Introduction to Machine Learning

Session 2: Distance-Based Machine Learning

Benjamin Paaßen

The University of Sydney

IK 2020, Günne

Recap: What is Machine Learning?

- There is a 'true' data generating process $y = f(x) + \epsilon$

Recap: What is Machine Learning?

- There is a 'true' data generating process $y = f(x) + \epsilon$, from which we only see a sample $\mathcal{D} = \{(x_1, y_1), \ldots, (x_N, y_N)\}$

Recap: What is Machine Learning?

- There is a 'true' data generating process $y = f(x) + \epsilon$, from which we only see a sample $\mathcal{D} = \{(x_1, y_1), \ldots, (x_N, y_N)\}$
- We learn a rule $f_{\mathcal{D}}$ from that sample

Recap: What is Machine Learning?

- There is a 'true' data generating process $y = f(x) + \epsilon$, from which we only see a sample $\mathcal{D} = \{(x_1, y_1), \ldots, (x_N, y_N)\}$
- We learn a rule $f_{\mathcal{D}}$ from that sample (that's the machine learning part)

Recap: What is Machine Learning?

- There is a 'true' data generating process $y = f(x) + \epsilon$, from which we only see a sample $\mathcal{D} = \{(x_1, y_1), \ldots, (x_N, y_N)\}$
- We learn a rule $f_{\mathcal{D}}$ from that sample (that's the machine learning part)
- ... that hopefully generalizes to the entire ground truth distribution

Recap: What is Machine Learning?

- There is a 'true' data generating process $y = f(x) + \epsilon$, from which we only see a sample $\mathcal{D} = \{(x_1, y_1), \ldots, (x_N, y_N)\}$
- We learn a rule $f_{\mathcal{D}}$ from that sample (that's the machine learning part)
- ... that hopefully generalizes to the entire ground truth distribution, i.e.: $f_{\mathcal{D}}(x) \approx f(x)$ for all x

Smoothness assumption

- Question: When can we hope to generalize at all?

Smoothness assumption

- Question: When can we hope to generalize at all?
- Idea: If the 'true' function f is **smooth**

Smoothness assumption

- Question: When can we hope to generalize at all?
- Idea: If the 'true' function f is **smooth**

Definition

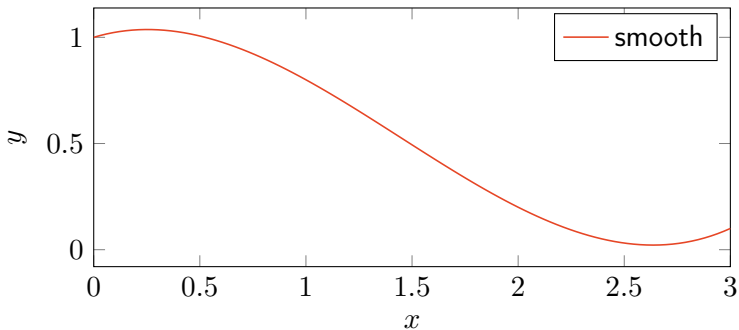
We call a function f **smooth** if x being close to x' implies that $f(x)$ is close to $f(x')$.

Smoothness assumption

- Question: When can we hope to generalize at all?
- Idea: If the 'true' function f is **smooth**

Definition

We call a function f **smooth** if x being close to x' implies that $f(x)$ is close to $f(x')$.

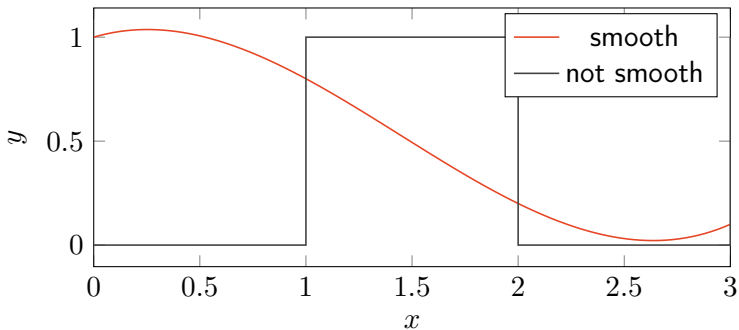


Smoothness assumption

- Question: When can we hope to generalize at all?
- Idea: If the 'true' function f is **smooth**

Definition

We call a function f **smooth** if x being close to x' implies that $f(x)$ is close to $f(x')$.

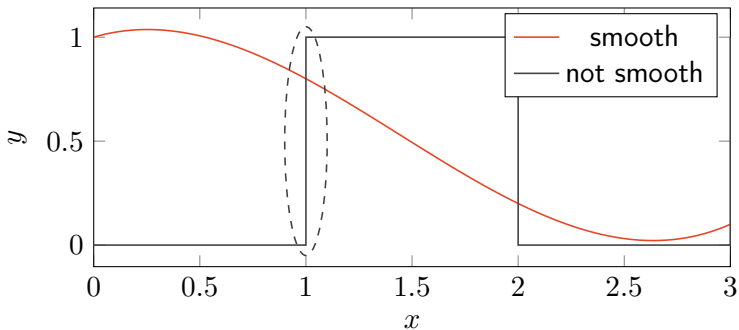


Smoothness assumption

- Question: When can we hope to generalize at all?
- Idea: If the 'true' function f is **smooth**

Definition

We call a function f **smooth** if x being close to x' implies that $f(x)$ is close to $f(x')$.



Distance

- Smoothness is closely related to notions of **distance**

Distance

- Smoothness is closely related to notions of **distance**, i.e. f is smooth if small distance in input implies small distance in output

Distance

- Smoothness is closely related to notions of **distance**, i.e. f is smooth if small distance in input implies small distance in output

Definition

Let $\mathcal{X}$ be some set.

Distance

- Smoothness is closely related to notions of **distance**, i.e. f is smooth if small distance in input implies small distance in output

Definition

Let $\mathcal{X}$ be some set. A function $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is called a **distance** if for all $x, x', x'' \in \mathcal{X}$ it holds:

Distance

- Smoothness is closely related to notions of **distance**, i.e. f is smooth if small distance in input implies small distance in output

Definition

Let $\mathcal{X}$ be some set. A function $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is called a **distance** if for all $x, x', x'' \in \mathcal{X}$ it holds:

$$d(x, x') \geq 0 \qquad \text{Non-Negativity}$$

Distance

- Smoothness is closely related to notions of **distance**, i.e. f is smooth if small distance in input implies small distance in output

Definition

Let $\mathcal{X}$ be some set. A function $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is called a **distance** if for all $x, x', x'' \in \mathcal{X}$ it holds:

$$d(x, x') \geq 0 \qquad \text{Non-Negativity}$$

$$d(x, x') = 0 \iff x = x' \qquad \text{Self-Identity}$$

Distance

- Smoothness is closely related to notions of **distance**, i.e. f is smooth if small distance in input implies small distance in output

Definition

Let $\mathcal{X}$ be some set. A function $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is called a **distance** if for all $x, x', x'' \in \mathcal{X}$ it holds:

$$
\begin{aligned}
d(x, x') &\geq 0 && && \text{Non-Negativity} \\
d(x, x') &= 0 && \iff x = x' && \text{Self-Identity} \\
d(x, x') &= d(x', x) && && \text{Symmetry}
\end{aligned}
$$

Distance

- Smoothness is closely related to notions of **distance**, i.e. f is smooth if small distance in input implies small distance in output

Definition

Let $\mathcal{X}$ be some set. A function $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is called a **distance** if for all $x, x', x'' \in \mathcal{X}$ it holds:

$$
\begin{aligned}
d(x, x') &\geq 0 && && \text{Non-Negativity} \\
d(x, x') &= 0 && \iff x = x' && \text{Self-Identity} \\
d(x, x') &= d(x', x) && && \text{Symmetry} \\
d(x, x') + d(x', x'') &\geq d(x, x'') && && \text{Triangular Inequality}
\end{aligned}
$$

Distance

- Smoothness is closely related to notions of **distance**, i.e. f is smooth if small distance in input implies small distance in output

Definition

Let $\mathcal{X}$ be some set. A function $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is called a **distance** if for all $x, x', x'' \in \mathcal{X}$ it holds:

$$
\begin{aligned}
d(x, x') &\geq 0 && && \text{Non-Negativity} \\
d(x, x') &= 0 && \iff x = x' && \text{Self-Identity} \\
d(x, x') &= d(x', x) && && \text{Symmetry} \\
d(x, x') + d(x', x'') &\geq d(x, x'') && && \text{Triangular Inequality}
\end{aligned}
$$

- Idea: Use distances as **interface to the data** with key principle: Low distance means similar prediction

Distance

- Smoothness is closely related to notions of **distance**, i.e. f is smooth if small distance in input implies small distance in output

Definition

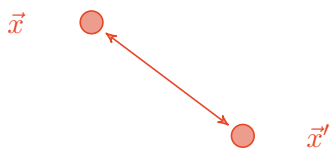
Let $\mathcal{X}$ be some set. A function $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is called a **distance** if for all $x, x', x'' \in \mathcal{X}$ it holds:

$$
\begin{aligned}
d(x, x') &\geq 0 && && \text{Non-Negativity} \\
d(x, x') &= 0 && \iff x = x' && \text{Self-Identity} \\
d(x, x') &= d(x', x) && && \text{Symmetry} \\
d(x, x') + d(x', x'') &\geq d(x, x'') && && \text{Triangular Inequality}
\end{aligned}
$$

- Idea: Use distances as **interface to the data** with key principle: Low distance means similar prediction
- Note: Almost all algorithms implemented in `scikit-learn.org`

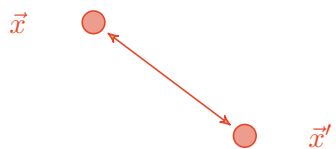
Examples of Distances

Euclidean Distance



Examples of Distances

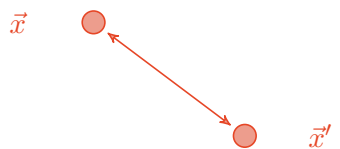
Euclidean Distance



$$d(x, x') = \sqrt{(\vec{x} - \vec{x}')^T \cdot (\vec{x} - \vec{x}')}$$

Examples of Distances

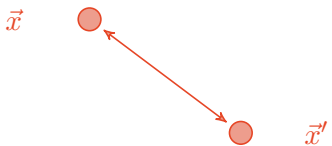
Euclidean Distance



$$d(x, x') = \sqrt{(\vec{x} - \vec{x}')^T \cdot (\vec{x} - \vec{x}')}$$
$$= \sqrt{\sum_{j=1}^{n} (x_j - x'_j)^2}$$

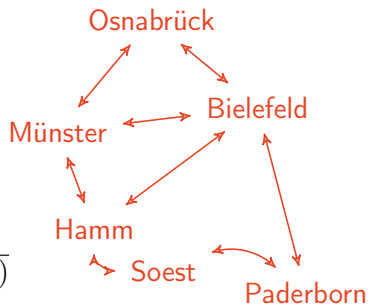
Examples of Distances

Euclidean Distance



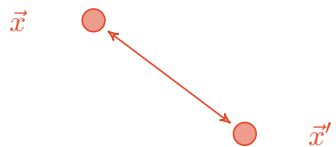
$$d(x, x') = \sqrt{(\vec{x} - \vec{x}')^T \cdot (\vec{x} - \vec{x}')}$$
$$= \sqrt{\sum_{j=1}^{n} (x_j - x'_j)^2}$$

Shortest Path Distance



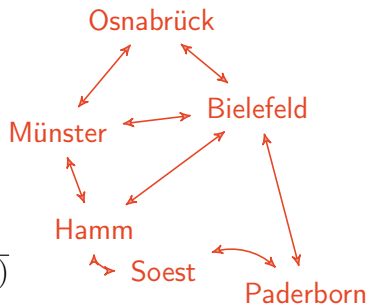
Examples of Distances

Euclidean Distance



$$d(x, x') = \sqrt{(\vec{x} - \vec{x}')^T \cdot (\vec{x} - \vec{x}')} = \sqrt{\sum_{j=1}^{n} (x_j - x'_j)^2}$$

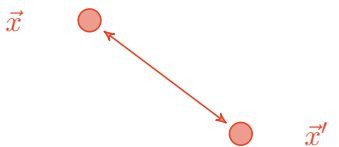
Shortest Path Distance



$$d(x, x') = \min_{\substack{x_1,\ldots,x_T \\ x_1=x, x_T=x' \\ (x_t, x_{t+1}) \in E}} \sum_{t=1}^{T-1} w(x_t, x_{t+1})$$

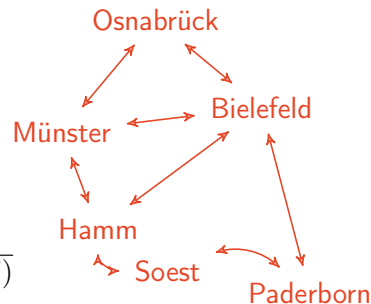
Examples of Distances

Euclidean Distance



$$d(x, x') = \sqrt{(\vec{x} - \vec{x}')^T \cdot (\vec{x} - \vec{x}')}$$
$$= \sqrt{\sum_{j=1}^{n} (x_j - x'_j)^2}$$

Shortest Path Distance



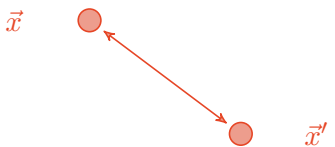
$$d(x, x') = \min_{\substack{x_1,\ldots,x_T \\ x_1=x, x_T=x' \\ (x_t, x_{t+1}) \in E}} \sum_{t=1}^{T-1} w(x_t, x_{t+1})$$

String Edit Distance

bla
↓rep(a,u)
blu
↓ins(b)
blub

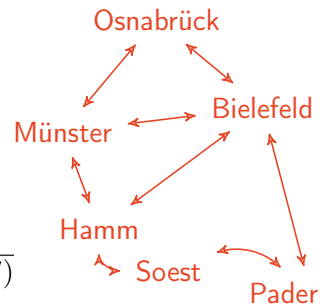
Examples of Distances

Euclidean Distance



$$d(x,x') = \sqrt{(\vec{x}-\vec{x'})^T \cdot (\vec{x}-\vec{x'})}$$
$$= \sqrt{\sum_{j=1}^{n}(x_j - x'_j)^2}$$

Shortest Path Distance



$$d(x,x') = \min_{\substack{x_1,\ldots,x_T \\ x_1=x, x_T=x' \\ (x_t,x_{t+1})\in E}} \sum_{t=1}^{T-1} w(x_t, x_{t+1})$$

String Edit Distance

bla
↓rep(a,u)
blu
↓ins(b)
blub

$$d(x,x') = \min_{\substack{\delta_1,\ldots,\delta_T \\ \delta_T \circ \ldots \circ \delta_1(x)=x'}} T$$

Regression

Nearest Neighbor Regression

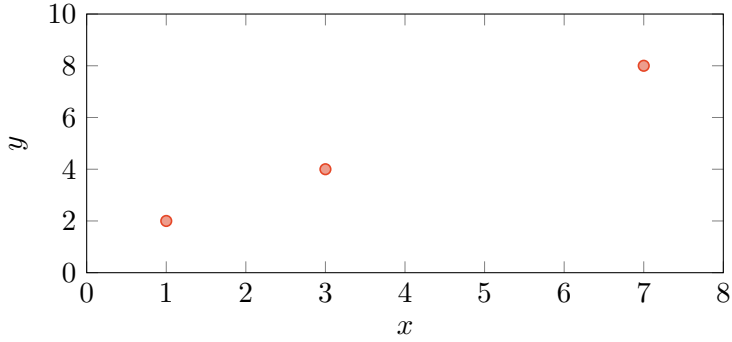
- Assign the output for the closest input

Nearest Neighbor Regression

- Assign the output for the closest input, i.e. $f_{\mathcal{D}}(x) = y_i$ where $i = \arg\min_i d(x, x_i)$

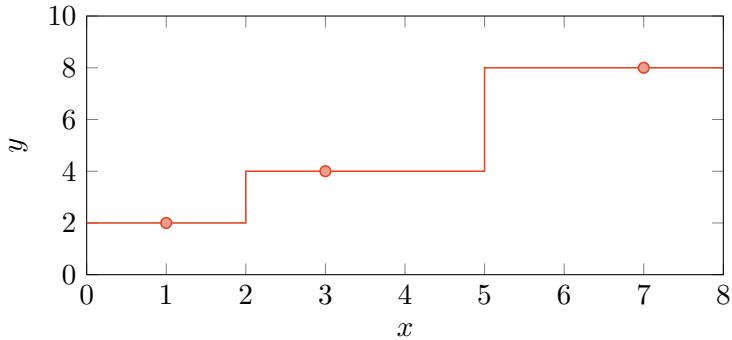
Nearest Neighbor Regression

- Assign the output for the closest input, i.e. $f_{\mathcal{D}}(x) = y_i$ where $i = \arg\min_i d(x, x_i)$



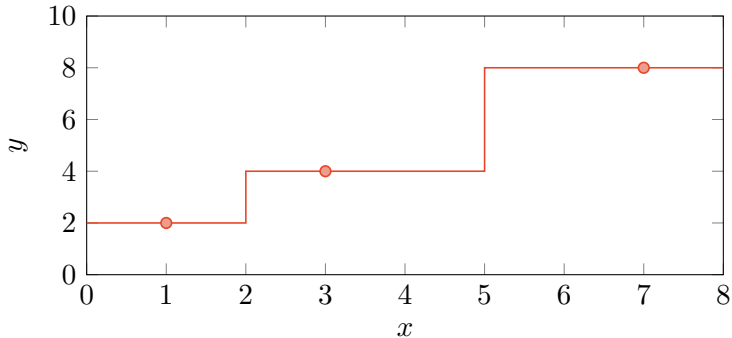
Nearest Neighbor Regression

- Assign the output for the closest input, i.e. $f_{\mathcal{D}}(x) = y_i$ where $i = \arg\min_i d(x, x_i)$



Nearest Neighbor Regression

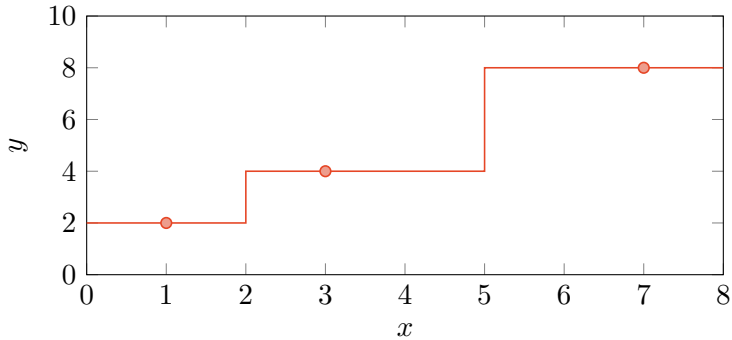
- Assign the output for the closest input, i.e. $f_{\mathcal{D}}(x) = y_i$ where $i = \arg\min_i d(x, x_i)$



- Not a smooth model, but 'approaching smoothness' for enough data

Nearest Neighbor Regression

- Assign the output for the closest input, i.e. $f_{\mathcal{D}}(x) = y_i$ where $i = \arg\min_i d(x, x_i)$



- Not a smooth model, but 'approaching smoothness' for enough data
- straightforward extension: average of k nearest neighbors

Kernel regression (aka Gaussian Processes) (Rasmussen and Williams 2005)

- Idea: Average predictions based on **closeness**:

Kernel regression (aka Gaussian Processes) (Rasmussen and Williams 2005)

- Idea: Average predictions based on **closeness**:

$$f_{\mathcal{D}}(x) = \sum_{i=1}^{N} k(x, x_i) \cdot \alpha_i = \vec{k}(x)^T \cdot \vec{\alpha}$$

where $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ measures **closeness** and α_i is the ith prediction

Kernel regression (aka Gaussian Processes) (Rasmussen and Williams 2005)

- Idea: Average predictions based on **closeness**:

$$f_{\mathcal{D}}(x) = \sum_{i=1}^{N} k(x, x_i) \cdot \alpha_i = \vec{k}(x)^T \cdot \vec{\alpha}$$

  where $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ measures **closeness** and α_i is the ith prediction

- For us, k is always the **Gaussian** or **RBF** kernel with hyper-parameter $\psi \in \mathbb{R}^+$:

Kernel regression (aka Gaussian Processes) (Rasmussen and Williams 2005)

- Idea: Average predictions based on **closeness**:

$$f_{\mathcal{D}}(x) = \sum_{i=1}^{N} k(x, x_i) \cdot \alpha_i = \vec{k}(x)^T \cdot \vec{\alpha}$$

  where $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ measures **closeness** and α_i is the ith prediction

- For us, k is always the **Gaussian** or **RBF** kernel with hyper-parameter $\psi \in \mathbb{R}^+$:

$$k(x, x') = \exp\big(-\frac{1}{2} \cdot \frac{d(x, x')^2}{\psi^2}\big)$$

Kernel regression (aka Gaussian Processes) (Rasmussen and Williams 2005)

- Idea: Average predictions based on **closeness**:

$$f_{\mathcal{D}}(x) = \sum_{i=1}^{N} k(x, x_i) \cdot \alpha_i = \vec{k}(x)^T \cdot \vec{\alpha}$$

  where $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ measures **closeness** and α_i is the ith prediction

- For us, k is always the **Gaussian** or **RBF** kernel with hyper-parameter $\psi \in \mathbb{R}^+$:

$$k(x, x') = \exp\big(-\frac{1}{2} \cdot \frac{d(x, x')^2}{\psi^2}\big)$$

- As loss, we use the regularized squared error as for linear regression:

Kernel regression (aka Gaussian Processes) (Rasmussen and Williams 2005)

- Idea: Average predictions based on **closeness**:

$$f_{\mathcal{D}}(x) = \sum_{i=1}^{N} k(x, x_i) \cdot \alpha_i = \vec{k}(x)^T \cdot \vec{\alpha}$$

  where $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ measures **closeness** and α_i is the ith prediction

- For us, k is always the **Gaussian** or **RBF** kernel with hyper-parameter $\psi \in \mathbb{R}^+$:

$$k(x, x') = \exp\big(-\frac{1}{2} \cdot \frac{d(x, x')^2}{\psi^2}\big)$$

- As loss, we use the regularized squared error as for linear regression:

$$\ell(\vec{\alpha}) = \sum_{i=1}^{N} \big(\vec{k}_i \cdot \vec{\alpha} - y_i\big)^2 + \lambda \cdot \vec{\alpha}^T \cdot \boldsymbol{K} \cdot \vec{\alpha}$$

Kernel regression (aka Gaussian Processes) (Rasmussen and Williams 2005)

- Idea: Average predictions based on **closeness**:

$$f_{\mathcal{D}}(x) = \sum_{i=1}^{N} k(x, x_i) \cdot \alpha_i = \vec{k}(x)^T \cdot \vec{\alpha}$$

where $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ measures **closeness** and α_i is the ith prediction

- For us, k is always the **Gaussian** or **RBF** kernel with hyper-parameter $\psi \in \mathbb{R}^+$:

$$k(x, x') = \exp\big(-\frac{1}{2} \cdot \frac{d(x, x')^2}{\psi^2}\big)$$

- As loss, we use the regularized squared error as for linear regression:

$$\ell(\vec{\alpha}) = \sum_{i=1}^{N} \big(\vec{k}_i \cdot \vec{\alpha} - y_i\big)^2 + \lambda \cdot \vec{\alpha}^T \cdot \boldsymbol{K} \cdot \vec{\alpha}$$

where $\boldsymbol{K}_{i,j} = k(x_i, x_j)$ and $\vec{k}_i$ is the ith column of $\boldsymbol{K}$

Kernel Regression (II)

- Let's consider gradient/derivative:

Kernel Regression (II)

- Let's consider gradient/derivative:

$$\nabla_{\vec{\alpha}} \ell(\vec{\alpha}) = 2 \sum_{i=1}^{N} \vec{k}_i \cdot \left(\vec{k}_i \cdot \vec{\alpha} - y_i \right) + 2\lambda \cdot \boldsymbol{K} \cdot \vec{\alpha}$$

Kernel Regression (II)

- Let's consider gradient/derivative:

$$\nabla_{\vec{\alpha}} \ell(\vec{\alpha}) = 2\Big(\sum_{i=1}^{N} \vec{k}_i \cdot \vec{k}_i\Big) \cdot \vec{\alpha} - 2\sum_{i=1}^{N} \vec{k}_i \cdot y_i + 2\lambda \cdot \boldsymbol{K} \cdot \vec{\alpha}$$

Kernel Regression (II)

- Let's consider gradient/derivative:

$$\nabla_{\vec{\alpha}} \ell(\vec{\alpha}) = 2\boldsymbol{K} \cdot \boldsymbol{K}^T \cdot \vec{\alpha} - 2\boldsymbol{K} \cdot \vec{y} + 2\lambda \cdot \boldsymbol{K} \cdot \vec{\alpha}$$

Kernel Regression (II)

- Let's consider gradient/derivative:

$$\nabla_{\vec{\alpha}}\ell(\vec{\alpha}) = 2\boldsymbol{K}\cdot\boldsymbol{K}^T\cdot\vec{\alpha} - 2\boldsymbol{K}\cdot\vec{y} + 2\lambda\cdot\boldsymbol{K}\cdot\vec{\alpha} = 2\boldsymbol{K}\cdot\Big(\big[\boldsymbol{K}+\lambda\cdot\boldsymbol{I}\big]\cdot\alpha - \vec{y}\Big)$$

Kernel Regression (II)

- Let's consider gradient/derivative:

$$\nabla_{\vec{\alpha}}\ell(\vec{\alpha}) = 2\boldsymbol{K}\cdot\boldsymbol{K}^T\cdot\vec{\alpha} - 2\boldsymbol{K}\cdot\vec{y} + 2\lambda\cdot\boldsymbol{K}\cdot\vec{\alpha} = 2\boldsymbol{K}\cdot\Big(\big[\boldsymbol{K}+\lambda\cdot\boldsymbol{I}\big]\cdot\alpha - \vec{y}\Big)$$

- Solution for $\nabla_{\vec{\alpha}}\ell(\vec{\alpha}) = 0$: $\vec{\alpha} = \big(\boldsymbol{K} + \lambda\cdot\boldsymbol{I}\big)^{-1}\cdot\vec{y}$

Kernel Regression (II)

- Let's consider gradient/derivative:

$$\nabla_{\vec{\alpha}}\ell(\vec{\alpha}) = 2\boldsymbol{K}\cdot\boldsymbol{K}^T\cdot\vec{\alpha} - 2\boldsymbol{K}\cdot\vec{y} + 2\lambda\cdot\boldsymbol{K}\cdot\vec{\alpha} = 2\boldsymbol{K}\cdot\Big(\big[\boldsymbol{K}+\lambda\cdot\boldsymbol{I}\big]\cdot\alpha - \vec{y}\Big)$$

- Solution for $\nabla_{\vec{\alpha}}\ell(\vec{\alpha}) = 0$: $\vec{\alpha} = \big(\boldsymbol{K}+\lambda\cdot\boldsymbol{I}\big)^{-1}\cdot\vec{y}$
- Hessian: $\nabla^2_{\vec{\alpha}}\ell(\vec{\alpha}) = 2\boldsymbol{K}\cdot\big[\boldsymbol{K}+\lambda\cdot\boldsymbol{I}\big]$ is positive (semi-definite)

Kernel Regression (II)

- Let's consider gradient/derivative:

$$\nabla_{\vec{\alpha}}\ell(\vec{\alpha}) = 2\boldsymbol{K}\cdot\boldsymbol{K}^T\cdot\vec{\alpha} - 2\boldsymbol{K}\cdot\vec{y} + 2\lambda\cdot\boldsymbol{K}\cdot\vec{\alpha} = 2\boldsymbol{K}\cdot\Big(\big[\boldsymbol{K}+\lambda\cdot\boldsymbol{I}\big]\cdot\alpha - \vec{y}\Big)$$

- Solution for $\nabla_{\vec{\alpha}}\ell(\vec{\alpha}) = 0$: $\vec{\alpha} = \big(\boldsymbol{K}+\lambda\cdot\boldsymbol{I}\big)^{-1}\cdot\vec{y}$
- Hessian: $\nabla^2_{\vec{\alpha}}\ell(\vec{\alpha}) = 2\boldsymbol{K}\cdot\big[\boldsymbol{K}+\lambda\cdot\boldsymbol{I}\big]$ is positive (semi-definite) $\Rightarrow$ convex

Kernel Regression (II)

- Let's consider gradient/derivative:

$$\nabla_{\vec{\alpha}}\ell(\vec{\alpha}) = 2\boldsymbol{K}\cdot\boldsymbol{K}^T\cdot\vec{\alpha} - 2\boldsymbol{K}\cdot\vec{y} + 2\lambda\cdot\boldsymbol{K}\cdot\vec{\alpha} = 2\boldsymbol{K}\cdot\Big(\big[\boldsymbol{K}+\lambda\cdot\boldsymbol{I}\big]\cdot\alpha - \vec{y}\Big)$$

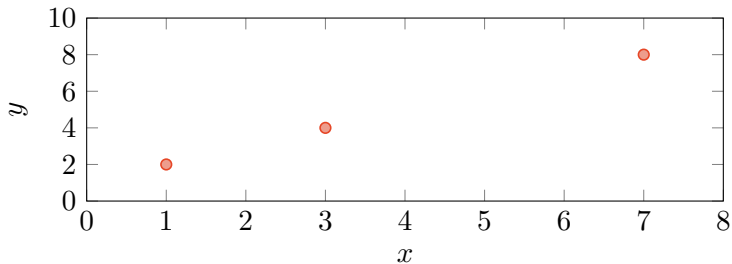
- Solution for $\nabla_{\vec{\alpha}}\ell(\vec{\alpha}) = 0$: $\vec{\alpha} = \big(\boldsymbol{K}+\lambda\cdot\boldsymbol{I}\big)^{-1}\cdot\vec{y}$
- Hessian: $\nabla^2_{\vec{\alpha}}\ell(\vec{\alpha}) = 2\boldsymbol{K}\cdot\big[\boldsymbol{K}+\lambda\cdot\boldsymbol{I}\big]$ is positive (semi-definite) $\Rightarrow$ convex $\Rightarrow$ solution is optimal

Kernel Regression (II)

- Let's consider gradient/derivative:

$$\nabla_{\vec{\alpha}}\ell(\vec{\alpha}) = 2\boldsymbol{K}\cdot\boldsymbol{K}^T\cdot\vec{\alpha} - 2\boldsymbol{K}\cdot\vec{y} + 2\lambda\cdot\boldsymbol{K}\cdot\vec{\alpha} = 2\boldsymbol{K}\cdot\Big(\big[\boldsymbol{K}+\lambda\cdot\boldsymbol{I}\big]\cdot\alpha - \vec{y}\Big)$$

- Solution for $\nabla_{\vec{\alpha}}\ell(\vec{\alpha}) = 0$: $\vec{\alpha} = \big(\boldsymbol{K} + \lambda\cdot\boldsymbol{I}\big)^{-1}\cdot\vec{y}$
- Hessian: $\nabla^2_{\vec{\alpha}}\ell(\vec{\alpha}) = 2\boldsymbol{K}\cdot\big[\boldsymbol{K}+\lambda\cdot\boldsymbol{I}\big]$ is positive (semi-definite) $\Rightarrow$ convex $\Rightarrow$ solution is optimal

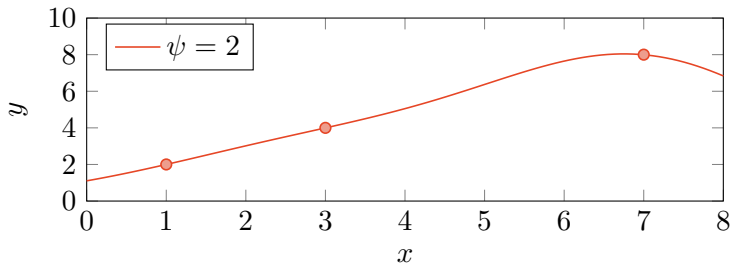


Kernel Regression (II)

- Let's consider gradient/derivative:

$$\nabla_{\vec{\alpha}}\ell(\vec{\alpha}) = 2\boldsymbol{K}\cdot\boldsymbol{K}^T\cdot\vec{\alpha} - 2\boldsymbol{K}\cdot\vec{y} + 2\lambda\cdot\boldsymbol{K}\cdot\vec{\alpha} = 2\boldsymbol{K}\cdot\Big(\big[\boldsymbol{K}+\lambda\cdot\boldsymbol{I}\big]\cdot\alpha - \vec{y}\Big)$$

- Solution for $\nabla_{\vec{\alpha}}\ell(\vec{\alpha}) = 0$: $\vec{\alpha} = \big(\boldsymbol{K} + \lambda\cdot\boldsymbol{I}\big)^{-1}\cdot\vec{y}$
- Hessian: $\nabla^2_{\vec{\alpha}}\ell(\vec{\alpha}) = 2\boldsymbol{K}\cdot\big[\boldsymbol{K}+\lambda\cdot\boldsymbol{I}\big]$ is positive (semi-definite) $\Rightarrow$ convex $\Rightarrow$ solution is optimal

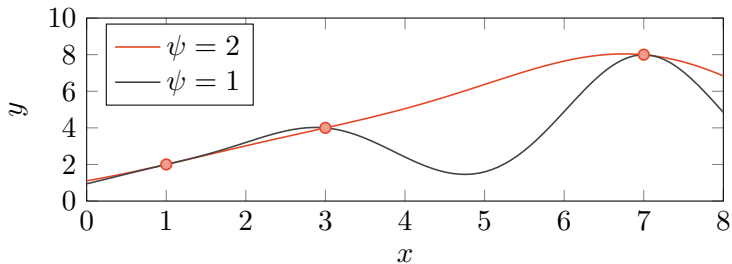


Kernel Regression (II)

- Let's consider gradient/derivative:

$$\nabla_{\vec{\alpha}}\ell(\vec{\alpha}) = 2\boldsymbol{K}\cdot\boldsymbol{K}^T\cdot\vec{\alpha} - 2\boldsymbol{K}\cdot\vec{y} + 2\lambda\cdot\boldsymbol{K}\cdot\vec{\alpha} = 2\boldsymbol{K}\cdot\Big(\big[\boldsymbol{K}+\lambda\cdot\boldsymbol{I}\big]\cdot\alpha - \vec{y}\Big)$$

- Solution for $\nabla_{\vec{\alpha}}\ell(\vec{\alpha}) = 0$: $\vec{\alpha} = \big(\boldsymbol{K} + \lambda\cdot\boldsymbol{I}\big)^{-1}\cdot\vec{y}$
- Hessian: $\nabla^2_{\vec{\alpha}}\ell(\vec{\alpha}) = 2\boldsymbol{K}\cdot\big[\boldsymbol{K}+\lambda\cdot\boldsymbol{I}\big]$ is positive (semi-definite) $\Rightarrow$ convex $\Rightarrow$ solution is optimal



Classification

Classification

- Until now, we only considered **regression** problems with smooth outputs

Classification

- Until now, we only considered **regression** problems with smooth outputs
- What if my output y is a **discrete** decision?

Classification

- Until now, we only considered **regression** problems with smooth outputs
- What if my output y is a **discrete** decision? e.g. does this image show a cat or a dog (or neither)?

Classification

- Until now, we only considered **regression** problems with smooth outputs
- What if my output y is a **discrete** decision? e.g. does this image show a cat or a dog (or neither)?

Definition: Classification

Let $\mathcal{X}$ be some set and $\mathcal{Y}$ be a **finite** set, i.e. $\mathcal{Y} = \{1, \dots, L\}$.

Classification

- Until now, we only considered **regression** problems with smooth outputs
- What if my output y is a **discrete** decision? e.g. does this image show a cat or a dog (or neither)?

Definition: Classification

Let $\mathcal{X}$ be some set and $\mathcal{Y}$ be a **finite** set, i.e. $\mathcal{Y} = \{1, \ldots, L\}$. Further, let f be some function from $\mathcal{X}$ to $\mathcal{Y}$.

Classification

- Until now, we only considered **regression** problems with smooth outputs
- What if my output y is a **discrete** decision? e.g. does this image show a cat or a dog (or neither)?

Definition: Classification

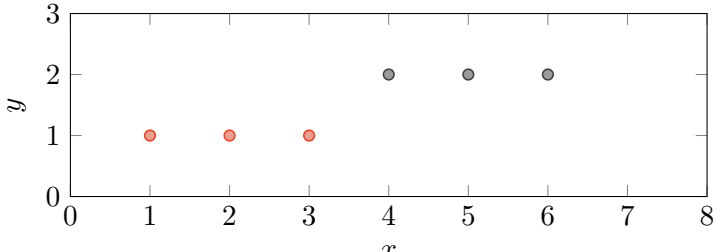
Let $\mathcal{X}$ be some set and $\mathcal{Y}$ be a **finite** set, i.e. $\mathcal{Y} = \{1, \ldots, L\}$. Further, let f be some function from $\mathcal{X}$ to $\mathcal{Y}$. Trying to infer f from example data is what we call a **classification** problem.

Classification

- Until now, we only considered **regression** problems with smooth outputs
- What if my output y is a **discrete** decision? e.g. does this image show a cat or a dog (or neither)?

Definition: Classification

Let $\mathcal{X}$ be some set and $\mathcal{Y}$ be a **finite** set, i.e. $\mathcal{Y} = \{1, \ldots, L\}$. Further, let f be some function from $\mathcal{X}$ to $\mathcal{Y}$. Trying to infer f from example data is what we call a **classification** problem.

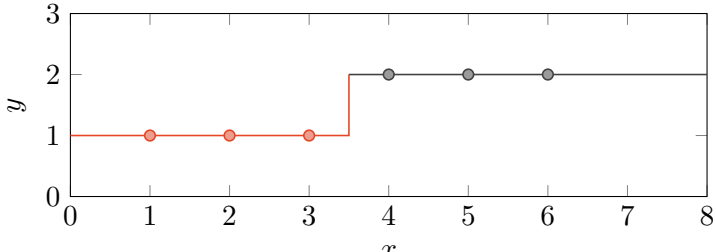


Classification

- Until now, we only considered **regression** problems with smooth outputs
- What if my output y is a **discrete** decision? e.g. does this image show a cat or a dog (or neither)?

Definition: Classification

Let $\mathcal{X}$ be some set and $\mathcal{Y}$ be a **finite** set, i.e. $\mathcal{Y} = \{1, \ldots, L\}$. Further, let f be some function from $\mathcal{X}$ to $\mathcal{Y}$. Trying to infer f from example data is what we call a **classification** problem.



Nearest Neighbor Classification (Cover and Hart 1967)

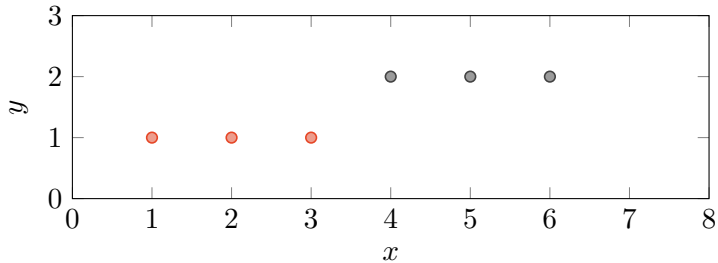
- Assign the output for the closest input

Nearest Neighbor Classification (Cover and Hart 1967)

- Assign the output for the closest input, i.e. $f_{\mathcal{D}}(x) = y_i$ where $i = \arg\min_i d(x, x_i)$

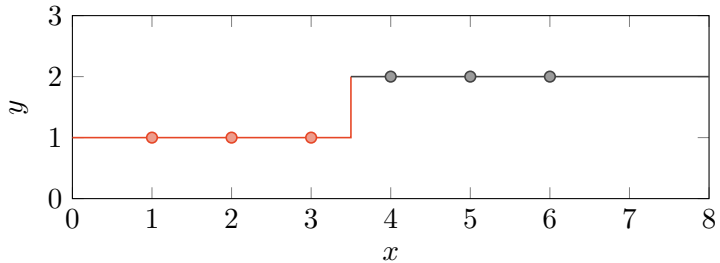
Nearest Neighbor Classification (Cover and Hart 1967)

- Assign the output for the closest input, i.e. $f_{\mathcal{D}}(x) = y_i$ where $i = \arg\min_i d(x, x_i)$



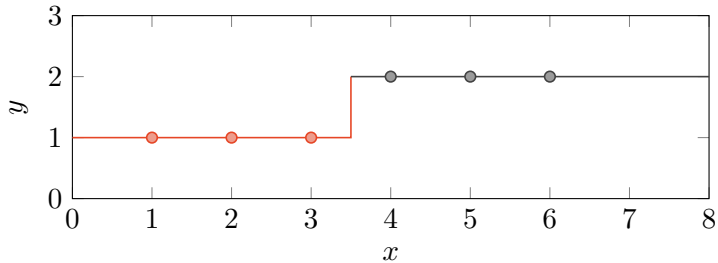
Nearest Neighbor Classification (Cover and Hart 1967)

- Assign the output for the closest input, i.e. $f_{\mathcal{D}}(x) = y_i$ where $i = \arg\min_i d(x, x_i)$



Nearest Neighbor Classification (Cover and Hart 1967)

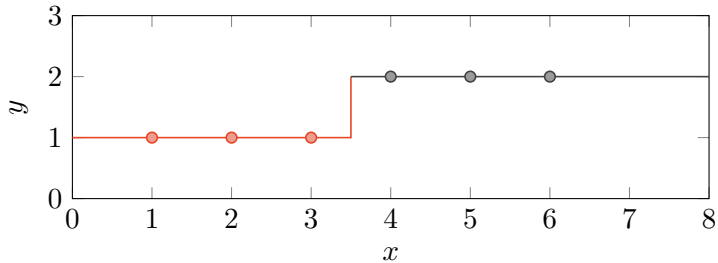
- Assign the output for the closest input, i.e. $f_{\mathcal{D}}(x) = y_i$ where $i = \arg\min_i d(x, x_i)$



- Strong baseline in many classification problems (for a reasonable distance measure)

Nearest Neighbor Classification (Cover and Hart 1967)

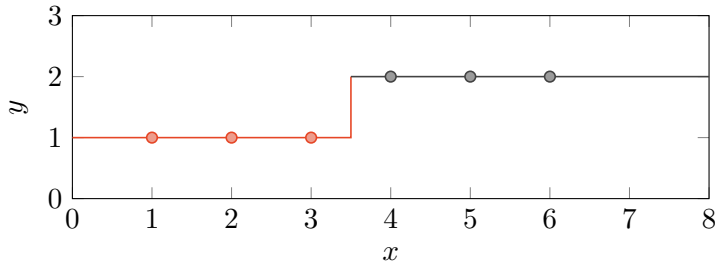
- Assign the output for the closest input, i.e. $f_{\mathcal{D}}(x) = y_i$ where $i = \arg\min_i d(x, x_i)$



- Strong baseline in many classification problems (for a reasonable distance measure)
- straightforward extension: majority vote of k nearest neighbors

Nearest Neighbor Classification (Cover and Hart 1967)

- Assign the output for the closest input, i.e. $f_{\mathcal{D}}(x) = y_i$ where $i = \arg\min_i d(x, x_i)$



- Strong baseline in many classification problems (for a reasonable distance measure)
- straightforward extension: majority vote of k nearest neighbors
- possible extensions for **metric learning** (Weinberger and Saul 2009)

Smoothness in Classification

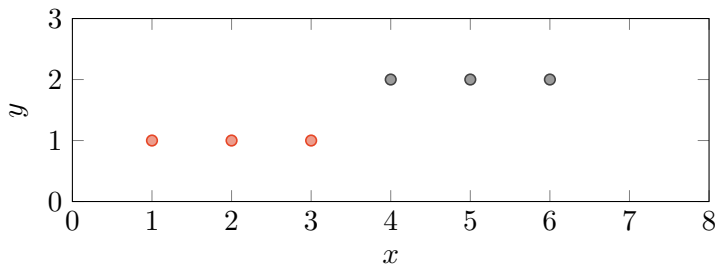
- Because $\mathcal{Y}$ is discrete, a classifier f can never be smooth

Smoothness in Classification

- Because $\mathcal{Y}$ is discrete, a classifier f can never be smooth
- **but** we can assume that f is generated as $f(x) = \arg\max_l g_l(x)$ for smooth functions $g_l : \mathcal{X} \to \mathbb{R}$.

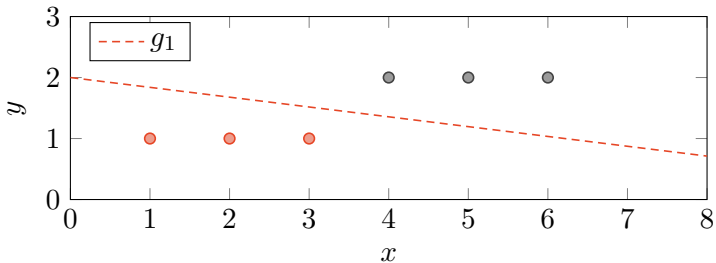
Smoothness in Classification

- Because $\mathcal{Y}$ is discrete, a classifier f can never be smooth
- **but** we can assume that f is generated as $f(x) = \arg\max_l g_l(x)$ for smooth functions $g_l : \mathcal{X} \to \mathbb{R}$.



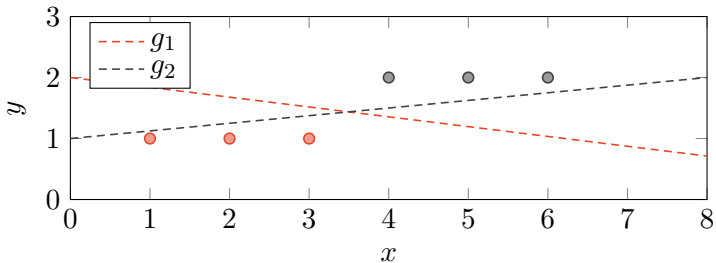
Smoothness in Classification

- Because $\mathcal{Y}$ is discrete, a classifier f can never be smooth
- **but** we can assume that f is generated as $f(x) = \arg\max_l g_l(x)$ for smooth functions $g_l : \mathcal{X} \to \mathbb{R}$.



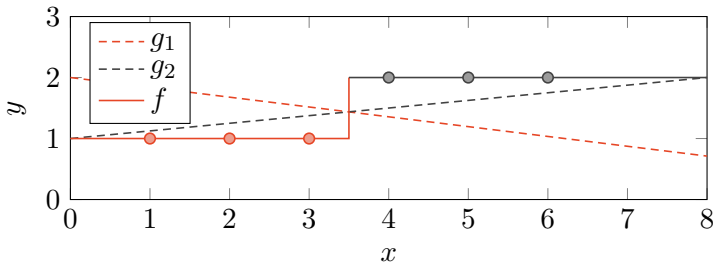
Smoothness in Classification

- Because $\mathcal{Y}$ is discrete, a classifier f can never be smooth
- **but** we can assume that f is generated as $f(x) = \arg\max_l g_l(x)$ for smooth functions $g_l : \mathcal{X} \to \mathbb{R}$.



Smoothness in Classification

- Because $\mathcal{Y}$ is discrete, a classifier f can never be smooth
- **but** we can assume that f is generated as $f(x) = \arg\max_l g_l(x)$ for smooth functions $g_l : \mathcal{X} \to \mathbb{R}$.



Support Vector Machine

(Suykens and Vandewalle 1999)

- Consider only 2-class problems, set $g_1(x) = 0$ and assume g_2 is a kernel regression function

Support Vector Machine

(Suykens and Vandewalle 1999)

- Consider only 2-class problems, set $g_1(x) = 0$ and assume g_2 is a kernel regression function, i.e. $g_2(x) = \vec{k}(x)^T \cdot \vec{\alpha}$

Support Vector Machine

(Suykens and Vandewalle 1999)

- Consider only 2-class problems, set $g_1(x) = 0$ and assume g_2 is a kernel regression function, i.e. $g_2(x) = \vec{k}(x)^T \cdot \vec{\alpha} + b$

Support Vector Machine

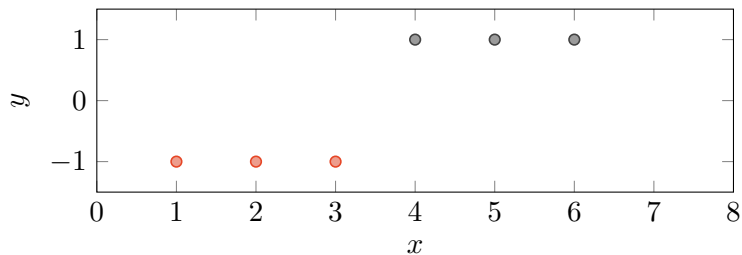
(Suykens and Vandewalle 1999)

- Consider only 2-class problems, set $g_1(x) = 0$ and assume g_2 is a kernel regression function, i.e. $g_2(x) = \vec{k}(x)^T \cdot \vec{\alpha} + b$
- Then, we wish to find the **smoothest** function g_2 that still ensures a **margin of safety** for classification:

Support Vector Machine

(Suykens and Vandewalle 1999)

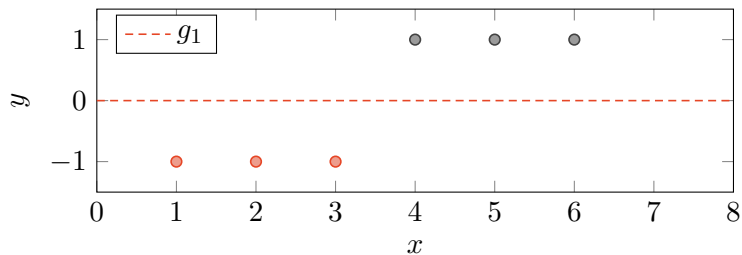
- Consider only 2-class problems, set $g_1(x) = 0$ and assume g_2 is a kernel regression function, i.e. $g_2(x) = \vec{k}(x)^T \cdot \vec{\alpha} + b$
- Then, we wish to find the **smoothest** function g_2 that still ensures a **margin of safety** for classification:



Support Vector Machine

(Suykens and Vandewalle 1999)

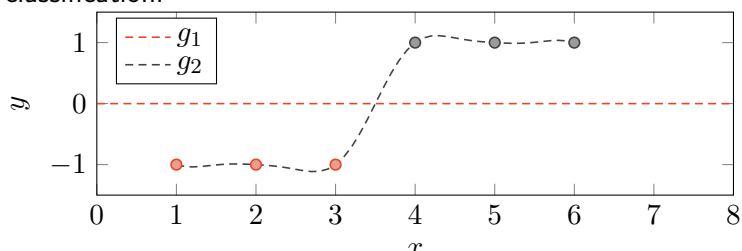
- Consider only 2-class problems, set $g_1(x) = 0$ and assume g_2 is a kernel regression function, i.e. $g_2(x) = \vec{k}(x)^T \cdot \vec{\alpha} + b$
- Then, we wish to find the **smoothest** function g_2 that still ensures a **margin of safety** for classification:



Support Vector Machine

(Suykens and Vandewalle 1999)

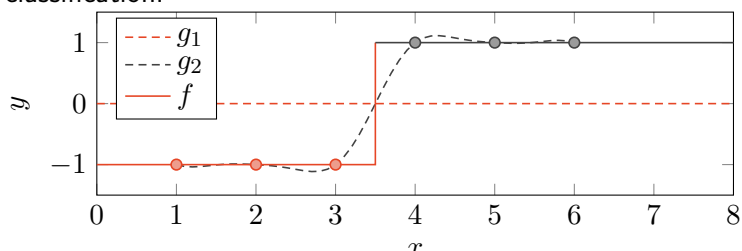
- Consider only 2-class problems, set $g_1(x) = 0$ and assume g_2 is a kernel regression function, i.e. $g_2(x) = \vec{k}(x)^T \cdot \vec{\alpha} + b$
- Then, we wish to find the **smoothest** function g_2 that still ensures a **margin of safety** for classification:



Support Vector Machine

(Suykens and Vandewalle 1999)

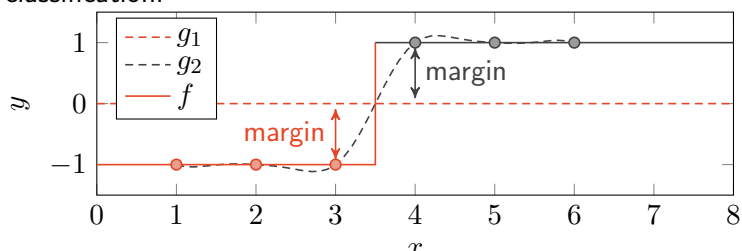
- Consider only 2-class problems, set $g_1(x) = 0$ and assume g_2 is a kernel regression function, i.e. $g_2(x) = \vec{k}(x)^T \cdot \vec{\alpha} + b$
- Then, we wish to find the **smoothest** function g_2 that still ensures a **margin of safety** for classification:



Support Vector Machine

(Suykens and Vandewalle 1999)

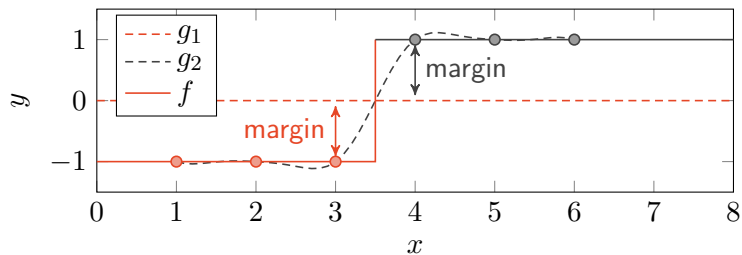
- Consider only 2-class problems, set $g_1(x) = 0$ and assume g_2 is a kernel regression function, i.e. $g_2(x) = \vec{k}(x)^T \cdot \vec{\alpha} + b$
- Then, we wish to find the **smoothest** function g_2 that still ensures a **margin of safety** for classification:



Support Vector Machine

(Suykens and Vandewalle 1999)

- Consider only 2-class problems, set $g_1(x) = 0$ and assume g_2 is a kernel regression function, i.e. $g_2(x) = \vec{k}(x)^T \cdot \vec{\alpha} + b$
- Then, we wish to find the **smoothest** function g_2 that still ensures a **margin of safety** for classification:

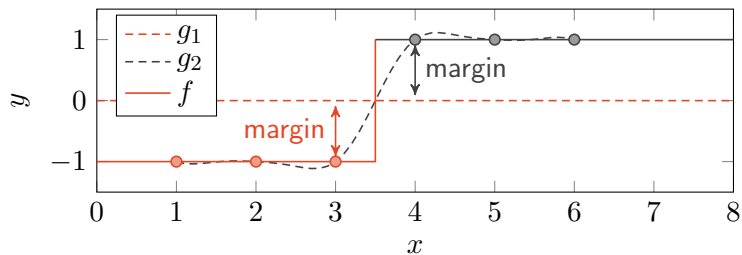


- We impose the constraints: $g_2(x_i) \geq 1$ if $y_i = 2$ and $g_2(x_i) \leq -1$ if $y_i = 1$

Support Vector Machine

(Suykens and Vandewalle 1999)

- Consider only 2-class problems, set $g_1(x) = 0$ and assume g_2 is a kernel regression function, i.e. $g_2(x) = \vec{k}(x)^T \cdot \vec{\alpha} + b$
- Then, we wish to find the **smoothest** function g_2 that still ensures a **margin of safety** for classification:

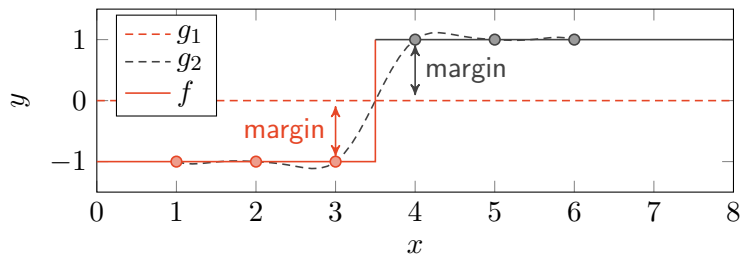


- We impose the constraints: $g_2(x_i) \geq 1 - \epsilon_i$ if $y_i = 2$ and $g_2(x_i) \leq -1 + \epsilon_i$ if $y_i = 1$

Support Vector Machine

(Suykens and Vandewalle 1999)

- Consider only 2-class problems, set $g_1(x) = 0$ and assume g_2 is a kernel regression function, i.e. $g_2(x) = \vec{k}(x)^T \cdot \vec{\alpha} + b$
- Then, we wish to find the **smoothest** function g_2 that still ensures a **margin of safety** for classification:



- We impose the constraints: $g_2(x_i) \geq 1 - \epsilon_i$ if $y_i = 2$ and $g_2(x_i) \leq -1 + \epsilon_i$ if $y_i = 1$ for **slack variables** ϵ_i

Support Vector Machine (II)

- We obtain the optimization problem:

$$\min_{\vec{\alpha},b,\vec{\epsilon}} \quad \frac{C}{2} \cdot \sum_{i=1}^{N} \epsilon_i^2 + \frac{1}{2} \vec{\alpha}^T \cdot \boldsymbol{K} \cdot \vec{\alpha}$$

Support Vector Machine (II)

- We obtain the optimization problem:

$$\begin{aligned} \min_{\vec{\alpha},b,\vec{\epsilon}} \quad & \frac{C}{2} \cdot \sum_{i=1}^{N} \epsilon_i^2 + \frac{1}{2} \vec{\alpha}^T \cdot \boldsymbol{K} \cdot \vec{\alpha} & \\ \text{s.t.} \quad & \vec{k}_i^T \cdot \vec{\alpha} + b \geq 1 - \epsilon_i & \forall i : y_i = 2 \\ & \vec{k}_i^T \cdot \vec{\alpha} + b \leq -1 + \epsilon_i & \forall i : y_i = 1 \end{aligned}$$

Support Vector Machine (II)

- We obtain the optimization problem:

$$\begin{aligned} \min_{\vec{\alpha},b,\vec{\epsilon}} \quad & \frac{C}{2} \cdot \sum_{i=1}^{N} \epsilon_i^2 + \frac{1}{2} \vec{\alpha}^T \cdot \boldsymbol{K} \cdot \vec{\alpha} \\ \text{s.t.} \quad & (2 \cdot y_i - 3) \cdot (\vec{k}_i^T \cdot \vec{\alpha} + b) \geq 1 - \epsilon_i \qquad \forall i \end{aligned}$$

Support Vector Machine (II)

- We obtain the optimization problem:

$$\begin{aligned} \min_{\vec{\alpha},b,\vec{\epsilon}} \quad & \frac{C}{2} \cdot \sum_{i=1}^{N} \epsilon_i^2 + \frac{1}{2} \vec{\alpha}^T \cdot \boldsymbol{K} \cdot \vec{\alpha} \\ \text{s.t.} \quad & \tilde{y}_i \cdot (\vec{k}_i^T \cdot \vec{\alpha} + b) \geq 1 - \epsilon_i \qquad \forall i \end{aligned}$$

Support Vector Machine (II)

- We obtain the optimization problem:

$$\begin{aligned} \min_{\vec{\alpha},b,\vec{\epsilon}} \quad & \frac{C}{2} \cdot \sum_{i=1}^{N} \epsilon_i^2 + \frac{1}{2} \vec{\alpha}^T \cdot \boldsymbol{K} \cdot \vec{\alpha} \\ \text{s.t.} \quad & \tilde{y}_i \cdot (\vec{k}_i^T \cdot \vec{\alpha} + b) \geq 1 - \epsilon_i \qquad \forall i \end{aligned}$$

- Nicer form via **Wolfe dual** (Paaßen 2019; Boyd and Vandenberghe 2004):

$$\begin{aligned} \min_{\vec{\alpha}} \quad & \frac{1}{2} \vec{\alpha}^T \cdot \left(\boldsymbol{K} + \frac{1}{C} \boldsymbol{I} \right) \cdot \vec{\alpha} - \tilde{y}^T \cdot \vec{\alpha} \\ \text{s.t.} \quad & \sum_{i=1}^{N} \alpha_i = 0 \qquad \text{and} \qquad \alpha_i \cdot \tilde{y}_i \geq 0 \qquad \forall i \end{aligned}$$

Support Vector Machine (II)

- We obtain the optimization problem:

$$\begin{aligned} \min_{\vec{\alpha},b,\vec{\epsilon}} \quad & \frac{C}{2} \cdot \sum_{i=1}^{N} \epsilon_i^2 + \frac{1}{2} \vec{\alpha}^T \cdot \boldsymbol{K} \cdot \vec{\alpha} \\ \text{s.t.} \quad & \tilde{y}_i \cdot (\vec{k}_i^T \cdot \vec{\alpha} + b) \geq 1 - \epsilon_i \qquad \forall i \end{aligned}$$

- Nicer form via **Wolfe dual** (Paaßen 2019; Boyd and Vandenberghe 2004):

$$\begin{aligned} \min_{\vec{\alpha}} \quad & \frac{1}{2} \vec{\alpha}^T \cdot \left(\boldsymbol{K} + \frac{1}{C} \boldsymbol{I} \right) \cdot \vec{\alpha} - \tilde{y}^T \cdot \vec{\alpha} \\ \text{s.t.} \quad & \sum_{i=1}^{N} \alpha_i = 0 \qquad \text{and} \qquad \alpha_i \cdot \tilde{y}_i \geq 0 \qquad \forall i \end{aligned}$$

- problem is still convex, but constrained

Support Vector Machine (II)

- We obtain the optimization problem:

$$\begin{aligned} \min_{\vec{\alpha},b,\vec{\epsilon}} \quad & \frac{C}{2} \cdot \sum_{i=1}^{N} \epsilon_i^2 + \frac{1}{2} \vec{\alpha}^T \cdot \boldsymbol{K} \cdot \vec{\alpha} \\ \text{s.t.} \quad & \tilde{y}_i \cdot (\vec{k}_i^T \cdot \vec{\alpha} + b) \geq 1 - \epsilon_i \qquad \forall i \end{aligned}$$

- Nicer form via **Wolfe dual** (Paaßen 2019; Boyd and Vandenberghe 2004):

$$\begin{aligned} \min_{\vec{\alpha}} \quad & \frac{1}{2} \vec{\alpha}^T \cdot \left(\boldsymbol{K} + \frac{1}{C} \boldsymbol{I} \right) \cdot \vec{\alpha} - \tilde{y}^T \cdot \vec{\alpha} \\ \text{s.t.} \quad & \sum_{i=1}^{N} \alpha_i = 0 \qquad \text{and} \qquad \alpha_i \cdot \tilde{y}_i \geq 0 \qquad \forall i \end{aligned}$$

- problem is still convex, but constrained $\Rightarrow$ new techniques needed, esp. quadratic programming

Support Vector Machine (II)

- We obtain the optimization problem:

$$\begin{aligned} \min_{\vec{\alpha},b,\vec{\epsilon}} \quad & \frac{C}{2} \cdot \sum_{i=1}^{N} \epsilon_i^2 + \frac{1}{2} \vec{\alpha}^T \cdot \boldsymbol{K} \cdot \vec{\alpha} \\ \text{s.t.} \quad & \tilde{y}_i \cdot (\vec{k}_i^T \cdot \vec{\alpha} + b) \geq 1 - \epsilon_i \qquad \forall i \end{aligned}$$

- Nicer form via **Wolfe dual** (Paaßen 2019; Boyd and Vandenberghe 2004):

$$\begin{aligned} \min_{\vec{\alpha}} \quad & \frac{1}{2} \vec{\alpha}^T \cdot \left(\boldsymbol{K} + \frac{1}{C} \boldsymbol{I} \right) \cdot \vec{\alpha} - \tilde{y}^T \cdot \vec{\alpha} \\ \text{s.t.} \quad & \sum_{i=1}^{N} \alpha_i = 0 \qquad \text{and} \qquad \alpha_i \cdot \tilde{y}_i \geq 0 \qquad \forall i \end{aligned}$$

- problem is still convex, but constrained $\Rightarrow$ new techniques needed, esp. quadratic programming
- Extensions to multiple classes by means of one-versus-one classification

Unsupervised Learning

Unsupervised Learning

- In many settings, we do not actually have ground truth examples for the **output** (e.g. exploratory data analysis)

Unsupervised Learning

- In many settings, we do not actually have ground truth examples for the **output** (e.g. exploratory data analysis) $\Rightarrow$ Learning is then called **unsupervised**

Unsupervised Learning

- In many settings, we do not actually have ground truth examples for the **output** (e.g. exploratory data analysis) $\Rightarrow$ Learning is then called **unsupervised**
- Mostly based on **reconstruction** or **compression**

Unsupervised Learning

- In many settings, we do not actually have ground truth examples for the **output** (e.g. exploratory data analysis) $\Rightarrow$ Learning is then called **unsupervised**
- Mostly based on **reconstruction** or **compression**

Reconstruction Principle

Let $\mathcal{X}$ be some set, p be a density on it, and d be a distance on that set.

Unsupervised Learning

- In many settings, we do not actually have ground truth examples for the **output** (e.g. exploratory data analysis) $\Rightarrow$ Learning is then called **unsupervised**
- Mostly based on **reconstruction** or **compression**

Reconstruction Principle

Let $\mathcal{X}$ be some set, p be a density on it, and d be a distance on that set. Our goal is to find two models $f : \mathcal{X} \to \mathcal{Y}$ and $f^{-1} : \mathcal{Y} \to \mathcal{X}$ for some "smaller" set $\mathcal{Y}$

Unsupervised Learning

- In many settings, we do not actually have ground truth examples for the **output** (e.g. exploratory data analysis) $\Rightarrow$ Learning is then called **unsupervised**
- Mostly based on **reconstruction** or **compression**

Reconstruction Principle

Let $\mathcal{X}$ be some set, p be a density on it, and d be a distance on that set. Our goal is to find two models $f : \mathcal{X} \to \mathcal{Y}$ and $f^{-1} : \mathcal{Y} \to \mathcal{X}$ for some "smaller" set $\mathcal{Y}$, such that

$$\ell_{\text{rec}}(f, f^{-1}) = \int d\big(x, f^{-1}(f(x))\big)^2 \cdot p(x) dx$$

is as small as possible. We call ℓ_{rec} the **reconstruction** loss.

Unsupervised Learning

- In many settings, we do not actually have ground truth examples for the **output** (e.g. exploratory data analysis) $\Rightarrow$ Learning is then called **unsupervised**
- Mostly based on **reconstruction** or **compression**

Reconstruction Principle

Let $\mathcal{X}$ be some set, p be a density on it, and d be a distance on that set. Our goal is to find two models $f : \mathcal{X} \to \mathcal{Y}$ and $f^{-1} : \mathcal{Y} \to \mathcal{X}$ for some "smaller" set $\mathcal{Y}$, such that

$$\ell_{\text{rec}}(f, f^{-1}) = \int d\big(x, f^{-1}(f(x))\big)^2 \cdot p(x)dx \approx \frac{1}{N}\sum_{i=1}^{N} d\big(x_i, f^{-1}(f(x)\big)^2$$

is as small as possible. We call ℓ_{rec} the **reconstruction** loss.

Example: Constant Coding

- Idea: We compress every point to a constant: $f(\vec{x}) = \vec{c}$

Example: Constant Coding

- Idea: We compress every point to a constant: $f(\vec{x}) = \vec{c}$ and $f^{-1}(\vec{c}) = \vec{c}$

Example: Constant Coding

- Idea: We compress every point to a constant: $f(\vec{x}) = \vec{c}$ and $f^{-1}(\vec{c}) = \vec{c}$
- What is the best constant $\vec{c}$?

Example: Constant Coding

- Idea: We compress every point to a constant: $f(\vec{x}) = \vec{c}$ and $f^{-1}(\vec{c}) = \vec{c}$
- What is the best constant $\vec{c}$? $\Rightarrow$ Minimize reconstruction loss

Example: Constant Coding

- Idea: We compress every point to a constant: $f(\vec{x}) = \vec{c}$ and $f^{-1}(\vec{c}) = \vec{c}$
- What is the best constant $\vec{c}$? $\Rightarrow$ Minimize reconstruction loss

$$\min_{\vec{c}} \quad \sum_{i=1}^{N} \left(\vec{x}_i - f^{-1}(f(\vec{x}))\right)^2$$

Example: Constant Coding

- Idea: We compress every point to a constant: $f(\vec{x}) = \vec{c}$ and $f^{-1}(\vec{c}) = \vec{c}$
- What is the best constant $\vec{c}$? $\Rightarrow$ Minimize reconstruction loss

$$\min_{\vec{c}} \quad \sum_{i=1}^{N} \left(\vec{x}_i - \vec{c}\right)^2$$

Example: Constant Coding

- Idea: We compress every point to a constant: $f(\vec{x}) = \vec{c}$ and $f^{-1}(\vec{c}) = \vec{c}$
- What is the best constant $\vec{c}$? $\Rightarrow$ Minimize reconstruction loss

$$\min_{\vec{c}} \quad \sum_{i=1}^{N} \left(\vec{x}_i - \vec{c}\right)^2$$

$$\nabla_{\vec{c}} \ell_{\text{rec}} = \sum_{i=1}^{N} 2(\vec{c} - \vec{x}_i)$$

Example: Constant Coding

- Idea: We compress every point to a constant: $f(\vec{x}) = \vec{c}$ and $f^{-1}(\vec{c}) = \vec{c}$
- What is the best constant $\vec{c}$? $\Rightarrow$ Minimize reconstruction loss

$$\min_{\vec{c}} \quad \sum_{i=1}^{N} \left(\vec{x}_i - \vec{c}\right)^2$$

$$\nabla_{\vec{c}} \ell_{\text{rec}} = \sum_{i=1}^{N} 2(\vec{c} - \vec{x}_i) \stackrel{!}{=} 0$$

Example: Constant Coding

- Idea: We compress every point to a constant: $f(\vec{x}) = \vec{c}$ and $f^{-1}(\vec{c}) = \vec{c}$
- What is the best constant $\vec{c}$? $\Rightarrow$ Minimize reconstruction loss

$$\min_{\vec{c}} \quad \sum_{i=1}^{N} \left(\vec{x}_i - \vec{c}\right)^2$$

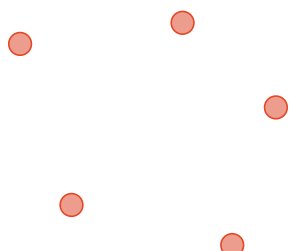
$$\nabla_{\vec{c}} \ell_{\text{rec}} = \sum_{i=1}^{N} 2(\vec{c} - \vec{x}_i) \stackrel{!}{=} 0 \qquad \Longleftrightarrow \qquad \vec{c} = \frac{1}{N} \sum_{i=1}^{N} \vec{x}_i$$

Example: Constant Coding

- Idea: We compress every point to a constant: $f(\vec{x}) = \vec{c}$ and $f^{-1}(\vec{c}) = \vec{c}$
- What is the best constant $\vec{c}$? $\Rightarrow$ Minimize reconstruction loss

$$\min_{\vec{c}} \quad \sum_{i=1}^{N} \left(\vec{x}_i - \vec{c}\right)^2$$

$$\nabla_{\vec{c}} \ell_{\text{rec}} = \sum_{i=1}^{N} 2(\vec{c} - \vec{x}_i) \stackrel{!}{=} 0 \qquad \Longleftrightarrow \qquad \vec{c} = \frac{1}{N} \sum_{i=1}^{N} \vec{x}_i$$

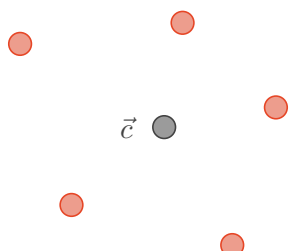


Example: Constant Coding

- Idea: We compress every point to a constant: $f(\vec{x}) = \vec{c}$ and $f^{-1}(\vec{c}) = \vec{c}$
- What is the best constant $\vec{c}$? $\Rightarrow$ Minimize reconstruction loss

$$\min_{\vec{c}} \quad \sum_{i=1}^{N} \left(\vec{x}_i - \vec{c}\right)^2$$

$$\nabla_{\vec{c}} \ell_{\text{rec}} = \sum_{i=1}^{N} 2(\vec{c} - \vec{x}_i) \stackrel{!}{=} 0 \qquad \Longleftrightarrow \qquad \vec{c} = \frac{1}{N} \sum_{i=1}^{N} \vec{x}_i$$

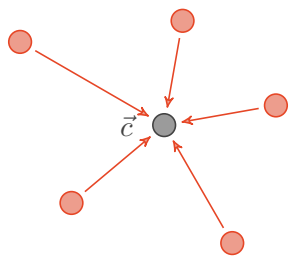


Example: Constant Coding

- Idea: We compress every point to a constant: $f(\vec{x}) = \vec{c}$ and $f^{-1}(\vec{c}) = \vec{c}$
- What is the best constant $\vec{c}$? $\Rightarrow$ Minimize reconstruction loss

$$\min_{\vec{c}} \quad \sum_{i=1}^{N} \left(\vec{x}_i - \vec{c}\right)^2$$

$$\nabla_{\vec{c}} \ell_{\text{rec}} = \sum_{i=1}^{N} 2(\vec{c} - \vec{x}_i) \stackrel{!}{=} 0 \qquad \Longleftrightarrow \qquad \vec{c} = \frac{1}{N} \sum_{i=1}^{N} \vec{x}_i$$



Dimensionality Reduction

Principal Component Analysis

(Pearson 1901; Bishop 2006)

- Idea: Find a **linear** function $f : \mathbb{R}^n \to \mathbb{R}^m$ with $m \ll n$ that reconstructs points well

Principal Component Analysis

(Pearson 1901; Bishop 2006)

- Idea: Find a **linear** function $f : \mathbb{R}^n \to \mathbb{R}^m$ with $m \ll n$ that reconstructs points well
- More precisely: $f(\vec{x}) = \boldsymbol{W} \cdot (\vec{x} - \vec{b})$ and $f^{-1}(\vec{y}) = \boldsymbol{V} \cdot \vec{y} + \vec{b}$ with parameters $\boldsymbol{W} \in \mathbb{R}^{m \times n}$, $\boldsymbol{V} \in \mathbb{R}^{n \times m}$, and $\vec{b} \in \mathbb{R}^n$

Principal Component Analysis

(Pearson 1901; Bishop 2006)

- Idea: Find a **linear** function $f : \mathbb{R}^n \to \mathbb{R}^m$ with $m \ll n$ that reconstructs points well
- More precisely: $f(\vec{x}) = \boldsymbol{W} \cdot (\vec{x} - \vec{b})$ and $f^{-1}(\vec{y}) = \boldsymbol{V} \cdot \vec{y} + \vec{b}$ with parameters $\boldsymbol{W} \in \mathbb{R}^{m \times n}$, $\boldsymbol{V} \in \mathbb{R}^{n \times m}$, and $\vec{b} \in \mathbb{R}^n$
- Derivation is, sadly, out of scope (refer to e.g. Bishop (2006) instead)

Principal Component Analysis

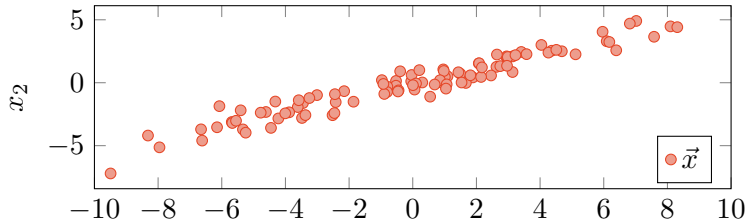
(Pearson 1901; Bishop 2006)

- Idea: Find a **linear** function $f : \mathbb{R}^n \to \mathbb{R}^m$ with $m \ll n$ that reconstructs points well
- More precisely: $f(\vec{x}) = \boldsymbol{W} \cdot (\vec{x} - \vec{b})$ and $f^{-1}(\vec{y}) = \boldsymbol{V} \cdot \vec{y} + \vec{b}$ with parameters $\boldsymbol{W} \in \mathbb{R}^{m \times n}$, $\boldsymbol{V} \in \mathbb{R}^{n \times m}$, and $\vec{b} \in \mathbb{R}^n$
- Derivation is, sadly, out of scope (refer to e.g. Bishop (2006) instead)
- Key points: $\vec{b}$ is the data mean; $\boldsymbol{W}$ are eigenvectors of the data covariance matrix corresponding to the largest eigenvalues; $\boldsymbol{V} = \boldsymbol{W}^T$

Principal Component Analysis

(Pearson 1901; Bishop 2006)

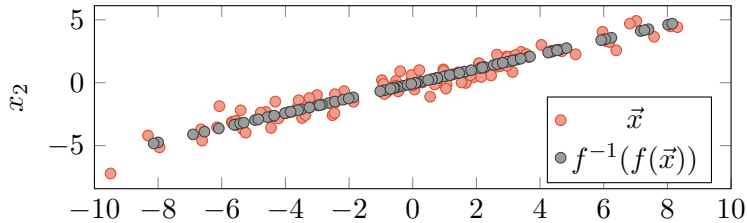
- Idea: Find a **linear** function $f : \mathbb{R}^n \to \mathbb{R}^m$ with $m \ll n$ that reconstructs points well
- More precisely: $f(\vec{x}) = \boldsymbol{W} \cdot (\vec{x} - \vec{b})$ and $f^{-1}(\vec{y}) = \boldsymbol{V} \cdot \vec{y} + \vec{b}$ with parameters $\boldsymbol{W} \in \mathbb{R}^{m \times n}$, $\boldsymbol{V} \in \mathbb{R}^{n \times m}$, and $\vec{b} \in \mathbb{R}^n$
- Derivation is, sadly, out of scope (refer to e.g. Bishop (2006) instead)
- Key points: $\vec{b}$ is the data mean; $\boldsymbol{W}$ are eigenvectors of the data covariance matrix corresponding to the largest eigenvalues; $\boldsymbol{V} = \boldsymbol{W}^T$



Principal Component Analysis

(Pearson 1901; Bishop 2006)

- Idea: Find a **linear** function $f : \mathbb{R}^n \to \mathbb{R}^m$ with $m \ll n$ that reconstructs points well
- More precisely: $f(\vec{x}) = \boldsymbol{W} \cdot (\vec{x} - \vec{b})$ and $f^{-1}(\vec{y}) = \boldsymbol{V} \cdot \vec{y} + \vec{b}$ with parameters $\boldsymbol{W} \in \mathbb{R}^{m \times n}$, $\boldsymbol{V} \in \mathbb{R}^{n \times m}$, and $\vec{b} \in \mathbb{R}^n$
- Derivation is, sadly, out of scope (refer to e.g. Bishop (2006) instead)
- Key points: $\vec{b}$ is the data mean; $\boldsymbol{W}$ are eigenvectors of the data covariance matrix corresponding to the largest eigenvalues; $\boldsymbol{V} = \boldsymbol{W}^T$



t-SNE

(Van der Maaten and Hinton 2008)

- Idea: Find low-dimensional points $\vec{y}_1, \ldots, \vec{y}_N$ **directly**, such that **closeness** is maintained

t-SNE

(Van der Maaten and Hinton 2008)

- Idea: Find low-dimensional points $\vec{y}_1, \ldots, \vec{y}_N$ **directly**, such that **closeness** is maintained
- High-dimensional closeness of j to i: $p_{j|i} = \exp\left(-\frac{1}{2}\frac{d(x_i,x_j)^2}{\sigma_i^2}\right)$

t-SNE

(Van der Maaten and Hinton 2008)

- Idea: Find low-dimensional points $\vec{y}_1, \ldots, \vec{y}_N$ **directly**, such that **closeness** is maintained
- High-dimensional closeness of j to i: $p_{j|i} = \exp\left(-\frac{1}{2}\frac{d(x_i,x_j)^2}{\sigma_i^2}\right)$; normalize by sum over all j

t-SNE

(Van der Maaten and Hinton 2008)

- Idea: Find low-dimensional points $\vec{y}_1, \ldots, \vec{y}_N$ **directly**, such that **closeness** is maintained
- High-dimensional closeness of j to i: $p_{j|i} = \exp\left(-\frac{1}{2}\frac{d(x_i,x_j)^2}{\sigma_i^2}\right)$; normalize by sum over all j; symmetrize as $p_{i,j} = \frac{1}{2 \cdot N}(p_{j|i} + p_{i|j})$

t-SNE

(Van der Maaten and Hinton 2008)

- Idea: Find low-dimensional points $\vec{y}_1, \ldots, \vec{y}_N$ **directly**, such that **closeness** is maintained
- High-dimensional closeness of j to i: $p_{j|i} = \exp\left(-\frac{1}{2}\frac{d(x_i,x_j)^2}{\sigma_i^2}\right)$; normalize by sum over all j; symmetrize as $p_{i,j} = \frac{1}{2 \cdot N}(p_{j|i} + p_{i|j})$
- Low-dimensional closeness: $q_{i,j} = 1/\left(1 + d_{\mathsf{euc}}(y_i, y_j)^2\right)$

t-SNE

(Van der Maaten and Hinton 2008)

- Idea: Find low-dimensional points $\vec{y}_1, \ldots, \vec{y}_N$ **directly**, such that **closeness** is maintained
- High-dimensional closeness of j to i: $p_{j|i} = \exp\left(-\frac{1}{2}\frac{d(x_i,x_j)^2}{\sigma_i^2}\right)$; normalize by sum over all j; symmetrize as $p_{i,j} = \frac{1}{2 \cdot N}(p_{j|i} + p_{i|j})$
- Low-dimensional closeness: $q_{i,j} = 1/\left(1 + d_{\mathsf{euc}}(y_i, y_j)^2\right)$
- Loss function: Kullback-Leibler divergence:

$$\ell_{\mathsf{KL}}(\vec{y}_1, \ldots, \vec{y}_N) = \sum_{i=1}^{N} \sum_{j=1}^{N} p_{i,j} \cdot \log\left(\frac{p_{i,j}}{q_{i,j}}\right)$$

t-SNE

(Van der Maaten and Hinton 2008)

- Idea: Find low-dimensional points $\vec{y}_1, \ldots, \vec{y}_N$ **directly**, such that **closeness** is maintained
- High-dimensional closeness of j to i: $p_{j|i} = \exp\left(-\frac{1}{2}\frac{d(x_i,x_j)^2}{\sigma_i^2}\right)$; normalize by sum over all j; symmetrize as $p_{i,j} = \frac{1}{2 \cdot N}(p_{j|i} + p_{i|j})$
- Low-dimensional closeness: $q_{i,j} = 1/\left(1 + d_{\mathsf{euc}}(y_i, y_j)^2\right)$
- Loss function: Kullback-Leibler divergence:

$$\ell_{\mathsf{KL}}(\vec{y}_1, \ldots, \vec{y}_N) = \sum_{i=1}^{N}\sum_{j=1}^{N} p_{i,j} \cdot \log\left(\frac{p_{i,j}}{q_{i,j}}\right)$$

- Optimization via gradient descent/related methods

t-SNE

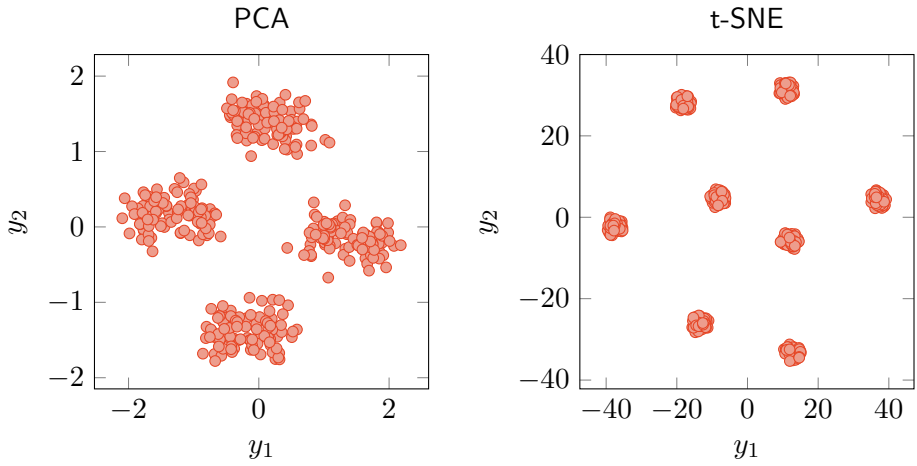
(Van der Maaten and Hinton 2008)

- Idea: Find low-dimensional points $\vec{y}_1, \ldots, \vec{y}_N$ **directly**, such that **closeness** is maintained
- High-dimensional closeness of j to i: $p_{j|i} = \exp\left(-\frac{1}{2}\frac{d(x_i,x_j)^2}{\sigma_i^2}\right)$; normalize by sum over all j; symmetrize as $p_{i,j} = \frac{1}{2 \cdot N}(p_{j|i} + p_{i|j})$
- Low-dimensional closeness: $q_{i,j} = 1/\left(1 + d_{\mathsf{euc}}(y_i, y_j)^2\right)$
- Loss function: Kullback-Leibler divergence:

$$\ell_{\mathsf{KL}}(\vec{y}_1, \ldots, \vec{y}_N) = \sum_{i=1}^{N}\sum_{j=1}^{N} p_{i,j} \cdot \log\left(\frac{p_{i,j}}{q_{i,j}}\right)$$

- Optimization via gradient descent/related methods
- Challenge: Extending map to new data points; refer e.g. to Gisbrecht, Schulz, and Hammer (2015)

PCA versus t-SNE example



Notes on PCA versus t-SNE

Rules of thumb:

- PCA is very fast and easily applicable to new data; very useful e.g. as pre-processing for big data

Notes on PCA versus t-SNE

Rules of thumb:

- PCA is very fast and easily applicable to new data; very useful e.g. as pre-processing for big data
- t-SNE is better suited for visualizations and insight; especially for clustered data

Notes on PCA versus t-SNE

Rules of thumb:

- PCA is very fast and easily applicable to new data; very useful e.g. as pre-processing for big data
- t-SNE is better suited for visualizations and insight; especially for clustered data
- Evaluating the quality of dimensionality reduction is difficult, refer e.g. to Mokbel et al. (2013)

Clustering

Single-linkage clustering

(Sibson 1973)

- Idea: Represent data points by a **cluster index**: $f(x) \in \{1, \ldots, L\}$

Single-linkage clustering

(Sibson 1973)

- Idea: Represent data points by a **cluster index**: $f(x) \in \{1, \ldots, L\}$, $f^{-1}(l) = ?$

Single-linkage clustering

(Sibson 1973)

- Idea: Represent data points by a **cluster index**: $f(x) \in \{1, \ldots, L\}$, $f^{-1}(l) = ?$
- Algorithm: Start with one cluster per point

Single-linkage clustering

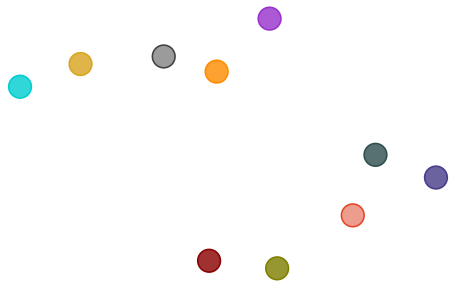
(Sibson 1973)

- Idea: Represent data points by a **cluster index**: $f(x) \in \{1, \ldots, L\}$, $f^{-1}(l) = ?$
- Algorithm: Start with one cluster per point, then merge clusters that are closest to each other

Single-linkage clustering

(Sibson 1973)

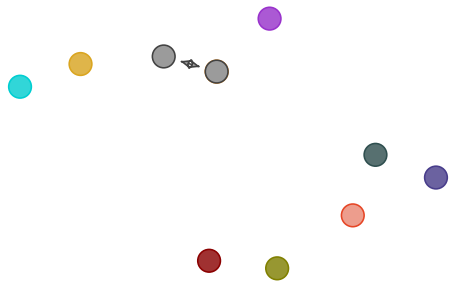
- Idea: Represent data points by a **cluster index**: $f(x) \in \{1, \ldots, L\}$, $f^{-1}(l) = ?$
- Algorithm: Start with one cluster per point, then merge clusters that are closest to each other



Single-linkage clustering

(Sibson 1973)

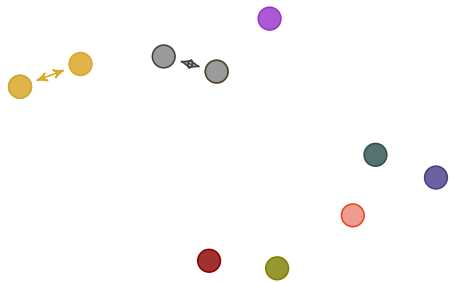
- Idea: Represent data points by a **cluster index**: $f(x) \in \{1, \ldots, L\}$, $f^{-1}(l) = ?$
- Algorithm: Start with one cluster per point, then merge clusters that are closest to each other



Single-linkage clustering

(Sibson 1973)

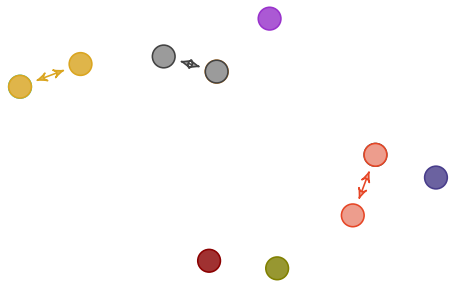
- Idea: Represent data points by a **cluster index**: $f(x) \in \{1, \ldots, L\}$, $f^{-1}(l) = ?$
- Algorithm: Start with one cluster per point, then merge clusters that are closest to each other



Single-linkage clustering

(Sibson 1973)

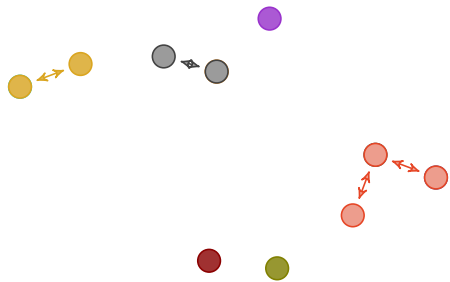
- Idea: Represent data points by a **cluster index**: $f(x) \in \{1, \ldots, L\}$, $f^{-1}(l) = ?$
- Algorithm: Start with one cluster per point, then merge clusters that are closest to each other



Single-linkage clustering

(Sibson 1973)

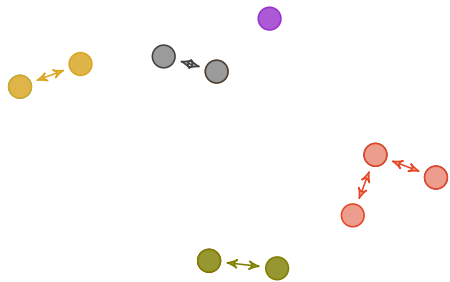
- Idea: Represent data points by a **cluster index**: $f(x) \in \{1, \ldots, L\}$, $f^{-1}(l) = ?$
- Algorithm: Start with one cluster per point, then merge clusters that are closest to each other



Single-linkage clustering

(Sibson 1973)

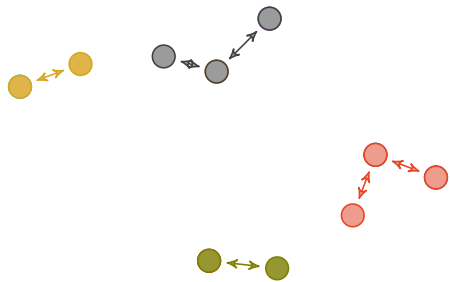
- Idea: Represent data points by a **cluster index**: $f(x) \in \{1, \ldots, L\}$, $f^{-1}(l) = ?$
- Algorithm: Start with one cluster per point, then merge clusters that are closest to each other



Single-linkage clustering

(Sibson 1973)

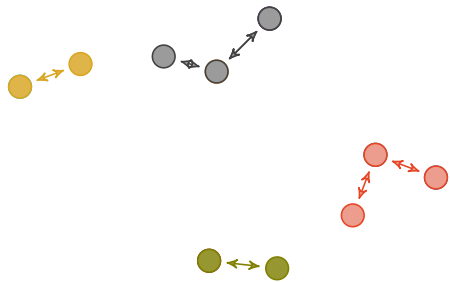
- Idea: Represent data points by a **cluster index**: $f(x) \in \{1, \ldots, L\}$, $f^{-1}(l) = ?$
- Algorithm: Start with one cluster per point, then merge clusters that are closest to each other



Single-linkage clustering

(Sibson 1973)

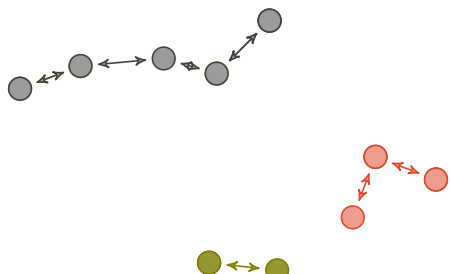
- Idea: Represent data points by a **cluster index**: $f(x) \in \{1, \ldots, L\}$, $f^{-1}(l) = ?$
- Algorithm: Start with one cluster per point, then merge clusters that are closest to each other



Single-linkage clustering

(Sibson 1973)

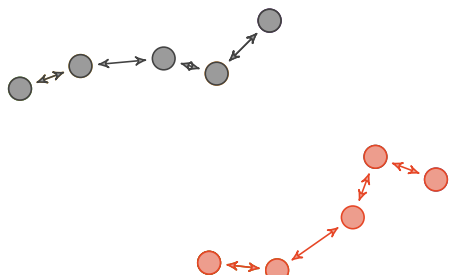
- Idea: Represent data points by a **cluster index**: $f(x) \in \{1, \ldots, L\}$, $f^{-1}(l) = ?$
- Algorithm: Start with one cluster per point, then merge clusters that are closest to each other



Single-linkage clustering

(Sibson 1973)

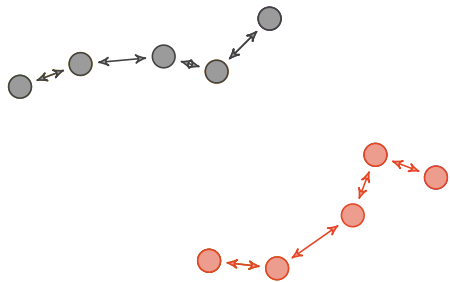
- Idea: Represent data points by a **cluster index**: $f(x) \in \{1, \ldots, L\}$, $f^{-1}(l) = ?$
- Algorithm: Start with one cluster per point, then merge clusters that are closest to each other



Single-linkage clustering

(Sibson 1973)

- Idea: Represent data points by a **cluster index**: $f(x) \in \{1, \ldots, L\}$, $f^{-1}(l) = ?$
- Algorithm: Start with one cluster per point, then merge clusters that are closest to each other

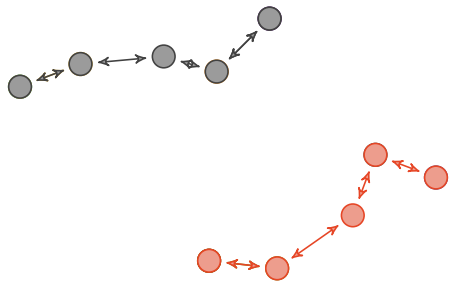


- Provides not only clusters but a dendrogram / "evolutionary tree"

Single-linkage clustering

(Sibson 1973)

- Idea: Represent data points by a **cluster index**: $f(x) \in \{1, \ldots, L\}$, $f^{-1}(l) = ?$
- Algorithm: Start with one cluster per point, then merge clusters that are closest to each other



- Provides not only clusters but a dendrogram / "evolutionary tree"
- Precise behavior depends on the definition of cluster closeness (Ward 1963)

K-means clustering (Hartigan and Wong 1979)

- Idea: Represent data points by a **prototype**: $f(x) = \vec{w}_k \in \mathbb{R}^n$ with $k \in \{1, \ldots, K\}$

K-means clustering (Hartigan and Wong 1979)

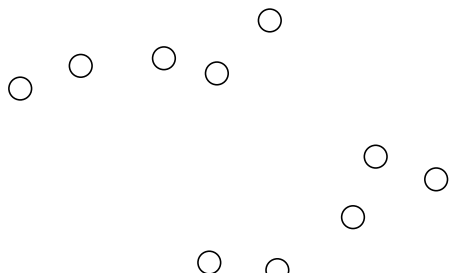
- Idea: Represent data points by a **prototype**: $f(x) = \vec{w}_k \in \mathbb{R}^n$ with $k \in \{1, \ldots, K\}$, $f^{-1}(\vec{w}_k) = \vec{w}_k$

K-means clustering (Hartigan and Wong 1979)

- Idea: Represent data points by a **prototype**: $f(x) = \vec{w}_k \in \mathbb{R}^n$ with $k \in \{1, \ldots, K\}$, $f^{-1}(\vec{w}_k) = \vec{w}_k$
- Iteratively assign datapoints to prototypes and prototypes to data means to minimize reconstruction loss

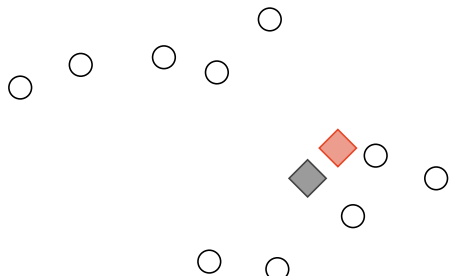
K-means clustering (Hartigan and Wong 1979)

- Idea: Represent data points by a **prototype**: $f(x) = \vec{w}_k \in \mathbb{R}^n$ with $k \in \{1, \ldots, K\}$, $f^{-1}(\vec{w}_k) = \vec{w}_k$
- Iteratively assign datapoints to prototypes and prototypes to data means to minimize reconstruction loss



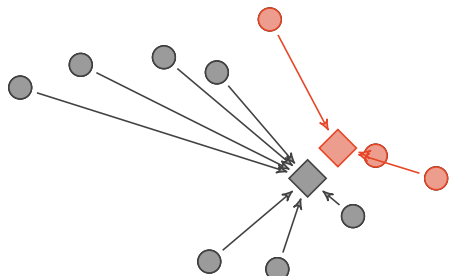
K-means clustering (Hartigan and Wong 1979)

- Idea: Represent data points by a **prototype**: $f(x) = \vec{w}_k \in \mathbb{R}^n$ with $k \in \{1, \ldots, K\}$, $f^{-1}(\vec{w}_k) = \vec{w}_k$
- Iteratively assign datapoints to prototypes and prototypes to data means to minimize reconstruction loss



K-means clustering (Hartigan and Wong 1979)

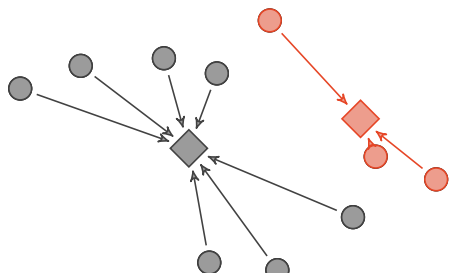
- Idea: Represent data points by a **prototype**: $f(x) = \vec{w}_k \in \mathbb{R}^n$ with $k \in \{1, \ldots, K\}$, $f^{-1}(\vec{w}_k) = \vec{w}_k$
- Iteratively assign datapoints to prototypes and prototypes to data means to minimize reconstruction loss



assignment step

K-means clustering (Hartigan and Wong 1979)

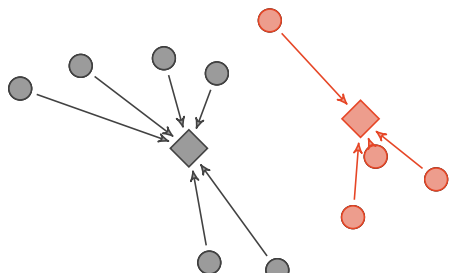
- Idea: Represent data points by a **prototype**: $f(x) = \vec{w}_k \in \mathbb{R}^n$ with $k \in \{1, \ldots, K\}$, $f^{-1}(\vec{w}_k) = \vec{w}_k$
- Iteratively assign datapoints to prototypes and prototypes to data means to minimize reconstruction loss



mean step

K-means clustering (Hartigan and Wong 1979)

- Idea: Represent data points by a **prototype**: $f(x) = \vec{w}_k \in \mathbb{R}^n$ with $k \in \{1, \ldots, K\}$, $f^{-1}(\vec{w}_k) = \vec{w}_k$
- Iteratively assign datapoints to prototypes and prototypes to data means to minimize reconstruction loss

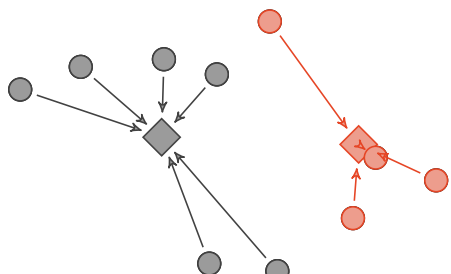


assignment step

K-means clustering

(Hartigan and Wong 1979)

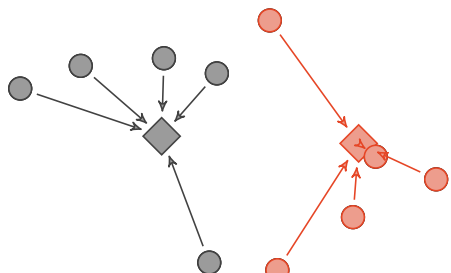
- Idea: Represent data points by a **prototype**: $f(x) = \vec{w}_k \in \mathbb{R}^n$ with $k \in \{1, \ldots, K\}$, $f^{-1}(\vec{w}_k) = \vec{w}_k$
- Iteratively assign datapoints to prototypes and prototypes to data means to minimize reconstruction loss



mean step

K-means clustering (Hartigan and Wong 1979)

- Idea: Represent data points by a **prototype**: $f(x) = \vec{w}_k \in \mathbb{R}^n$ with $k \in \{1, \ldots, K\}$, $f^{-1}(\vec{w}_k) = \vec{w}_k$
- Iteratively assign datapoints to prototypes and prototypes to data means to minimize reconstruction loss

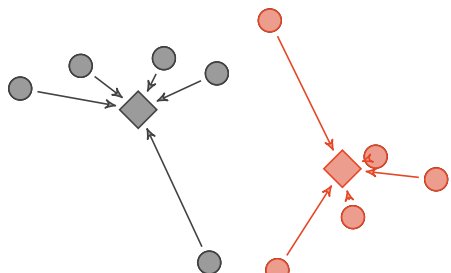


assignment step

K-means clustering

(Hartigan and Wong 1979)

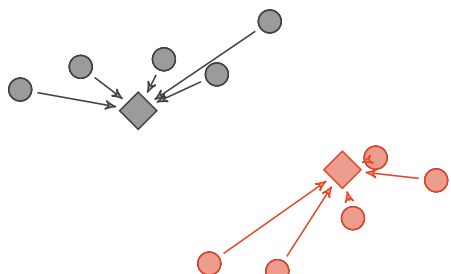
- Idea: Represent data points by a **prototype**: $f(x) = \vec{w}_k \in \mathbb{R}^n$ with $k \in \{1, \ldots, K\}$, $f^{-1}(\vec{w}_k) = \vec{w}_k$
- Iteratively assign datapoints to prototypes and prototypes to data means to minimize reconstruction loss



mean step

K-means clustering (Hartigan and Wong 1979)

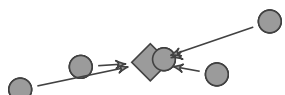
- Idea: Represent data points by a **prototype**: $f(x) = \vec{w}_k \in \mathbb{R}^n$ with $k \in \{1, \ldots, K\}$, $f^{-1}(\vec{w}_k) = \vec{w}_k$
- Iteratively assign datapoints to prototypes and prototypes to data means to minimize reconstruction loss



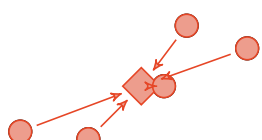
assignment step

K-means clustering (Hartigan and Wong 1979)

- Idea: Represent data points by a **prototype**: $f(x) = \vec{w}_k \in \mathbb{R}^n$ with $k \in \{1, \ldots, K\}$, $f^{-1}(\vec{w}_k) = \vec{w}_k$
- Iteratively assign datapoints to prototypes and prototypes to data means to minimize reconstruction loss

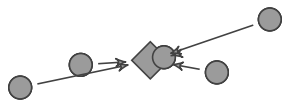


mean step

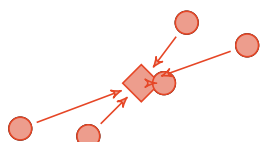


K-means clustering (Hartigan and Wong 1979)

- Idea: Represent data points by a **prototype**: $f(x) = \vec{w}_k \in \mathbb{R}^n$ with $k \in \{1, \ldots, K\}$, $f^{-1}(\vec{w}_k) = \vec{w}_k$
- Iteratively assign datapoints to prototypes and prototypes to data means to minimize reconstruction loss



mean step



- Simple and fast, but sensitive to initialization

Relational Neural Gas (Hammer and Hasenfuss 2010)

- Idea 1: Assign data points "softly" to multiple prototypes

Relational Neural Gas

(Hammer and Hasenfuss 2010)

- Idea 1: Assign data points "softly" to multiple prototypes:
  $\gamma_{k|i} = \exp(-\frac{r_{k|i}}{\lambda})$ for ranks $r_{k|i}$ and decreasing λ

Relational Neural Gas

(Hammer and Hasenfuss 2010)

- Idea 1: Assign data points "softly" to multiple prototypes:
  $\gamma_{k|i} = \exp(-\frac{r_{k|i}}{\lambda}) / \sum_{l=1}^{K} \exp(-\frac{r_{l|i}}{\lambda})$ for ranks $r_{k|i}$ and decreasing λ

Relational Neural Gas (Hammer and Hasenfuss 2010)

- Idea 1: Assign data points "softly" to multiple prototypes: $\gamma_{k|i} = \exp(-\frac{r_{k|i}}{\lambda}) / \sum_{l=1}^{K} \exp(-\frac{r_{l|i}}{\lambda})$ for ranks $r_{k|i}$ and decreasing λ
- Idea 2: prototypes can always be represented as **convex combinations** of data point

Relational Neural Gas

(Hammer and Hasenfuss 2010)

- Idea 1: Assign data points "softly" to multiple prototypes: $\gamma_{k|i} = \exp(-\frac{r_{k|i}}{\lambda}) / \sum_{l=1}^{K} \exp(-\frac{r_{l|i}}{\lambda})$ for ranks $r_{k|i}$ and decreasing λ
- Idea 2: prototypes can always be represented as **convex combinations** of data point: $\vec{w}_k = \sum_{i=1}^{N} \gamma_{k|i} \cdot \vec{x}_i / \sum_{i=1}^{N} \gamma_{k|i}$

Relational Neural Gas (Hammer and Hasenfuss 2010)

- Idea 1: Assign data points "softly" to multiple prototypes: $\gamma_{k|i} = \exp(-\frac{r_{k|i}}{\lambda}) / \sum_{l=1}^{K} \exp(-\frac{r_{l|i}}{\lambda})$ for ranks $r_{k|i}$ and decreasing λ
- Idea 2: prototypes can always be represented as **convex combinations** of data point: $\vec{w}_k = \sum_{i=1}^{N} \gamma_{k|i} \cdot \vec{x}_i / \sum_{i=1}^{N} \gamma_{k|i}$

$\Rightarrow$ Works solely based on pairwise distances

Relational Neural Gas (Hammer and Hasenfuss 2010)

- Idea 1: Assign data points "softly" to multiple prototypes: $\gamma_{k|i} = \exp(-\frac{r_{k|i}}{\lambda}) / \sum_{l=1}^{K} \exp(-\frac{r_{l|i}}{\lambda})$ for ranks $r_{k|i}$ and decreasing λ
- Idea 2: prototypes can always be represented as **convex combinations** of data point: $\vec{w}_k = \sum_{i=1}^{N} \gamma_{k|i} \cdot \vec{x}_i / \sum_{i=1}^{N} \gamma_{k|i}$

$\Rightarrow$ Works solely based on pairwise distances; let $\boldsymbol{D}^2$ be matrix of pairwise squared distances and $\vec{\alpha}_k = (\gamma_{k|1}, \ldots, \gamma_{k|N}) / \sum_{i=1}^{N} \gamma_{k|i}$

Relational Neural Gas

(Hammer and Hasenfuss 2010)

- Idea 1: Assign data points "softly" to multiple prototypes: $\gamma_{k|i} = \exp(-\frac{r_{k|i}}{\lambda}) / \sum_{l=1}^{K} \exp(-\frac{r_{l|i}}{\lambda})$ for ranks $r_{k|i}$ and decreasing λ
- Idea 2: prototypes can always be represented as **convex combinations** of data point: $\vec{w}_k = \sum_{i=1}^{N} \gamma_{k|i} \cdot \vec{x}_i / \sum_{i=1}^{N} \gamma_{k|i}$

$\Rightarrow$ Works solely based on pairwise distances; let $\boldsymbol{D}^2$ be matrix of pairwise squared distances and $\vec{\alpha}_k = (\gamma_{k|1}, \ldots, \gamma_{k|N}) / \sum_{i=1}^{N} \gamma_{k|i}$; then:

$$d(x, w_k)^2 = \sum_{i=1}^{N} \alpha_{k,i} \cdot d(x, x_i)^2 - \frac{1}{2} \vec{\alpha}_k \cdot \boldsymbol{D}^2 \cdot \vec{\alpha}_k^T$$

Relational Neural Gas (Hammer and Hasenfuss 2010)

- Idea 1: Assign data points "softly" to multiple prototypes: $\gamma_{k|i} = \exp(-\frac{r_{k|i}}{\lambda}) / \sum_{l=1}^{K} \exp(-\frac{r_{l|i}}{\lambda})$ for ranks $r_{k|i}$ and decreasing λ
- Idea 2: prototypes can always be represented as **convex combinations** of data point: $\vec{w}_k = \sum_{i=1}^{N} \gamma_{k|i} \cdot \vec{x}_i / \sum_{i=1}^{N} \gamma_{k|i}$

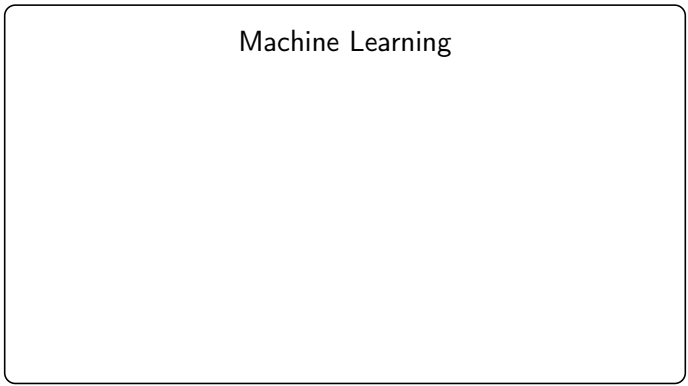
$\Rightarrow$ Works solely based on pairwise distances; let $\boldsymbol{D}^2$ be matrix of pairwise squared distances and $\vec{\alpha}_k = (\gamma_{k|1}, \ldots, \gamma_{k|N}) / \sum_{i=1}^{N} \gamma_{k|i}$; then:

$$d(x, w_k)^2 = \sum_{i=1}^{N} \alpha_{k,i} \cdot d(x, x_i)^2 - \frac{1}{2} \vec{\alpha}_k \cdot \boldsymbol{D}^2 \cdot \vec{\alpha}_k^T$$

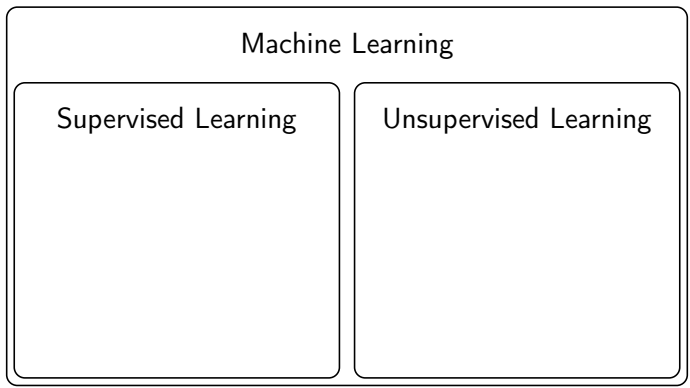
- Reference implementation: `https://gitlab.ub.uni-bielefeld.de/bpaassen/proto-dist-ml`

Summary

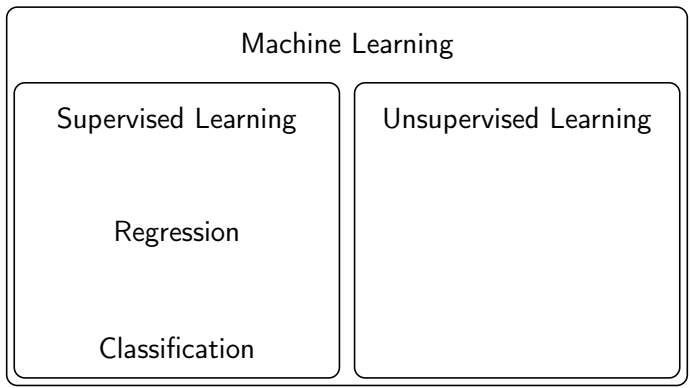
Classic Tasks in Machine Learning



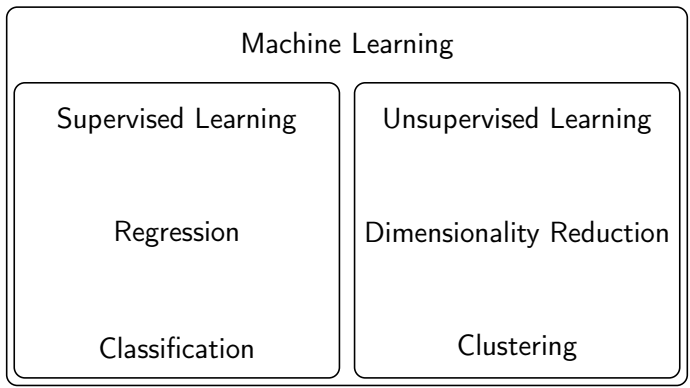
Classic Tasks in Machine Learning



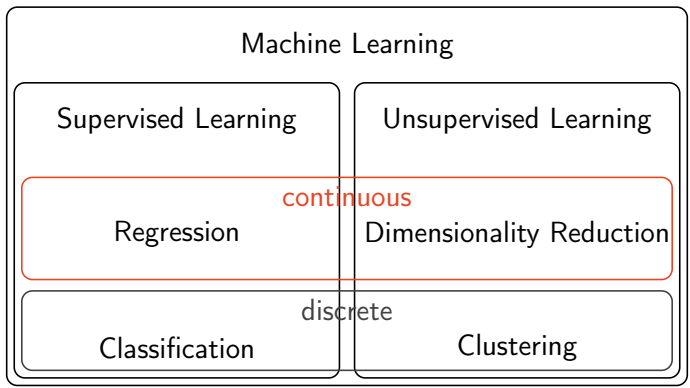
Classic Tasks in Machine Learning



Classic Tasks in Machine Learning

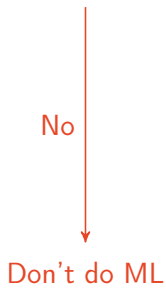


Classic Tasks in Machine Learning



So you want to do Machine Learning?

Do you have data?



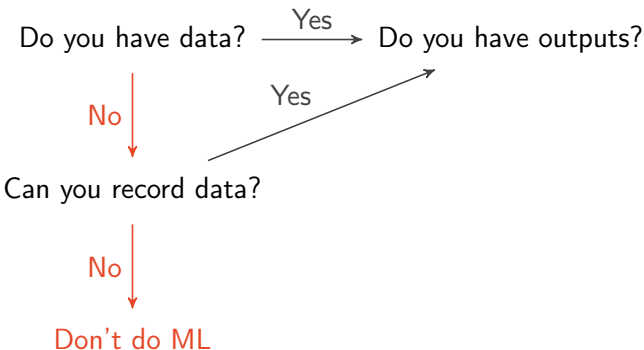
So you want to do Machine Learning?

Do you have data?

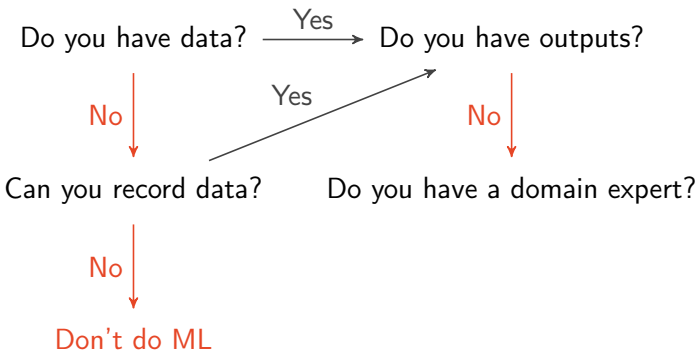
Can you record data?

Don't do ML

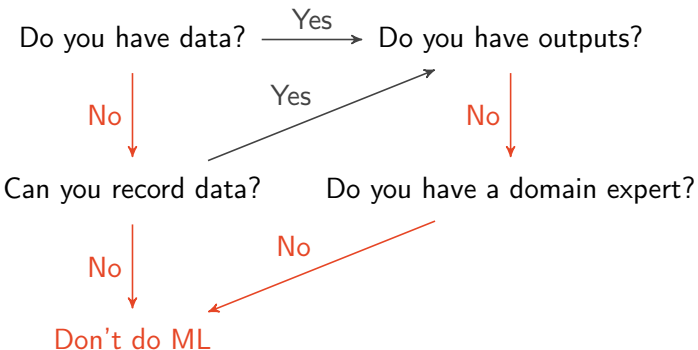
So you want to do Machine Learning?



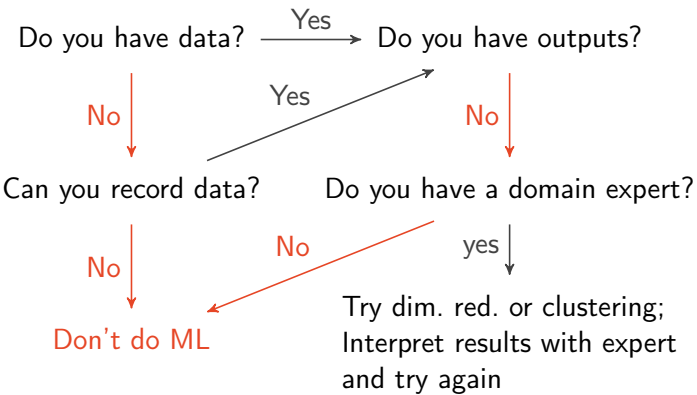
So you want to do Machine Learning?



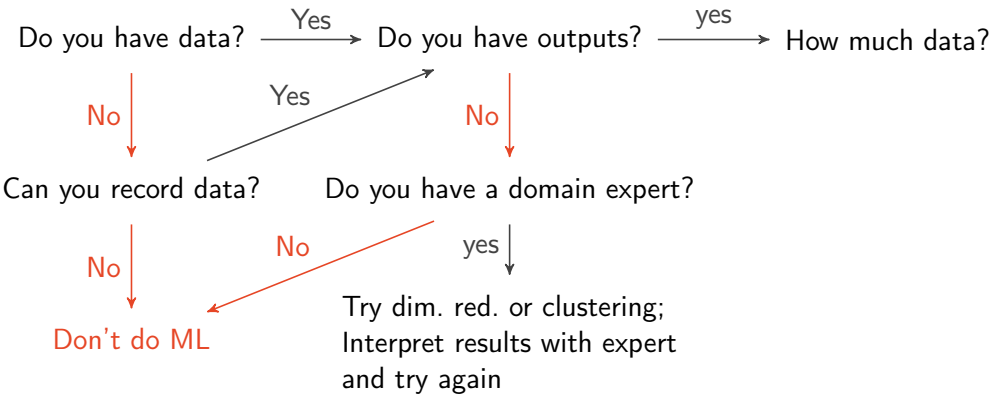
So you want to do Machine Learning?



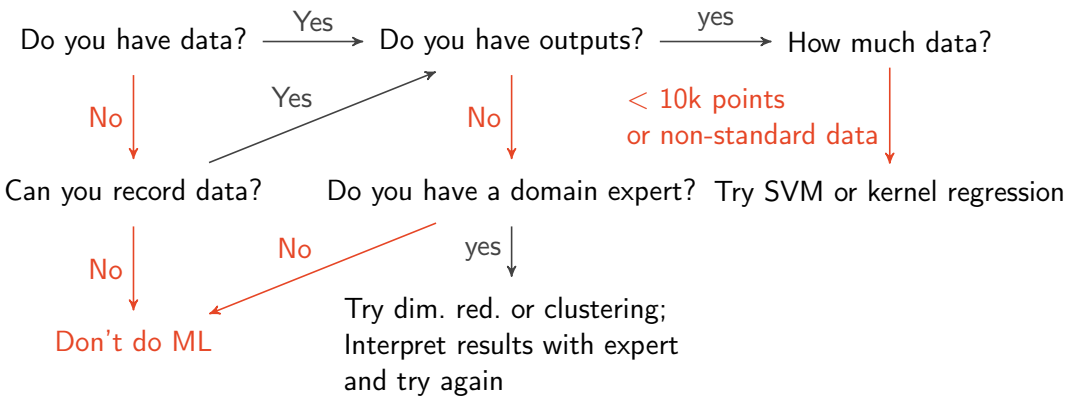
So you want to do Machine Learning?



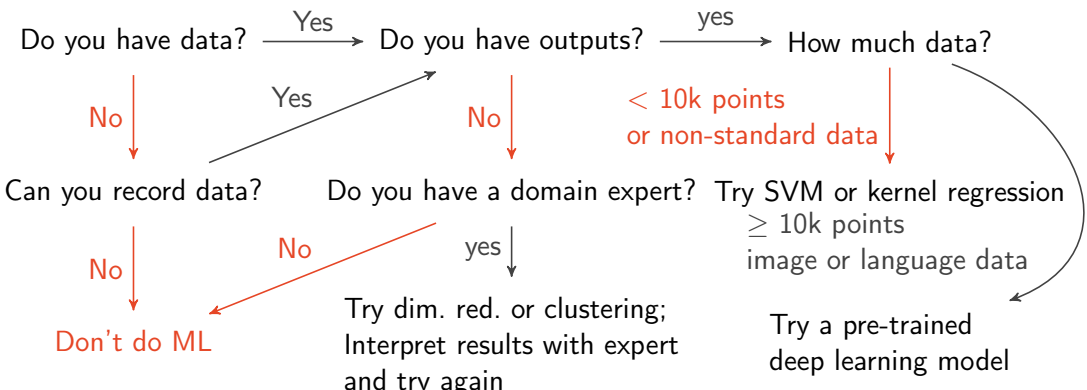
So you want to do Machine Learning?



So you want to do Machine Learning?



So you want to do Machine Learning?



Literature

Literature I

Bishop, Christopher M. (2006). **Pattern Recognition and Machine Learning**. Berlin/Heidelberg, Germany: Springer. ISBN: 0387310738.

Boyd, Stephen and Lieven Vandenberghe (2004). **Convex Optimization**. Cambridge, UK: Cambridge University Press. URL: http://web.stanford.edu/~boyd/cvxbook/.

Cover, Thomas M. and Peter E. Hart (1967). "Nearest neighbor pattern classification". In: **IEEE Transactions on Information Theory** 13.1, pp. 21–27. DOI: 10.1109/TIT.1967.1053964.

Gisbrecht, Andrej, Alexander Schulz, and Barbara Hammer (2015). "Parametric nonlinear dimensionality reduction using kernel t-SNE". In: **Neurocomputing** 147, pp. 71–82. DOI: 10.1016/j.neucom.2013.11.045.

Hammer, Barbara and Alexander Hasenfuss (2010). "Topographic Mapping of Large Dissimilarity Data Sets". In: **Neural Computation** 22.9, pp. 2229–2284.

Literature II

Hartigan, J. A. and M. A. Wong (1979). "Algorithm AS 136: A K-Means Clustering Algorithm". In: **Journal of the Royal Statistical Society. Series C (Applied Statistics)** 28.1, pp. 100–108. DOI: 10.2307/2346830.

Mokbel, Bassam et al. (2013). "Visualizing the quality of dimensionality reduction". In: **Neurocomputing** 112, pp. 109–123. DOI: 10.1016/j.neucom.2012.11.046.

Paaßen, Benjamin (2019). **Lecture Notes on Applied Optimization**. Bielefeld University. URL: https://pub.uni-bielefeld.de/record/2935200.

Pearson, Karl (1901). "LIII. On lines and planes of closest fit to systems of points in space". In: **The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science** 2.11, pp. 559–572. DOI: 10.1080/14786440109462720.

Rasmussen, Carl Edward and Christopher K. I. Williams (2005). **Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)**. Cambridge, MA, USA: The MIT Press.

Literature III

Sibson, R. (1973). "SLINK: An optimally efficient algorithm for the single-link cluster method". In: **The Computer Journal** 16.1, pp. 30–34. DOI: 10.1093/comjnl/16.1.30.

Suykens, J.A.K. and J. Vandewalle (1999). "Least Squares Support Vector Machine Classifiers". In: **Neural Processing Letters** 9.3, pp. 293–300. DOI: 10.1023/A:1018628609742.

Van der Maaten, Laurens and Geoffrey Hinton (2008). "Visualizing Data using t-SNE". In: **Journal of Machine Learning Research** 9, pp. 2579–2605. URL: http://www.jmlr.org/papers/v9/vandermaaten08a.html.

Ward, Joe (1963). "Hierarchical Grouping to Optimize an Objective Function". In: **Journal of the American Statistical Association** 58.301, pp. 236–244. DOI: 10.1080/01621459.1963.10500845.

Literature IV

Weinberger, Kilian Q. and Lawrence K. Saul (2009). "Distance Metric Learning for Large Margin Nearest Neighbor Classification". In: **Journal of Machine Learning Research** 10, pp. 207–244. URL: http://www.jmlr.org/papers/v10/weinberger09a.html.