Machine Learning

A. Supervised Learning
A.5. Nearest-Neighbor Models

Lars Schmidt-Thieme, Nicolas Schilling

Information Systems and Machine Learning Lab (ISMLL)
Institute for Computer Science
University of Hildesheim, Germany

Outline

1. Distance Measures
2. K-Nearest Neighbor Models
3. Kernel Regression

Outline

Motivation

So far, regression and classification methods covered in the lecture can be used for

- numerical variables,
- binary variables (re-interpreted as numerical), and
- nominal variables (coded as set of binary indicator variables).

Often one is also interested in more complex variables such as

- set-valued variables,
- sequence-valued variables (e.g., strings),
- ...

Motivation

There are two kinds of approaches to deal with such variables:

feature extraction:

try to derive binary or numerical variables,
then use standard methods on the feature vectors.

kernel methods:

try to establish a distance measure between two variables,
then use methods that use only distances between objects
(but no feature vectors).

Distance measures

Let d be a **distance measure** (also called **metric**) on a set $\mathcal{X}$, i.e.,

$$d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_0^+$$

with

1. d is **positiv definite**: $d(x, y) \geq 0$ and $d(x, y) = 0 \Leftrightarrow x = y$
2. d is **symmetric**: $d(x, y) = d(y, x)$
3. d is **subadditive**: $d(x, z) \leq d(x, y) + d(y, z)$ (triangle inequality)

(for all $x, y, z \in \mathcal{X}$.)

Example: **Euclidean metric** on $\mathcal{X} := \mathbb{R}^n$:

$$d(x, y) := (\sum_{i=1}^{n} (x_i - y_i)^2)^{\frac{1}{2}}$$

Minkowski Metric / L_p metric

$$d(x, y) := (\sum_{i=1}^{n} |x_i - y_i|^p)^{\frac{1}{p}} \qquad p \in \mathbb{R}, p \geq 1$$

$p = 1$ (taxicab distance; Manhattan distance):

$$d(x, y) := \sum_{i=1}^{n} |x_i - y_i|$$

$p = 2$ (Euclidean distance):

$$d(x, y) := (\sum_{i=1}^{n} (x_i - y_i)^2)^{\frac{1}{2}}$$

$p = \infty$ (maximum distance; Chebyshev distance):

$$d(x, y) := \max_{i=1,\ldots,n} |x_i - y_i|$$

Minkowski Metric / L_p metric / Example

Example:

$$x := \begin{pmatrix} 1 \\ 3 \\ 4 \end{pmatrix}, \quad y := \begin{pmatrix} 2 \\ 4 \\ 1 \end{pmatrix}$$

$$d_{L_1}(x, y) = |1-2| + |3-4| + |4-1| = 1 + 1 + 3 = 5$$

$$d_{L_2}(x, y) = \sqrt{(1-2)^2 + (3-4)^2 + (4-1)^2} = \sqrt{1+1+9} = \sqrt{11} \approx 3.32$$

$$d_{L_\infty}(x, y) = \max\{|1-2|, |3-4|, |4-1|\} = \max\{1, 1, 3\} = 3$$

Similarity measures

Instead of a distance measure sometimes **similarity measures** are used, i.e.,

$$\text{sim} : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_0^+$$

with

- sim is symmetric: $\text{sim}(x, y) = \text{sim}(y, x)$.

Some similarity measures have stronger properties:

- sim is **discerning**: $\text{sim}(x, y) \leq 1$ and $\text{sim}(x, y) = 1 \Leftrightarrow x = y$
- $\text{sim}(x, z) \geq \text{sim}(x, y) + \text{sim}(y, z) - 1$.

Some similarity measures have values in $[-1, 1]$ or even $\mathbb{R}$ where negative values denote "dissimilarity".

Distance vs. Similarity measures

A discerning similarity measure can be turned into a semi-metric (pos. def. & symmetric, but not necessarily subadditive) via

$$d(x, y) := 1 - \text{sim}(x, y)$$

In the same way, a metric can be turned into a discerning similarity measure
(with values eventually in $]-\infty, 1]$).

Cosine Similarity

The angle between two vectors in $\mathbb{R}^n$ can be used as distance measure

$$d(x, y) := \text{angle}(x, y) := \arccos\left(\frac{\langle x, y\rangle}{||x||_2 \, ||y||_2}\right)$$

To avoid the arccos, often the cosine of the angle is used as similarity measure (**cosine similarity**):

$$\text{sim}(x, y) := \cos \text{angle}(x, y) := \frac{\langle x, y\rangle}{||x||_2 \, ||y||_2}$$

Example:

$$x := \begin{pmatrix} 1 \\ 3 \\ 4 \end{pmatrix}, \quad y := \begin{pmatrix} 2 \\ 4 \\ 1 \end{pmatrix}$$

$$\text{sim}(x, y) = \frac{1 \cdot 2 + 3 \cdot 4 + 4 \cdot 1}{\sqrt{1 + 9 + 16}\sqrt{4 + 16 + 1}} = \frac{18}{\sqrt{26}\sqrt{21}} \approx 0.77$$

Distances for Nominal Variables

For binary variables there is only one reasonable distance measure:

$$d(x, y) := 1 - I(x = y) \quad \text{with } I(x = y) := \begin{cases} 1 & \text{if } x = y \\ 0 & \text{otherwise} \end{cases}$$

This coincides with the L_∞ distance for the indicator/dummy variables.

The same distance measure is useful for nominal variables with more than two possible values.

For hierarchical variables, i.e., a nominal variable with levels arranged in a hierarchy, there are more advanced distance measures (not covered here).

Distances for Set-valued Variables

For set-valued variables (which values are subsets of a set A) the **Hamming distance** often is used:

$$d(x, y) := |(x \setminus y) \cup (y \setminus x)| = |\{a \in A \,|\, I(a \in x) \neq I(a \in y)\}|$$

(the number of elements contained in only one of the two sets).

Example:

$$d(\{a, e, p, l\}, \{a, b, n\}) = 5, \quad d(\{a, e, p, l\}, \{a, e, g, n, o, r\}) = 6$$

Also often used is the similarity measure **Jaccard coefficient**:

$$\text{sim}(x, y) := \frac{|x \cap y|}{|x \cup y|}$$

Example:

$$\text{sim}(\{a, e, p, l\}, \{a, b, n\}) = \frac{1}{6}, \quad \text{sim}(\{a, e, p, l\}, \{a, e, g, n, o, r\}) = \frac{2}{8}$$

Distances for Strings / Sequences

edit distance / **Levenshtein distance**:

$d(x, y) :=$ minimal number of deletions, insertions or substitutions to transform x in y

Examples:

$$d(\text{man}, \text{men}) = 1$$
$$d(\text{house}, \text{spouse}) = 2$$

$$d(\text{order}, \text{express order}) = 8$$

Distances for Strings / Sequences

The edit distance is computed recursively. With

$$x_{1:i} := (x_{i'})_{i'=1,\dots,i} = (x_1, x_2, \dots, x_i), \quad i \in \mathbb{N}$$

we compute the number of operations to transform $x_{1:i}$ into $y_{1:j}$ as

$$\begin{aligned} c(x_{1:i}, y_{1:j}) := \min\{ & c(x_{1:i-1}, y_{1:j}) + 1, && // \text{ delete } x_i, x_{1:i-1} \rightsquigarrow y_{1:j} \\ & c(x_{1:i}, y_{1:j-1}) + 1, && // \ x_{1:i} \rightsquigarrow y_{1:j-1}, \text{insert } y_j \\ & c(x_{1:i-1}, y_{1:j-1}) + I(x_i \neq y_j)\} && // \ x_{1:i-1} \rightsquigarrow y_{1:j-1}, \text{substitute } y_j \end{aligned}$$

starting from

$$\begin{aligned} c(x_{1:0}, y_{1:j}) = \quad c(\emptyset, y_{1:j}) := \quad j \quad // \text{ insert } y_1, \dots, y_j \\ c(x_{1:i}, y_{1:0}) = \quad c(x_{1:i}, \emptyset) := \quad i \quad // \text{ delete } x_1, \dots, x_i \end{aligned}$$

Such a recursive computing scheme is called **dynamic programming**.

Distances for Strings / Sequences

Example: compute $d(\text{excused}, \text{exhausted})$.

d	9							
e	8							
t	7							
s	6							
u	5							
a	4							
h	3							
x	2							
e	1							
	0	1	2	3	4	5	6	7
$y[j]/x[i]$		e	x	c	u	s	e	d

Distances for Strings / Sequences

Example: compute $d(\text{excused}, \text{exhausted})$.

d	9	8	7	7	6	5	4	3
e	8	7	6	6	5	4	3	4
t	7	6	5	5	4	3	3	4
s	6	5	4	4	3	2	3	4
u	5	4	3	3	2	3	4	5
a	4	3	2	2	2	3	4	5
h	3	2	1	1	2	3	4	5
x	2	1	0	1	2	3	4	5
e	1	0	1	2	3	4	5	6
	0	1	2	3	4	5	6	7
$y[j]/x[i]$		e	x	c	u	s	e	d

Distances for Strings / Sequences

Example: compute $d(\text{excused}, \text{exhausted})$.

d	9	8	7	7	6	5	4	3
e	8	7	6	6	5	4	3	4
t	7	6	5	5	4	3	3	4
s	6	5	4	4	3	2	3	4
u	5	4	3	3	2	3	4	5
a	4	3	2	2	2	3	4	5
h	3	2	1	1	2	3	4	5
x	2	1	0	1	2	3	4	5
e	1	0	1	2	3	4	5	6
	0	1	2	3	4	5	6	7
$y[j]/x[i]$		e	x	c	u	s	e	d

Outline

Neighborhoods

Let d be a distance measure.
For a dataset

$$D \subseteq X \times Y$$

and $x \in \mathcal{X}$ let

$$D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$$

be an enumeration with increasing distance to x, i.e., $d(x, x_i) \leq d(x, x_{i+1})$ (ties broken arbitrarily).
The first $K \in \mathbb{N}$ points of such an enumeration, i.e.,

$$C_K(x) := \{(x_1, y_1), (x_2, y_2), \ldots (x_K, y_K)\}$$

are called a **K-neighborhood** of x (in D).

Nearest Neighbor Regression

The **K-nearest neighbor regressor**

$$\hat{y}(x) := \frac{1}{K} \sum_{(x',y') \in C_K(x)} y'$$

The **K-nearest neighbor classifier**

$$\hat{p}(Y = y \mid x) := \frac{1}{K} \sum_{(x',y') \in C_K(x)} I(y = y')$$

and then predict the class with maximal predicted probability

$$\hat{y}(x) := \arg\max_{y \in \mathcal{Y}} \hat{p}(Y = y \mid x)$$

i.e., the majority class w.r.t. the classes of the neighbors.

Nearest Neighbor Regression Algorithm

1: **procedure** PREDICT-KNN-REG($q \in \mathbb{R}^M, \mathcal{D}^{\text{train}} := \{(x_1, y_1), \ldots, (x_N, y_N)\} \in \mathbb{R}^M \times \mathbb{R}, K \in \mathbb{N}, d$)
2: allocate array D of size N
3: **for** $n := 1, \ldots, N$ **do**
4: $D_n := d(q, x_n)$
5: $D = \text{SORT}(D)$
6: $C := \{(x_i, y_i) \in D | i \leq K\}$
7: $\hat{y} := \frac{1}{K} \sum_{k=1}^{K} y_{C_k}$
8: **return** $\hat{y}$

Nearest Neighbor Classification Algorithm

procedure PREDICT-KNN-CLASS($q \in \mathbb{R}^M, \mathcal{D}^{\text{train}} := \{(x_1, y_1), \ldots, (x_N, y_N)\} \in \mathbb{R}^M \times \mathcal{Y}, K \in \mathbb{N}, d$)
  allocate array D of size N
  for $n := 1, \ldots, N$ **do**
    $D_n := d(q, x_n)$
  $D = \text{SORT}(D)$
  $C := \{(x_i, y_i) \in D | i \leq K\}$
  allocate array $\hat{p}$ of size $\mathcal{Y}$
  for $k := 1, \ldots, K$ **do**
    $\hat{p}_{C_k} := \hat{p}_{C_k} + 1$
  for $y \in \mathcal{Y}$ **do**
    $\hat{p}_y := \frac{1}{K} \hat{p}_y$
  return $(\hat{p})_{y \in \mathcal{Y}}$

Decision Boundaries

For 1-nearest neighbor, the predictor space is partitioned in regions of points that are closest to a given data point:

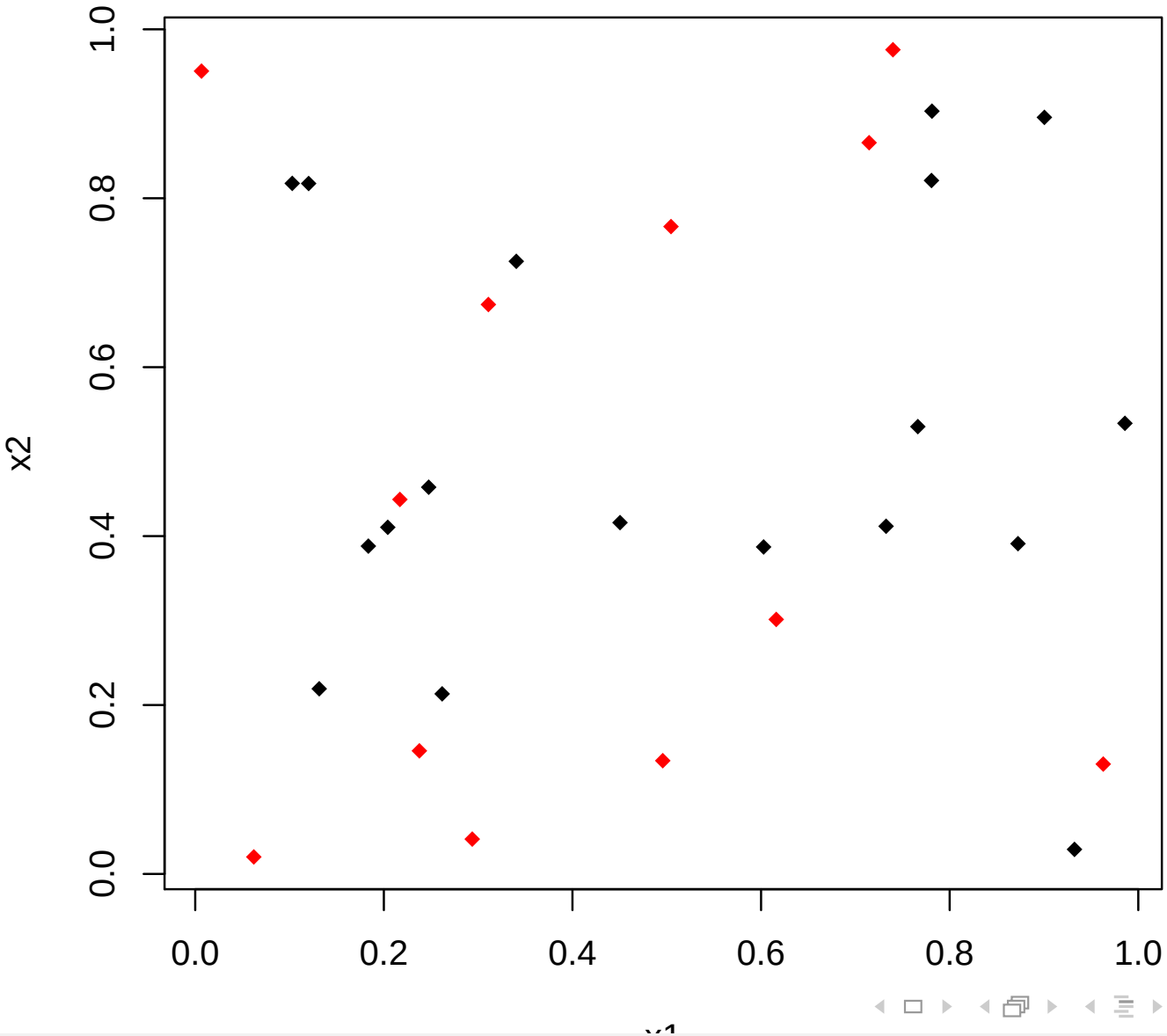
$$\text{region}_D(x_1), \text{region}_D(x_2), \ldots, \text{region}_D(x_N)$$

with

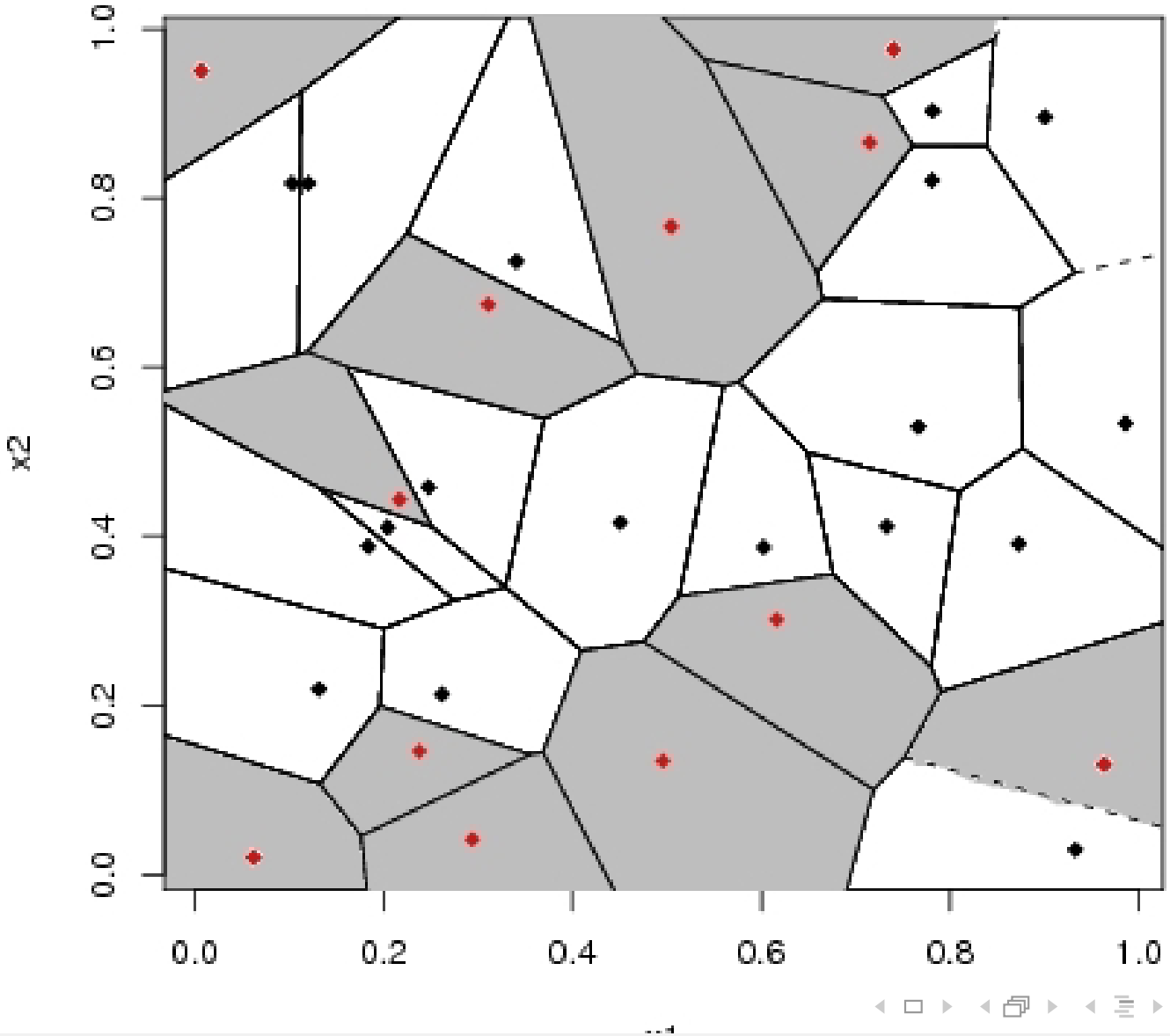
$$\text{region}_D(x) := \{x' \in \mathcal{X} \mid d(x', x) \leq d(x', x'') \quad \forall (x'', y'') \in D\}$$

These regions often are called **cells**, the whole partition a **Voronoi tesselation**.

Decision Boundaries



Decision Boundaries



Complexity of K-Nearest Neighbor Classifier

The K-Nearest Neighbor classifier does not need any learning algorithm as it just stores all the training examples.

On the other hand, predicting using a K-nearest neighbor classifier is slow:

- To predict the class of a new point x, the distance $d(x, x_i)$ from x to each of the N training examples $(x_1, y_1), \ldots, (x_N, y_N)$ has to be computed.
- If the predictor space is $\mathcal{X} := \mathbb{R}^M$, for one such computation we need $O(M)$ operations.
- We then keep track of the K points with the smallest distance.

So in total one needs $O(NM + NK)$ operations.

Partial Distances / Lower Bounding

In practice, nearest neighbor classifiers often can be accelerated by several methods.

Partial distances:

Compute the distance to each training point x' only partially, e.g.,

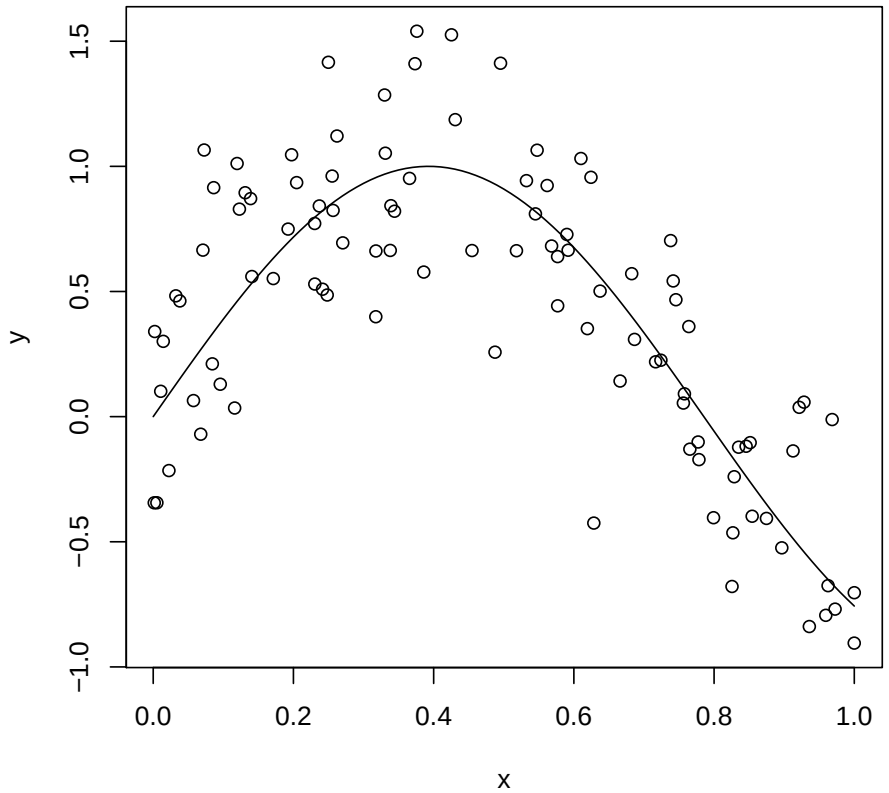
$$d_r(x, x') := \left(\sum_{m=1}^{r} (x_m - x'_m)^2\right)^{\frac{1}{2}}, \quad r \leq M$$

As d_r is non-decreasing in r, once $d_r(x, x')$ exceeds the K-th smallest distance computed so far, the training point x' can be dropped.

This is a heuristic:
it may accelerate computations, but it also may slow it down
(as there are additional comparisons of the partial distances with the K smallest distance).

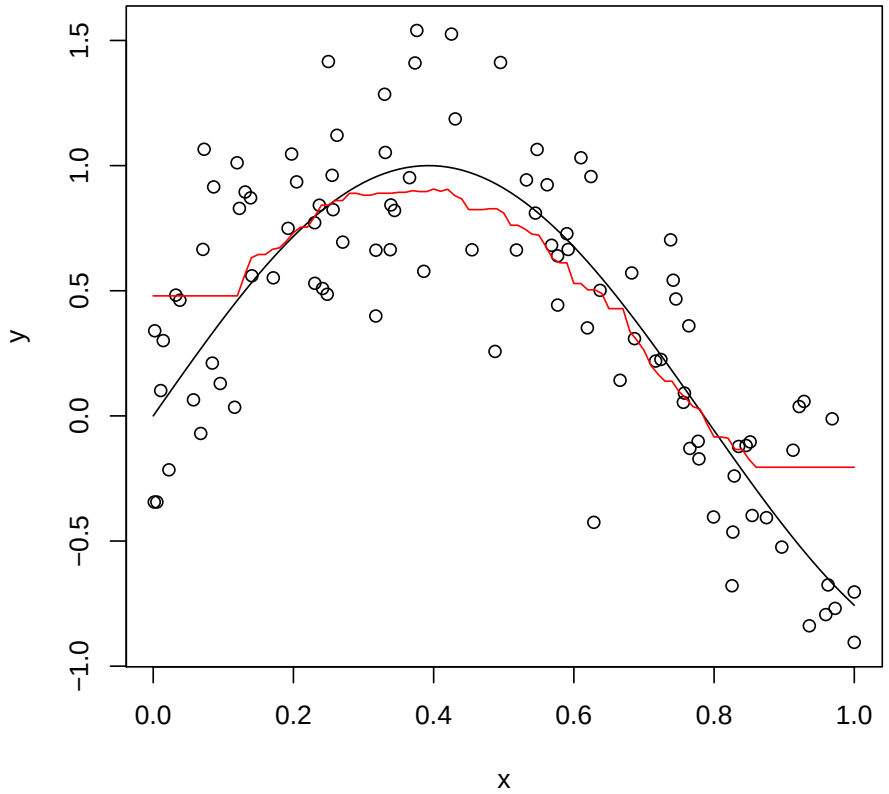
Outline

K-Nearest Neighbor is locally constant



Points generated by the model $y = \sin(4x) + \mathcal{N}(0, 1/3)$ with $x \sim \text{unif}(0, 1)$.

K-Nearest Neighbor is locally constant



K-Nearest Neighbor is locally constant

K-nearest neighbor models are

- based on discrete decisions if a point is a K-nearest neighbor or not,
- in effect, locally constant,
- and thus not continuous.

Formulation using window functions

Discrete decisions can be captured by binary window functions, consider

$$K(x, x_0) := \begin{cases} 1, & \text{if } (x, y) \in N_k(x_0) \\ 0, & \text{otherwise} \end{cases}$$

Then, using this formulation we can rewrite the KNN regressor as:

$$\hat{y}(x_0) = \frac{\sum_{(x,y)\in X} K(x, x_0) y}{\sum_{(x,y)\in X} K(x, x_0)}$$

instead of

$$\hat{y}(x_0) = \frac{\sum_{(x,y)\in N_k(x_0)} y}{k}$$

On the window size

In K-nearest neighbor the size of the window varies from point to point: it depends on the density of the data:

- in dense parts the effective window size is small
- in sparse pars the effective window size is large

Alternatively, it is also possible to set the size of the windows to a constant λ, e.g.,

$$K_\lambda(x, x_0) := \begin{cases} 1, & \text{if } |x - x_0| \leq \lambda \\ 0, & \text{otherwise} \end{cases}$$

Kernel Regression

Instead of discrete windows, one typically uses continuous windows, i.e., continuous weights

$$K(x, x_0)$$

that reflect the distance of a training point x to a prediction point x_0, called **kernel**, e.g.,

$$K(x, x_0) := \begin{cases} 1 - \frac{|x - x_0|}{\lambda}, & \text{if } |x - x_0| \leq \lambda \\ 0, & \text{otherwise} \end{cases}$$

Instead of a binary neighbor/not-neighbor decision, a continuous kernel captures a "degree of neighborship".

Epanechnikov Kernel

Kernels are similarity measures:
the closer two points, the larger the kernel value.

Epanechnikov kernel

$$K_\lambda(x, y) := D\left(\frac{|x - y|}{\lambda}\right)$$

with

$$D(t) := \begin{cases} \frac{3}{4}(1 - t^2), & t < 1 \\ 0, & \text{otherwise} \end{cases}$$

The constant $\lambda \in \mathbb{R}^+$ is called **bandwidth**.

More kernels

Tri-cube kernel

$$D(t) := \begin{cases} (1-t^3)^3, & t < 1 \\ 0, & \text{otherwise} \end{cases}$$

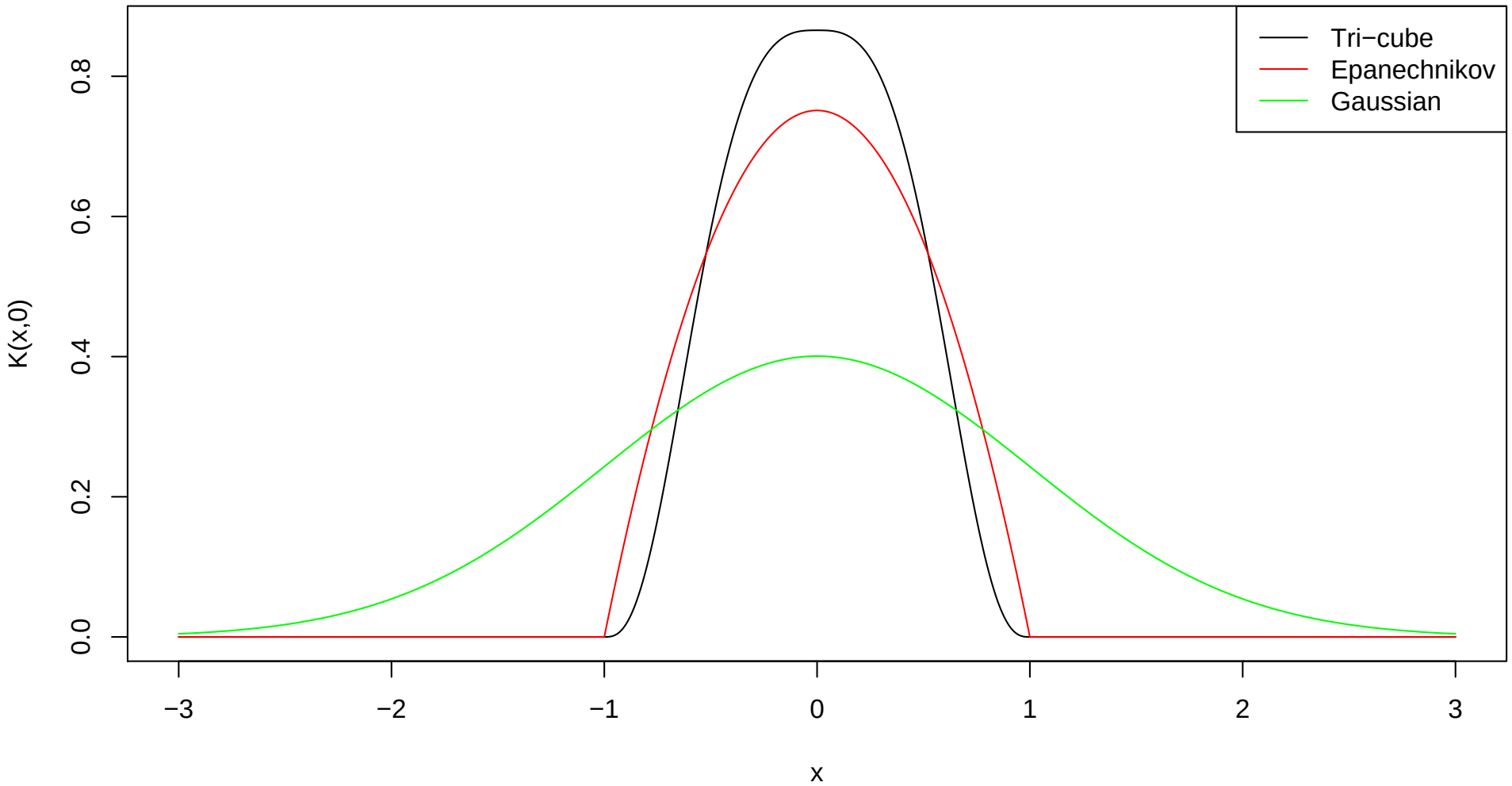
Gaussian kernel

$$D(t) := \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}t^2}$$

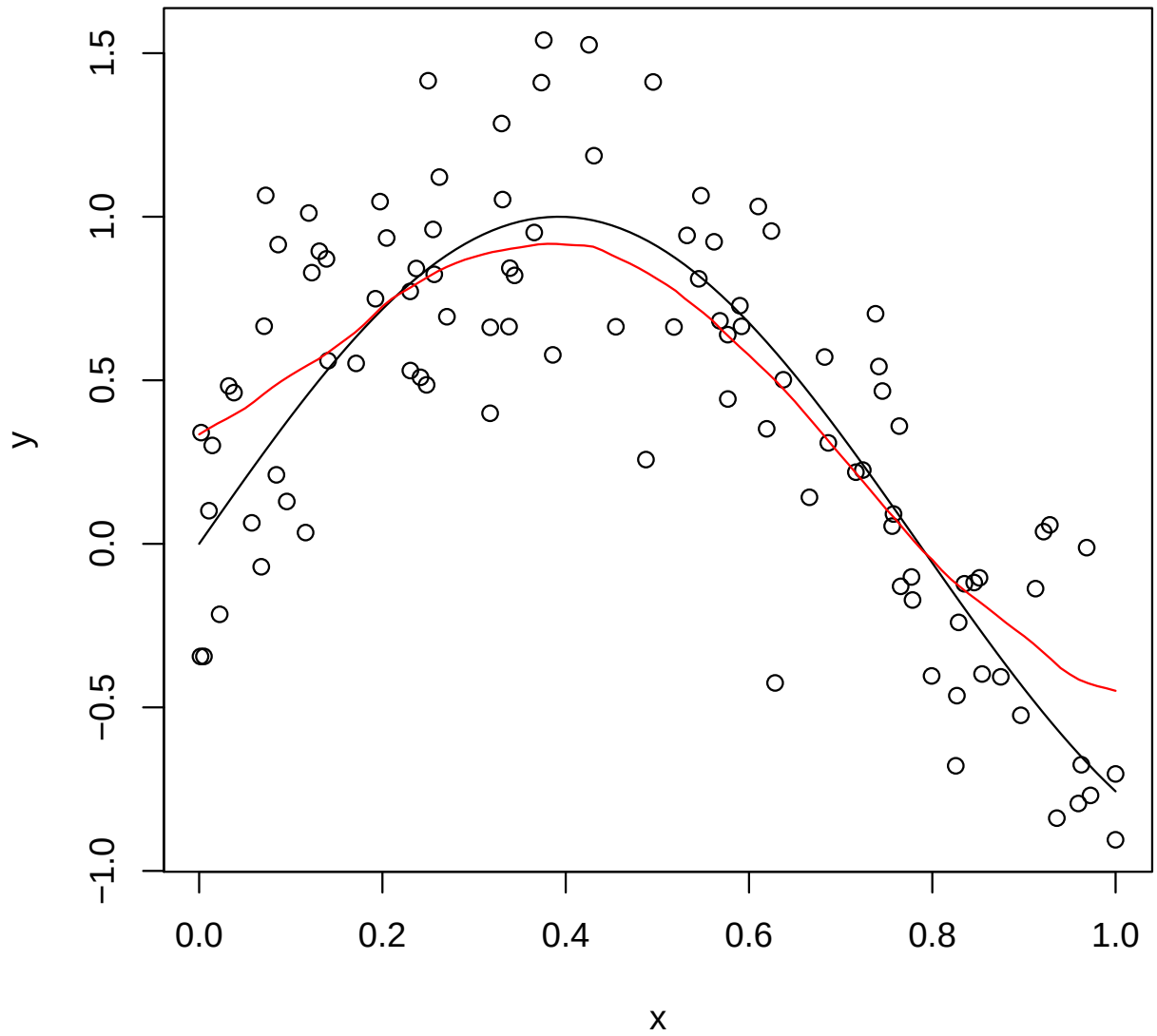
The Epanechnikov and Tri-cube kernel have compact support $[x_0 - \lambda, x_0 + \lambda]$.

The Gaussian kernel has noncompact support, λ acts as standard deviation.

Kernels



Example / Epanechnikov Kernel, $\lambda = 0.2$



Choosing the Bandwidth

If the bandwidth λ is small

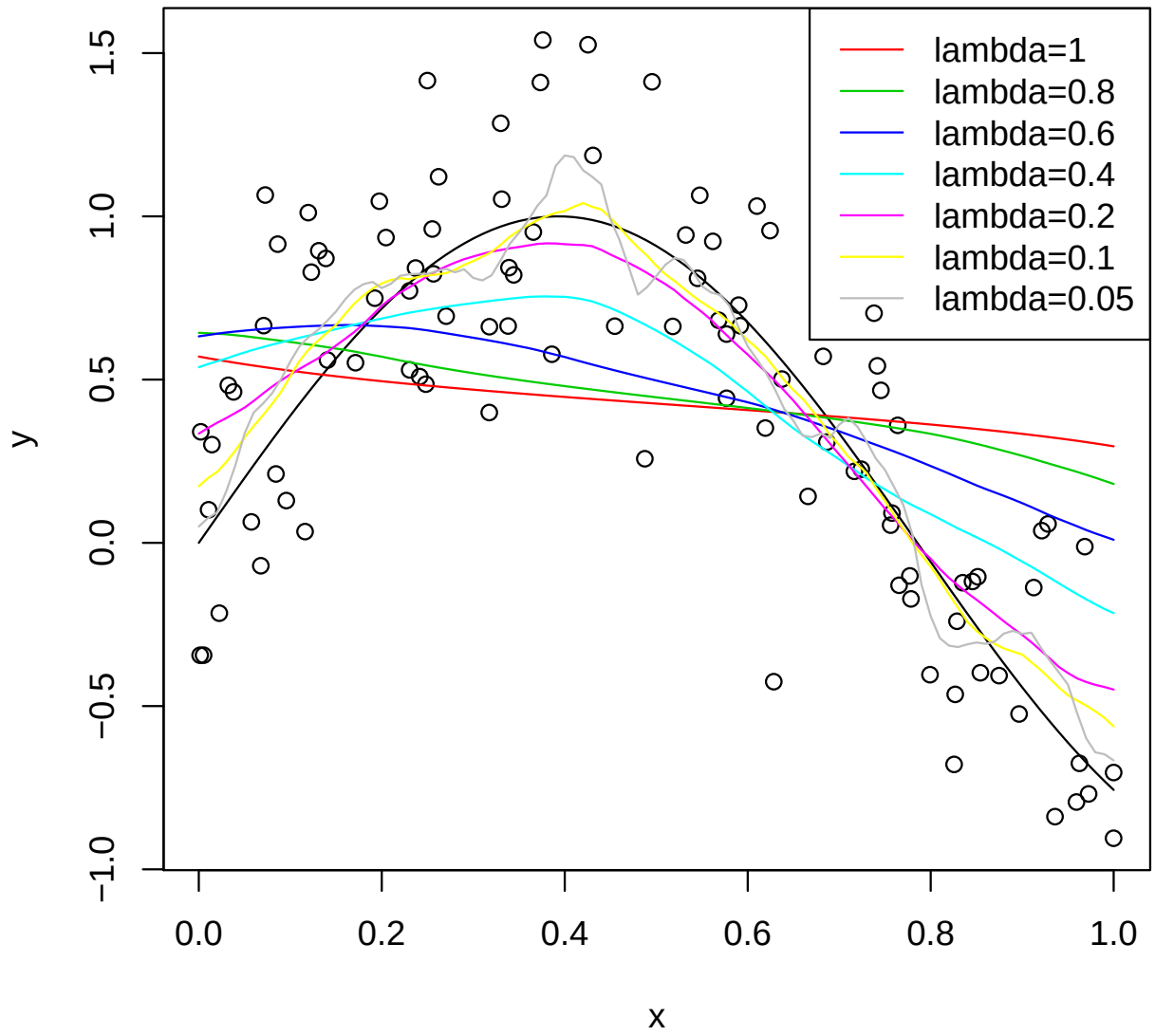
larger variance – as averaged over fewer points
smaller bias – as closer instances are used
$\Rightarrow$ risks to be too bumpy

If the bandwidth λ is large

smaller variance – as averaged over more points
larger bias – as instances further apart are used
$\Rightarrow$ risks to be too rigid / over-smoothed

The bandwidth λ is a parameter (sometimes called a **hyperparameter**) of the model that needs to be optimized / estimated by data.

Example / Epanechnikov Kernel, various bandwidths



Summary

- Simple classification and regression models can be built by
  - averaging over target values (regression)
  - counting the occurrences of the target class (classification)

  of training instances close by (measured in some **distance measure**).
- The **nearest neighbor** takes always a fixed number K of nearest points into account.
  - Alternatively, one also could weight points with some similarity measure (called **kernel**),
    $\Rightarrow$ the model is called **kernel regression** and **kernel classification**.
- There are no learning tasks for these models, as simply all training instances are stored ("memory-based methods").
- Therefore, to compute predictions is more costly than for say linear models. There are some acceleration techniques
  - **partial distances / lower bounding**

Further Readings

- [HTFF05, chapter 13.3, 2.3.2], [Mur12, chapter 1.4.2, 14.1+2+4], [JWHT13, chapter 2.2.3,].

References

Trevor Hastie, Robert Tibshirani, Jerome Friedman, and James Franklin.
The elements of statistical learning: data mining, inference and prediction, volume 27.
2005.

Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani.
An introduction to statistical learning.
Springer, 2013.

Kevin P. Murphy.
Machine learning: a probabilistic perspective.
The MIT Press, 2012.