Cambridge AI+

Lecture 1 (Part 2): Nearest Neighbour

Thomas Sauerwald

University of Cambridge, Department of Computer Science
email: thomas.sauerwald@cl.cam.ac.uk

January 2021

Outline

Nearest Neighbour Algorithm

Additional Material

The Intuition behind Nearest Neighbour Algorithm

Idea of Nearest Neighbour

For any new (unseen) data point to be classified, find the most similar data points and make prediction based on these classes.

The Intuition behind Nearest Neighbour Algorithm

Idea of Nearest Neighbour

For any new (unseen) data point to be classified, find the most similar data points and make prediction based on these classes.

- Example: We want to classify fruits into 🍌, 🍊 and 🍎

The Intuition behind Nearest Neighbour Algorithm

Idea of Nearest Neighbour

For any new (unseen) data point to be classified, find the most similar data points and make prediction based on these classes.

- Example: We want to classify fruits into 🍌, 🍊 and 🍎
- We will measure the color (from yellow to red)

$\longrightarrow$ Colour

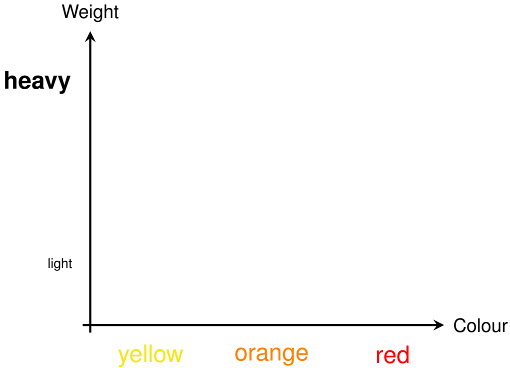
yellow orange red

The Intuition behind Nearest Neighbour Algorithm

Idea of Nearest Neighbour

For any new (unseen) data point to be classified, find the most similar data points and make prediction based on these classes.

- Example: We want to classify fruits into 🍌, 🍊 and 🍎
- We will measure the color (from yellow to red) and weight

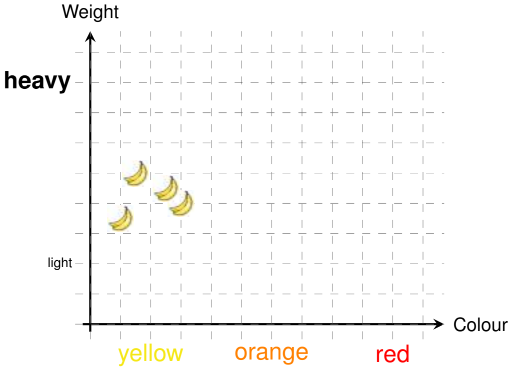


The Intuition behind Nearest Neighbour Algorithm

Idea of Nearest Neighbour

For any new (unseen) data point to be classified, find the most similar data points and make prediction based on these classes.

- Example: We want to classify fruits into 🍌, 🍊 and 🍎
- We will measure the color (from yellow to red) and weight

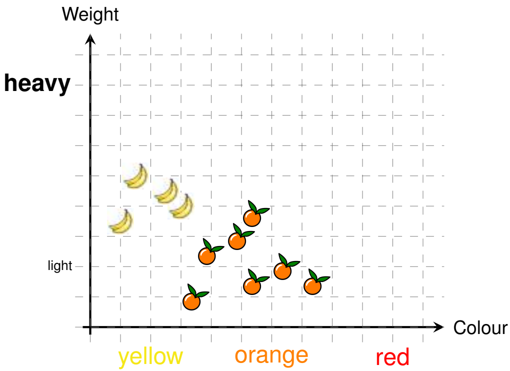


The Intuition behind Nearest Neighbour Algorithm

Idea of Nearest Neighbour

For any new (unseen) data point to be classified, find the most similar data points and make prediction based on these classes.

- Example: We want to classify fruits into 🍌, 🍊 and 🍎
- We will measure the color (from yellow to red) and weight

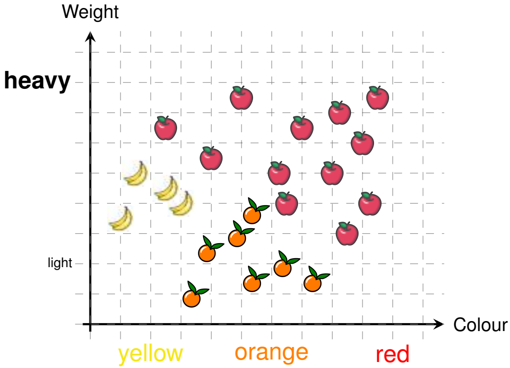


The Intuition behind Nearest Neighbour Algorithm

Idea of Nearest Neighbour

For any new (unseen) data point to be classified, find the most similar data points and make prediction based on these classes.

- Example: We want to classify fruits into 🍌, 🍊 and 🍎
- We will measure the color (from yellow to red) and weight

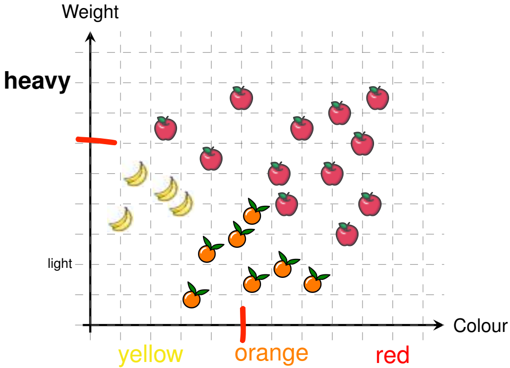


The Intuition behind Nearest Neighbour Algorithm

Idea of Nearest Neighbour

For any new (unseen) data point to be classified, find the most similar data points and make prediction based on these classes.

- Example: We want to classify fruits into 🍌, 🍊 and 🍎
- We will measure the color (from yellow to red) and weight
- We now see a fruit with colour 6 (orange) and weight 6 (quite heavy)

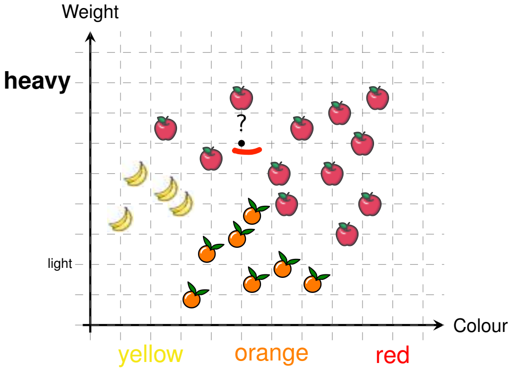


The Intuition behind Nearest Neighbour Algorithm

Idea of Nearest Neighbour

For any new (unseen) data point to be classified, find the most similar data points and make prediction based on these classes.

- Example: We want to classify fruits into 🍌, 🍊 and 🍎
- We will measure the color (from yellow to red) and weight
- We now see a fruit with colour 6 (orange) and weight 6 (quite heavy)

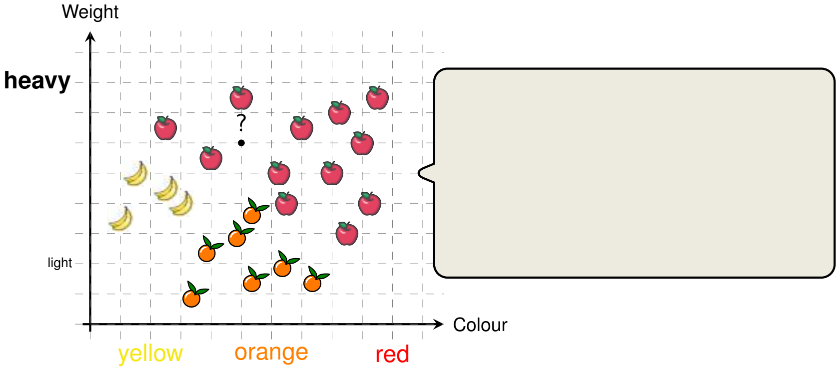


The Intuition behind Nearest Neighbour Algorithm

Idea of Nearest Neighbour

For any new (unseen) data point to be classified, find the most similar data points and make prediction based on these classes.

- Example: We want to classify fruits into 🍌, 🍊 and 🍎
- We will measure the color (from yellow to red) and weight
- We now see a fruit with colour 6 (orange) and weight 6 (quite heavy)

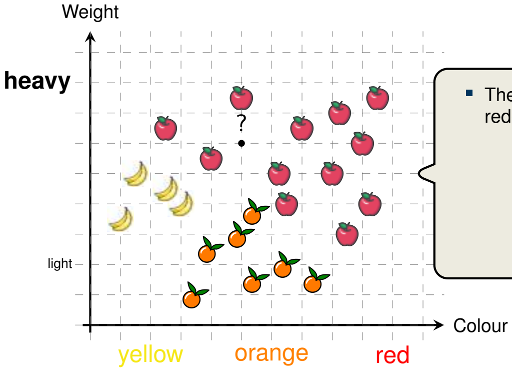


The Intuition behind Nearest Neighbour Algorithm

Idea of Nearest Neighbour

For any new (unseen) data point to be classified, find the most similar data points and make prediction based on these classes.

- Example: We want to classify fruits into 🍌, 🍊 and 🍎
- We will measure the color (from yellow to red) and weight
- We now see a fruit with colour 6 (orange) and weight 6 (quite heavy)



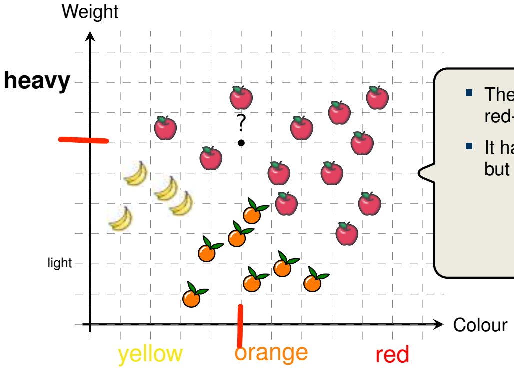
- The unknown fruit is heavier and more red-ish than all seen banana

The Intuition behind Nearest Neighbour Algorithm

Idea of Nearest Neighbour

For any new (unseen) data point to be classified, find the most similar data points and make prediction based on these classes.

- Example: We want to classify fruits into 🍌, 🍊 and 🍎
- We will measure the color (from yellow to red) and weight
- We now see a fruit with colour 6 (orange) and weight 6 (quite heavy)



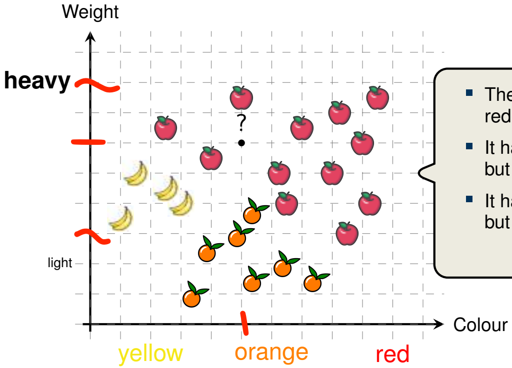
- The unknown fruit is heavier and more red-ish than all seen banana
- It has the (average) colour of an orange, but it is heavier

The Intuition behind Nearest Neighbour Algorithm

Idea of Nearest Neighbour

For any new (unseen) data point to be classified, find the most similar data points and make prediction based on these classes.

- Example: We want to classify fruits into 🍌, 🍊 and 🍎
- We will measure the color (from yellow to red) and weight
- We now see a fruit with colour 6 (orange) and weight 6 (quite heavy)



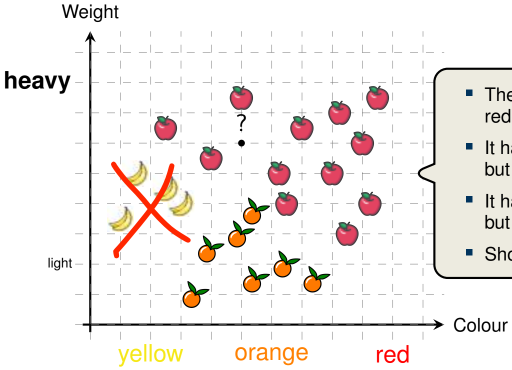
- The unknown fruit is heavier and more red-ish than all seen banana
- It has the (average) colour of an orange, but it is heavier
- It has the (average) weight of an apple, but it is more yellow-ish

The Intuition behind Nearest Neighbour Algorithm

Idea of Nearest Neighbour

For any new (unseen) data point to be classified, find the most similar data points and make prediction based on these classes.

- Example: We want to classify fruits into 🍌, 🍊 and 🍎
- We will measure the color (from yellow to red) and weight
- We now see a fruit with colour 6 (orange) and weight 6 (quite heavy)



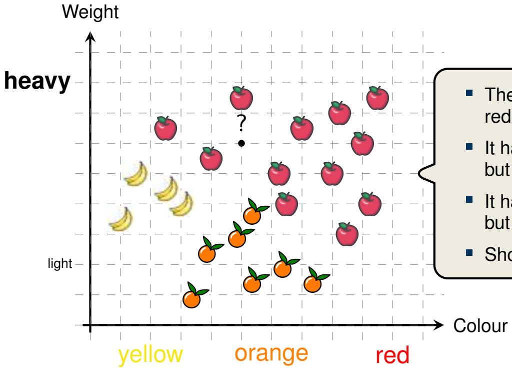
- The unknown fruit is heavier and more red-ish than all seen banana
- It has the (average) colour of an orange, but it is heavier
- It has the (average) weight of an apple, but it is more yellow-ish
- Should it be an apple or orange?

The Intuition behind Nearest Neighbour Algorithm

Idea of Nearest Neighbour

For any new (unseen) data point to be classified, find the most similar data points and make prediction based on these classes.

- Example: We want to classify fruits into 🍌, 🍊 and 🍎
- We will measure the color (from yellow to red) and weight
- We now see a fruit with colour 6 (orange) and weight 6 (quite heavy)



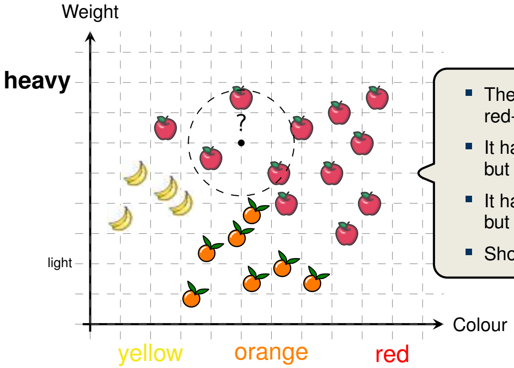
- The unknown fruit is heavier and more red-ish than all seen banana
- It has the (average) colour of an orange, but it is heavier
- It has the (average) weight of an apple, but it is more yellow-ish
- Should it be an apple or orange?

The Intuition behind Nearest Neighbour Algorithm

Idea of Nearest Neighbour

For any new (unseen) data point to be classified, find the most similar data points and make prediction based on these classes.

- Example: We want to classify fruits into ,  and 
- We will measure the color (from yellow to red) and weight
- We now see a fruit with colour 6 (orange) and weight 6 (quite heavy)



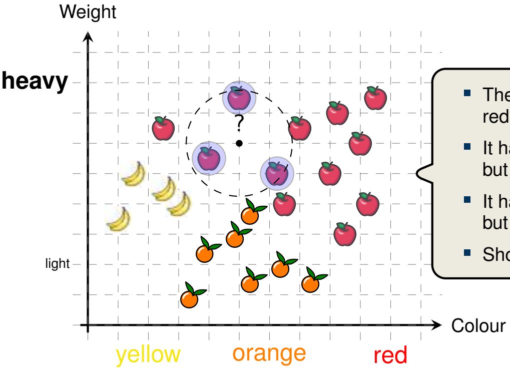
- The unknown fruit is heavier and more red-ish than all seen banana
- It has the (average) colour of an orange, but it is heavier
- It has the (average) weight of an apple, but it is more yellow-ish
- Should it be an apple or orange?

The Intuition behind Nearest Neighbour Algorithm

Idea of Nearest Neighbour

For any new (unseen) data point to be classified, find the most similar data points and make prediction based on these classes.

- Example: We want to classify fruits into 🍌, 🍊 and 🍎
- We will measure the color (from yellow to red) and weight
- We now see a fruit with colour 6 (orange) and weight 6 (quite heavy)



- The unknown fruit is heavier and more red-ish than all seen banana
- It has the (average) colour of an orange, but it is heavier
- It has the (average) weight of an apple, but it is more yellow-ish
- Should it be an apple or orange?

Towards the k-NN: A Simple Example in One Dimension

Input	Weight	Apple or Orange

Towards the k-NN: A Simple Example in One Dimension

Input	Weight	Apple or Orange
Training Point 1	0.4	$+$

Towards the k-NN: A Simple Example in One Dimension

Input	Weight	Apple or Orange
Training Point 1	0.4	$+$
Training Point 2	0.35	$-$

Towards the k-NN: A Simple Example in One Dimension

Input	Weight	Apple or Orange
Training Point 1	0.4	$+$
Training Point 2	0.35	$-$
Training Point 3	0.58	$+$

Towards the *k*-NN: A Simple Example in One Dimension

Input	Weight	Apple or Orange
Training Point 1	0.4	$+$
Training Point 2	0.35	$-$
Training Point 3	0.58	$+$
Training Point 4	0.2	$-$

Towards the k-NN: A Simple Example in One Dimension

Input	Weight	Apple or Orange
Training Point 1	0.4	$+$
Training Point 2	0.35	$-$
Training Point 3	0.58	$+$
Training Point 4	0.2	$-$
Training Point 5	0.7	$-$

Towards the k-NN: A Simple Example in One Dimension

Input	Weight	Apple or Orange
Training Point 1	0.4	$+$
Training Point 2	0.35	$-$
Training Point 3	0.58	$+$
Training Point 4	0.2	$-$
Training Point 5	0.7	$-$
Training Point 6	0.84	$+$

Towards the k-NN: A Simple Example in One Dimension

Input	Weight	Apple or Orange
Training Point 1	0.4	$+$
Training Point 2	0.35	$-$
Training Point 3	0.58	$+$
Training Point 4	0.2	$-$
Training Point 5	0.7	$-$
Training Point 6	0.84	$+$
Training Point 7	0.1	$+$

Towards the k-NN: A Simple Example in One Dimension

Input	Weight	Apple or Orange
Training Point 1	0.4	$+$
Training Point 2	0.35	$-$
Training Point 3	0.58	$+$
Training Point 4	0.2	$-$
Training Point 5	0.7	$-$
Training Point 6	0.84	$+$
Training Point 7	0.1	$+$
Training Point 8	0.75	$+$

Towards the k-NN: A Simple Example in One Dimension

Input	Weight	Apple or Orange
Training Point 1	0.4	$+$
Training Point 2	0.35	$-$
Training Point 3	0.58	$+$
Training Point 4	0.2	$-$
Training Point 5	0.7	$-$
Training Point 6	0.84	$+$
Training Point 7	0.1	$+$
Training Point 8	0.75	$+$
Training Point 9	0.25	$-$

Towards the k-NN: A Simple Example in One Dimension

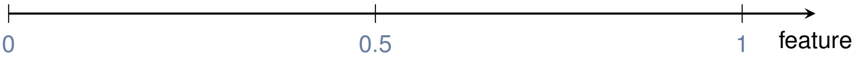
Input	Weight	Apple or Orange
Training Point 1	0.4	$+$
Training Point 2	0.35	$-$
Training Point 3	0.58	$+$
Training Point 4	0.2	$-$
Training Point 5	0.7	$-$
Training Point 6	0.84	$+$
Training Point 7	0.1	$+$
Training Point 8	0.75	$+$
Training Point 9	0.25	$-$
Training Point 10	0.05	$-$

Towards the k-NN: A Simple Example in One Dimension

Input	Weight	Apple or Orange
Training Point 1	0.4	$+$
Training Point 2	0.35	$-$
Training Point 3	0.58	$+$
Training Point 4	0.2	$-$
Training Point 5	0.7	$-$
Training Point 6	0.84	$+$
Training Point 7	0.1	$+$
Training Point 8	0.75	$+$
Training Point 9	0.25	$-$
Training Point 10	0.05	$-$
Test Point 11	0.16	??

Towards the *k*-NN: A Simple Example in One Dimension

Input	Weight	Apple or Orange
Training Point 1	0.4	$+$
Training Point 2	0.35	$-$
Training Point 3	0.58	$+$
Training Point 4	0.2	$-$
Training Point 5	0.7	$-$
Training Point 6	0.84	$+$
Training Point 7	0.1	$+$
Training Point 8	0.75	$+$
Training Point 9	0.25	$-$
Training Point 10	0.05	$-$
Test Point 11	0.16	??



Towards the k-NN: A Simple Example in One Dimension

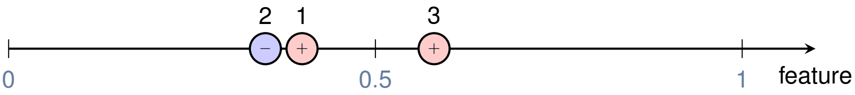
Input	Weight	Apple or Orange
Training Point 1	0.4	$+$
Training Point 2	0.35	$-$
Training Point 3	0.58	$+$
Training Point 4	0.2	$-$
Training Point 5	0.7	$-$
Training Point 6	0.84	$+$
Training Point 7	0.1	$+$
Training Point 8	0.75	$+$
Training Point 9	0.25	$-$
Training Point 10	0.05	$-$
Test Point 11	0.16	??

Towards the *k*-NN: A Simple Example in One Dimension

Input	Weight	Apple or Orange
Training Point 1	0.4	+
Training Point 2	0.35	−
Training Point 3	0.58	+
Training Point 4	0.2	−
Training Point 5	0.7	−
Training Point 6	0.84	+
Training Point 7	0.1	+
Training Point 8	0.75	+
Training Point 9	0.25	−
Training Point 10	0.05	−
Test Point 11	0.16	??

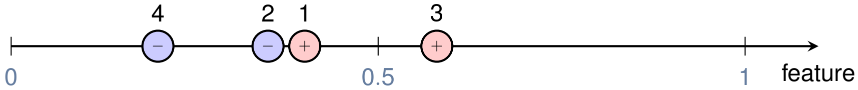
Towards the k-NN: A Simple Example in One Dimension

Input	Weight	Apple or Orange
Training Point 1	0.4	$+$
Training Point 2	0.35	$-$
Training Point 3	0.58	$+$
Training Point 4	0.2	$-$
Training Point 5	0.7	$-$
Training Point 6	0.84	$+$
Training Point 7	0.1	$+$
Training Point 8	0.75	$+$
Training Point 9	0.25	$-$
Training Point 10	0.05	$-$
Test Point 11	0.16	??



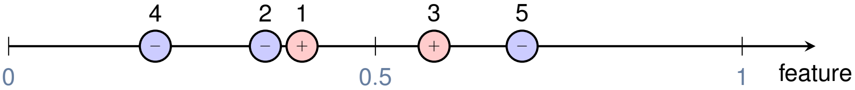
Towards the *k*-NN: A Simple Example in One Dimension

Input	Weight	Apple or Orange
Training Point 1	0.4	+
Training Point 2	0.35	−
Training Point 3	0.58	+
Training Point 4	0.2	−
Training Point 5	0.7	−
Training Point 6	0.84	+
Training Point 7	0.1	+
Training Point 8	0.75	+
Training Point 9	0.25	−
Training Point 10	0.05	−
Test Point 11	0.16	??



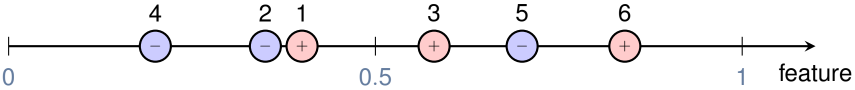
Towards the k-NN: A Simple Example in One Dimension

Input	Weight	Apple or Orange
Training Point 1	0.4	$+$
Training Point 2	0.35	$-$
Training Point 3	0.58	$+$
Training Point 4	0.2	$-$
Training Point 5	0.7	$-$
Training Point 6	0.84	$+$
Training Point 7	0.1	$+$
Training Point 8	0.75	$+$
Training Point 9	0.25	$-$
Training Point 10	0.05	$-$
Test Point 11	0.16	??



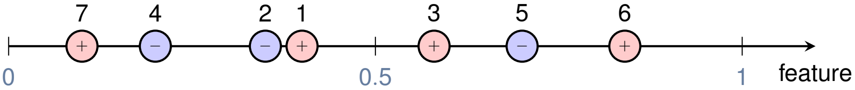
Towards the *k*-NN: A Simple Example in One Dimension

Input	Weight	Apple or Orange
Training Point 1	0.4	$+$
Training Point 2	0.35	$-$
Training Point 3	0.58	$+$
Training Point 4	0.2	$-$
Training Point 5	0.7	$-$
Training Point 6	0.84	$+$
Training Point 7	0.1	$+$
Training Point 8	0.75	$+$
Training Point 9	0.25	$-$
Training Point 10	0.05	$-$
Test Point 11	0.16	??



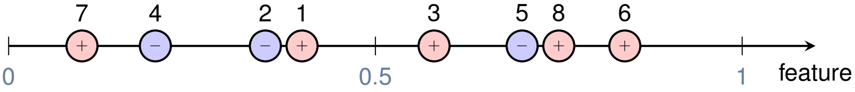
Towards the *k*-NN: A Simple Example in One Dimension

Input	Weight	Apple or Orange
Training Point 1	0.4	$+$
Training Point 2	0.35	$-$
Training Point 3	0.58	$+$
Training Point 4	0.2	$-$
Training Point 5	0.7	$-$
Training Point 6	0.84	$+$
Training Point 7	0.1	$+$
Training Point 8	0.75	$+$
Training Point 9	0.25	$-$
Training Point 10	0.05	$-$
Test Point 11	0.16	??



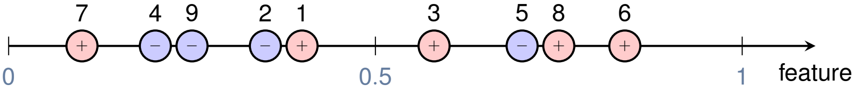
Towards the *k*-NN: A Simple Example in One Dimension

Input	Weight	Apple or Orange
Training Point 1	0.4	$+$
Training Point 2	0.35	$-$
Training Point 3	0.58	$+$
Training Point 4	0.2	$-$
Training Point 5	0.7	$-$
Training Point 6	0.84	$+$
Training Point 7	0.1	$+$
Training Point 8	0.75	$+$
Training Point 9	0.25	$-$
Training Point 10	0.05	$-$
Test Point 11	0.16	??



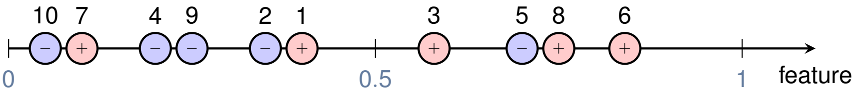
Towards the *k*-NN: A Simple Example in One Dimension

Input	Weight	Apple or Orange
Training Point 1	0.4	+
Training Point 2	0.35	−
Training Point 3	0.58	+
Training Point 4	0.2	−
Training Point 5	0.7	−
Training Point 6	0.84	+
Training Point 7	0.1	+
Training Point 8	0.75	+
Training Point 9	0.25	−
Training Point 10	0.05	−
Test Point 11	0.16	??



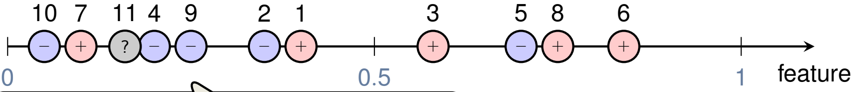
Towards the k-NN: A Simple Example in One Dimension

Input	Weight	Apple or Orange
Training Point 1	0.4	$+$
Training Point 2	0.35	$-$
Training Point 3	0.58	$+$
Training Point 4	0.2	$-$
Training Point 5	0.7	$-$
Training Point 6	0.84	$+$
Training Point 7	0.1	$+$
Training Point 8	0.75	$+$
Training Point 9	0.25	$-$
Training Point 10	0.05	$-$
Test Point 11	0.16	??



Towards the *k*-NN: A Simple Example in One Dimension

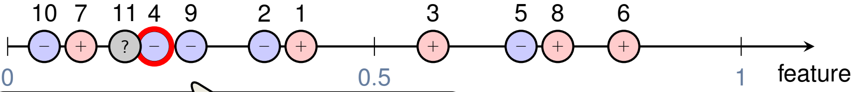
Input	Weight	Apple or Orange
Training Point 1	0.4	$+$
Training Point 2	0.35	$-$
Training Point 3	0.58	$+$
Training Point 4	0.2	$-$
Training Point 5	0.7	$-$
Training Point 6	0.84	$+$
Training Point 7	0.1	$+$
Training Point 8	0.75	$+$
Training Point 9	0.25	$-$
Training Point 10	0.05	$-$
Test Point 11	0.16	??



Quiz 1: How would you classify point 11?

Towards the k-NN: A Simple Example in One Dimension

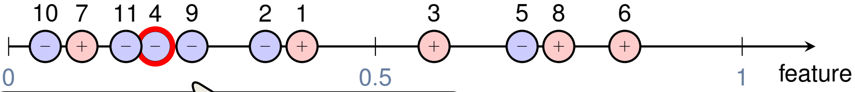
Input	Weight	Apple or Orange
Training Point 1	0.4	$+$
Training Point 2	0.35	$-$
Training Point 3	0.58	$+$
Training Point 4	0.2	$-$
Training Point 5	0.7	$-$
Training Point 6	0.84	$+$
Training Point 7	0.1	$+$
Training Point 8	0.75	$+$
Training Point 9	0.25	$-$
Training Point 10	0.05	$-$
Test Point 11	0.16	??



Quiz 1: How would you classify point 11?

Towards the k-NN: A Simple Example in One Dimension

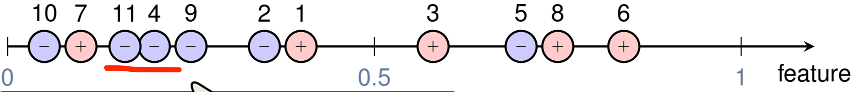
Input	Weight	Apple or Orange
Training Point 1	0.4	$+$
Training Point 2	0.35	$-$
Training Point 3	0.58	$+$
Training Point 4	0.2	$-$
Training Point 5	0.7	$-$
Training Point 6	0.84	$+$
Training Point 7	0.1	$+$
Training Point 8	0.75	$+$
Training Point 9	0.25	$-$
Training Point 10	0.05	$-$
Test Point 11	0.16	??



Quiz 1: How would you classify point 11?

Towards the k-NN: A Simple Example in One Dimension

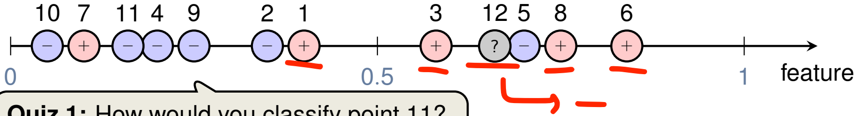
Input	Weight	Apple or Orange
Training Point 1	0.4	$+$
Training Point 2	0.35	$-$
Training Point 3	0.58	$+$
Training Point 4	0.2	$-$
Training Point 5	0.7	$-$
Training Point 6	0.84	$+$
Training Point 7	0.1	$+$
Training Point 8	0.75	$+$
Training Point 9	0.25	$-$
Training Point 10	0.05	$-$
Test Point 11	0.16	$-$



Quiz 1: How would you classify point 11?

Towards the k-NN: A Simple Example in One Dimension

Input	Weight	Apple or Orange
Training Point 1	0.4	$+$
Training Point 2	0.35	$-$
Training Point 3	0.58	$+$
Training Point 4	0.2	$-$
Training Point 5	0.7	$-$
Training Point 6	0.84	$+$
Training Point 7	0.1	$+$
Training Point 8	0.75	$+$
Training Point 9	0.25	$-$
Training Point 10	0.05	$-$
Test Point 11	0.16	$-$



Quiz 1: How would you classify point 11?

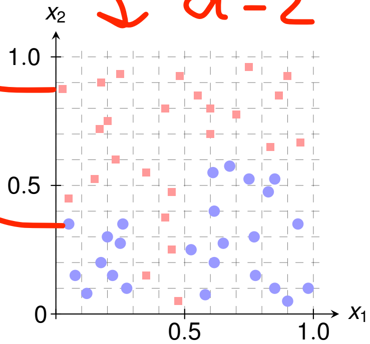
The Setup (A bit more formal...)

- Let $\mathcal{X} = \mathbb{R}^d$ (domain set)
- Let $\mathcal{Y} = \{-1, +1\}$ (label set)
- Let $S = ((\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_m, y_m))$ (training set)

$+1$

-1

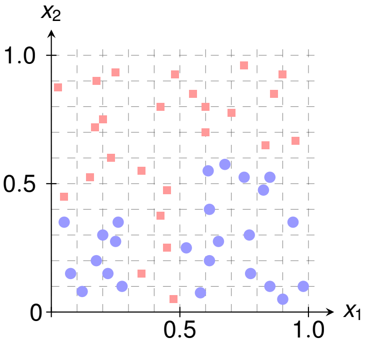
$d=2$



$\mathcal{X} = [0,1]^2$

The Setup (A bit more formal...)

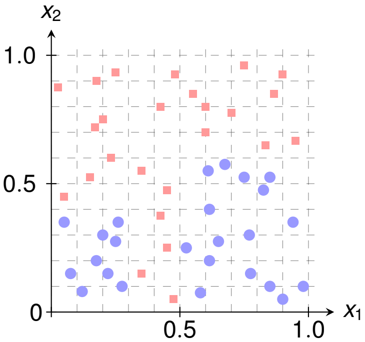
- Let $\mathcal{X} = \mathbb{R}^d$ (domain set)
- Let $\mathcal{Y} = \{-1, +1\}$ (label set)
- Let $S = ((\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_m, y_m))$ (training set)



Goal: Find a predictor $h : \mathcal{X} \to \mathcal{Y}$, which labels any unseen data point.

The Setup (A bit more formal...)

- Let $\mathcal{X} = \mathbb{R}^d$ (domain set)
- Let $\mathcal{Y} = \{-1, +1\}$ (label set)
- Let $S = ((\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_m, y_m))$ (training set)

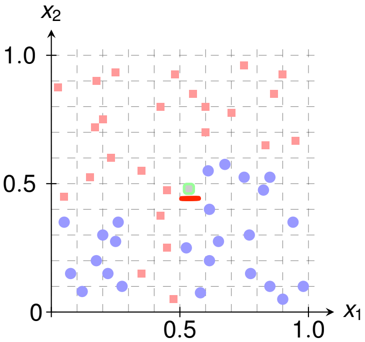


Goal: Find a predictor $h : \mathcal{X} \to \mathcal{Y}$, which labels any unseen data point.

sometimes also called classifier or prediction rule

The Setup (A bit more formal...)

- Let $\mathcal{X} = \mathbb{R}^d$ (domain set)
- Let $\mathcal{Y} = \{-1, +1\}$ (label set)
- Let $S = ((\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_m, y_m))$ (training set)

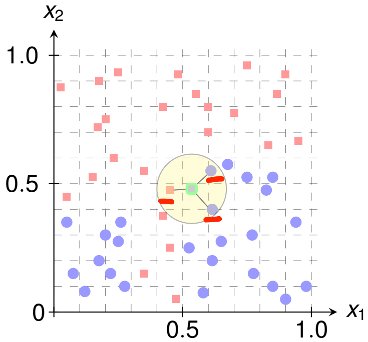


Goal: Find a predictor $h : \mathcal{X} \to \mathcal{Y}$, which labels any unseen data point.

sometimes also called classifier or prediction rule

The Setup (A bit more formal...)

- Let $\mathcal{X} = \mathbb{R}^d$ (domain set)
- Let $\mathcal{Y} = \{-1, +1\}$ (label set)
- Let $S = ((\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_m, y_m))$ (training set)



Goal: Find a predictor $h : \mathcal{X} \to \mathcal{Y}$, which labels any unseen data point.

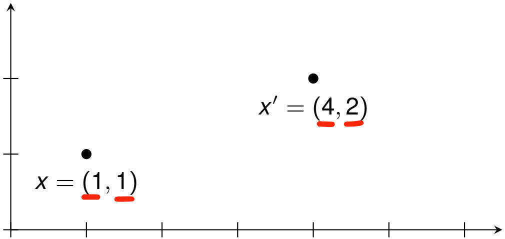
sometimes also called classifier or prediction rule

The k-NN Algorithm

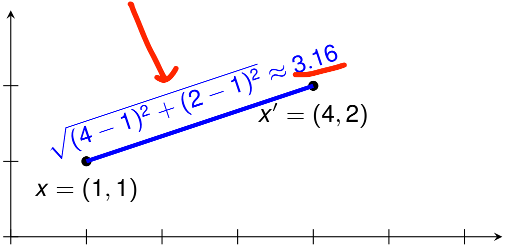
- Let $d(\mathbf{x}, \mathbf{x}') := \sqrt{\sum_{i=1}^{d} (x_i - x_i')^2}$ be the Euclidean Distance

The k-NN Algorithm

- Let $d(\mathbf{x}, \mathbf{x}') := \sqrt{\sum_{i=1}^{d} (x_i - x_i')^2}$ be the Euclidean Distance



- Let $d(\mathbf{x}, \mathbf{x}') := \sqrt{\sum_{i=1}^{d} (x_i - x_i')^2}$ be the Euclidean Distance



The k-NN Algorithm

- Let $d(\mathbf{x}, \mathbf{x}') := \sqrt{\sum_{i=1}^{d} (x_i - x_i')^2}$ be the Euclidean Distance ✓
- For every $\mathbf{x} \in \mathcal{X}$, let $\pi_1(\mathbf{x}), \ldots, \pi_m(\mathbf{x})$ be a reordering of $\{1, 2, \ldots, m\}$ according to their distance to $\mathbf{x}$, i.e., for every $1 \leq i < m$:

$$d(\mathbf{x}, x_{\pi_i(\mathbf{x})}) \leq d(\mathbf{x}, x_{\pi_{i+1}(\mathbf{x})}).$$

The k-NN Algorithm

- Let $d(\mathbf{x}, \mathbf{x}') := \sqrt{\sum_{i=1}^{d} (x_i - x_i')^2}$ be the Euclidean Distance
- For every $\mathbf{x} \in \mathcal{X}$, let $\pi_1(\mathbf{x}), \ldots, \pi_m(\mathbf{x})$ be a reordering of $\{1, 2, \ldots, m\}$ according to their distance to $\mathbf{x}$, i.e., for every $1 \leq i < m$:

$$d(\mathbf{x}, x_{\pi_i(\mathbf{x})}) \leq d(\mathbf{x}, x_{\pi_{i+1}(\mathbf{x})}).$$

For each point $\mathbf{x}$, we sort the elements in training set increasingly in their distance to $\mathbf{x}$.

The k-NN Algorithm

- Let $d(\mathbf{x}, \mathbf{x}') := \sqrt{\sum_{i=1}^{d} (x_i - x_i')^2}$ be the Euclidean Distance
- For every $\mathbf{x} \in \mathcal{X}$, let $\pi_1(\mathbf{x}), \ldots, \pi_m(\mathbf{x})$ be a reordering of $\{1, 2, \ldots, m\}$ according to their distance to $\mathbf{x}$, i.e., for every $1 \leq i < m$:

$$\boxed{d(\mathbf{x}, x_{\pi_i(\mathbf{x})}) \leq d(\mathbf{x}, x_{\pi_{i+1}(\mathbf{x})}).}$$

For each point $\mathbf{x}$, we sort the elements in training set increasingly in their distance to $\mathbf{x}$.

k-NN

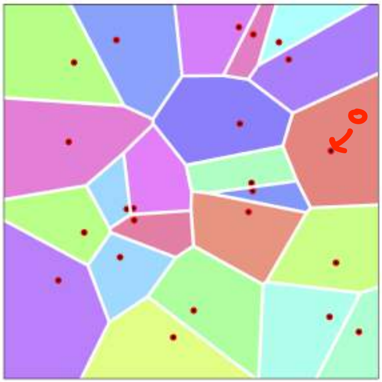
input: a training sample $S = (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$
output: for every point $\mathbf{x} \in \mathcal{X}$,
return the majority label among $\boxed{\{y_{\pi_i(\mathbf{x})} : i \leq k\}}$

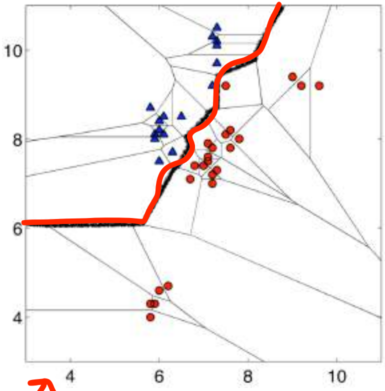
Source: SS&BD

role k!

Special Case: 1-NN

$\boxed{K} = 1$, $d = 2$

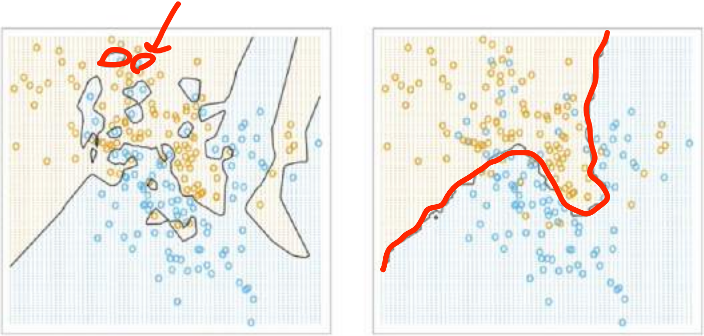




Source: Zemel, Urtasun, Fidler

- For $k = 1$, the produced decision boundaries are Voronoi-cells
- Any new point will be classified according to the centre of each cell

How many Neighbours should we choose?

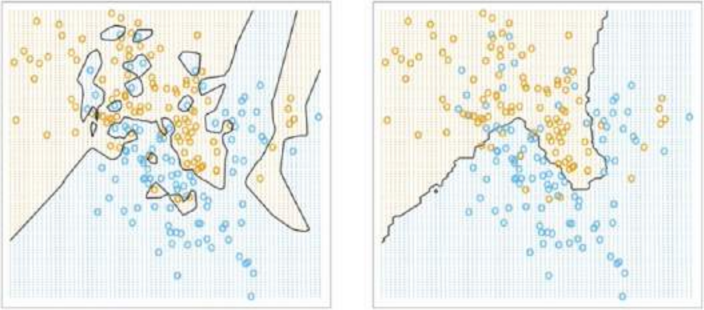


Source: Lecture by Ulrike von Luxburg

$k = 1$ $k = 15$

How many Neighbours should we choose?

For small k, k-NN overfits the data!



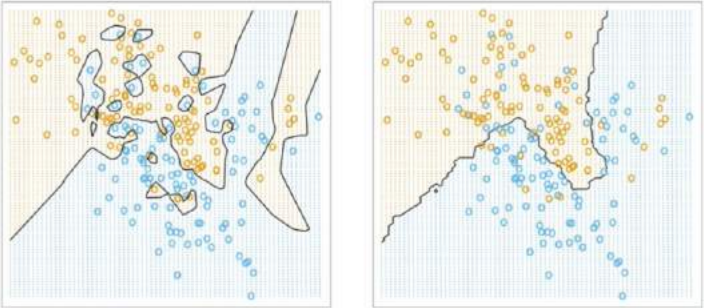
Source: Lecture by Ulrike von Luxburg

$k = 1$ $k = 15$

How many Neighbours should we choose?

For small k, k-NN overfits the data!

Rule-of-thumb: $k \approx \log m$ is good



Source: Lecture by Ulrike von Luxburg

$k = 1$ $\qquad$ $k = 15$

How many Neighbours should we choose?

For small k, k-NN overfits the data!

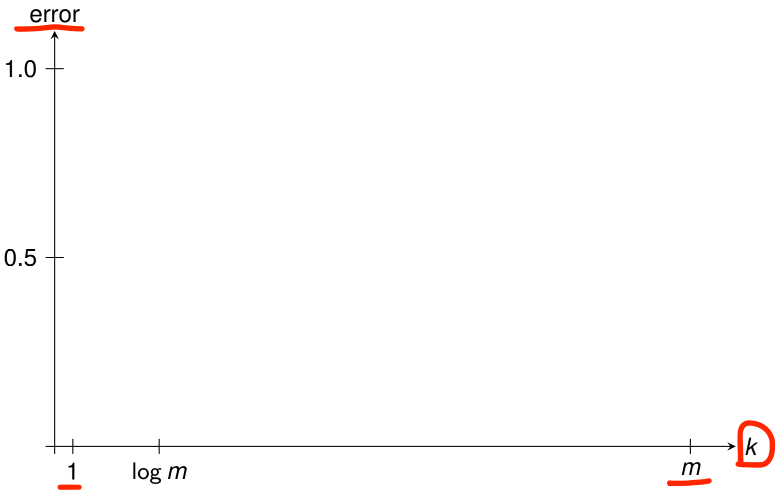
Rule-of-thumb: $k \approx \log m$ is good

Source: Lecture by Ulrike von Luxburg

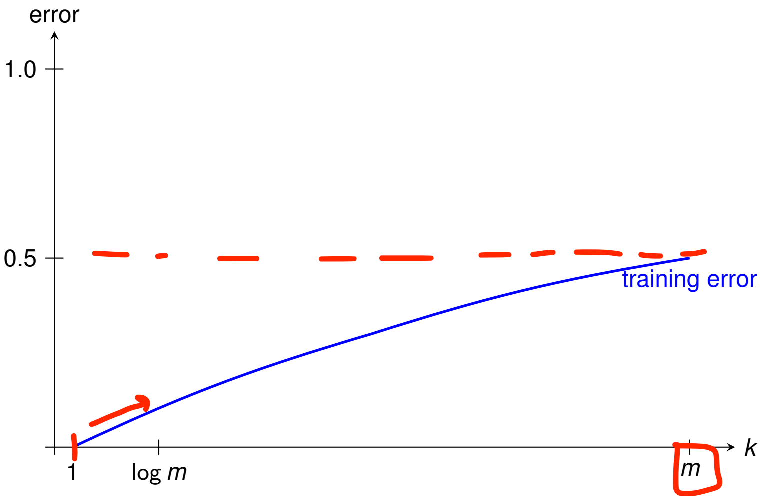
$k = 1$ $k = 15$

Quiz 2: What happens if $k = m$, where m is the total number of points in our training set?

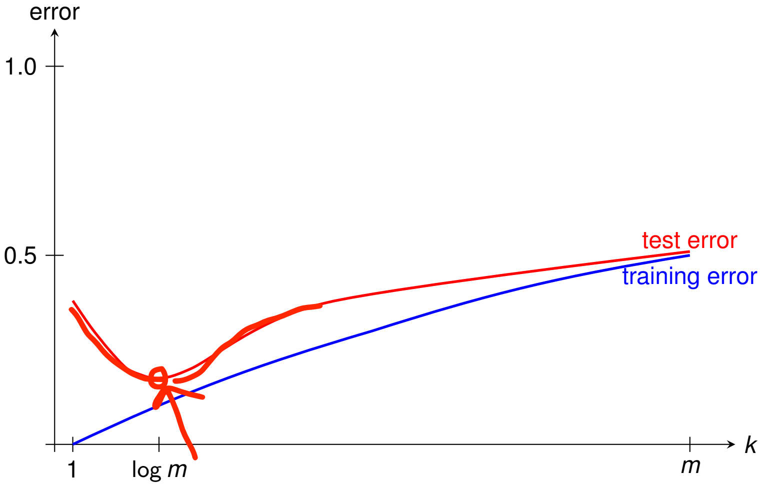
Training Error, Test Error and Overfitting



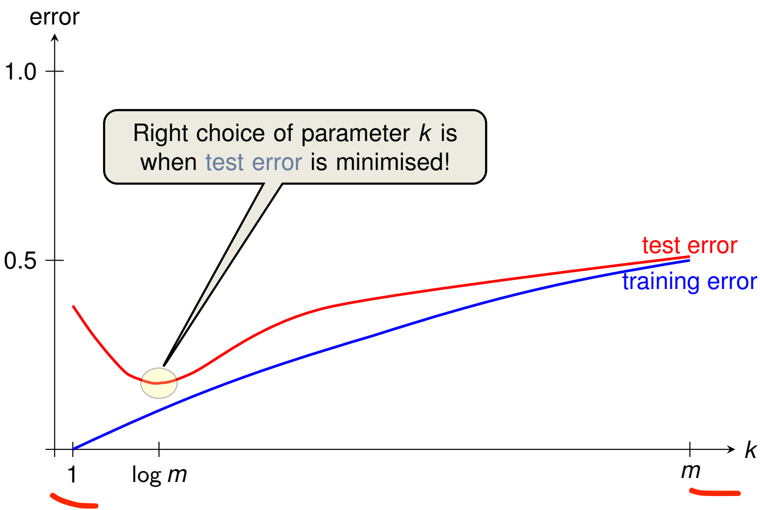
Training Error, Test Error and Overfitting



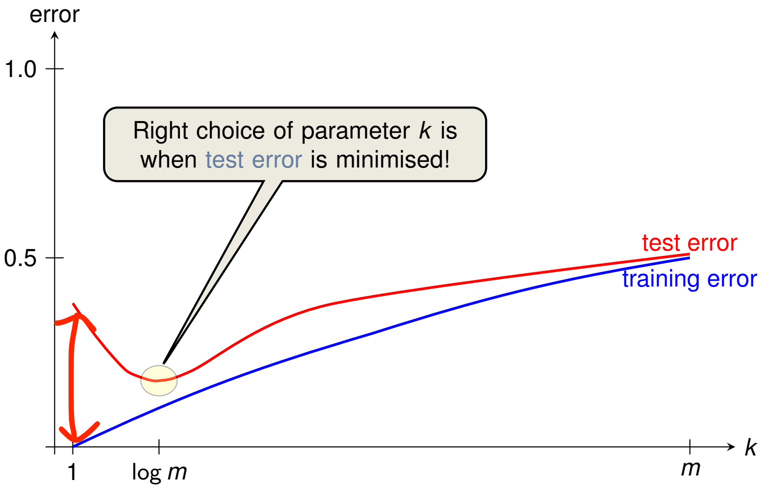
Training Error, Test Error and Overfitting



Training Error, Test Error and Overfitting

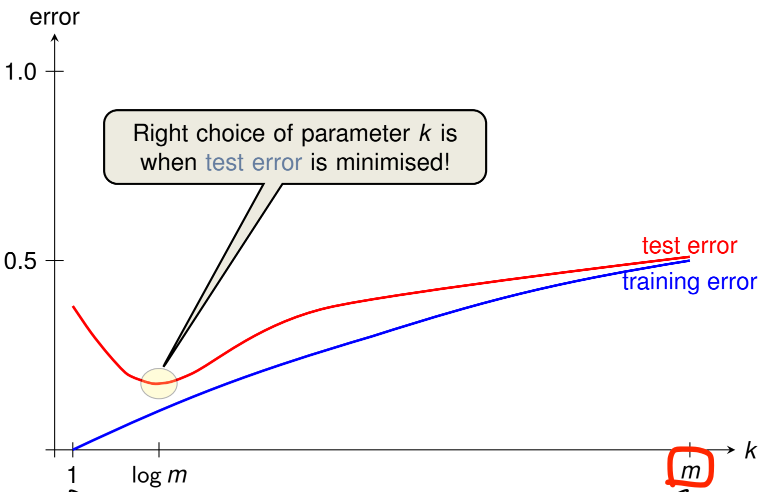


Training Error, Test Error and Overfitting



Overfitting: Small Training Error and (too) Complex Prediction Rule

Training Error, Test Error and Overfitting



Overfitting: Small Training Error and (too) Complex Prediction Rule

Underfitting: High Training Error and (too) Simple Prediction Rule

Feature Weighting and Distance Weighting

Feature Weighting

- k-NN is not scaling invariant, i.e., multiplying the the same feature of each data point may affect the results

Feature Weighting and Distance Weighting

Feature Weighting

- k-NN is not scaling invariant, i.e., multiplying the the same feature of each data point may affect the results

$\Rightarrow$ Normalise all features so that the square deviation from mean is the same for all d features

Feature Weighting and Distance Weighting

Feature Weighting

- k-NN is not scaling invariant, i.e., multiplying the the same feature of each data point may affect the results

$\Rightarrow$ Normalise all features so that the square deviation from mean is the same for all d features

Distance Weighting

- In k-NN, we simply take majority over the k nearest neighbour without taking distance into account

Feature Weighting and Distance Weighting

Feature Weighting

- k-NN is not scaling invariant, i.e., multiplying the the same feature of each data point may affect the results

$\Rightarrow$ Normalise all features so that the square deviation from mean is the same for all d features

Distance Weighting

- In k-NN, we simply take majority over the k nearest neighbour without taking distance into account
- There are several weighting-schemes:

Feature Weighting and Distance Weighting

Feature Weighting

- k-NN is not scaling invariant, i.e., multiplying the the same feature of each data point may affect the results

$\Rightarrow$ Normalise all features so that the square deviation from mean is the same for all d features

Distance Weighting

- In k-NN, we simply take majority over the k nearest neighbour without taking distance into account
- There are several weighting-schemes:
  1. Harmonic Weighting: i-th closest point is weighted by $1/i$

Feature Weighting and Distance Weighting

Feature Weighting

- k-NN is not scaling invariant, i.e., multiplying the the same feature of each data point may affect the results

$\Rightarrow$ Normalise all features so that the square deviation from mean is the same for all d features

Distance Weighting

- In k-NN, we simply take majority over the k nearest neighbour without taking distance into account
- There are several weighting-schemes:
  1. Harmonic Weighting: i-th closest point is weighted by $1/i$
  2. Inverse Distance: i-th closest point is weighted by $1/\operatorname{dist}(\mathbf{x}_i, \mathbf{x})$

Feature Weighting and Distance Weighting

Feature Weighting

- k-NN is not scaling invariant, i.e., multiplying the the same feature of each data point may affect the results

$\Rightarrow$ Normalise all features so that the square deviation from mean is the same for all d features

Distance Weighting

- In k-NN, we simply take majority over the k nearest neighbour without taking distance into account
- There are several weighting-schemes:
  1. Harmonic Weighting: i-th closest point is weighted by $1/i$
  2. Inverse Distance: i-th closest point is weighted by $1/\operatorname{dist}(\mathbf{x}_i, \mathbf{x})$
  3. Inverse Square Distance: i-th closest point is weighted by $1/\operatorname{dist}(\mathbf{x}_i, \mathbf{x})^2$

Feature Weighting and Distance Weighting

Feature Weighting

- k-NN is not scaling invariant, i.e., multiplying the the same feature of each data point may affect the results

$\Rightarrow$ Normalise all features so that the square deviation from mean is the same for all d features

Distance Weighting

- In k-NN, we simply take majority over the k nearest neighbour without taking distance into account
- There are several weighting-schemes:
  1. Harmonic Weighting: i-th closest point is weighted by $1/i$
  2. Inverse Distance: i-th closest point is weighted by $1/\operatorname{dist}(\mathbf{x}_i, \mathbf{x})$
  3. Inverse Square Distance: i-th closest point is weighted by $1/\operatorname{dist}(\mathbf{x}_i, \mathbf{x})^2$
- Alternatively, one can take Gaussian Weighting over all points: each point x_i is weighted by $\exp(-c \cdot (x_i - \mathbf{x})^2)$

Feature Weighting and Distance Weighting

Feature Weighting

- k-NN is not scaling invariant, i.e., multiplying the the same feature of each data point may affect the results

$\Rightarrow$ Normalise all features so that the square deviation from mean is the same for all d features

Distance Weighting

- In k-NN, we simply take majority over the k nearest neighbour without taking distance into account
- There are several weighting-schemes:
  1. Harmonic Weighting: i-th closest point is weighted by $1/i$
  2. Inverse Distance: i-th closest point is weighted by $1/\operatorname{dist}(\mathbf{x}_i, \mathbf{x})$
  3. Inverse Square Distance: i-th closest point is weighted by $1/\operatorname{dist}(\mathbf{x}_i, \mathbf{x})^2$
- Alternatively, one can take Gaussian Weighting over all points: each point x_i is weighted by $\exp(-c \cdot (x_i - \mathbf{x})^2)$

No need to choose k, but c controls sensitivity of nearby points.

Feature Weighting and Distance Weighting

Feature Weighting

- k-NN is not scaling invariant, i.e., multiplying the the same feature of each data point may affect the results

$\Rightarrow$ Normalise all features so that the square deviation from mean is the same for all d features

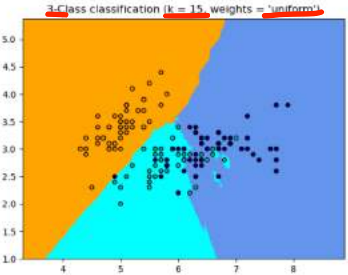
Distance Weighting

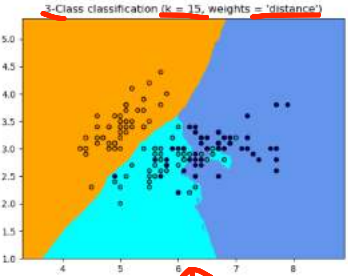
- In k-NN, we simply take majority over the k nearest neighbour without taking distance into account
- There are several weighting-schemes:
  1. Harmonic Weighting: i-th closest point is weighted by $1/i$
  2. Inverse Distance: i-th closest point is weighted by $1/\operatorname{dist}(\mathbf{x}_i, \mathbf{x})$
  3. Inverse Square Distance: i-th closest point is weighted by $1/\operatorname{dist}(\mathbf{x}_i, \mathbf{x})^2$
- Alternatively, one can take Gaussian Weighting over all points: each point x_i is weighted by $\exp(-c \cdot (x_i - \mathbf{x})^2)$

No need to choose k, but c controls sensitivity of nearby points.

We see some examples of Gaussian Weighting in next lecture!

Uniform Weighting vs. Inverse Distance Weighting

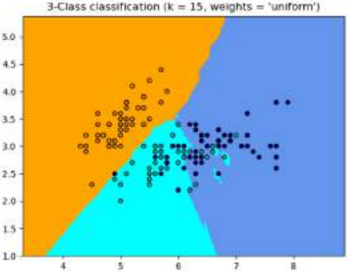


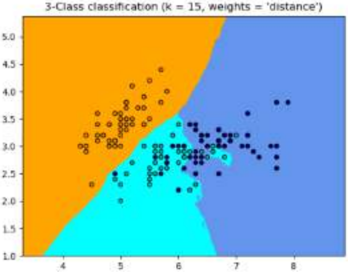


Source: https://scikit-learn.org/stable/modules/neighbors.html

smooth

Uniform Weighting vs. Inverse Distance Weighting





Source: https://scikit-learn.org/stable/modules/neighbors.html

Distance-Weighting usually produces smoother decision boundaries and is less likely to overfit (or underfit).

Summary of Nearest Neighbour

+ very easy to implement and understand
+ does not use any training/learning phase
+ can be applied to almost any prediction problem (often a good "baseline")

Summary of Nearest Neighbour

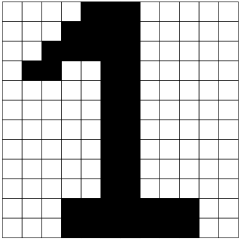
- \+ very easy to implement and understand
- \+ does not use any training/learning phase
- \+ can be applied to almost any prediction problem (often a good "baseline")
- – results may crucially depend on the choice of distance function, representation of features and k → $\log m$
- – finding nearest neighbour is costly

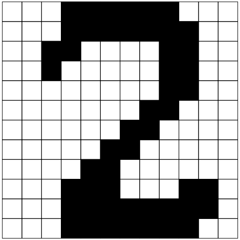
Summary of Nearest Neighbour

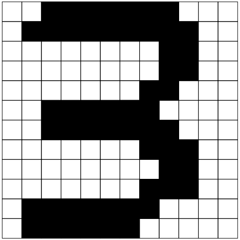
+ very easy to implement and understand
+ does not use any training/learning phase
+ can be applied to almost any prediction problem (often a good "baseline")
– results may crucially depend on the choice of distance function, representation of features and k
– finding nearest neighbour is costly

in practice, we often use a powerful pre-processing tool called **Dimensionality Reduction**!

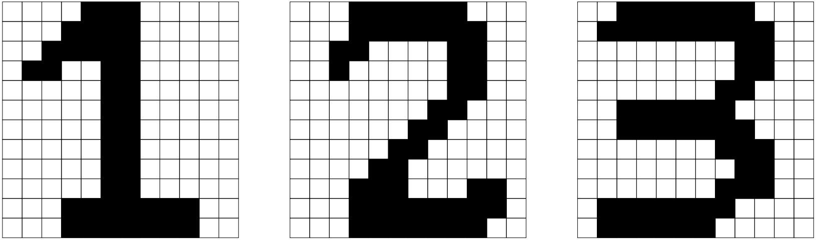
Exercise 2: Applying k-NN to Character Recognition (1/2)







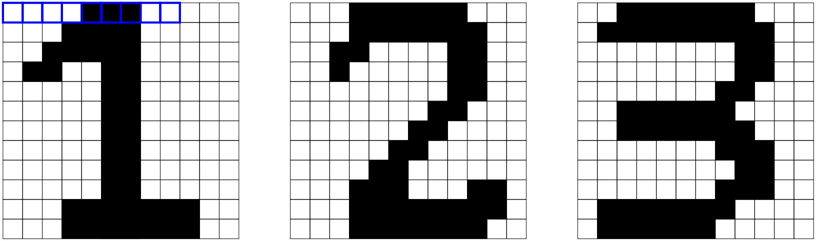
Exercise 2: Applying k-NN to Character Recognition (1/2)



- For the computer, each image is just a long sequence of bits

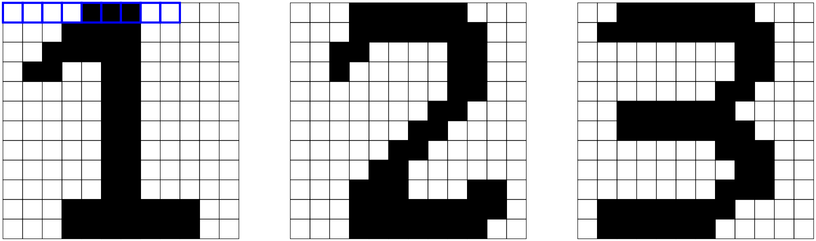
...

Exercise 2: Applying k-NN to Character Recognition (1/2)



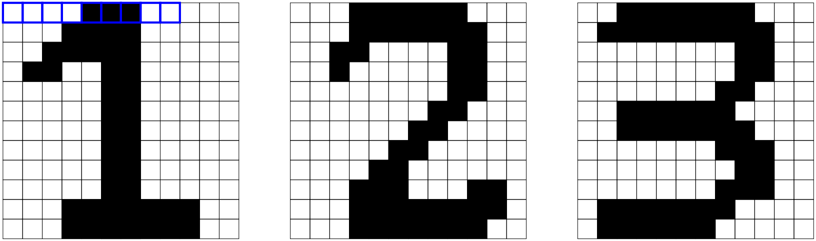
- For the computer, each image is just a long sequence of bits
  Example: the first image may start $0, 0, 0, 0, 1, 1, 1, 0, 0, \ldots$

Exercise 2: Applying k-NN to Character Recognition (1/2)



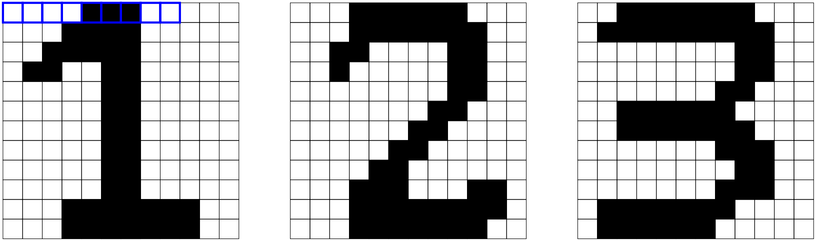
- For the computer, each image is just a long sequence of bits
  Example: the first image may start $0, 0, 0, 0, 1, 1, 1, 0, 0, \ldots$
- For a 1024×768 image, we have 786432 pixels

Exercise 2: Applying k-NN to Character Recognition (1/2)



- For the computer, each image is just a long sequence of bits
  Example: the first image may start $0, 0, 0, 0, 1, 1, 1, 0, 0, \ldots$
- For a 1024×768 image, we have 786432 pixels
- if coloured, we have 3 RGB values in the range $[0, 255]$ (8 bits)

Exercise 2: Applying k-NN to Character Recognition (1/2)



- For the computer, each image is just a long sequence of bits
  Example: the first image may start $0, 0, 0, 0, 1, 1, 1, 0, 0, \ldots$
- For a 1024×768 image, we have 786432 pixels
- if coloured, we have 3 RGB values in the range $[0, 255]$ (8 bits)
- Overall the picture is represented $786432 \cdot 3 \cdot 8 = 18,874,368$ bits

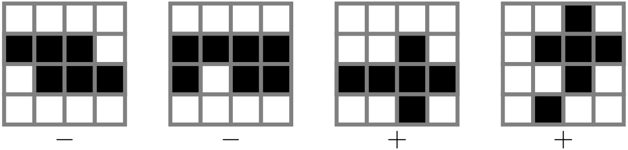
Exercise 2: Applying k-NN to Character Recognition (2/2)

Exercise 2: Applying k-NN to Character Recognition (2/2)

Apply 2-NN with distance being the number of differently coloured cells.

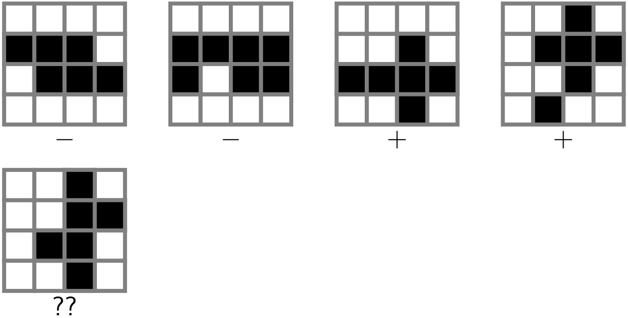
Exercise 2: Applying *k*-NN to Character Recognition (2/2)

Apply 2-NN with distance being the number of differently coloured cells.



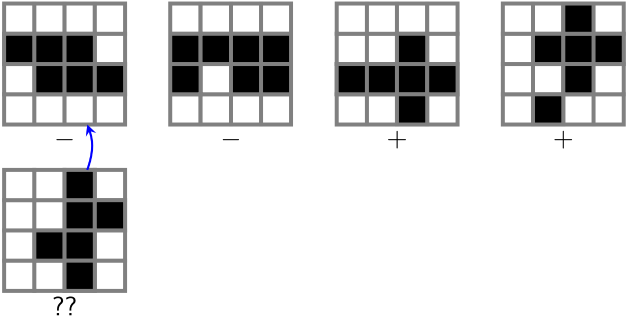
Exercise 2: Applying k-NN to Character Recognition (2/2)

Apply 2-NN with distance being the number of differently coloured cells.



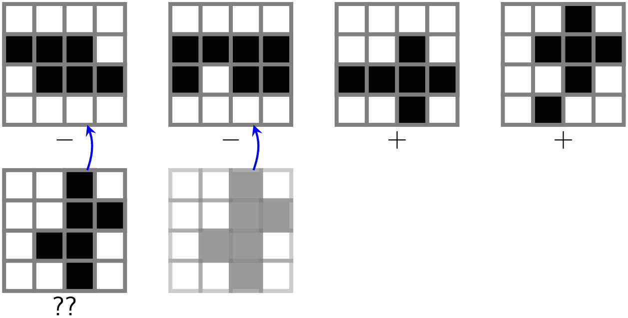
Exercise 2: Applying k-NN to Character Recognition (2/2)

Apply 2-NN with distance being the number of differently coloured cells.

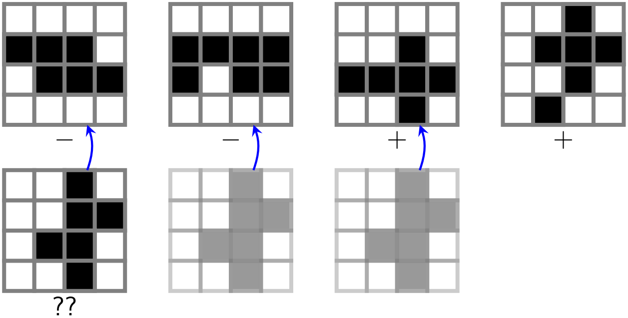


Exercise 2: Applying k-NN to Character Recognition (2/2)

Apply 2-NN with distance being the number of differently coloured cells.

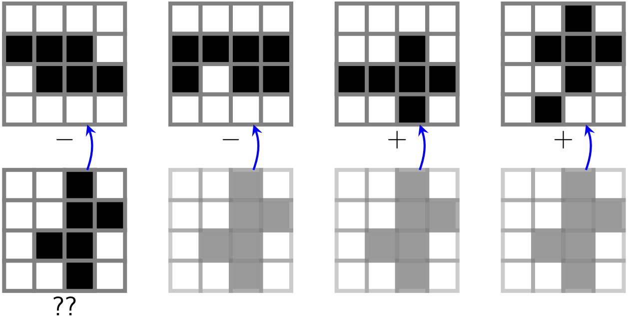


Apply 2-NN with distance being the number of differently coloured cells.



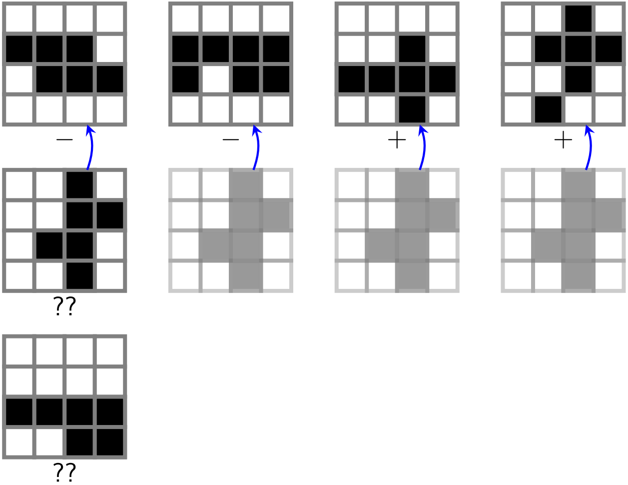
Exercise 2: Applying k-NN to Character Recognition (2/2)

Apply 2-NN with distance being the number of differently coloured cells.



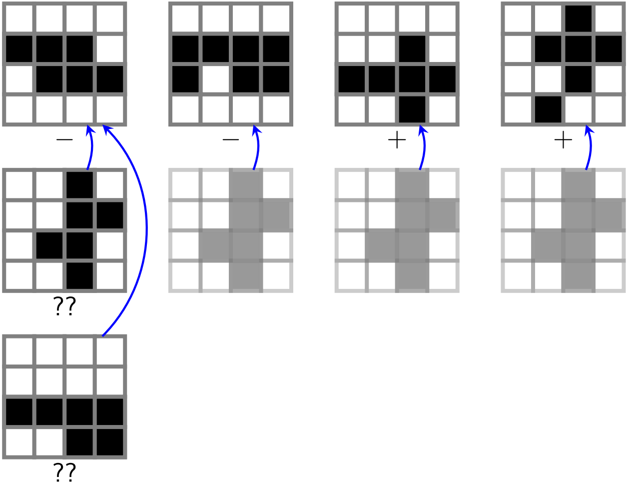
Exercise 2: Applying k-NN to Character Recognition (2/2)

Apply 2-NN with distance being the number of differently coloured cells.



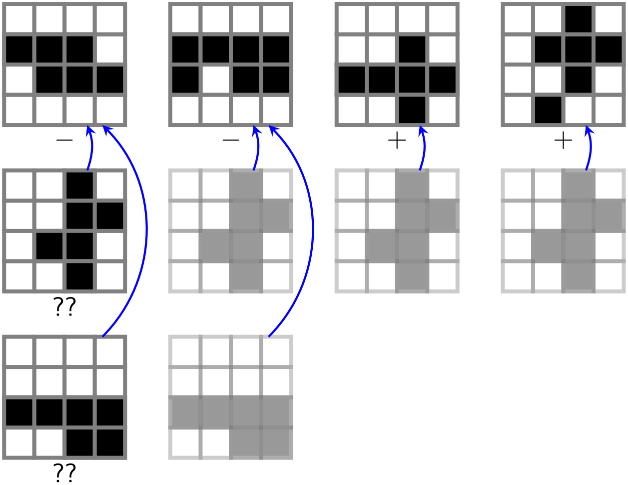
Exercise 2: Applying k-NN to Character Recognition (2/2)

Apply 2-NN with distance being the number of differently coloured cells.



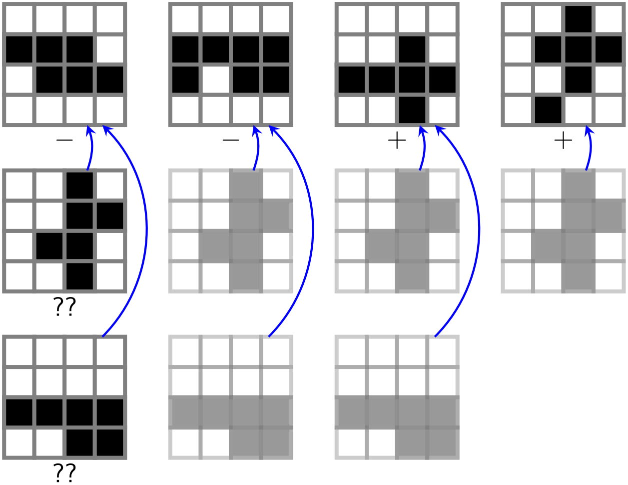
Exercise 2: Applying k-NN to Character Recognition (2/2)

Apply 2-NN with distance being the number of differently coloured cells.



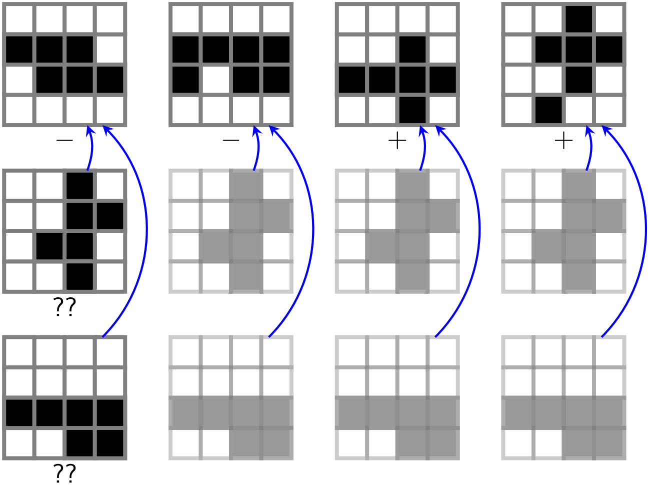
Exercise 2: Applying k-NN to Character Recognition (2/2)

Apply 2-NN with distance being the number of differently coloured cells.



Exercise 2: Applying k-NN to Character Recognition (2/2)

Apply 2-NN with distance being the number of differently coloured cells.



References

A. Blum, J. Hopcroft and R. Kannan
Foundations of Data Science
1st edition, Cambridge University Press, 2020.
https://www.cs.cornell.edu/jeh/book.pdf

J. Leskovec, A. Rajaraman, J. D. Ullmann
Mining of Massive Datasets.
3rd edition, Cambridge University Press, 2020.

S. Shalev-Shwartz and S. Ben-David
Understanding Machine Learning: From Theory to Algorithms
Cambridge University Press, 2014.
https://www.cs.huji.ac.il/~shais/UnderstandingMachineLearning/understanding-machine-learning-theory-algorithms.pdf

Outline

Nearest Neighbour Algorithm

Additional Material

Outlook: How to Use k-NN for Movie Rating Prediction

Algorithm

- Input:
  - Rating matrix with n users and p movies,
  - integer $k \geq 1$,
  - unknown rating $x_{i,j}$ (to be predicted by algorithm)
- Find a set of k movies M most similar to item j that are rated by user i
- Output:

$$x_{i,j} = \frac{\sum_{\ell \in M} \mathrm{sim}(j,\ell) \cdot x_{i,\ell}}{\sum_{\ell \in M} \mathrm{sim}(j,\ell)}$$

Some Ideas on efficient Implementation

- m number of points, d dimension, want: k nearest neighbours to input x

Some Ideas on efficient Implementation

- m number of points, d dimension, want: k nearest neighbours to input x

Basic Algorithm

- Compute all distances from the m points to x
- Scan list for the nearest neighbour of x and then remove it from list
- Running Time is $O(m \cdot d + m \cdot k)$

Some Ideas on efficient Implementation

- m number of points, d dimension, want: k nearest neighbours to input x

Basic Algorithm

- Compute all distances from the m points to x
- Scan list for the nearest neighbour of x and then remove it from list
- Running Time is $O(m \cdot d + m \cdot k)$

Algorithm based on Sorting

- Compute all distances from the m points to x
- Sort all these m distances increasingly
- Running Time is $O(m \cdot d + m \cdot \log m)$

Some Ideas on efficient Implementation

- m number of points, d dimension, want: k nearest neighbours to input x

Basic Algorithm

- Compute all distances from the m points to x
- Scan list for the nearest neighbour of x and then remove it from list
- Running Time is $O(m \cdot d + m \cdot k)$

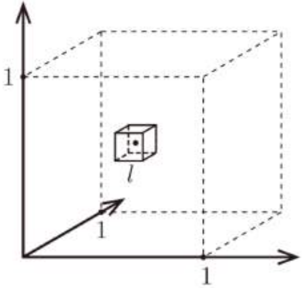
Algorithm based on Sorting

- Compute all distances from the m points to x
- Sort all these m distances increasingly
- Running Time is $O(m \cdot d + m \cdot \log m)$

Clever Algorithm

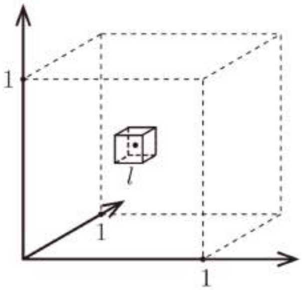
- Compute all distances from the m points to x
- Use Quick-Select to find the k nearest distances (without sorting)
- Running Time is $O(m \cdot d + m)$ if we want the list of k nearest points, and $O(m \cdot d + m + k \log k)$ if we want the k nearest points in order.

Curse of Dimensionality



Source: Kilian Weinberger

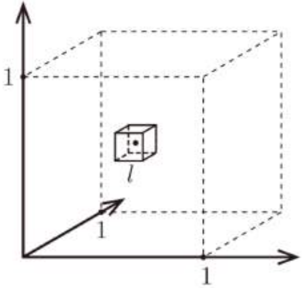
Curse of Dimensionality



Source: Kilian Weinberger

- Suppose $m = 1000$ points are "randomly" spread across $[0, 1]^d$

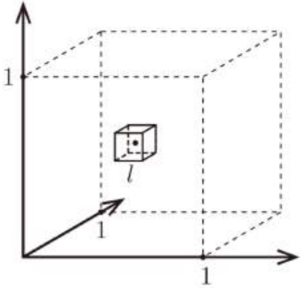
Curse of Dimensionality



Source: Kilian Weinberger

- Suppose $m = 1000$ points are "randomly" spread across $[0, 1]^d$
- If we want to find the 10 nearest neighbour, how large must be the subcube be?

Curse of Dimensionality

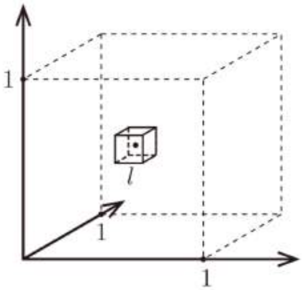


Source: Kilian Weinberger

- Suppose $m = 1000$ points are "randomly" spread across $[0, 1]^d$
- If we want to find the 10 nearest neighbour, how large must be the subcube be?

d	ℓ
2	0.1
10	0.63
100	0.955
1000	0.9954

Curse of Dimensionality



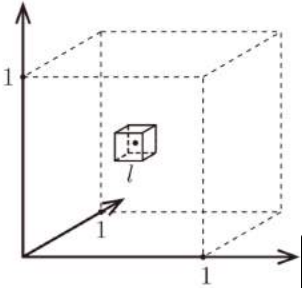
Source: Kilian Weinberger

- Suppose $m = 1000$ points are "randomly" spread across $[0, 1]^d$
- If we want to find the 10 nearest neighbour, how large must be the subcube be?

d	ℓ
2	0.1
10	0.63
100	0.955
1000	0.9954

To find the closest neighbour, we need to search the entire space!

Curse of Dimensionality



Source: Kilian Weinberger

In high dimensions, almost all points have the same (far) distance!

- Suppose $m = 1000$ points are "randomly" spread across $[0,1]^d$
- If we want to find the 10 nearest neighbour, how large must be the subcube be?

d	ℓ
2	0.1
10	0.63
100	0.955
1000	0.9954

To find the closest neighbour, we need to search the entire space!