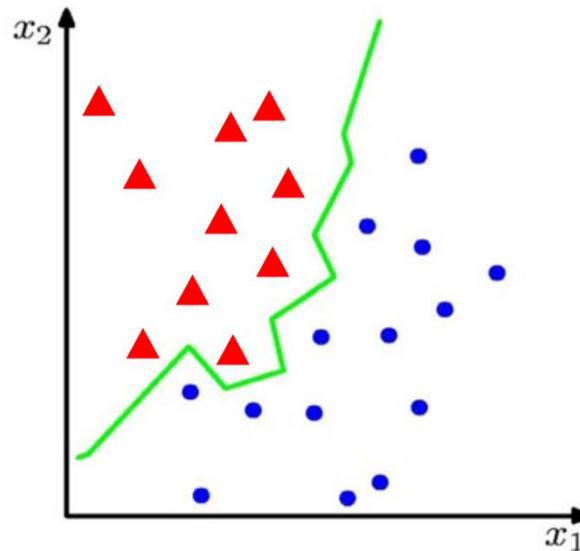


# k-Nearest Neighbors. Local Learning. Curse of Dimensionality.

Radu Ionescu, Prof. PhD.  
[raducu.ionescu@gmail.com](mailto:raducu.ionescu@gmail.com)

Faculty of Mathematics and Computer Science  
University of Bucharest

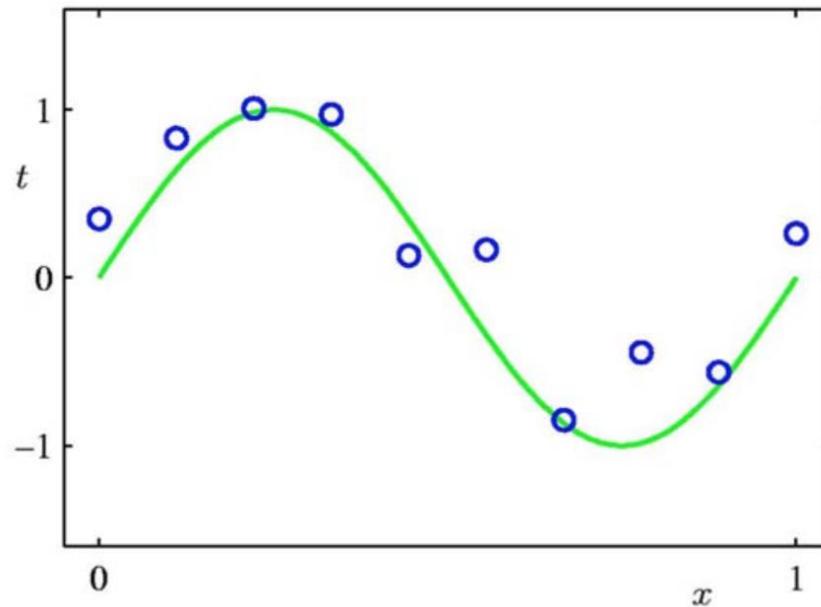
# Classification from labeled samples



- Suppose we have a set of  $N$  training examples:  
 $(x_1, \dots, x_N)$  and  $(y_1, \dots, y_N)$ ,  $x_i \in \mathbb{R}^d$ ,  $y_i \in \{-1, 1\}$
- The classification problem consists of estimating the function  $g(x)$  such that:

$$g(x_i) = y_i$$

# Regression from labeled samples



- Suppose we have a set of  $N$  training examples:  
 $(x_1, \dots, x_N)$  and  $(y_1, \dots, y_N)$ ,  $x_i, y_i \in \mathbb{R}$
- The regression problem consists of estimating the function  $g(x)$  such that:

$$g(x_i) = y_i$$

# Learning from labeled samples

- Suppose we have a set of  $N$  training examples:  
 $(x_1, \dots, x_N)$  and  $(y_1, \dots, y_N)$ ,  $x_i, y_i \in \mathbb{R}$
- The learning problem consists of estimating  $g(x)$  such that:

$$g(x_i) = y_i$$

- The loss function (e.g. MSE):

$$\mathcal{L}(y, g(\mathbf{x}))$$

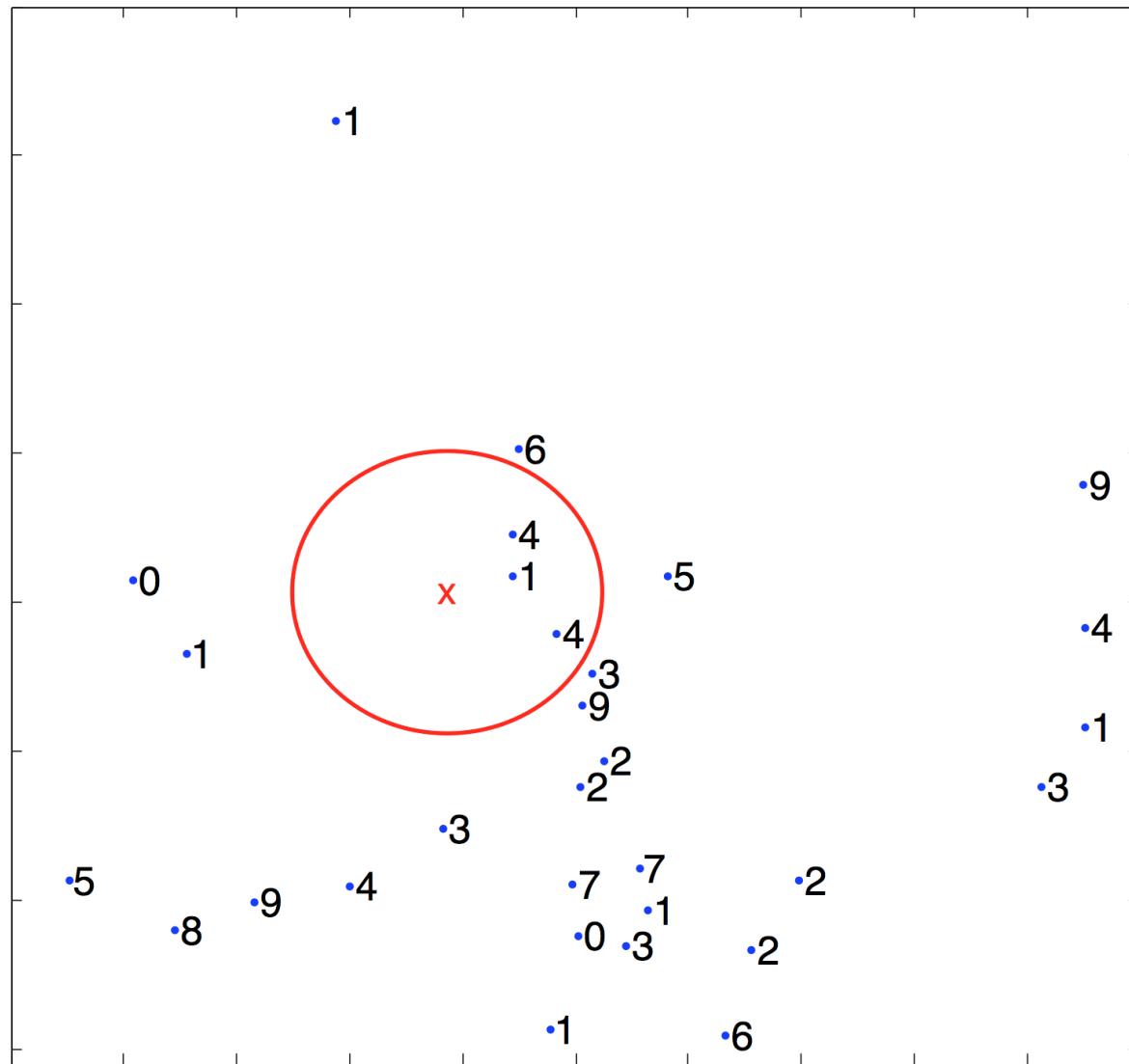
- Generalization error:

$$L(g) = E_P \mathcal{L}(y, g(\mathbf{x})) = \int \mathcal{L}(y, g(\mathbf{x})) dP(\mathbf{x}, y)$$

- Empirical (estimated) error:

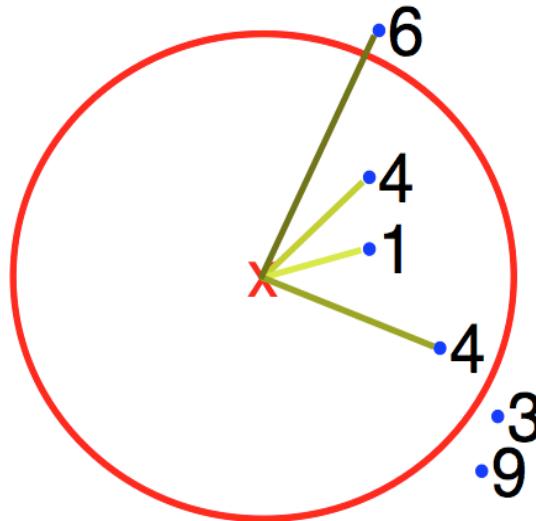
$$L_e(g) = \frac{1}{\ell} \sum_{i=1}^{\ell} \mathcal{L}(y_i, g(\mathbf{x}_i))$$

# What is the label for sample x?



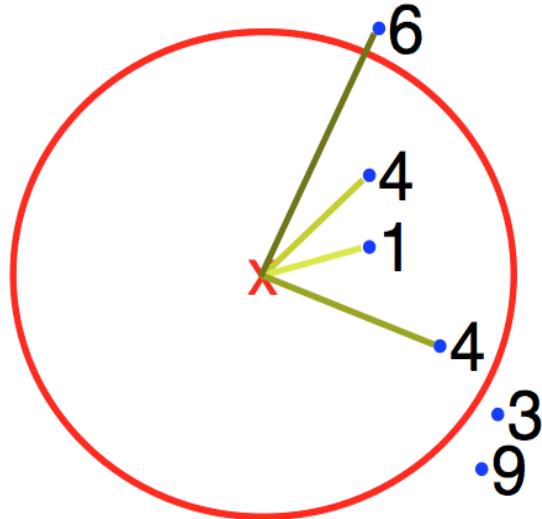
# k-Nearest Neighbors

# k-Nearest Neighbors (k-NN)



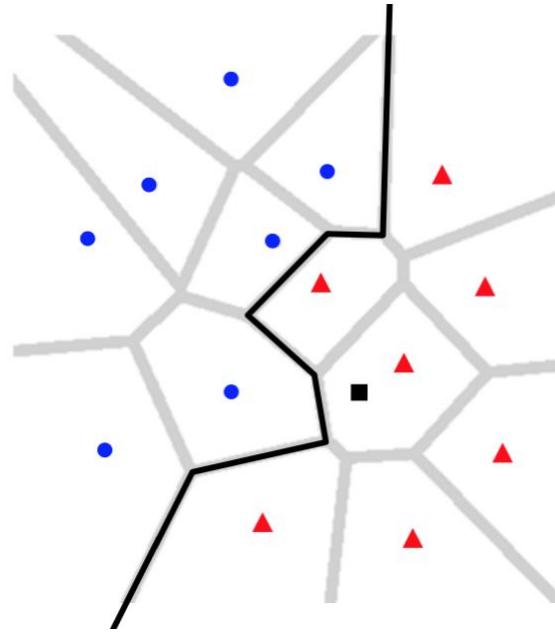
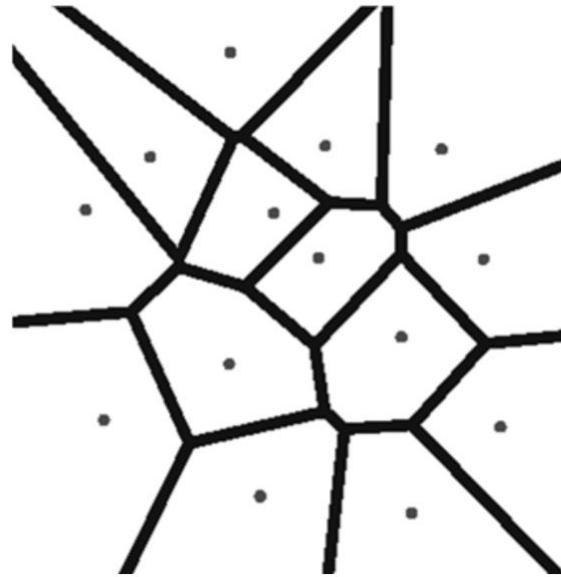
- The k-NN algorithm:
  - 1) For test sample  $x$ , we find the  $k$  nearest neighbors
  - 2) We label the test sample  $x$  using a plurality voting on  $k$  neighbors

# k-Nearest Neighbors (k-NN)



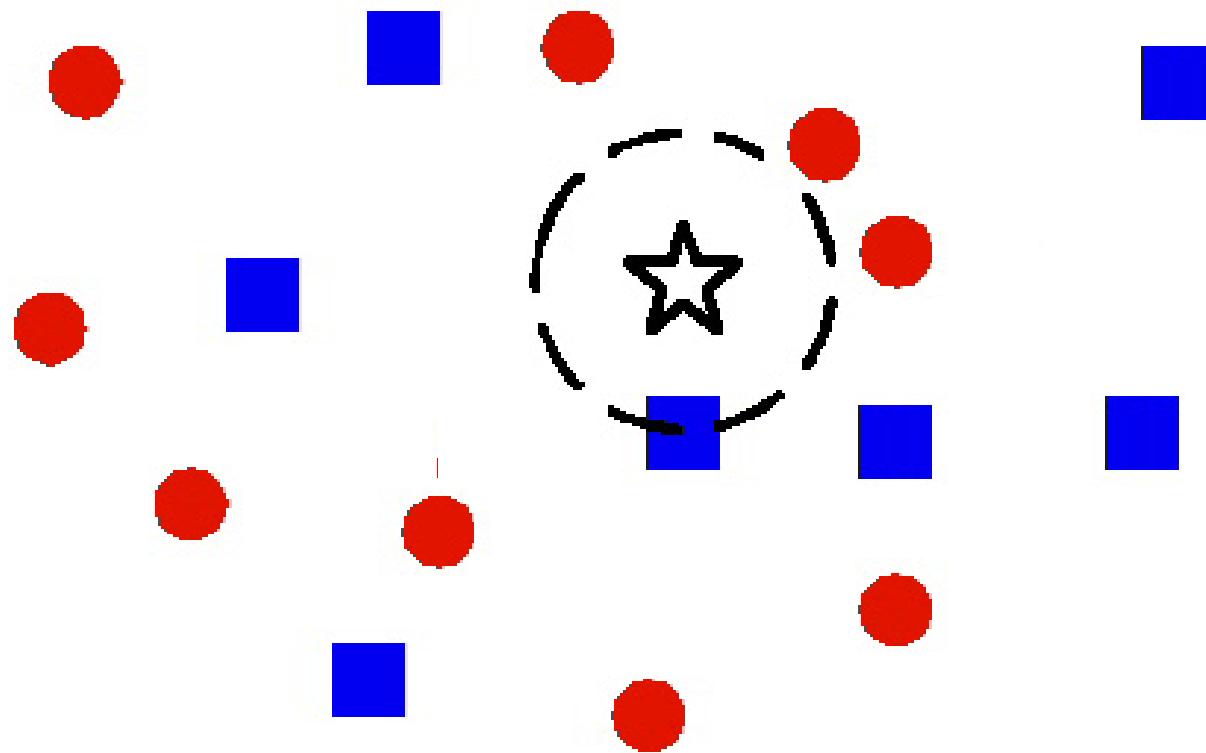
- How do we break ties?
  - 1) Choose a random label from the tied ones
  - 2) Apply 1-NN (no ties possible)
  - 3) Use the distances to the test sample as weights

# What happens for $k = 1$ ?

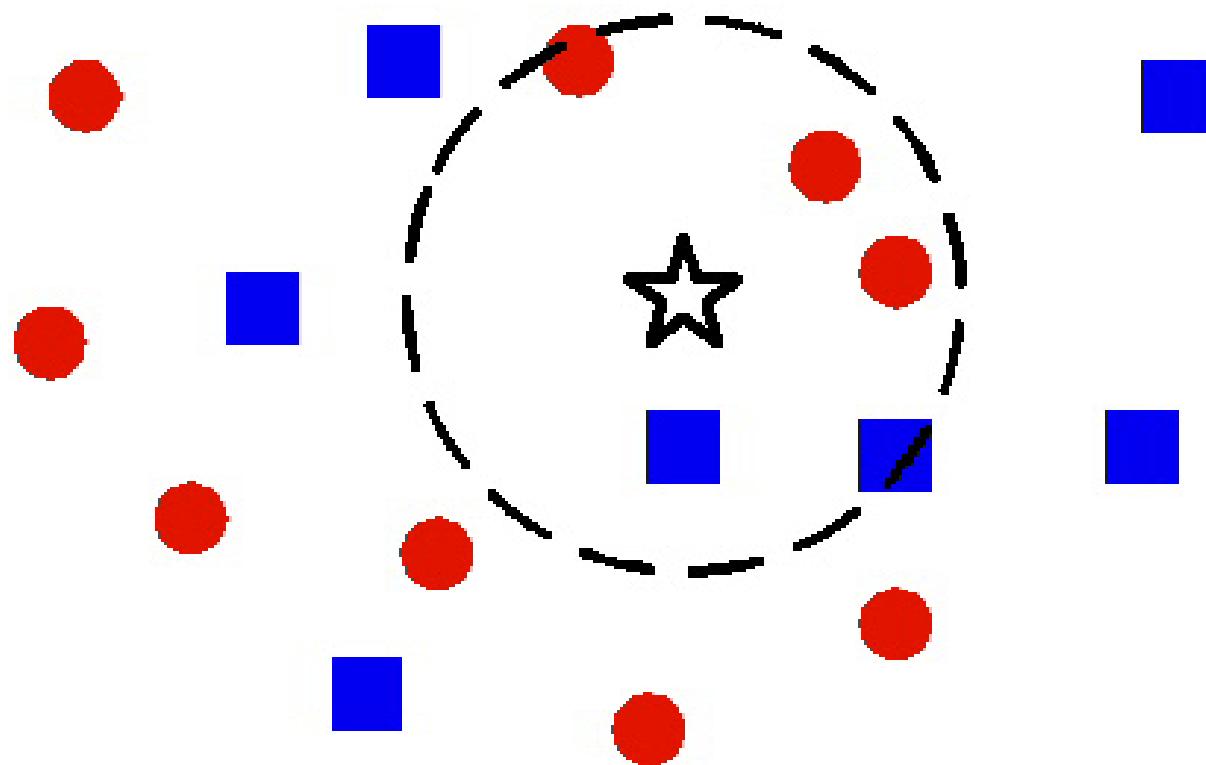


- We obtain a Voronoi diagram:
  - The space is partitioned into regions
  - The separating borders pass through areas where the distances between training sample pairs are equal
- The separating borders are nonlinear

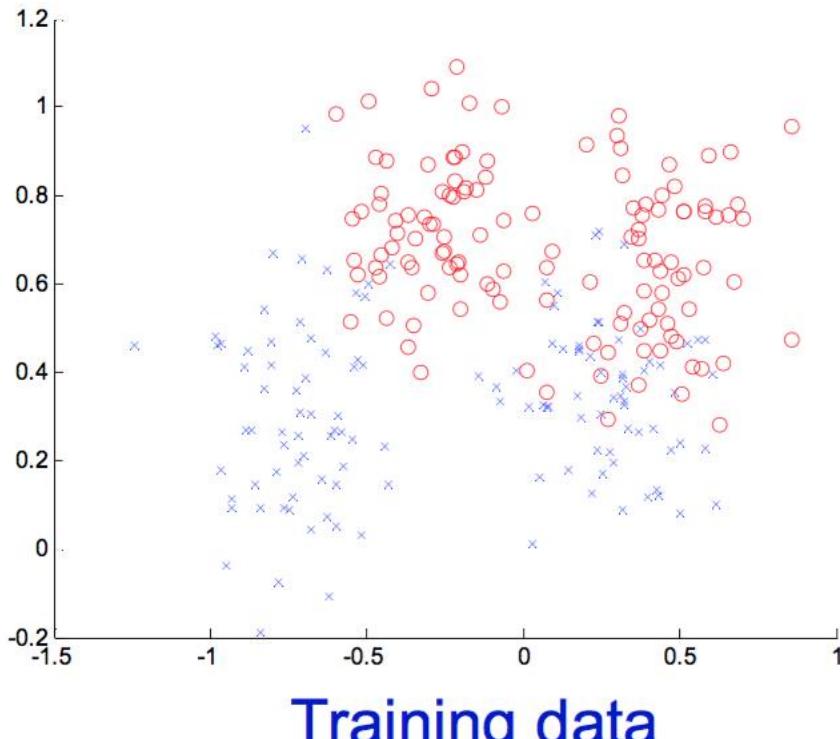
# 1-NN versus k-NN



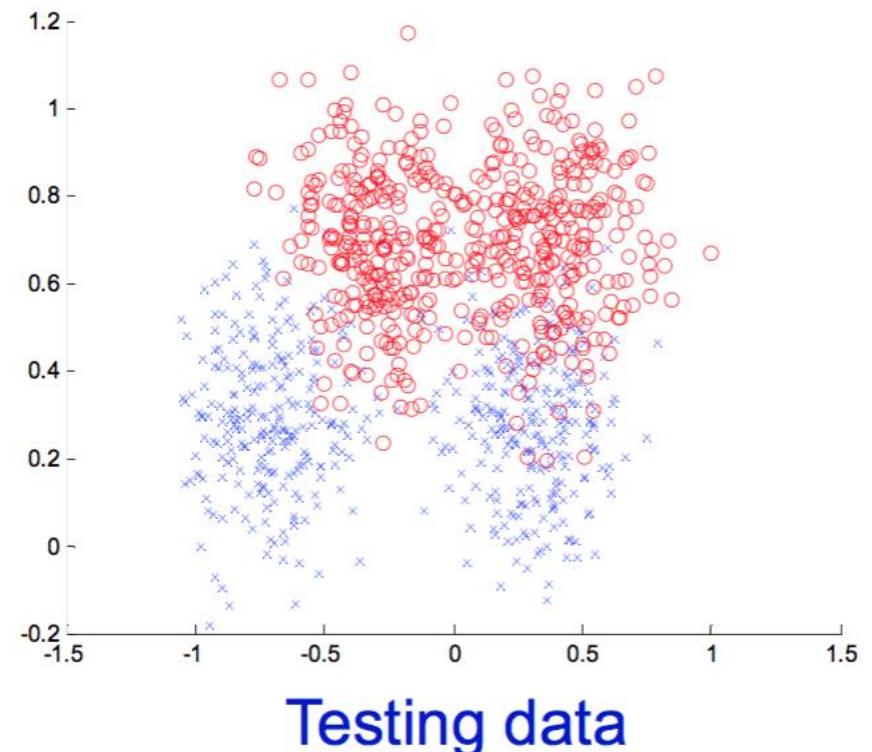
# 1-NN versus k-NN



# The underlying hypothesis of k-NN



Training data

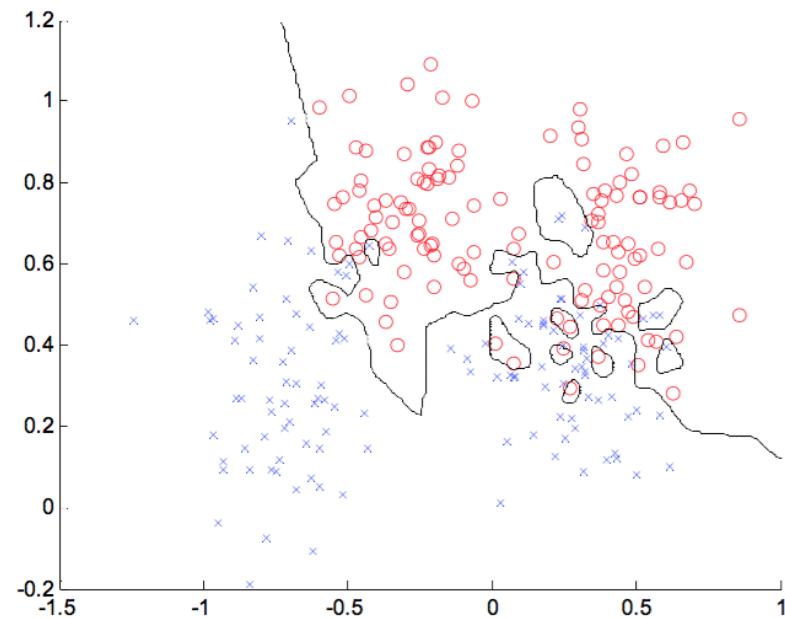


Testing data

- The training data and the test data are sampled from the same distribution
- Becomes unlikely for a representative pattern in the training set to be absent in the test set

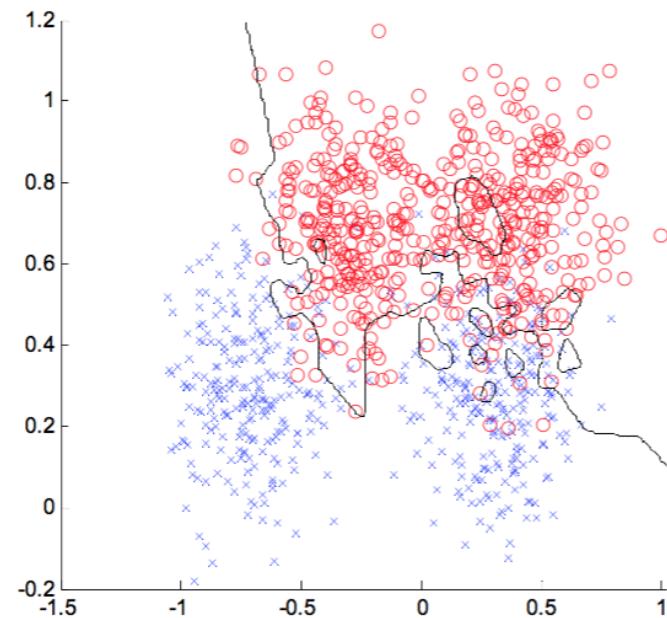
# What happens for different values of k?

Training data



error = 0.0

Testing data

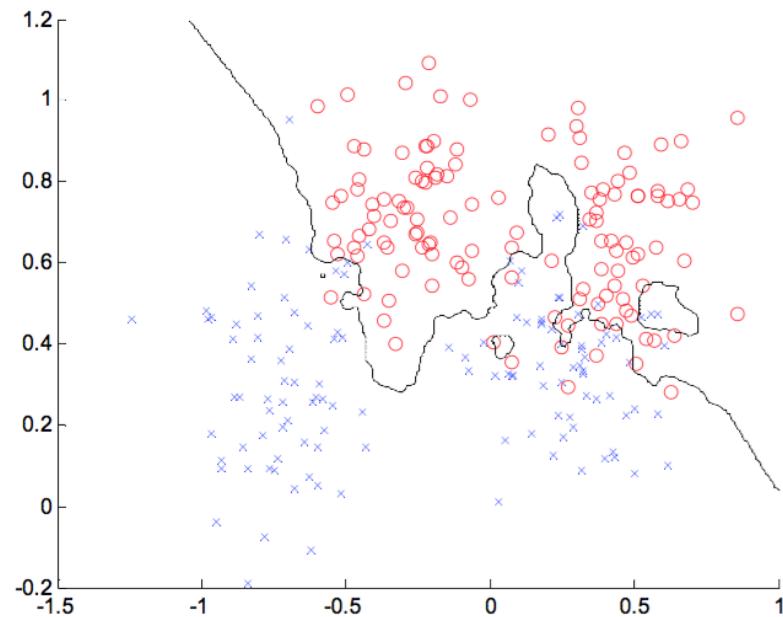


error = 0.15

- $k = 1$

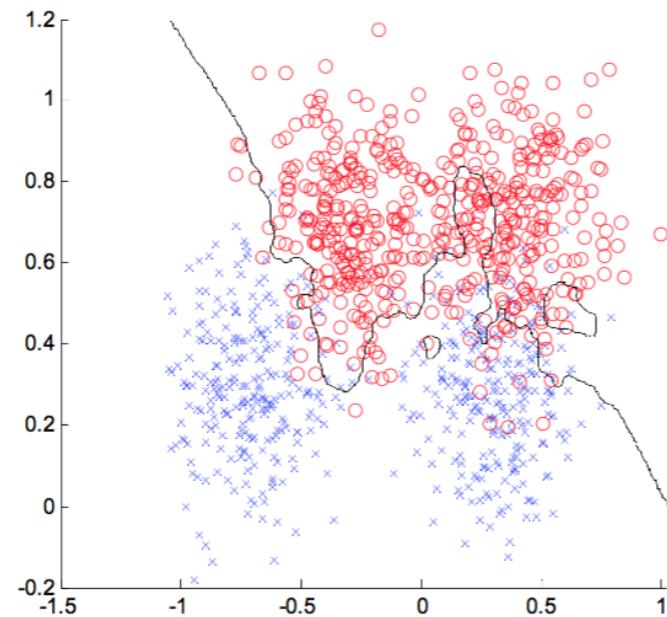
# What happens for different values of k?

Training data



error = 0.0760

Testing data

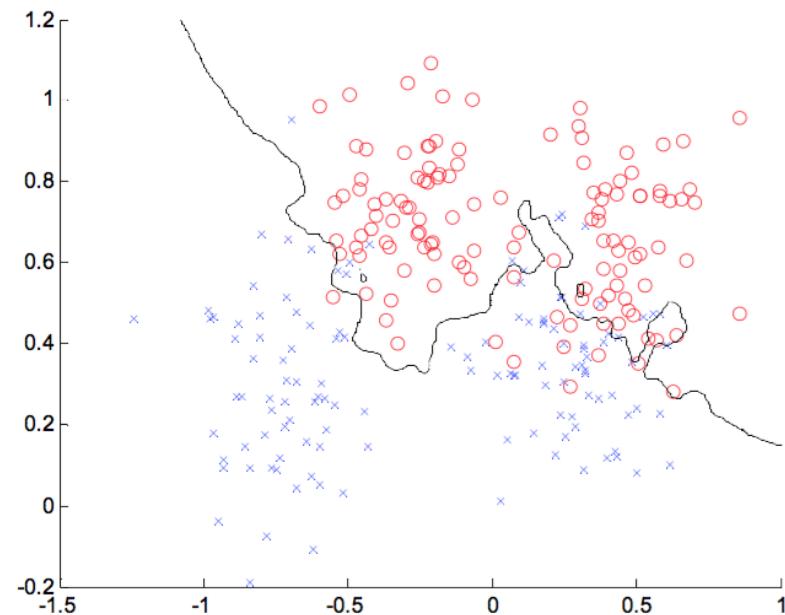


error = 0.1340

- $k = 3$

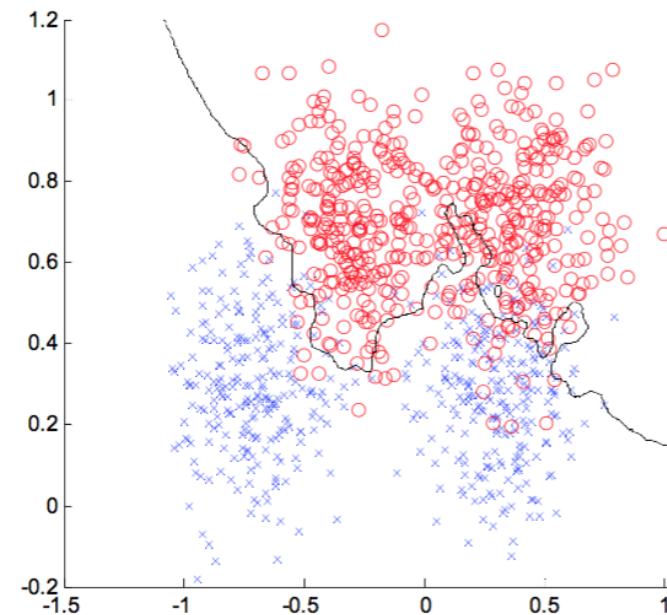
# What happens for different values of k?

Training data



error = 0.1320

Testing data

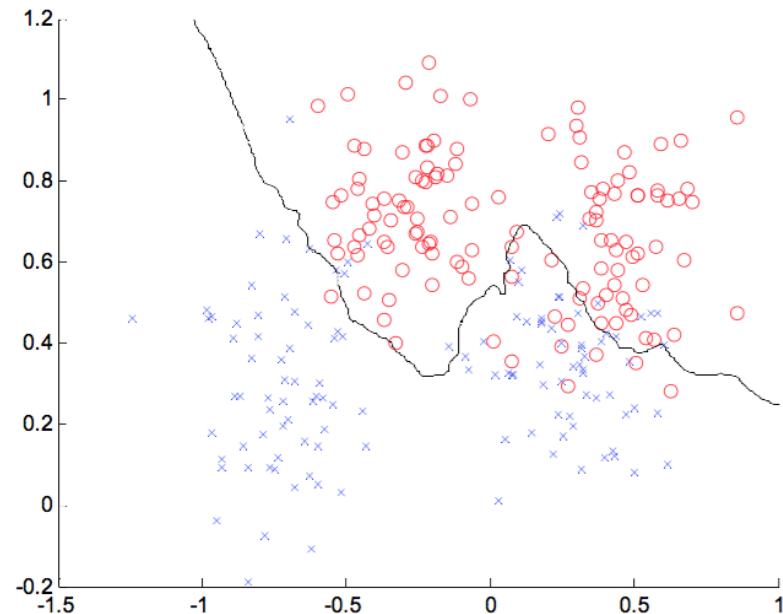


error = 0.1110

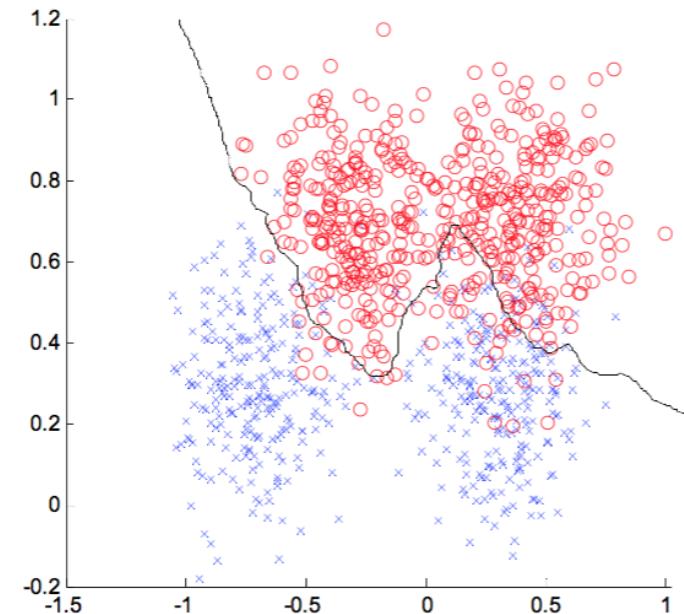
- $k = 7$

# What happens for different values of k?

Training data



Testing data



error = 0.1120

error = 0.0920

- $k = 21$

# What do we need for a memory-based classifier?

- A distance function:
  - **The Euclidean distance**
  - **The Edit (Levenstein) distance**
  - **The Hamming distance**
- How many neighbors should we consider?
- How do we “train” the model on the examples from the vicinity?

# Let's consider a particular 1-NN

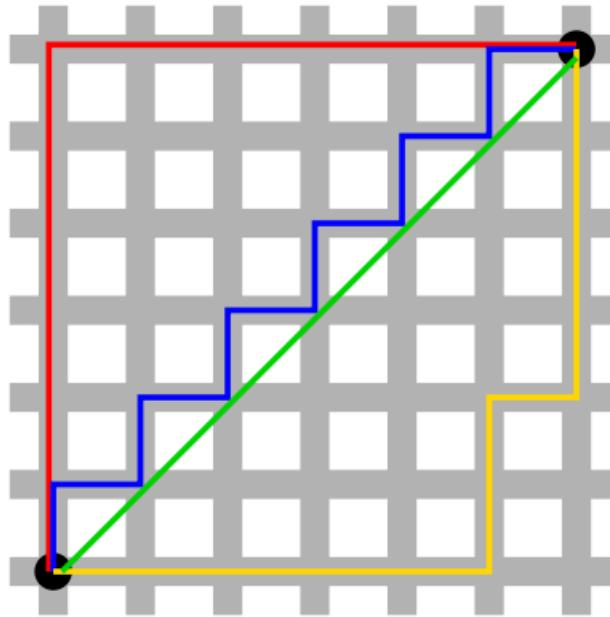
- A distance function:
  - **E.g., the Euclidean distance**
- How many neighbors should we consider?
  - 1
- How do we “train” the model on the examples from the vicinity?
  - **We just predict the label of the nearest neighbor**

# The Euclidean distance ( $L_2$ )

- For the vectors  $x = (5, 1, 3, 0)$  and  $y = (2, 1, 4, 1)$ , we have:

$$\begin{aligned}d_{L_2}(x, y) &= \sqrt{(x_1 - y_1)^2 + \cdots + (x_n - y_n)^2} \\&= \sqrt{(5 - 2)^2 + (1 - 1)^2 + (3 - 4)^2 + (0 - 1)^2} \\&= \sqrt{9 + 1 + 1} = \sqrt{11} \\&\cong 3.32\end{aligned}$$

# The Manhattan distance ( $L_1$ )



- For the vectors  $x = (5, 1, 3, 0)$  and  $y = (2, 1, 4, 1)$ , we have:

$$\begin{aligned}d_{L_1}(x, y) &= |x_1 - y_1| + \cdots + |x_n - y_n| \\&= |5 - 2| + |1 - 1| + |3 - 4| + |0 - 1| \\&= 3 + 1 + 1 = 5\end{aligned}$$

# The Minkowski distance ( $L_p$ )

- For the vectors  $x = (x_1, \dots, x_n)$  and  $y = (y_1, \dots, y_n)$ , we have:

$$d_{L_p}(x, y) = \sqrt[p]{|x_1 - y_1|^p + \dots + |x_n - y_n|^p}$$

- The Minkowski distance is a generalization for the Euclidean distance ( $p = 2$ ) and the Manhattan distance ( $p = 1$ )
- If  $p < 1$ , then  $d_{L_{p<1}}$  is no longer a distance. The triangle inequality is violated for  $x = (0,0)$ ,  $y = (1,1)$  and  $z = (0,1)$ :

$$d_{L_{p<1}}(x, y) > d_{L_{p<1}}(x, z) + d_{L_{p<1}}(z, y)$$

# The Hamming distance

- Useful, for instance, when the samples are represented by categorical features or when the samples are DNA sequences
- For the vectors  $x = (A, G, T, C)$  and  $y = (G, G, T, A)$ , we have:

$$d_{Hamming}(x, y) = 1 + 0 + 0 + 1 = 2$$

- We are counting how many features (components) are different among the two vectors

# The Edit (Levenstein) distance

- Useful, for instance, when the samples are strings (text documents, DNA sequences) or temporal sequences (videos)
- The distance is given by the number of changes (insert, delete, replace) necessary to transform one object into the other
- For video sequences, we use Dynamic Time Warping (DTW)

# Application: Gesture recognition

- What gesture corresponds to the motion performed by the user?



# Let's consider 10 gesture classes



# Gesture recognition problem

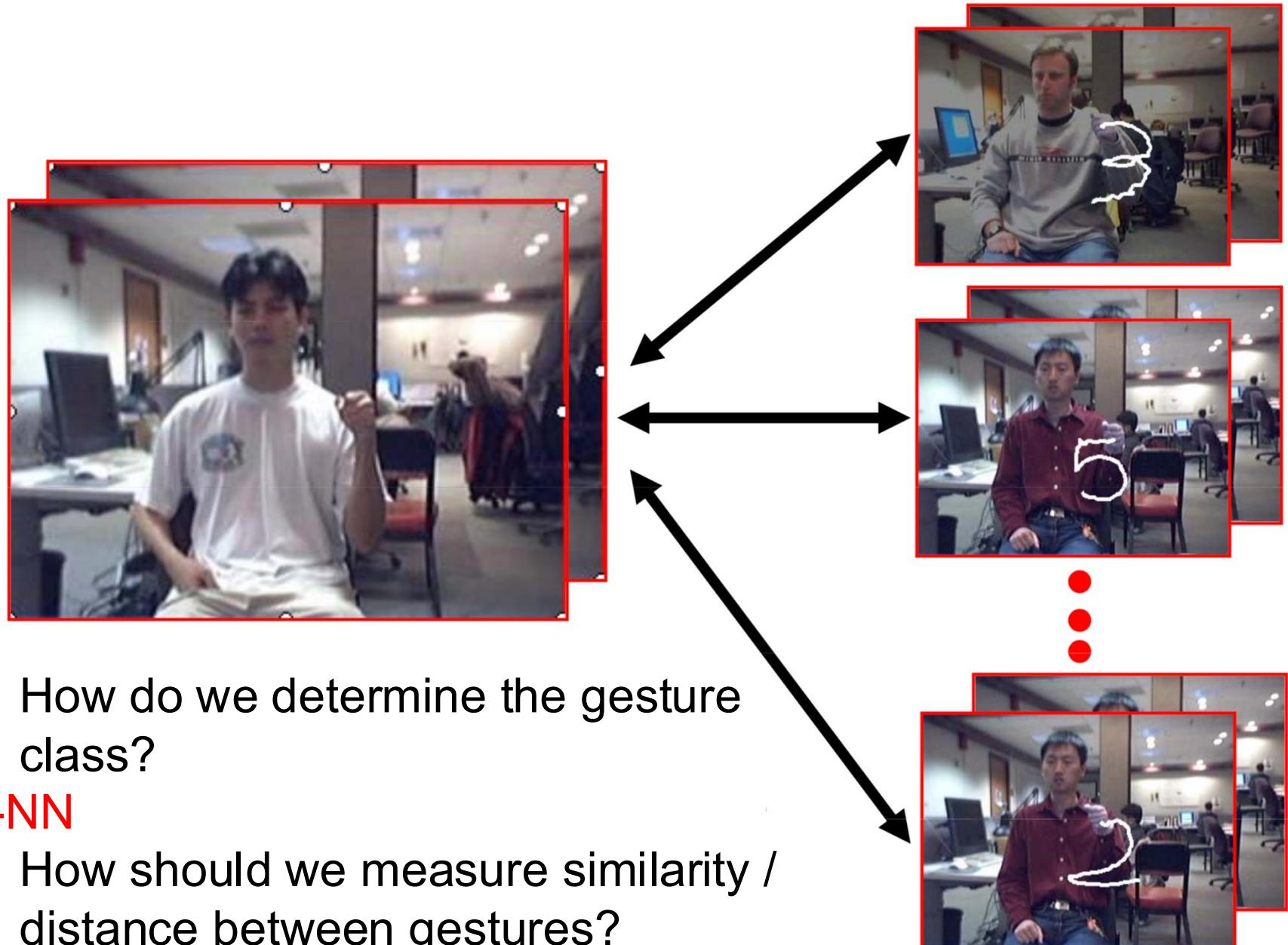
- We have to recognize 10 simple gestures performed by a user
- Each gesture corresponds to a number from 0 to 9
- Only the trajectory of the hand matters, not the handshape / fingers pose:
  - This is just a choice we make for this example application
  - Many systems, e.g. sign language, need to use handshape as well

# Decomposing gesture recognition

We need modules for:

- Computing how the person moved
  - Person detection / tracking
  - Hand detection / tracking
  - Handshape recognition?
- **Recognizing what the motion means**
  - Motion estimation and recognition are quite different tasks
  - When we see someone using sign language, we know how they move, but not what the motion means

# Gesture recognition

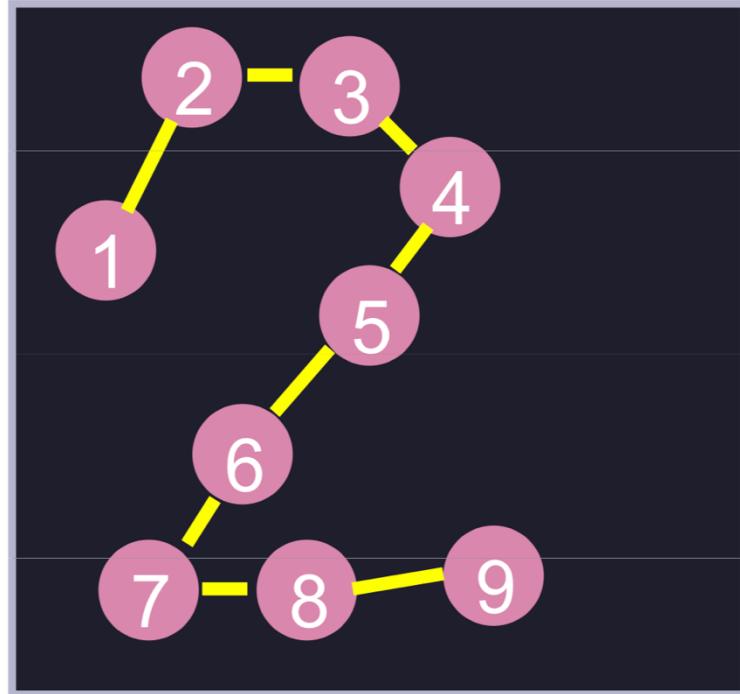
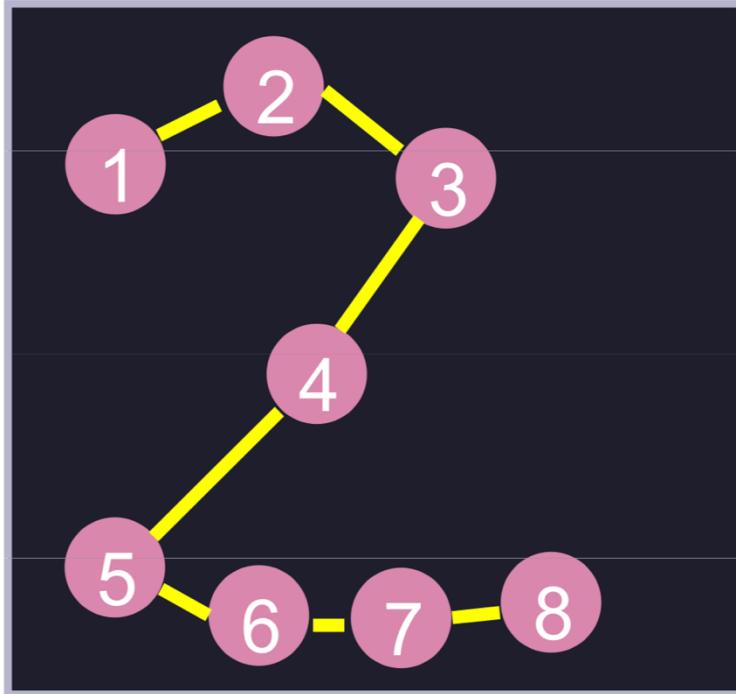


# If hand location is known



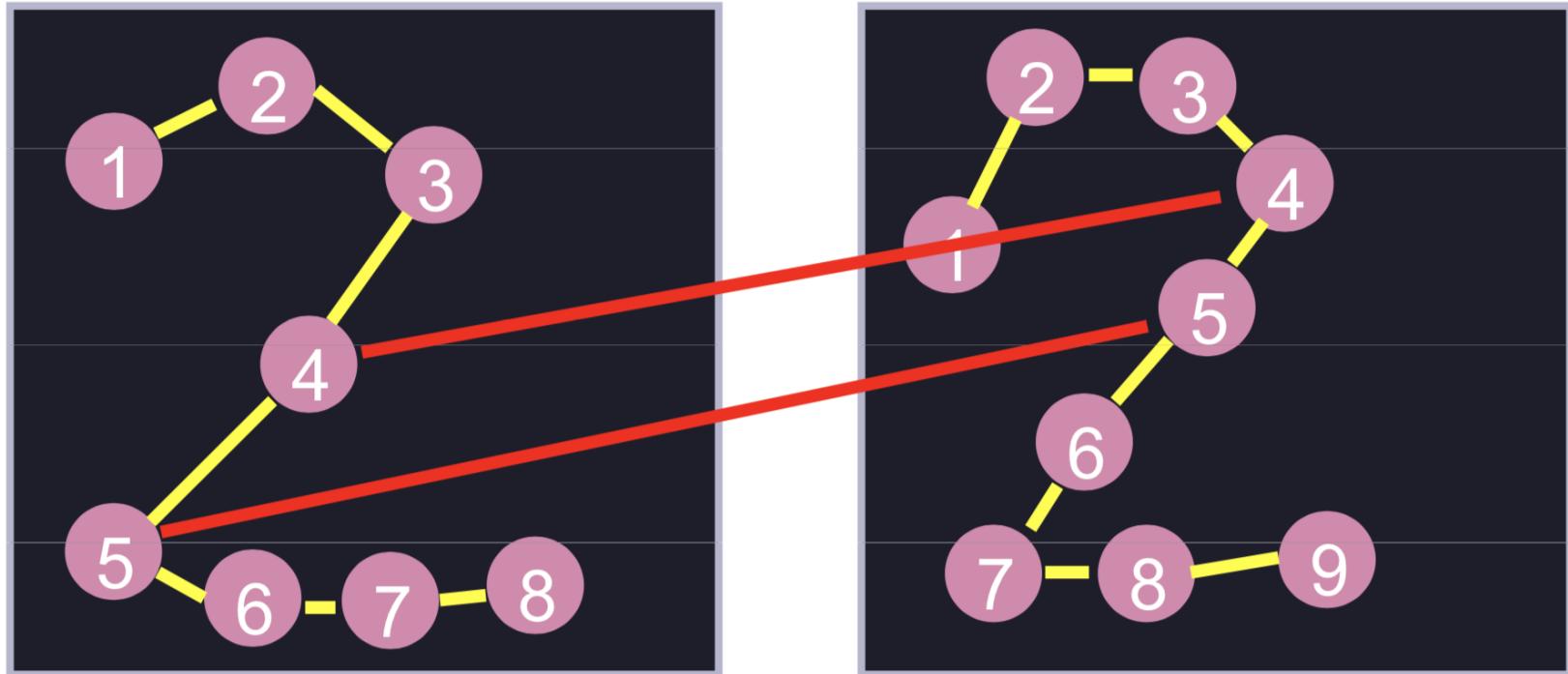
- Assumption: hand location is known in all frames of the database gestures
- The training data can be annotated offline using manual annotation / colored gloves / sensors
- For the test data, we need a hand detector  
(we do not cover this part)

# Comparing trajectories



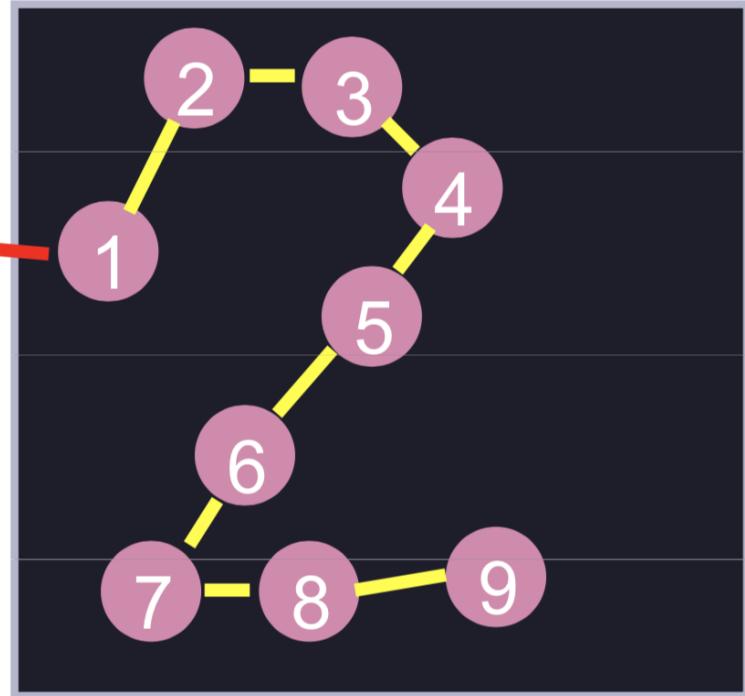
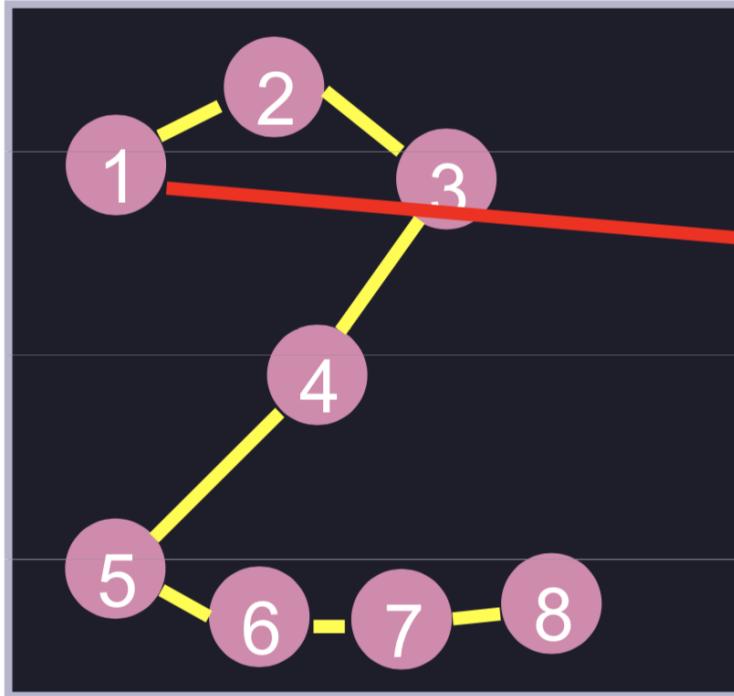
- We can make a trajectory based on the location of the hand at each frame
- How do we compare trajectories?

# Matching trajectories



- Comparing i-th frame to i-th frame is problematic
- What do we do with frame 9 from the right?

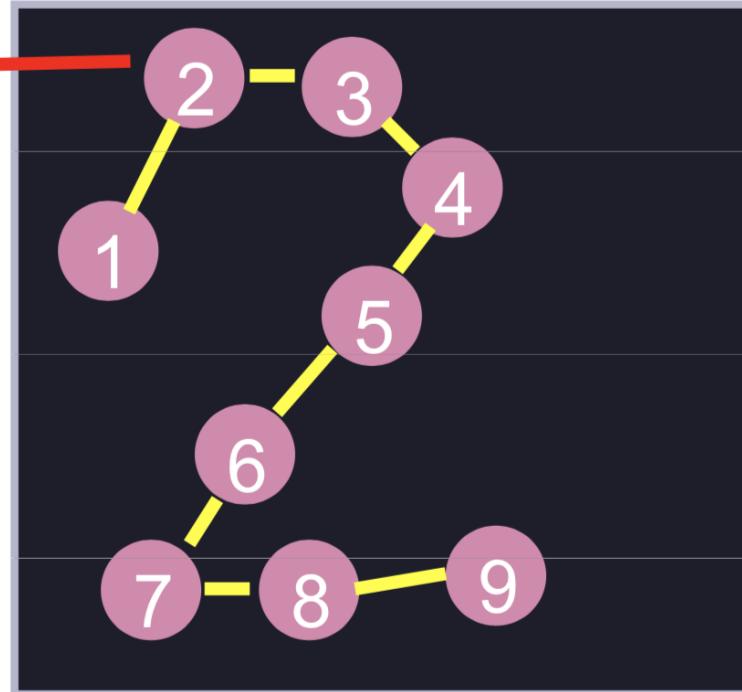
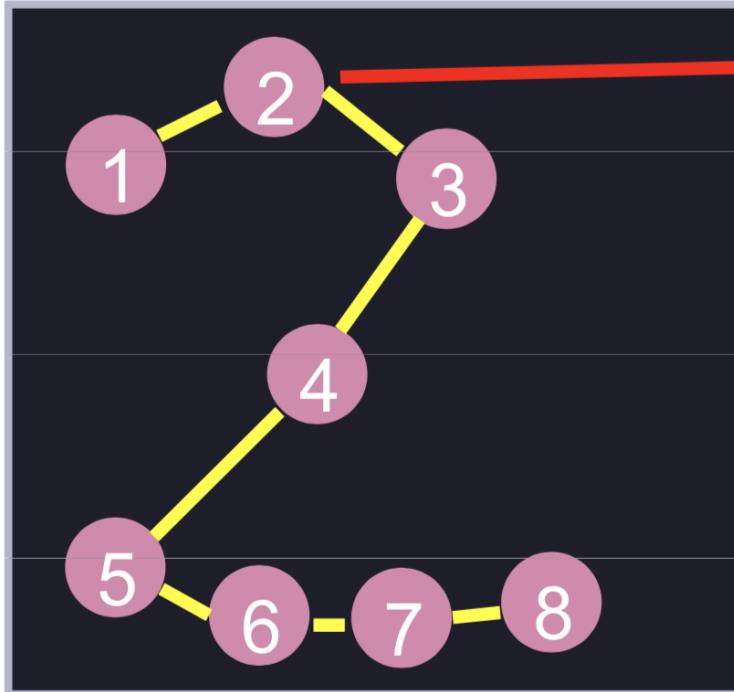
# Matching trajectories



- We can align them:

((1, 1)

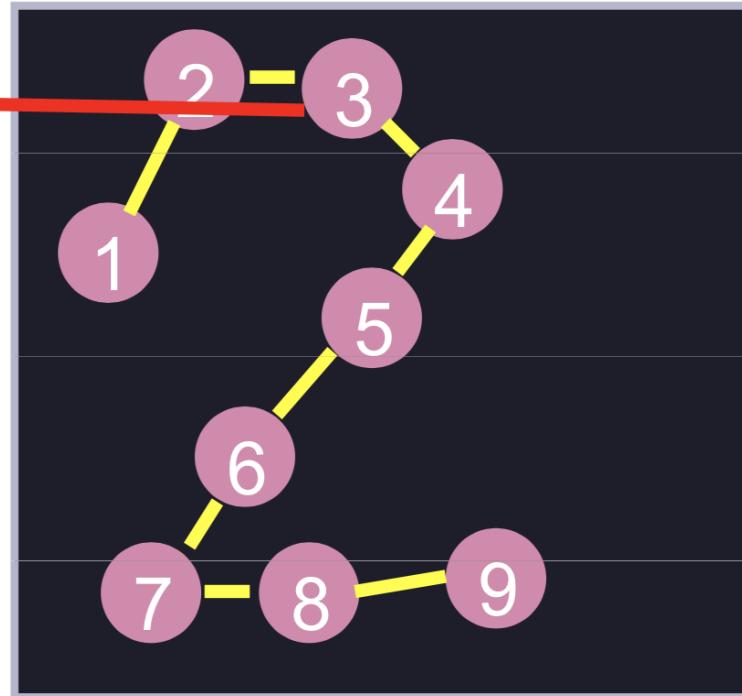
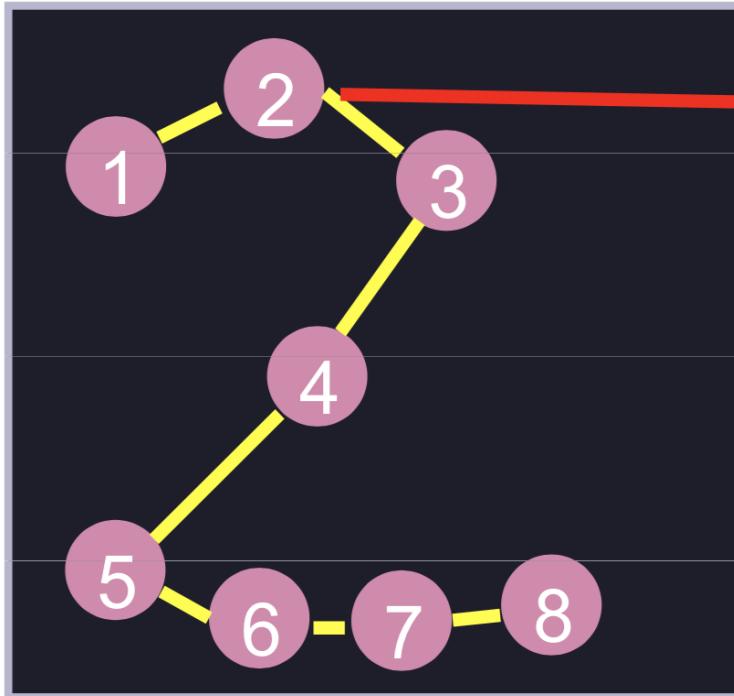
# Matching trajectories



- We can align them:

$((1, 1), (2, 2)$

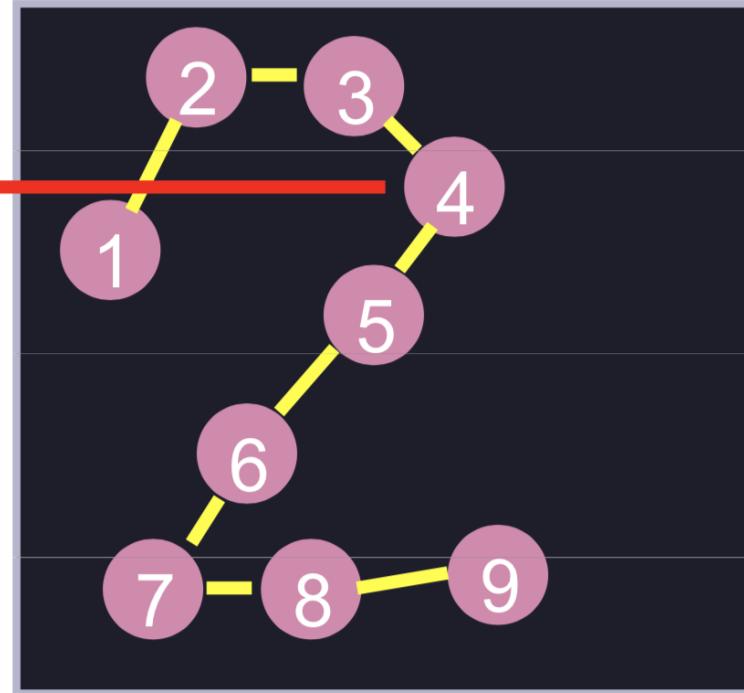
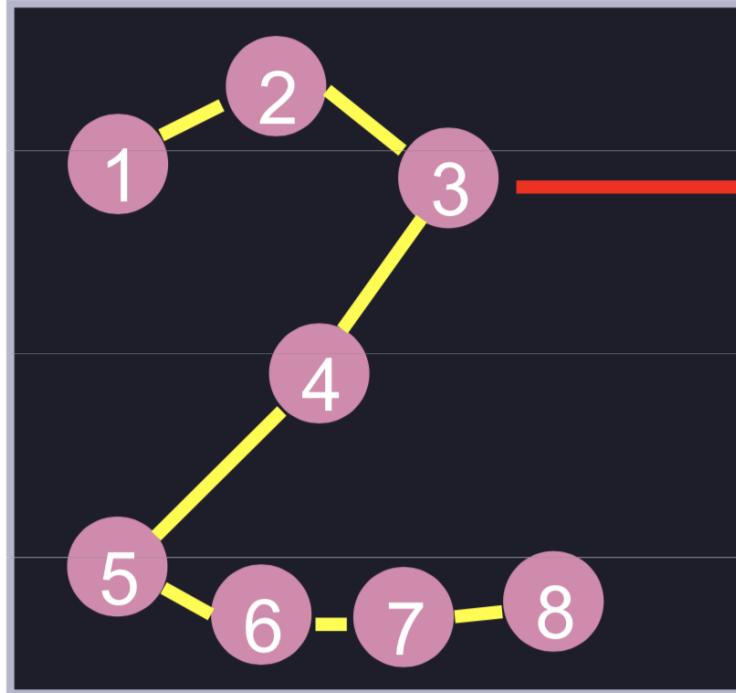
# Matching trajectories



- We can align them:

$((1, 1), (2, 2), \textcolor{red}{(2, 3)}$

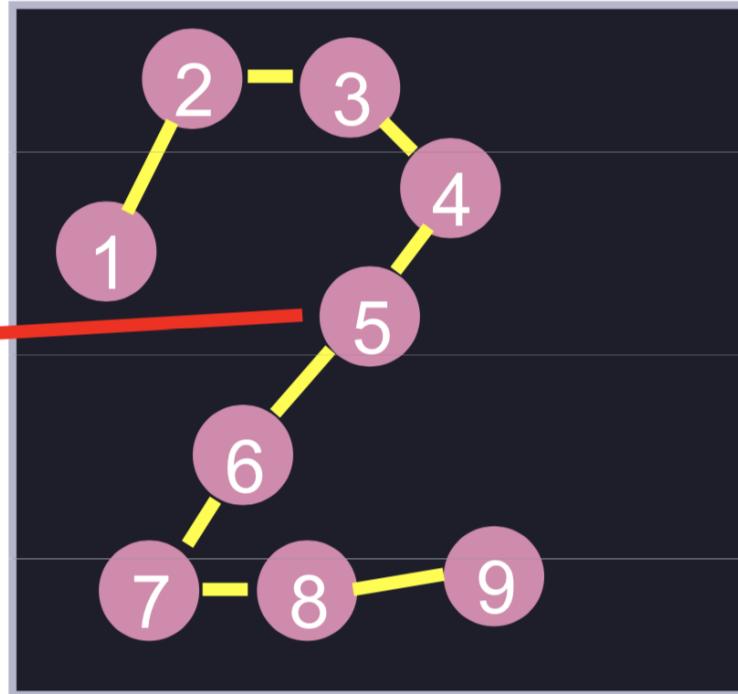
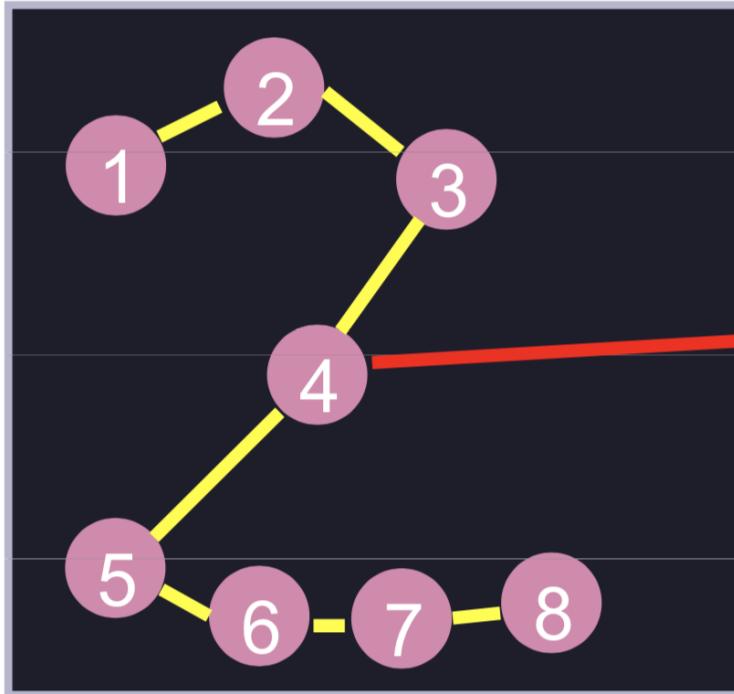
# Matching trajectories



- We can align them:

$((1, 1), (2, 2), (2, 3), (3, 4)$

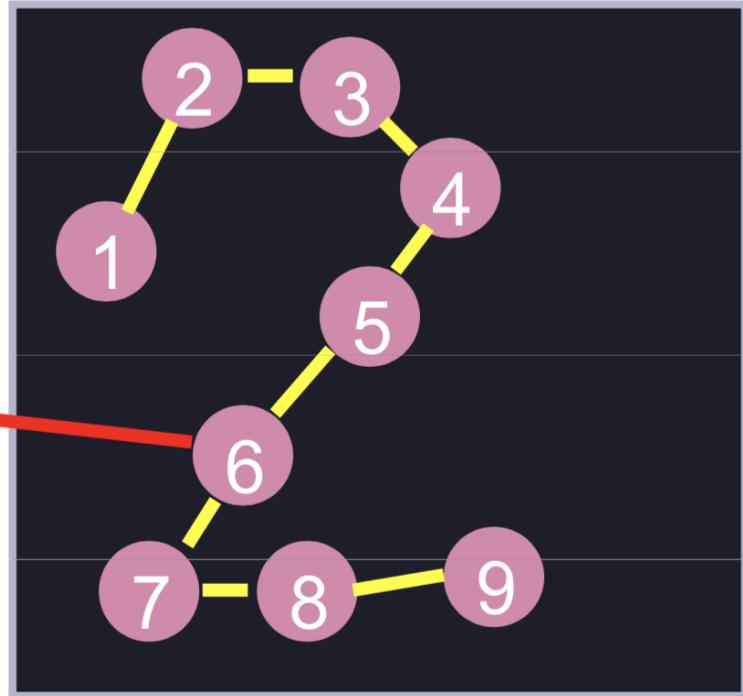
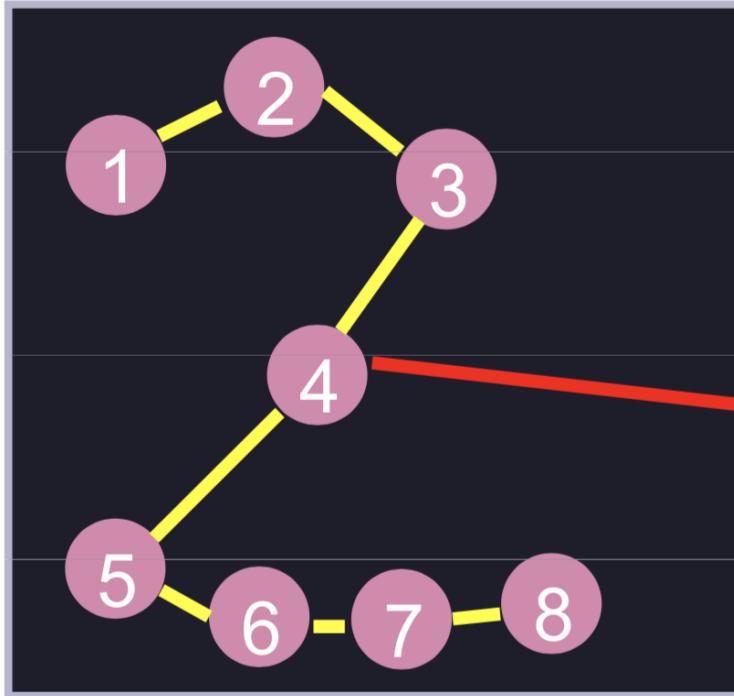
# Matching trajectories



- We can align them:

$((1, 1), (2, 2), (2, 3), (3, 4), (4, 5)$

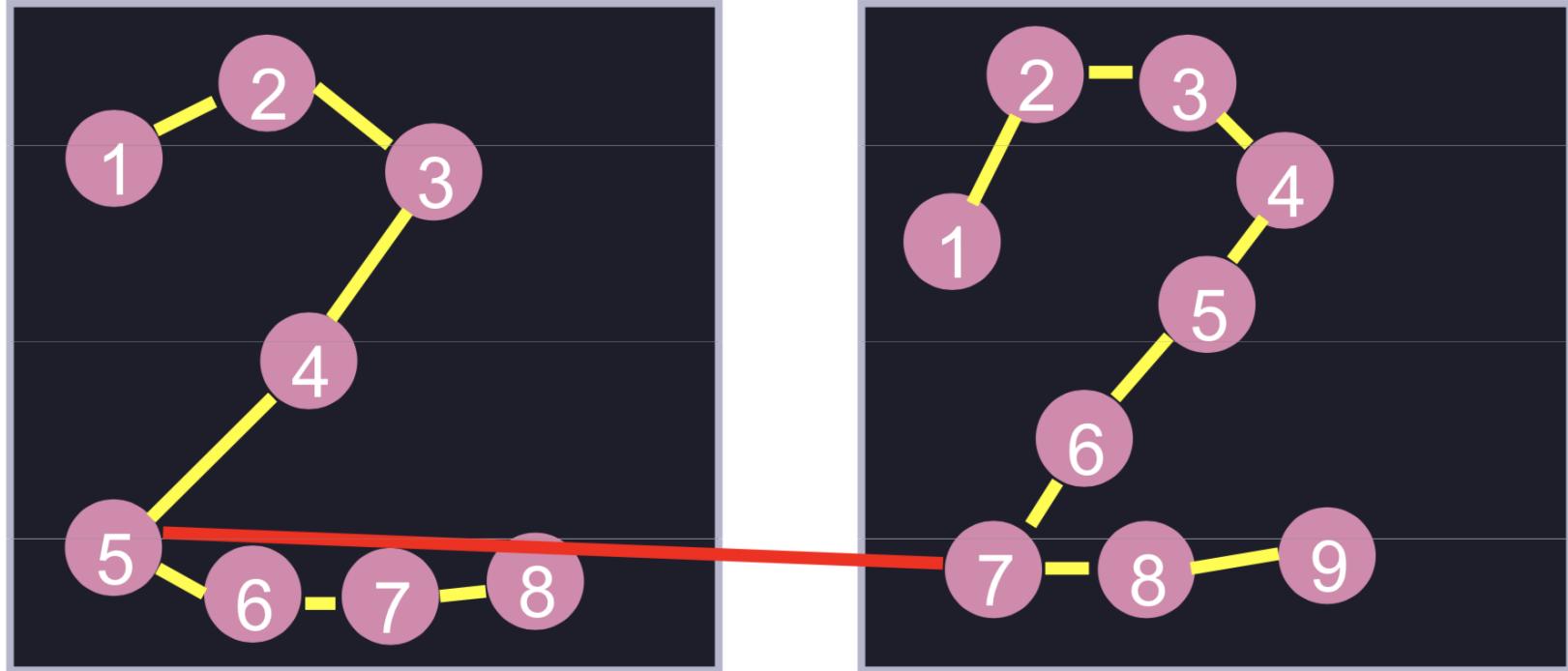
# Matching trajectories



- We can align them:

$((1, 1), (2, 2), (2, 3), (3, 4), (4, 5), \textcolor{red}{(4, 6)}$

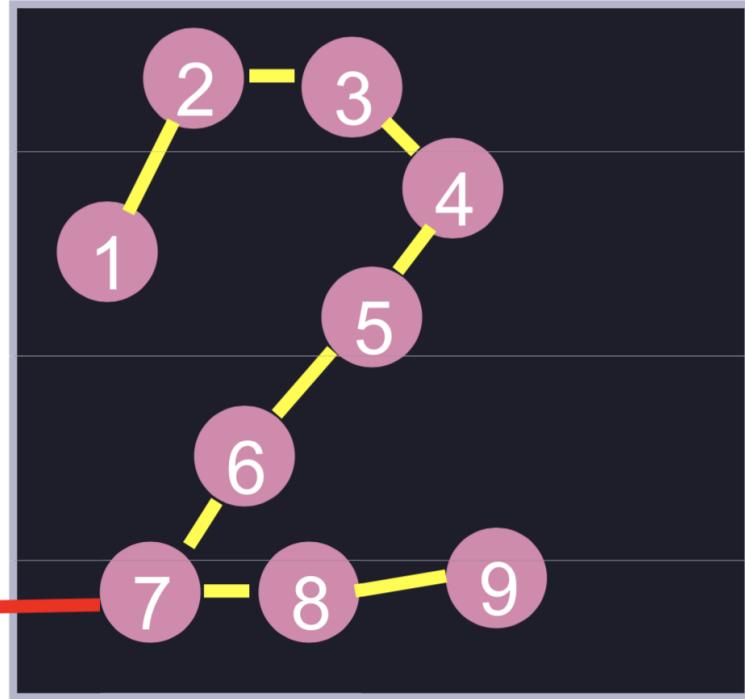
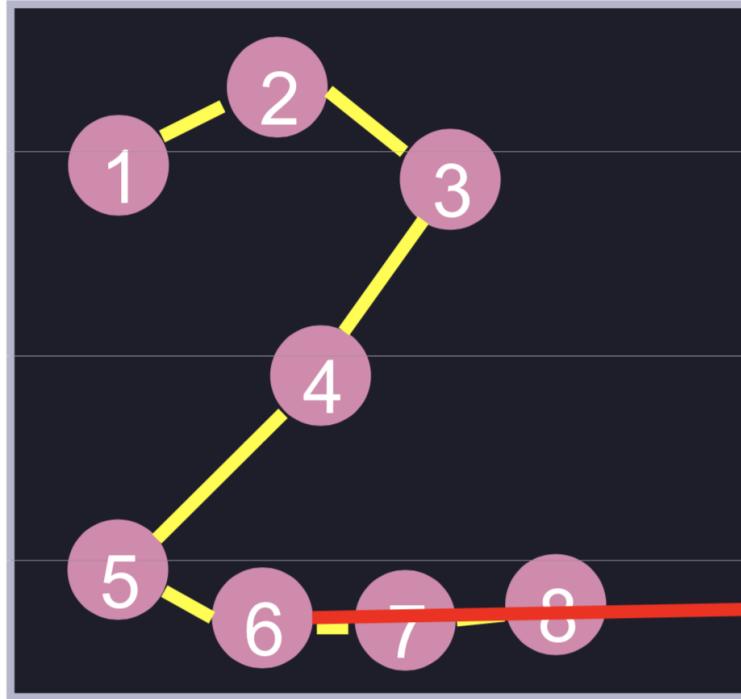
# Matching trajectories



- We can align them:

$((1, 1), (2, 2), (2, 3), (3, 4), (4, 5), (4, 6), (5, 7)$

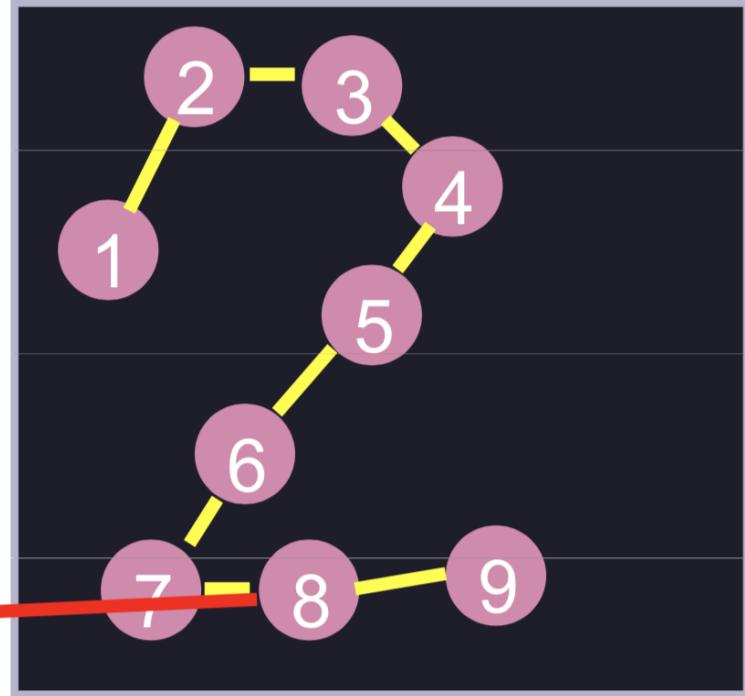
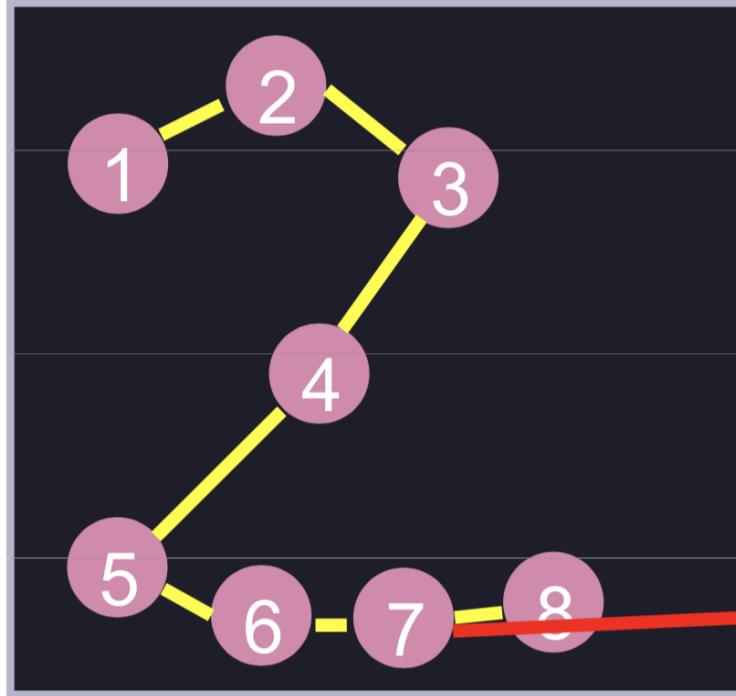
# Matching trajectories



- We can align them:

$((1, 1), (2, 2), (2, 3), (3, 4), (4, 5), (4, 6), (5, 7), \textcolor{red}{(6, 7)})$

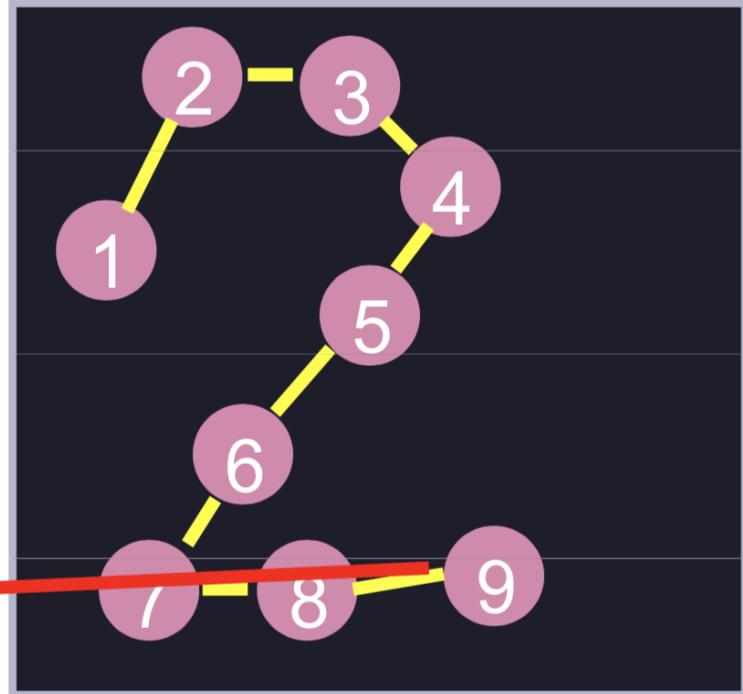
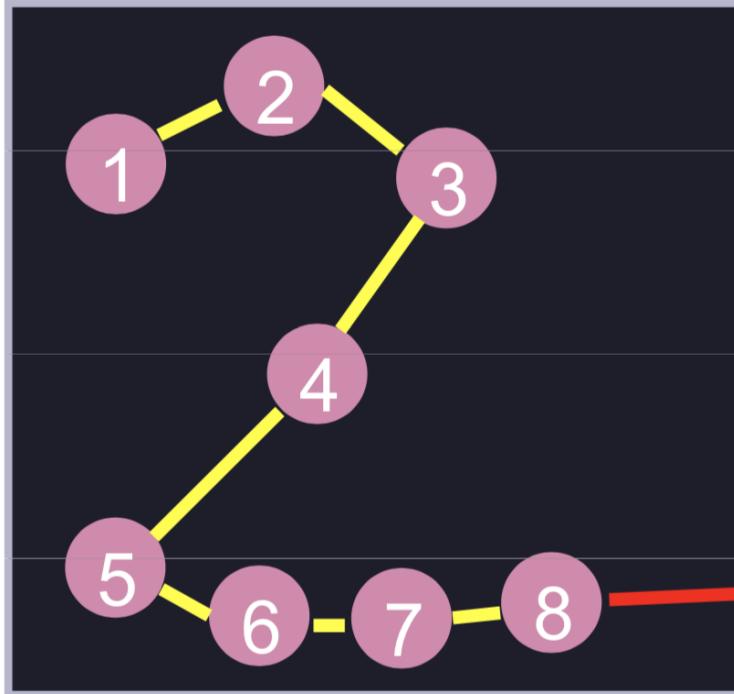
# Matching trajectories



- We can align them:

$((1, 1), (2, 2), (2, 3), (3, 4), (4, 5), (4, 6), (5, 7), (6, 7), (7, 8)$

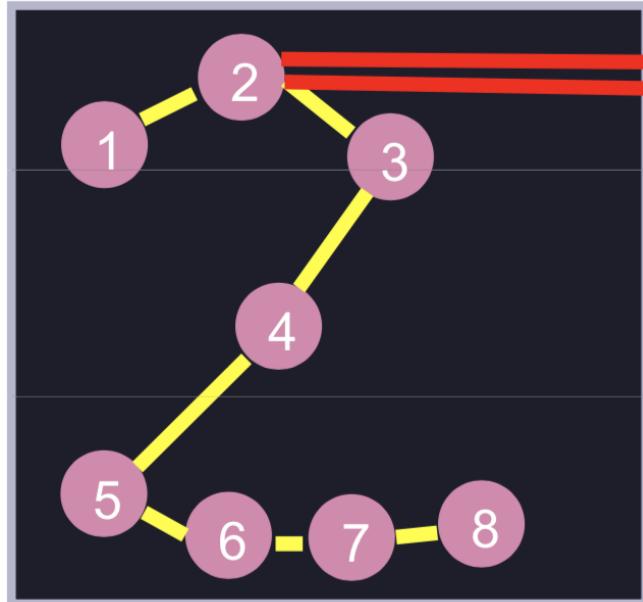
# Matching trajectories



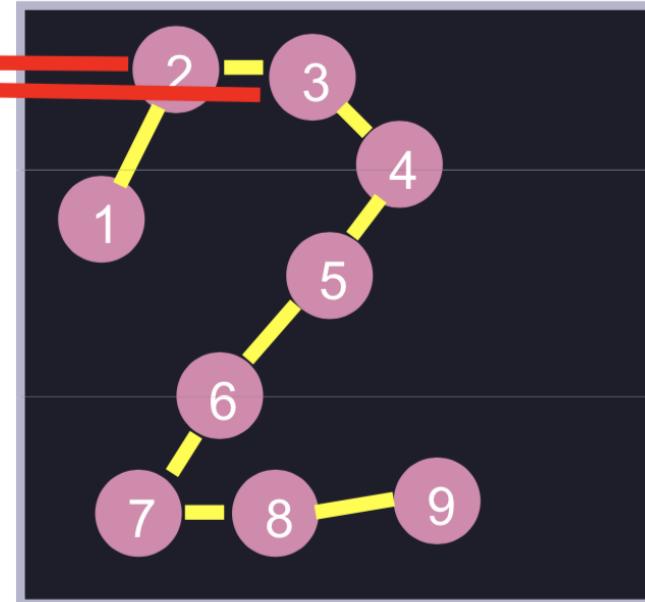
- We can align them:

$((1, 1), (2, 2), (2, 3), (3, 4), (4, 5), (4, 6), (5, 7), (6, 7), (7, 8), (8, 9))$

# Matching trajectories



$$M = (M_1, M_2, \dots, M_8).$$



$$Q = (Q_1, Q_2, \dots, Q_9).$$

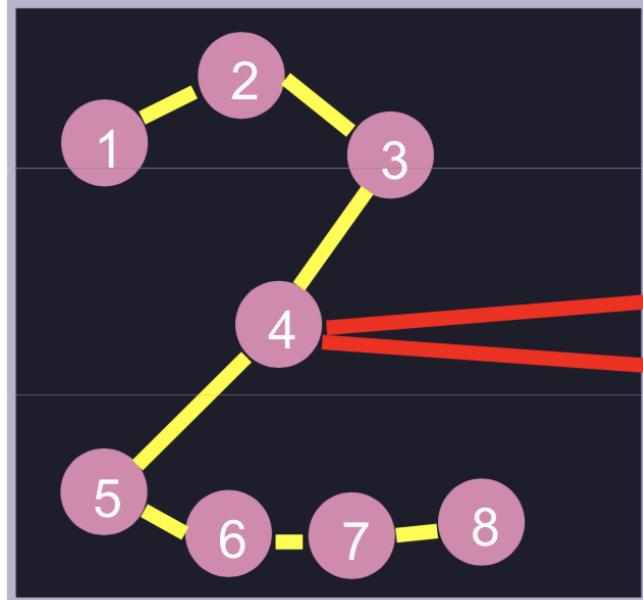
- We can align them:

$((1, 1), (2, 2), (2, 3), (3, 4), (4, 5), (4, 6), (5, 7), (6, 7), (7, 8), (8, 9))$

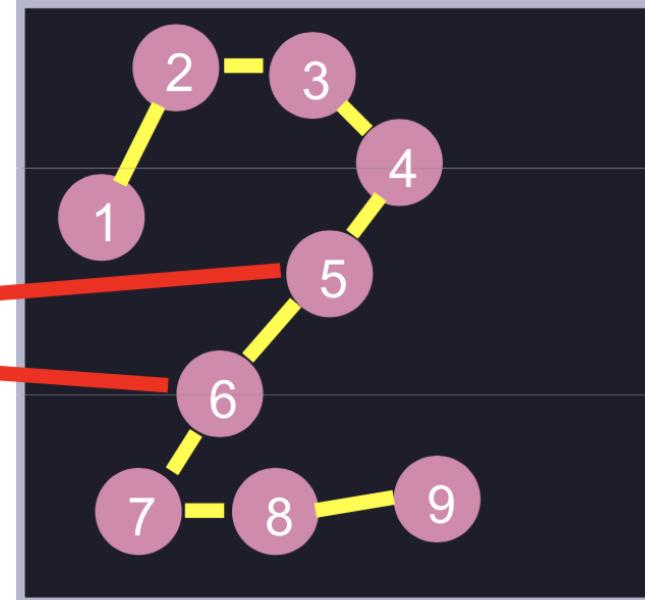
- Can be many-to-many:

$M_2$  is matched with  $Q_2$  and  $Q_3$

# Matching trajectories



$$M = (M_1, M_2, \dots, M_8).$$



$$Q = (Q_1, Q_2, \dots, Q_9).$$

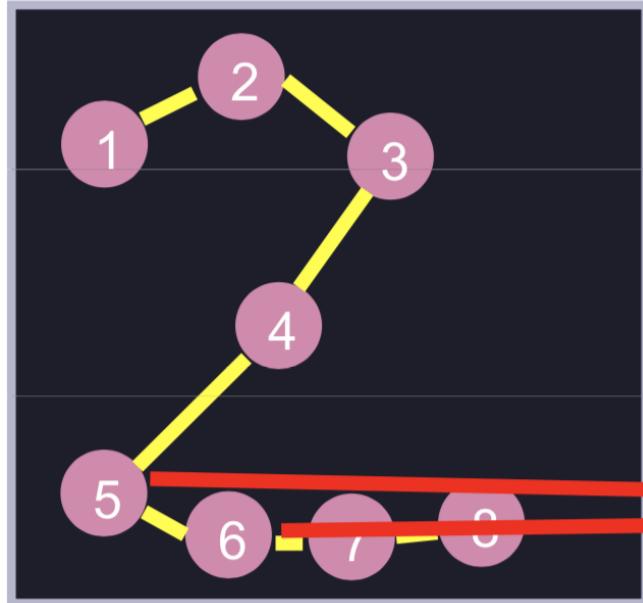
- We can align them:

$((1, 1), (2, 2), (2, 3), (3, 4), (4, 5), (4, 6), (5, 7), (6, 7), (7, 8), (8, 9))$

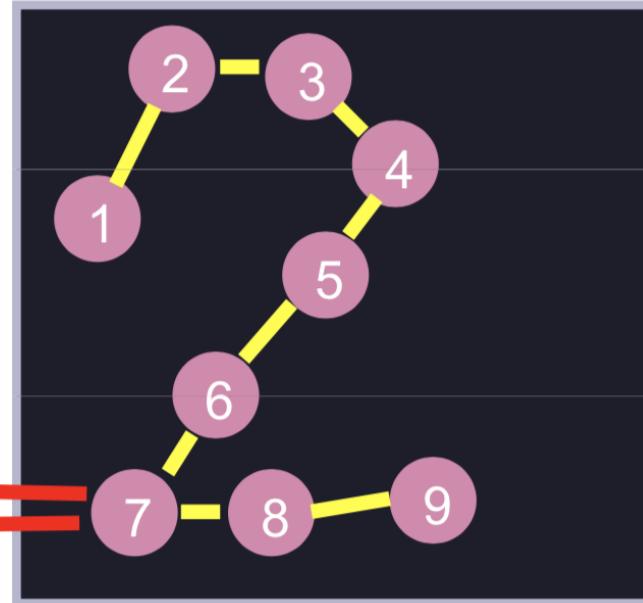
- Can be many-to-many:

$M_4$  is matched with  $Q_5$  and  $Q_6$

# Matching trajectories



$$M = (M_1, M_2, \dots, M_8).$$



$$Q = (Q_1, Q_2, \dots, Q_9).$$

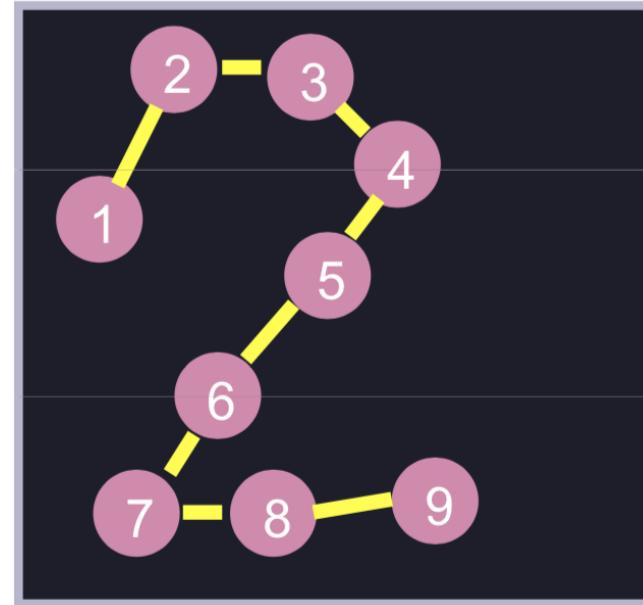
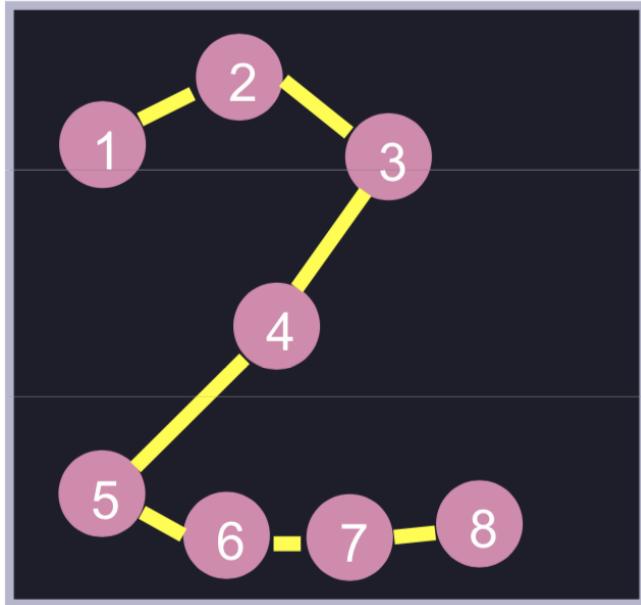
- We can align them:

$((1, 1), (2, 2), (2, 3), (3, 4), (4, 5), (4, 6), (5, 7), (6, 7), (7, 8), (8, 9))$

- Can be many-to-many:

$M_5$  and  $M_6$  are matched with  $Q_7$

# Matching trajectories

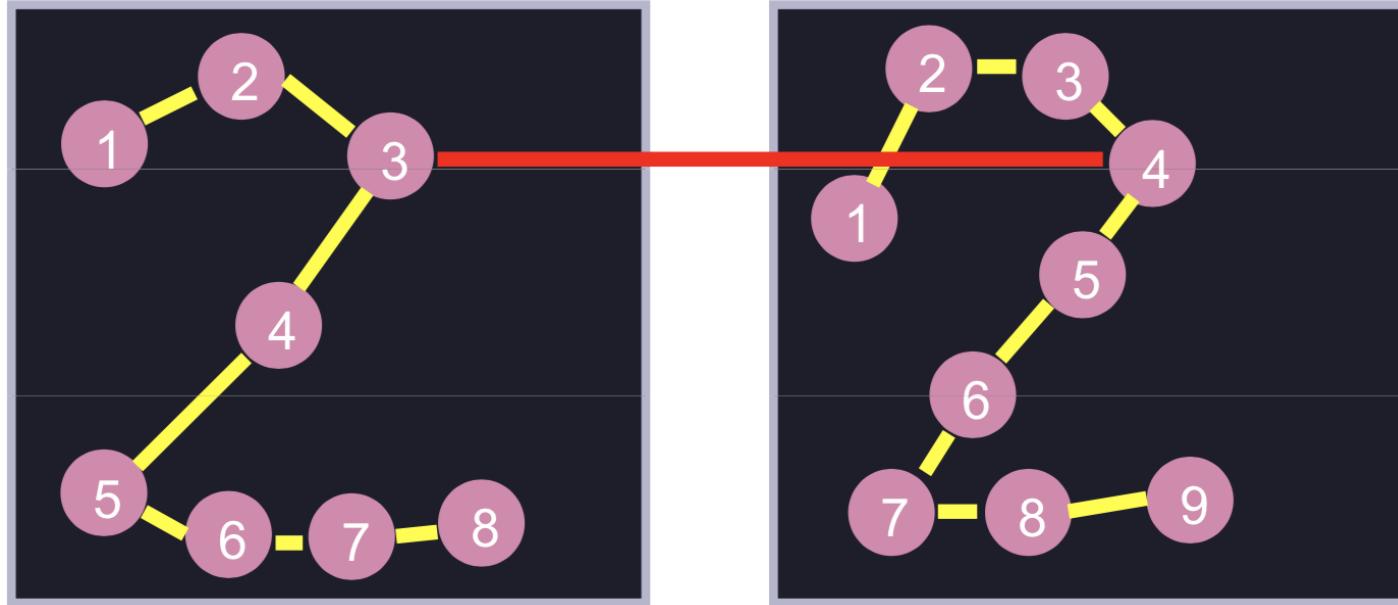


- What is the cost of alignment?

$$C = \text{cost}(s_1, t_1) + \text{cost}(s_2, t_2) + \dots + \text{cost}(s_m, t_n)$$

- We could use the Euclidean distance:  $\text{cost}(s_i, t_i) = d_{L_2}(s_i, t_i)$

# Matching trajectories

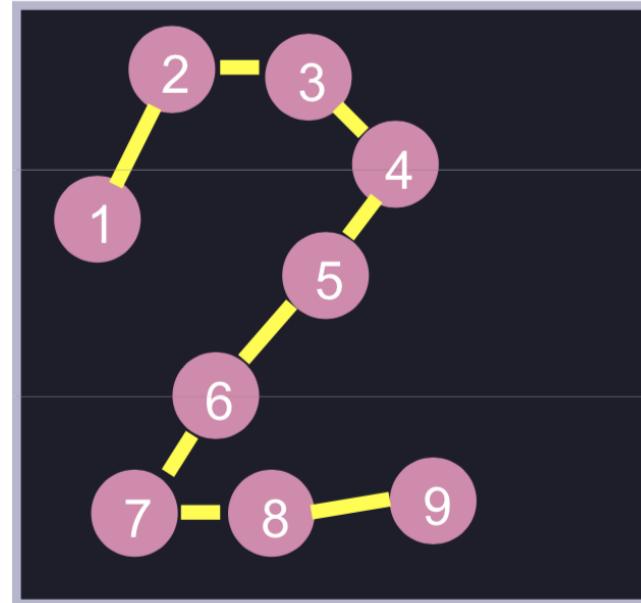
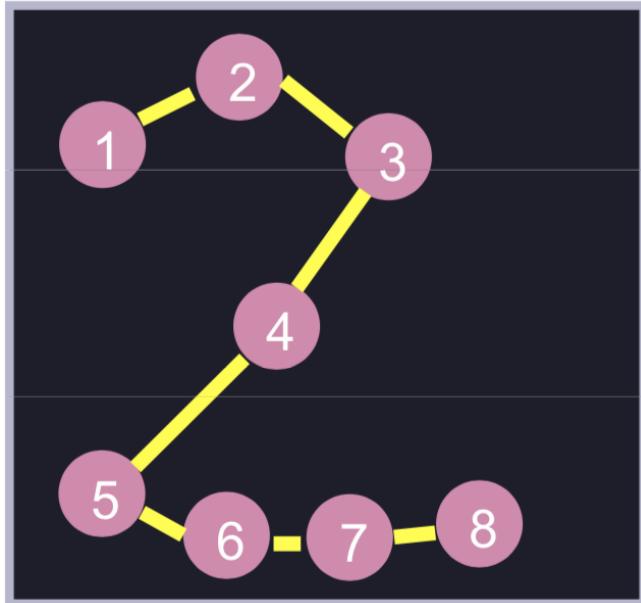


- What is the cost of alignment?

$$C = \text{cost}(s_1, t_1) + \text{cost}(s_2, t_2) + \dots + \text{cost}(s_m, t_n)$$

- We could use the Euclidean distance:  $\text{cost}(s_i, t_i) = d_{L_2}(s_i, t_i)$
- Example:  $\text{cost}(3,4) = d_{L_2}(M_3, Q_4)$

# Matching trajectories

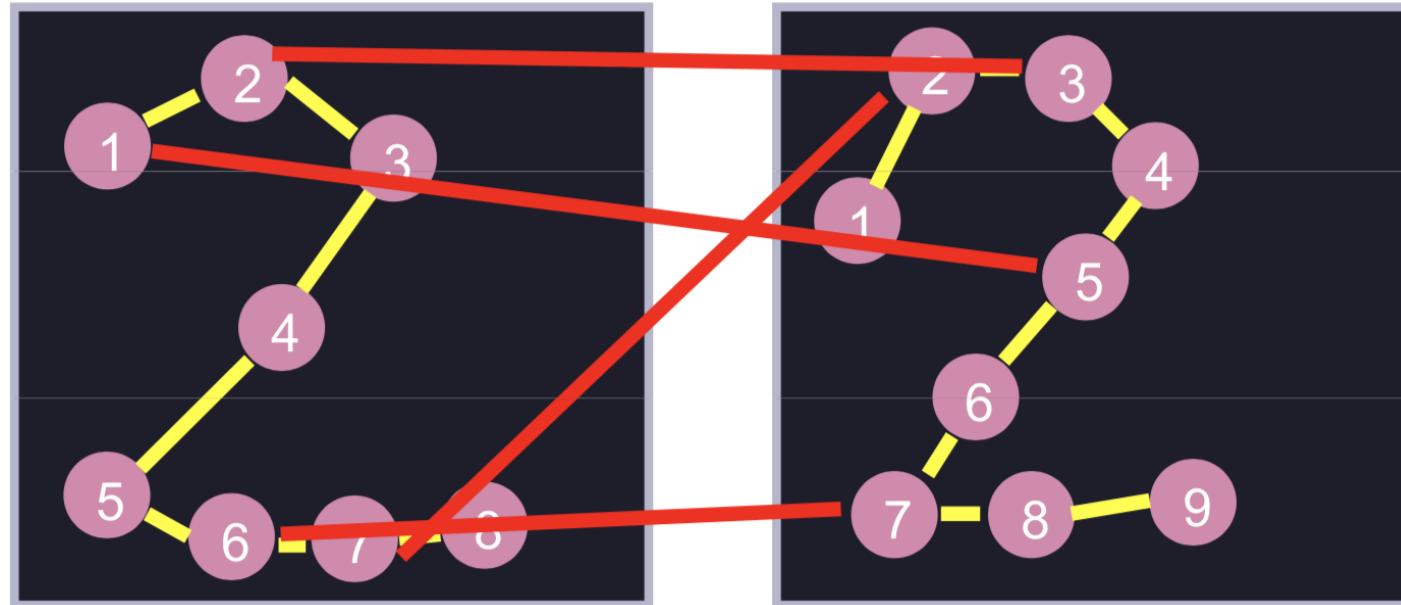


- We can align them:

$((1, 1), (2, 2), (2, 3), (3, 4), (4, 5), (4, 6), (5, 7), (6, 7), (7, 8), (8, 9))$

- What are the rules of alignment?

# Matching trajectories



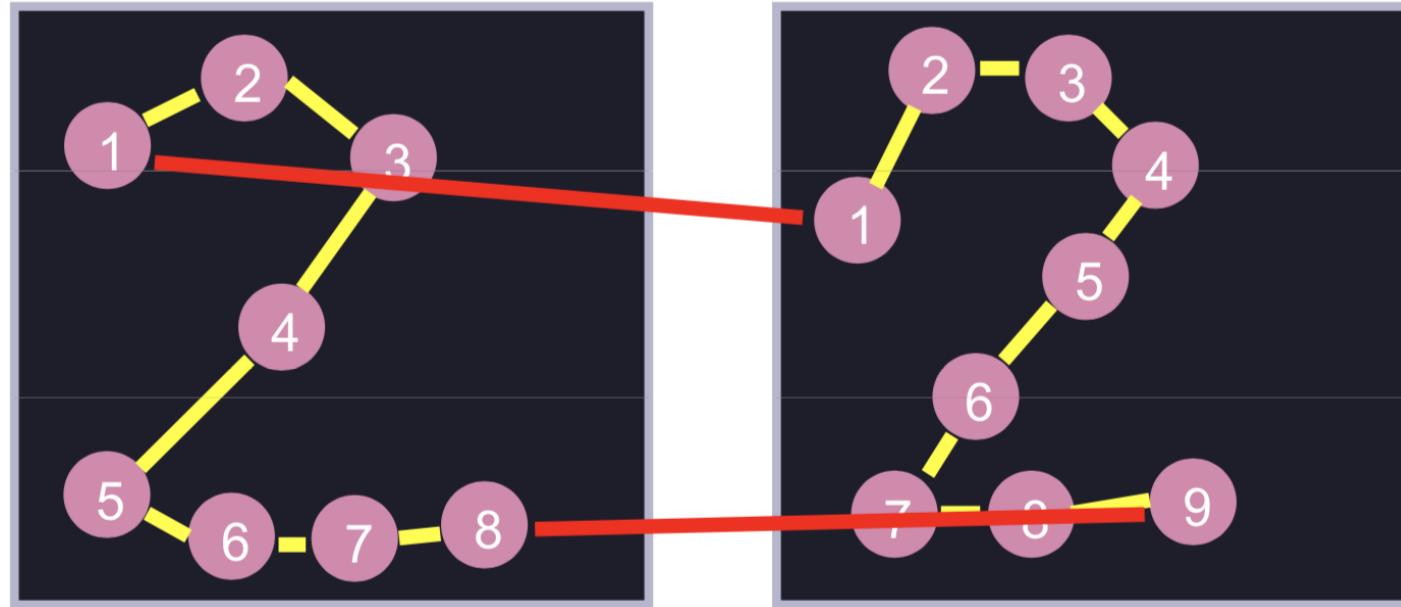
- We can align them:

$((1, 1), (2, 2), (2, 3), (3, 4), (4, 5), (4, 6), (5, 7), (6, 7), (7, 8), (8, 9))$

- What are the rules of alignment?

- Is alignment  $((1, 5), (2, 3), (6, 7), (7, 1))$  legal?
- Depends on what makes sense in our application

# Rules of alignment for DTW



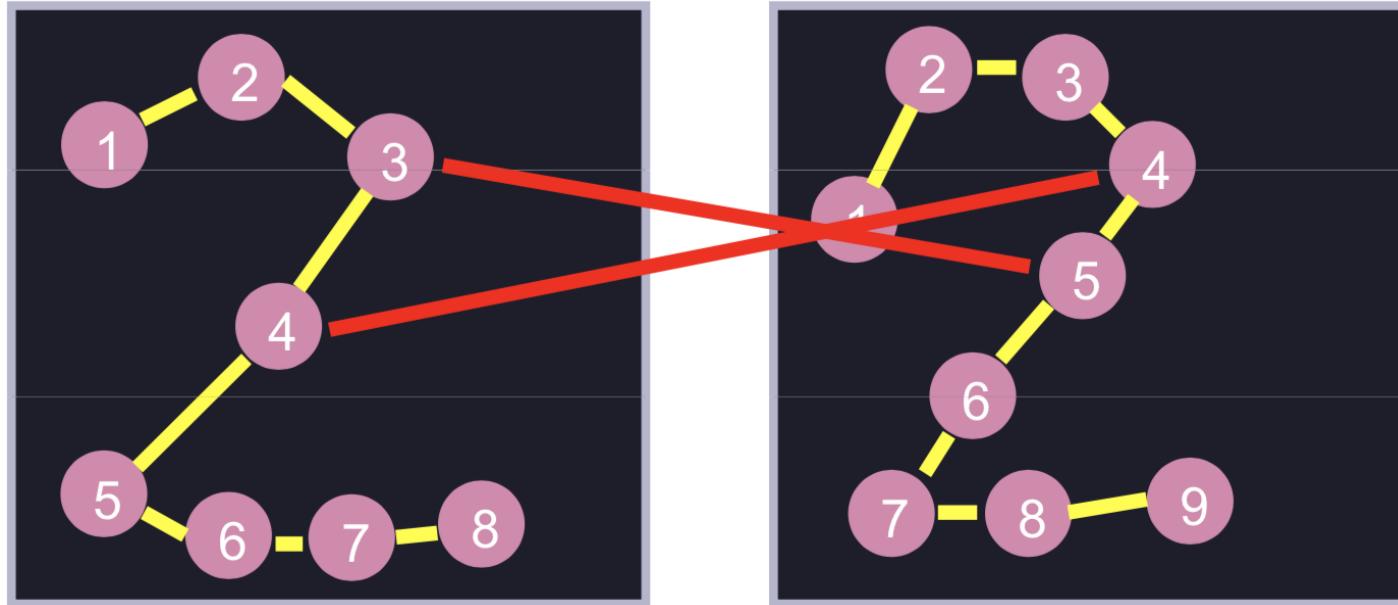
- We can align them:

$((1, 1), (2, 2), (2, 3), (3, 4), (4, 5), (4, 6), (5, 7), (6, 7), (7, 8), (8, 9))$

- DTW rules: boundaries

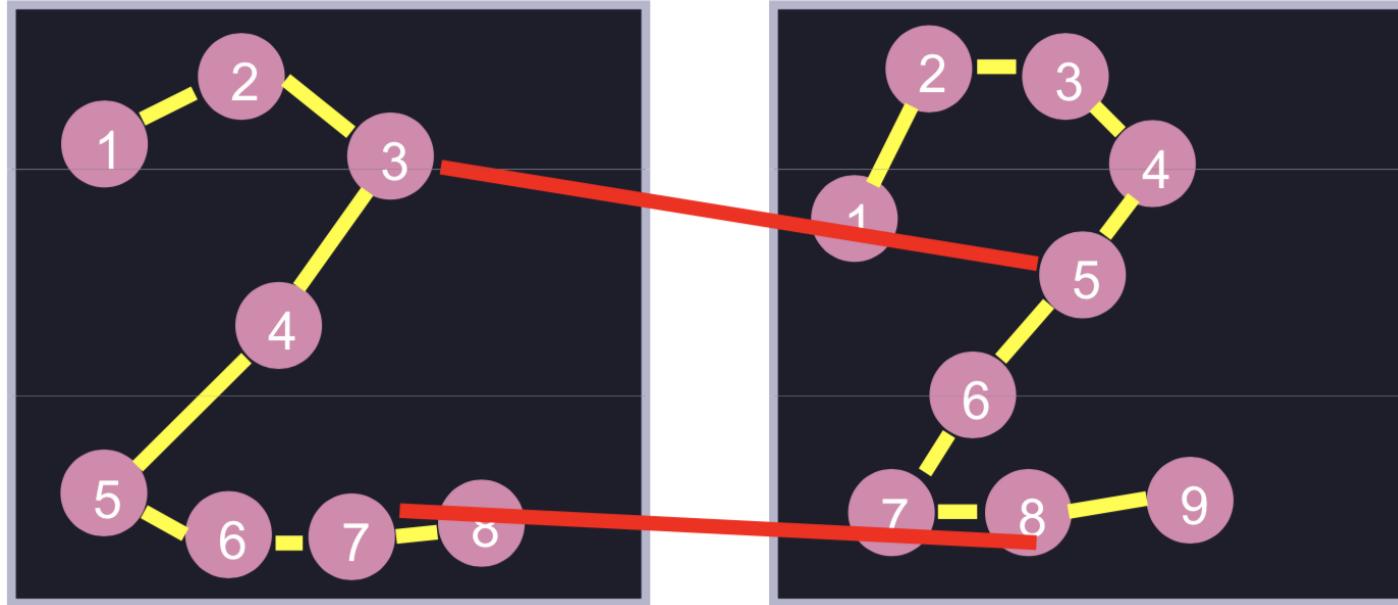
- First elements match:  $(s_1, t_1)$
- Last elements match:  $(s_m, t_n)$

# Rules of alignment for DTW



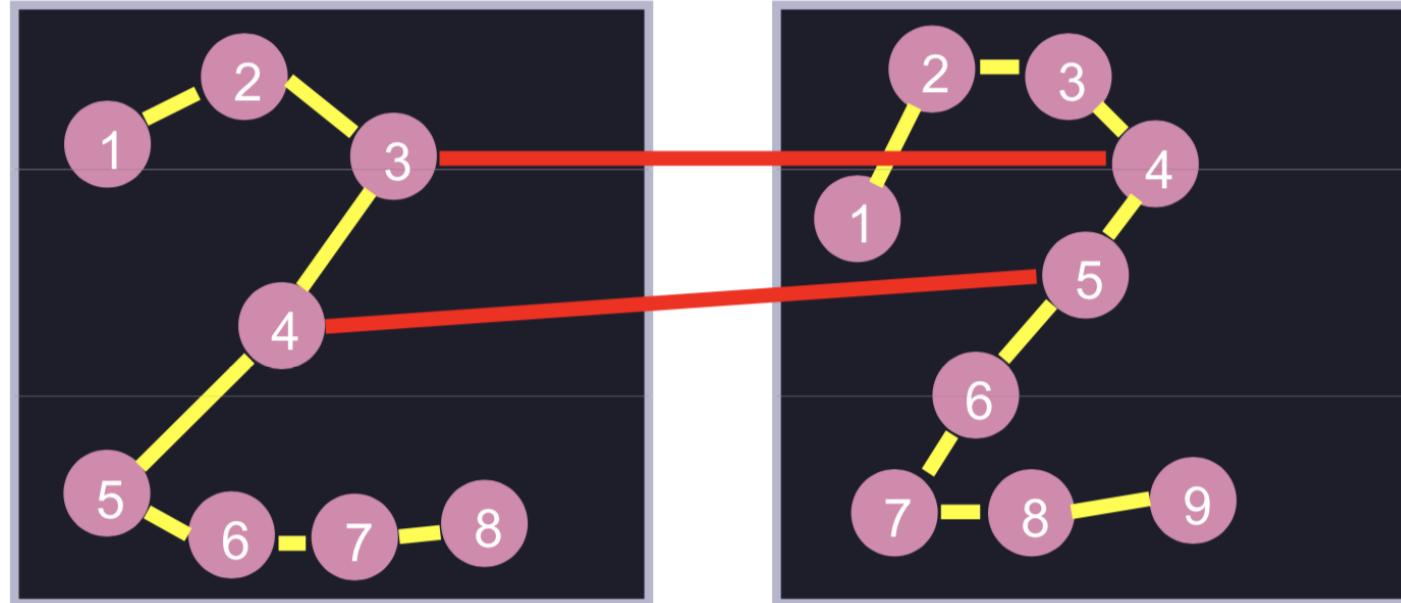
- Illegal alignment (violating monotonicity):  
 $(\dots, (3, 5), (4, 4), \dots)$
- DTW rules: monotonicity (the alignment cannot go backwards)
  - $0 \leq (s_{i+1} - s_i)$
  - $0 \leq (t_{i+1} - t_i)$

# Rules of alignment for DTW



- Illegal alignment (violating continuity):  
 $(\dots, (3, 5), (7, 8), \dots)$
- DTW rules: continuity (the alignment cannot skip elements)
  - $(s_{i+1} - s_i) \leq 1$
  - $(t_{i+1} - t_i) \leq 1$

# Rules of alignment for DTW



- Valid alignment:

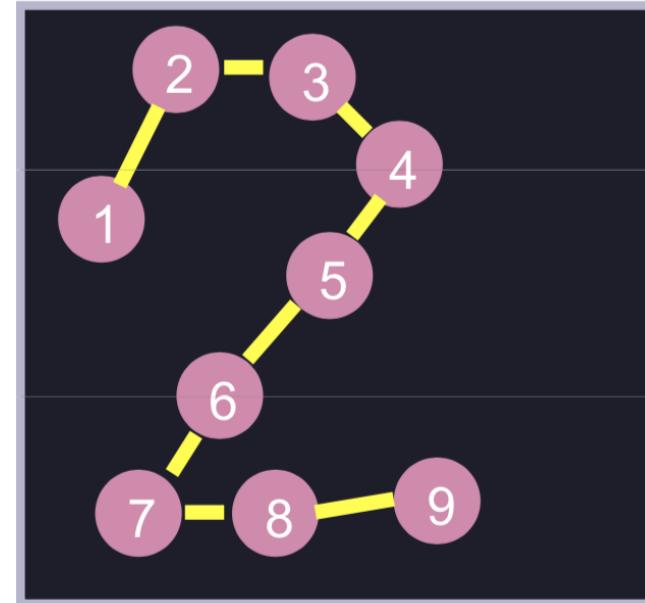
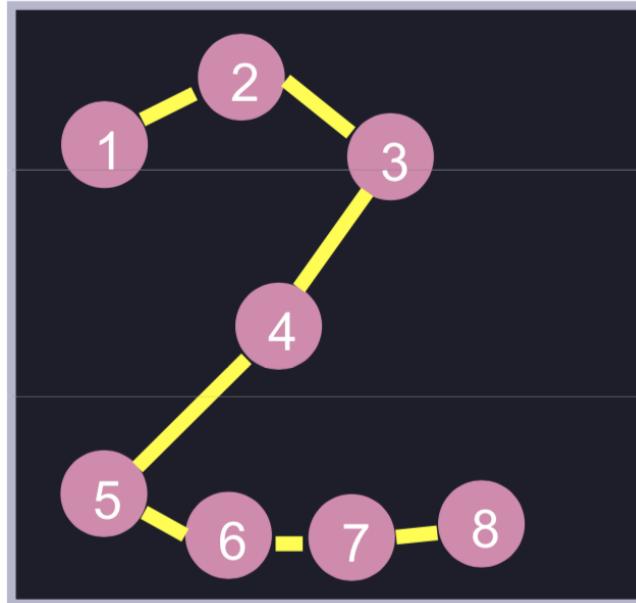
$((1, 1), (2, 2), (2, 3), (3, 4), (4, 5), (4, 6), (5, 7), (6, 7), (7, 8), (8, 9))$

- DTW rules: monotonicity and continuity

➤  $0 \leq (s_{i+1} - s_i) \leq 1$

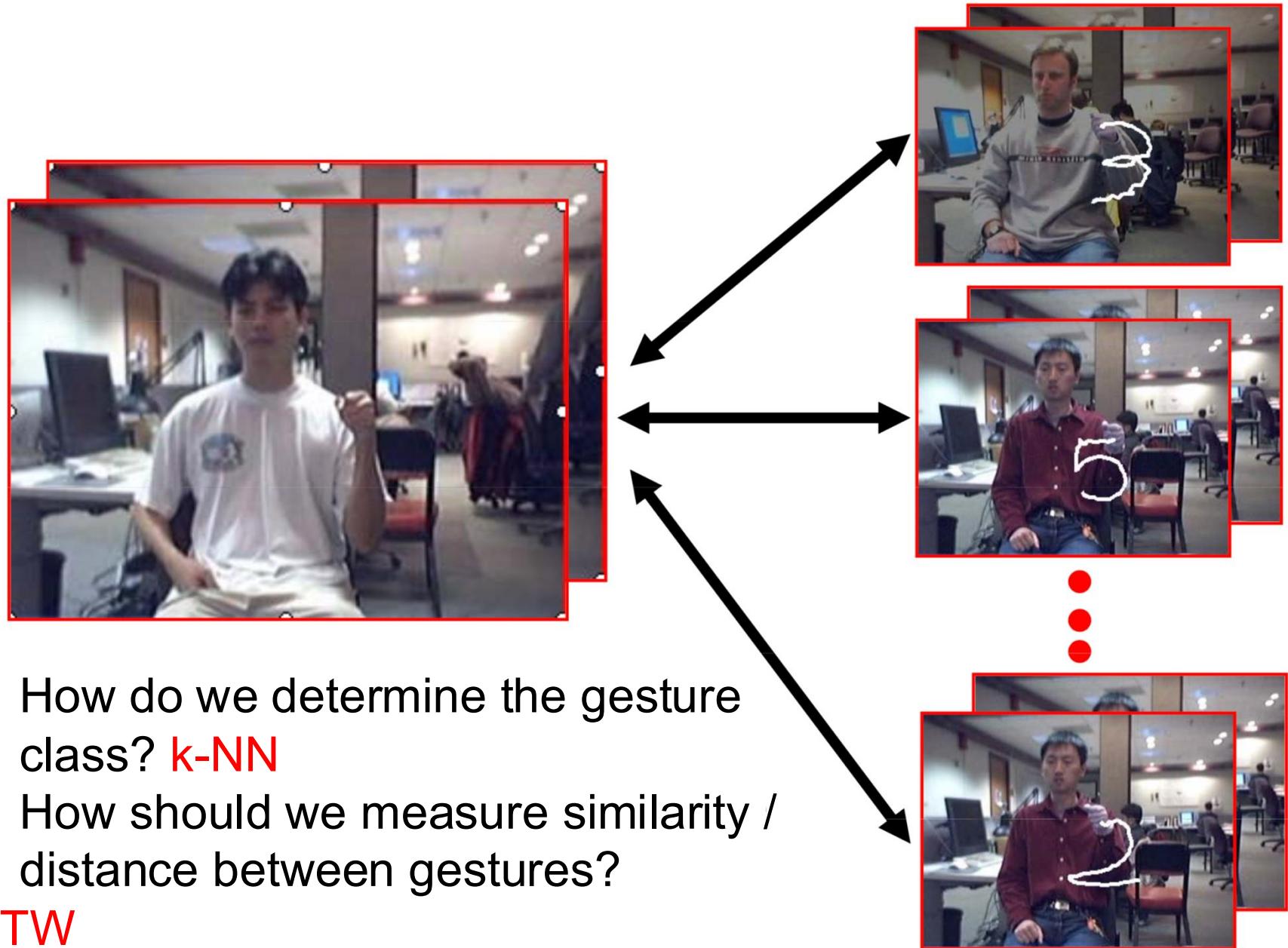
➤  $0 \leq (t_{i+1} - t_i) \leq 1$

# Dynamic Time Warping

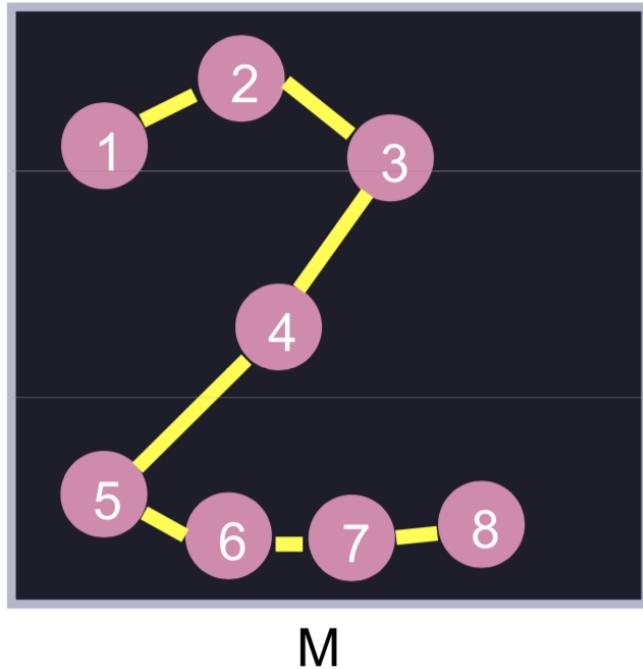


- Dynamic Time Warping (DTW) is a distance measure between sequences of points
- The DTW distance is the cost of the optimal alignment between two trajectories
  - The alignment must obey the DTW rules defined in the previous slides

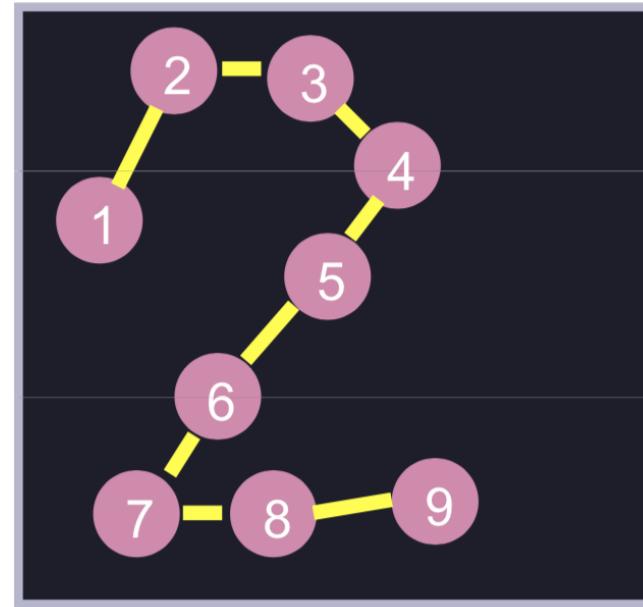
# Gesture recognition



# Computing DTW (Edit) distance



$M$



$Q$

- Training sample:  $M = (M_1, M_2, \dots, M_8)$
- Test sample:  $Q = (Q_1, Q_2, \dots, Q_9)$
- Each  $M_i$  and  $Q_j$  can be, for example, a 2D pixel location of the hand (center of the bounding box)

# Computing DTW (Edit) distance

- Training sample:  $M = (M_1, M_2, \dots, M_8)$
- Test sample:  $Q = (Q_1, Q_2, \dots, Q_9)$
- We want optimal alignment between  $M$  and  $Q$
- Dynamic programming strategy:
  - Break problem up into smaller, interrelated problems  $P(i,j)$ , which we solve using a recursive relation
  - Problem  $P(i,j)$ : find optimal alignment between  $(M_1, M_2, \dots, M_i)$  and  $(Q_1, Q_2, \dots, Q_j)$

# Computing DTW (Edit) distance

- Solve problem  $P(1, j)$ :
  - The optimal alignment is:  $((1, 1), (1, 2), \dots, (1, j))$
- Solve problem  $P(i, 1)$ :
  - The optimal alignment is:  $((1, 1), (2, 1), \dots, (i, 1))$
- Solve problem  $P(i, j)$ :
  - Find best solution from:  
 $(i, j-1), (i-1, j), (i-1, j-1)$
  - Add the pair  $(i, j)$  to the solution best solution from above

# The DTW algorithm

- Input:

- Training sample:  $M = [M_1, M_2, \dots, M_8]$
- Test sample:  $Q = [Q_1, Q_2, \dots, Q_9]$

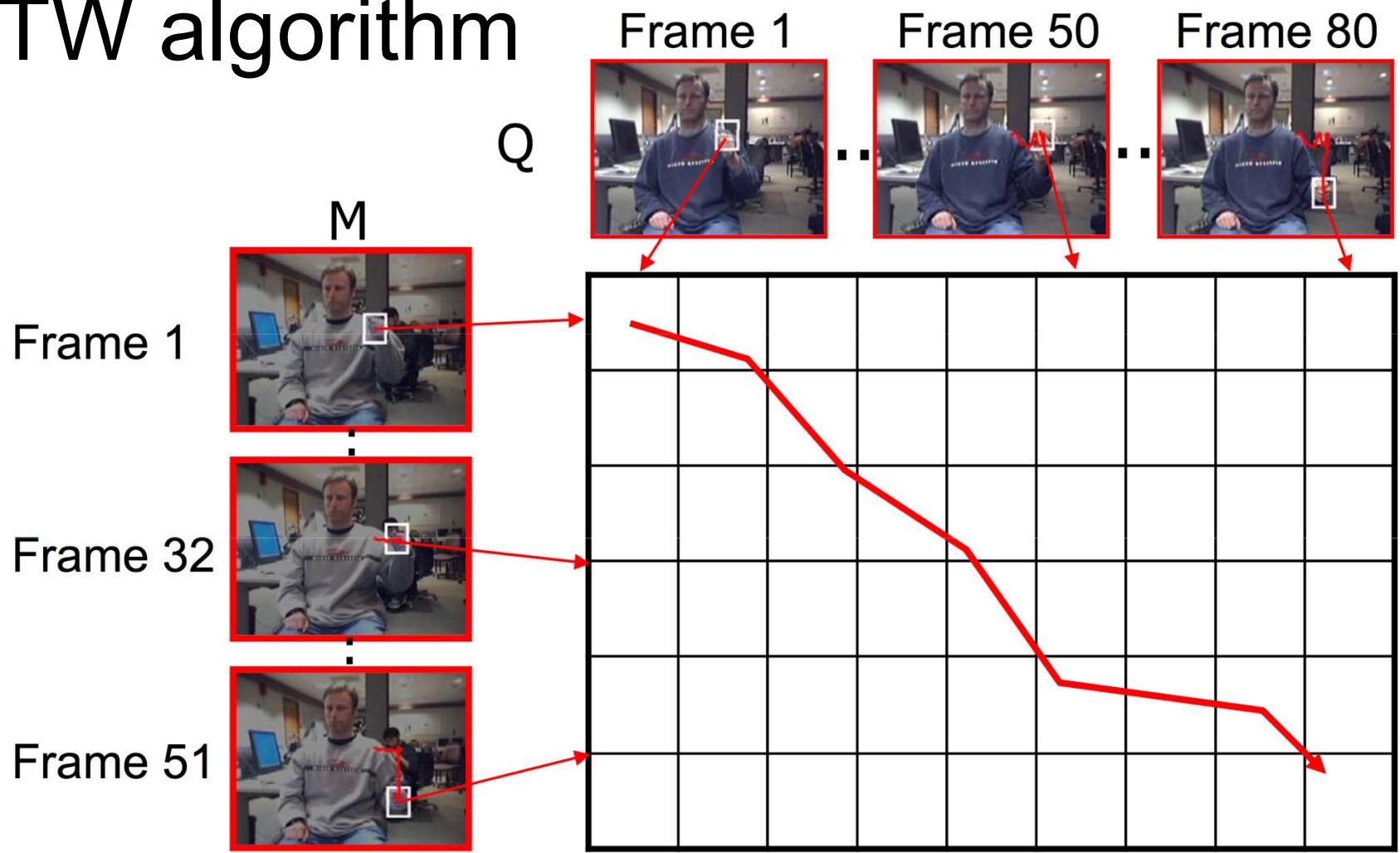
- Algorithm:

```
C = np.zeros((m, n))
C[0, 0] = cost(M1, Q1)
for i in range(1, m):
    C[i, 0] = C[i-1, 0] + cost(Mi, Q1)
for j in range(1, n):
    C[0, j] = C[0, j-1] + cost(M1, Qj)
for i in range(1, m):
    for j in range(1, n):
        C[i, j] = cost(Mi, Qj) + np.min(C[i-1, j], C[i, j-1], C[i-1, j-1])
```

- Return:

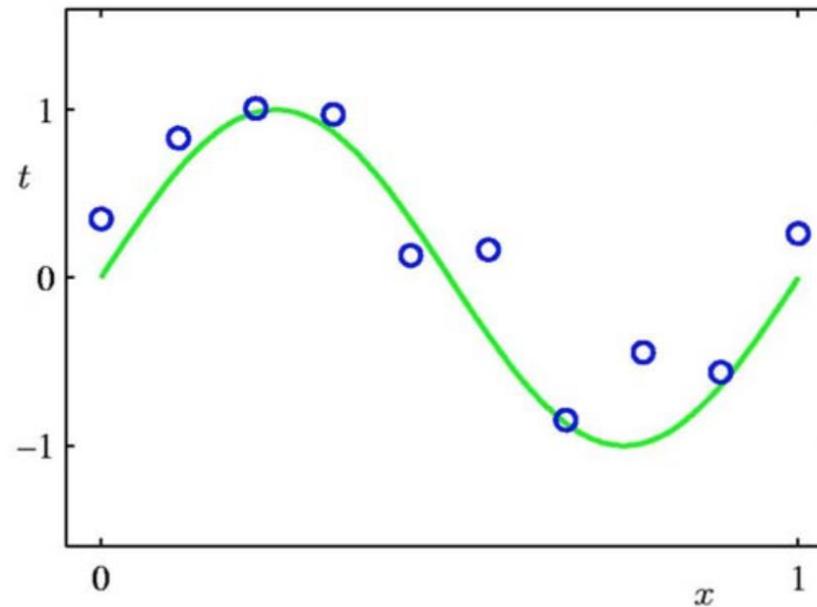
$C[m-1, n-1]$

# The DTW algorithm



- For each cell  $(i, j)$ :
  - Compute optimal alignment of  $M[1:i]$  and  $Q[1:j]$
  - Answer depends only on  $(i-1, j)$ ,  $(i, j-1)$ ,  $(i-1, j-1)$
  - Time: linear to size of table, quadratic to length of gestures

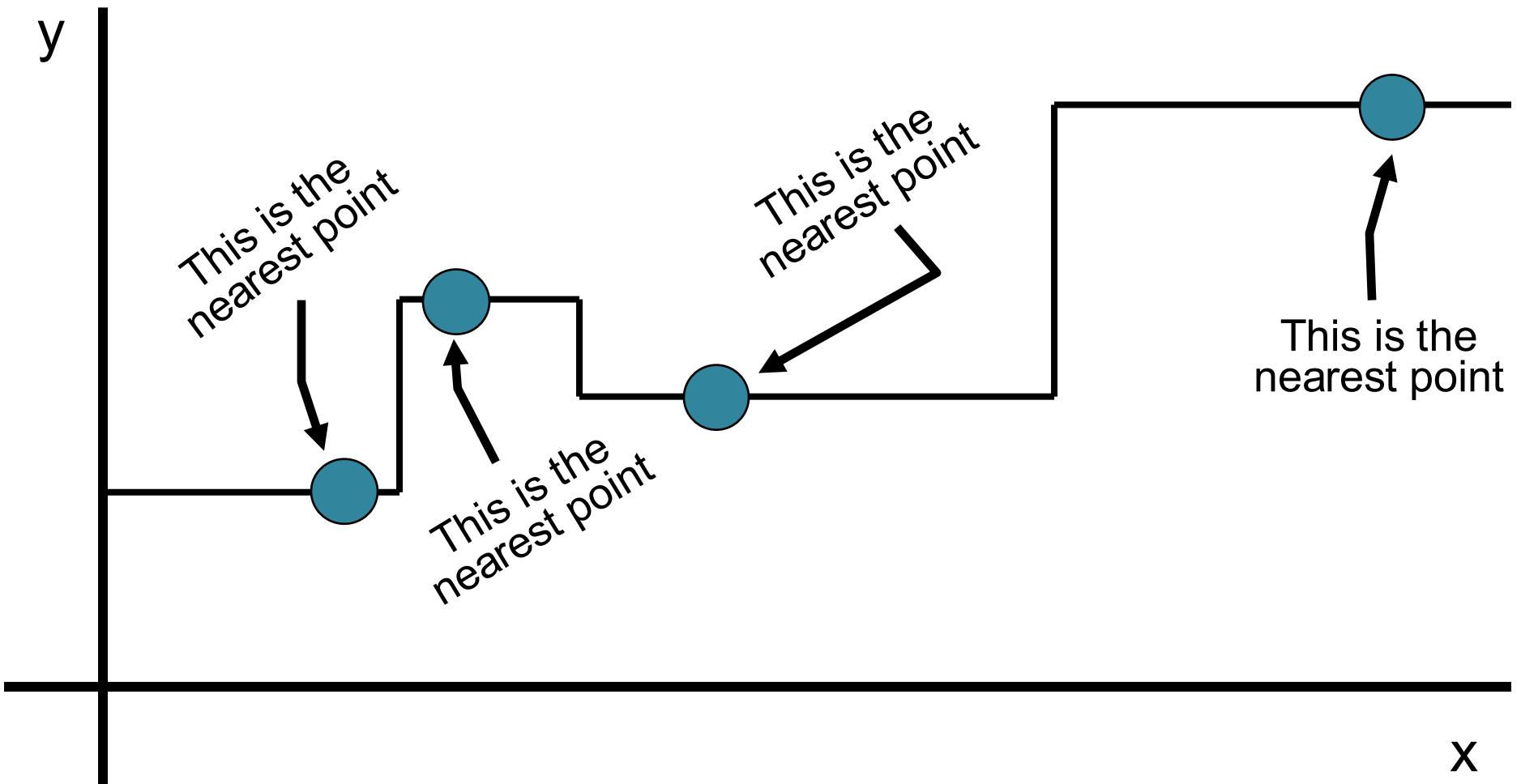
# Regression from labeled samples



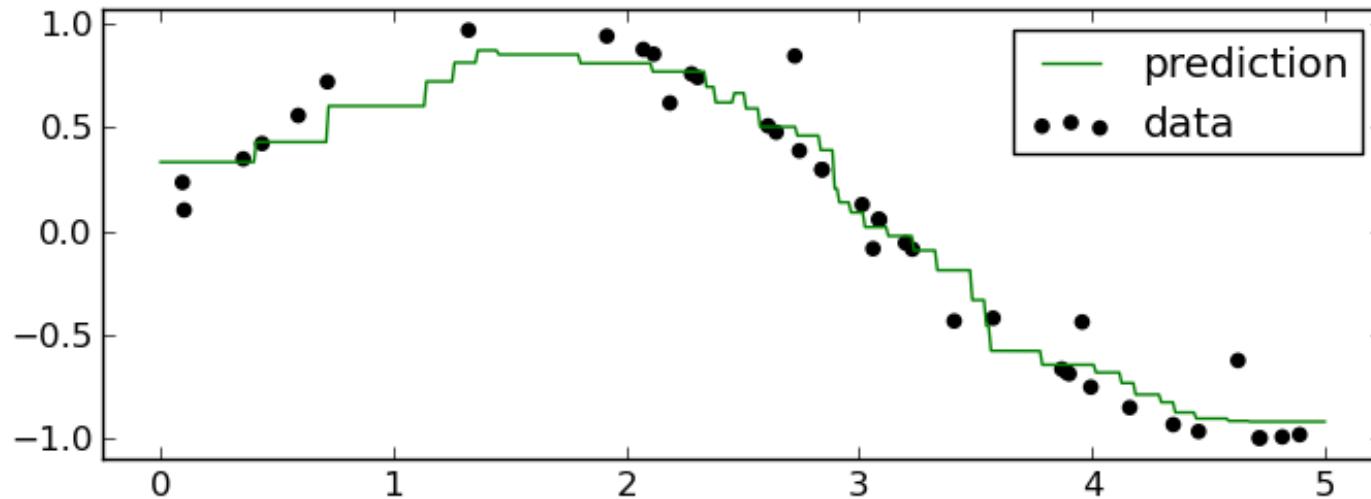
- Suppose we have a set of  $N$  training examples:  
 $(x_1, \dots, x_N)$  and  $(y_1, \dots, y_N)$ ,  $x_i, y_i \in \mathbb{R}$
- The regression problem consists of estimating the function  $g(x)$  such that:

$$g(x_i) = y_i$$

# 1-NN for regression tasks



# k-NN for regression tasks



- k-NN regression algorithm:
  - 1) For each test sample  $x$ , we find the nearest  $k$  neighbors and their labels
  - 2) The output- $y$  is the mean of the labels of the  $k$  neighbors:

$$f(x) = \frac{1}{K} \sum_{i=1}^K y_i$$

# Advantages and properties of k-NN

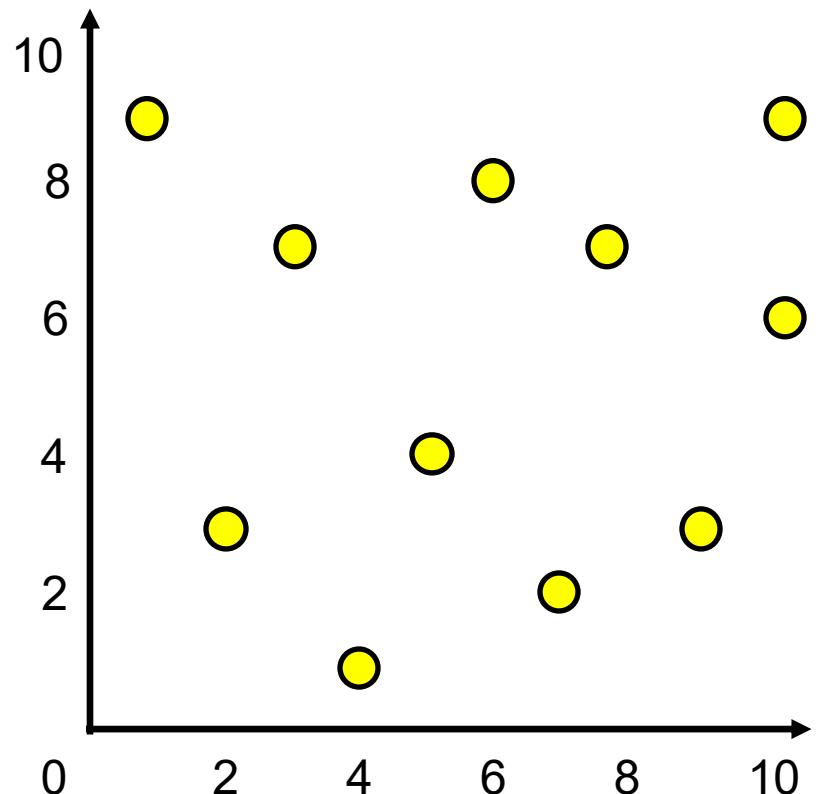
- k-NN is a very simple model
- Can be directly applied to multi-class problems
- The decision boundary is non-linear
- The quality of the results grows with the number of training samples
- We have a single parameter that requires tuning ( $k$ )
- The training error grows with  $k$ , but the decision boundary becomes smoother:
  - Regularization method that increases the generalization capacity (to some extent)

# Disadvantages of k-NN

- What does nearest mean? We have to define a distance
- Is the Euclidean distance always the best choice?
- The computational cost is quite high: we need to store and pass through the whole training set during inference (at test time)
- Possible solutions to avoid the high computational time:
  - Partition the feature space using k-d trees
  - Use locality sensitive hashing
- Suffers from the “curse of dimensionality”

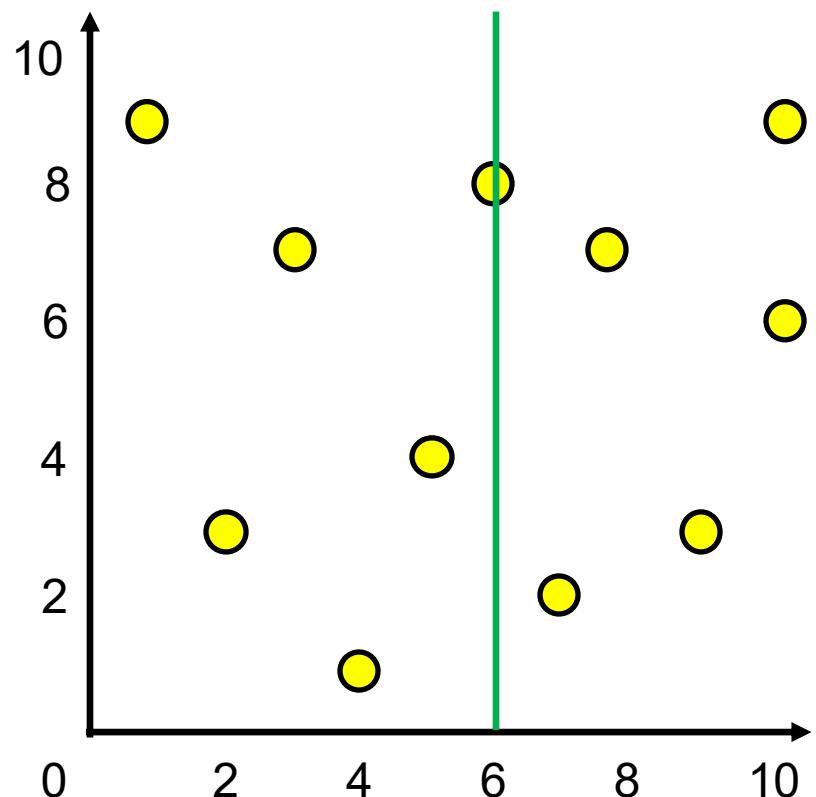
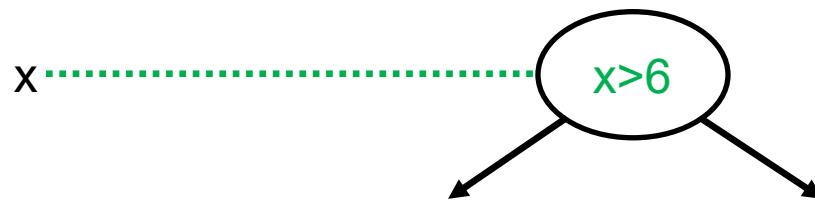
# Approximate k-NN based on k-d tree

- Build a k-d tree from training data:



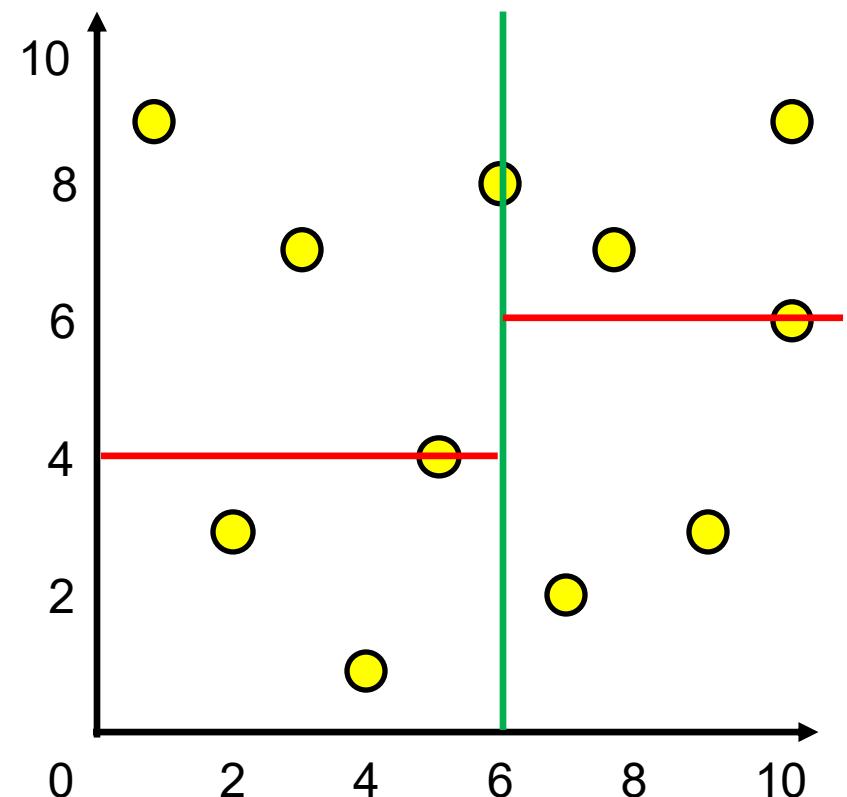
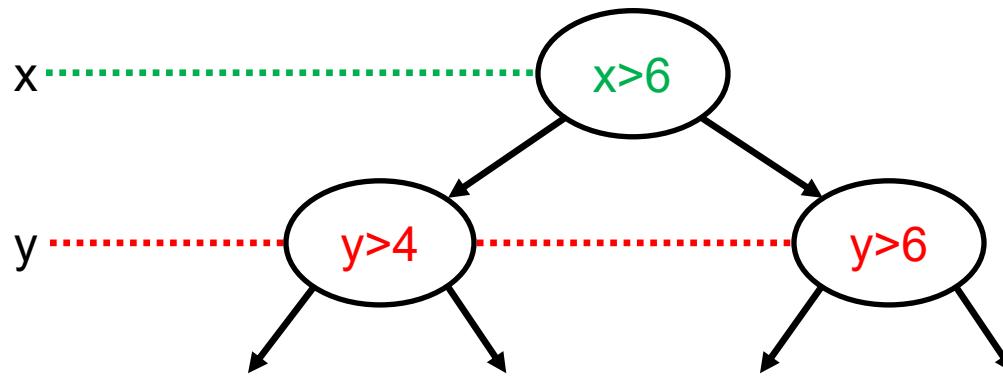
# Approximate k-NN based on k-d tree

- Build a k-d tree from training data:  
pick random/alternate dimension, find median, split data, repeat



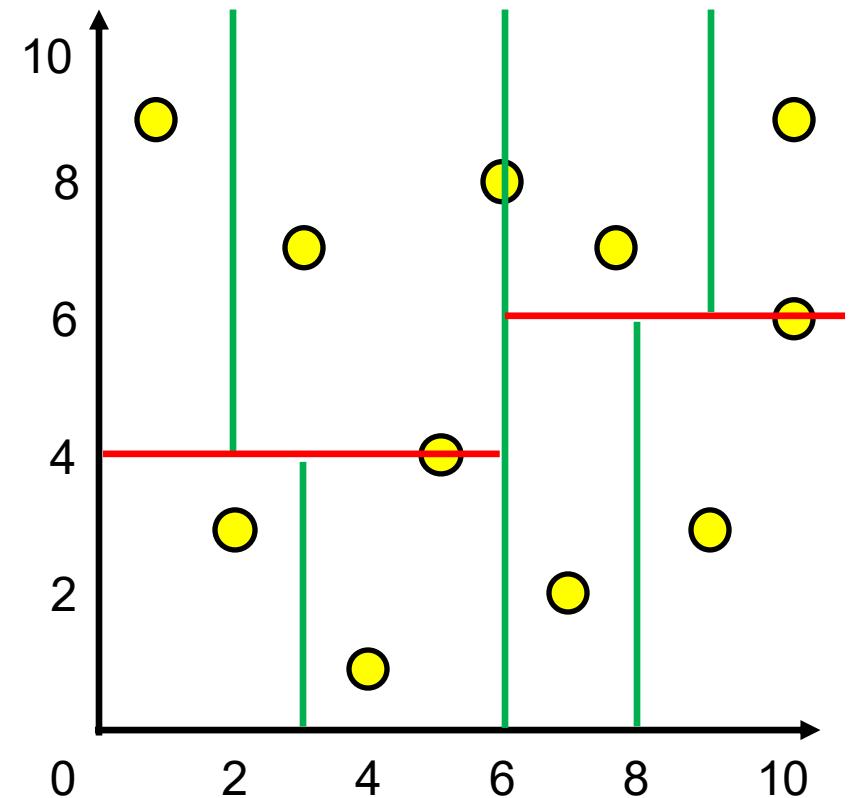
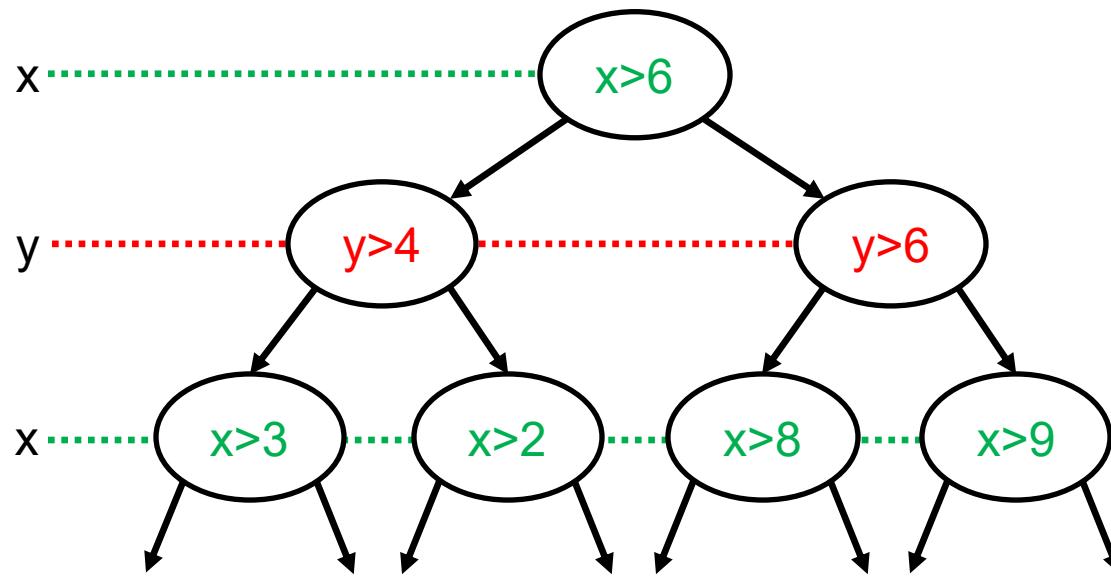
# Approximate k-NN based on k-d tree

- Build a k-d tree from training data:  
pick random/alternate dimension, find median, split data, repeat



# Approximate k-NN based on k-d tree

- Build a k-d tree from training data:  
pick random/alternate dimension, find median, split data, repeat



# Approximate k-NN based on k-d tree

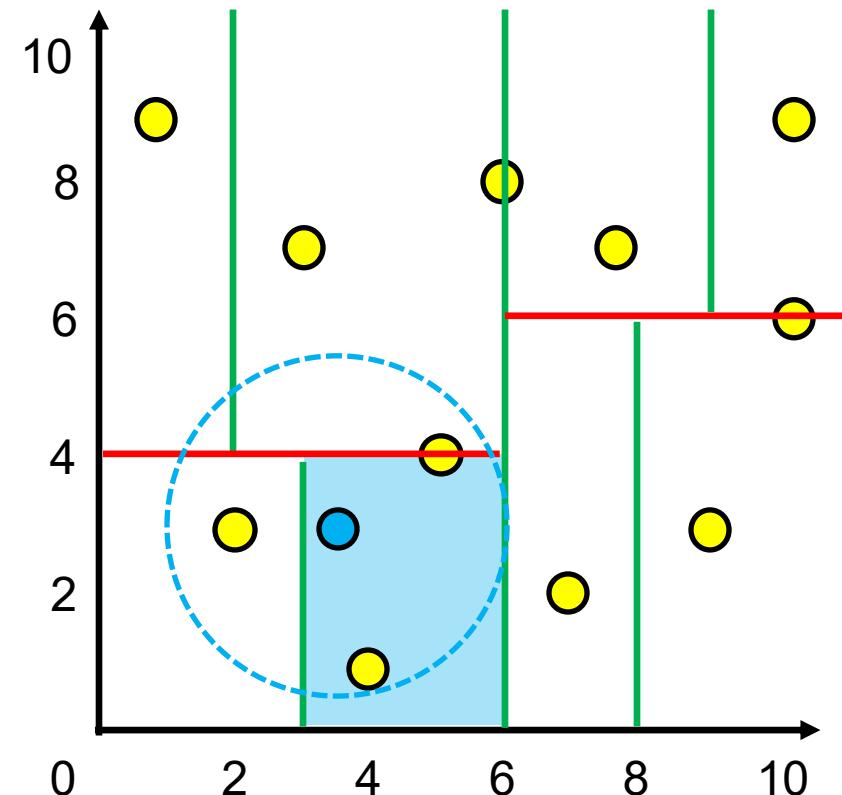
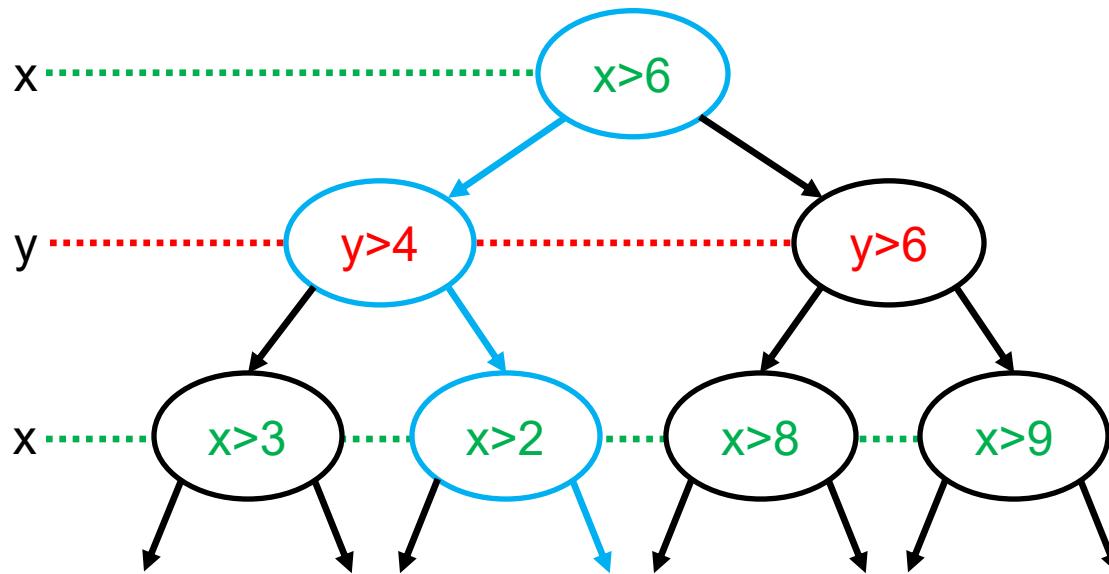
- Build a k-d tree from training data:

pick random/alternate dimension, find median, split data, repeat

- Find NN for test sample:

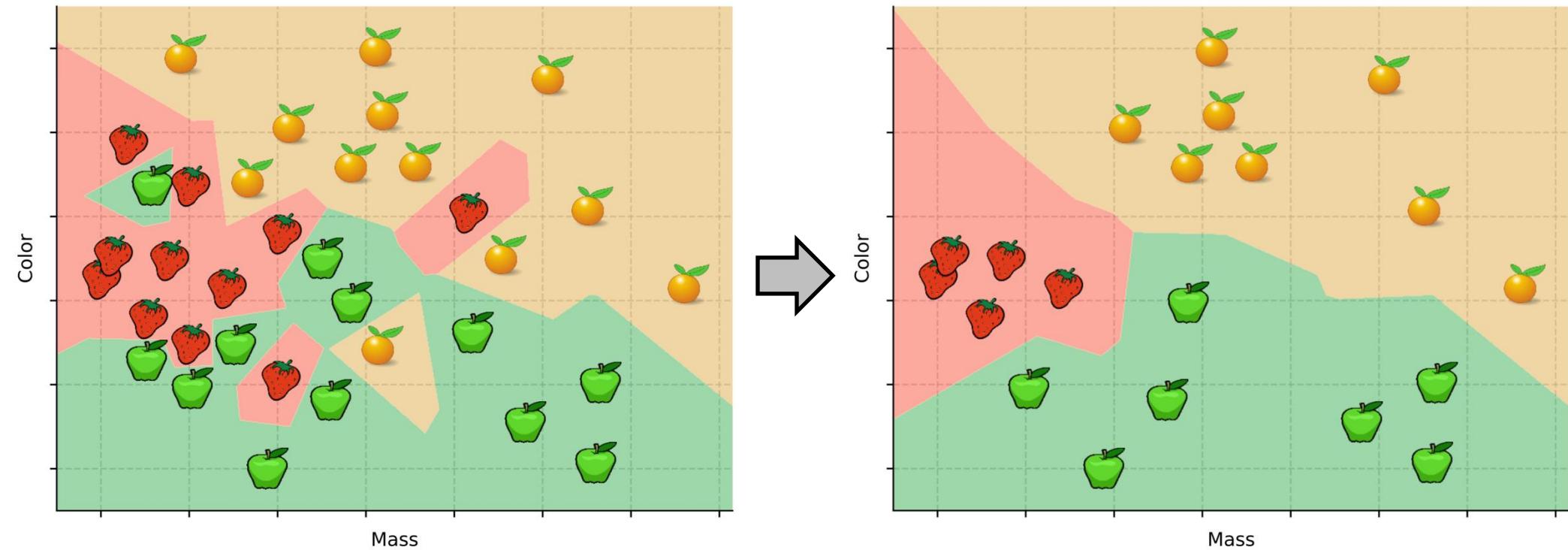
- 1) Find region containing test point

- 2) Compare to all points in region



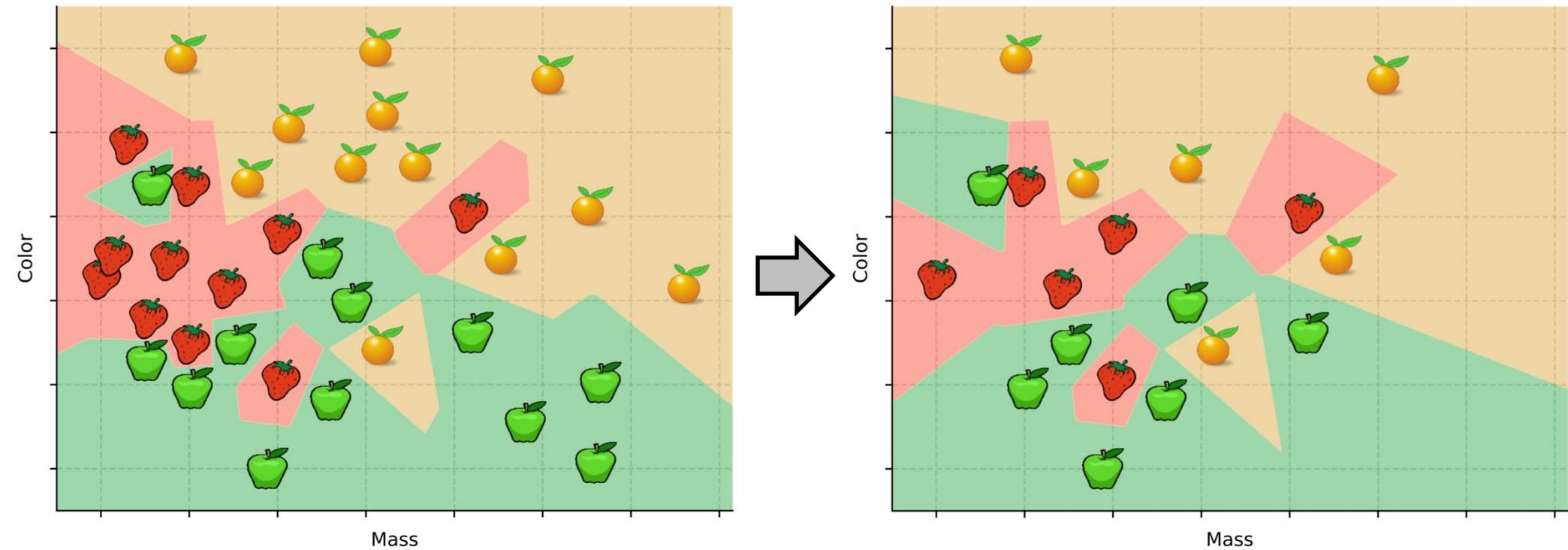
# Edited Nearest Neighbors

- Eliminate training samples that have a different label from its nearest k neighbors
- Regularization technique that produces smooth border by removing training outliers

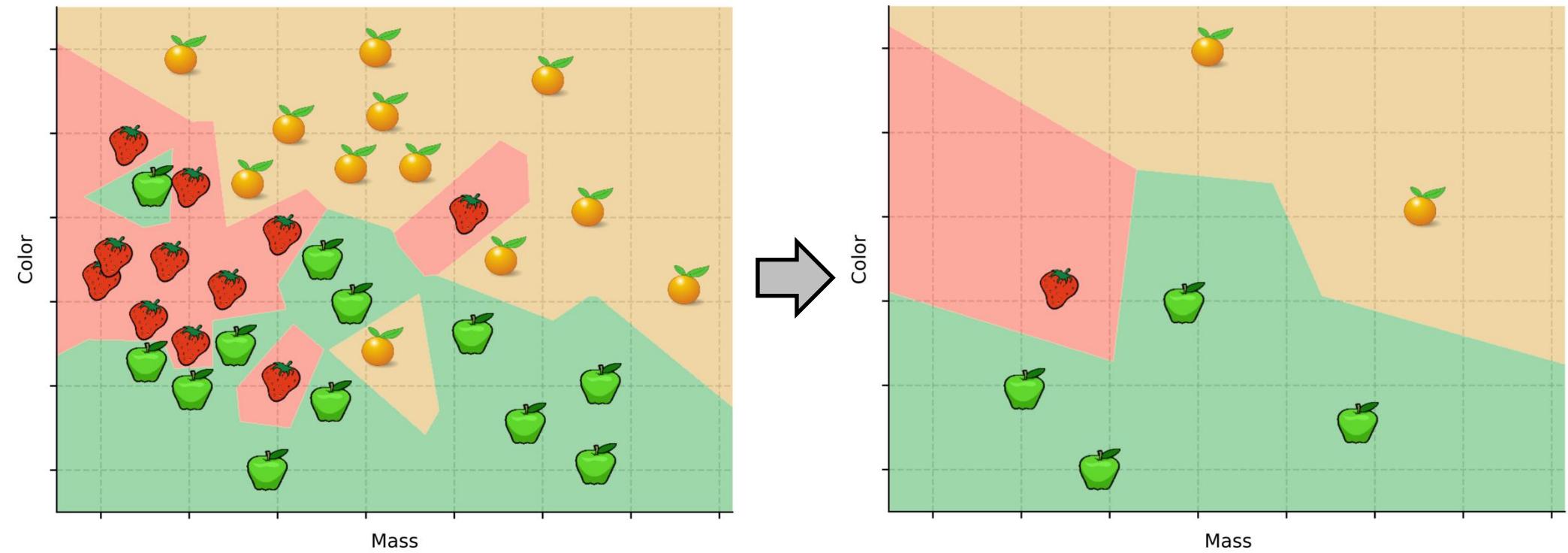


# Condensed Nearest Neighbors

- Eliminate training samples that have no effect on the decision boundary
- Algorithm:
  - 1) Randomly select a set  $P$  of samples, at least one for each class
  - 2) Each training sample that is not in  $P$  should be added to  $P$ , if the k-NN based on  $P$  classifies it incorrectly



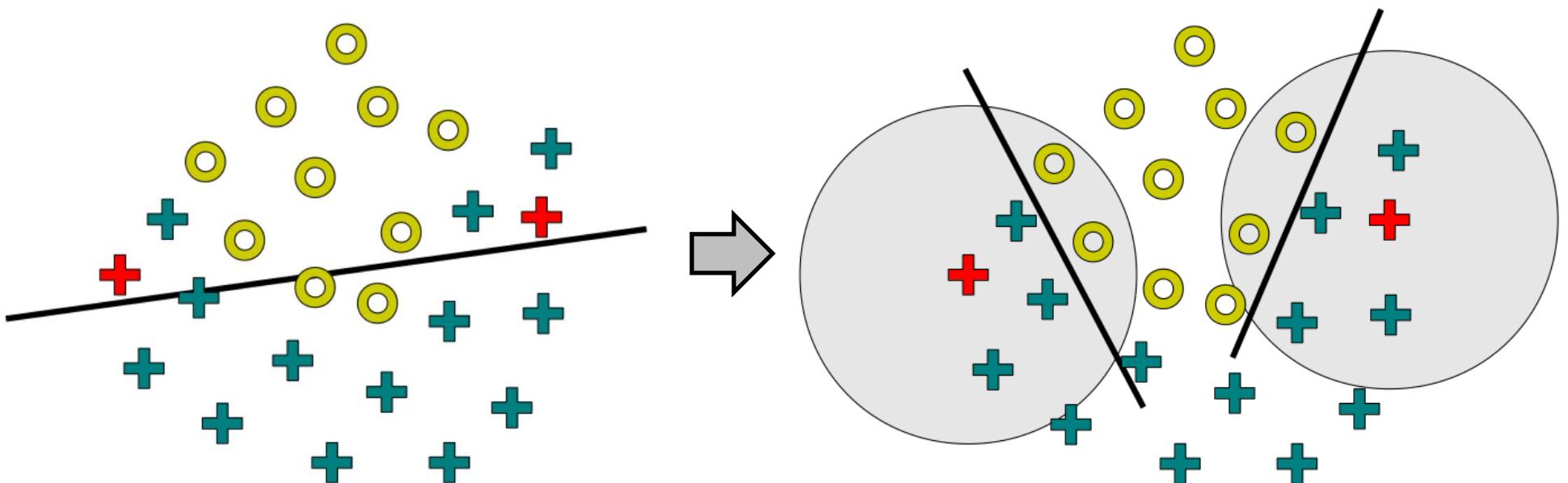
# Edited + Condensed Nearest Neighbors



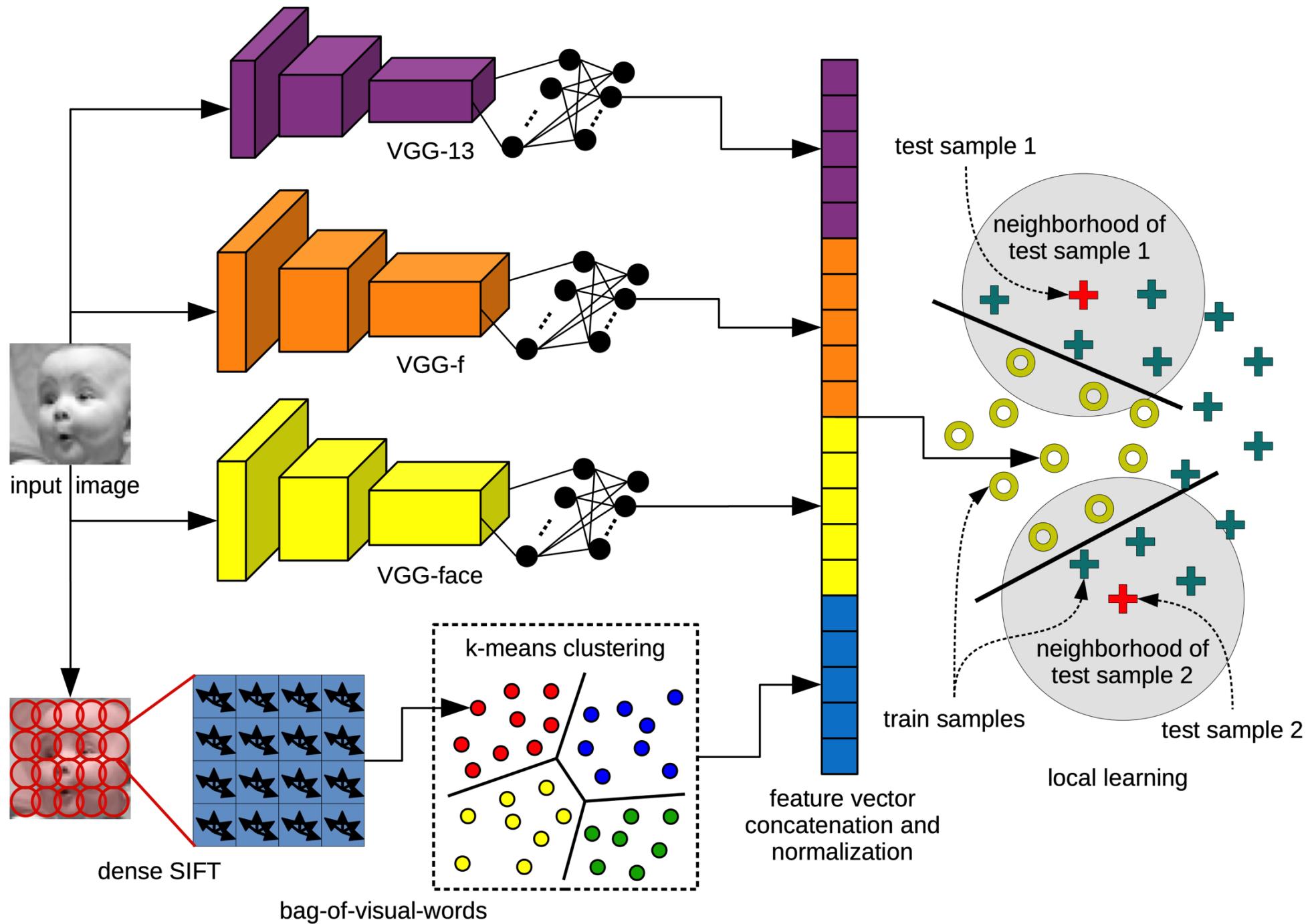
# Local Learning

# Local Learning

- Local learning algorithm:
  - 1) select training samples located in the vicinity of the given test sample
  - 2) train a classifier on the selected examples
  - 3) apply the classifier to predict the class label of the test sample



# Local Learning



# Curse of Dimensionality

# Curse of dimensionality

- In machine learning, we often use high-dimensional feature vectors
- Example:
  - When we analyze gray-scale images of 200x200 pixels, then we are working with a feature space of 40.000 dimensions
  - If the images are color (represented in the RGB or HSV color spaces), the dimensionality of the feature space grows to 120.000

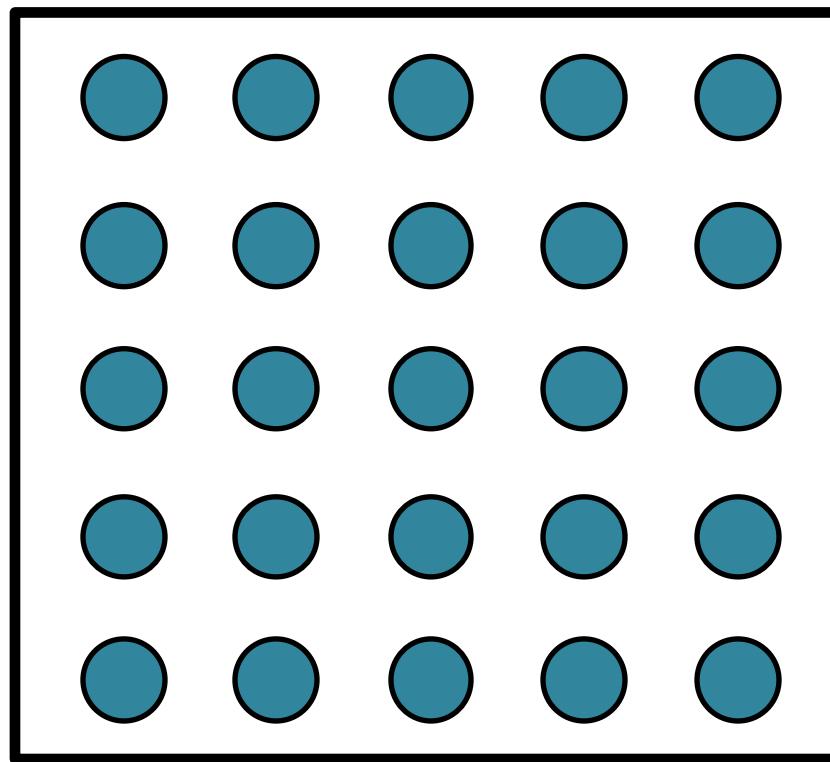
# Curse of dimensionality

- To “fill” a 1D space (e.g.  $\mathbb{R}^1$ ) we need 5 data points:



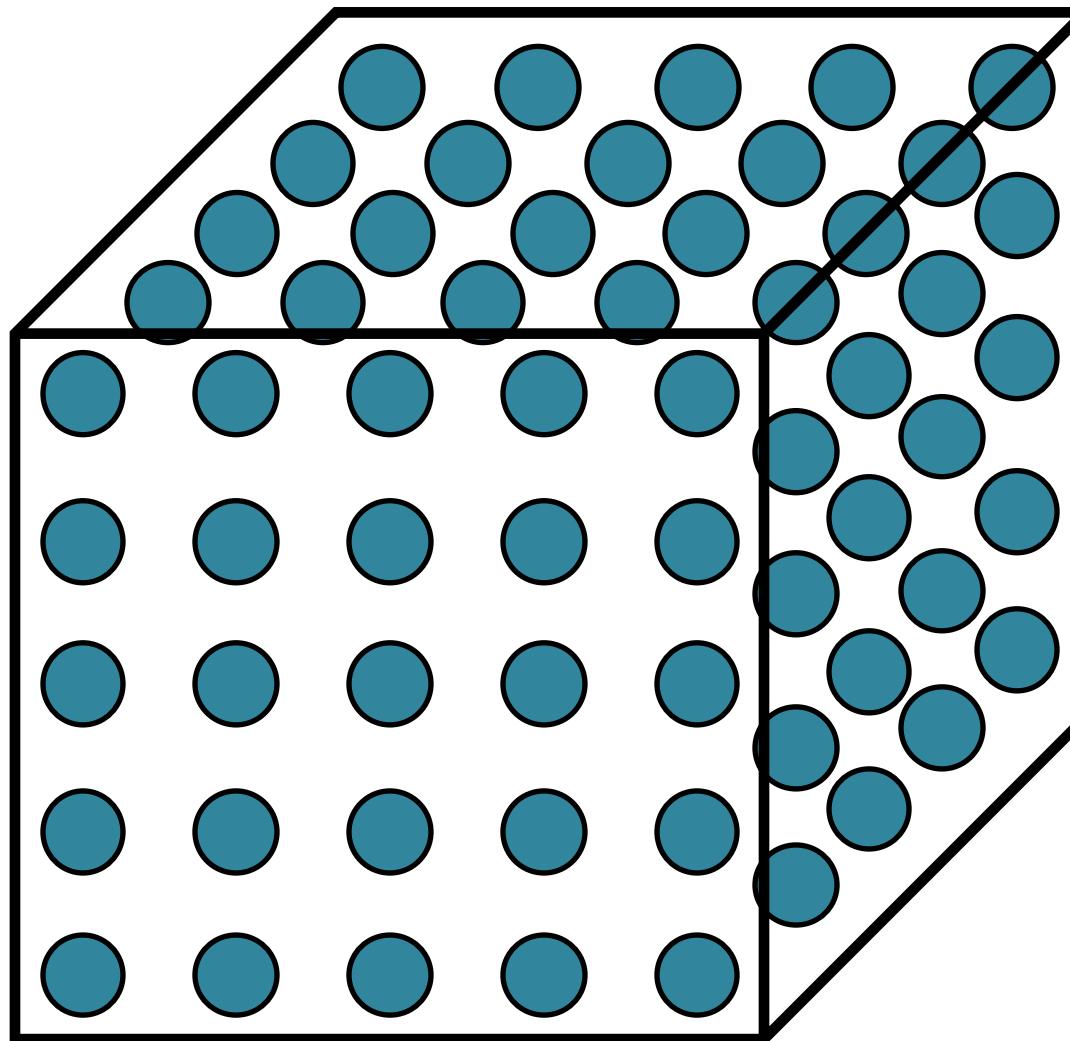
# Curse of dimensionality

- To “fill” a 2D space (e.g.  $\mathbb{R}^2$ ) we need 25 data points:



# Curse of dimensionality

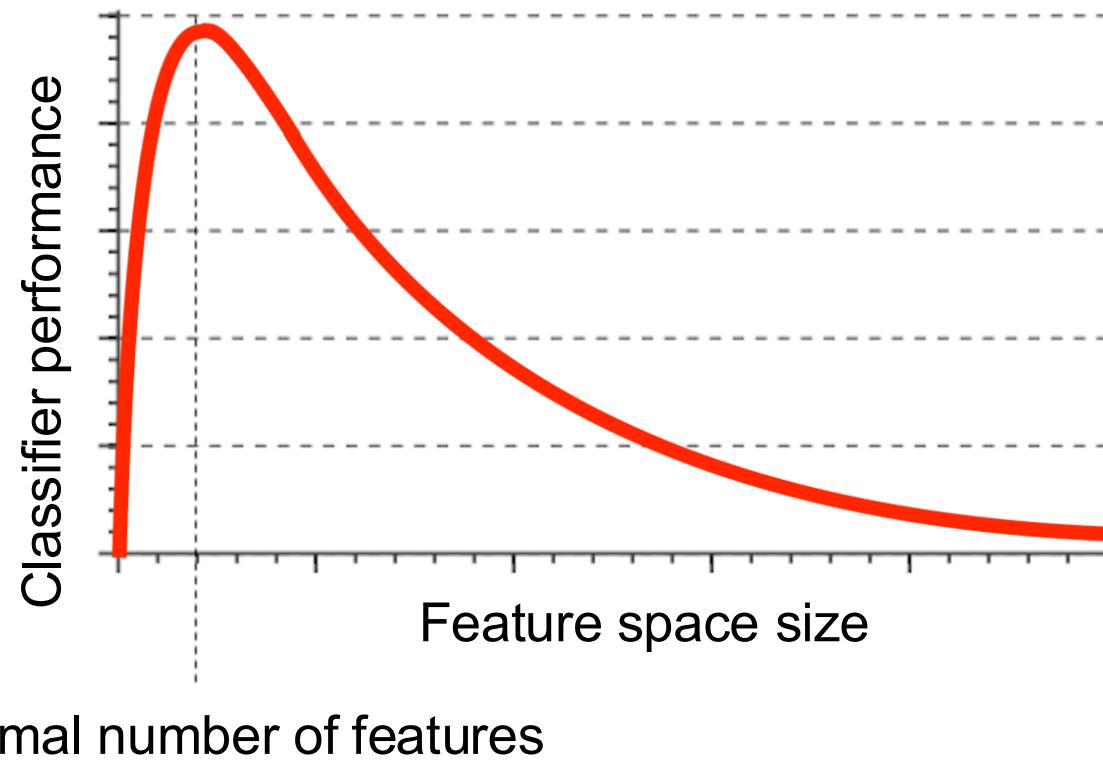
- To “fill” a 3D space (e.g.  $\mathbb{R}^3$ ) we need 125 data points:



# Curse of dimensionality

- To “fill” an nD space (e.g.  $\mathbb{R}^n$ ) we need an exponential number of data points
- If we have a large number of features that describe our data, then the system requires an extremely large training set in order to learn a model with good generalization
- Most of the times, large training sets are not available in practice

# Hughes phenomenon



- The Hughes phenomenon shows that, as the number of features grows, the performance of the classifier increases until we reach the optimal number of features
- Adding more features while keeping the size of the training set unchanged degrades the classifier performance

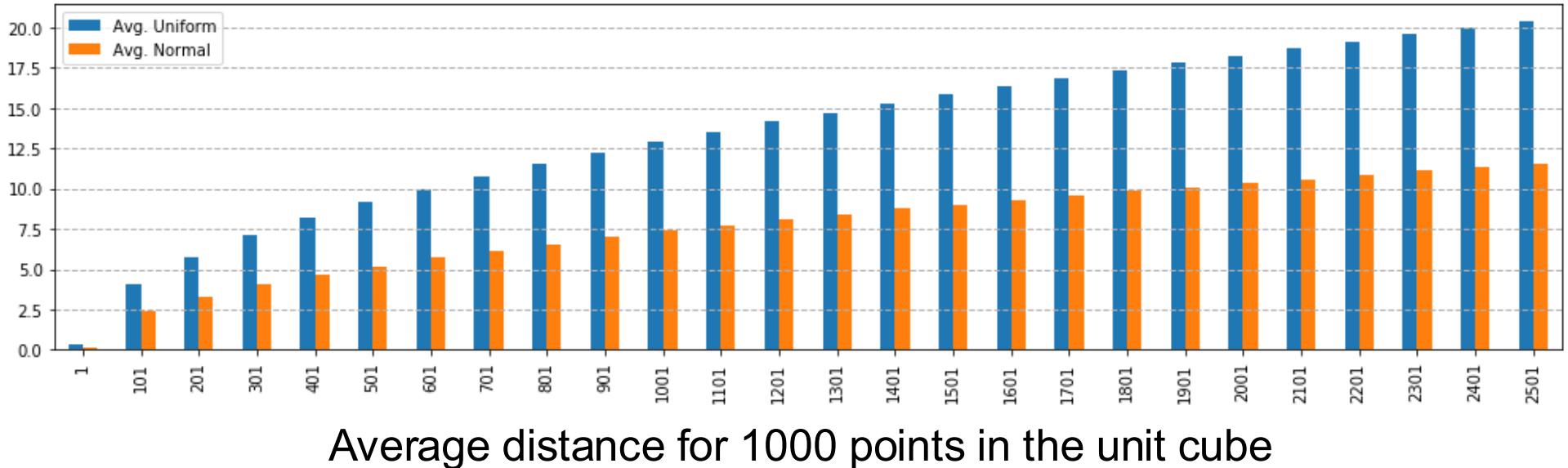
# Curse of dimensionality

- Increasing the dimension of an Euclidean feature space, implies the addition of positive terms in the computation of the Euclidean distance:

$$(x, y) = \sqrt{(x_1 - y_1)^2 + \cdots + (x_n - y_n)^2}$$

- In other words, because the number of features grows for a fixed number of data samples, the feature space becomes increasingly sparse (less dense)

# Curse of dimensionality



- By increasing the feature space size, the average distance grows rapidly
- Hence, the more dimensions we have, the more data is necessary to overcome the curse of dimensionality!
- When the distance between observations increases, the learning task becomes more difficult, since the probability of finding training samples that are truly similar with the test sample decreases