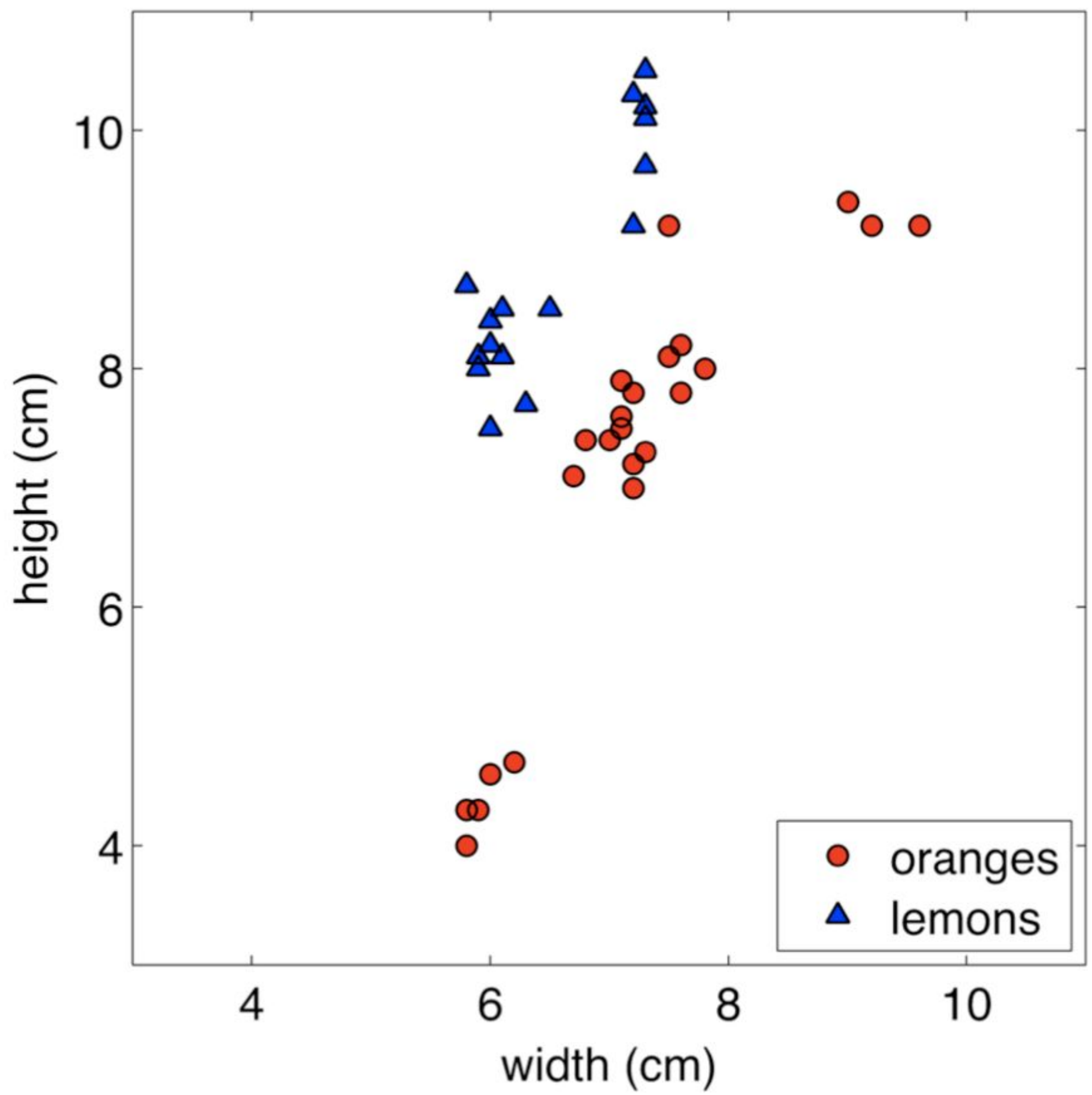
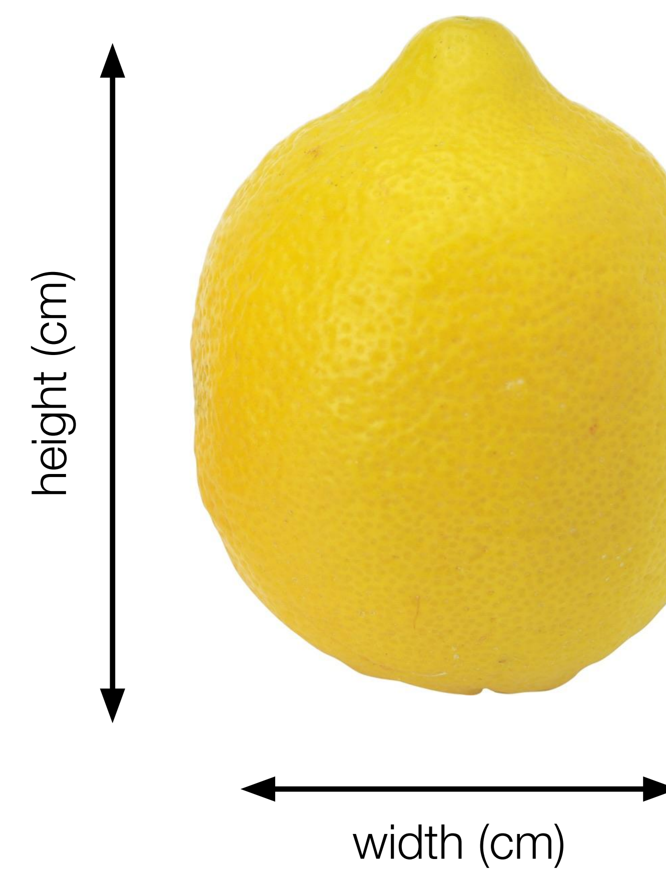
k-Nearest Neighbors

Intuition

- An example: “oranges” vs “lemons”



Binary classifier based on two simple features:



Nearest Neighbors

- Assume that your training examples corresponds to points in d-dimensional Euclidean space
  - Key idea: the value of the target function for a new sample is estimated from the known (stored) training examples
    - This is done by computing distances between the new sample and all the training samples
    - Decision rule: assign the label of the nearest example

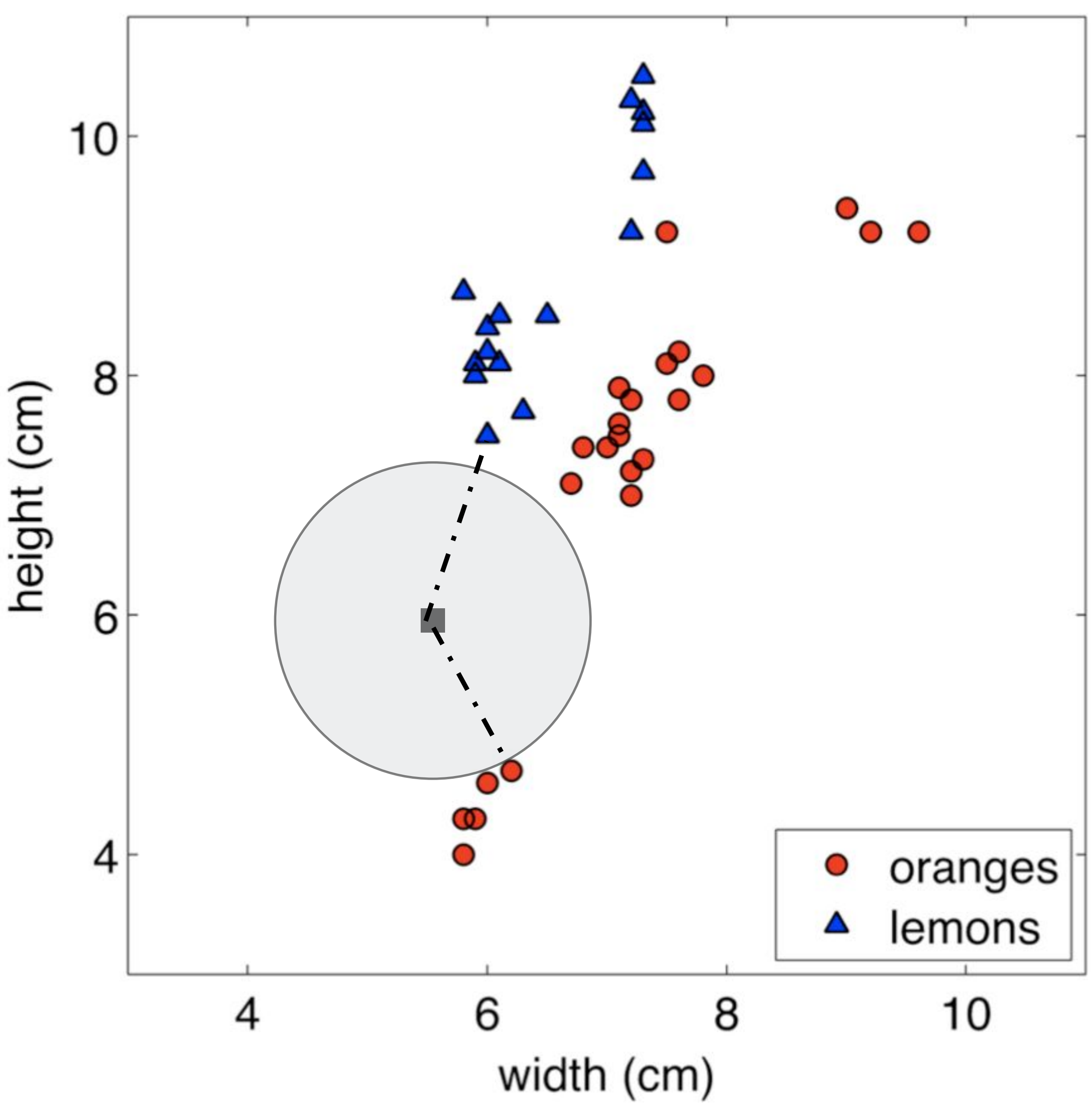
Algorithm:

Find (x^*, y^*) (from the stored training set) closest to the test sample x

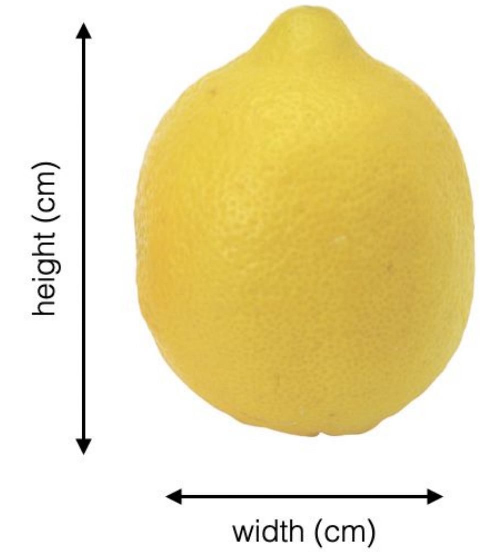
i.e. $x^* = \underset{x^{(i)} \in TrainSet}{\arg\min} \text{Distance}(x^{(i)}, x)$. Output: $y = y^*$

Nearest Neighbors

- An example: “oranges” vs “lemons”



Binary classifier based on two simple features:



$$= |X - Y| = \sqrt{\sum_{i=1}^{i=n} (x_i - y_i)^2}$$

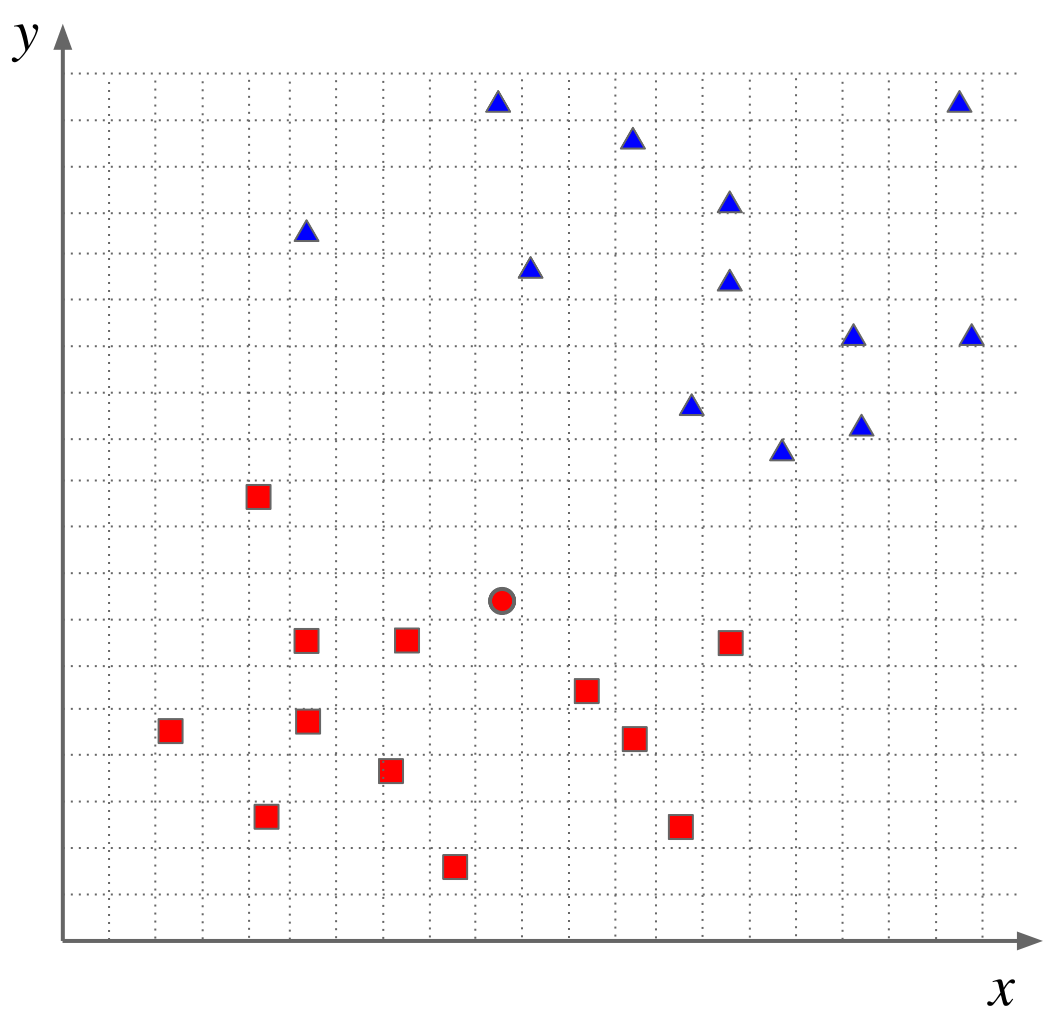
Nearest Neighbor classifier

- Compute distances (Euclidean):

$$d = \mathrm{sqrt}((x_q - x_p)^2 + (y_q - y_p)^2)$$

Class 1

Class 2



x_q	y_q
10	8

x_q	y_q	d	x_q	y_q	d
9	2	6.1	16	11	6.7
5	3	7.1	14	12	5.7
14	3	6.4	18	12	8.9
8	4	4.5	18	14	10
3	5	7.6	20	14	11.7
6	5	5.0	11	15	7.1
13	5	4.2	15	15	8.6
12	6	2.8	6	16	8.9
6	7	4.1	15	17	10.3
8	7	2.2	13	18	10.4
15	7	5.1	10	19	11
5	10	5.4	20	19	14.9
15	7		10	19	
5	10		20	19	

Nearest Neighbor classifier

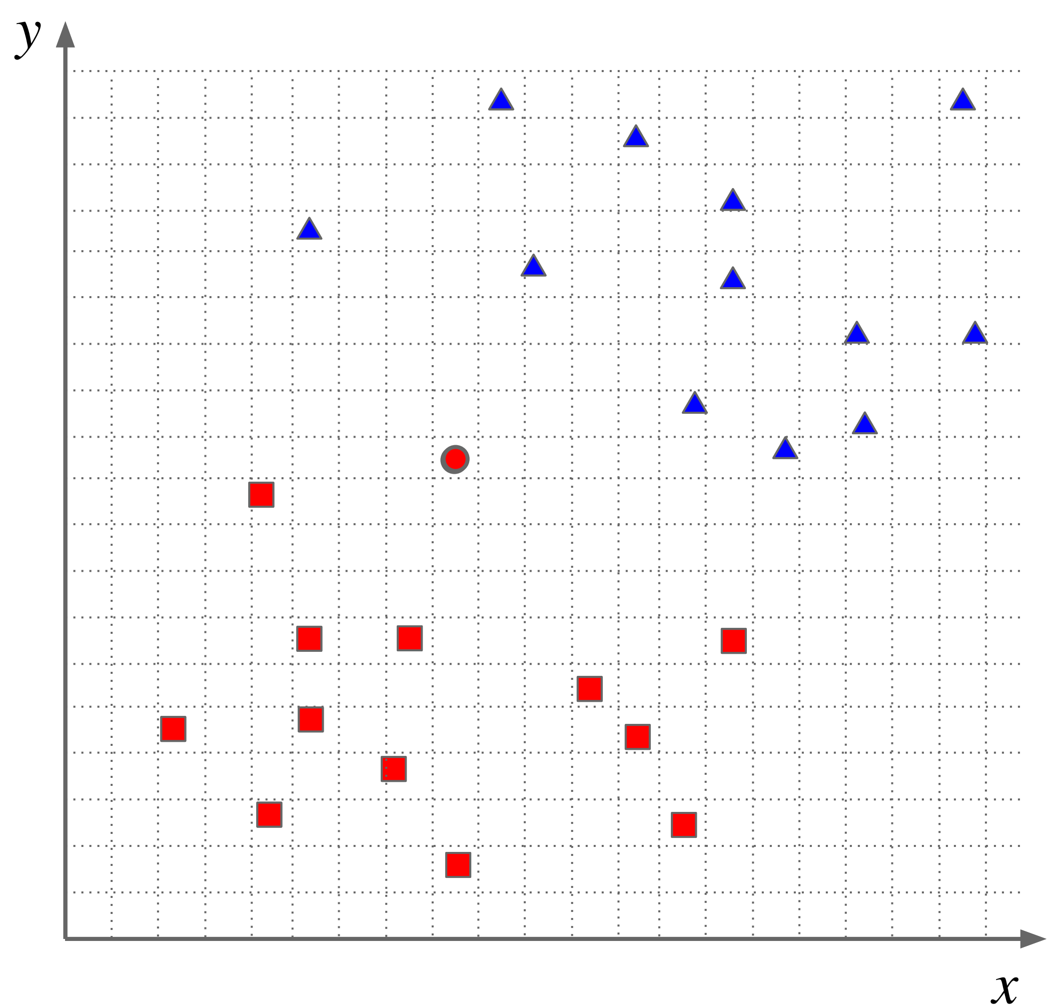
- Compute distances (Euclidean):

$$d = \mathrm{sqrt}((x_q - x_p)^2 + (y_q - y_p)^2)$$

Class 1

Class 2

blue or red?

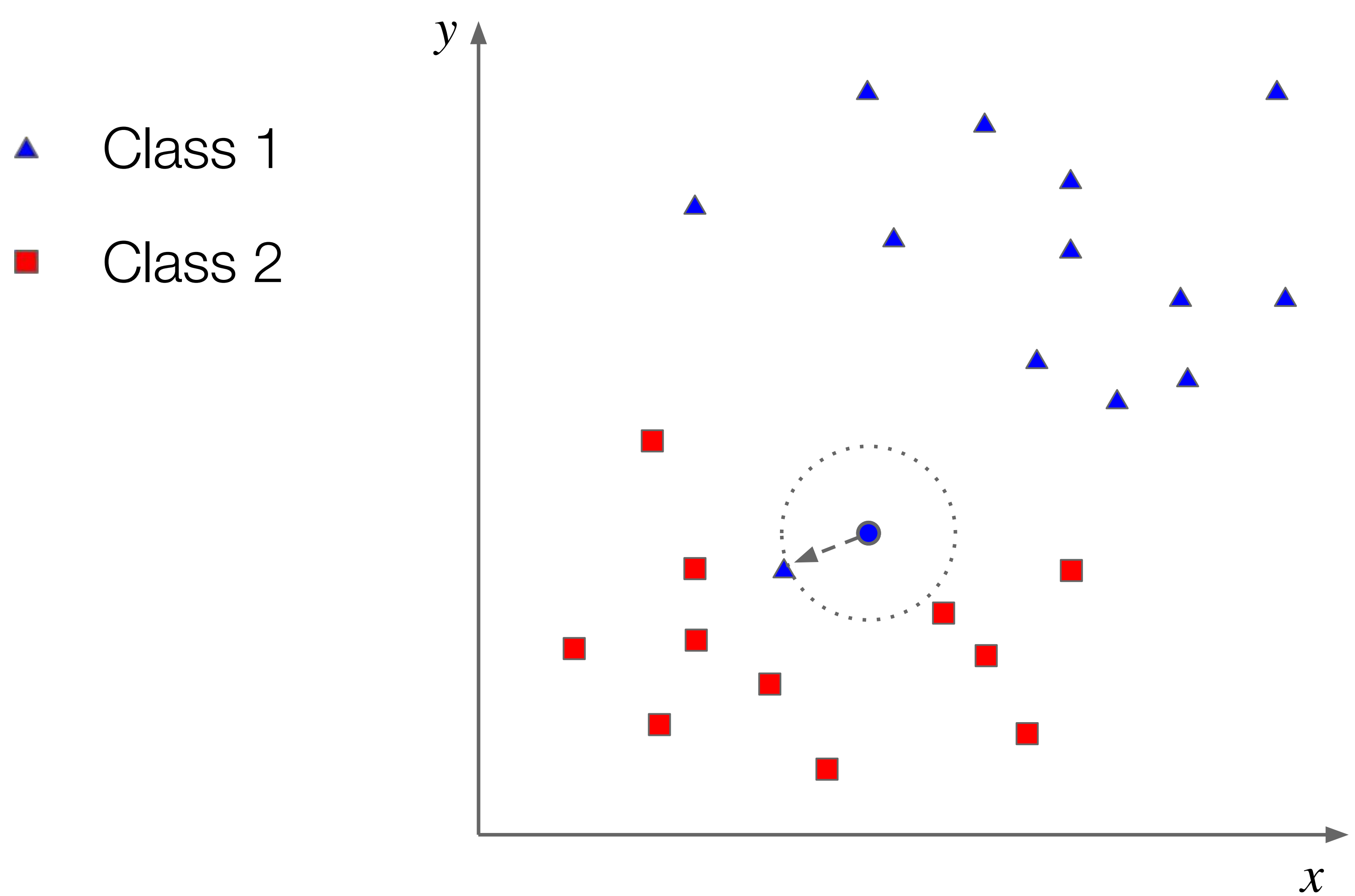


x_q	y_q
9	11

x_q	y_q	d	x_q	y_q	d
9	2	9.0	16	11	7.0
5	3	8.9	14	12	5.1
14	3	9.4	18	12	9.1
8	4	7.1	18	14	9.5
3	5	8.5	20	14	11.4
6	5	6.7	11	15	4.5
13	5	7.2	15	15	7.2
12	6	5.8	6	16	5.8
6	7	5.0	15	17	8.5
8	7	4.1	13	18	8.1
15	7	7.2	10	19	8.1
5	10	4.1	20	19	13.6
15	7		10	19	
5	10		20	19	

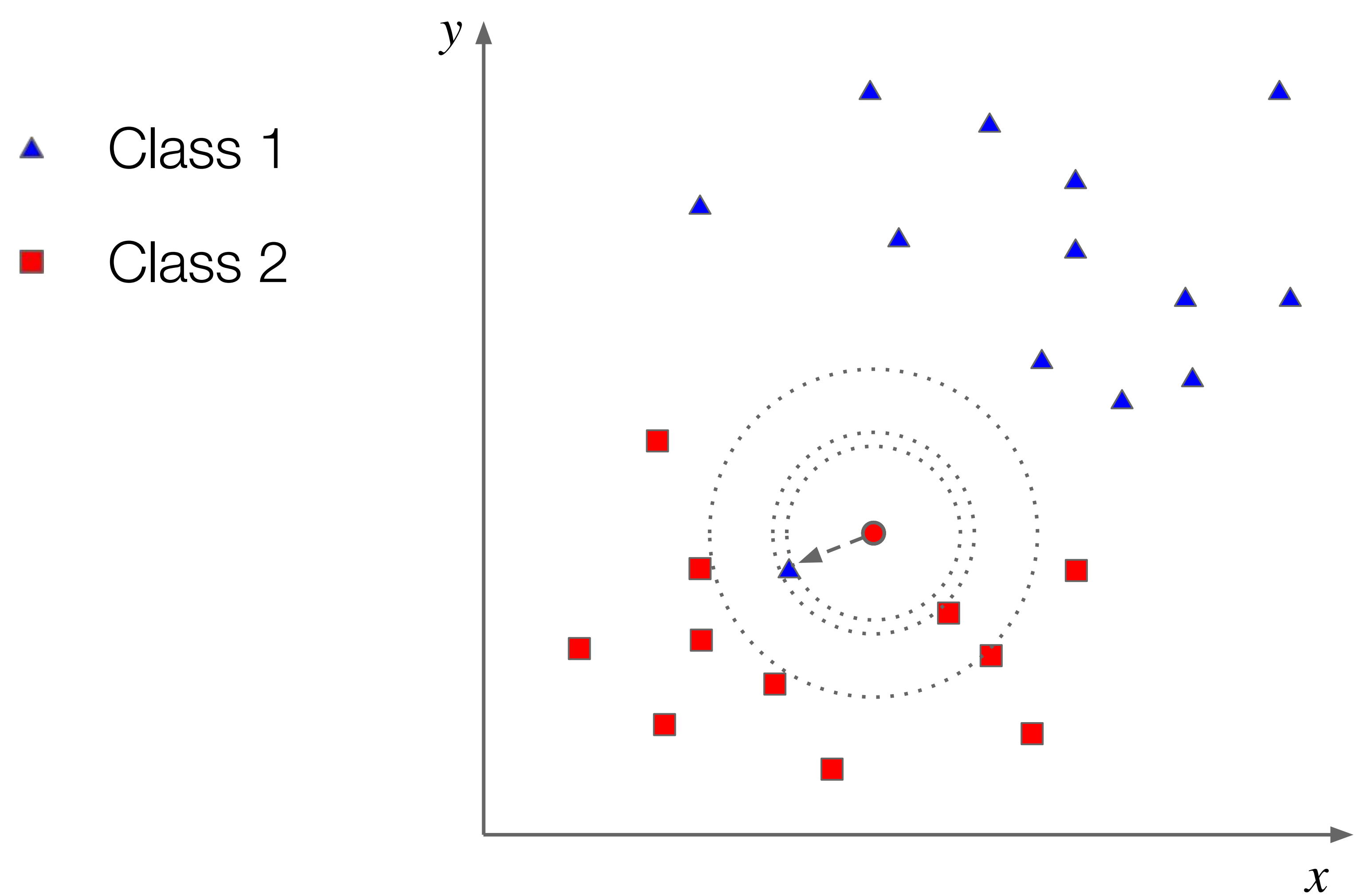
Nearest Neighbor classifier

- NN is sensitive to the outliers!



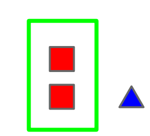
k-Nearest Neighbors classifier

- A “generalization”: from NN to k-NN



k=3
(a.k.a. 3-NN)

majority voting:



k-Nearest Neighbors

- Assume that your training examples corresponds to points in d-dimensional Euclidean space
  - Key idea: the value of the target function for a new sample is estimated from the known (stored) training examples
    - This is done by computing distances between the new sample and all the training samples
    - Decision rule: assign the label of the majority class among the k nearest neighbors

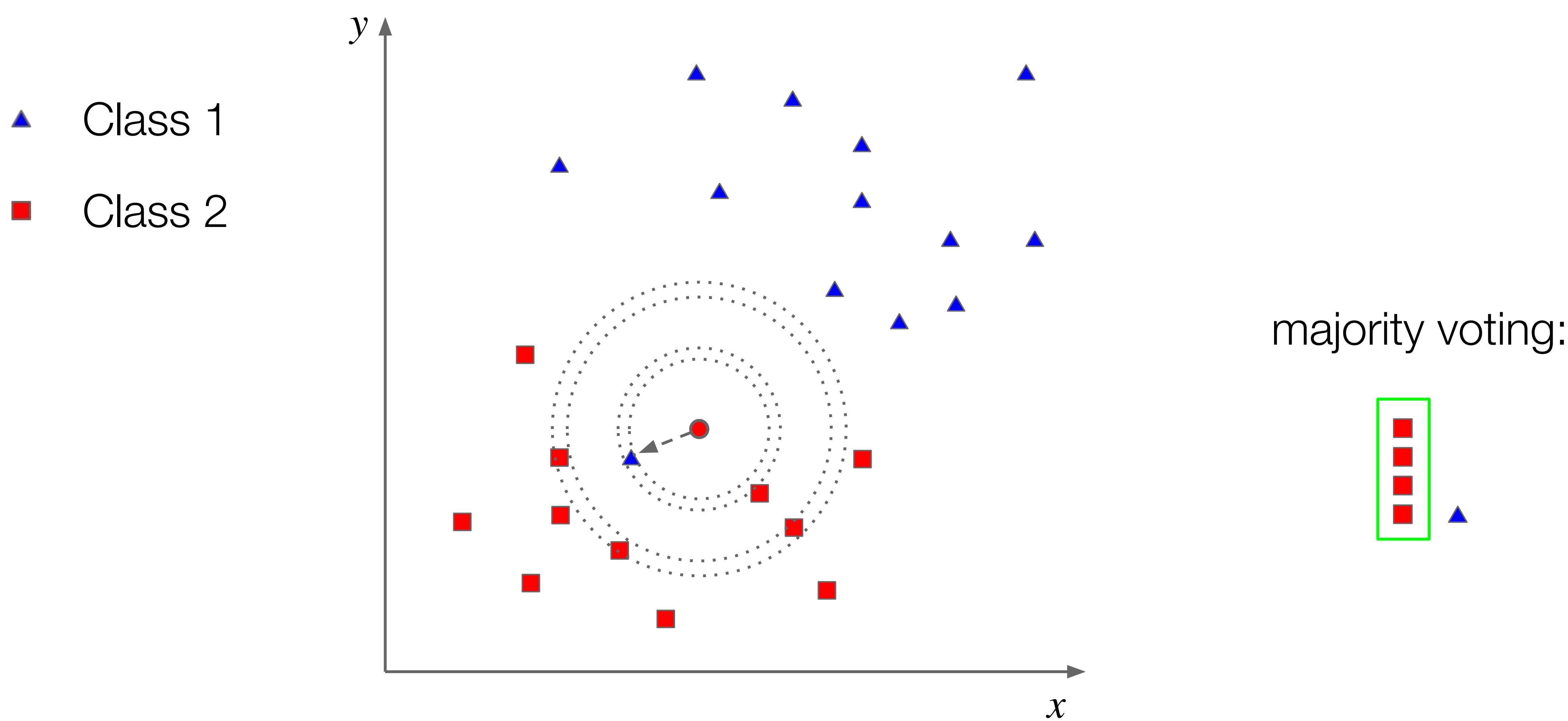
Algorithm:

Find k examples $(x^{(i)}, y^{(i)})$ (from training set) closest to the test sample x

Output: $y = \text{argmax}_{y^{(z)}} \sum_{j=1}^{k} \delta(y^{(z)}, y^{(j)})$

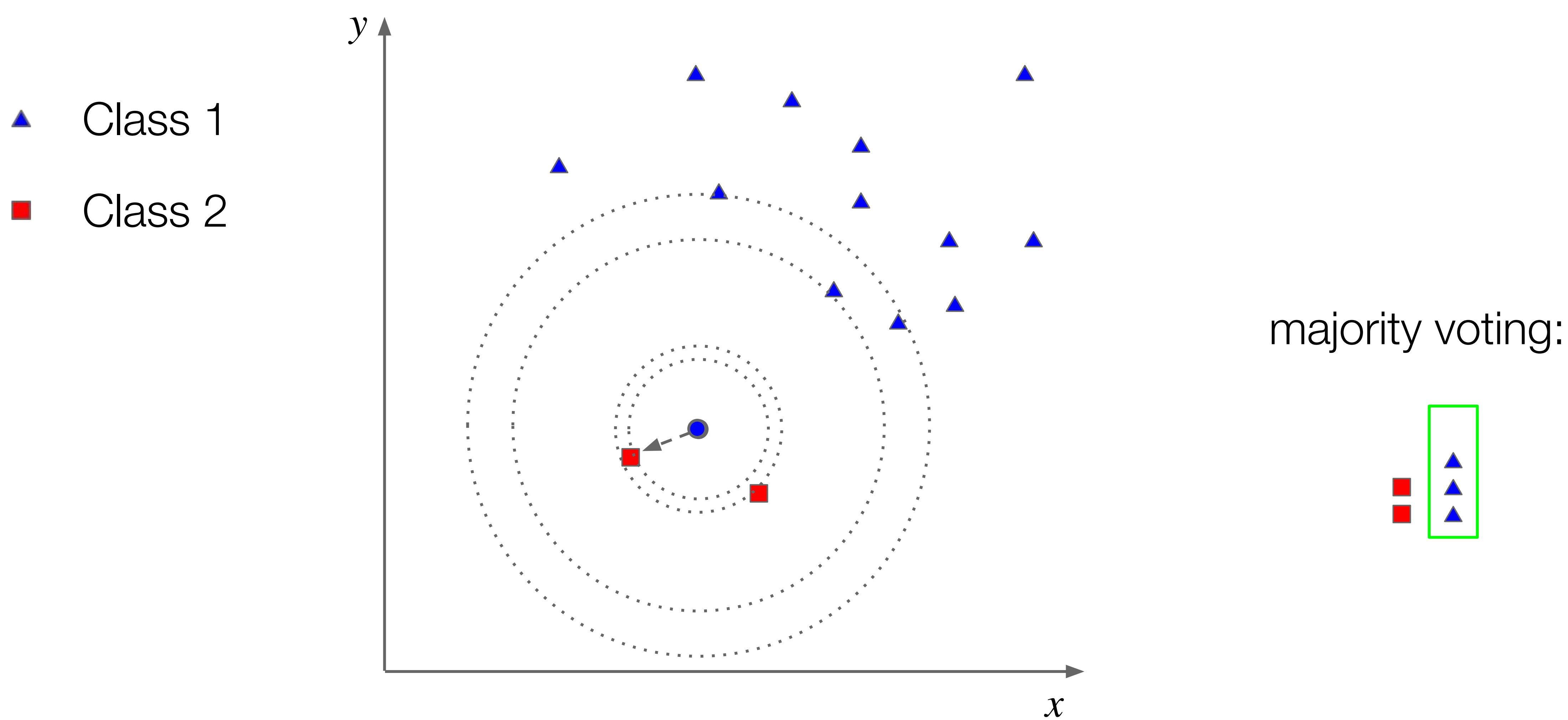
k-Nearest Neighbors classifier

- Small k: sensitive to outliers



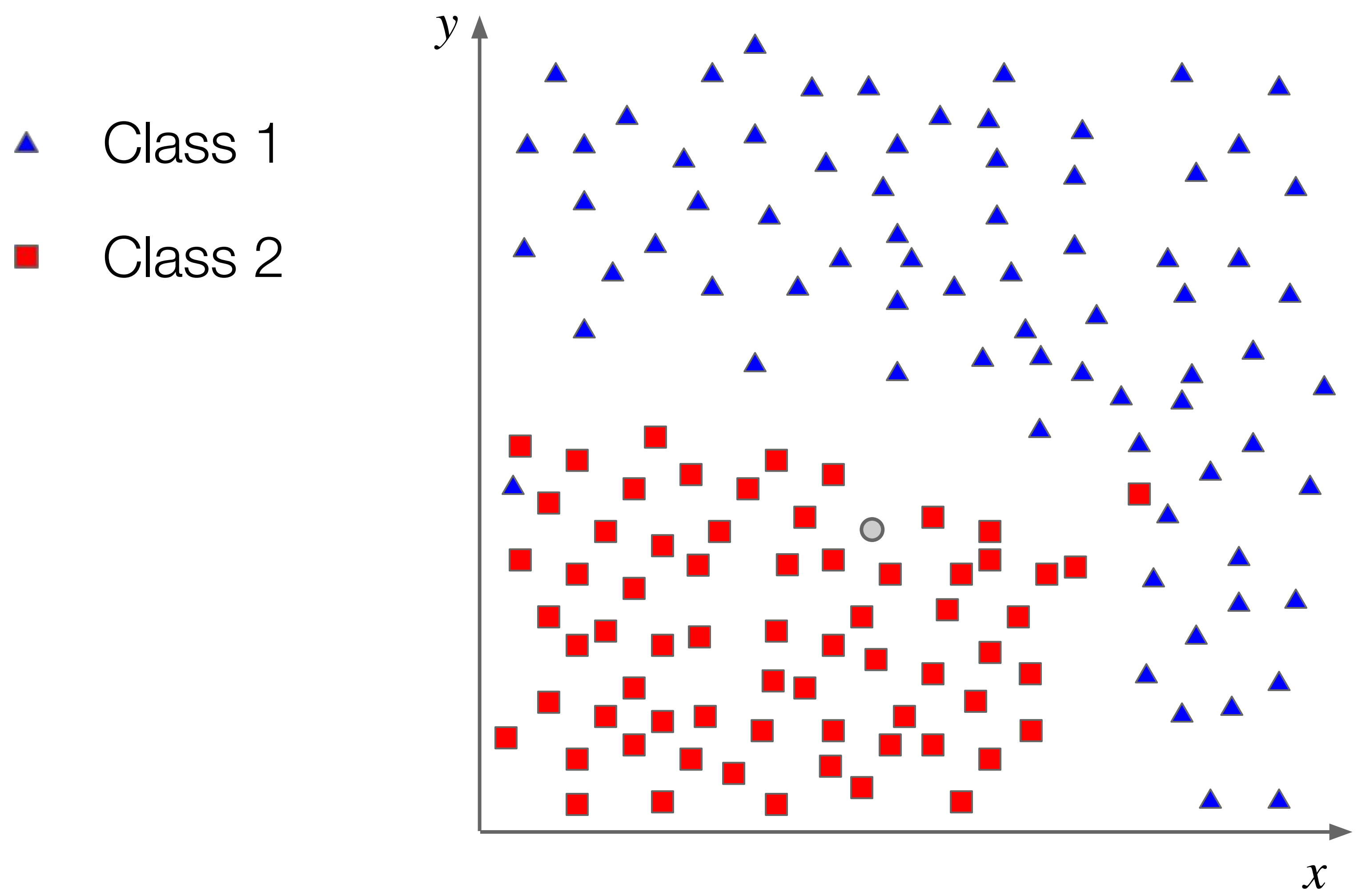
k-Nearest Neighbors classifier

- Parameter k has a very strong effect
  - Large k: everything is classified as the most frequent class



k-Nearest Neighbors classifier

the more data, the better! .. we might have issues with areas not well covered



k-Nearest Neighbors classifier

- k-NN recipe: how do we choose k?
  - Large k may lead to better performance (if the training set is sufficiently large)
  - If we pick k too large we may end up looking at examples that are not “real” neighbors (are far away the test sample)

Hyperparameter Selection

- For knn, k (the number of nearest neighbours) is an **hyperparameter**
- Different hyperparameters will affect how model generalizes over unknown data points
- How do we calculate the optimal hyperparameter?
  - The process is called **model selection**
    - we will see how we can do model validation in the next set of slides

Limitations of KNN

- Think about the inference stage, what do we need to compute the classification?

Limitations of KNN

- Think about the inference stage, what do we need to compute the classification?
  - The **entire training set**
    - it means that every time we want to infer the class of a data point
    - We need to compute its distance for each training point
    - Computationally expensive at testing time!!

Limitations of KNN

- What about the impact of the features?

Limitations of KNN

- What about the impact of the features?
  - In the exemple, we used height and width, which are expressed with the same unit measure
  - What if it was height, width, and weight? how do you compare cm and grams?
- A possible solution is called feature scaling
  - e.g., Each numerical feature is transformed in a range [0, 1]
  - e.g., have each feature with mean = 0 and variance = 1

$$z = \frac{x - \mu}{\sigma}$$

k-NN Summary

- k-NN naturally forms complex decision boundaries; it adapts to data density
- If we have lots of samples, k-NN typically works well
- Main limitations/problems:
  - Sensitive to class noise and scales of features (attributes)
  - Distances are less meaningful in high dimensions
  - Scales linearly with number of training examples: i.e. it is extremely expensive at test time

Restriction Bias

- How we Limit the Hypothesis Space
- KNN does not explicitly restrict the hypothesis space to a predefined set of functions like parametric models (e.g., linear regression). However, it implicitly limits the hypothesis space by assuming that:
  - The possible hypotheses are constrained to those that assign labels based on local neighborhoods.
  - The function is **non-parametric**, meaning it does not assume any specific functional form (e.g., linear or polynomial).

Preference Bias

- How we Order the Hypothesis Space
    - As we do not learn a real algorithm, we do not assume explicitly an “order” of the hypotheses

- KNN makes implicit assumptions about the data distribution and prioritizes certain hypotheses over others:
    - Locality assumption: Closer points are more relevant for classification or regression.
    - Majority rule: In classification, the label is determined by the majority vote within the neighborhood, preferring piecewise constant decision boundaries (region-based).
    - Distance metric matters: The preference is heavily influenced by the choice of the distance metric (e.g., Euclidean, Manhattan).