Machine Learning

A. Supervised Learning

A.5. Nearest-Neighbor Models

Lars Schmidt-Thieme

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. Scalable Nearest Neighbor

Syllabus

Tue. 21.10.	(1)	0. Introduction
		A. Supervised Learning
Wed. 22.10.	(2)	A.1 Linear Regression
Tue. 28.10.	(3)	A.2 Linear Classification
Wed. 29.10.	(4)	A.3 Regularization
Tue. 4.11.	(5)	A.4 High-dimensional Data
Wed. 5.11.	(6)	A.5 Nearest-Neighbor Models
Tue. 11.11.	(7)	A.6 Support Vector Machines
Wed. 12.12.	(8)	A.7 Decision Trees
Tue. 18.11.	(9)	A.8 A First Look at Bayesian and Markov Networks
		B. Unsupervised Learning
Wed. 19.11.	(10)	B.1 Clustering
Tue. 25.11.	(11)	B.2 Dimensionality Reduction
Wed. 26.11.	(12)	B.3 Frequent Pattern Mining
		C. Reinforcement Learning
Tue. 2.12.	(13)	C.1 State Space Models
Wed. 3.12.	(14)	C.2 Markov Decision Processes

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

Minkowski Metric / L_p metric on $\mathcal{X} := \mathbb{R}^n$:

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

with $p \in \mathbb{R}, p \geq 1$.

$p = 1$ (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):

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

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 transfo

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

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

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

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

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**

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

and then predict the class with maximal predicted probability

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

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: $C := \text{ARGMIN-K}(D, K)$
6: $\hat{y} := \frac{1}{K} \sum_{k=1}^{K} y_{C_k}$
7: **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**
    $D_n := d(q, x_n)$
  $C := \text{ARGMIN-K}(D, K)$
  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 $(\hat{p})_{y \in \mathcal{Y}}$

Compute the argmin

procedure ARGMIN-K($x \in \mathbb{R}^N, K \in \mathbb{N}$)
    allocate array M of size K
    for $n = 1, \ldots, \min(K, N)$ **do**
        INSERT-TOPK(M, n, π_x)
    for $n = K + 1, \ldots, N$ **do**
        if $x_n < x_{M_K}$ **then**
            INSERT-TOPK(M, n, π_x)
    return M

procedure INSERT-TOPK($M \in \mathcal{X}^K, n \in \mathcal{X}, \pi : \mathcal{X} \to \mathbb{R}$)
    $k :=$ FIND-SORTED(M, n, π)
    for $l := K, K-1, \ldots, k+1$ **do**
        $M_l := M_{l-1}$
    $M_{k+1} := n$

Note: $\pi_x(n) := x_n$ comparison by x-values. Here, $\mathcal{X} := \mathbb{N}$.

Compute the argmin / find (naive)

1: **procedure** FIND-SORTED-LINEAR($x \in \mathcal{X}^K, z \in \mathcal{X}, \pi : X \to \mathbb{R}$)
2: $k := K$
3: **while** $k > 0$ and $\pi(z) < \pi(x_k)$ **do**
4: $k := k - 1$
5: **return** k

Note: Esp. for larger K it is better to use binary search.

Decision Boundaries

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

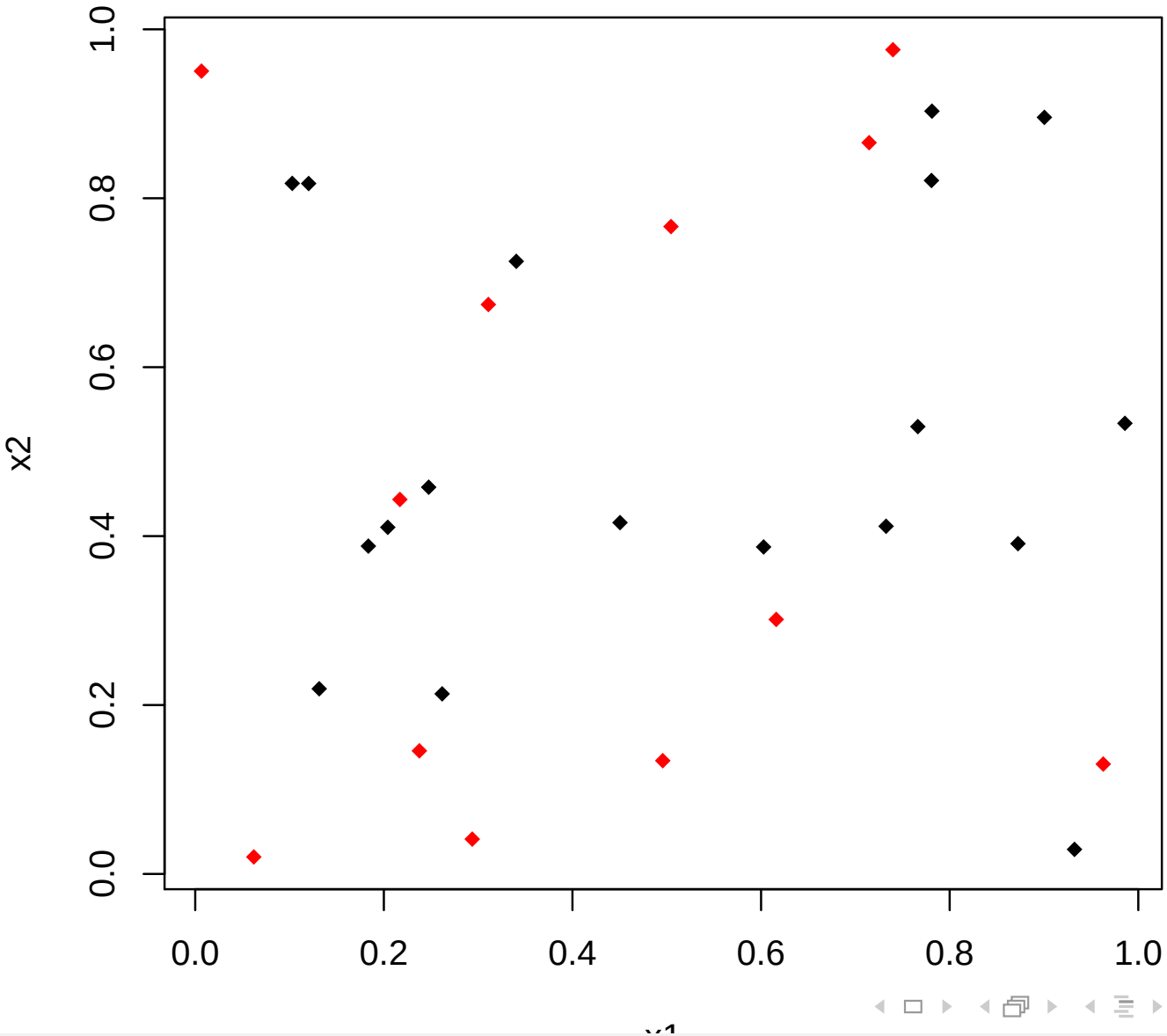
$$\text{region}_D(x_1), \text{region}_D(x_2), \ldots, \text{region}_D(x_N)$$

with

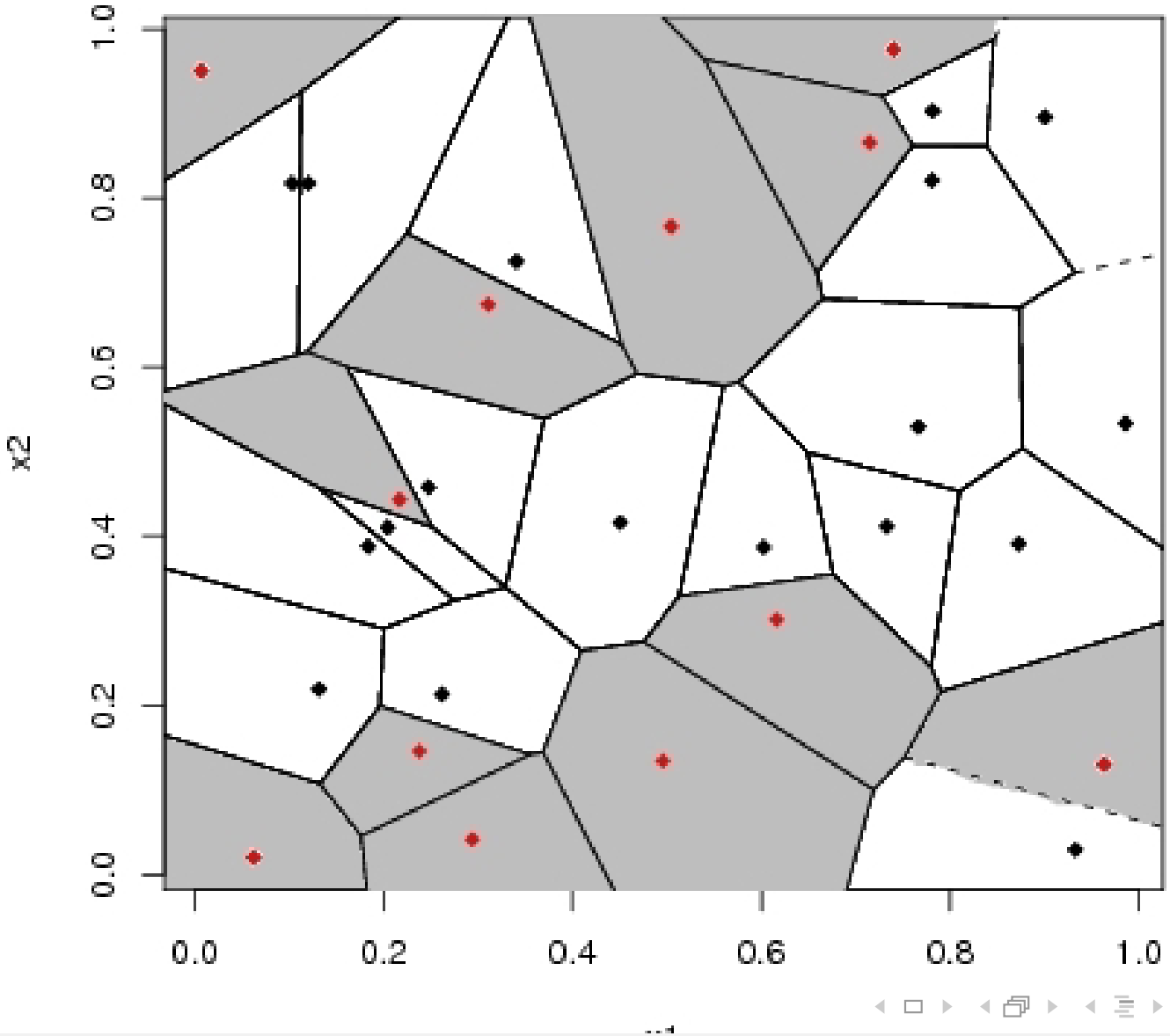
$$\text{region}_D(x) := \{x' \in \mathcal{X} \mid d(x', x) \leq d(x', x'') \quad \forall (x'', y'') \in D\}$$

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

Decision Boundaries



Decision Boundaries



Outline

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.,

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

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).

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: $C := \text{ARGMIN-K}(D, K)$
6: $\hat{y} := \frac{1}{K} \sum_{k=1}^{K} y_{C_k}$
7: **return** $\hat{y}$

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: $\quad C := \pi_1(\text{ARGCLOS-K}(q, x_1, x_2, \ldots, x_N, K))$
3: $\quad \hat{y} := \frac{1}{K} \sum_{k=1}^{K} y_{C_k}$
4: $\quad$ **return** $\hat{y}$

Note: $\pi_1(n, d) := n$ retains neighbor index n and discards its distance d.

Find Neighbors / Without Lower Bounding

1: **procedure** ARGCLOS-K($q \in \mathbb{R}^M, x_1, \ldots, x_N \in \mathbb{R}^M, K \in \mathbb{N}$)
2: allocate array M of size K for pairs $\mathbb{N} \times \mathbb{R}$.
3: **for** $n = 1, \ldots, \min(K, N)$ **do**
4: $d := \sum_{m=1}^{M} (q_m - x_{n,m})^2$
5: INSERT-TOPK($M, (n, d), \pi_2$)
6: **for** $n = K + 1, \ldots, N$ **do**
7: $d := \sum_{m=1}^{M} (q_m - x_{n,m})^2$
8: **if** $d < \pi_2(M_K)$ **then**
9: INSERT-TOPK($M, (n, d), \pi_2$)
10: **return** M

Note: ARGCLOS-K returns the K points closest to q and their distances.
$\pi_2(n, d) := d$ comparison by second component (distance).

Find Neighbors / With Lower Bounding

1: **procedure** ARGCLOS-K($q \in \mathbb{R}^M, x_1, \ldots, x_N \in \mathbb{R}^M, K \in \mathbb{N}$)
2: allocate array M of size K for pairs $\mathbb{N} \times \mathbb{R}$.
3: **for** $n = 1, \ldots, \min(K, N)$ **do**
4: $d := \sum_{m=1}^{M} (q_m - x_{n,m})^2$
5: INSERT-TOPK($M, (n, d), \pi_2$)
6: **for** $n = K + 1, \ldots, N$ **do**
7: $d := 0$
8: $m := 1$
9: **while** $m \leq M$ and $d < \pi_2(M_K)$ **do**
10: $d := d + (q_m - x_{n,m})^2$
11: $m := m + 1$
12: **if** $d < \pi_2(M_K)$ **then**
13: INSERT-TOPK($M, (n, d), \pi_2$)
14: **return** M

Note: ARGCLOS-K returns the K points closest to q and their distances.
$\pi_2(n, d) := d$ comparison by second component (distance).

Search trees

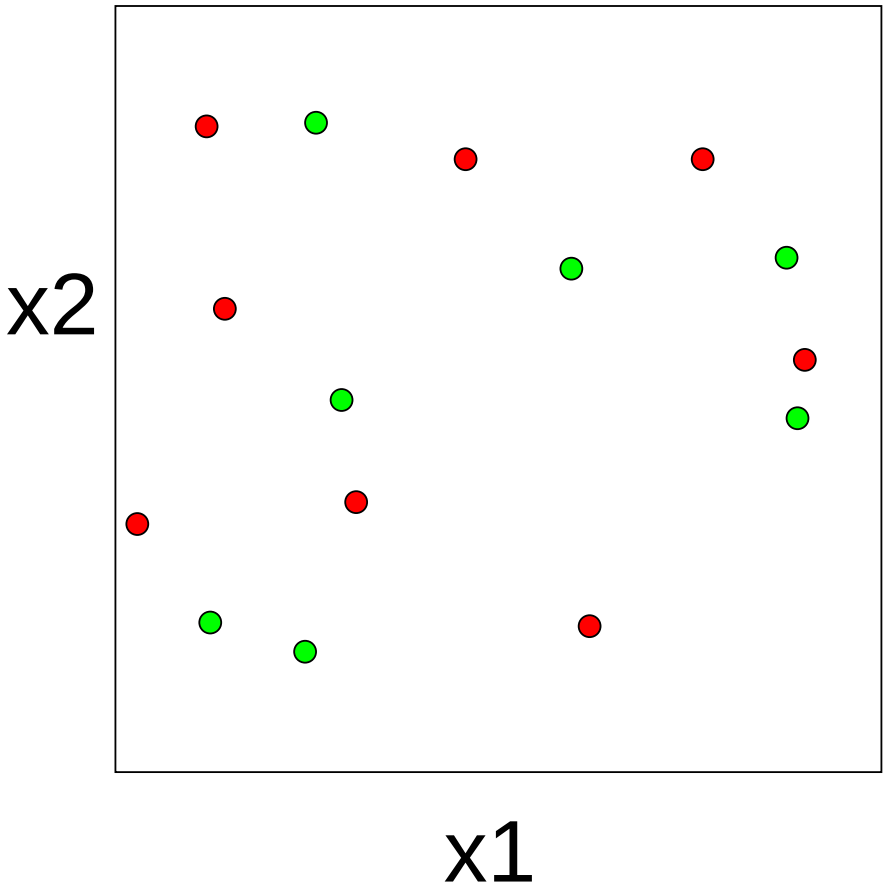
Search trees:

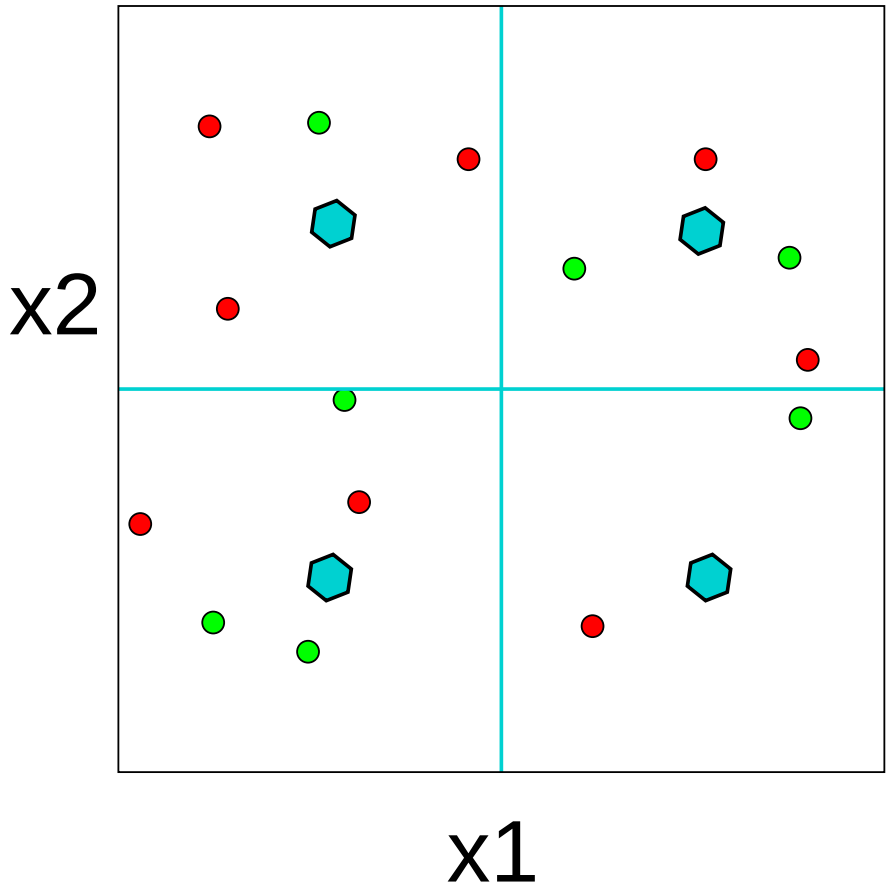
Do not compute the distance of a new point x to **all** training examples, but

1. organize the training examples as a tree (or a DAG) with
   - sets of training examples at the leaves and
   - a prototype (e.g., the mean of the training examples at all descendent leaves) at each intermediate node.
2. starting at the root, recursively
   - compute the distance to all children of the actual node and
   - branch to the child with the smallest distance,
3. compute distances only to training examples in the leaf finally found.

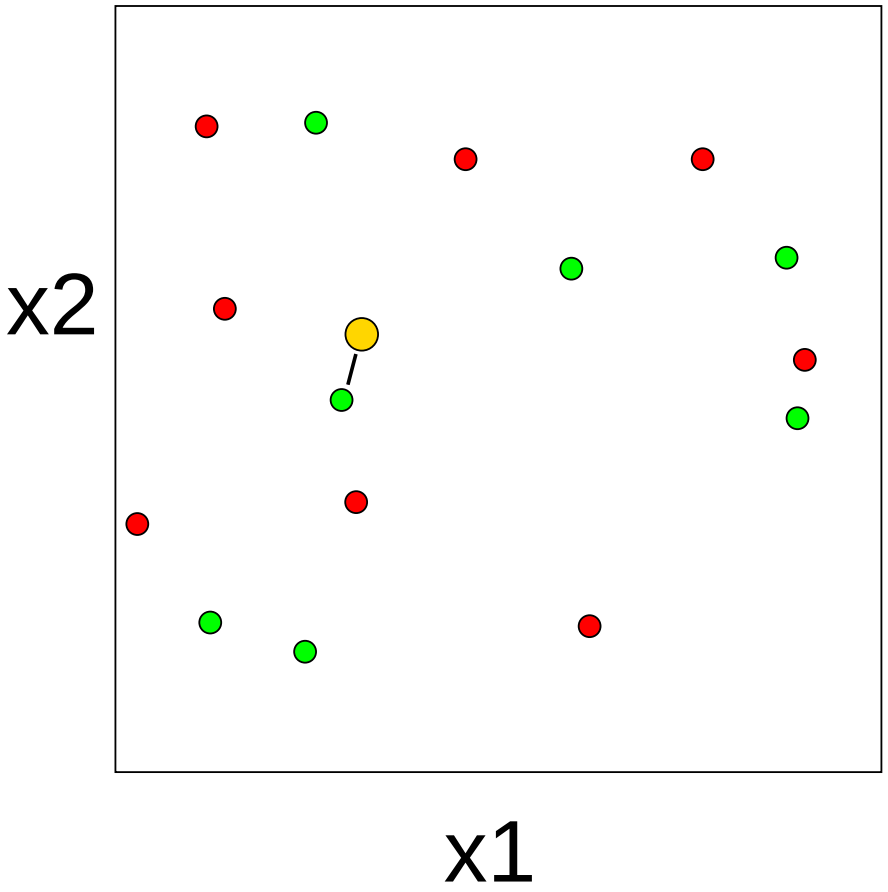
This is an approximation.

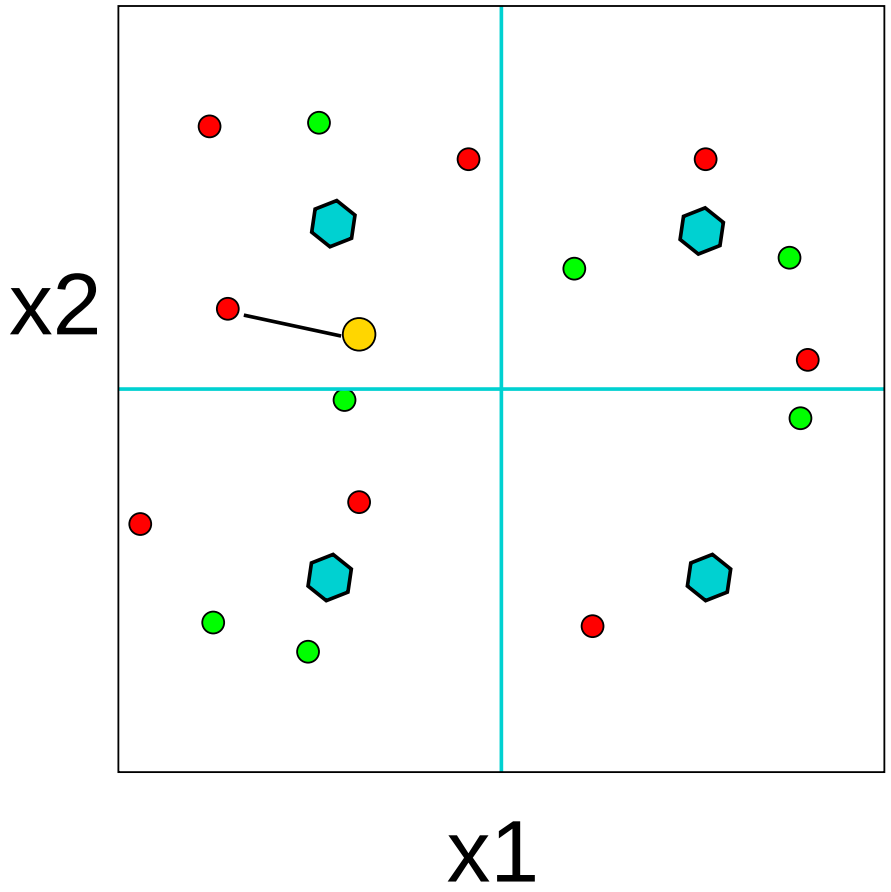
Search trees





Search trees





Approximate Nearest Neighbor

- for low dimensions, **k-d trees** (k-dimensional trees) can be used
  - only useful for very low dimensions (2d, 3d)
  - in computation geometry, computer graphics, computer vision
- for higher dimensions **locality-sensitive hashing** performs better
  - only works with specific distances (Euclidean/L2, L1, Hamming)

Locality-Sensitive Hashing [DIIM04]

- idea: create a hash key function h that puts
  - close instances into the same bin, but
  - far instances into different bins.

  allowing some errors.
- for $x \in \mathbb{R}^M$, the **discretized projection on a random line** is

$$h_{a,b,r}(x) := \left\lfloor \frac{a^T x + b}{s} \right\rfloor, \quad a \in \mathbb{R}^M, b \in [0, s], s \in \mathbb{R}^+$$

$$\text{where } a_m \sim \mathcal{N}(0,1), b \sim \text{unif}(0, s)$$

- use the concatenation of L such projection keys as hash key

$$\begin{aligned} h_{A,b,r}(x) &:= (h_{A_{l,.},b_l,s}(x))_{l=1,\ldots,L} \\ &= (\left\lfloor \frac{1}{s}(Ax + b)_l \right\rfloor)_{l=1,\ldots,L}, \quad A \in \mathbb{R}^{L \times M}, b \in [0, s]^L, s \in \mathbb{R}^+ \end{aligned}$$

- build H such hash maps and test all points found in any of them.

Editing

Editing / Pruning / Condensing:

shrink the set of training data points,
e.g., select a subset of the original training data points.

Example: remove all points with cells that are surrounded by cells of points of the same class.

$$X_{\text{edited}} := \{(x, y) \in X \mid \exists (x', y') \in X, R(x') \cap R(x) \neq \emptyset \text{ and } y' \neq y\}$$

This basic editing algorithm

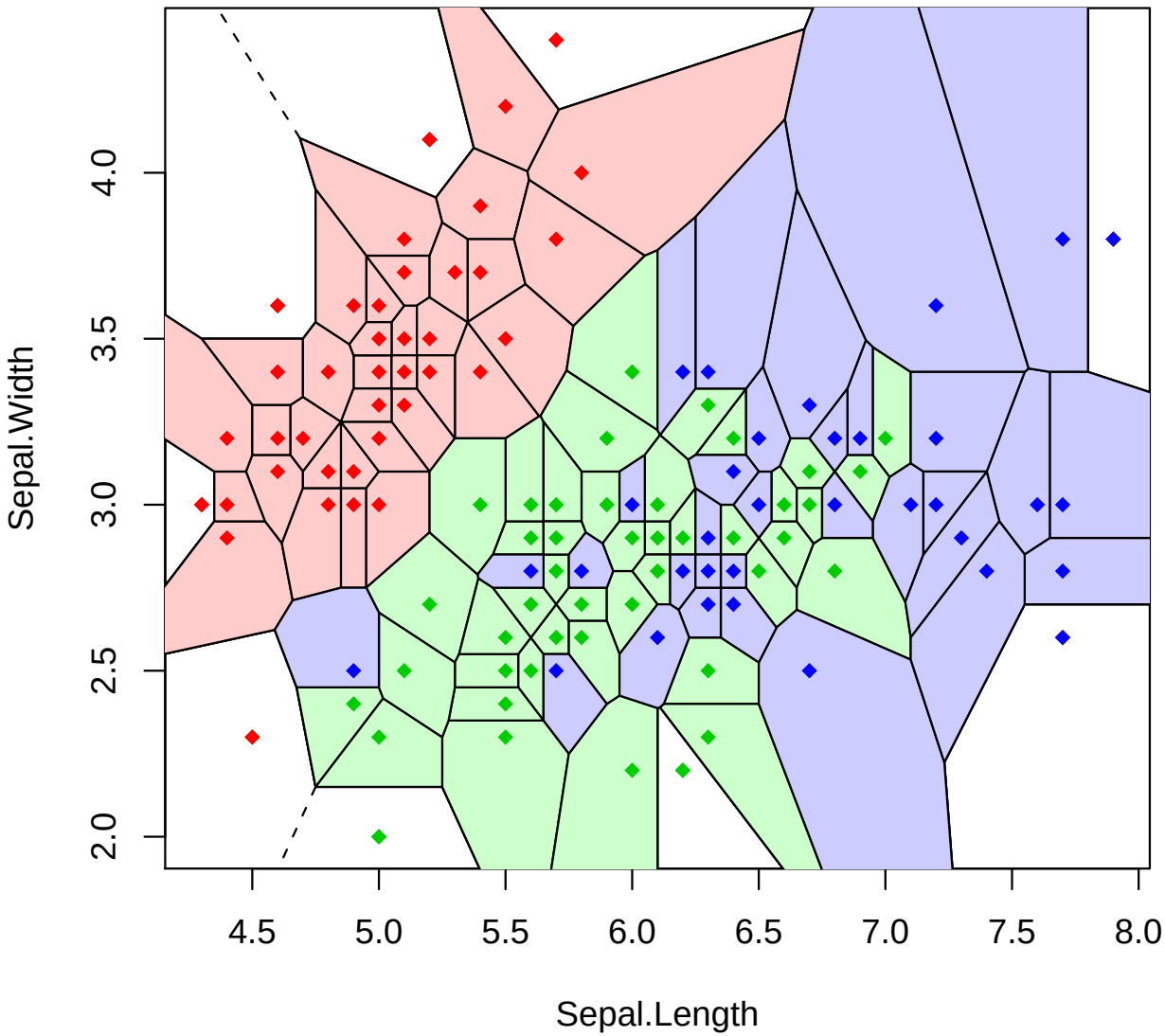
- retains the decision function,
- has complexity $O(M^3 N^{\lfloor \frac{M}{2} \rfloor} \log N)$
  (with $\lfloor x \rfloor := \max\{n \in \mathbb{N} \mid n \leq x\}$; Duda et al. 2001, p. 186).

See e.g., Ottmann/Widmayer 2002, p. 501–515 for computing Voronoi diagrams in two dimensions.

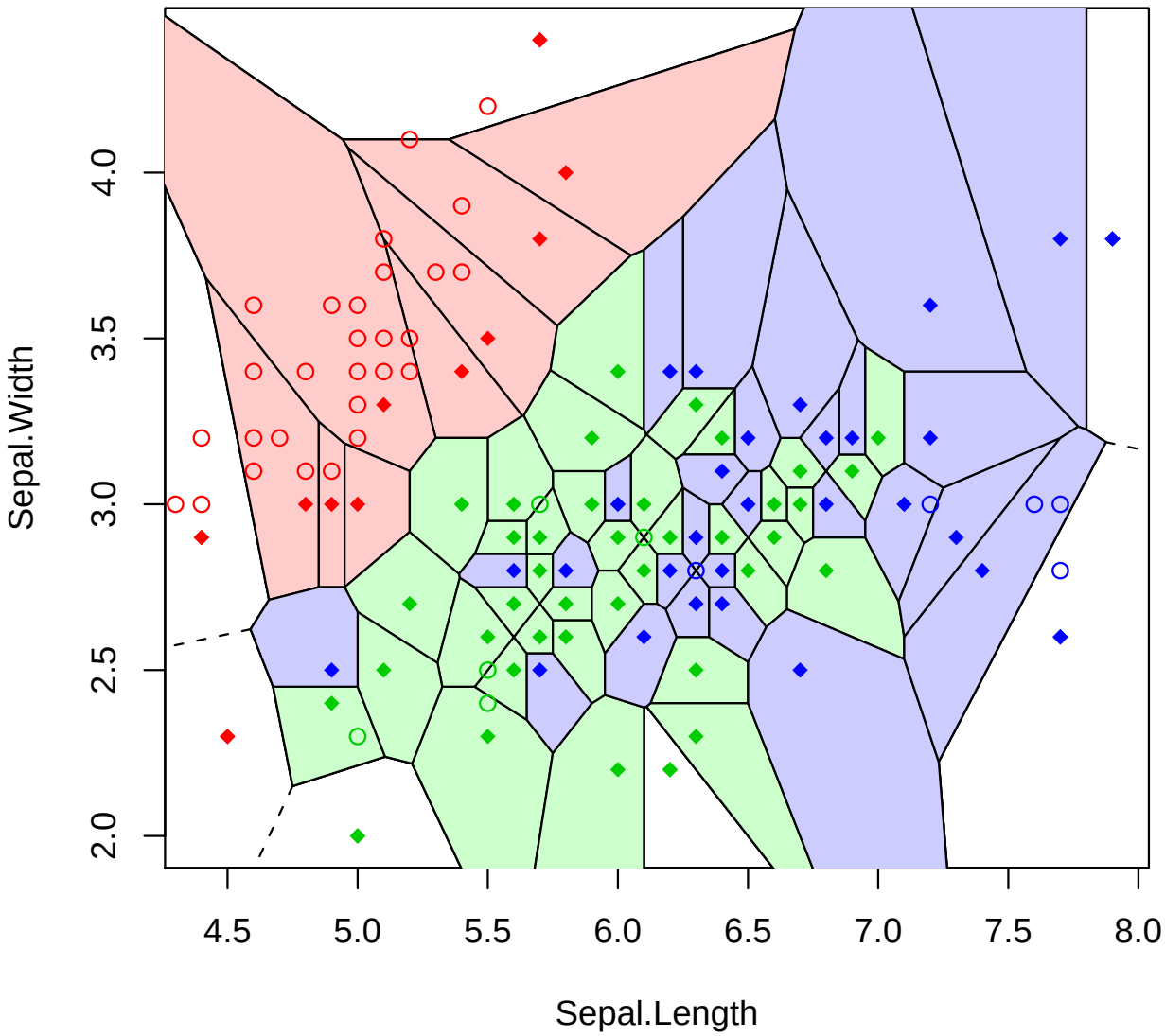
Editing

procedure KNN-EDIT-TRAINING-DATA($\mathcal{D}^{\text{train}} \subseteq \mathbb{R}^M \times \mathcal{Y}$)
  compute Voronoi cells $R(x)$ for all $(x, y) \in \mathcal{D}^{\text{train}}$,
    esp. Voronoi neighbors $N(x) := \{(x', y') \in \mathcal{D}^{\text{train}} \mid R(x) \cap R(x') \neq \emptyset\}$
  $E := \emptyset$
  for $(x, y) \in \mathcal{D}^{\text{train}}$ **do**
    hasNeighborOfOtherClass := false
    for $(x', y') \in N(x)$ **do**
      if $y \neq y'$ **then**
        hasNeighborOfOtherClass := true
    if not hasNeighborOfOtherClass **then**
      $E := E \cup \{(x, y)\}$
  return $\mathcal{D}^{\text{train}} \setminus E$

Editing



Editing



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** or **Parzen window**),
    $\Rightarrow$ 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 several acceleration techniques
  - **partial distances / lower bounding**
  - **search trees** / **locality-sensitive hashing**
  - **editing**

Further Readings

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

References

Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni.
Locality-sensitive hashing scheme based on p-stable distributions.
In *Proceedings of the twentieth annual symposium on Computational geometry*, pages 253–262. ACM, 2004.

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.