is fragile even with respect to basic preprocessing)



(some) Limits of K-nn: computational cost

- Note that K-nn makes the local approximation to the target function for each new example to be predicted:

➢ The computational cost is deferred to the **prediction phase**!

Moreover: high retrieval cost:

- Computationally intensive (in time) for each new input: computing the distances from the test sample *to all* stored vectors
  - The time is proportional to the number of stored patterns
  - "ad-hoc" proximity search algorithms to optimize
  - E.g. by indexing the patterns or other approximate nearest neighbor search techniques
- Cost in space (all the training data)

(some) Limits of K-nn

- K-nn models offer little interpretation.
  - Subjectivity of interpretation
  - Dependence on the metric

(some) Limits of K-nn: curse of dim.

K-nn provides a good approximation if we can find a significant set of data close to any ***x***, with dense sampling

When can it fail?

- When we have a lot of input variables (high n, *i.e. high dim.*), K-nn methods often fails because of the ***curse of dimensionality***:

Many manifestations of this problem: we will examine a few

1. *It is hard to find nearby ("similar") points in high dimensions!*
2. *Low sampling density for high-dim data*
3. *Irrelevant features issue*

These are important in terms of ML,
*as they inherently affect the **generalization capability** !!!*

(some) Limits of K-nn: curse of dim.

1. It is hard to find nearby points in high dimensions!

- K-nearest neighbors can fail in high dimensions, because it becomes difficult to gather K observations "close" (<u>in terms of of variables values</u>, i.e. "<u>similar</u>") to a query point x_q:
  - near neighborhoods tend to be **spatially large**, and estimates are **not longer local**
  - reducing the spatial size of the neighborhood means reducing K, and the variance of the estimate increases (→ overfitting).

- Why? See the next slide…

(some) Limits of K-nn: curse of dim.

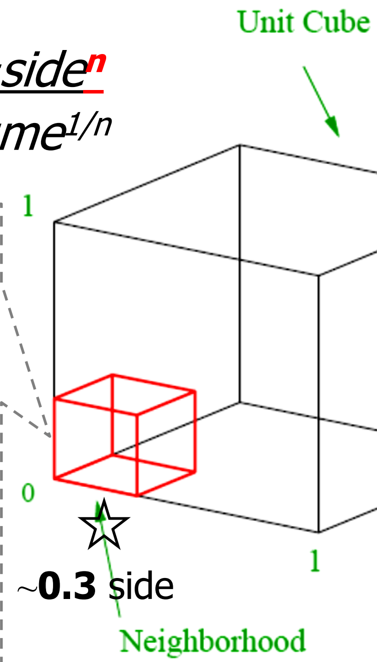
Dip. Informatica
University of Pisa

volume=side^n

side=volume$^{1/n}$

Square **2D**
$(0.3)^2=$
0.09= **9%**
Subcube **3D**
$(0.3)^3=$
0.027=**2.7%**

In **10D**:
$(0.3)^{10}$ =
~ **0.0006%**
of the
volume



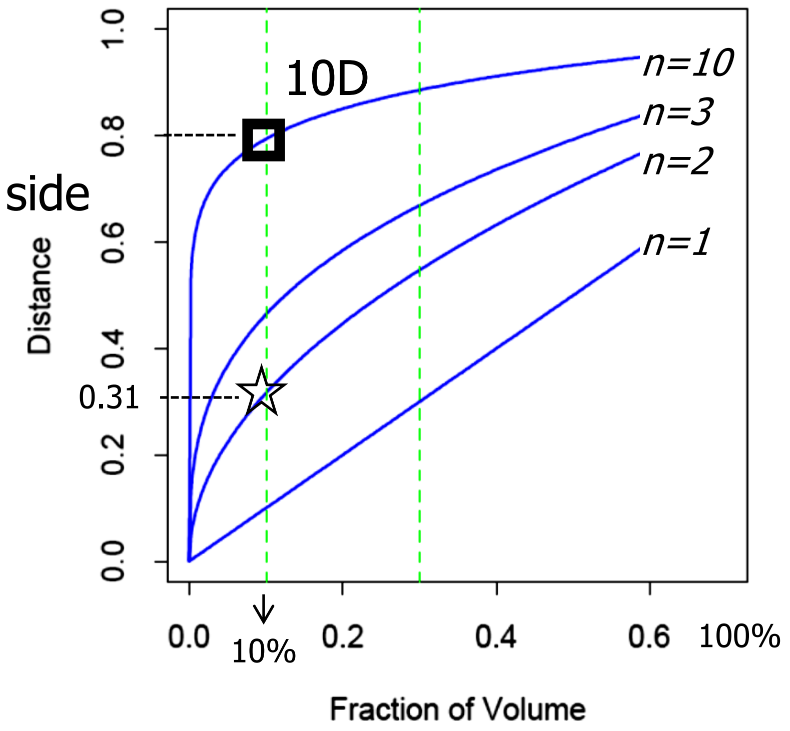


Figure 2.6: *The curse of dimensionality is well illustrated by a subcubical neighborhood for* <u>*uniform*</u> *data in a unit cube.*

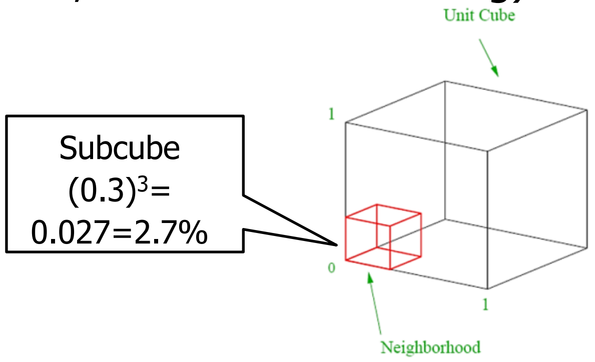
The figure on the right shows the ***sidelength*** ***(=r$^{1/n}$)*** *of the subcube needed to capture a* ***fraction r*** *of the volume of the data, for different dimensions n.*

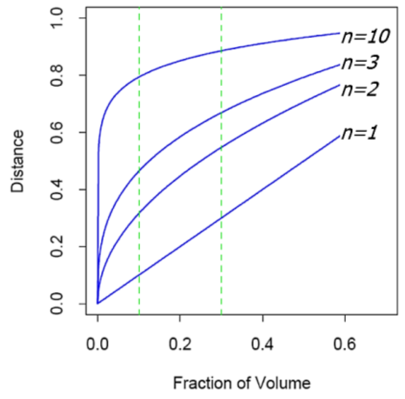
In ten dimensions we need to cover <u>*80%*</u> ◻ *of the range of each coordinate to capture 10% of the data (since 0.1^1/10=~ 0.8), while in 2D* <u>*30%*</u> *(0.316 sidelenght) was suff.* ☆

Further explanation
Loosing of generalization capability

1. Image that to have *k* data you need 10% of the volume. How much ***sidelength*** (features values range) do you need?
   - From 30% to 80% moving from 2D to 10D (**loosing similarity and locality**!!!)
   - And for 1% of the volume in 10D you still need 63% of the input range
2. On the other side:
   - 1D: with 0.3 ***sidelength*** we take ~30% of data volume
   - 2D: with 0.3 ***sidelength*** we take 9% (~10%) of data volume (the red square)
   - 3D: with 0.3 ***sidelength*** we take ~2.7% of data volume (the red cube)
   - 10D: with 0.3 ***sidelength*** we take ~0.0006 % of data volume.

   This sidelength can be not sufficient to get *k* data, unless we use **small K** (which, on the other hand, can lead to overfitting)





(some) Limits of K-nn: curse of dim.

2. Low sampling density for high-dim data

- Sampling density (*l/volume*) is proportional to $l^{1/n}$ (*l* data, *n*=data dim). Hence, for example:
  - if 100 points are sufficient to estimate a function in IR^1 (1-dim input),
  - 100^{10} (=100.000.000.000.000.000.000) are needed to achieve similar accuracy in IR^{10} (10-dim input)

(some) Limits of K-nn: curse of dim.

3. Irrelevant features: The Curse of Noisy

if the target depends on only few of many features in ***x*** (e.g. 2 out 20), we could retrieve a "similar pattern" with the similarity dominated by the large number of irrelevant features

This issue grows with the dimensionality

An improvement

Irrelevant features:

- We may weight features according to their relevance
  - Stretching the axes along some dimension
  - Weights can be searched by a (expensive) model selection approach or other approaches …
- Feature selection approaches: it eliminates some variables $\rightarrow$ reduce input dimension

Summary: K-nn design choices

- The metric d to measure the closeness between patterns (e.g. Euclidian, Hamming, Manhattan distance…, weights on input features).
  Often this is the *key* for a successful application!

- K (number of neighbors: control underfitting/overfitting)

Often necessary to select:

- A subset of data (set of prototypes): e.g. by clustering
- A subset of features

Various approaches to deal with these issues.

Extension in ML

- Extensions to other local models in ML
  - Kernel smoothers
  - Local linear regression
  - Prototype methods
  - Case-based reasoning

K-nn in the course: some general lessons

- Too low variance is poor (rigid linear models), too high variance is dangerous (K-nn)

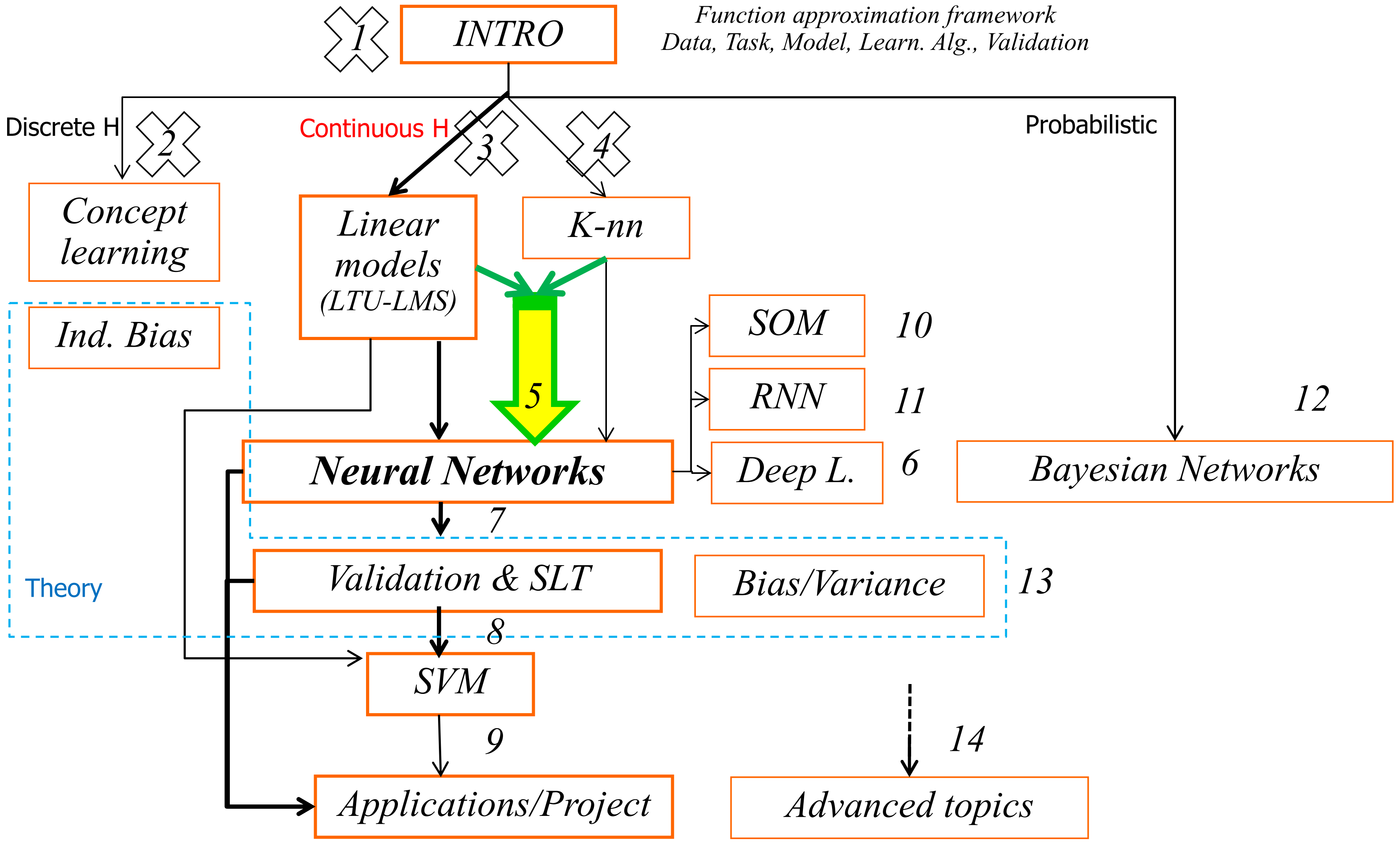
- Other concepts presented todays that are interesting in general:
  - **Smoothing** techniques (in K-nn by increasing K)
  - **Curse of dimensionality** (the volume of the problem space increases so fast that the available data become sparse)
  - Again an **instance of the Statistical Learning Theory** bound plot: see the misclassification plot moving K
  - **Inductive Bias** for K-nn (metric based issue, which is underestimated in many books)

ML Course structure
And now?

- K-nn does not build a “learned model”, not regularity/knowledge extraction/synthesis
  - It does not match the “learning” objective (that can forget the examples after the model is built)
- In the next lectures, we will look at model
  - Compact as the LTU model (all the knowledge in few parameters)
  - More flexible than the linear model (flexible as the K-nn)
    - with a suitable support to the control of the complexity

ML Course structure
Where we go

Dip. Informatica
University of Pisa



Bibliography: Linear and K-nn (I)

- Hastie, Tibshirani, Friedman, The Elements of Statistical Learning, Springer Verlag, 2001-2017 (check the new Ed.): **chap 2** (note: exists an on-line pdf version)
- Mitchell:
  - Linear model and LMS alg.: chap 4.4
  - K-nn: **chap 8**
- Haykin (2nd edition):
  - Linear model and LMS alg.: chap 3 up to 3.7

    [details on Newton and Gauss-Newton methods are not strictly needed]
- Haykin (3rd edition):
  - Linear model and LMS alg.: chap 3

    [details on Newton and Gauss-Newton and other advanced approaches are not strictly needed]

Bibliography: Linear and K-nn (II)

- Moroever…
- Further readings (not mandatory!):
  - Gently introduction to linear models:

    AIMA (Russel, Norvig, Artificial Intelligence: A Modern Approach), ed .3: **chap 18.6** (18.6.1,18.6.2,18.6.3)

  - To go in deep for **Linear least squares**

  You can start from www resource as (With nice examples):

  https://en.wikipedia.org/wiki/Linear_least_squares_%28mathematics%29

  - **LS in general**

  http://en.wikipedia.org/wiki/Least_squares

Suggestion for the next lectures

- Have a look to the slide of the next lectures and
- in particolar to the proof of the Perceptron Convergence Theorem

For information

Alessio Micheli

micheli@di.unipi.it

www.di.unipi.it/groups/ciml

Dipartimento di Informatica
Università di Pisa - Italy

**Computational Intelligence &
Machine Learning Group**