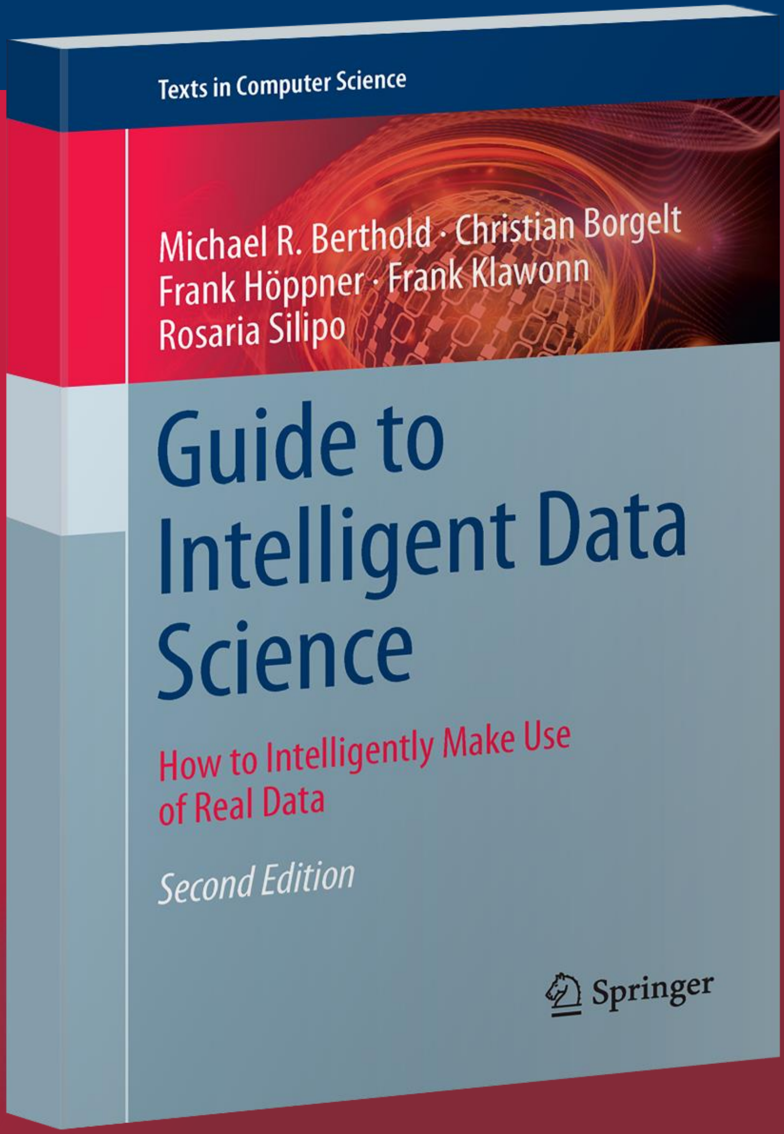
Nearest Neighbor Predictors



Summary of this lesson

“Good fences make good neighbors”
-Robert Frost

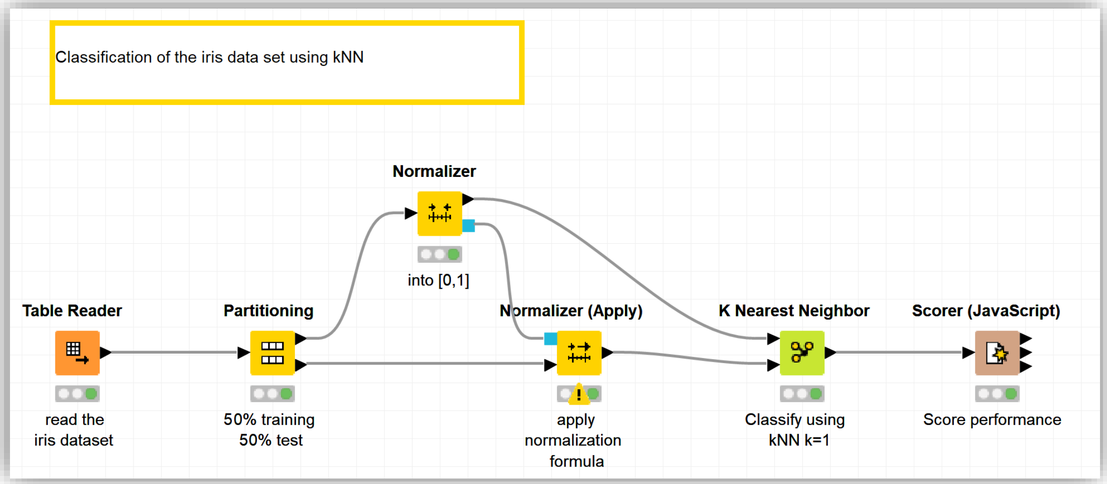
Can we learn from surrounding elements?

**This lesson refers to chapter 9 of the GIDS book*

Content of this lesson

- Lazy learners vs eager learners
- k-nearest neighbor (kNN) predictors
- Weighting & prediction functions
- Choosing parameter k

- Datasets used : iris dataset
- Example Workflow:
  - „Classification of the iris data using kNN“ https://kni.me/w/ZVkD_W8LnSh_t9Na
    - normalization
    - kNN with k=1



Lazy and Eager Learners

- One of the simplest learning methods
- Predict class labels or target values from nearest neighbors
- Majority voting – classification
- Averaging – numeric prediction

An example of lazy learners, in contrast to eager learners

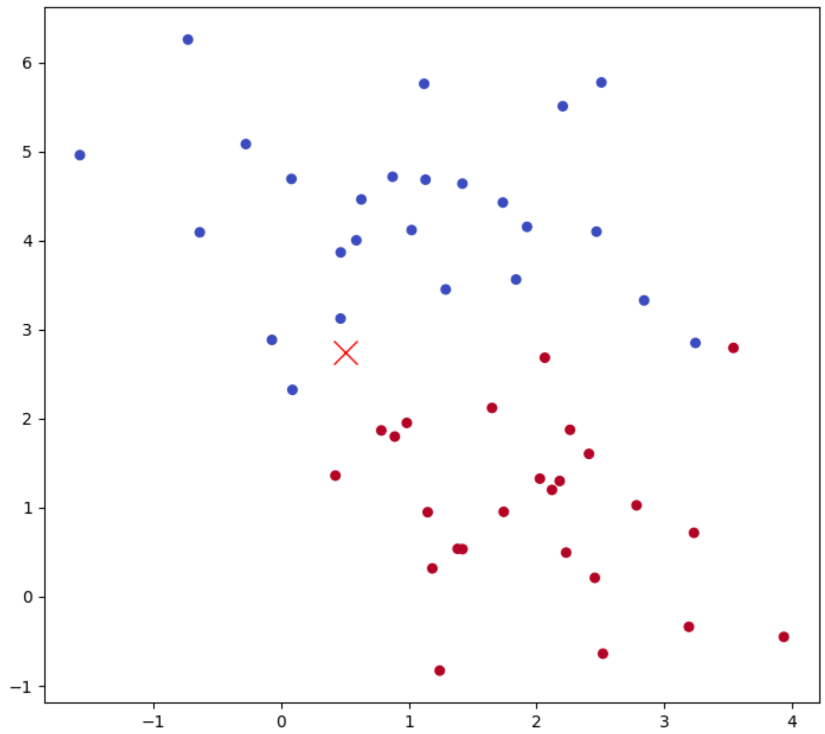
- **Lazy learners**: Save all data from training, use it for classifying

  (The learner was lazy, classifier had to do the work)

- **Eager learners**: Build a (compact) model/structure during training, use the model for classification.

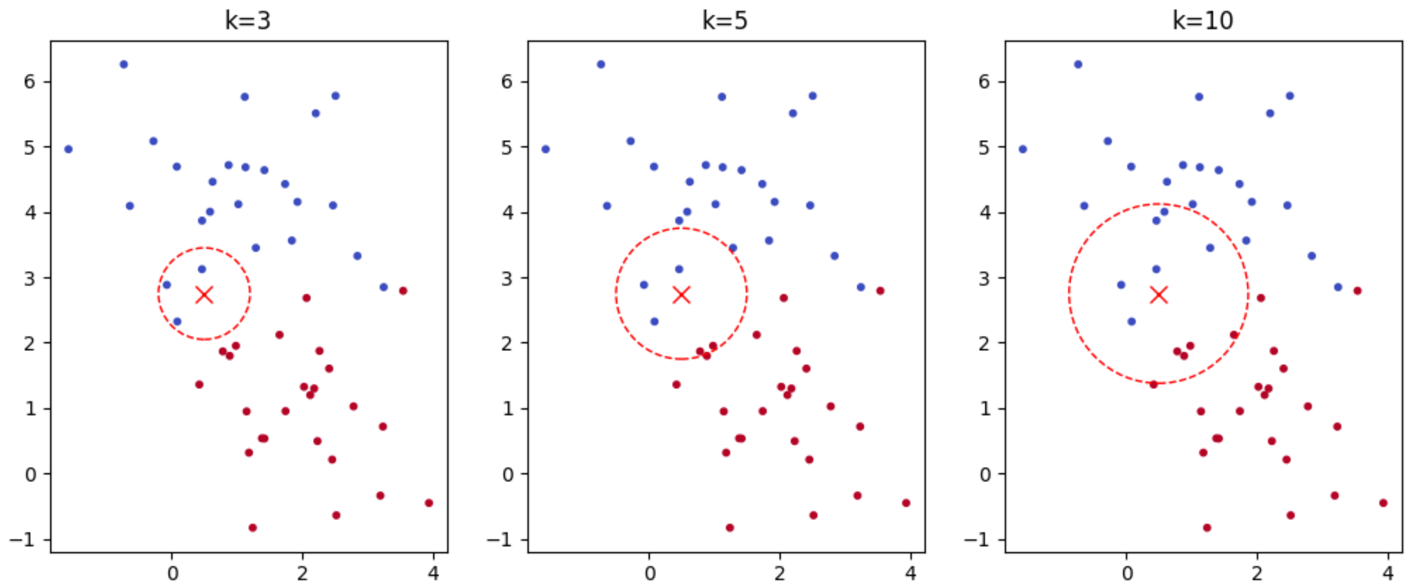
  (The learner was eager/worked harder, classifier had simple life)

- How to classify a new observation (red X)? Blue or red?
- Solution: the majority vote of its neighbors,



Examining k-nearest neighbors, decided based on the majority vote

- k=3: 3 blues ➔ classified as blue
- k=5: 3 blues, 2 reds ➔ classified as blue
- k=10: 6 blues, 4 reds ➔ classified as blue



An example of lazy learners, in contrast to eager learners

- ***Lazy learners***: Save all data from training, use it for classifying

  (The learner was lazy, classifier had to do the work)

- ***Eager learners***: Build a (compact) model/structure during training, use the model for classification.

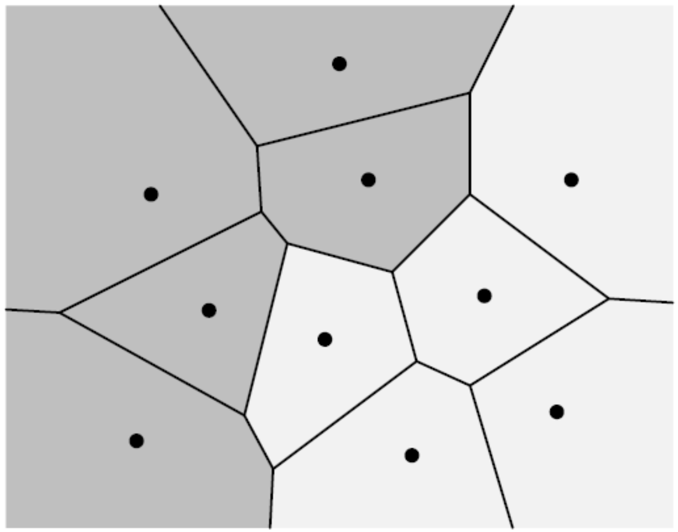
  (The learner was eager/worked harder, classifier had simple life)

k-Nearest Neighbor Predictors

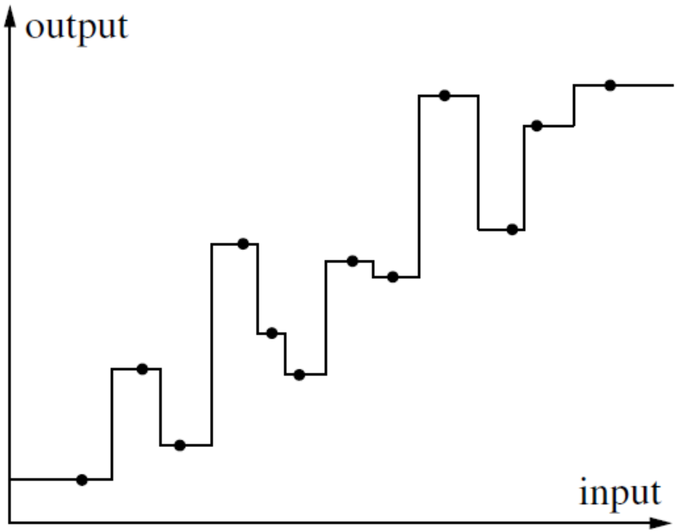
Nearest-Neighbor Predictors

- Nearest neighbor predictors are special case of ***instance-based learning***
- Instead of constructing a model that generalizes beyond the training data, the training examples are merely stored.
- Predictions for new cases are derived directly from these stored examples and their (known) classes or target values.

– For a new instance, use the target value of the closest neighbor in the training set



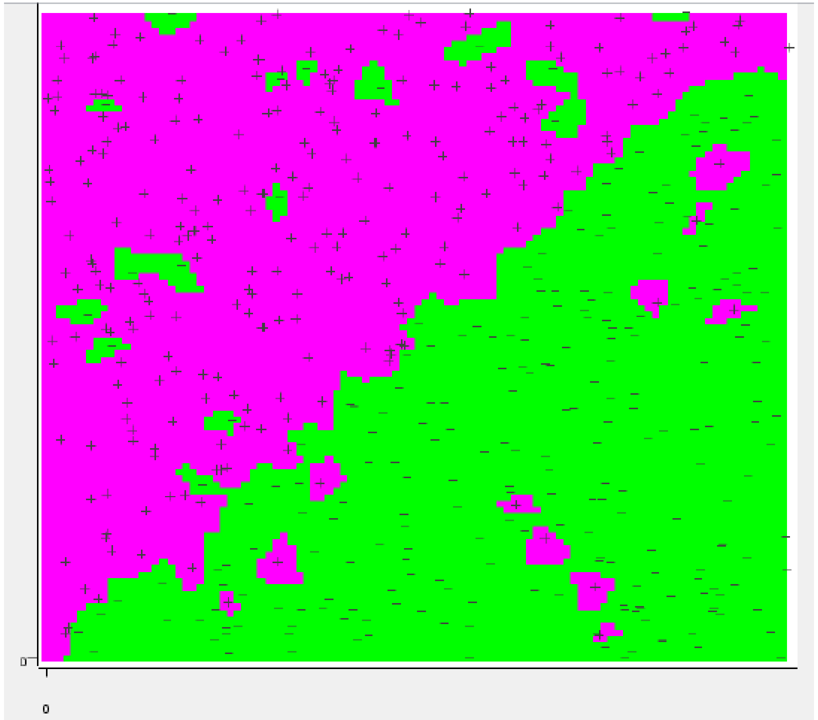
Classification



Regression

– Nearest neighbor predictors are sensitive to noises

➔ How can we overcome this?



Prediction with k neighbors ($k > 1$) taken into account

➔ ***k-nearest neighbor predictor***

- **Classification**: Choose the majority class among the k nearest neighbors for prediction
- **Regression**: Take the mean value of the k nearest neighbors for prediction

Problem:

- All k nearest neighbors have the same influence on the prediction.

➔ Closer nearest neighbors should have higher influence

Ingredients of kNN

Distance Metric:

– Determines which of the training examples are nearest to a query data point
– Possible scaling or weighting of some attributes

Number of Neighbors (k):

– The number of neighbors to be considered
– In theory it can range from 1 to all data points

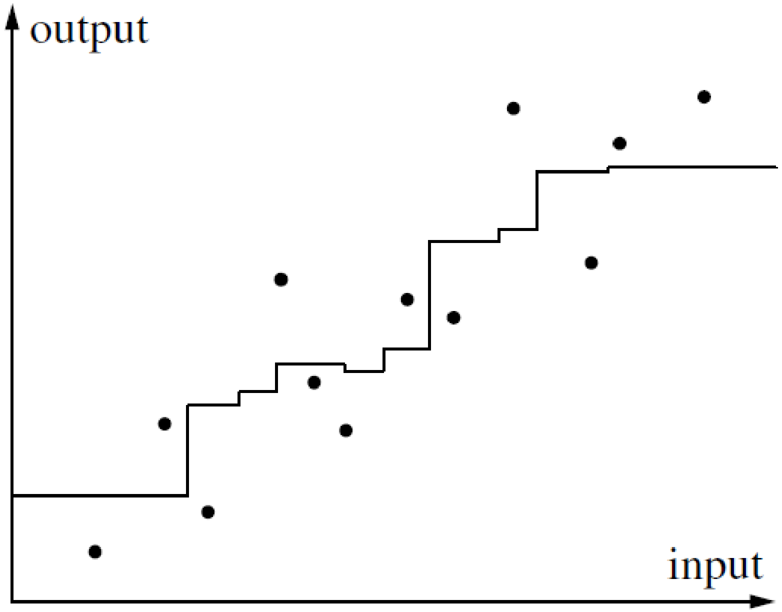
Weighting function:

- Weighting function defined from the query point
- Higher (lower) values for smaller (larger) distances

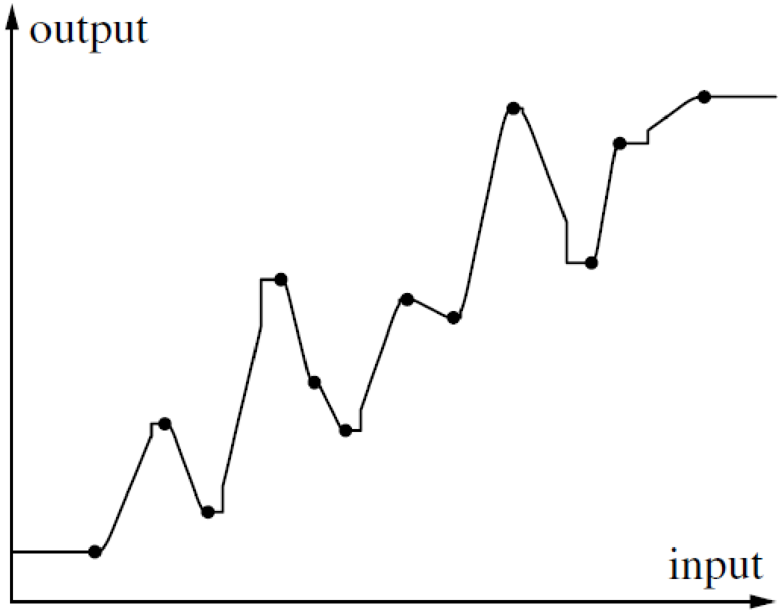
Prediction function:

- A way to compute the prediction from the neighbors
- Neighbors may differ from each other ➔ may not produce a unique prediction

Average (3 nearest neighbors)



Distance weighted (2 nearest neighbors)



Distance metric

- Problem dependent – often Euclidean

Number of Neighbors (k)

- Often chosen by cross-validation
- Should use an odd number to avoid possible ties in classification

Weighting function

- Example: tri-cubic weighting function $w(s_i, q, k) = \left(1 - \left(\frac{d(s_i, q)}{d_{max}(q, k)}\right)^3\right)^3$
- q: Query point
- s_i: Input vector of the i-th nearest neighbor
- k: Number of neighbors to be considered
- d: Distance function
- $d_{max}(q, k)$: Maximum distance between any two nearest neighbors and the distances of the nearest neighbours to the query point

Prediction function

Regression

- Weighted average of the target of the nearest neighbors

Classification

- Sum up the weights for each class among the nearest neighbors.
- Choose the class with the highest weighted sum

A k-nearest neighbor predictor with a weighting function

- Interpreted as an n-nearest neighbor predictor with a modified weighting function
- The modified weighting function simply assigns 0 to all instances not belonging to the k nearest neighbors.

More general approach

- Use a kernel function assigning distance-dependent weights to all instances in the training data set.

Kernel Functions

A kernel function $K(\cdot)$:

- $K(d)$ a function of distance d (originating from a query point)
- $K(d) \geq 0$
- $K(0) = 1$ (or it peaks at 0)
- $K(d)$ decreases monotonically as d incerases

Kernel Function Examples

Typical examples for kernel functions

$$K_{rect}(d) = \begin{cases} 1 & \text{if } d \leq \sigma \\ 0 & \text{otherwise} \end{cases}$$

$$K_{triangle}(d) = K_{rect}(d) \cdot \left(1 - \frac{d}{\sigma}\right)$$

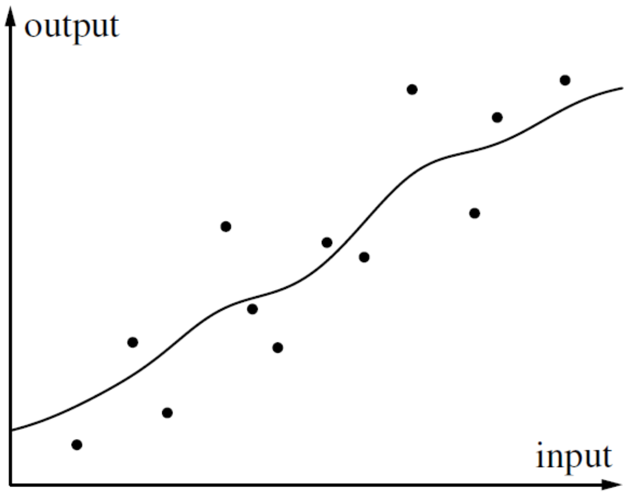
$$K_{tricubic}(d) = K_{rect}(d) \cdot \left(1 - \frac{d^3}{\sigma^3}\right)^3$$

$$K_{Gauss}(d) = exp\left(-\frac{d^2}{2\sigma^2}\right)$$

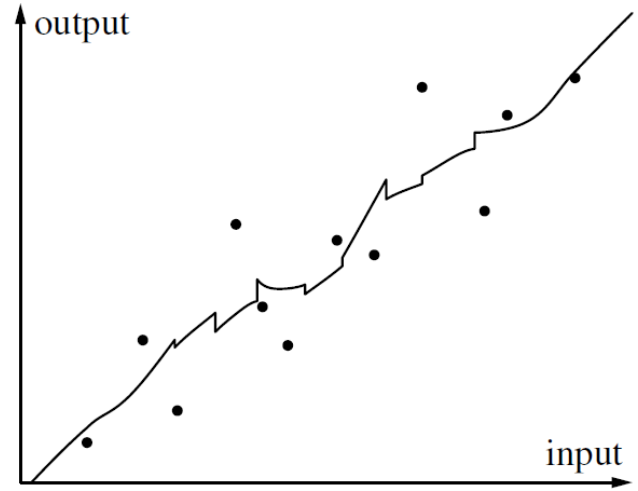
Where $\sigma > 0$ is a predefined constant

- For regression, we can use weighted averaging of the target
- Alternatively, we can also compute a local weighed-regression function at the query point

Kernel weighted regression



Distance-weighted local regression (k=4, tricubic)



Adjusting the Distance Function

- Choice of a distance function ➔ crucial in nearest neighbor methods
- Weighted features in a distance function ➔ more emphasis on important features
- Feature weights can be found based on heuristic strategies
  - Hill climbing, simulated annealing, evolutionary algorithms, etc.
- Can be evaluated via cross-validation

Nearest neighbor methods

- Pro: no training is needed
- Con: prediction on a large data set is computationally demanding

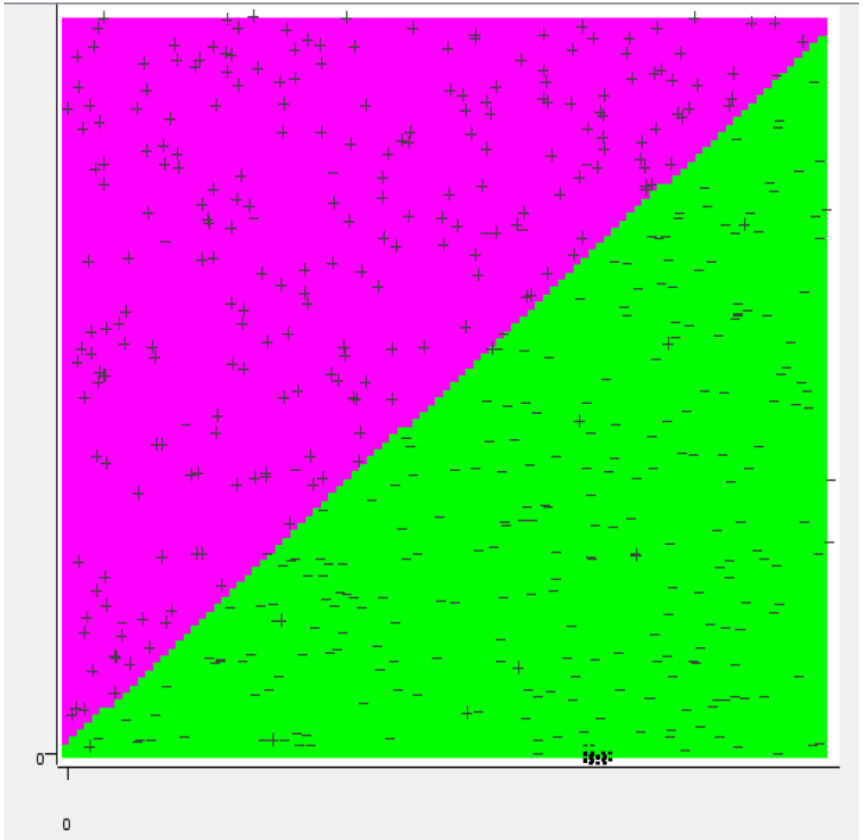
Solutions:

- Smaller subset of the training data for the nearest neighbor predictor
- Prototypes by merging close instances, e.g., by averaging

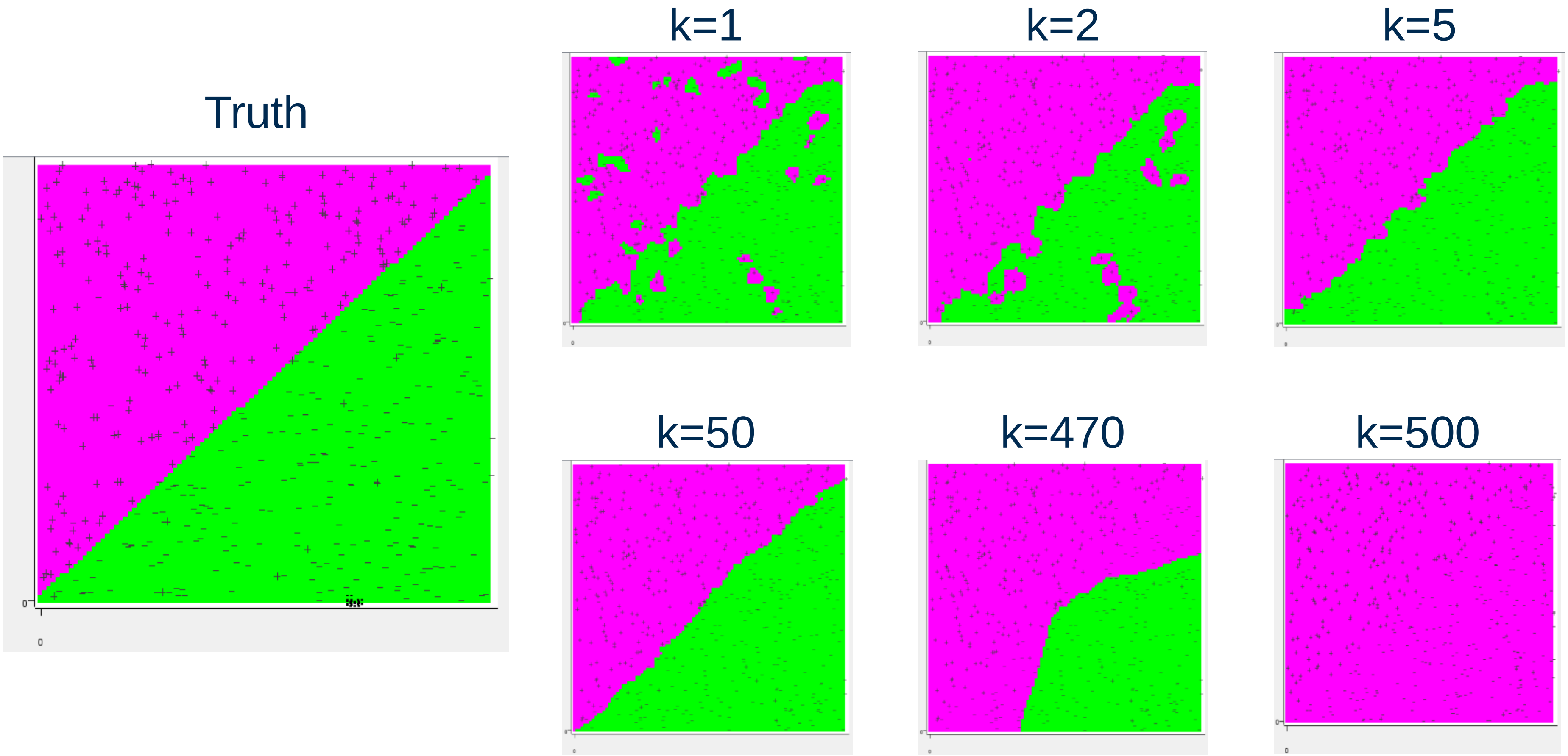
➔ Can be carried out based on cross-validation and using heuristic optimization strategies

Choice of Parameter k

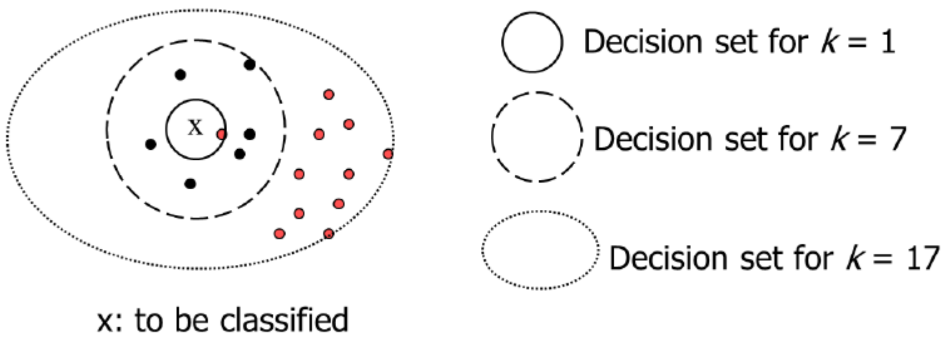
– Linear classification problem (with some noise)



– Decision boundaries with different k

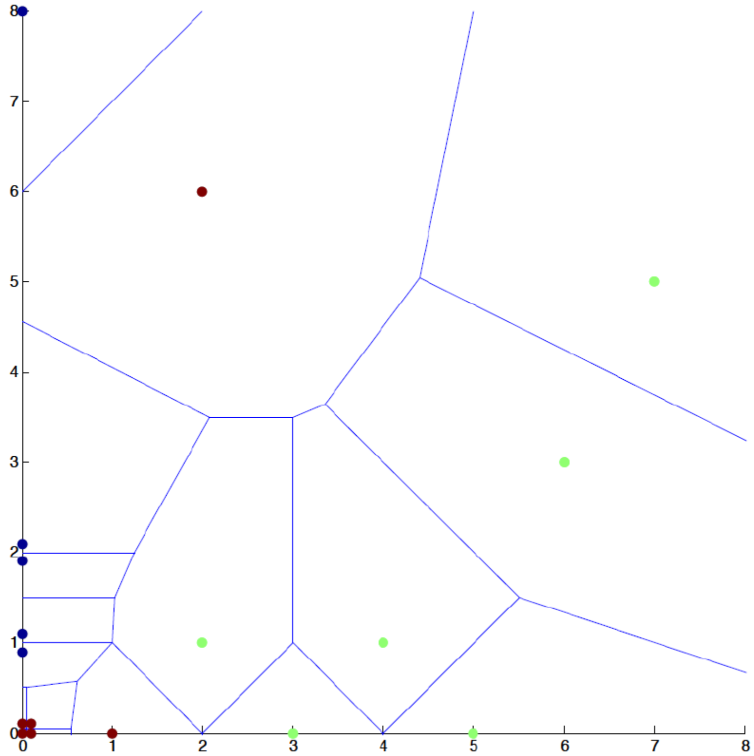


- k=1: y=piecewise constant labeling
- k too small: very sensitive to outliers
- k too large: many objects from other classes in the decision set
- k = N: y=globally constant (majority) label



➔ k can be determined manually, or heuristically (such as cross-validation)

– Simple classifier, k=1. Voronoi tessellation of input space



Special Case, k=1

Highly localized classifier, perfectly fits separable training data

Bias of the Learning Algorithm?

- No variations in search: simple store all examples

Model Bias?

- Classification via Nearest Neighbor

Hypothesis Space?

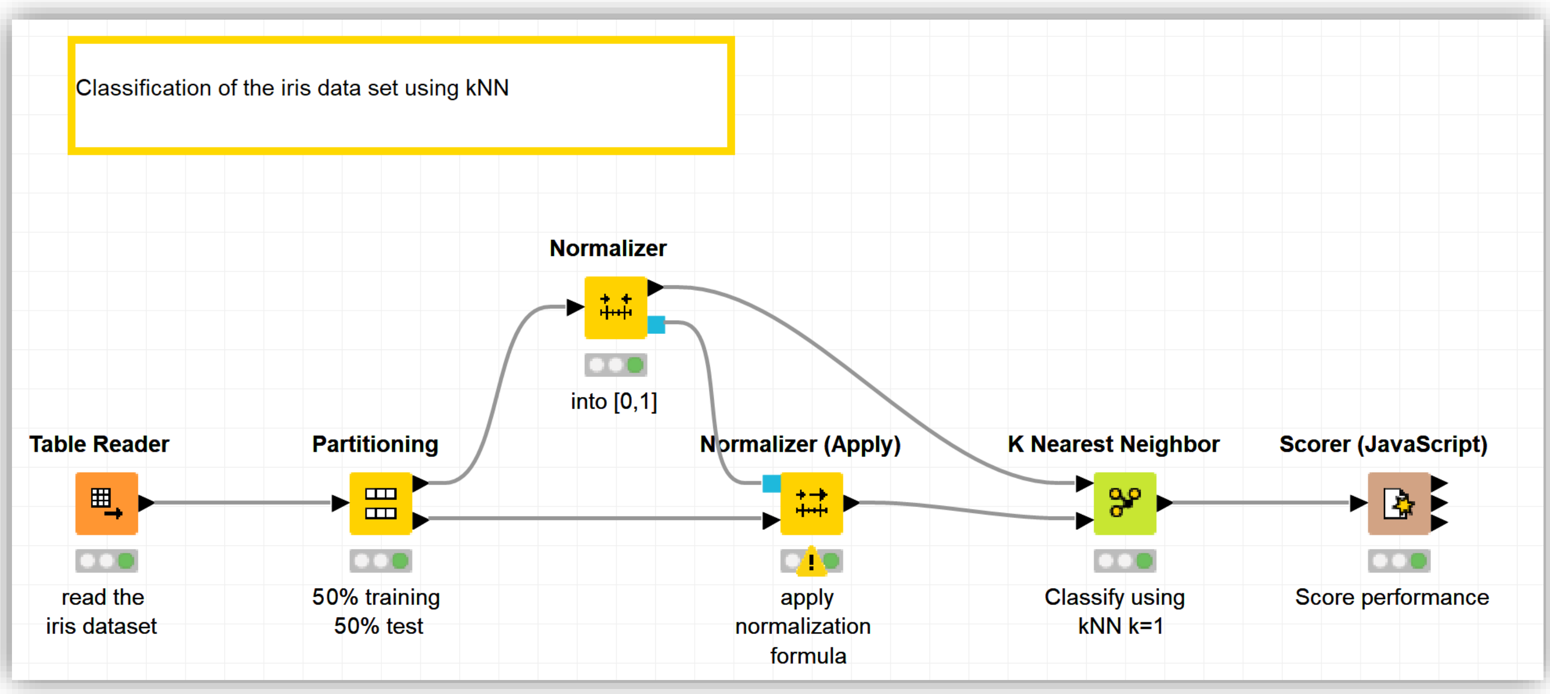
- One hypothesis only: Voronoi partitioning of space

Nearest neighbor classifiers are:

- Instance-based classifiers ➔ remember all training cases
- Sensitive to neighborhood – things to consider:
  - Number of neighbors k
  - Distance function
  - Weighting function
  - Prediction function

Practical Examples with KNIME Analytics Platform

- Classification of the iris data using kNN



Thank you

For any questions please contact: education@knime.com