University of Osnabrück, Institute of Cognitive Science

Machine Learning

8 – Local Methods

SS 2018

Gunther Heidemann

Overview

1. Instance-based learning
2. Locally weighted regression
3. Properties of local methods
4. Neural networks based on local representations:
   - Radial basis functions
   - Self-organizing maps

For self-organizing maps see, e.g.,

Helge Ritter, Thomas Martinetz, Klaus Schulten: *Neuronale Netze*, Addison-Wesley

Teuvo Kohonen: *Self-Organizing Maps*, Springer

Instance-based learning

Idea:

Simply store the training examples $D = \{(\vec{x}^n, \vec{t}^n)\}$, $\vec{x}^n \in \mathbb{R}^{d_{in}}$, $\vec{t}^n \in \mathbb{R}^{d_{out}}$.

This leads to the nearest neighbor algorithm:

Training: Memorize all examples.

Application: For an unknown input $\vec{x}$, find the best match $\vec{x}^n$ of the training samples. Output is $\vec{t}^n$.

K-nearest neighbor:

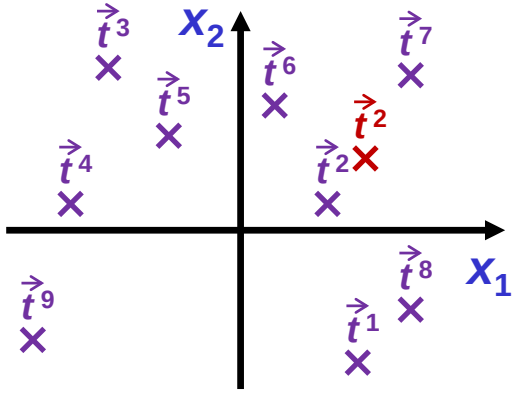
For unknown input $\vec{x}$, find the set S of the k nearest neighbors of stored samples.

- For discrete valued output: Vote among k nearest neighbors.
- For real valued output, use mean of k nearest neighbors:

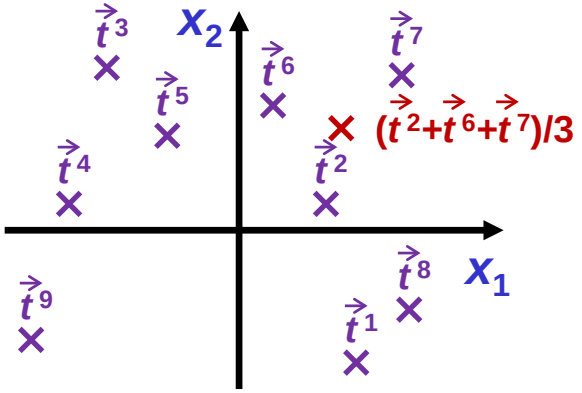
$$\vec{y} = 1/k \sum_{i \in S} \vec{t}^i.$$

Instance-based learning

Nearest neighbor:



3-nearest neighbor:



$\vec{t}^i$ Stored example

$\vec{t}^2$ Unknown input and its classification

[H]

Properties of nearest neighbor approach

- Plain nearest neighbor approach assigns stored output to complete Voronoi tesselation cell around the sample input → hard boundaries.
- *K*-nearest neighbors allow continuous transitions.
- Suitable choice for *k* depends on the local intrinsic data dimensionality.
- "Training"
  - is very fast,
  - requires memory ~ # examples (no compression),
  - does not waste information,
  - does not require parameter settings or complex procedures.
- Application:
  - May be slow (for many stored examples),
  - sensitive to errors and noise.

Distance-weighted *k*-nearest neigbors

Idea:

Nearer neighbors are more important than far ones.

Note choice of k is

- global (same for all samples),
- difficult as intrinsic dimension is unknown and may change locally.

Improve *k*-nearest neighbors by weighting with the distance to the input $\vec{x}$ (S is the set of the indices of the k nearest neighbors):

$$\vec{y} \quad = \quad (1 / \sum_{i \in S} w_i) \ \sum_{i \in S} \ w_i \, \vec{t}^{\,i}$$

with the inverse distance as weight:

$$w_i \quad = \quad 1 / \|\vec{x} - \vec{x}^{\,i}\|$$

Precautions for "direct hit" $\vec{x} = \vec{x}^{\,i}$ are necessary.

Note now the neighbors can be the entire set of examples !

Locally weighted regression

K-nearest neighbors approximate $\vec{y}(\vec{x})$ **locally** for each sample point.

Idea:

Construct better local approximation of $\vec{y}(\vec{x})$ by computing a fit function in the region surrounding the sample points.

Two choices to make:

- Fit function, e.g., linear or quadratic. Linear approach:

$$\vec{y}(\vec{x}) = \vec{w}_0 + \vec{w}_1 \cdot (\vec{x})_1 + \vec{w}_2 \cdot (\vec{x})_2 \ldots + \vec{w}_{d_{in}} \cdot (\vec{x})_{d_{in}}, \qquad \vec{w}_i \in \mathbb{R}^{d_{out}}.$$

- Error function which will be minimized, e.g., by gradient descent to get the best parameters of the fit function (in the linear case $\{\vec{w}_i\}$).

  The error function should be *local*.

Locally weighted regression

Possible error functions:

1. Squared error over only the k nearest neighbors:

$$E_1(\vec{x}^n) = \tfrac{1}{2} \sum_{\vec{x}^i \in \{k \text{ nearest neighbors of } \vec{x}^n\}} (\vec{t}^n - \vec{y}(\vec{x}^i))^2.$$

2. Error over the entire data set D where the error of each training sample $\vec{x}^i$ is weighted by a decreasing function K of its distance to $\vec{x}^n$:

$$E_2(\vec{x}^n) = \tfrac{1}{2} \sum_{\vec{x}^i \in D} K(\|\vec{x}^n - \vec{x}^i\|) \cdot (\vec{t}^n - \vec{y}(\vec{x}^i))^2.$$

3. Combine 1 and 2:

$$E_3(\vec{x}^n) = \tfrac{1}{2} \sum_{\vec{x}^i \in \{k \text{ nearest nbrs. of } \vec{x}^n\}} K(\|\vec{x}^n - \vec{x}^i\|) \cdot (\vec{t}^n - \vec{y}(\vec{x}^i))^2.$$

In general, the error functions can be minimized by gradient descent.
In the linear case, methods exist to compute the coefficients directly.

Local and non-local methods

MLP is not local:

Adaptation of a single weight based on a single example may influence the performance of the entire net (all output channels) and on the complete set of inputs.

In particular, “death” (removal) of a neuron may have major impact, which does not correspond to neurobiology.

Local methods are local with respect to the input space: The output is computed individually for different regions of the input space, so adaptation has only local effects.

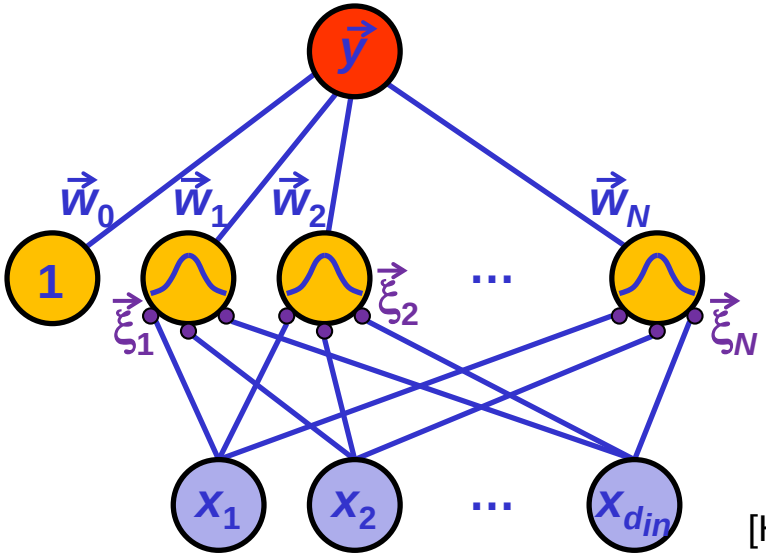
Radial basis functions

- RBFs provide a global approximation of a target function by a linear combination of local approximations.
- Related to
  - distance weighted regression,
  - neural networks.
- Like MLP, RBFs represent a mapping $\vec{x} \rightarrow \vec{y}$, $\vec{x} \in \mathbb{R}^{d_{in}}$, $\vec{y} \in \mathbb{R}^{d_{out}}$.

Radial basis functions

Architecture of RBF network:

- Single layer of units / neurons.
- Each neuron gets the same input.
- Activation of a neuron according to match between input and weights.



[H]

- Activation function is unimodal (usually a Gaussian), not sigmoid!
- Activation function is usually called kernel function, since it defines an “area of responsibility” in the input space.
- Neurons contribute to vector valued output by their weights.
- Highly activated neurons contribute more.
- Thus the output function is represented by local functions with “compact support”.

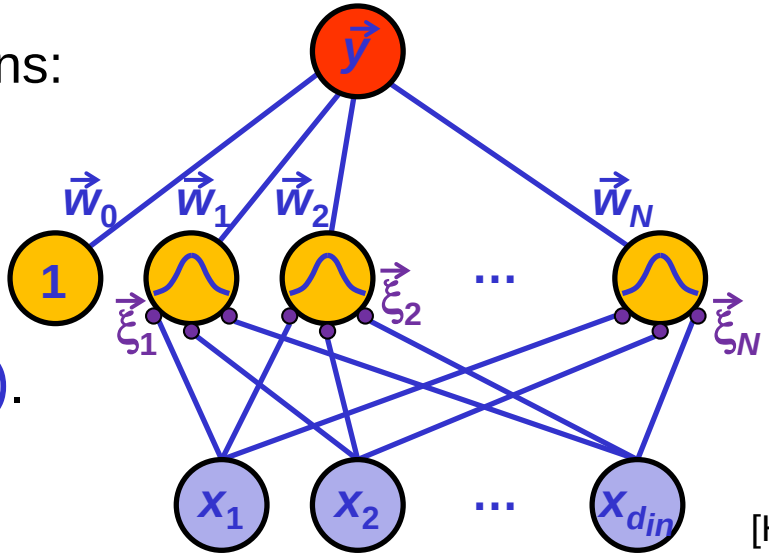
Radial basis functions

Output of RBF network with N neurons:

$$\vec{y} = \vec{w}_0 + \sum_{i=1\ldots N} \vec{w}_i \, K_i\left(\|\vec{x} - \vec{\xi}_i\|\right),$$

with the kernel function

$$K_i\left(\|\vec{x} - \vec{\xi}_i\|\right) = \exp\left(-(1/2\sigma_i^2)\, \|\vec{x} - \vec{\xi}_i\|^2\right).$$



[H]

Training concerns three tasks:

1. Find suitable "centers" or *input weights* $\vec{\xi}_i \in \mathbb{R}^{d_{in}}$.
2. Find suitable "radii of influence" σ_i.
3. Find *output weights* $\vec{w}_i \in \mathbb{R}^{d_{out}}$ to form output.

Several methods exist to solve these tasks.

Radial basis functions

Finding suitable input weights $\vec{\xi}_i$:

- Use examples (instances): $\vec{\xi}_i = \vec{x}^i$, or
- clustering on input part of examples.

Finding the radii:

E.g., define the radius by distance to nearest neighbor, controlled by γ:

$$\sigma_i = \gamma \cdot \min_{k \neq i} |\vec{\xi}_i - \vec{\xi}_k|.$$

Output weights $\vec{w}_i$: Perceptron like rule

$$\Delta \vec{w}_i = \varepsilon (\vec{t} - \vec{y}) K_i(||\vec{x} - \vec{\xi}_i||).$$

Alternative to “bottom up” adaptation in isolated steps: EM on all parameters.

RBF vs. MLP

Compare RBF and MLP:

- Effect of an adaptation step:
  - RBF: Only input component acts locally on one / some basis functions → affects only performance on data in this input area.
  - MLP: Input-output pair may change all weights → may affect performance on all data.
- Both have architectural parameters:
  - RBF: One easy to interpret parameter (# basis functions)
  - MLP: # layers, # neurons in each layer, interpretation difficult.

RBF vs. MLP

- Both have adaptation parameters:
  - RBF:
    - Clustering parameters,
    - radii,
    - stepsize for supervised training.
  - MLP:
    - Stepsize,
    - various others such as momentum.
  - → Parameters of RBFs are decoupled and easy to interpret.
  - → Effect of MLP parameters is difficult to predict as they interact in a complex way during the minimization.

Self-organizing maps

One of the big questions:

Given signal data (low level), how do we get abstract, symbolic representations (high level) ?

Some important aspects of this task:

- Concept formation. As long as we can not get that, at least find reasonable prototypes.
- Filtering relevant from irrelevant information.
- Finding "structure", in particular, relations between concepts / prototypes.

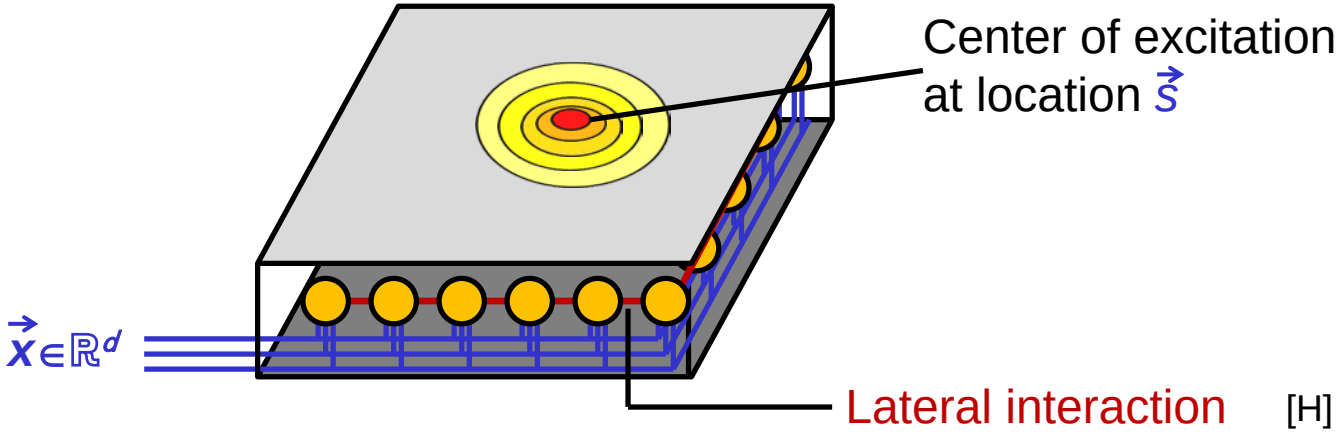
→ A highly useful tool would be a *topology preserving mapping* from signals to a higher level.

Teuvo Kohonen, 1982: Self-organizing map (SOM)

Also: *Kohonen-net.*

Self-organizing maps

2d layer of neurons:



[H]

- For the first time, we consider the spatial (physical) arrangement of neurons in a layer.
- All neurons receive the same input $\vec{x} \in \mathbb{R}^d$.
- Competition:
  - The neuron at location $\vec{s}$ with best matching weights $\vec{w}_{\vec{s}}$ "wins", i.e., has highest excitation $y_{\vec{s}} \in \mathbb{R}$.
  - The best match neuron adapts its weights but lateral interaction causes neighboring neurons to adapt, too.

Self-organizing maps

Notation:

Input: $\vec{x} \in \mathbb{R}^d$.

Weights of neuron at grid location $\vec{r}$: $\vec{w}_{\vec{r}}$.

Excitation of neuron at grid location $\vec{r}$: $y_{\vec{r}}$.

Grid location of maximum excitation: $\vec{s}$,

determined by $\vec{w}_{\vec{s}} \cdot \vec{x} > \vec{w}_{\vec{r}} \cdot \vec{x}$ for all $\vec{r} \neq \vec{s}$.

Excitation over the layer caused by lateral interactions between the excitation center $\vec{s}$ and surrounding locations $\vec{r}$ is modeled by a unimodal function, usually the Gaussian:

$$h_{\vec{r}\,\vec{s}} = \exp\left(-|\vec{r} - \vec{s}|^2 / 2\sigma^2\right).$$

Adaptation rule (Kohonen's rule):

$$\Delta\vec{w}_{\vec{r}} = \varepsilon \cdot h_{\vec{r}\,\vec{s}} \cdot (\vec{x} - \vec{w}_{\vec{r}}).$$

Self-organizing maps

Interpretation of the adaptation rule

$$\Delta\vec{w}_r = \varepsilon \cdot h_{r\,s} \cdot (\vec{x} - \vec{w}_r)$$

as Hebb rule ($\varepsilon \cdot \vec{x}$) with decay term ($-\varepsilon \cdot \vec{w}_r$) to gradually "forget" earlier input and grid-distance weighting $h_{r\,s}$.

- Local (with respect to the layer) adaptation of weights.
- Parameter σ determines size of adaptation region.
- Excitation of a neuron within the region of adaptation is increased by repeating a particular input.
- Neighbors (in the layer) specialize to similar (in the input space) inputs.
- The assignment of input vectors to grid locations enforces development of a topology preserving map.

Self-organizing maps

Adaptation procedure:

1. Get input $\vec{x}$ randomly.
2. Find best match neuron at $\vec{s}$ such that $\vec{w}_s \cdot \vec{x} > \vec{w}_r \cdot \vec{x}$ for all $\vec{r} \neq \vec{s}$.
3. Adpatation:

$$\forall \vec{r}: \quad \Delta \vec{w}_r = \varepsilon \cdot h_{rs} \cdot (\vec{x} - \vec{w}_r).$$

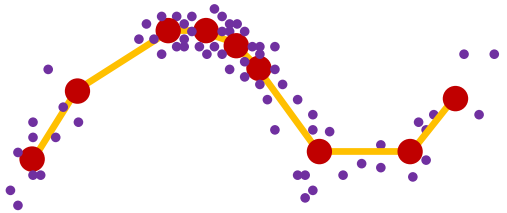
4. Decrease step size ε and size σ of adaptation region, e.g., by

$$\varepsilon(t+1) = \varepsilon(t)\,(1 - \varepsilon^*),$$
$$\sigma(t+1) = \sigma(t)\,(1 - \sigma^*)$$

with ε^*, σ^* small.

Self-organizing maps

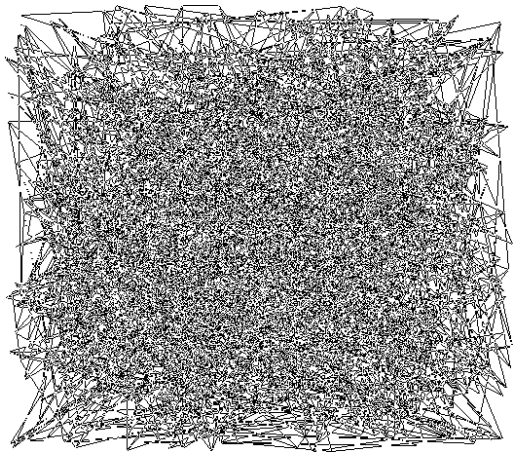
- Maps a space of high dimension to two (or more) dimensions.
- Most applications employ 2d SOM.
- Mapping preserves topology of data in the high dimensional space.
- More nodes in regions of dense data.
- This is achieved by lateral interaction (excitation within short range, inhibition over long range).
- Applications:
  - Principal curve / surface computation for dimension reduction (section 6).
  - Cluster analysis.
  - Visualization of the above.



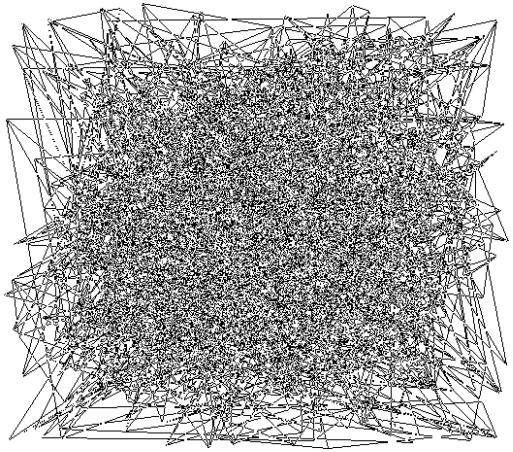
[H]

1d SOM as principal curve, clusters are connected.

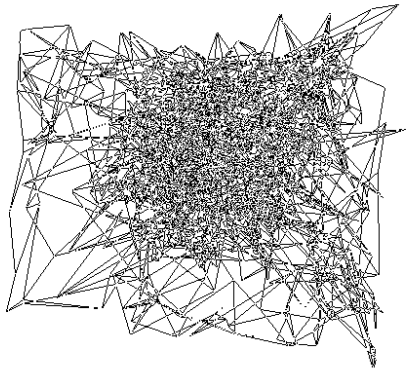
Self-organizing maps



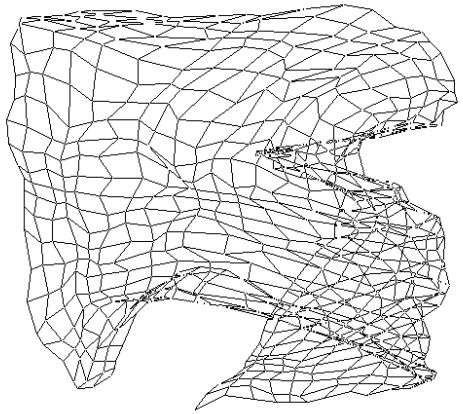
Initialization



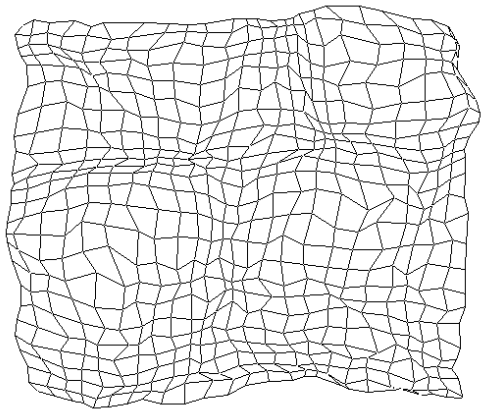
10 steps



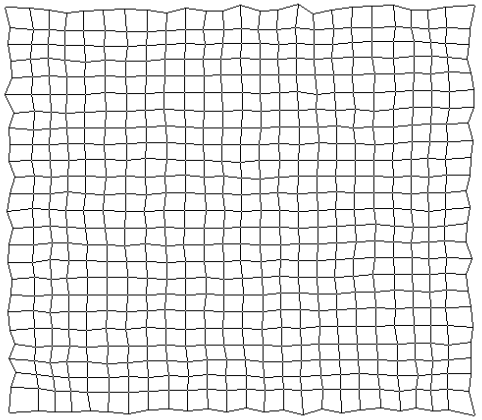
100 steps



1000 steps



10000 steps



100000 steps

Unfolding 2d rectangular SOM grid with random initialization on a 2d square

[de.wikipedia.org/wiki/Selbstorganisierende_Karte, 10.7.2012]

SOM applications in the CV group

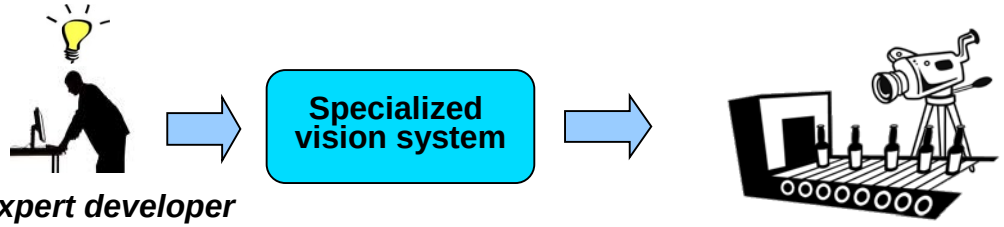
Semi-supervised learning:

- Problem: Performance of most ML applications is limited by the availability of labeled examples!
- Unsupervised learning is infeasible because the desired result can not be coded in the adaptation rules.
- Semi-supervised learning employs
  - unsupervised learning to find structure within data
  - subsequent labeling (supervision) of already found structures.
- Much more efficient as no longer single examples are labeled but clustered examples.
- Applications:
  - Computer Vision: “Vision 2.0”
  - Natural Language Processing: Coreference recognition

Vision 2.0

Today: Vision 1.0

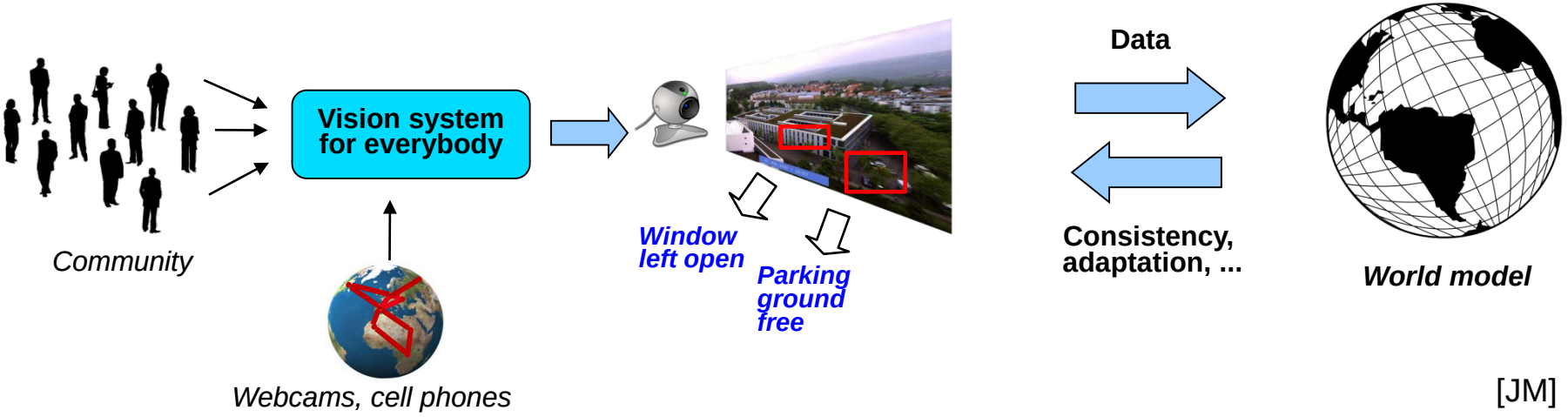
Special systems for special tasks designed by specialists



[JM]

Future: Vision 2.0

A simple system, configured by anybody for arbitrary tasks, e.g., as part of WWW.



[JM]

Vision 2.0

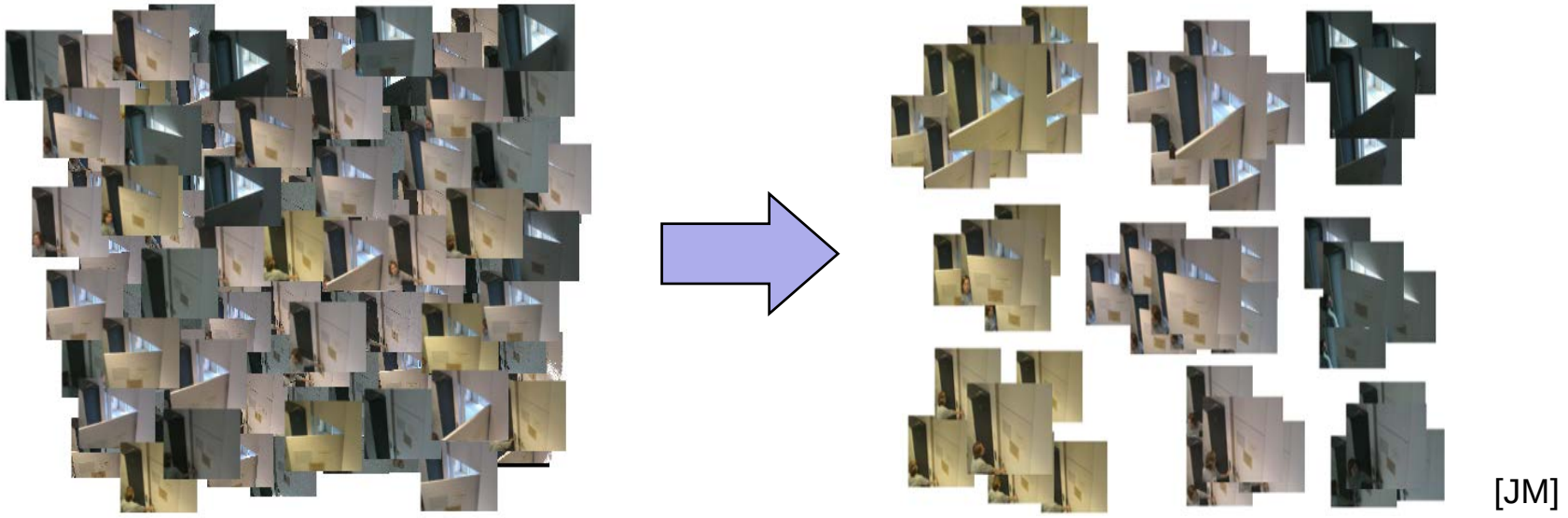
Key components of vision system for everybody:

- Automatic feature extraction
- Automatic classifier adaptation
- Semi-automatic labeling

Idea:

1. Capture loads of unlabeled images from a domain.
2. Extract standard features from all images.
3. Use SOM to find structure in feature space.
4. Visualize projection of images onto SOM.
5. Assign labels to clusters of related images (which may comprise thousands).
6. Train classifier.

Semiautomatic learning



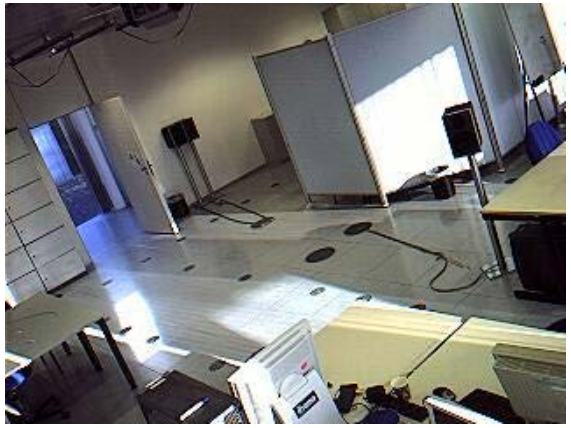
[JM]

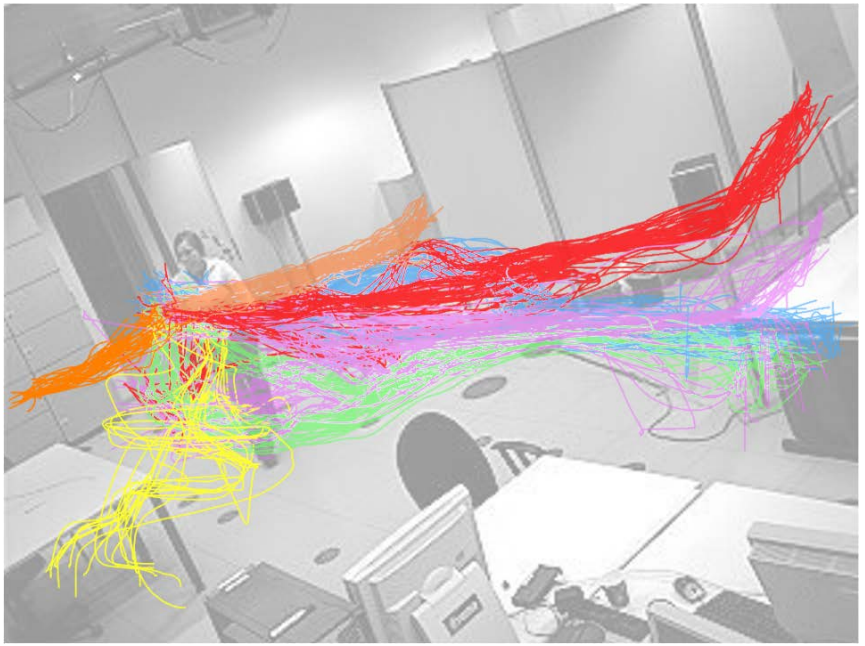
Task:	Classify open/closed office door.
Left:	Numerous images captured at different times of day.
Problem:	Open/closed is not the only variance but also illumination.
Right:	Projection onto SOM reveals clusters which correspond to open/closed doors at various illuminations.
Labeling:	Assign labels, e.g., to open (first row), intermediate (middle), closed (bottom) door.
Note:	Unforeseen classes or a “rejection class” may arise.

Vision 2.0

A similar procedure can be used for labeling trajectories in image sequences:

1. Capture long video sequence.
2. Extract candidate trajectories.
3. Use SOM to find structure in space of trajectory features.
4. Visualize clusters of trajectories.
5. Assign labels to clusters.
6. Train classifier.





Summary

- Local methods
  - are robust during training since a single example affects only part of the system,
  - often have better manageable parameters than global methods.
- Instance based learning is most simple approach.
- Refinement: NN → KNN → Distance weighted KNN → Local linear regression.
- Generalization of local linear regression to RBFs, which can be viewed as neural networks.
- SOMs are rooted in RBFs, basic biological considerations, dimension reduction and clustering.
- SOMs allow topology preserving dimension reduction.
- Important application is (interactive) visualization.

Image sources

[M] Online material available at www.cs.cmu.edu/~tom/mlbook.html
for the textbook: Tom M. Mitchell: *Machine Learning*, McGraw-Hill

[JM] Julia Möhrmann, 2010.

[H] Gunther Heidemann, 2012.