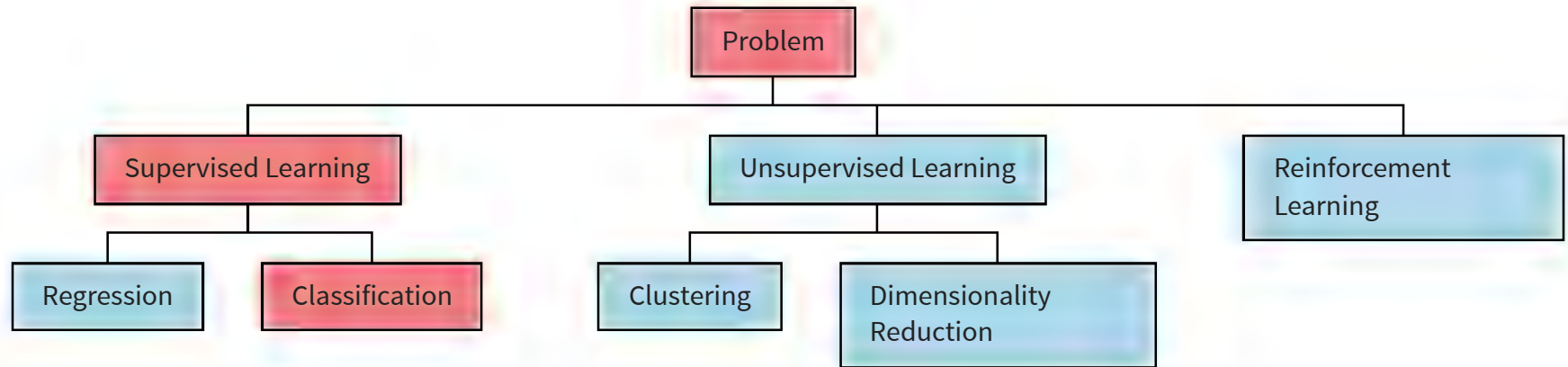


# TAG 2

Klassifikation, SVM, Decision Tree

# CLASSIFICATION



# CLASSIFICATION

Der Output besteht aus einer Menge von **Klassen**.

- Identifikation von Spam-E-mails
- Objekterkennung in Bildern

Guten Tag Herr  
Meyer...

Modell

Not Spam

GLÜCKWUNSCH HERR  
INFO...

Modell

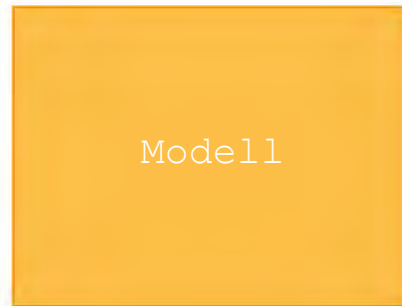
Spam

# CLASSIFICATION - BEISPIEL - NAME DER BLUMENART

- Input(s): length (cm), width (cm)
- Output: "Setosa" oder "Other"

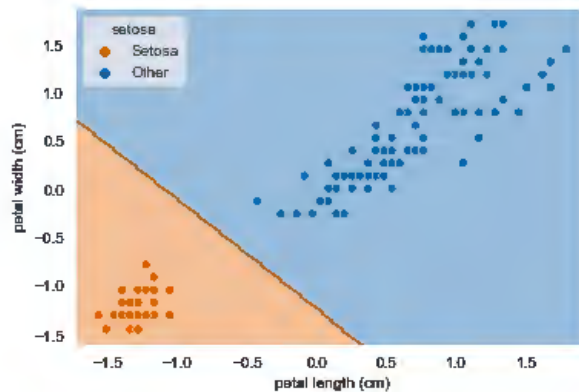
INPUT SPACE  
2 FEATURES

length: 1.5cm  
width: 0.25cm

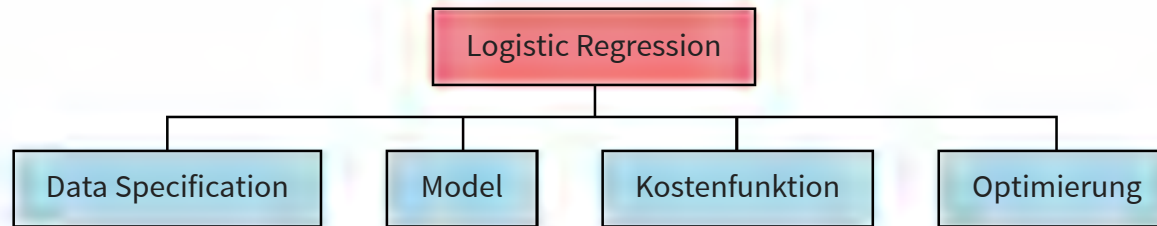


OUTPUT SPACE

Setosa

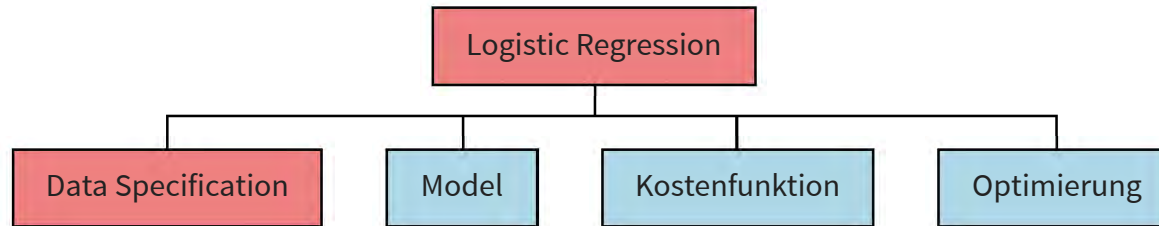


# LOGISTIC REGRESSION



Logistic Regression macht **keine Regression**, sondern eine **Klassifikation**!

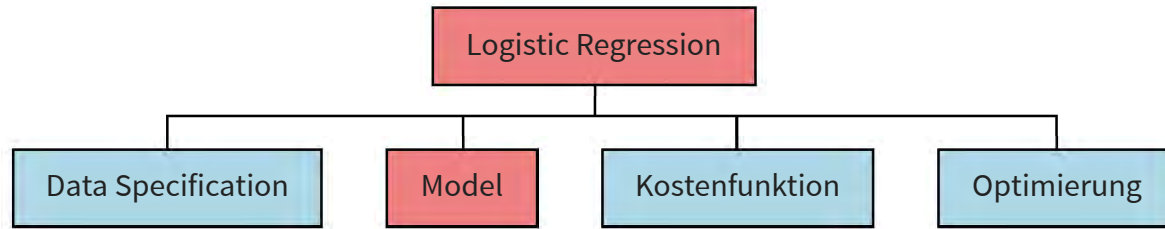
# LOGISTIC REGRESSION



# LOGISTIC REGRESSION - DATA SPECIFICATION

1. Was ist die **kategorische Ziel-Variable**, z.B. `name`
2. Welche **Features** wählen wir, z.B. um eine Blume zu repräsentieren (`petal-length (cm)`, `petal-width (cm)`, ...)
3. Kategorische Features müssen **encoded** werden.
4. Wenn **Regularisiert** (default): Numerische Features müssen **standardisiert** werden.

# LOGISTIC REGRESSION





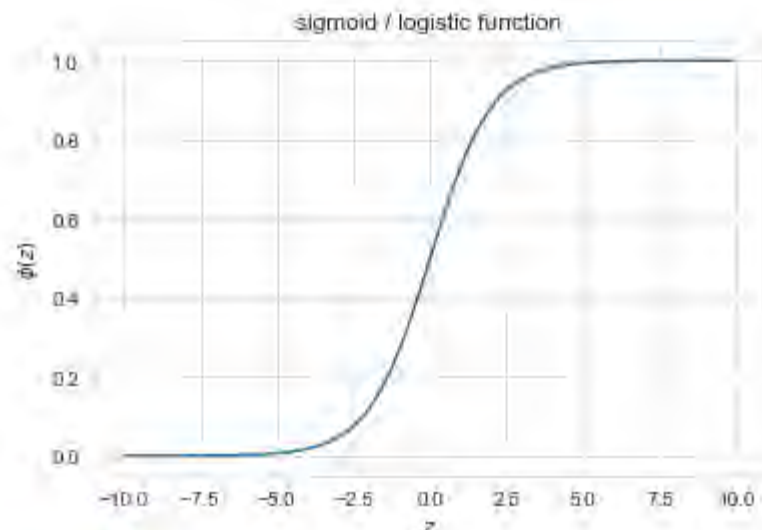
# LOGISTIC REGRESSION - INTUITION

1. **Modifikation**: lineares Modell ändern, dass Output zwischen 0 und 1 liegt.
2. **Interpretation**: Output als **Wahrscheinlichkeit** für eine von zwei Klassen interpretieren.

# LOGISTIC REGRESSION - INTUITION

1. **Modifikation**: lineares Modell ändern, dass Output zwischen 0 und 1 liegt.
2. **Interpretation**: Output als Wahrscheinlichkeit für eine von zwei Klassen interpretieren.

$$\phi(z) = \frac{1}{1 + e^{-z}}$$



$$\phi(\beta_0 + x_1 \beta_1)$$

# LOGISTIC REGRESSION - INTUITION

1. **Modifikation**: lineares Modell ändern, dass Output zwischen 0 und 1 liegt.
2. **Interpretation**: Output als **Wahrscheinlichkeit** für eine von zwei Klassen interpretieren.

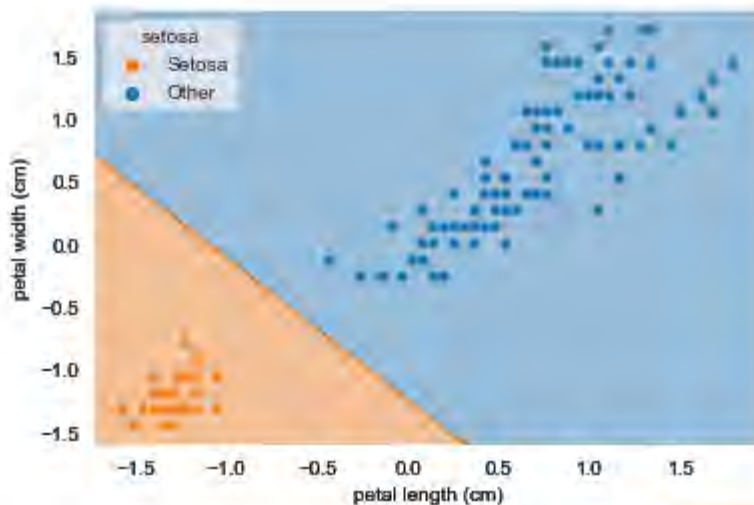
$$p(y = 1 | x_1) = \phi(\beta_0 + x_1 \beta_1)$$

$$p(y = 0 | x_1) = 1 - \phi(\beta_0 + x_1 \beta_1)$$

$$\hat{y} = \arg \max_k p(y = k | x_1)$$

# LOGISTISCHE REGRESSION - INTUITION

Beispiel: Ob **Setosa** Blumenart anhand 2 Features  
(petal-length, petal-width).



petal-length:  
1.5cm  
petal-width:  
0.25cm

$$\phi(\beta_0 + x_1 \beta_1 + x_2 \beta_2)$$

Setosa: 99%

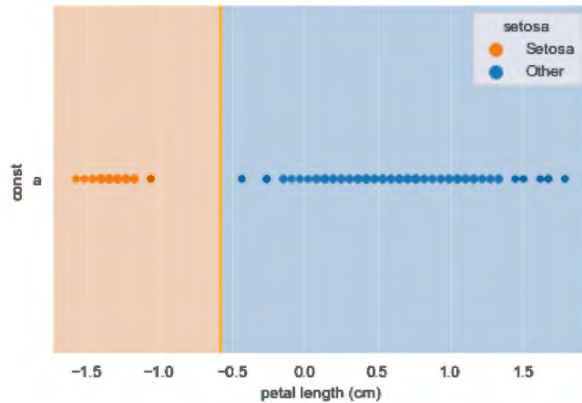
Dazu verwenden wir das  
folgende **Modell**

$$\phi(\beta_0 + x_1 \beta_1 + x_2 \beta_2)$$

# LOGISTIC REGRESSION - MEHRERE FEATURES

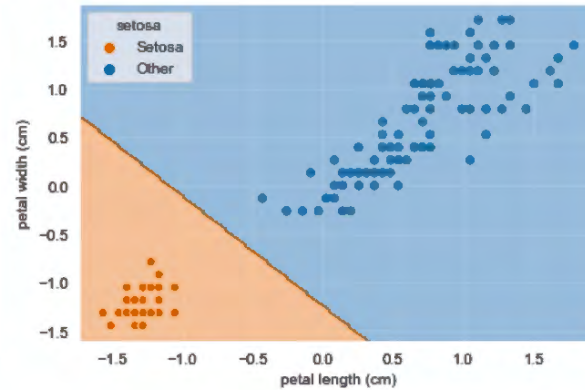
## 1 FEATURE

$$\phi(\beta_0 + x_1 \beta_1)$$



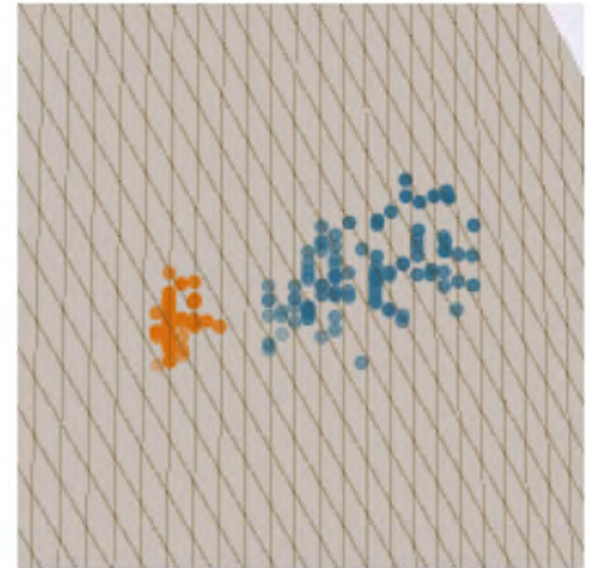
## 2 FEATURE

$$\phi(\beta_0 + x_1 \beta_1 + x_2 \beta_2)$$

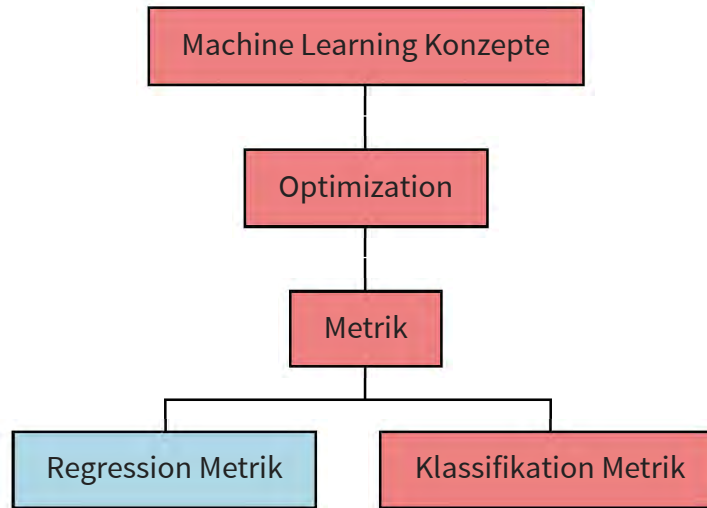


## 3 FEATURE

$$\phi(\beta_0 + x_1 \beta_1 + x_2 \beta_2 + x_3 \beta_3)$$



# KLASSIFIKATION METRIK



# CLASSIFICATION - FEHLER EINES MODELLS MESSEN?

- Was für Fälle sind möglich pro Datenpunkt? Bei z.B. Spam / Nicht Spam?

Modell: Spam

Wahrheit: Spam

True Positive

Modell: Not Spam

Wahrheit: Not Spam

True Negative

Modell: Spam

Wahrheit: Not Spam

False Positive

Modell: Not Spam

Wahrheit: Spam

False Negative

Unterscheidung wichtig, Beispiel Corona-Test:

False Positive => unnötige Quarantäne

False Negative => Virus verbreitet sich

# CLASSIFICATION - FEHLER EINES MODELLS MESSEN?

- Was für Fälle sind möglich pro Datenpunkt? Bei Setosa / Versicolor / Virginica

Modell: Setosa

Modell: Versicolor

Modell: Virginica

Modell: Setosa

Modell: Setosa

Modell: Virginica

Modell: Virginica

Modell: Versicolor

Modell: Versicolor

Wahrheit: Setosa

Wahrheit: Versicolor

Wahrheit: Virginica

Wahrheit: Versicolor

Wahrheit: Virginica

Wahrheit: Setosa

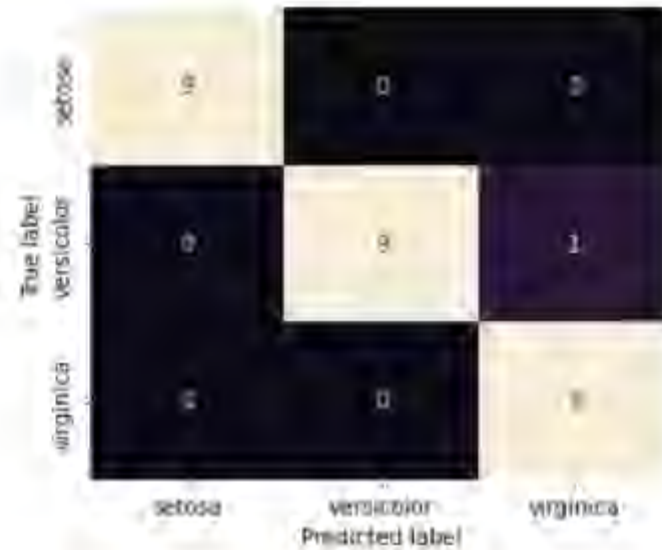
Wahrheit: Versicolor

Wahrheit: Setosa

Wahrheit: Virginica



# CLASSIFICATION - CONFUSION MATRIX



Confusion Matrix zeigt die Fehler pro Datenpunkt!

# CLASSIFICATION - METRIK - BEISPIELE

Accuracy	Binär:	$\frac{TP + TN}{TP + TN + FP + FN}$
	Generell:	$\frac{\text{Korrekt}}{\text{Korrekt} + \text{Inkorrekt}}$

Accuracy wird oft verwendet.

Accuracy ist irreführend bei imbalanced Klassen

# CLASSIFICATION - METRIK - BEISPIELE

Precision

Binär:

$$\frac{TP}{TP + FP}$$

Recall

Binär:

$$\frac{TP}{TP + FN}$$

F1-Score

Binär:

$$F_1 = 2 \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

F1 ist geeignet bei imbalanced Klassen, wenn Anzahl negative Beispiele >> Anzahl positive Beispiele

# CLASSIFICATION - METRIK - BEISPIELE

	Predicted	
	Spam	Not Spam
Spam	5	5
Not Spam	5	5

$$\text{Accuracy} = \frac{5 + 5}{5 + 5 + 5 + 5} = \frac{10}{20} = 50\%$$

$$\text{Precision} = \frac{5}{5 + 5} = \frac{5}{10} = 50\%$$

$$\text{Recall} = \frac{5}{5 + 5} = \frac{5}{10} = 50\%$$

$$\text{F1-Score} = 2 * \frac{50 * 50}{50 + 50} = 50\%$$

# CLASSIFICATION - METRIK - BEISPIEL FÜR F1-SCORE

	Predicted	
	Spam	Not Spam
Spam	0	1
Not Spam	0	99

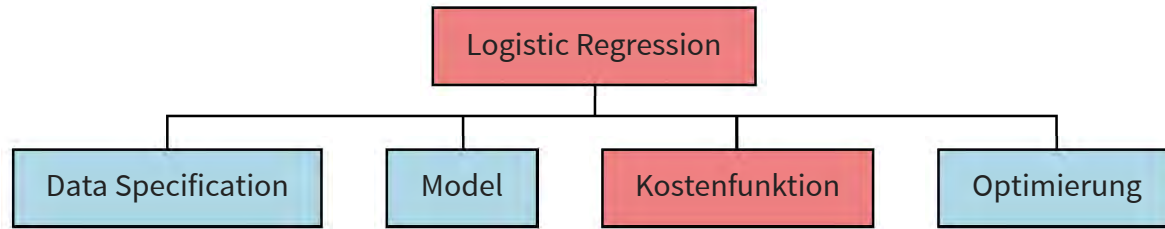
$$\text{Accuracy} = \frac{0 + 99}{0 + 99 + 0 + 1} = \frac{99}{100} = 99\%$$

$$\text{Precision} = \frac{0}{0 + 0} = \frac{0}{0} = 0\%$$

$$\text{Recall} = \frac{0}{0 + 1} = \frac{0}{1} = 0\%$$

$$\text{F1-Score} = 2 * \frac{0 * 0}{0 + 0} = 0\%$$

# LOGISTIC REGRESSION



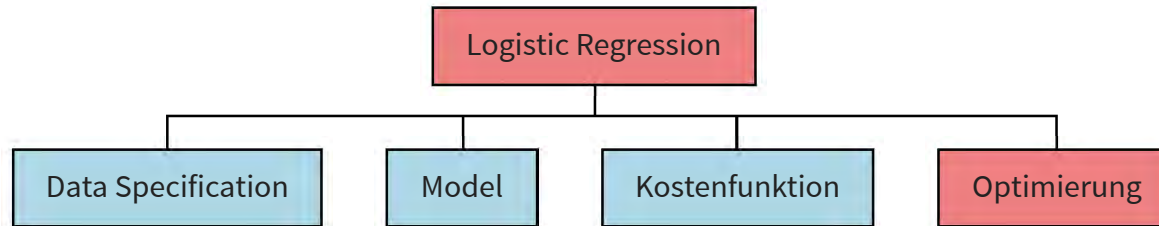
# KOSTENFUNKTION - MAXIMUM LIKELIHOOD

Bei der **Logistic Regression** gibt das Modell eine **Wahrscheinlichkeit** für eine Klasse (Likelihood).  
Mit dieser können wir eine **spezielle Kostenfunktion** erstellen, die **Maximum Likelihood**:

$$\phi(\mathbf{x}\boldsymbol{\beta}) = \phi(\beta_0 + x_1 \beta_1 + \dots + x_p \beta_p)$$

$$p(\vec{\mathbf{y}}|\mathbf{X}) = \prod_{\substack{\mathbf{x}^{(i)} \\ \text{aller positive Samples}}} \phi(\mathbf{x}^{(i)} \boldsymbol{\beta}) \prod_{\substack{\mathbf{x}^{(i)} \\ \text{aller negativen Samples}}} 1 - \phi(\mathbf{x}^{(i)} \boldsymbol{\beta})$$

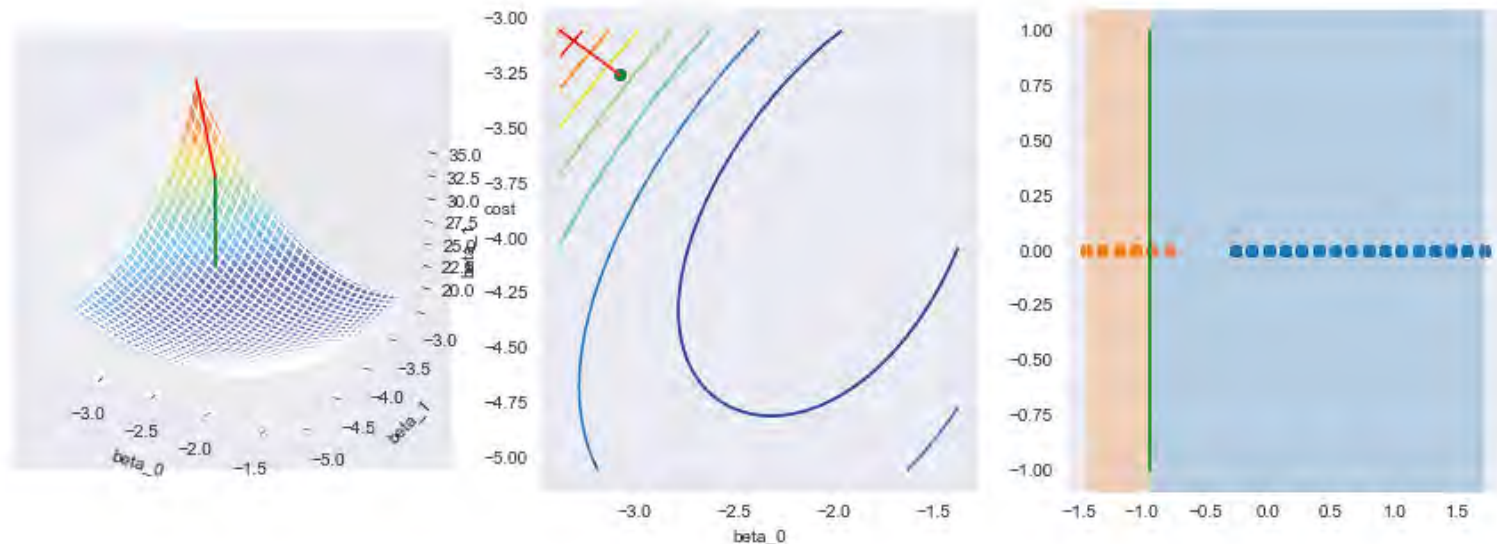
# LOGISTIC REGRESSION





# LOGISTIC REGRESSION OPTIMIERUNG - GRADIENT DESCENT

Idee: Gradient Descent anwenden auf der Maximum Likelihood Kostenfunktion



Andere Verfahren (z.B. Coordinate Descent) möglich

# LOGISTIC REGRESSION OPTIMIERUNG

In der Praxis muss man noch folgendes machen:

1. **Konvention** Minimize statt Maximize: Negieren
2. **Numerische Stabilität**: Logarithmus nehmen

$$p(\vec{y}|X) = \prod_{\substack{x^{(i)} \\ \text{aller positive Samples}}} \phi(x^{(i)} \beta) \prod_{\substack{x^{(i)} \\ \text{aller negativen Samples}}} 1 - \phi(x^{(i)} \beta)$$

# LOGISTIC REGRESSION OPTIMIERUNG

In der Praxis muss man noch folgendes machen:

1. **Konvention** Minimize statt Maximize: Negieren
2. Numerische Stabilität: Logarithmus nehmen

$$p(\vec{y}|X) = \prod_{\mathbf{x}^{(i)} \text{ aller positive Samples}} \phi(\mathbf{x}^{(i)} \beta) \prod_{\mathbf{x}^{(i)} \text{ aller negativen Samples}} 1 - \phi(\mathbf{x}^{(i)} \beta)$$

$$-p(\vec{y}|X; \beta) = - \left( \prod_{\mathbf{x}^{(i)} \text{ aller positive Samples}} \phi(\mathbf{x}^{(i)} \beta) \prod_{\mathbf{x}^{(i)} \text{ aller negativen Samples}} 1 - \phi(\mathbf{x}^{(i)} \beta) \right)$$

Nur Konvention!

# LOGISTIC REGRESSION OPTIMIERUNG

In der Praxis muss man noch folgendes machen:

1. **Konvention** Minimize statt Maximize: Negieren

2. **Numerische Stabilität**: Logarithmus nehmen

$$-p(\vec{y}|X; \beta) = - \left( \prod_{\mathbf{x}^{(i)} \text{ aller positive Samples}} \phi(\mathbf{x}^{(i)} \beta) \prod_{\mathbf{x}^{(i)} \text{ aller negativen Samples}} 1 - \phi(\mathbf{x}^{(i)} \beta) \right)$$

$$-\log(p(\vec{y}|X; \beta)) = - \left( \sum_{\mathbf{x}^{(i)} \text{ aller positive Samples}} \log(\phi(\mathbf{x}^{(i)} \beta)) + \sum_{\mathbf{x}^{(i)} \text{ aller negativen Samples}} \log(1 - \phi(\mathbf{x}^{(i)} \beta)) \right)$$

"Nur" notwendig wegen **Floating Point Precision!**

# LOGISTIC REGRESSION - MEHRERE KLASSEN - SOFTMAX

- Sigmoid ( $\phi$ ) gibt uns für den Beweis einer Klasse, die Wahrscheinlichkeit für diese Klasse.
- Softmax gibt uns für Beweise für mehrere Klassen, die Wahrscheinlichkeiten der Klassen.

$$\sigma(z_i) = \frac{e^{z_i}}{\sum_{l=0}^{K-1} e^{z_l}}$$

$z_1 =$	-2
$z_2 =$	5
$z_3 =$	1

$$\sigma(z_1) = 0.09\%$$

$$\sigma(z_2) = 98.11\%$$

$$\sigma(z_3) = 1.8\%$$

## LOGISTIC REGRESSION - 3 KLASSEN

- 3 eigene Lineare Modelle (3 Outputs = 3 Beweise)
- 3 Outputs zu 3 Wahrscheinlichkeiten (Softmax)
- Weiterhin Maximum Likelihood als Kostenfunktion (Cross-Entropy)

3 Linearen Modelle werden zusammen trainiert

# LOGISTIC REGRESSION - CODE

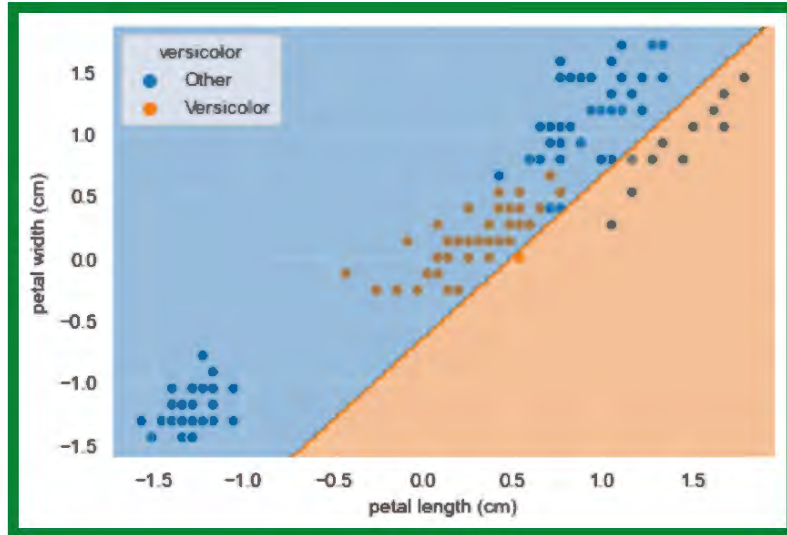
```
logistic_regression.ipynb
```



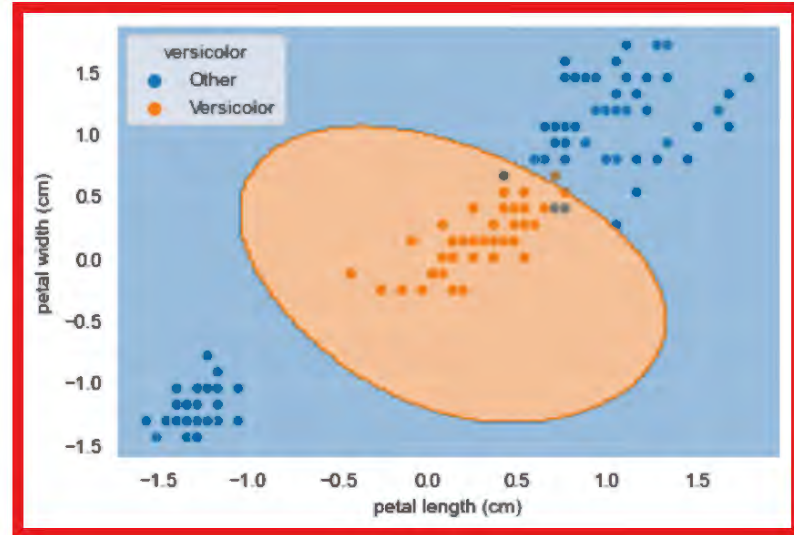
# LOGISTIC REGRESSION - LIMITS

2 FEATURES

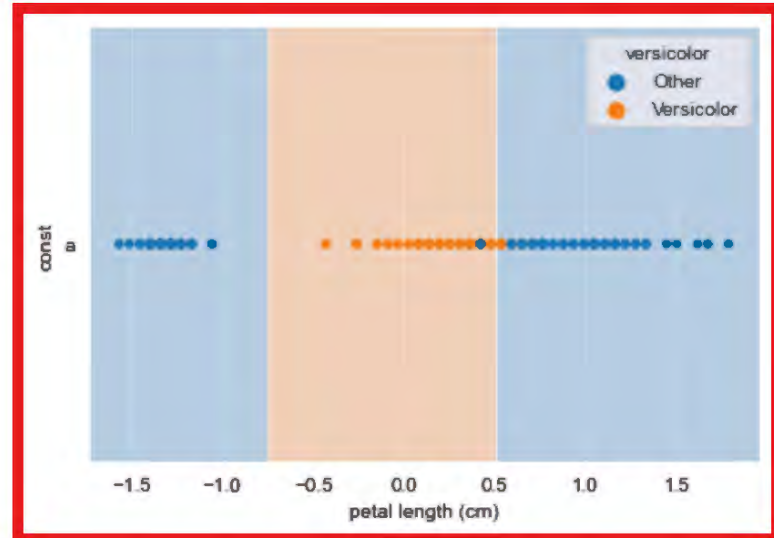
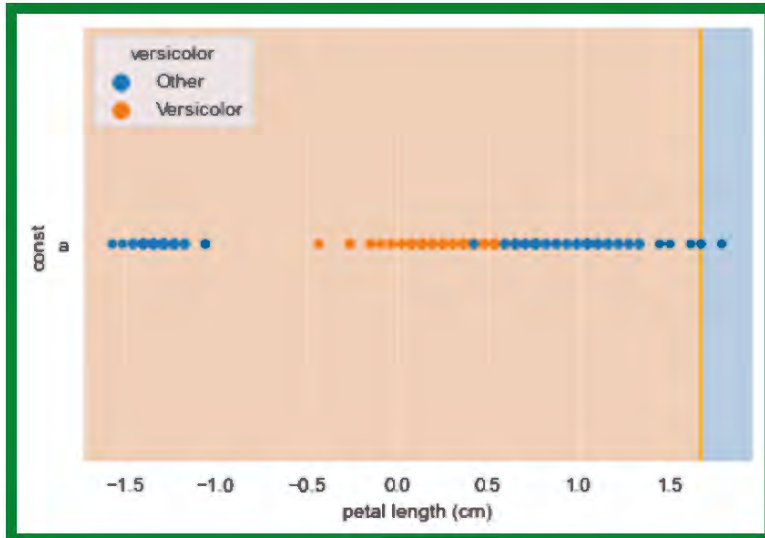
POSSIBLE



IMPOSSIBLE (IN FEATURE SPACE)



1 FEATURE





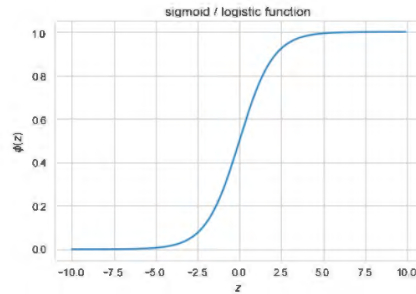
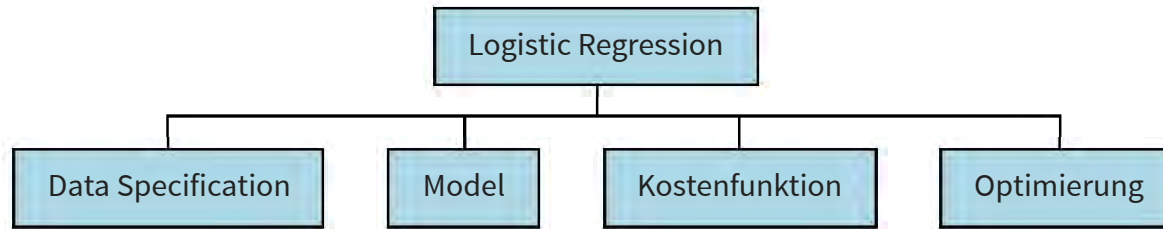
# ALTERNATIVE & ADDITIONAL RESOURCES

- [Alternative] Logistic Regression
    - [sklearn User Guide](#)
- 

[Additional]

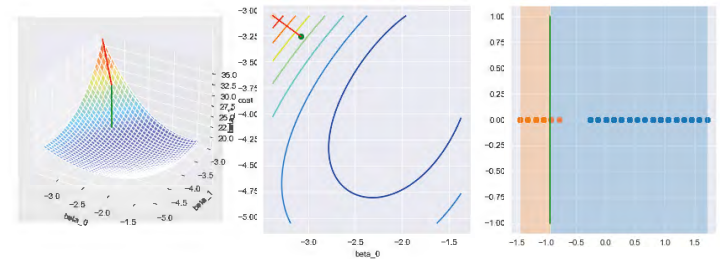
- [Video: Softmax Function Explained In Depth with 3D Visuals](#)

# LOGISTIC REGRESSION



$$\phi(\beta_0 + x_1 \beta_1 + \dots + x_p \beta_p)$$

$$p(\vec{y}|X) = \prod_{\substack{x^{(i)} \\ \text{aller positive Samples}}} \phi(x^{(i)} \beta) \prod_{\substack{x^{(i)} \\ \text{aller negativen Samples}}} 1 - \phi(x^{(i)} \beta)$$



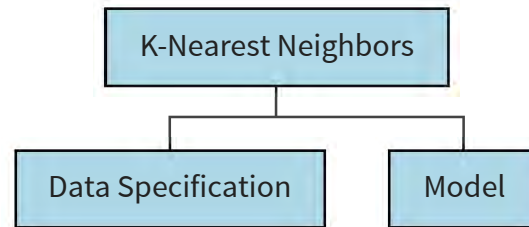
# QUESTIONS



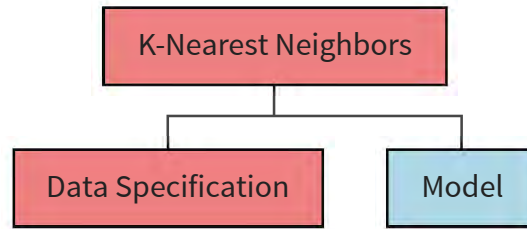
# QUESTIONS

1. Was ist Ziel einer Klassifikation?
2. Was für Annahmen trifft die Logistic Regression?
3. Was ist der Zusammenhang mit der Linearen Regression

# K-NEAREST NEIGHBORS



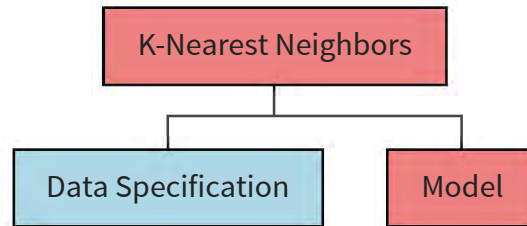
# K-NEAREST NEIGHBORS



# K-NEAREST-NEIGHBORS - DATA SPECIFICATION

- Was ist die **Ziel-Variable**, z.B. `name`
- Welche **Features** wählen wir, z.B. um eine Blume zu repräsentieren (`petal-length (cm)`, `petal-width (cm)`, ...)
- Kategorische Features müssen encoded werden.
- Numerische Features müssen standardisiert werden.

# K-NEAREST NEIGHBORS



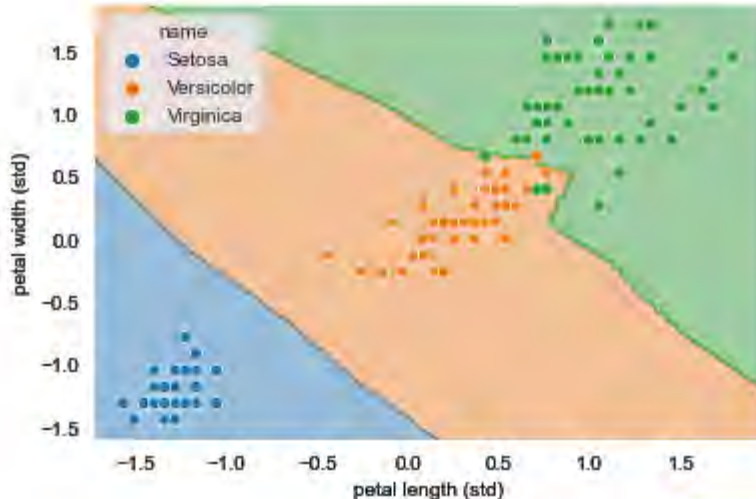


# K-NEAREST-NEIGHBORS - INTUITION

1. Wir lernen **keine Parameter**  $\beta$  ( ).
2. Wir suchen die nächsten **k** Punkte im Train Set für Vorhersage.

# K-NEAREST-NEIGHBORS - INTUITION

Beispiel: **Blumenart** anhand 2 Features (petal-length, petal-width).



Als **Modell** suchen wir die nächsten  $k$  Punkte im Train Set.

petal-length  
(std): -1.25  
petal-width  
(std): -1.00

Lookup &  
average  
neighbors

**Setosa**: 100%

# K-NEAREST-NEIGHBORS - CODE

`k_nearest_neighbor.ipynb`



# K-NEAREST-NEIGHBORS - REGRESSION

- Kann auch für **Regression** eingesetzt werden.
- Einfach Labels (z.B. Kosten) von K-Nearest-Neighbors als Vorhersage **mitteln** (allenfalls noch gewichtet nach Distanz zum Nachbarn).

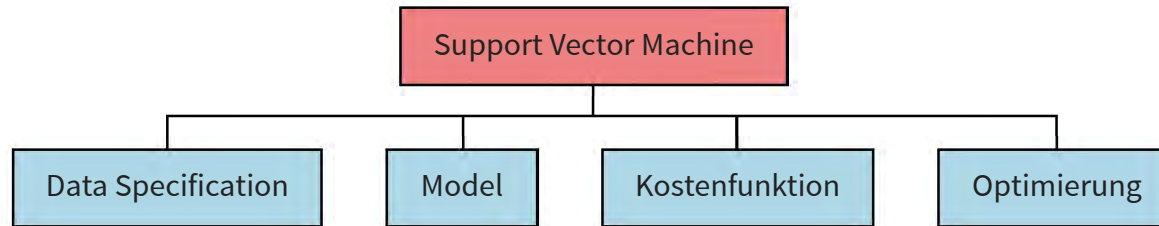
# K-NEAREST-NEIGHBORS - SUMMARY

- **Kein Optimierung-Algorithmus:** "Nur" effizientes Abspeichern des Train-Sets (KD-Tree)
- Bei vielen Features empirisch meistens **nicht gut**.  
Grund: **Curse of Dimensionality**
- Einfacher Algorithmus, wo man **kennen sollte**.

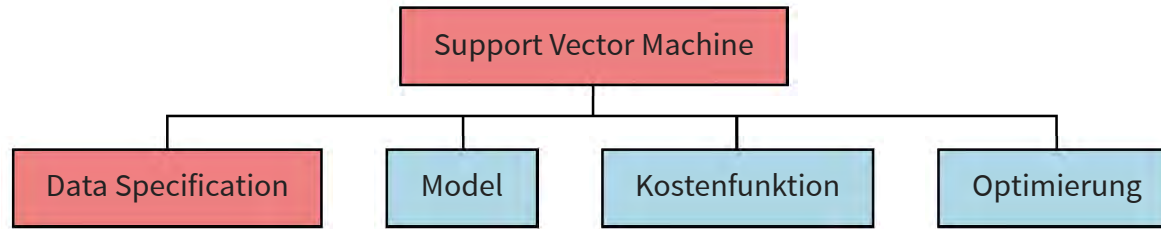
# ALTERNATIVE & ADDITIONAL RESOURCES

- [Alternative]
  - [Clustering | KMeans Algorithm \(Machine Learning | Andrew Ng\)](#)
  - [Wikipedia](#)
- [Additional]
  - [Survey Paper: Survey of Nearest Neighbor Techniques](#)

# SUPPORT VECTOR MACHINE



# SUPPORT VECTOR MACHINE

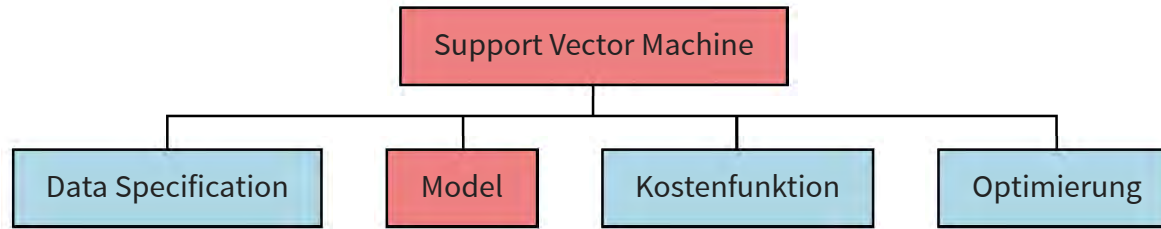




# SUPPORT VECTOR MACHINE - DATA SPECIFICATION

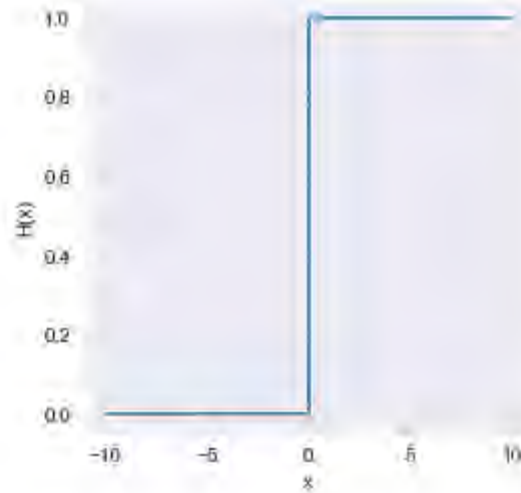
1. Was ist die **kategorische Ziel-Variable**, z.B. `name`
2. Welche **Features** wählen wir, z.B. um eine Blume zu repräsentieren (`petal-length (cm)`, `petal-width (cm)`, ...)
3. Kategorische Features müssen **encoded** werden.
4. Numerische Features müssen **standardisiert** werden.

# SUPPORT VECTOR MACHINE

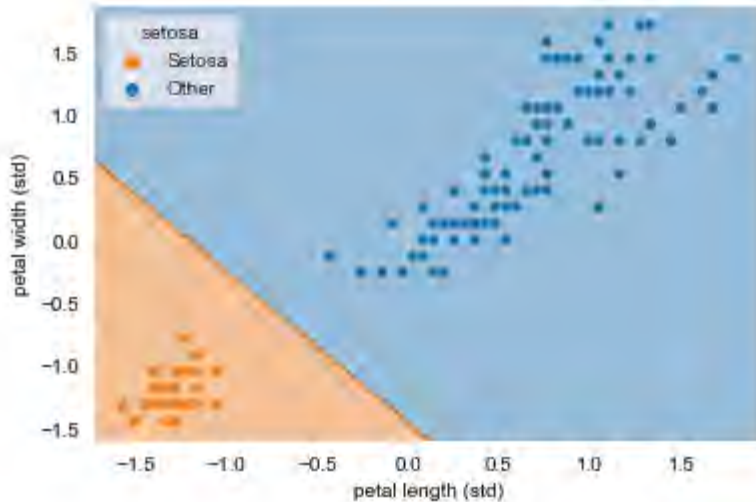


# HEAVISIDE STEP FUNCTION

$$H(x)$$



# SUPPORT VECTOR MACHINE - MODEL



Wir verwenden das  
folgende Modell

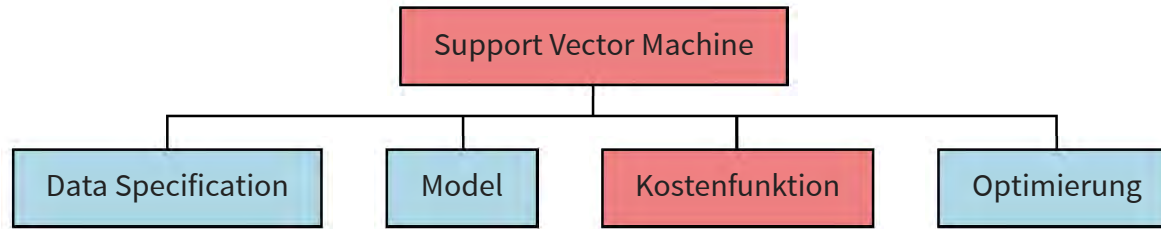
$$H(\beta_0 + x_1 \beta_1 + x_2 \beta_2)$$

petal-length  
(std): -1.25  
petal-width  
(std): -1.00

$$H(\beta_0 + x_1 \beta_1 + x_2 \beta_2)$$

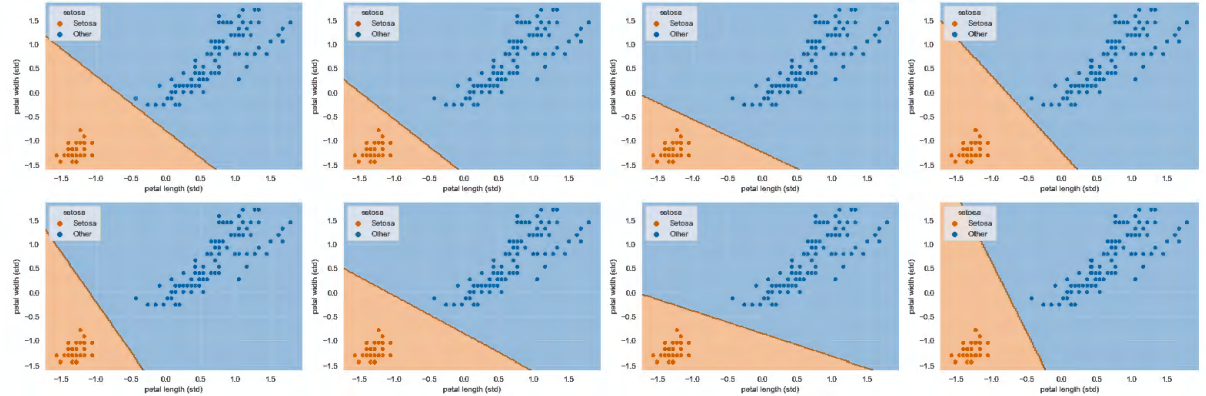
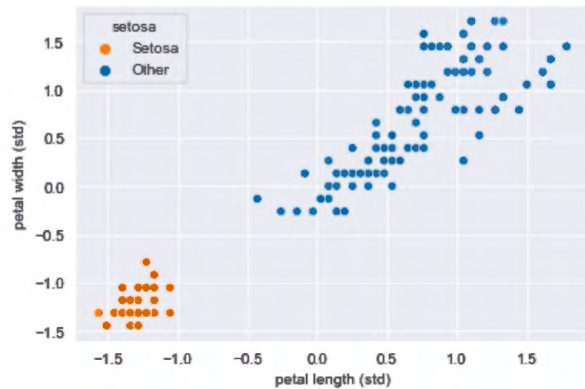
1 (Setosa)

# SUPPORT VECTOR MACHINE



# SUPPORT VECTOR MACHINE - INTUITION

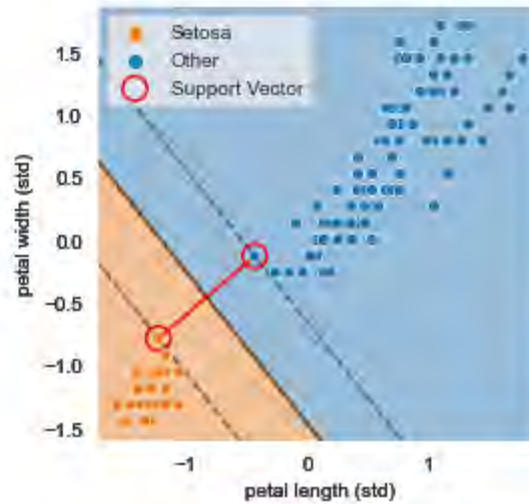
Was wäre hier die beste lineare Decision Boundary?



# SUPPORT VECTOR MACHINE - INTUITION

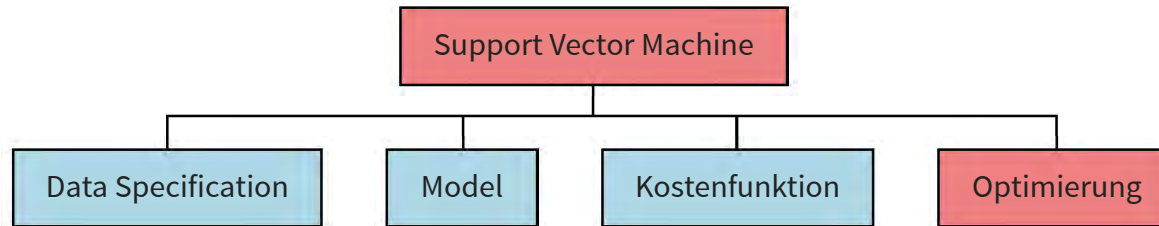
Annahme: Decision Boundary mit **maximalem Abstand** zum nächsten Punkt ist die Beste!

(LINEARE) SUPPORT VECTOR MACHINE



Idee: Verallgemeinert auf **neue** Punkte am Besten

# SUPPORT VECTOR MACHINE





# SVM - OPTIMIZATION - HARD-MARGIN - INTUITIVE FORM

$$\underset{\beta}{\text{maximize}} \quad M$$

$$\text{subject to} \quad \|\beta\| = 1$$

$$\text{subject to} \quad \beta x^{(i)} \geq M$$
$$y^{(i)} = 1$$

$$\text{subject to} \quad -(\beta x^{(i)}) \geq M$$
$$y^{(i)} = 0$$

**Solver** löst Problemstellung für uns und gibt optimale  $\beta$  zurück.

## Normiere anstatt Bedingung

VORHER

$$\begin{aligned}
 &\text{maximize} && M \\
 &\quad \beta \\
 &\text{subject to} && \|\beta\| = 1 \\
 &\text{subject to} && \beta x^{(i)} \geq M \\
 &\quad y^{(i)} = 1 \\
 &\text{subject to} && -(\beta x^{(i)}) \geq M \\
 &\quad y^{(i)} = 0
 \end{aligned}$$

NACHER

$$\begin{aligned}
 &\text{maximize} && M \\
 &\quad \beta \\
 &\text{subject to} && \frac{\beta}{\|\beta\|} x^{(i)} \geq M \\
 &\quad y^{(i)} = 1 \\
 &\text{subject to} && -\frac{\beta}{\|\beta\|} x^{(i)} \geq M \\
 &\quad y^{(i)} = 0
 \end{aligned}$$

Rechne mal  $\|\beta\|$

VORHER

$$\underset{\beta}{\text{maximize}} \quad M$$

$$\text{subject to} \quad \frac{\beta}{\|\beta\|} x^{(i)} \geq M \quad y^{(i)} = 1$$

$$\text{subject to} \quad -\frac{\beta}{\|\beta\|} x^{(i)} \geq M \quad y^{(i)} = 0$$

NACHER

$$\underset{\beta}{\text{maximize}} \quad M$$

$$\text{subject to} \quad \beta x^{(i)} \geq M \|\beta\| \quad y^{(i)} = 1$$

$$\text{subject to} \quad -(\beta x^{(i)}) \geq M \|\beta\| \quad y^{(i)} = 0$$

# SVM - OPTIMIZATION - HARD-MARGIN - ZWISCHENSCHRITT

Ersetze  $M$  mit  $\frac{1}{\|\beta\|}$

VORHER

$$\begin{aligned}
 &\text{maximize} && M \\
 &\text{subject to} && \beta x^{(i)} \geq M \\
 &&& y^{(i)} = 1 \\
 &\text{subject to} && -(\beta x^{(i)}) \geq -M \\
 &&& y^{(i)} = 0
 \end{aligned}$$

NACHER

$$\begin{aligned}
 &\text{maximize} && \frac{1}{\|\beta\|} \\
 &\text{subject to} && \beta x^{(i)} \geq 1 \\
 &&& y^{(i)} = 1 \\
 &\text{subject to} && -(\beta x^{(i)}) \geq -1 \\
 &&& y^{(i)} = 0
 \end{aligned}$$

# SVM - OPTIMIZATION - HARD-MARGIN - ÜBLICHE FORM

Maximiere statt minimiere

VORHER

$$\begin{aligned}
 &\text{maximize} && \frac{1}{\|\beta\|} \\
 &\text{subject to} && \beta x^i \geq 1 \\
 &&& y^i \beta x^i \geq 1 \\
 &&& y^i = 0
 \end{aligned}$$

NACHER

$$\begin{aligned}
 &\text{minimize} && \|\beta\| \\
 &\text{subject to} && \beta x^i \geq 1 \\
 &&& y^i \beta x^i \geq 1 \\
 &&& y^i = 0
 \end{aligned}$$

# SVM - OPTIMIZATION - HARD-MARGIN - ÜBLICHE FORM

ANFANG

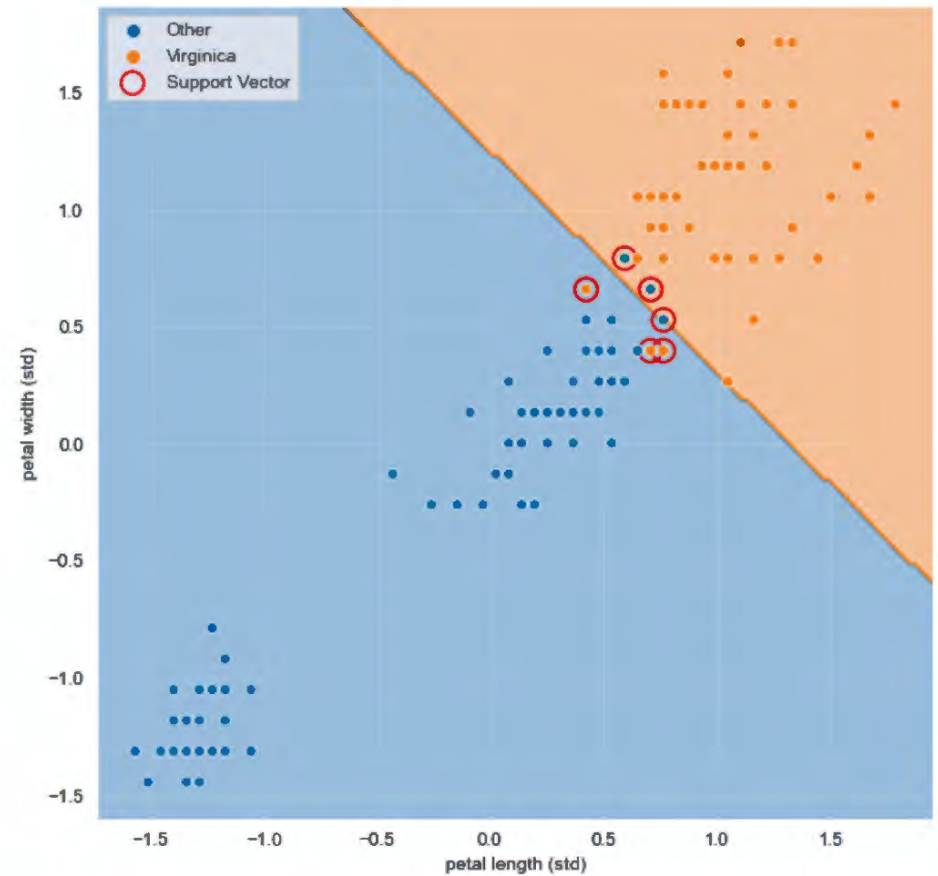
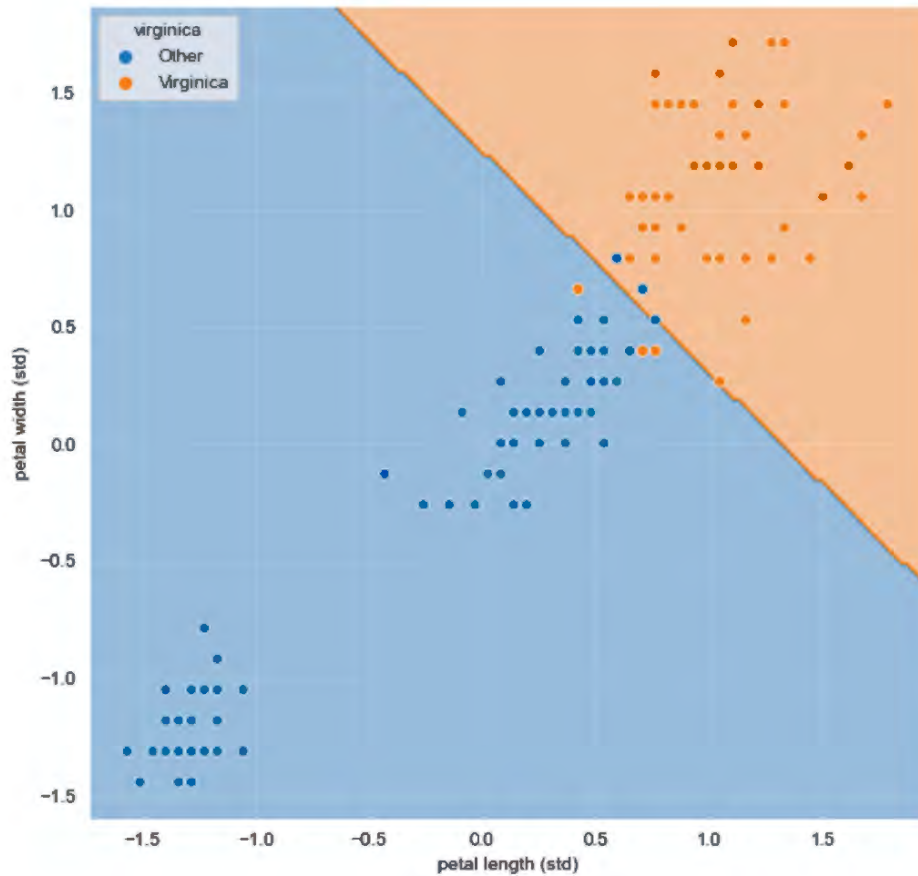
$$\begin{array}{ll} \underset{\beta}{\text{maximize}} & M \\ \text{subject to} & \|\beta\| = 1 \\ \text{subject to} & \beta x^{(i)} \geq M \\ & y^{(i)} = 1 \\ \text{subject to} & -(\beta x^{(i)}) \geq M \\ & y^{(i)} = 0 \end{array}$$

SCHLUSS

$$\begin{array}{ll} \underset{\beta}{\text{minimize}} & \|\beta\| \\ \text{subject to} & \beta x^{(i)} \geq 1 \\ & y^{(i)} = 1 \\ \text{subject to} & -(\beta x^{(i)}) \geq 1 \\ & y^{(i)} = 0 \end{array}$$

Problem ist äquivalent: Nur umformuliert. Warum das Ganze? Wir sind **M losgeworden!**

# SVM - HARD-MARGIN VS. SOFT-MARGIN



# SVM - OPTIMIZATION - SOFT-MARGIN

minimize

$$\|\beta\| + C \sum_{i=1}^N \xi_i$$

subject to

$$\beta x^{(i)} \geq 1 - \xi_i^{(i)}$$

$$y_i^{(i)} = 1$$

subject to

$$-(\beta x^{(i)}) \geq 1 - \xi_i^{(i)}$$

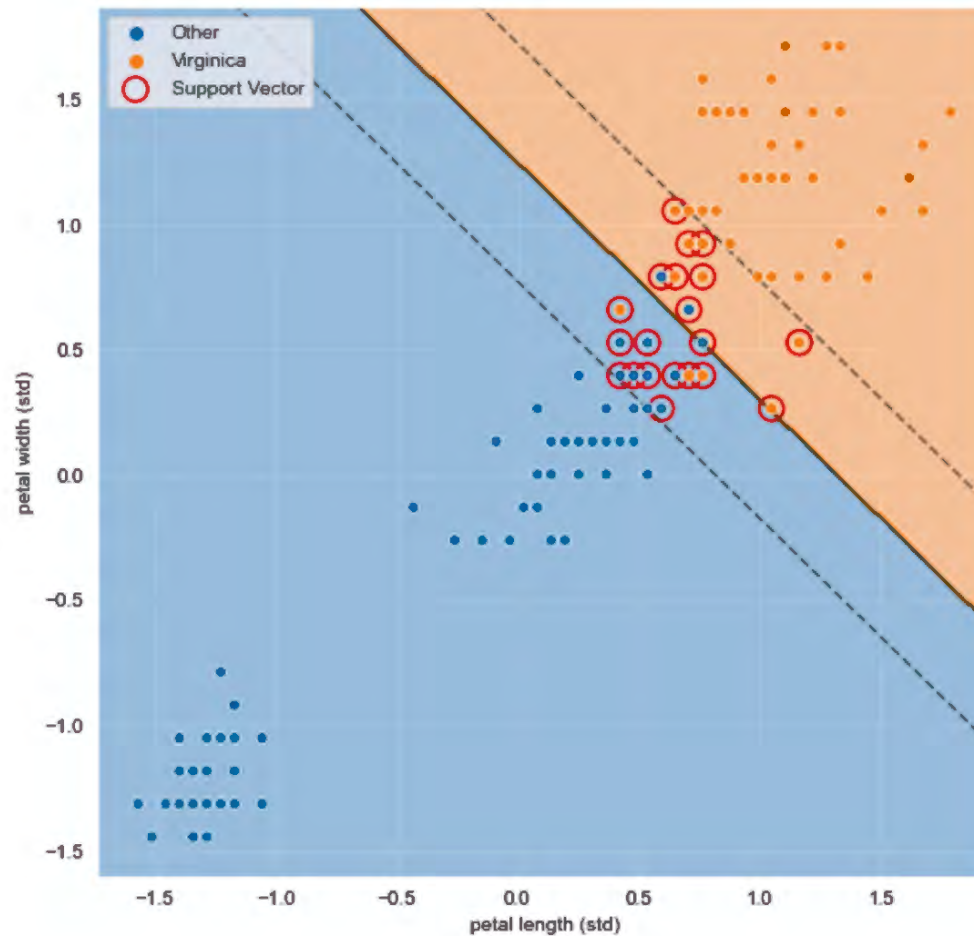
$$y_i^{(i)} = 0$$

subject to

$$\xi_i^{(i)} \geq 0$$



# SVM - HARD-MARGIN VS. SOFT-MARGIN



# SVM - HYPER PARAMETER - C

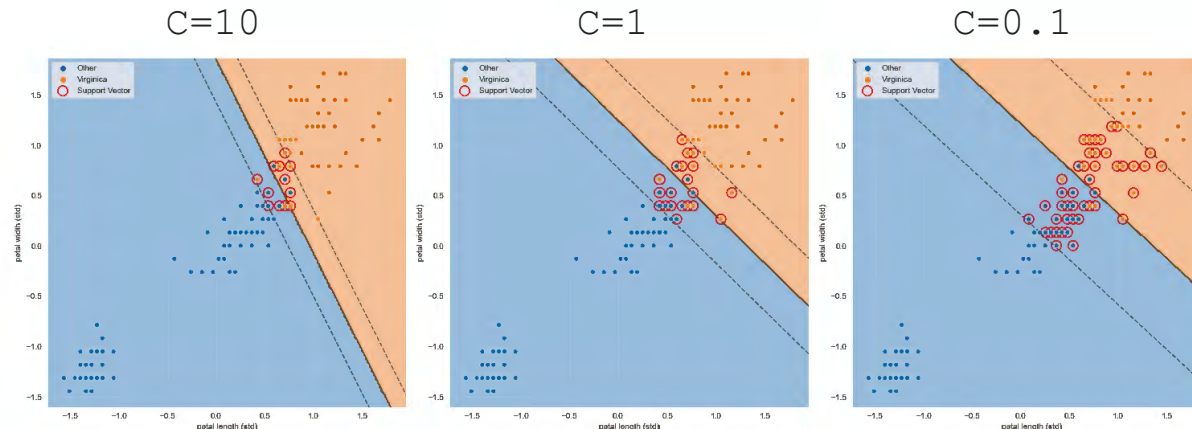
$$\text{minimize}_{\beta} \quad \|\beta\| + C \sum_i^N \xi_i$$

$$\text{subject to} \quad \beta x^{(i)} \geq 1 - \xi^{(i)} \\ y^{(i)} = 1$$

$$\text{subject to} \quad -(\beta x^{(i)}) \geq 1 - \xi^{(i)} \\ y^{(i)} = 0$$

$$\text{subject to} \quad \xi^{(i)} \geq 0$$

- Bestrafe mehr Margin-"Cheating"  
=> Invertierte Regularisierungsstärke



Kann mit Hyper Parameter Selection gesetzt werden.

# SUPPORT VECTOR MACHINE - MEHRERE KLASSEN

Keine Wahrscheinlichkeit, sprich Softmax geht nicht.

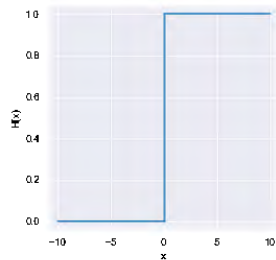
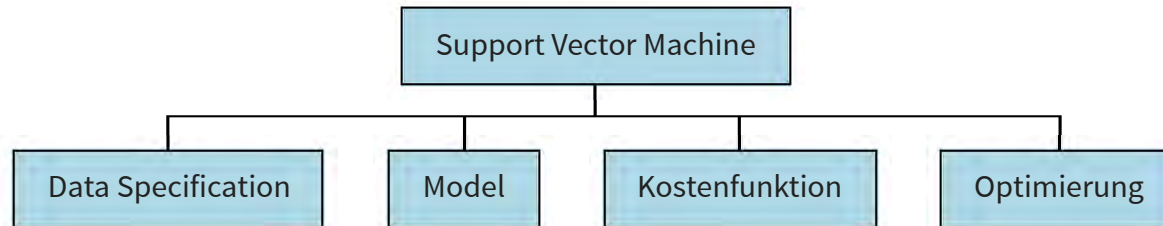
## One-vs-One Verfahren

- Ein Model **pro Klassen-Paar**:
  - **Setosa**-vs-**Versicolor**
  - **Setosa**-vs-**Virginica**
  - **Versicolor**-vs-**Virginica**
- Predict Klasse mit meisten "Votes".

---

[Additional] **One-vs-Rest**: [sklearn User Guide](#)

# SUPPORT VECTOR MACHINE



$$H(\beta_0 + x_1 \beta_1 + \dots + x_p \beta_p)$$

$$\begin{aligned}
 &\underset{\beta}{\text{minimize}} && M \\
 &\text{subject to} && \|\beta\| = 1 \\
 &\text{subject to} && \beta x^{(i)} \geq M \quad y^{(i)} = 1 \\
 &\text{subject to} && -(\beta x^{(i)}) \geq M \quad y^{(i)} = 0
 \end{aligned}$$

# QUESTIONS



# QUESTIONS

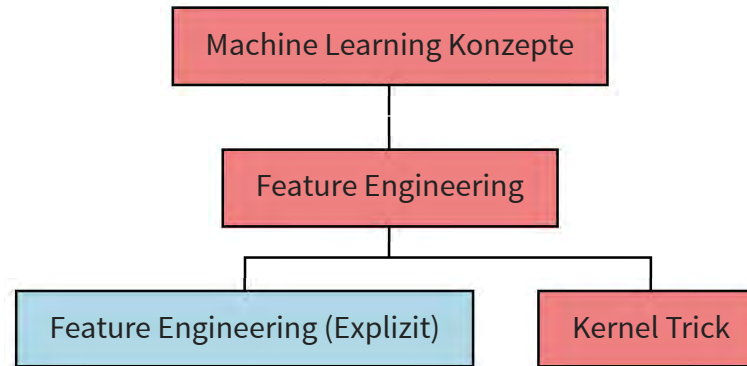
1. Was ist der Unterschied von der Logistischen Regression zur Support Vector Machine?
2. Was ist der Zusammenhang der SVM zur Linearen Regression?

# ÜBUNGSZEIT (60 MINUTEN)

`exercise/classification.ipynb`



# KERNEL TRICK





# KERNEL TRICK - INTUITION

REMINDER FEATURE ENGINEERING:

INPUT SPACE

$x$

Feature  
Engineering

FEATURE SPACE

$x, x^2$

Modell

OUTPUT SPACE

$y$

KERNEL-TRICK MACHT **IMPLIZITES FEATURE ENGINEERING** (MIT ANDEREM RECHENAUFWAND)

INPUT SPACE

$x$

Feature  
Engineering

FEATURE SPACE

$x, x^2$

SVM

OUTPUT SPACE

$y$

(Meistens) gleicher Effekt, anderes Verfahren

# KERNEL TRICK - REPRESENTER THEOREM

- Gewicht pro Datenpunkt, anstatt Gewicht pro Feature ist äquivalent

$$\vec{\beta} = \sum_{i=1}^N \alpha_i y^{(i)} \mathbf{x}^{(i)}$$

- Änderung auf das z.B. Linear Modell (Dual Form)

$$\hat{y} = \vec{\beta} \mathbf{x}$$

$$\hat{y} = \sum_{i=1}^N \alpha_i y^{(i)} (\mathbf{x}^{(i)} \mathbf{x})$$

- Wir können also  $N$   $\alpha$ 's lernen anstatt  $M$   $\beta$ 's ( $N$  = Anzahl Datenpunkte,  $M$  = Anzahl Features) und lernen das von aussen **identische Modell**
- Anzahl lernbare Parameter nicht mehr abhängig von Anzahl Features. Gut wenn  $M \gg N$

## KERNEL TRICK - KERNEL

- In der Dual-Form misst das Dot-Product  $(x^{(i)} x)$  die **Distanz** vom neuen Datenpunkt  $(x)$  zum Train-Set-Datenpunkten  $(x^{(i)})$ :

$$\hat{y} = \sum_{i=1}^N \alpha_i y^{(i)} (x^{(i)} x)$$

- **Kernel Trick**: Diese Messung von Distanz so verändern, dass es **der Distanz im Feature Space** entspricht

## KERNEL TRICK - BEISPIEL

$$\hat{y} = \sum_{i=1}^N \alpha_i y^{(i)} k(\mathbf{x}^{(i)}, \mathbf{x})$$

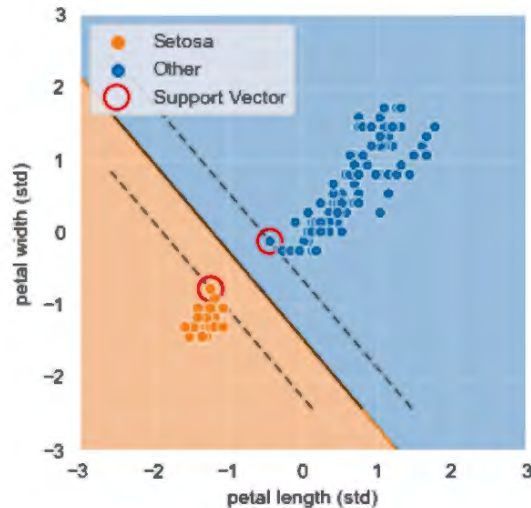
- Kernel  $k(\mathbf{x}^{(i)}, \mathbf{x})$  misst die Distanz vom neuen Datenpunkt zum Train-Set-Datenpunkten.

In der Praxis wird der Kernel-Trick meistens **nur mit der SVM** eingesetzt.

# KERNEL TRICK - MOST COMMON KERNELS

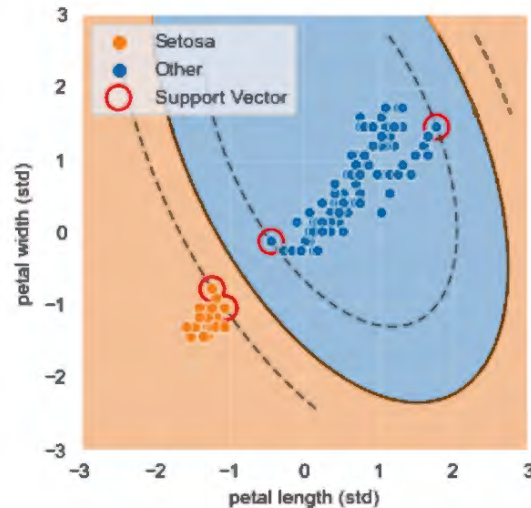
LINEAR

$$x_j x$$



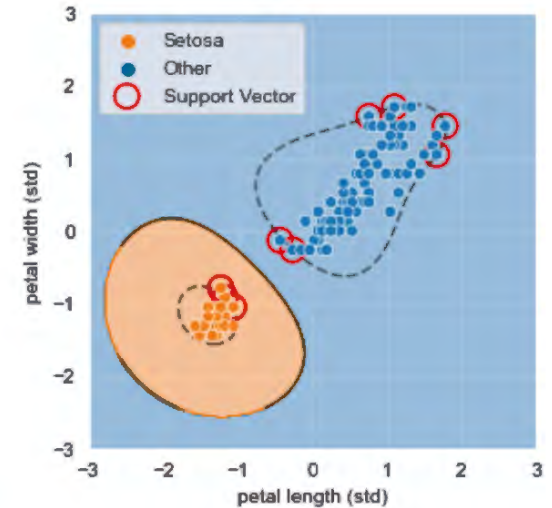
POLYNOMIAL

$$(x^{(i)} x + c)^d$$



RBF

$$\exp(-\gamma \|x - x^{(i)}\|^2)$$



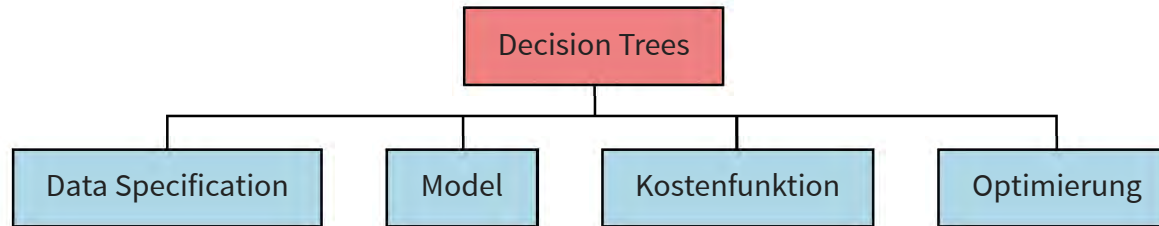
Kernel macht die Modelle nicht linear.

# SUPPORT VECTOR MACHINE (UND KERNEL TRICK) - CODE

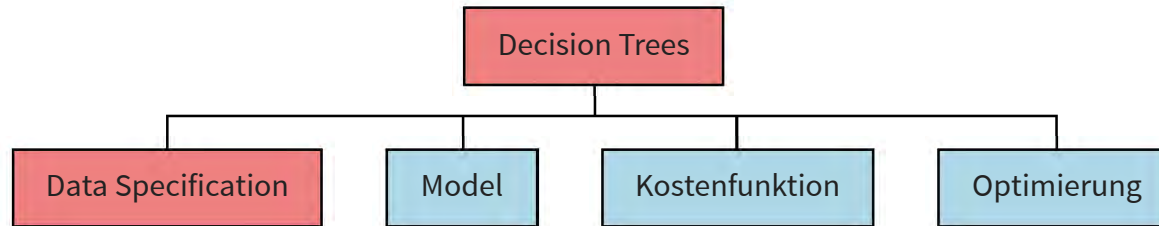
```
support_vector_machine.ipynb
```



# DECISION TREES



# DECISION TREES

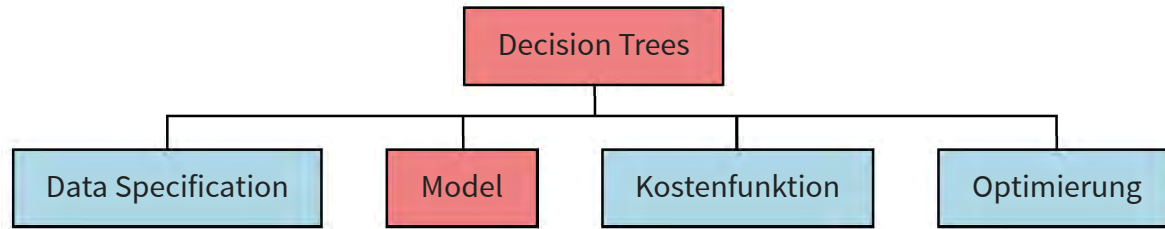




# DECISION TREES - DATA SPECIFICATION

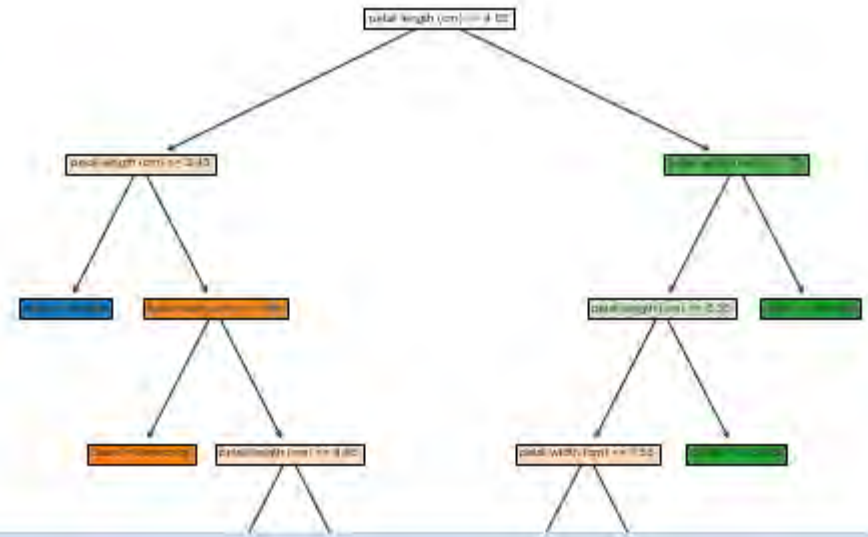
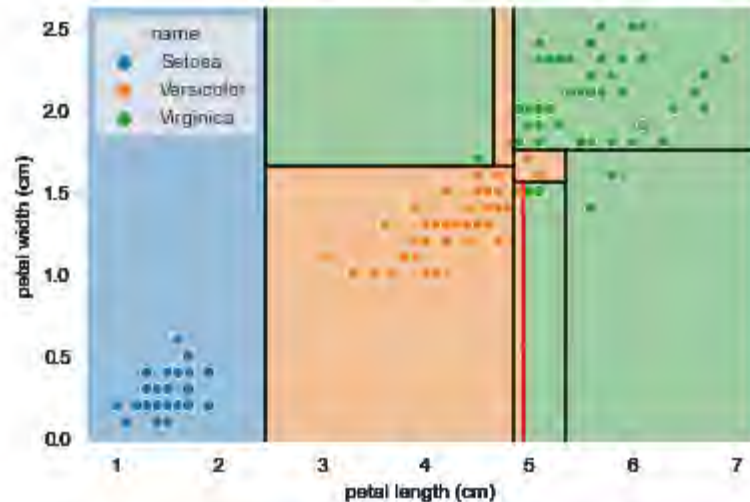
1. Was ist die **Ziel-Variable**, z.B. `name`
2. Welche **Features** wählen wir, z.B. um eine Blume zu repräsentieren (`petal-length (cm)`, `petal-width (cm)`, ...)
3. Kategorische Features müssen **encoded** werden.

# DECISION TREES



# DECISION TREES - INTUITION

- Feature-Space in Regionen aufteilen
- Mit Train-Samples in Regionen eine Vorhersage treffen.



Abbruchbedingung z.B. Maximale Tiefe vom Tree  
(z.B. max depth=3)

# DECISION TREES - MODELL

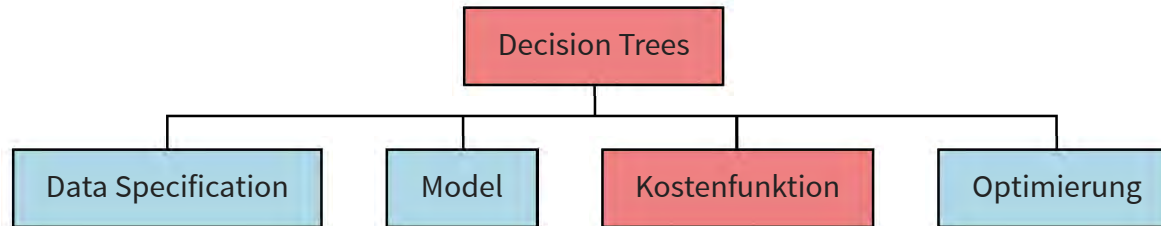
1. Feature Space unterteilen in Regionen ( $R_1, \dots$ )  
mittels **Decision Tree** ()
2. **Wahrscheinlichkeit** für Klasse ( $k$ ) pro Region ( $R_m$ )

$$p(y = k | R_m) = \frac{1}{N_m} \sum_{x^{(i)} \in R_m} I(y^{(i)} = k)$$

3. Klasse mit **höchster Wahrscheinlichkeit vorhersagen** für Region  $R_m$ :

$$\hat{y}_{R_m} = \arg \max_k p(y = k | R_m)$$

# DECISION TREES



## DECISION TREE - KOSTENFUNKTION

Impurity (Verunreinigt) der Regionen ( $R_1, \dots$ ) messen.

$$\sum_{R_m} \frac{N_{R_m}}{N} M(R_m)$$

Most common impurities:

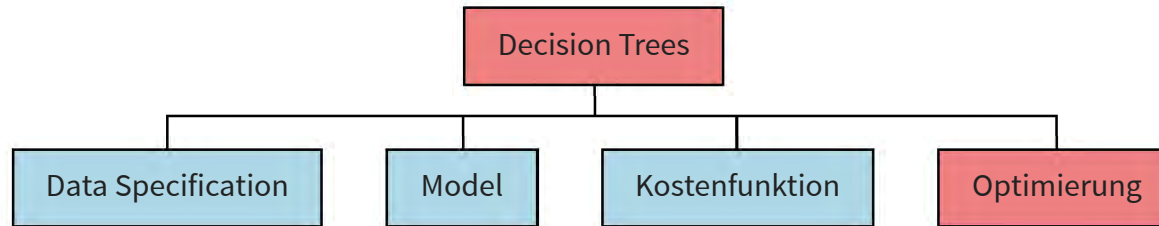
- Gini-index

$$M(R_m) = \sum_k p(y = k | R_m) (1 - p(y = k | R_m))$$

- Cross-Entropy

$$M(R_m) = - \sum_k p(y = k | R_m) \log(p(y = k | R_m))$$

# DECISION TREES



# DECISION TREE - OPTIMIERUNG

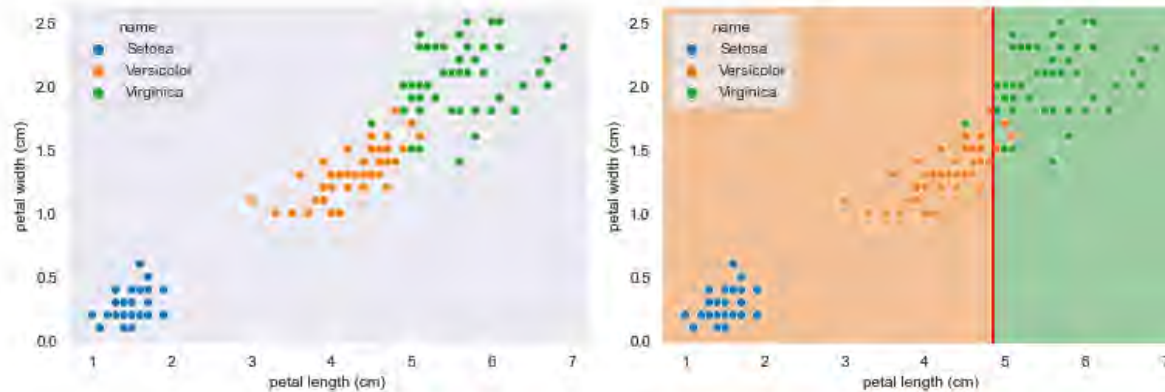
- Theoretisch existiert ein optimaler Tree, der die Kostenfunktion perfekt minimiert  
=> NP-Hard (braucht zu viel Rechenzeit)
- Praxis eine guten Tree mit einem Greedy-Algorithmus finden, der Kostenfunktion gut minimiert  
=> P-Hard (braucht wenig Rechenzeit)

Allenfalls nicht mal schlimm: Optimaler Tree ist "optimal" nur für Train-Set nicht neue Daten!



# DECISION TREE - OPTIMIERUNG

Greedy: Finde besten Split für aktuelle Region  $R$ , zum Unterteilen in  $R_1$  und  $R_2$  nach  $M$ .



$$R_1(f, t) = \{X | X_f \leq t\}$$

$$R_2(f, t) = \{X | X_f > t\}$$

Bestes Feature ( $f$ ) mit bestem Threshold ( $t$ )


$$f, t = \arg \min_{(f, t)} \left[ \frac{N_{R_1}}{N_R} M(R_1(f, t)) + \frac{N_{R_2}}{N_R} M(R_2(f, t)) \right]$$

# DECISION TREE - OPTIMIERUNG

$$\mathbf{f}, \mathbf{t} = \arg \min_{(\mathbf{f}, \mathbf{t})} \left[ \frac{N_{R_1}}{N_R} M(R_1(\mathbf{f}, \mathbf{t})) + \frac{N_{R_2}}{N_R} M(R_2(\mathbf{f}, \mathbf{t})) \right]$$

- Wiederholt angewendet auf  $R_1$  und/oder  $R_2$  je nach Abbruchbedingung.

# DECISION TREE - FÜR REGRESSION

1. Feature Space unterteilen in Regionen ( $R_1, \dots$ )  
mittels **Decision Tree** ()
2. Durchschnitt vorhersagen für  $R_m$ :

$$\hat{y}_{R_m} = \frac{1}{N_{R_m}} \sum_{x_i \in R_m} y_i$$

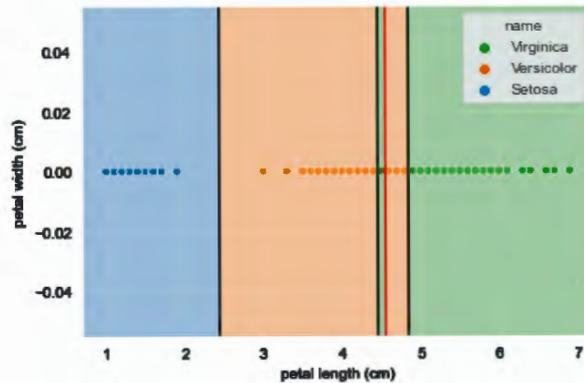
---

Beim Optimieren: Split nach **MSE** (oder andere Regression-Metrik)

# DECISION TREE - INTUITION

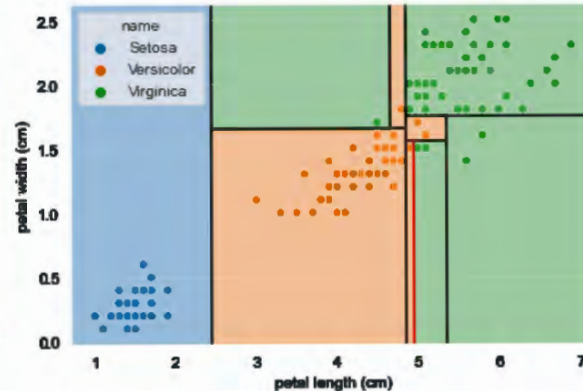
## 1 FEATURE

Unterteilen 1D Feature-Space  
in 1D Teilräume



## 2 FEATURE

Unterteilen 2D Feature-Space  
in 2D Teilräume



## 3 FEATURE

Unterteilen 3D Feature-Space  
in 3D Teilräume



Nicht lineare Zusammenhänge einfach lernbar.

## DECISION TREE - EIGENSCHAFTEN

- Decision Trees können **gut interpretiert** werden.
  - Decision Trees können schnell **overfitten**
  - Decision Trees können leicht **unterschiedlich** gemacht werden (nicht alle Daten, Zufall in Splits)  
Dadurch sind sie stark als **Ensemble**, wie **Random Forest** und **Gradient Boosting**
- 
- **Robuster bei vielen Features:** Trees haben eine inherente Eigenschaft für **Feature-Selection**.  
Unwichtige Features werden nicht/selten für Splits verwendet.

# RANDOM FOREST

- Trainiere viele **Decision Trees** (üblicherweise 100-1000) auf zufälligen Teilen der Daten
- Als finale Prediction wird dann der **Durchschnitt der trainierten Decision Trees** verwendet.

Diese Art von Ensemble nennt man **Bagging**

In der Praxis werden **Decision Trees** meistens als **Ensemble** eingesetzt und selten als einzelnes Modell.

# DECISION TREES - CODE

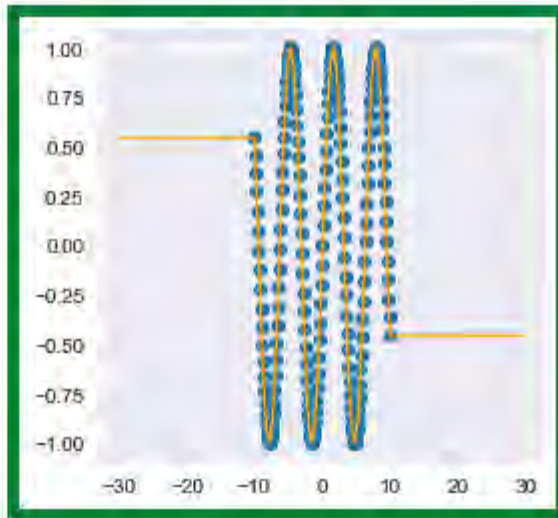
```
decision_trees.ipynb
```



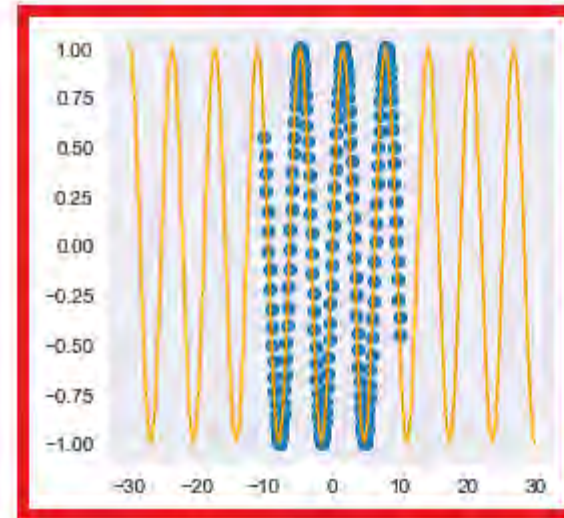
# DECISION TREES - LIMIT

GLOBAL PATTERN

POSSIBLE



IMPOSSIBLE (IN FEATURE SPACE)



Decision Trees (und deren Ensemble) lernen "nur"  
**interpolation** zwischen den Datenpunkten.



# ALTERNATIVE & ADDITIONAL RESOURCES

- [Alternative] Decision Tree
  - [Video: "Decision Tree Classification Clearly Explained!"](#)
  - [sklearn Documentation](#)
- [Additional] Ensemble, Random Forest, Gradient Boosting:
  - Wird MAS behandelt
  - [Video\(s\) about Bagging \(and Boosting\)](#)
  - Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow - Kapitel 7

# QUESTIONS



# QUESTIONS

1. Was ist ein (binary) Decision Tree?
2. Kann ein Decision Tree einfach eine lineare Decision Boundary lernen?

# DATA SCIENCE PITFALL - PEEKING

Aus Daten Lernen und Entscheidungen dürfen **nicht von ungesehenen Daten (Test-Daten)** beeinflusst werden. Ansonsten ist Evaluation des finalen Modells biased (zu optimistisch).

---

Beispiele:

- **Modelle** nur auf den **Train-Daten** fitten (z.B. LinearRegression)
  - **Preprocessing** nur auf den **Train-Daten** fitten (z.B. StandardScaler)
  - **Feature Engineering** nur auf den **Train-Daten** fitten (z.B. durchschnittlicher Quadratmeterpreis)
  - **Model-Selection** mit **Validation-Daten** ausführen (z.B. Cross-Validation)
  - **Data Analysis nicht auf den Test-Daten** ausführen (z.B. Pairplot)
- 

[Alternative Quelle]: [sklearn User Guide](#)

[Alternative Quelle]: Section 3.1 in Paper "How to avoid machine learning pitfalls: a guide for academic researchers"

# DATA SCIENCE PITFALL - MODEL ENTWICKLUNG

Folge folgenden Schritten:

1. **Schätze** was ist bestenfalls erreichbar gegeben dem Problem (z.B. 5-10%)
  - z.B. wie genau kann es eine Fachperson schätzen
  - Wenn Modell besseres Ergebnis erziele, könnte ein Bug der Grund sein
2. Erstelle eine **Baseline**, Upper-Bound (z.B. 50%)
3. Erstelle ein erstes **einfaches Modell** (z.B. 40%)
4. **Iteratives** verbessern vom Modell mit Daten, Annahmen
  - **Systematisches Vorgehen**: Planen, Umsetzen, Evaluieren
  - **Ein Schritt nach dem anderen**: Verbesserungs-Grund verstehen
    - z.B. ein neues Feature oder eine neue Modellart oder ein neues Feature Engineering
  - **Verkompliziere das Modell nicht** für "nur" minimale Verbesserungen
    - Occam's Razor, "einfachste Theorie allen anderen vorzuziehen"
    - Einfachheit vom Modell ist eine Qualität, betreffend Deployment, Maintenance, Extension

# DATA SCIENCE PITFALL - INCONSISTENT PREPROCESSING

Alle **Feature Preprocessing Schritte** müssen **identisch** ausgeführt werden **für alle Daten** (Train-Set, Validation-Set, Test-Set, in Produktion).

```
X_train, X_val, y_train, y_val = train_test_split(X, y, random_state=42)

ss = StandardScaler()
ss.fit(X_train) # "Lernphase": Berechnet Mean und Standardabweichung
X_train_std = ss.transform(X_train)

lr = Ridge()
lr.fit(X_train_std, y_train)

"""
Kritische Stelle:
1. We use the same StandardScaler as for X_train.
2. We do NOT fit the StandardScaler on the validation data!
"""

X_val_std = ss.transform(X_val)
y_val_hat = lr.predict(X_val_std)
```

# DATA SCIENCE PITFALL - METRIK WAHL

- Richtige Metrik ist vom Business Case (oder verfügbaren Daten) motiviert (z.B. prozentualer Fehler (MAPE) anstatt absoluter Fehler (MSE))
- Regression
  - Eine robuste Metrik (z.B. MAE) kann Model stabilisieren bei unsicherer Datenqualität.
- Klassifikation
  - Bei imbalanced Klassen verwende nicht Accuracy sondern eine entsprechende Metrik (z.B. F1-Score)
  - Wenn False Negative und False Positive unterschiedlich zu bewerten sind, dann Cost-Sensitive Learning, Threshold Moving

---

[Alternative Quelle]: Section 4.6 in Paper "How to avoid machine learning pitfalls: a guide for academic researchers"

# ÜBUNGSZEIT (WEITERE 60 MINUTEN)

`exercise/classification.ipynb`

