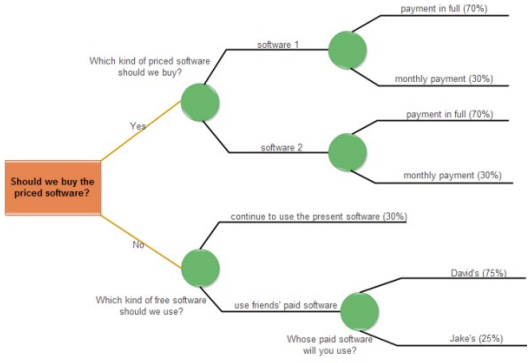
# Datengetriebene Methoden der Fehlerdiagnose

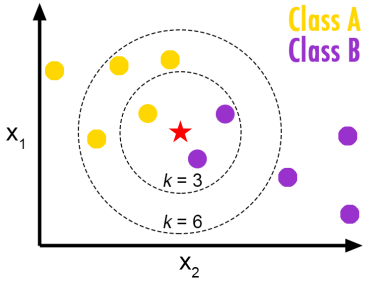
von

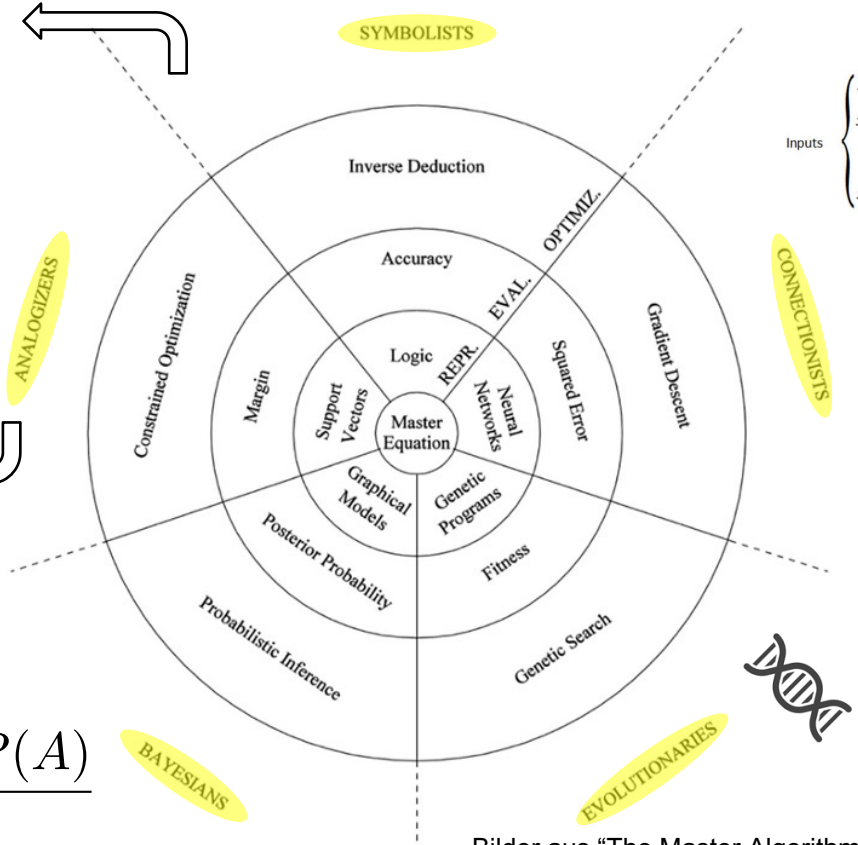
**Prof. Dr.-Ing. Oliver Nelles**

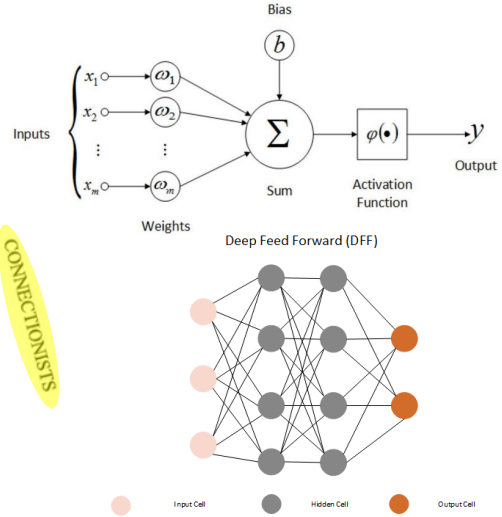
# Inhaltsübersicht

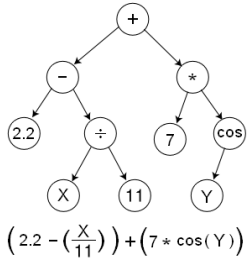
# Hintergrund: Künstliche Intelligenz (KI)



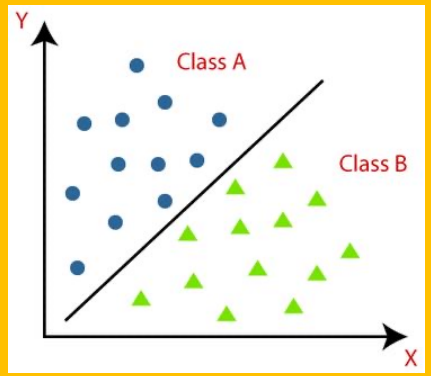

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Bilder aus “The Master Algorithm” von Pedro Domingos, Penguin, 2017

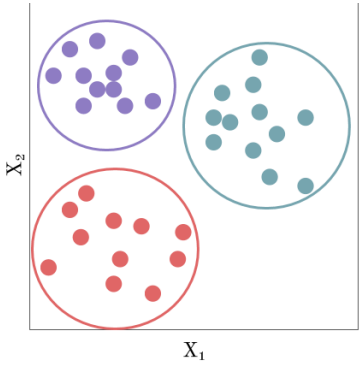
# Hintergrund: Künstliche Intelligenz (KI)

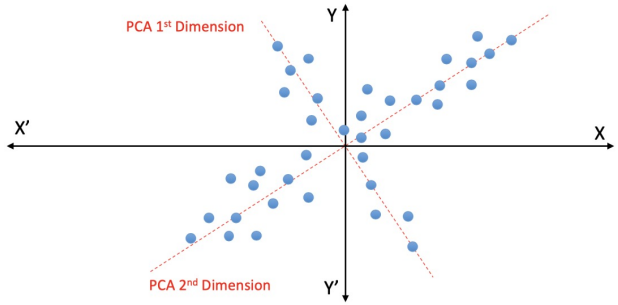
Überwachtes Lernen

## Klassifikation

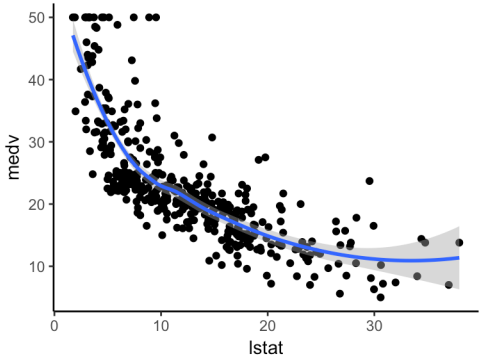



## Unüberwachtes Lernen

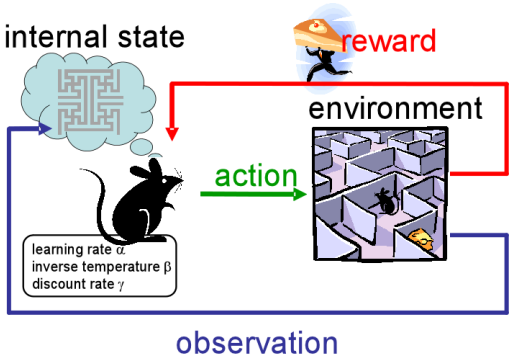


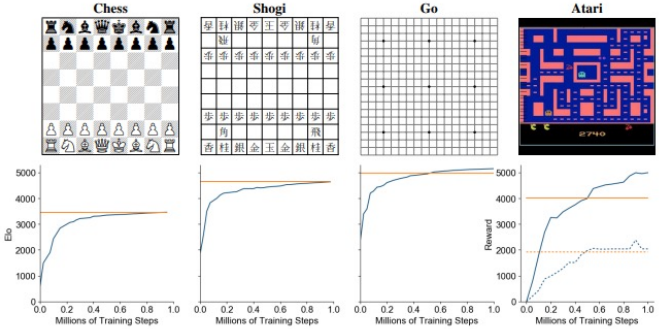




## Regression/Approximation




## Strategie/Reinforcement-Lernen

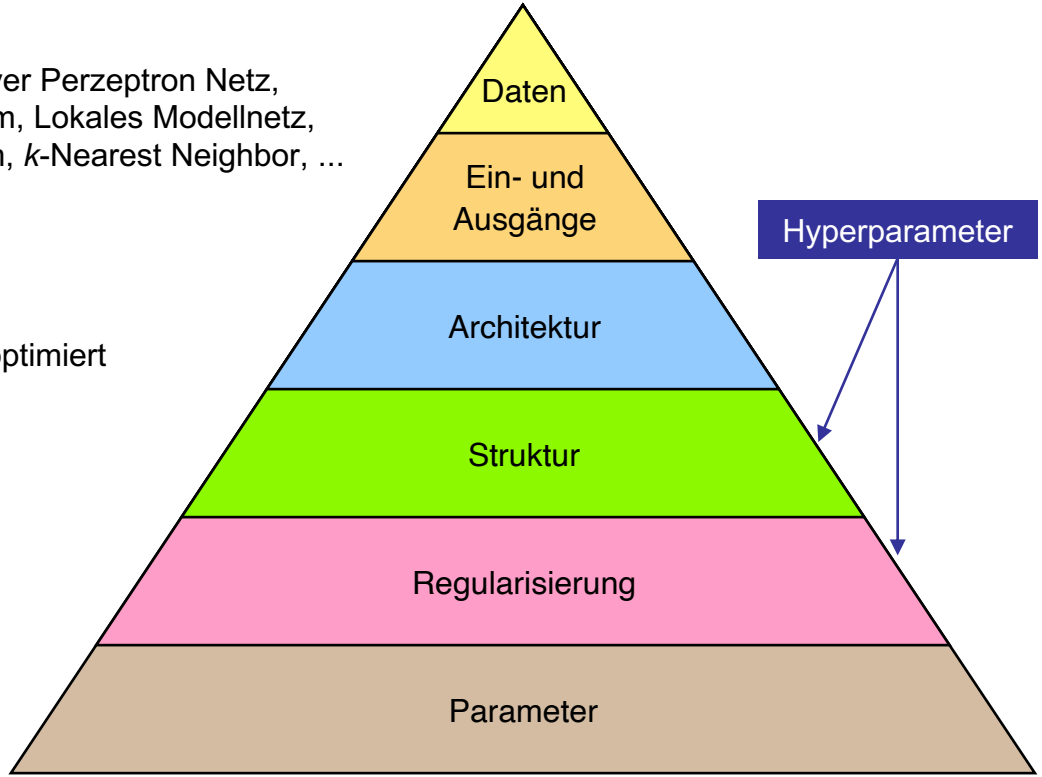


# Motivation: Datengetriebene Modelle

## Ebenen datengetriebener Modelle

- Architektur
  - Tabelle, Kennfeld, Lineares Modell, Polynom, Multilayer Perzeptron Netz, Radiales Basisfunktionen Netz, (Neuro)-Fuzzy-System, Lokales Modellnetz, Support Vector Machine, Gauß-Prozessmodell, Baum, *k*-Nearest Neighbor, ...
  - Durch **Experten** ausgewählt
- Struktur
  - Anzahl Neuronen, Regeln, Knoten, Terme
  - Mit **Validierungsdaten** oder **Informationskriterien** optimiert
- Regularisierung
  - Größe des Strafterms, wann „Early Stopping“?, wie viele Nebenbedingungen?
  - Mit **Validierungsdaten** optimiert
- Parameter
  - Gewichte, Koeffizienten, ...
  - Mit **Trainingsdaten** optimiert



# Motivation: Datengetriebene Modelle

## Qualität der Daten

- Für **datengetriebene** Methoden **entscheidend** für die **Qualität** des Modells bzw. Klassifikators
- Je mehr **Vorwissen** (*prior knowledge*) im Modell steckt, desto **weniger entscheidend** sind die **Daten** als Informationsquelle
- Höhere **Modellkomplexität**/flexibilität → Mehr Daten
- Mehr Eingänge bzw. höhere **Modelldimensionalität** → Mehr Daten
- Schlechtere **Datenqualität** → Mehr Daten

## Qualitätskriterien

- **Datenmenge** *N*
- Ausmaß/Größe von **Störungen** (systematisch) und **Rauschen** (stochastisch) in den Daten
- **Verteilung** der Daten
  - gleichmäßig
  - in Clustern bzw. Klumpen konzentriert
  - in manchen Regionen dicht, in anderen Regionen dünn

Dieser Aspekt wird behandelt in:
7. Dichteschätzung
8. Verteilung der Datenpunkte

→ Datengetriebene Methoden können nur dort „**gut**“ sein, wo auch **genügend Daten** in ausreichender **Qualität** vorhanden sind

**Gegenbeispiel: Extrapolation** → Das Verhalten wird durch die Wahl der **Modellarchitektur** und **-struktur** geprägt!

# Motivation: Fehler

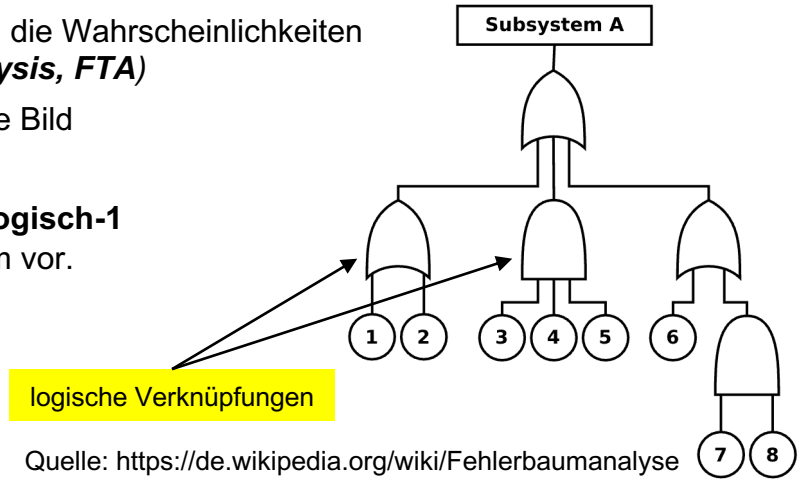
## Welche Fehler sind möglich?

- Bevor man ein System für Fehlererkennung und -diagnose aufbaut, muss man sich erstmal überlegen, **welche Fehler** überhaupt **passieren** können
- Das beliebteste Mittel zur **Vermeidung** von Fehlern durch ein gutes Design ist die **Fehlermöglichkeits- und -einflussanalyse (*Failure Mode and Effects Analysis, FMEA*)**
- Es werden aber mögliche Fehler übrig bleiben, obwohl die FMEA vielleicht deren Anzahl, Wahrscheinlichkeit und drastische Konsequenzen reduziert hat

- Ein Möglichkeit Ursache-Wirkungsbeziehungen zu veranschaulichen und die Wahrscheinlichkeiten von Fehler zu ermitteln bietet die **Fehlerbaumanalyse (*Fault Tree Analysis, FTA*)**
- Ist auch unter dem Namen **Fehlerzustandsbaumanalyse** bekannt, siehe Bild
- Systemanalyse basierend auf der **Boolescher Algebra**
- Ein **Fehlerbaum** beschreibt eine Ausfallsfunktion, die bei dem Zustand **logisch-1** einen Ausfall ausdrückt, bei **logisch-0** liegt ein **funktionsfähiges** System vor.
- Ist u.a. in der Nuklear-, Luft- und Raumfahrtindustrie vorgeschrieben

- Hieraus ergibt sich eine Liste mit Fehlern 1, 2, ..., *m*, die in einem Fehlererkennungs- und -diagnosesystem behandelt werden können

**8 Ereignisse können zum Ausfall von Subsystem A führen**




Quelle: https://de.wikipedia.org/wiki/Fehlerbaumanalyse

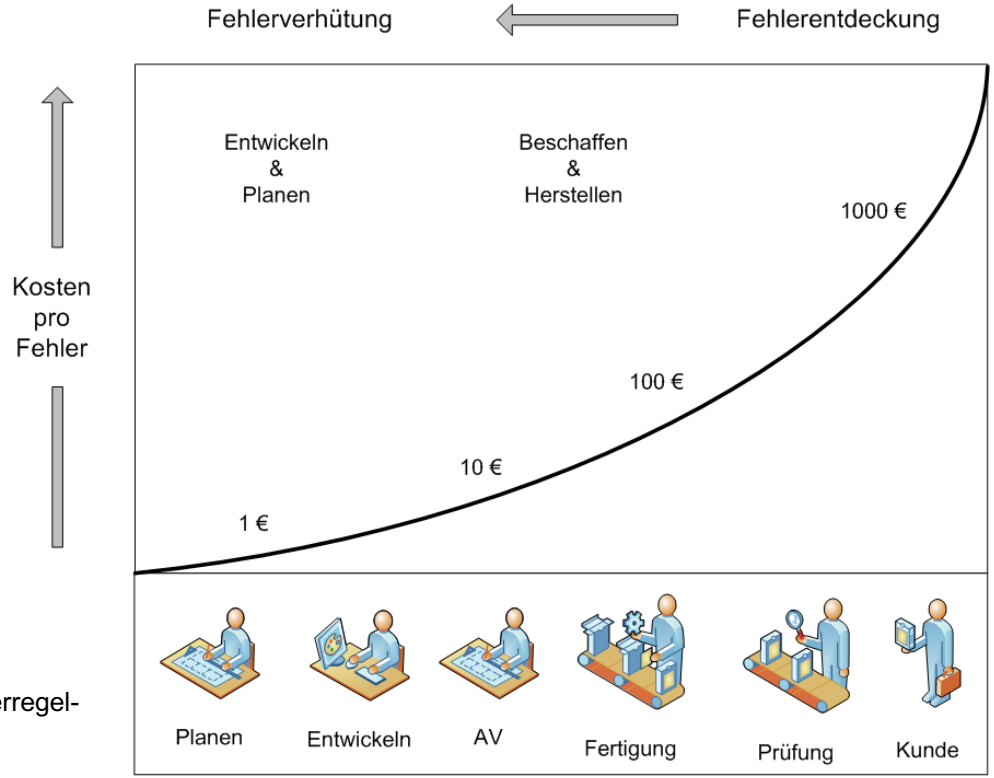
# Motivation: Fehler

## Fehler in welcher Stufe der Wertschöpfung?

- In dieser Vorlesung geht es hauptsächlich um **Fehler im laufenden Betrieb**, also in der letzten Stufe der Wertschöpfungskette
- Es ist aber wichtig sich zu vergegenwärtigen, dass die Kosten eines Fehlers stark ansteigen, je später er erkannt wird
- Das gilt im laufenden Betrieb aber auch davor: während aller Stufen der Wertschöpfung

Quelle: https://www.sixsigmablackbelt.de/fehlerkosten-10er-regel-zehnerregel-rule-of-ten/#Fehlerkosten_nach_DIN_55350

## Fehlerkosten: 10er Regel (*rule of ten*)



# Literatur

## Fehlerdiagnose

- Rolf Isermann: „Fault-Diagnosis Systems“, Springer, 2009
- Rolf Isermann: „Fault-Diagnosis Applications: Model-Based Condition Monitoring: Actuators, Drives, Machinery, Plants, Sensors, and Fault-tolerant Systems“, Springer, 2011
- Janos Gertler: „Fault Detection and Diagnosis in Engineering Systems“, CRC Press, 2019
- Steven X. Ding: „Advanced Methods for Fault Diagnosis and Fault-tolerant Control“, Springer 2020

## Klassifikation

- Brian D. Ripley: „Pattern Recognition and Neural Networks“, Cambridge University Press, Cambridge, 2008
- Trevor Hastie, Robert Tibshirani, Jerome Friedman: “The Elements of Statistical Learning: Data Mining, Inference, and Prediction“, Second Edition, Springer 2009
- Christopher M. Bishop: „Neural Networks for Pattern Recognition“, Oxford University Press, 1996
- Richard O. Duda, Peter E. Hart, David G. Stork: „Pattern Classification“, Wiley, 2000, neue Auflage erscheint bald

## Andere Vorlesungen

- Oliver Nelles: „Signalverarbeitung“
- Oliver Nelles: „Neuronale Netze und Fuzzy-Systeme“

# 1. Grundlagen Fehlererkennung und -diagnose

# Inhalt

## Datengetriebene Modelle

- Im Gegensatz zu **theoretischen Modellen** (***first principles models***) werden sie ausschließlich oder vorwiegend mit Hilfe von Daten erstellt.
- **Theoretische Modelle** nennt man auch **White-Box-Modelle**; **datengetriebene Modelle** nennt man auch **Black-Box-Modelle**.
- Gerade im Ingenieurbereich gibt sehr viele **Mischformen** aus beiden, sog. **Grey-Box-Modelle**, die stärker Richtung white tendieren können (light grey-box) oder stärker Richtung black (dark grey-box).

## Vorteile datengetriebener Modelle

- Wenig Expertenwissen notwendig
- Sehr kurze Entwicklungszeiten
- Billig
- Oft mit wenig Rechenaufwand auszuwerten
- Notwendiges Know-how ist universell, d.h. von einem Modell auf ein anderes übertragbar
- Kann mit neuen Daten relativ problemlos adaptiert oder erweitert werden

## Nachteile datengetriebener Modelle

- Viele Daten notwendig
- Expertenwissen für gute Versuchsplanung (*design of experiments*) und Datenakquisation notwendig
- Oft keine/wenig Transparenz, Interpretierbarkeit, Nachvollziehbarkeit
- Schlechtes Extrapolationsverhalten
- Oft nur mit viel Rechenaufwand zu trainieren
- Zur Zeit notwendiges Machine Learning Know-how nicht weit verbreitet

## Wichtige Begriffe

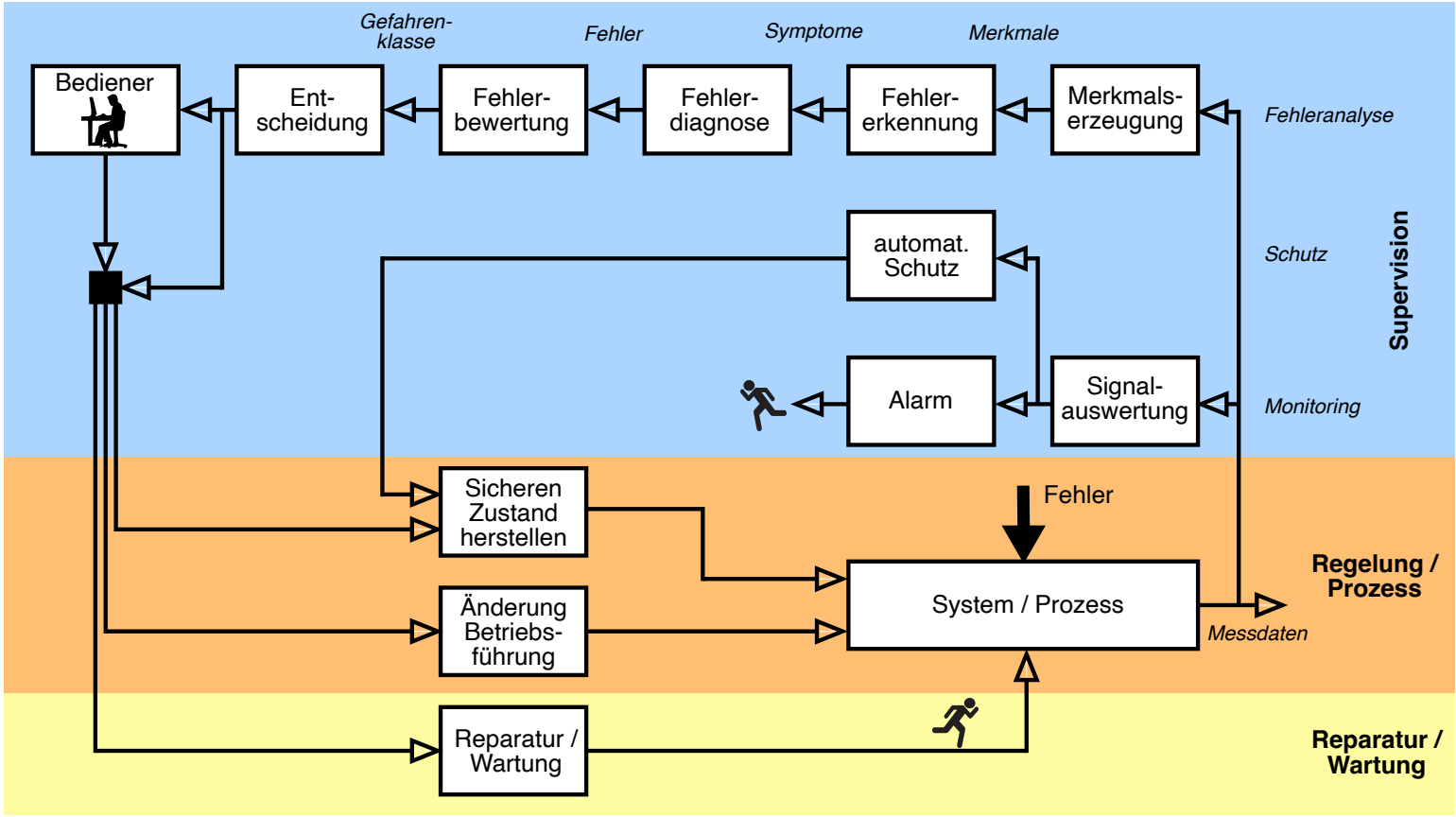
- **Überwachung** (***Monitoring***): **Ständig** wird ein System anhand der messbaren Signale überwacht, typischerweise auf Grenzwert/Toleranzüberschreitungen. Bei Überschreitung wird ein **Alarm** ausgelöst.
- **Supervision**: Zusätzlich zur Überwachung werden **Maßnahmen** zur Beseitigung des kritischen Zustand automatisch eingeleitet.

- **Fehlererkennung**: **Erkennung** eines Fehlers
- **Fehlerisolation**: Zusätzlich zur Erkennung werden Informationen über den **Ort** des Fehlers festgestellt
- **Fehlerdiagnose**: Zusätzlich zur Isolation werden Informationen über **Art**, **Größe** und **Ursache** des Fehlers festgestellt

- **Prozess**: Apparat, Maschine, Anlage unter Beobachtung
- **Aktor/Stellglied**: Beeinflusst den Prozess bzw. dessen Verhalten, z.B. Fahrpedal, Bremse, Ventil, ...
- **Sensor/Messglied**: Misst Signal und wandelt es in digital weiter verarbeitbare Information
- **Residuum**: Indikator für das Auftreten eines Fehlers. Ist ≈ 0 im fehlerfrei Fall und betragsmäßig >> 0 in Fehlerfall
- **Symptom**: Typisches für einen Fehler charakteristisches Merkmal aus Residuenabweichungen und deren Muster, d.h. welche Residuen wie stark anschlagen und welche nicht
- **Klassifikator**: Trennt Klassen voneinander, z.B. „In Ordnung“ von „Fehler“ oder auch „Fehler 1“ von „Fehler 2“ ... von „Fehler *m*“

# 1.1 Fehlererkennung und Fehlerdiagnose

**Zusammenspiel der Komponenten eines Fehler-diagnosesystems**



## Einfache signalbasierte Ansätze

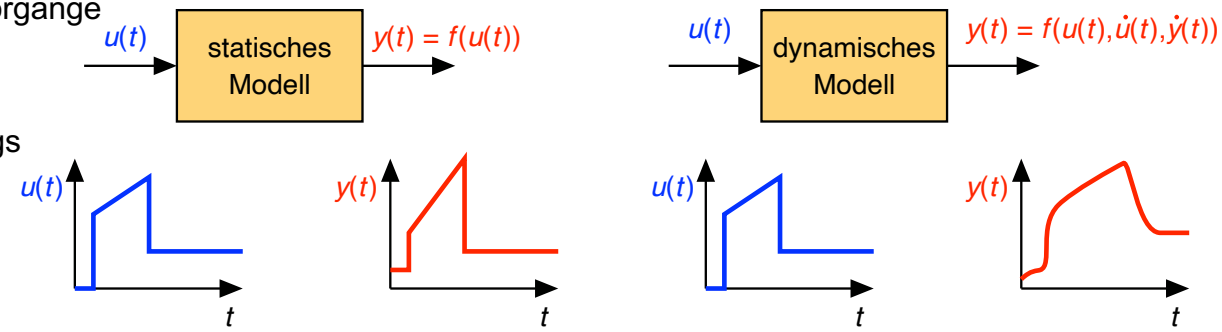
- Weit **verbreitet**
- Oft nur Vergleich mit **Schwelle** oder **Toleranzen**
- Benötigt meist große Änderung (**unempfindlich**)
- Zu wenig Information für Fehler*diagnose*

## Fortgeschrittene Ansätze (typischerweise modellbasiert)

- Frühe Erkennung des Fehler zu Beginn der Entwicklung
- Diagnose von Fehlern individuell in Stellgliedern, Sensoren und Teilprozessen
- Fehlererkennung im geschlossenen Regelkreis
- Überwachung auch während transienter Vorgänge

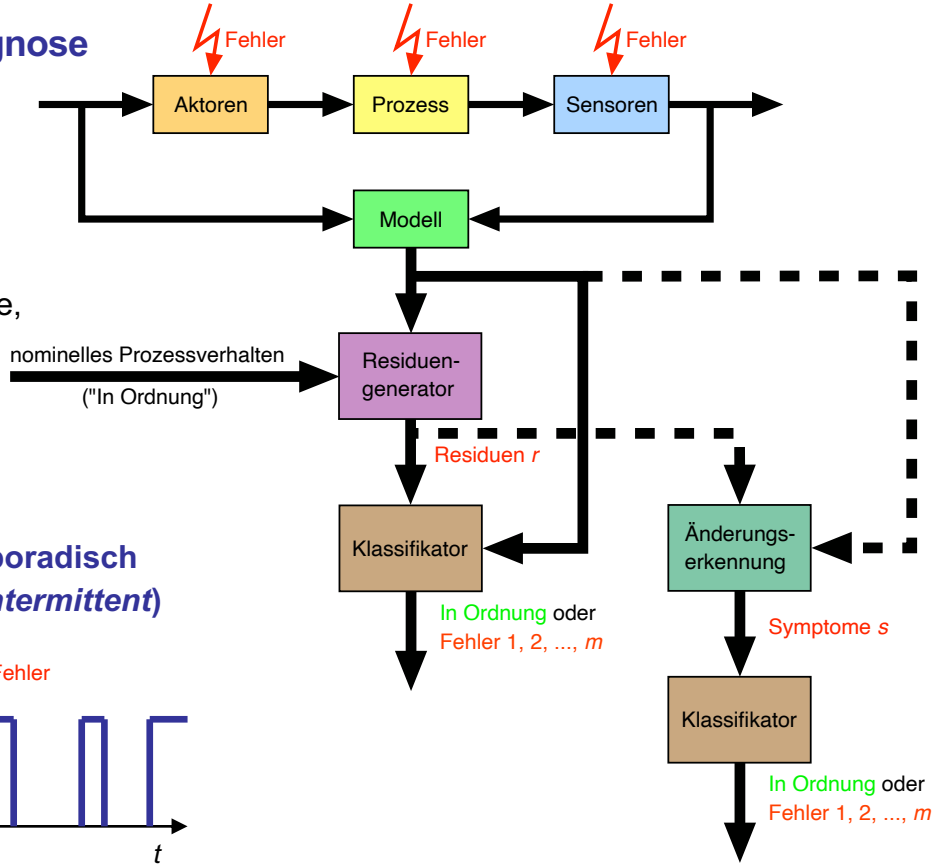
## Statische und dynamische Modelle

- Statisch: Ausgang ist Funktion des Eingangs
- Dynamisch: Ausgang hängt zusätzlich von der Historie (Zeitverlauf) von Ein- und Ausgang ab, d.h. hat Gedächtnis.



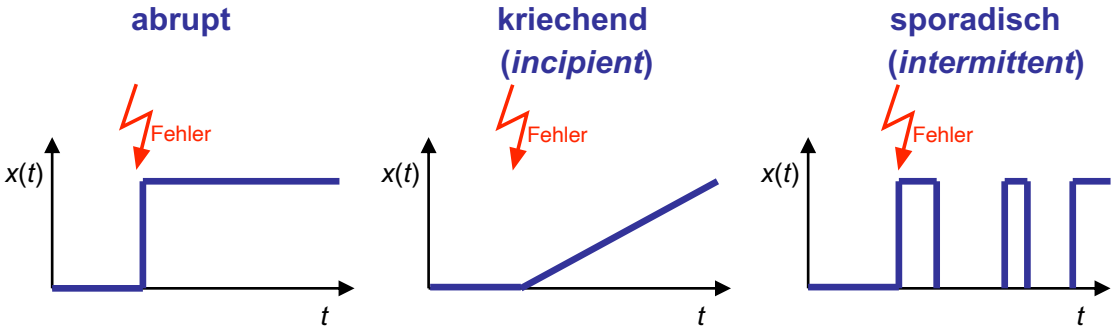
## Aufbau eines Systems zur Fehlererkennung und -diagnose

- Manchmal wird hinter die Residuengenerierung noch eine Symptomgenerierung geschaltet, die eine Änderung der Residuen detektiert (gestrichelt).
- Der Klassifikator kann den Residuen als Information auskommen oder noch weitere Informationen aus dem Modell benötigen, wie Arbeitspunkt oder dessen Änderungsrate, Umgebungsbedingungen, Signalcharakteristika, ...




## Fehlercharakteristiken

- Fehler können sich zeitlich unterschiedlich entwickeln

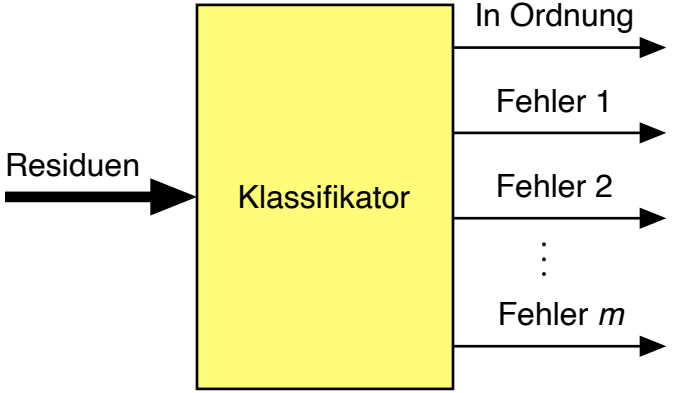


## Residuum als Fehlerindikator

- **Residuen ≈ 0** in Fall „**In Ordnung**“ und **|Residuen| >> 0** in Fall „**Fehler**“
- Ein Residuum ist virtuell (Rechengröße) und die gewünschten Eigenschaften entsprechen denen eines Sensors
  → Möglichst **empfindlich** auf **Fehler**, möglichst **unempfindlich** bezüglich **allem anderen**
- Bei mehreren, unterschiedlichen Fehlern: Gewünscht: Entkopplung der Residuen, d.h. im Idealfall
  Jedes **Residuum** zeigt **einen Fehler** an (hohe Empfindlichkeit) und ist gegenüber allen **anderen Fehlern unempfindlich**
  → Klassifikation wird trivial, da direkt von Residuuen auf die Fehler geschlossen werden kann
- Ansonsten: Ein **Klassifikator** kann oft auch **gekoppelte Residuen trennen**: Jedes Residuum = Eingang/Feature

## Klassifikation basierend auf den Residuen

- Für Klassifikatoren siehe Kapitel 2 und folgende...

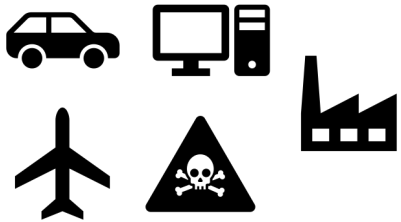



## Der Weg von den Residuen zu den Fehlern kann auch komplexer sein

- Klassifikator braucht **zusätzliche** Eingänge, wie Arbeitspunkte, Umweltbedingungen oder statistische Kennwerte
- **Mehrstufiges** Vorgehen, z.B.:
  1. Erkennung ob „In Ordnung“ oder „Fehler“
  2. Falls Fehler erkannt: Sammlung zusätzlicher Informationen, z.B. durch Testsignal oder Datenbankabfrage
  3. Erkennung der Art des Fehlers: Welcher Fehler?
  4. Diagnose der Fehlerdetails, wie Größe, Ort, Ursache
  5. Was dann? Welche Maßnahmen werden ergriffen? Mehrere alternative Möglichkeiten:
  - **sicheren Zustand** erreichen, wie z.B. Abschalten
  - auf **Notbetrieb** umschalten, wie nur noch halbe Kraft fahren oder Geschwindigkeit reduzieren
  - **Backup**-Systeme aktivieren, wie z.B. adaptiven Regler durch einfachen fixen PI-Regler ablösen
  - **rekonfigurieren**: Route umplanen, Umschalten auf eine andere Regelgröße, Strom mit Kurbel erzeugen, wenn Batterie leer

## Sicherer Zustand

- Oft: **System herunterfahren**: Auto anhalten, Produktion herunterfahren, Computer abschalten
- ABER das kann auch gefährlich oder sicherheitskritisch sein: Flugzeug, Chemiefabrik, Kernkraftwerk
  → Hier muss ein zuvor **wohldefinierter sicherer Zustand** erreicht werden!



## Symptome

- Wenn es nur **ein Residuum** gibt, ergibt sich das Symptom meist daraus. Es kann aber noch „aufbereitet“ werden, um robuster einen Fehler anzuzeigen, z.B. durch
  - Filterung,
  - Nichtlineare Transformation,
  - Transformation in den Frequenzbereich, ...
- Wenn es **mehrere Residuen** gibt, ergeben sich durch deren **Kombination** bzw. **Verknüpfung** eine Vielzahl neuer Möglichkeiten:
  - Tabelle mit **diskrete** Einträgen „+“, „-“, „0“, die in klassischen Regeln verknüpft werden, wie
    **WENN** $r_1$ = „-“ UND $r_2$ = „-“ UND $r_3$ = „+“ UND $r_4$ = „+“
    **DANN** Fehler „Druckleitung verstopft“
  - Leistungsfähiger (aber auch komplexer) ist die Formulierung in Form von Fuzzy-Regeln, wo die Größe der Residuen mit Zugehörigkeitsfunktionen **stetig/kontinuierlich** bewertet wird.
  - diese Regeln können mit Daten **feingetunt** werden
  - zahlreiche **datenbasierte** Klassifikatoren, siehe restliche Vorlesung...

Symptom 1

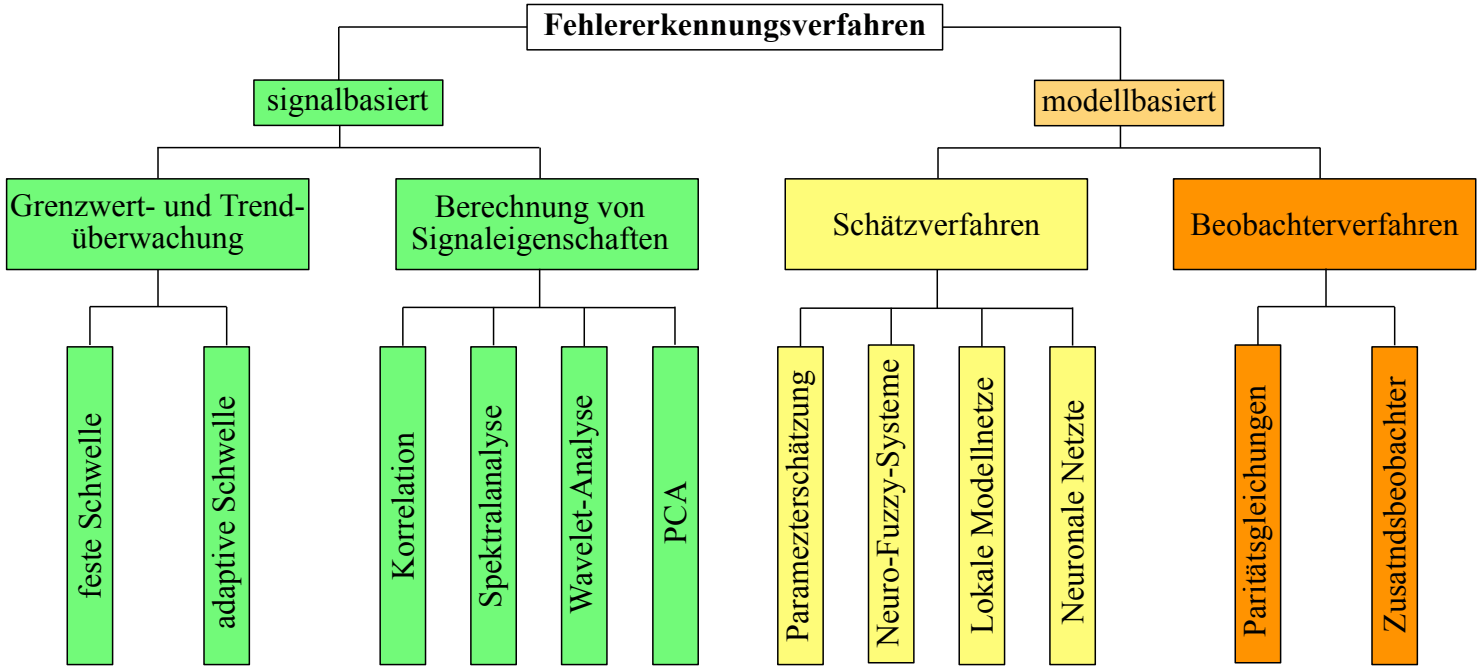
**Modellgestützte Fehlerdiagnose an einer hydraulischen Servoanlage**

| | $r_1$ | $r_2$ | $r_3$ | $r_4$ | $r_5$ |
|---|---|---|---|---|---|
| Druckleitung verstopft | - | - | + | + | 0 |
| Rückleitung verstopft | + | + | - | - | + |
| Steuerkante A-T erodiert | - | - | 0 | 0 | - |
| Steuerkante P-A erodiert | - | - | 0 | 0 | 0 |
| Steuerkante P-B erodiert | 0 | 0 | + | + | 0 |
| Steuerkante B-T erodiert | 0 | 0 | + | + | - |
| Riefenbildung Ventilschieber | 0 | 0 | - | - | - |
| Innere Leckage | + | + | - | - | 0 |

Quelle: Rolf Isermann: „Modellbasierte Überwachung und Fehlerdiagnose von kontinuierlichen technischen Prozessen“, at – Automatisierungstechnik, 6/2010

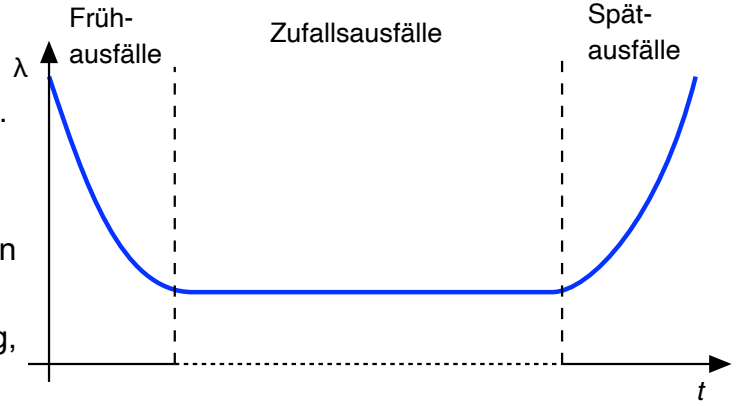
## Signalbasierte und modellbasierte Ansätze zur Fehlerdiagnose

- **Signalbasierte** Methoden sind meist **einfach** und sehr weit **verbreitet**
- **Modellbasierte** Methoden sind meist **komplex**, benötigen mehr **Expertise**, sind dafür aber **leistungsfähiger**



## Lebensdauer

- Die **Ausfallrate λ(*t*)** (*failure rate*) ist die Wahrscheinlichkeit, dass eine zum Zeitpunkt *t* noch nicht ausgefallene Baugruppe bis zum Zeitpunkt *t*+d*t* ausfällt.
- Die **Badewannenkurve** zeigt eine typischen Verlauf der Ausfallrate λ(*t*) eines technischen Produkts:
  - Frühphase: Überdurchschnittlich viele Ausfälle in der Inbetriebnahme wegen ungenügender Qualitätssicherung und Produkttestung.
  - Spätphase: Überdurchschnittlich viele Ausfälle durch Verschleiß, Ermüdung, Alterung.
  - Nutzungsphase: Kontante, niedrige Ausfallsrate bestimmt vom Zufall durch statistische Überlagerung vieler voneinander unabhängiger Faktoren.




- In der Nutzungsphase ist die Ausfallrate ungefähr konstant, deren Kehrwert bezeichnet man als die **mittlere Lebensdauer** bzw. **Mean Time To Failure (MTTF)**:

$$\mathrm{MTTF} = \int_0^\infty \underbrace{\mathrm{e}^{-\lambda t}}_{R(t)}\, dt = \frac{1}{\lambda}$$

- Die **Überlebenswahrscheinlichkeit *R*(*t*)** ist anfangs (*t* = 0) am größten und fällt dann exponentiell → 0 mit „Zeitkonstante“ 1/λ ab.

Überlebenswahrscheinlichkeit oder Zuverlässigkeitsfunktion
Die Annahme statistischer Unabhängigkeit der Ausfälle führt auf eine Poisson-Verteilung → e-Funktion.

Quelle: J. Lienig, H. Brümmer: „Elektronische Gerätetechnik“, Springer Vieweg, 2014

## Typische MTTF- und $\lambda$-Werte

| | MTTF [Jahre] | λ [1/h] |
|---|---|---|
| Stellglied / Aktor | 4,4 | $26 \cdot 10^{-6}$ |
| Pumpe | 2,5 | $44 \cdot 10^{-6}$ |
| Ventil | 13 | $9 \cdot 10^{-6}$ |
| E-Motor | 13 | $9 \cdot 10^{-6}$ |
| Kugellager | 71 | $1{,}6 \cdot 10^{-6}$ |
| Gleitlager | 48 | $2{,}4 \cdot 10^{-6}$ |
| Bürsten (für E-Motor) | 13 | $9 \cdot 10^{-6}$ |
| Riemen | 6 | $20 \cdot 10^{-6}$ |
| Kabel | 114 | $1 \cdot 10^{-6}$ |
| Drucktastenschalter | 11 | $10 \cdot 10^{-6}$ |
| Transistor | 11.000 | $10 \cdot 10^{-9}$ ← -9 |
| Mensch | 73 | $1{,}5 \cdot 10^{-6}$ |

## Redundanz

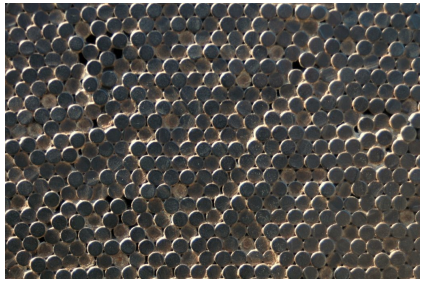
lat. redundare = im Überfluss vorhanden sein

Quelle: https://de.wikipedia.org/wiki/Redundanz_(Technik)

- Das **zusätzliche** Vorhandensein funktional gleicher oder vergleichbarer Ressourcen eines technischen Systems
- **Zweck**: Erhöhung der Ausfall-, Funktions- und Betriebssicherheit
- Untergliederung der Redundanzauslegung:
  - **Heiße Redundanz**: Mehrere Teilsystem arbeiten parallel
  - **Kalte Redundanz** oder **Standby-Redundanz** (passive Redundanz): Mehrere Teilsysteme existieren parallel, aber nur eines arbeitet – im Fehlerfall wird ein anderes aktiviert
  - **(*n*+1)-Redundanz**: Das Gesamtsystem besteht aus *n* Einheiten; es existiert eine zusätzliche Einheit. Fällt eine Einheit aus, kann diese zusätzliche Einheit übernehmen. Fällt eine weitere Einheit aus, steht das System nicht mehr voll zur Verfügung.

## Beispiele für Redundanz:

- Stahlseil: Besteht aus vielen Litzen →
- Notstromversorgung: Oft existieren mehr Notstromaggregate als notwendig, (*n*+1)-Redundanz ist üblich
- Eurofighter: Verkabelung doppelt (parallel), Computer dreifach (Voting)
- Raid-Systeme: (*n*+1)-Redundanz ist üblich, eine Festplatte kann kaputt gehen
- Produktion: Mehrere Zulieferer oder zumindest mehrere Standorte

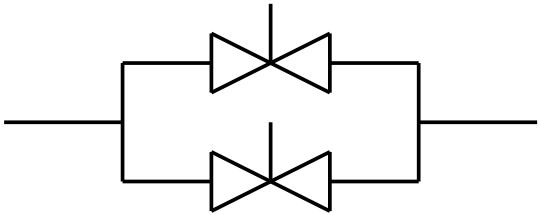


Quelle: https://de.wikipedia.org/wiki/Redundanz_(Technik)#/media/Datei:Closeup_of_wires_inside_cable_on_Golden_Gate_Bridge,_San_Francisco_(2006).jpg

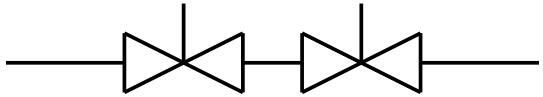
## Parallel oder in Serie?

- Beispiel: Flüssigkeitstransport
- **Parallel**: Nur eine Komponente muss funktionieren
- **Serie**: Alle Komponenten müssen funktionieren
- Was ist der sicher Zustand?
  - Kühlkreislauf: Geöffnet → Rohre und Ventile parallel
  - Durchleiten eines Gefahrenstoffs: Geschlossen → Ventile in Reihe

parallel (ODER)



in Serie (UND)



## Art der Teilsysteme:

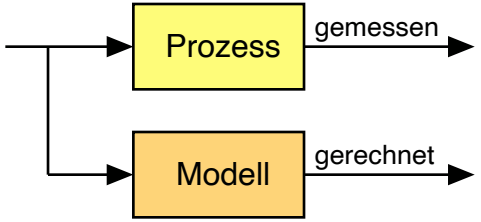
- Identische Ausführung (**homogene Redundanz**), z.B. viele gleiche Sensoren oder Bauteile
- Unterschiedliche Ausführungen (**diversitären Redundanz**), z.B. Sensoren oder Bauteile von
  - unterschiedlichen **Herstellern**
  - basierend auf unterschiedlichen **physikalischen** Prinzipien
  - basierend auf **Software**, evtl. sogar in verschiedenen Implementierungen von verschiedenen Programmier-Teams

Physikalische Redundanz

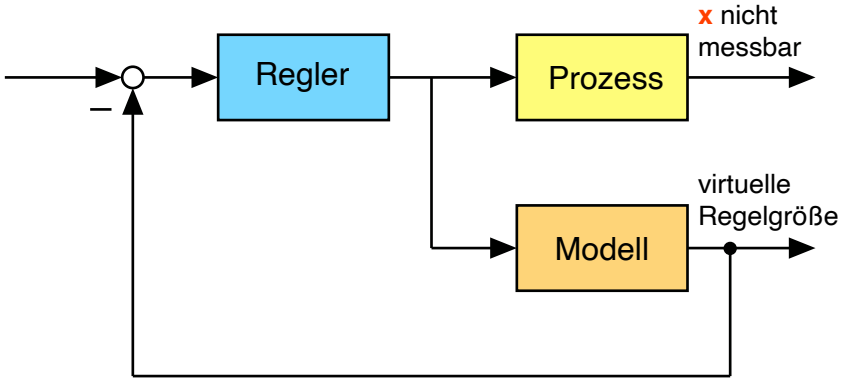
Analytische Redundanz

Das gilt nicht nur in der Technik. Es ist nachgewiesen, dass diverse Teams in der Arbeitswelt eine größere Produktivität aufweisen. Daher der Trend um das Thema **Diversity**.

## Virtueller Sensor (*soft sensor*)




- Modell bzw. Beobachter wird zur Rekonstruktion einer Variablen verwendet
- Zwei Möglichkeiten
  - Gleichzeitige Messung und Berechnung (über Modell oder Beobachter) einer Variablen
    → **Analytische Redundanz**
  - **Einsparung des Sensors** im Betrieb. Statt dessen wird der rekonstruierte Wert verwendet. Zur Erstellung des Modells oder Beobachters müssen Messungen herangezogen werden, z.B. am Prüfstand gemessen
    - Beispiel 1: Drehmoment oder Abgaswerte bei Verbrennungsmotoren
    - Beispiel 2: Temperator des Rotors bei Elektromotoren
    Das sind sehr wichtige Größen, die in Serie nicht gemessen werden können, aber für den Betrieb entscheidend sind.
- Vorteile:
  - Entweder Redundanz oder Einsparung
  - Oft Messung nicht möglich wegen
    - Unzugänglichkeit
    - feindlichen Umgebungsbedingungen (zu heiß oder kalt, zu hoher Druck, zu sauer, Funkenflug, Vibrationen, ...)
  - Regelung auf rekonstruierte Variable möglich: Ist **konzeptionell eine Regelung**. Da die Regelgröße aber nicht gemessen wird, ist es faktisch ein „Mittelding“ zwischen Regelung und Steuerung



## Herausforderungen für die Fehlerdiagnose

| Fehlertypen | Anforderung an Erkennung |
|---|---|
| abrupt (sprungartig) | schnell (Zeit) |
| kriechend (Drift, rampenartig) | empfindlich (Amplitude) |
| kein Fehler | weder zu schnell noch zu empfindlich |

genug Abtastwerte

robust bzgl. Rauschen

- Leistungsfähige Ansätze benötigen **Modelle**
- **Theoretische Modelle** (aus first principles) sind oft
  - komplex
  - teuer
  - rechenaufwändig
  - enthalten viele unbekannte Parameter, die schwer aus Datenblättern oder mit Identifikationsverfahren zu ermitteln sind
- **Datengetriebene Modelle** (mittels Machine Learning) leiden häufig unter
  - zu wenige Daten, insb. für die Fehlerfälle
  - unbalancierte Daten, d.h. viele mehr Daten für den „In Ordnung“-Fall als für die Fehlerfälle
  - Unzuverlässigkeit der Modelle, insb. im Extrapolationsbereich
- In der **Praxis** werden die meisten Fehlererkennungs- und Fehlerdiagnosesysteme **deaktiviert**, da sie zuviele **Fehlalarme** produzieren. Das folgende Dilemma ist zumindest teilweise daran schuld:
  - Der Hersteller des Fehlerdiagnosesystems ist für dessen Wirksamkeit verantwortlich, d.h. er will vermeiden, dass sein System Fehler nicht findet. Daher stellt er die Detektionsschwelle sehr niedrig (empfindlich) ein, was zu vielen Fehlalarmen führt.
  - Dem Kunden entstehen durch die Fehlalarme so hohe Kosten, dass das Fehlerdiagnosesystem destruktiv wirkt und deaktiviert wird.

## Schwelle oder Grenze (*threshold*)

- Sehr **einfach** und weit **verbreitet**
- Ein Signal $x(t)$ wird mit einer **Schwelle *T*** verglichen und ein Fehler erkannt, wenn
  - Überschreitung: $x(t) > T$
  - Unterschreitung: $x(t) < T$
  - Toleranzverletzung: $x(t) < T_{min}$ oder $x(t) > T_{max}$

Oder mit Residuum: $r(t) = x(t) - T$
$r(t) > 0$
$r(t) < 0$
$|r(t)| > 0$

- Da das Signal $x(t)$ oft **verrauscht** ist, kann es zu **ungewollten Fehlalarmen** kommen. Je stärker das Rauschen, umso häufiger
- Abhilfe:
  - Schwelle „großzügiger“ einstellen.
    → Empfindlichkeit bzgl. des Fehlers wird reduziert; im Extremfall wird der Fehler nicht mehr erkannt.
  - Signal $x(t)$ filtern (typischerweise mit einem Tiefpass), um Rauschen zu reduzieren.
    → Signal wird **verzögert** (durch Phasenverschiebung des Filters), umso stärker je langsamer der Filter eingestellt ist.
    → Alarm kommt evtl. zu spät.

**Generell gilt: Je stärker eine Tiefpassfilterung, desto mehr Verzögerung des Signals**

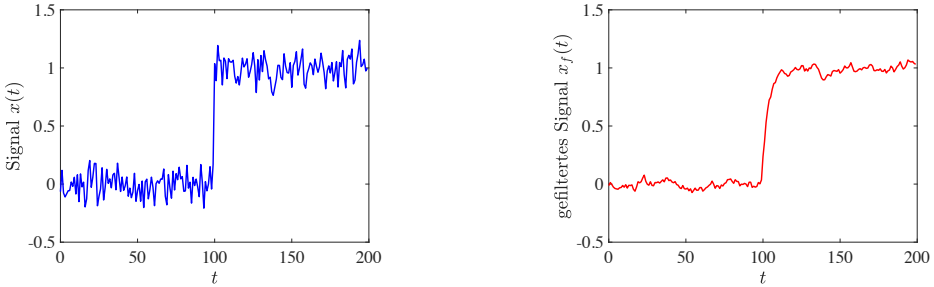
- Finden eines guten Tradeoffs zwischen beiden Zielen (**gute Rauschunterdrückung**, **wenig Verzögerung**) ist eine wichtige und häufige Ingenieuraufgabe!

## Effekt einer Tiefpassfilterung eines Signals

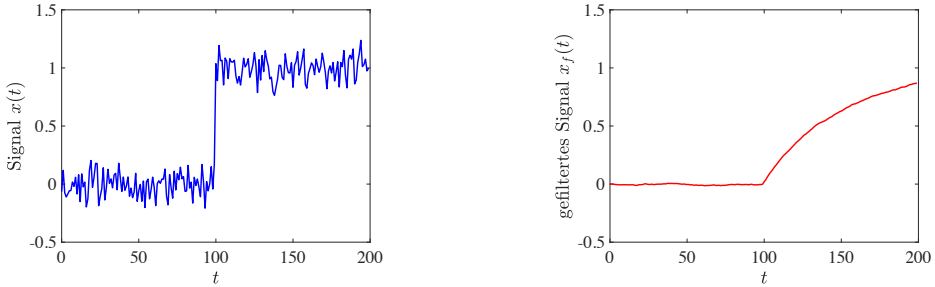
- Je stärker das Rauschen ist, desto „stärker“ möchte man den Filter einstellen.
- 2 Möglichkeiten für das Tuning
  - **Grenzfrequenz** $f_g$ bzw. Zeitkonstante des Filters. All Frequenzen darüber $f > f_g$ sollen weggefiltert werden
  - **Ordnung** und damit Flankensteilheit, d.h. wie scharf von Durchlass- in Sperrbereich übergegangen wird

  → **Beides** (Reduktion der Grenzfrequenz und Erhöhung der Ordnung) **erhöht** den **Phasenverzug**
- **Fehler** geschehen oft **abrupt** (sprunghaft) → **Schnelle Erkennung** ist **wichtig**!

### Schnelles Filter (großes $f_g$)




### Langsames Filter (niedriges $f_g$)



**Viel Dynamik → Modelle ungenau:**
- vernachlässigte Dynamik, d.h. Modellordnung < Prozessordnung → Amplituden- und Phasenfehler
- vernachlässigte Totzeiten → Phasenfehler

## Adaptive Schwelle oder Grenze (*adaptive threshold*)

- Statt einen **starren Tradeoff** zwischen guter Rauschunterdrückung (*starke Filterung*) und schneller Diagnose (*schwache Filterung*) einmal einzustellen, kann man diesen **Tradeoff situationsabhängig** machen!
- Bekannt aus der **Bildverarbeitung**, wo die Schwelle **lokal unterschiedlich** gewählt wird, z.B. je nach mittlerer Helligkeit der entsprechenden Bildregion.
- Hier unterscheiden wir die Situationen nicht anhand der örtlichen Gegebenheiten sondern anhand des **zeitlichen** Verlaufs:

  1. **Wenig Dynamik**: alle (relevanten) Signale sind halbwegs eingeschwungen oder fahren Strich
     → Modelle sind relativ genau. Unsicherheiten sind relativ gering → **Schwelle empfindlich** einstellen!
  2. **Viel Dynamik**: der Prozess ist in einem transienten Zustand, z.B. Übergang von einem Arbeitspunkt zu einem anderen
     → Modelle sind relativ ungenau. Unsicherheiten sind relativ groß → **Schwelle unempfindlich** einstellen!

- Toleranzverletzung bei fester Schwelle: $|r(t)| > T_{fix}$
- Toleranzverletzung bei adaptiver Schwelle: $|r(t)| > T_{fix} + c \cdot \sigma(t)$
  mit einer Tuning-Konstanten $c$
  und mit der Standardabweichung bzw. der Volatilität des Residuums $\sigma(t)$. Wenn alles **Strich fährt**: $\boldsymbol{\sigma(t) \rightarrow 0}$

Viele Möglichkeiten, eine adaptive Schwelle zu realisieren!

- Je mehr das Residuum „**zappelt**“, desto **großzügiger** wird die Fehlererkennung.
- Die **Standardabweichung** kann **rekursiv** aus der Standardabweichung im Zeitschritt davor $\sigma(k\text{-}1)$ berechnet werden:

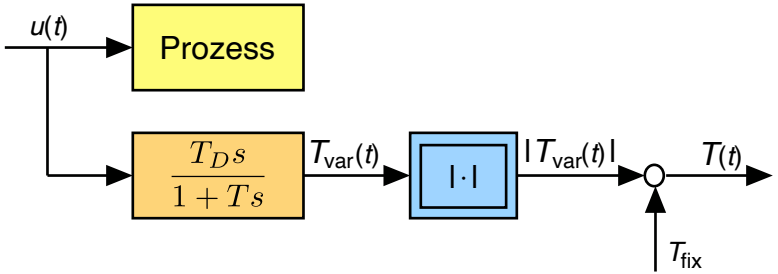
  Für diskrete Zeitschritte $t = kT_0$: $\sigma^2(k) = \frac{1}{k}\left[(k-1)\sigma^2(k-1) + (r(k) - \bar{r}(k))^2\right]$

Mittelwert des Residuums

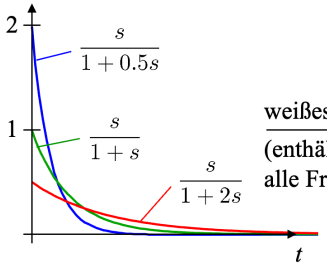
## Alternative Generierung: Adaptive Schwelle oder Grenze (*adaptive threshold*)

- Die adaptive Schwelle kann auch aus der Summe eines fixen, konstanten Teils $T_{fix}$ und eines variablen Teils $T_{var}$ erzeugt werden: $T = T_{fix} + T_{var}$
- Der variable Teil kann z.B. von Eingangssignal des Prozesses abhängig gemacht werden:
  - Eingang = const. → variable Schwelle $T_{var} = 0$
  - Eingang = ändert sich → variable Schwelle $T_{var}$ ~ Änderungsgeschwindigkeit des Eingangs
- Die Änderungsgeschwindigkeit könnte man durch ein D-Glied (Ableitung) erzeugen.
  Aus Robustheitsgründen (Unempfindlichkeit bzgl. Rauschen) verwendet man typischerweise ein **DT$_1$-Glied**: $\frac{T_D s}{1+Ts}$
  Das entspricht einer Ableitung mit anschließender Tiefpassfilterung mit einem PT$_1$ mit Grenzfrequenz $f_g = 1/T$

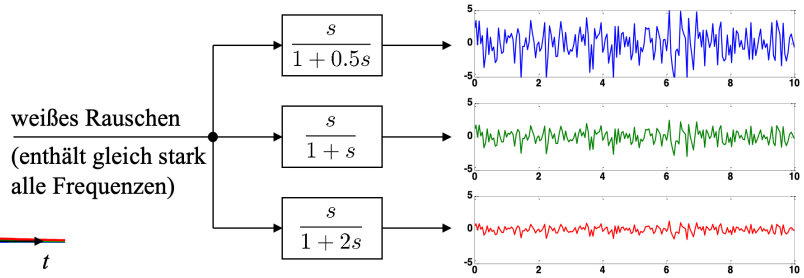
### Adaptive Schwelle




**DT$_1$-Sprungantworten:**




**Rauschverstärkung:**



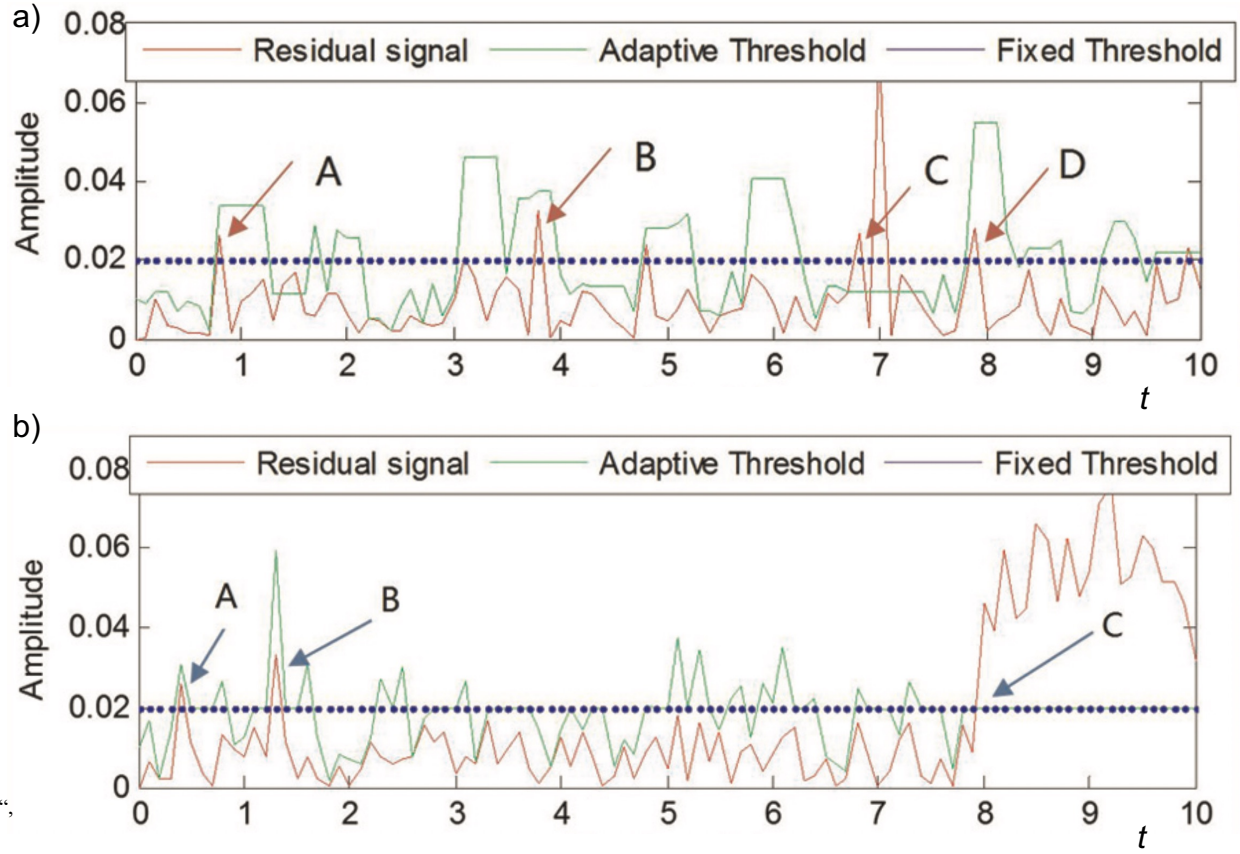
## Beispiel: Roboter

### a) Abrupter Fehler bei C, $t = 7\,s$

- Feste Schwelle würde bei A, B, C, D Fehler erkennen, d.h. 3 Fehlalarme produzieren
- Adaptive Schwelle erkennt den echten Fehler bei C und löst keine Fehlalarme aus

### b) Drift-Fehler bei C, $t = 8\,s$

- Feste Schwelle würde bei A, B, C Fehler erkennen, d.h. 2 Fehlalarme produzieren
- Adaptive Schwelle erkennt den echten Fehler bei C und löst keine Fehlalarme aus




Quelle: Lifeng Wu, Beibei Yao, Zhen Peng and Yong Guan:
„An adaptive threshold algorithm for sensor fault based on the grey theory “,
Advances in Mechanical Engineering, Vol. 9(2) 1–7, 2017

## Ausnutzung von Signaleigenschaften

- Wenn der Eingangsraum (*feature space*) **niedrigdimensional** ist (1D, 2D oder (bedingt) 3D) kann man sich die Verteilung der Datenpunkte **visualisieren** und die Klassen „In Ordnung“ und „Fehler *i*“, *i* = 1, ..., *m*, **trennen**
- Für **höherdimensionale Eingangsräume** (mehr als 3 Features) kann
  - **Hauptkomponentenanalyse**, evtl. darauf folgender mit Dimensionsreduktion (PCA)
  - **Clustering** mit Analyse der Verteilung der Clusterzentren und -ausdehnungen

  siehe Nelles: „Signalverarbeitung“
- Die **Frequenzeigenschaften** der Signale enthalten offensichtliche Abhängigkeiten vom Fehler / den Fehlern.
  Da sich diese Eigenschaften mit Fehlereintritt ändern (abrupt oder kriechend), darf nicht über die gesamte Zeit integriert werden, sondern es müssen Zeitintervalle/bereiche analysiert werden. Dies ist z.B. möglich mit:
  - **Kurzzeit-DFT**
  - **Wavelet-Analyse**

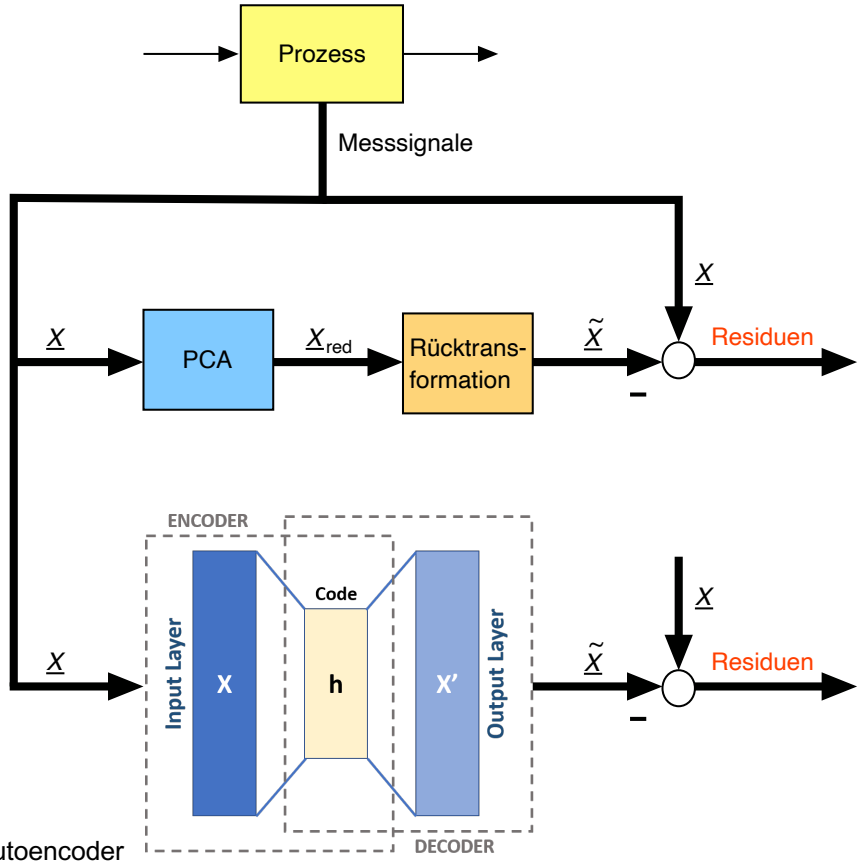
  siehe Nelles: „Signalverarbeitung“ und Kraemer: „Condition Monitoring“
- Die Berechnung und Analyse von **Korrelationen** ist sehr robust bzgl. Rauschen und die Qualität verbessert sich mit $1/\sqrt{N}$ für *N* Datenpunkte. Die korrelierten Signale können sein: direkte Messgrößen, Residuen oder daraus berechnete Symptome. Hilfreich zur Fehlererkennung und -diagnose:
  - lineare Kreuzkorrelationen bzw. Korrelationsmatrizen (multivariat), d.h. jedes x jedes

  siehe Nelles: „Signalverarbeitung“ und Kraemer: „Condition Monitoring“
  - lineare Auto- und Kreuzkorrelations*funktionen*, d.h. Korrelationen mit **zeitlich verschobenen** Signalen, z.B. $x(t)$ mit $x(t+\tau)$ oder $y(t+\tau)$ bzw. $x(k)$ mit $x(k+\kappa)$ oder $y(k+\kappa)$
  - nichtlineare Korrelationen über Faltungsnetze (*convolution neural networks, CNN*)

## Beispiel PCA (linear) oder auch Autoencoder (nichtlinear)

- Es gibt viele Möglichkeiten eine PCA oder deren nichtlineare Erweiterung einen Autoencoder zu Diagnosezwecken einzusetzen.

1. Signale, die auf Fehler empfindlich sind, werden spaltenweise in der Matrix $\underline{X}$ gesammelt.
2. Eine PCA transformiert (jedes Signal = eine Dimension) in ein neues Koordinatensystem mit Achsen absteigender Datenstreuung je Achse (gegeben durch die Singulärwerte)
3. Dimensionsreduktion: Die vermutlich unwichtigen Achsen (mit kleinen Singulärwerten) werden eliminiert.
   Hoffnung: Diese Achsen enthalten keine wesentliche Information.
4. Rücktransformation: Aus den ersten Achsen (principal components) der PCA wird die Matrix $\underline{X}$ zu $\underline{\tilde{X}}$ rekonstruiert. Der Autoencoder macht dies aus dem „Code h“ (Flaschenhals), so gut wie möglich.
5. Im fehlerfreien Fall ist $\underline{\tilde{X}} \approx \underline{X}$ und das Residuum $\approx \underline{0}$ (multivariat: so viele Dimensionen, wie Spalten in $\underline{X}$).
   Im Fehler-Fall klappt die Rekonstruktion der Signale nicht und das Residuum wird betragsmäßig groß – für jeden Fehler hoffentlich mit einer anderen Signatur.



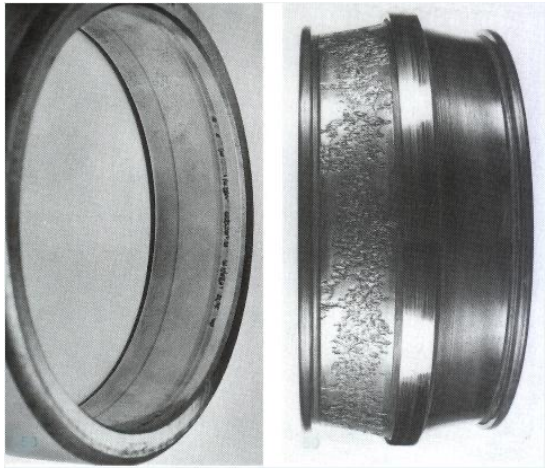

Quelle für Autoencoder: https://en.wikipedia.org/wiki/Autoencoder

## Beispiel für FFT

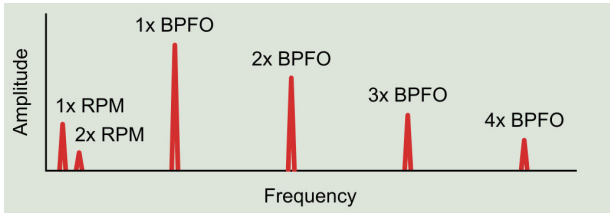
Quelle: https://power-mi.com/content/typical-bearing-defects-and-spectral-identification

- Fehlererkennung und -diagnose in Kugel- und Rollenlagern
- FFT des Drehzahlsignals zeigt eindeutige Fehlersignaturen. Diese unterscheiden sich deutlich voneinander je nach Fehlerart
- RPM = Lagerdrehzahl
  BPFO = Ball Pass Frequency Outer
  BPFI = Ball Pass Frequency Inner

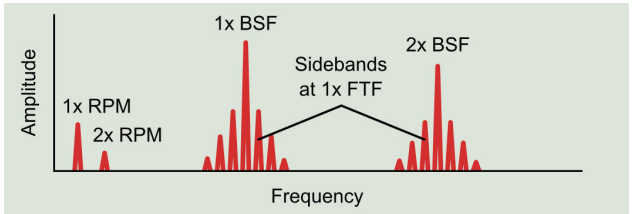
**„Fehler im Außenring“** **„Fehler im Innenring“**



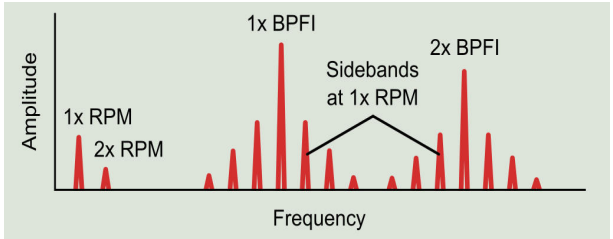
**„Fehler im Außenring“**



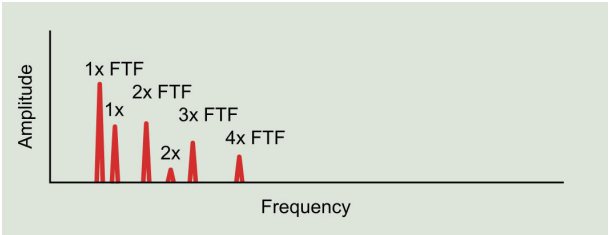

**„Fehler in der Kugel oder Rolle“**




**„Fehler im Innenring“**



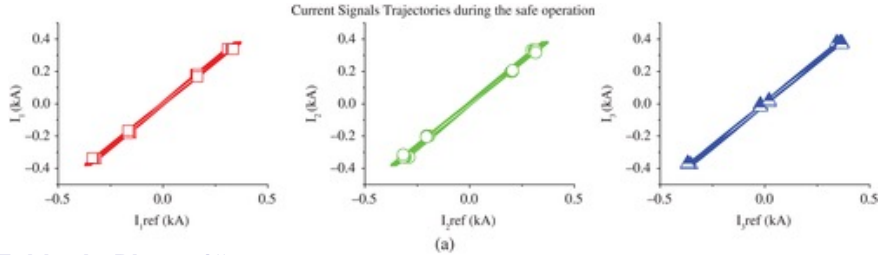

**„Fehler im Käfig“**



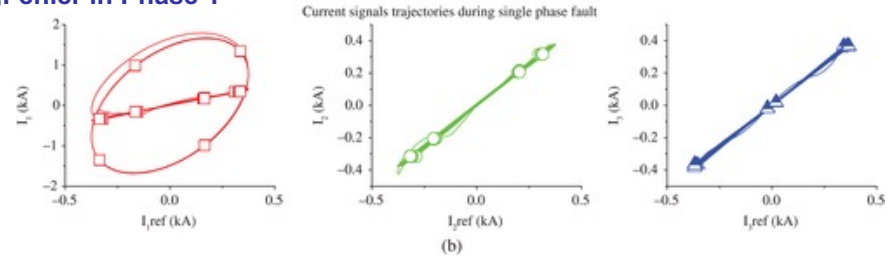
## Beispiel für Korrelation

- Fehlererkennung und -diagnose in Hochspannungsleitungen
- Mittels Korrelationskoeffizient zwischen Strommessung in jeder Phase und einem Referenzstromwert

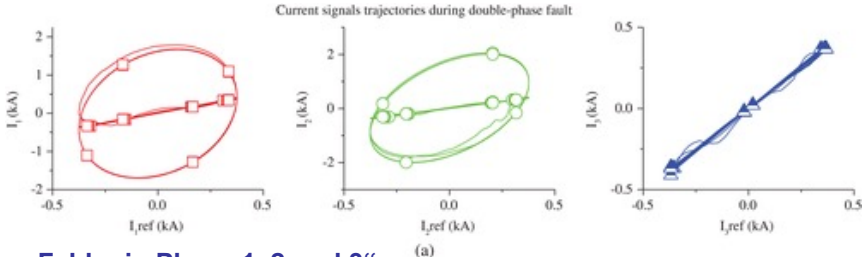
„In Ordnung“



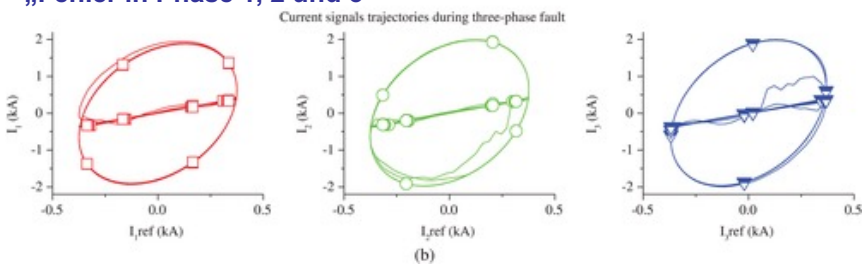

„Fehler in Phase 1“




„Fehler in Phase 1 und 2“




„Fehler in Phase 1, 2 und 3“

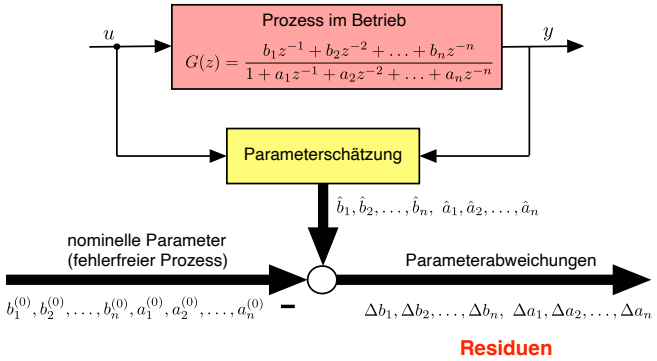



Quelle: https://onlinelibrary.wiley.com/doi/10.1002/tee.22705

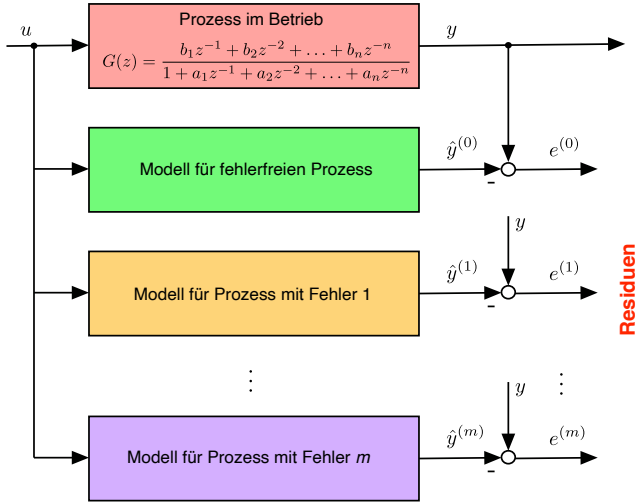
## Vergleich modellbasierte Ansätze

- Hier: Für **lineare** Modelle
- Erweiterung auf **nichtlineare** Modelle ist für **Paritätsgleichungen relativ einfach möglich**, obwohl nichtlineare dynamische Modelle meist sehr komplex sind; für **Parameterschätzung** und **Beobachtung** sind die Erweiterung auf nichtlineare Ansätze typischerweise **sehr aufwändig** und nur **strukturreinschränkend** möglich.

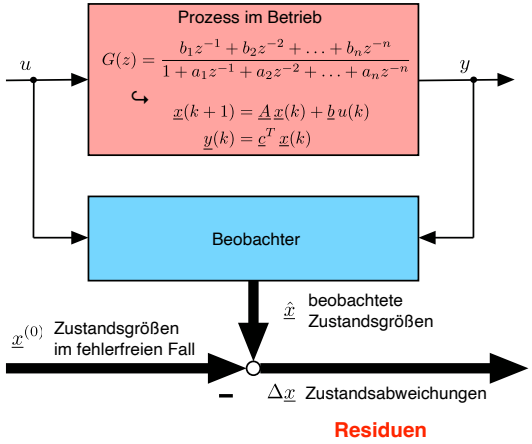
### Parameterschätzung




### Paritätsgleichungen



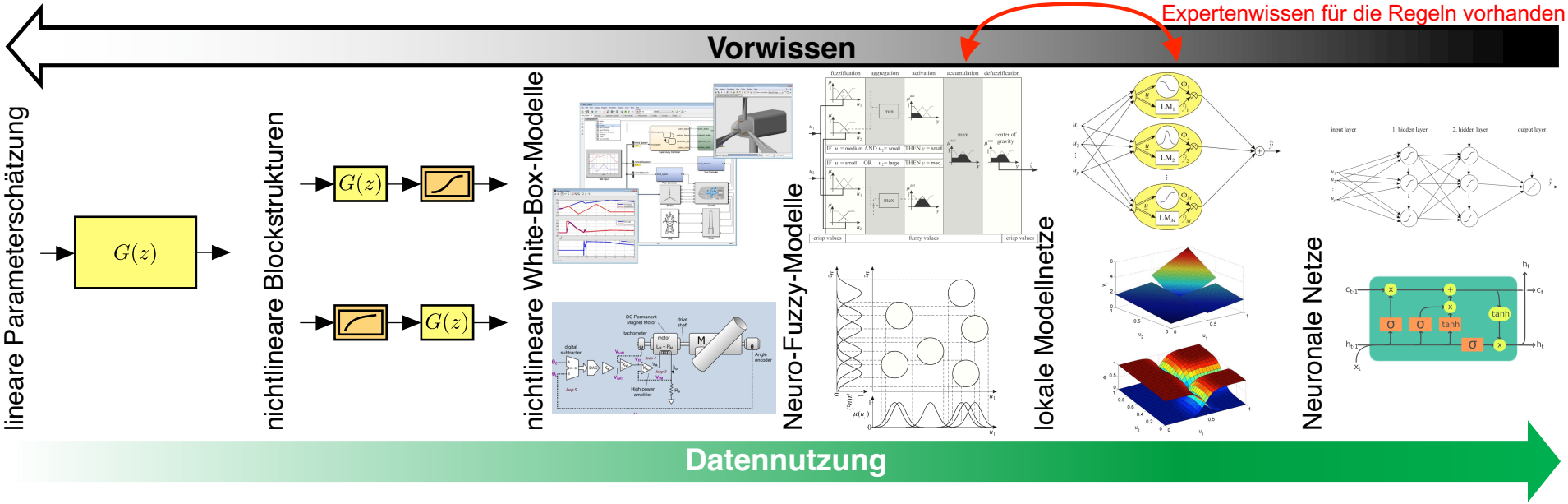

### Zustandsbeobachter



## Residuen basierend auf Parametern → Residuen basierend auf Signalen

- Bei modellbasierten Fehlererkennungsverfahren mit **Parameterschätzung** basieren die Residuen auf Abweichung der Parameterwerte, d.h. der **Differenz** der **nominellen** und **fehlerhaften Prozessparameter**
- Bei **linearen** Prozessen und Modellen ist dies relativ **leicht umsetzbar**; bei nichtlinearen Prozessen wird es deutlich anspruchsvoller, siehe Bild und nächste Folie für verschiedene Klassen von Modellansätzen

Reihenfolge dreht sich um, wenn wenig Expertenwissen für die Regeln vorhanden



## Modellierung nichtlinearer dynamischer Prozesse

- Sehr anspruchsvolle Aufgabe. Typischer Herangehensweisen sind:
- **Lineares Modell**
  - ändert sich je nach Betriebszustand, Arbeitspunkt, Umgebungsbedingungen
  - Trennung zwischen Fehlerereignis und „kein Fehler“ aber schlechtem Modell wegen Plant-Model-Mismatch **kaum möglich**
- **Nichtlineares White-Box-Modell**
  - typischerweise: Struktur aus First Principles (Physik, Chemie, ...), Parameter aus Datenblättern oder Messdaten
  - Parameterschätzung und Residuengenerierung **ähnlich wie im linearen Fall**, nur mittels nichtlinearer Optimierungsverfahren

- **Lokale Modellnetze (Gray-Box)**
  - Interpolation oder Umschalten zwischen arbeitspunktabhängigen **linearen** Modellen
  - eingeschränkt können die Parameter interpretiert werden – daher **ähnlich wie im linearen Fall**

- **Neuro-Fuzzy-Systeme (Gray-Box)**
  - basieren auf Regeln, die aus Expertenwissen kommen und/oder aus Messdaten identifiziert werden
  - stark eingeschränkt können die Parameter interpretiert werden – daher **ähnlich wie im linearen Fall**

- **Neuronale Netze und Deep Learning (Black-Box)**
  - **keine Interpretation** der Parameter möglich → Modellausgang verwenden: **signalbasiert**, **Paritätsgleichungen**

## Beispiel Feder-Masse-Dämpfer-System (schwingungsfähiges System 2. Ordnung)

$$\ddot{y}(t) + \underbrace{\frac{d}{m}}_{a_1}\dot{y}(t) + \underbrace{\frac{k}{m}}_{a_0} y(t) = \underbrace{\frac{1}{m}}_{b_0} u(t) \quad \Rightarrow$$

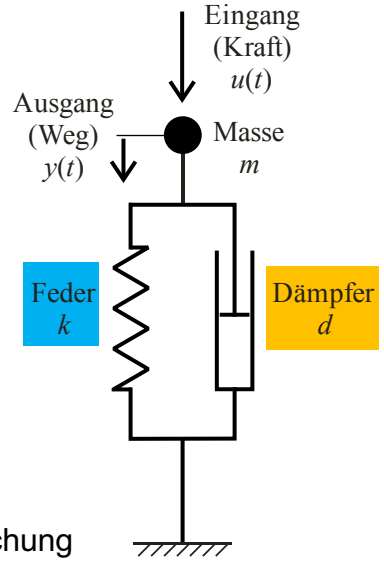
$$\ddot{y}(t) + 2D\omega_0\,\dot{y}(t) + \omega_0^2\,y(t) = K\omega_0^2\,u(t)$$

Die Parameter $a_1$, $a_0$, $b_0$ können aus Messdaten mit linearer Regression (Least Squares) geschätzt werden.

$$K = \frac{1}{k} \qquad D = \frac{d}{2}\sqrt{\frac{1}{mk}} \qquad \omega_D = \frac{1}{2m}\sqrt{4mk - d^2}$$

$$\omega_0 = \sqrt{\frac{k}{m}}$$

$$\omega_D = \omega_0\sqrt{1 - D^2}$$

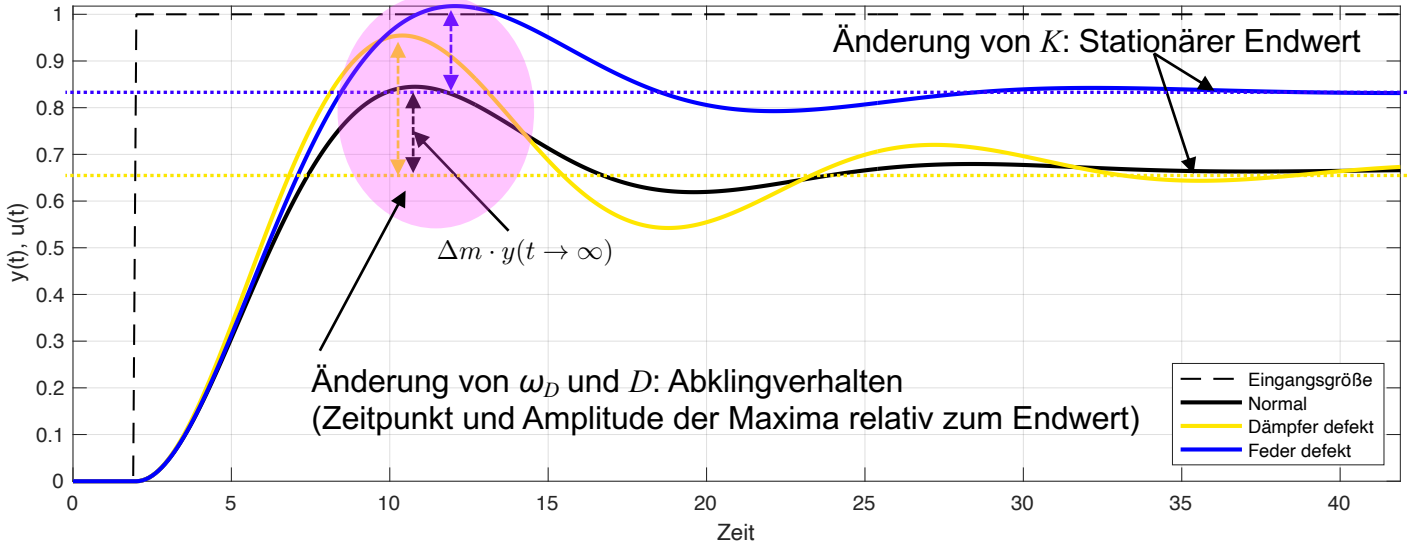



**Schäden an Feder oder Dämpfer führen zur Reduzierung der Federsteifigkeit $k$, bzw. der Dämpferkonstante $d$.**

- Dies ist mit **Parameterschätzverfahren** direkt an der Änderung der Koeffizienten der Differenzialgleichung ($a_1$, $a_0$, $b_0$) erkennbar. Hier sogar unabhängig voneinander ( $k$, und $d$ wirken auf unterschiedliche Koeffizienten).
- Mit Verfahren, die **Residuen** aus dem Modellfehler generieren (z.B. Paritätsgleichungen), ist die Fehlererkennung ebenfalls möglich; die Fehlerunterscheidung aber schwieriger:
  - der Proportionalitätsfaktor $K$ (am stationären Endwert erkennbar) wird nur von der Federsteifigkeit $k$ beeinflusst,
  - die Dämpfung $D$ und Eckfrequenz $\omega_0$ und damit auch die Eigenfrequenz $\omega_D$ jedoch von $k$ und $d$. Die beiden Größen wirken allerdings unterschiedlich: Ein kleineres $d$ reduziert die Dämpfung und erhöht die Eigenfrequenz, bei $k$ ist es umgekehrt.

## Antwort *y*(*t*) auf ein Sprungsignal am Eingang *u*(*t*):

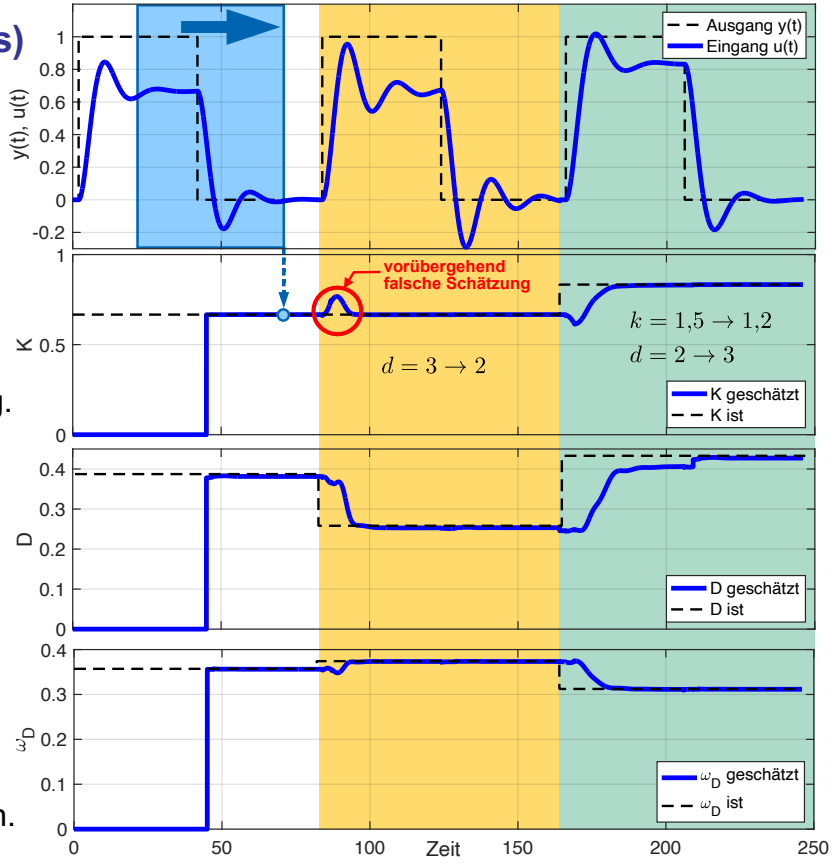
- Normales Verhalten (schwarze Linie) .
- Dämpfer defekt (gelbe Linie): Eigenfrequenz nimmt zu, Dämpfung nimmt ab (Abklingverhalten verschlechtert sich), Proportionalitätsfaktor $K$ (Verstärkung, stationärer Endwert) bleibt gleich.
- Feder defekt (blaue Linie): Proportionalitätsfaktor $K$ ändert sich, Eigenfrequenz nimmt ab, Dämpfung nimmt zu.
- Die Dämpfung kann anhand der *relativen Überschwingweite* Δ*m* bewertet werden.




$$\Delta m = \frac{y_{Max} - y(t \to \infty)}{y(t \to \infty)} = e^{\frac{-D\pi}{\sqrt{1-D^2}}}$$

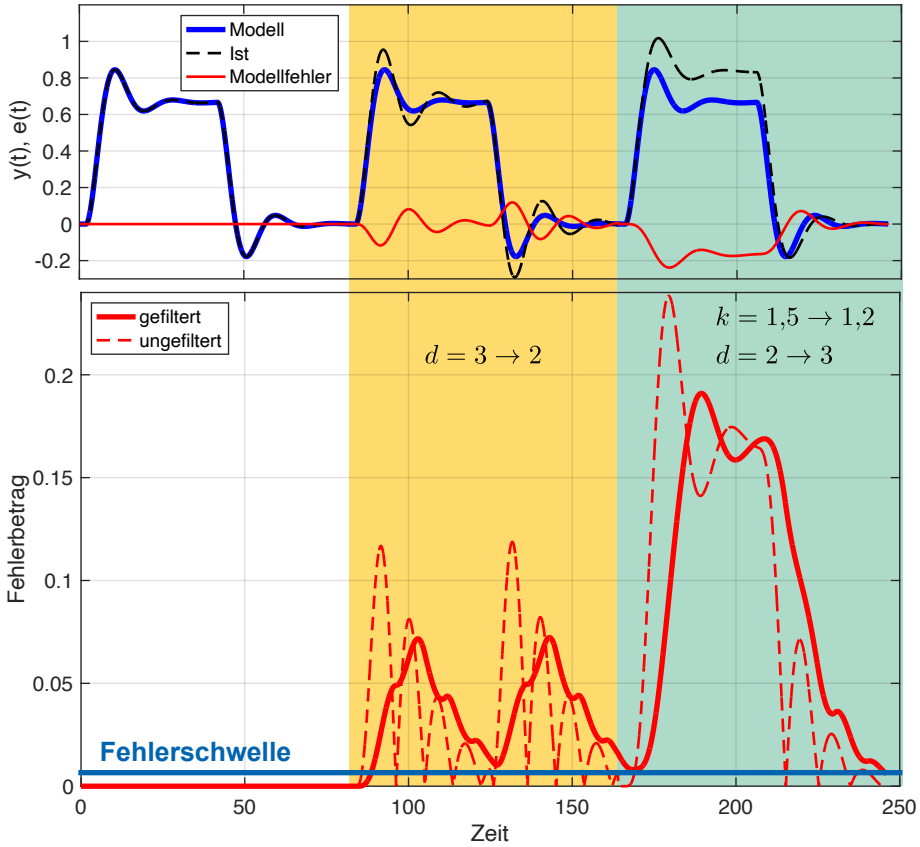
## Fehlererkennung durch *Parameterschätzung* (Least Squares)

- Parameter werden aus einem zurückliegenden Zeitraum geschätzt:
  - Zeitraum darf nicht zu kurz sein, damit genug Information für eine zuverlässige Schätzung vorliegt.
  - Zeitraum darf nicht zu lang sein, damit eine Störung nicht mit zu großer Verzögerung (zu spät!) erkannt wird.
  - Alternative: Rekursiver Least Squares Algorithmus (RLS).
    Aber auch hier ist ein vergleichbarer Kompromiss bei der Wahl des sogenannten Vergessensfaktors (Vergessen alter Information) nötig.
- Qualität der Schätzung von Anregung anhängig:
  - für gute Schätzung des Proportionalitätsfaktors niedrigfrequente (konstante) Signalanteile nötig.
  - für Schätzung von Dämpfung und Eigenfrequenz ständige dynamische Anregung in weitem Frequenzbereich nötig.
  - Beeinträchtigung der Schätzung durch Messrauschen.
  - Mangelnde Anregung kann sowohl bei Least Squares als auch bei Rekursivem Least Squares zu numerischen Problemen führen!
- Aus Parameteränderungen kann Fehler und Fehlerart ermittelt werden.



## Fehlererkennung mit *Paritätsgleichungen*
## Berechnung des Modellfehlers (Residuum):

- Oberes Bild zeigt unverändertes Modell, veränderliches tatsächliches System und den Fehler zwischen beiden.
  - Da Fehler unabhängig vom Vorzeichen unerwünscht sind, empfiehlt es sich z.B. den Fehlerbetrag zu verwenden.
- Schwellen zur Fehlererkennung (fest/variabel)
- Dynamische Fehler vs. stationäre Fehler:
  - stationärer Fehler (verändertes *K*) deutlich erkennbar (dritter Sprung nach oben).
  - wenn System auf Null fällt, ist auch stationärer Fehler nicht mehr erkennbar (rechtes Ende des Zeitverlaufs).
  - dynamische Fehler (verändertes *D*, $\omega_0$) klingen ggf. schnell auf Null ab, wenn die Anregung fehlt.
- Um dynamische Fehler besser zu erkennen, wurde der Fehler im unteren Bild gleitend mittelwertgefiltert und eine Schwelle gewählt, die in diesem Beispiel eine zuverlässige Fehlererkennung ermöglichen würde. Bei längeren Zeiträumen ohne Anregung würden aber dynamische Fehler nicht mehr erkannt!



## Nichtlineare dynamische Modelle

siehe auch Nelles: „Neuronale Netze und Fuzzy-Systeme“

- Zwei Herausforderungen: Dynamik und Nichtlinearität
- Zusätzliche Herausforderung: Finden eines gutem Bias/Varianz-Tradeoffs. Generelle Richtschnur:
  - mehr Daten → komplexere Modelle
  - höhere Datenqualität / weniger Rauschen → komplexere Modelle
  - zwei Möglichkeiten, ein Modell weniger komplex zu machen:
    a) **Struktur abspecken** (weniger Neuronen, Terme, ...) → **weniger Parameter**
    b) **Regularisierung**: Struktur unverändert, **Anzahl der Parameter unverändert**, Optimierung der Parameter wird **reglementiert** durch:
    - „Early Stopping“ bei Training
    - lokale Schätzung
    - sequentielle Schätzung
    - Strafterme (z.B. Ridge Regression) oder Nebenbedingungen
    - Dropout, d.h. zufälliges Weglassen von Neuronen beim Training

d.h. geringere *effektive* Anzahl an Parametern

d.h. nicht alle Parameter gleichzeitig!

- Typischerweise sehr **viele Parameter**
- In **White-Box**-Modellen (Struktur aus First Principles) haben die Parameter eine physikalische Bedeutung. **Parameterabweichung → Residuum**
- In **Black-Box**- und oft auch **Gray-Box**-Modellen: Parameter nicht/wenig interpretierbar → **Residuum** basiert meist auf Modell-**Ausgängen**, nicht -Parametern oder -Zuständen

**Komplexere Modelle:**
- niedriger Bias-Fehler (systematischer Fehler)
- höherer Varianz-Fehler (stochastischer Fehler)
- mehr Parameter
- mehr Rechenaufwand für Training und Nutzung

**Regularisierungsverfahren:**
- durch die sehr komplexen Modelle ist das Hauptproblem der große Varianz-Fehler
- **erhöhen** den **Bias**
- **reduzieren** die **Varianz**
- in Summe sollte die Bilanz positiv sein!

**Beispiel: Ridge Regression**
Verlustfunktion von Least Squares und Ridge Regression (Weight Decay bei NN):

$$J^{(\mathrm{LS})} = \sum_{i=1}^{N} e^2(i) \qquad J^{(\mathrm{Ridge})} = \sum_{i=1}^{N} e^2(i) + \sum_{j=1}^{n} \theta_j^2$$

Strafterm

## Interpretation von Bias und Varianz

- Veranschaulich am DART-Spiel oder Schießen auf Zielscheibe

## Bias

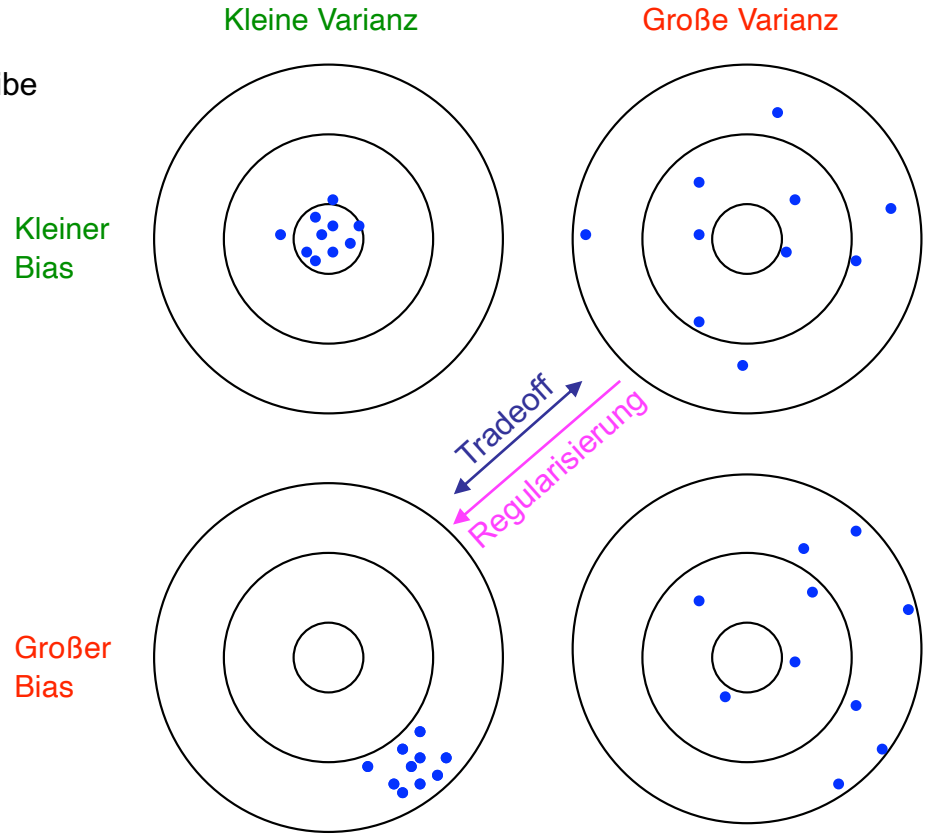
- Systematischer Fehler
- Soll so klein wie möglich sein

## Varianz

- Stochastischer (zufälliger) Fehler
- Soll so klein wie möglich sein

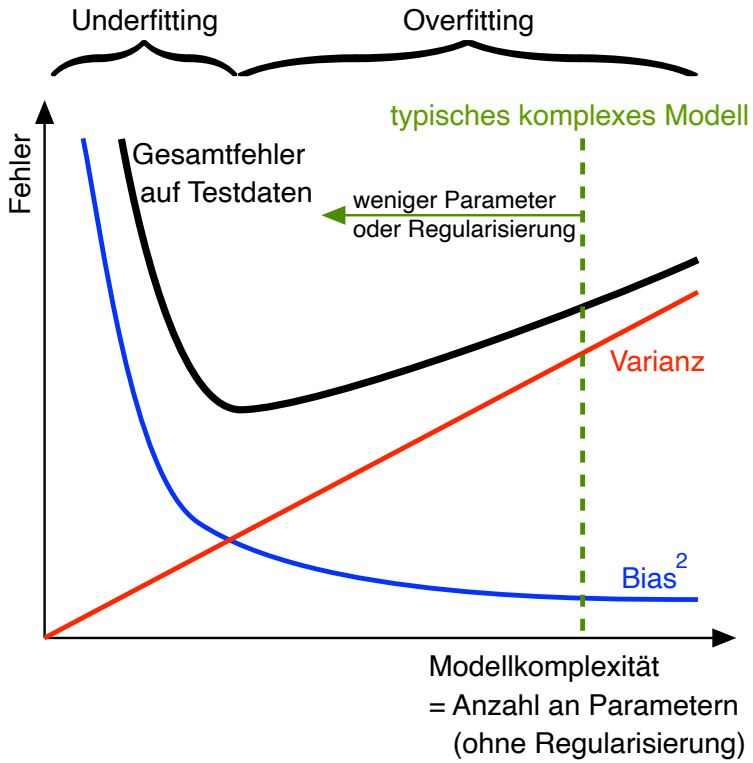
## Bias/Varianz-Tradeoff

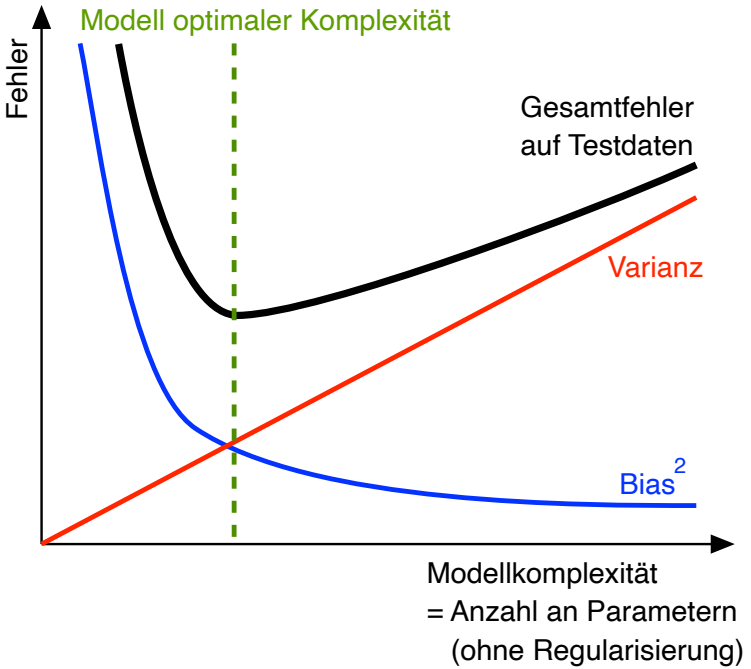
- Am liebsten wäre man oben links. Unten rechts ist Schrott. Leider kann man oben links meist nicht erreichen.
- Es findet typischerweise eine Kompromissbildung zwischen oben rechts und unten links statt: **Bias/Varianz-Tradeoff**.
- Oft wird etwas Bias geopfert (erhöht), um die Varianz zu verbessern (reduzieren): Das heißt **Regularisierung**.




Idee für Bild aus „The Master Algorithm“, Pedro Domingos, Pengin, 2017

## Bias/Varianz-Tradeoff und Regularisierung



# 1.5 Mehrfachfehler

**Beispiel Fukushima:** Erbeben und Tsunami waren nicht unabhängig! Vielmehr hat das Erdbeben den Tsunami ausgelöst ($\rho \to 1$).

## Einfachfehler

- Tritt typischerweise mit einer **niedrigen** Wahrscheinlichkeit ein
- Meist werden Diagnosesysteme „nur“ auf **Einfachfehler** ausgelegt
- Bei sehr **komplexen** Prozessen, werden aber auch Mehrfachfehler zunehmend wahrscheinlicher, z.B.
  - viele Komponenten, die zusammenwirken, z.B. Verbrennungsmotor mit Abgasbehandlung
    → hier ist bei Pkw eine On-Board Diagnosis (OBD) vorgeschrieben – seit 1988 in Kalifornien verpflichtend
    es werden vermutete Fehlerereignise in den Fehlerspeicher geschrieben, der später in der Werkstatt ausgelesen wird
  - viele Sensoren, oft örtlich verteilt, z.B. Windkraftanlage oder Bahnschienen auf Brücken (siehe 10. Case Study)
    → hier sollten die Sensoren überwacht werden (unterlagertes System)
    oft existiert viel Redundanz, so dass ein Sensorausfall unkritisch ist – allerdings sollte der Fehler erkannt und der Sensor deaktiviert werden, damit keine Fehlalarme getriggert werden

## Mehrfachfehler

- Tritt bei einfachen Prozessen typischerweise mit einer **extrem niedrigen** Wahrscheinlichkeit ein
- Wenn 2 Fehler **unabhängig** voneinander sind, ist deren gleichzeitiges Auftreten **extrem unwahrscheinlich**:

$$p^{(\text{2 Fehler gleichzeitig})} = p^{(\text{Fehler 1})} \cdot p^{(\text{Fehler 2})}$$

- Sind 2 Fehler aber **korreliert** (nicht unabhängig) **gilt** dies **nicht**! Mit zunehmender Korrelation wird $p^{(\text{2 Fehler gleichzeitig})}$ größer.
  Für $\rho \to 1$ geht $p^{(\text{2 Fehler gleichzeitig})} \to \max(p^{(\text{Fehler 1})}, p^{(\text{Fehler 2})})$, d.h. der Doppelfehler ist gleich wahrscheinlich wie der einfache.

# 2. 2-Klassen-Klassifikation

# Inhalt



## Empfohlene Videos zum Thema Klassifikation:

- Videos zu Statistical Learning von T. Hastie und R. Tibshirani (Stanford)
  https://www.youtube.com/watch?v=RN_dweQpcpo&list=PLAOUn-KLSAVP1jLuk5iBP2mhb6yB9QO1r

# 2.1 Klassifikation versus Regression

## Gemeinsamkeiten

- Ein oder mehrere Eingangsgrößen
- **Eingangsgrößen** können von Typ sein:
  - **reell / kontinuierlich** Normalfall
  - diskret: Liste von Berufen, Liste von Materialien, ...
  - binär: 0 oder 1, ja oder nein

**Viele Eingänge** → Macht das Problem sehr viel schwieriger
Der Eingangsraum wird exponentiell größer.
Demensprechend braucht man auch sehr viel mehr Daten.
Nur einige Modellarchitekturen können hochdimensionale Probleme gut handhaben.

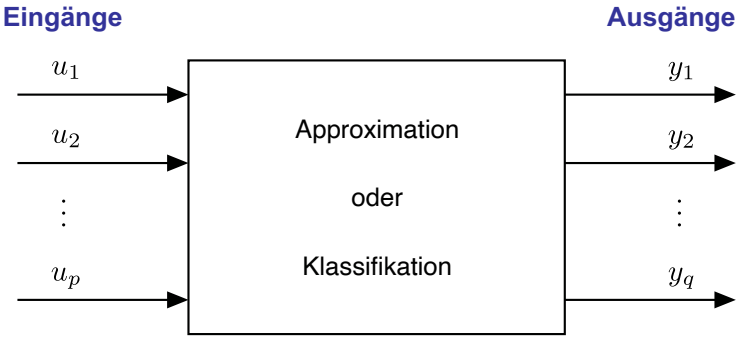
**Viele Ausgänge** → Der Aufwand steigt nur linear.
Man kann auch je Ausgang je ein Problem mit nur 1 Ausgang generieren.

- **Fluch der Dimension** (***curse of dimensionality***): starkes oft exponentielles Anwachsen des Aufwands mit der Eingangsdimension $p$. Aufwand kann sein: Rechenbedarf für Training oder Nutzung, Speicherbedarf, ...
- Ein oder mehrere Ausgangsgrößen

siehe auch Nelles: „Neuronale Netze und Fuzzy-Systeme“

## Unterschiede bei den Ausgangsgrößen

- **Regression** / **Approximation**: $y_i$, $i$ = 1, 2, ..., $q$ sind **reell**.
  Es sollen **Funktionen** approximiert werden $y_i = f_i(u_1, u_2, ..., u_p)$.
- Klassifikation: $y_i$, $i$ = 1, 2, ..., $q$ sind **binär**
  oder sie sind reell (und summieren sich typischerweise zu 1
  → Interpretation als Wahrscheinlichkeiten) und
  werden in etwas Binäres umgewandelt.
  Es sollen **Klassen** zugeordnet werden, wie „In Ordnung“ und „Fehler“.



## Wichtigste Fehlermaße für einen binären (2-Klassen) Klassifikator

Wenn P die Anzahl der Elemente der **positiven** und N die Anzahl Elemente der **negativen** Klasse sind, TP und TN die Anzahl der jeweils **richtig klassifizierten,** sowie FP und FN die Anzahl der jeweils **falsch klassifizierten** positiven bzw. negativen Elemente, dann berechnen sich die wichtigsten Klassifikationsfehler wie folgt:

- **True Positive Rate** (TPR) auch **Sensitivity** oder **Recall**: $TPR = \frac{TP}{P} = \frac{TP}{TP+FN} = 1 - FNR$
- **False Negative Rate** (FNR) auch **Miss Rate**: $FNR = \frac{FN}{P} = \frac{FN}{TP+FN} = 1 - TPR$

$\sum = 1$ — Bezug auf tatsächlich positive

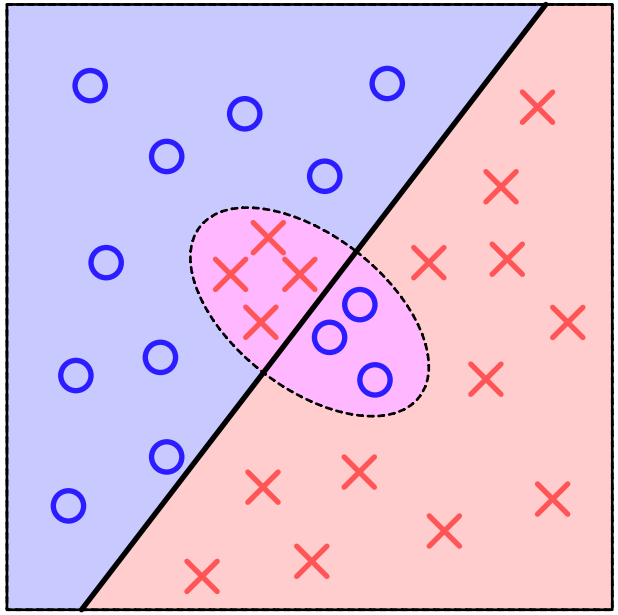
- **True Negative Rate** (TNR) auch **Specificity** oder **Selectivity**: $TNR = \frac{TN}{N} = \frac{TN}{TN+FP} = 1 - FPR$
- **False Positive Rate** (FPR) auch **Fall-Out**: $FPR = \frac{FP}{N} = \frac{FP}{TN+FP} = 1 - TNR$

$\sum = 1$ — Bezug auf tatsächlich negative

- **Positive Predictive Value** (PPV) auch **Precision**: $PPV = \frac{TP}{TP+FP}$ — Bezug auf positive tatsächlich und falsch
- **Negative Predictive Value** (NPV): $NPV = \frac{TN}{TN+FN}$ — Bezug auf negative tatsächlich und falsch

## Beispiel Fehlermaße und Konfusionsmatrix

Ein linearer Klassifikator trennt 13 positive (O) und 16 negative (X) Datenpunkte bis auf wenige Ausnahmen (3 O und 4 X) korrekt. Es ergeben sich die Fehlerraten:

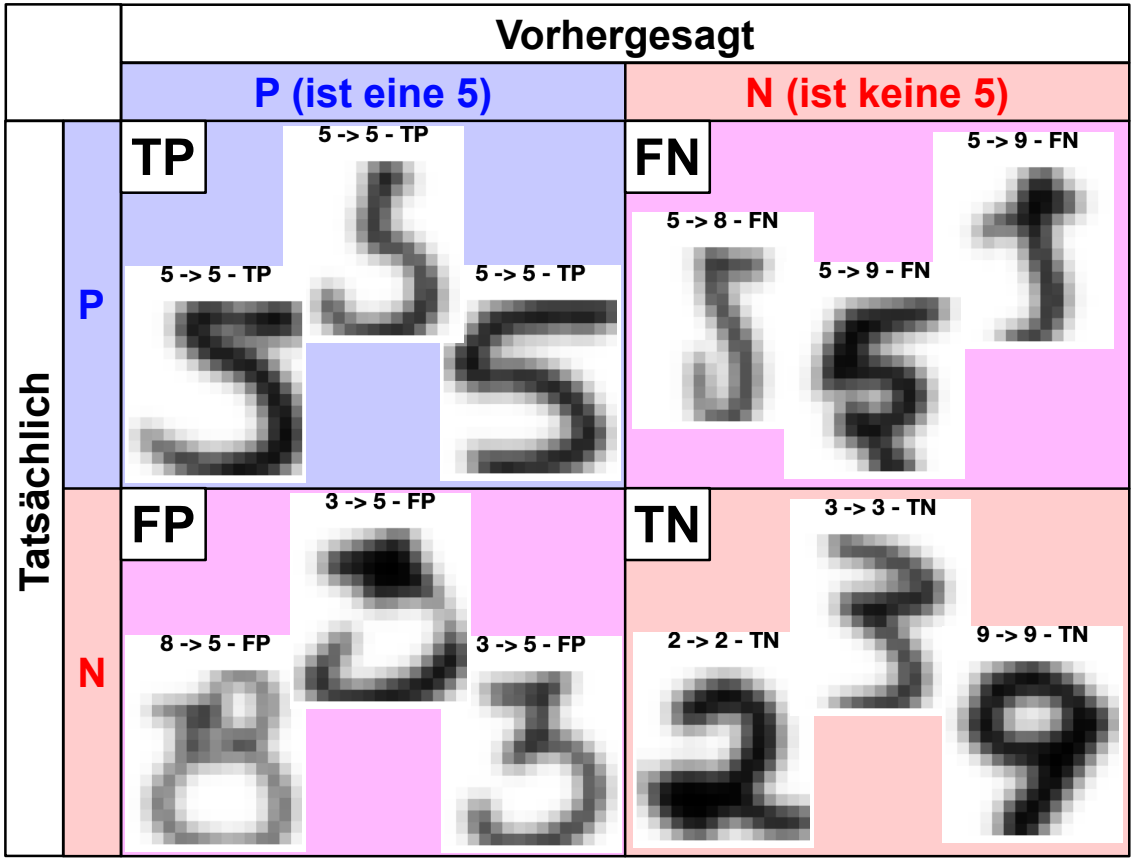


| | | Vorhergesagt P (O) | Vorhergesagt N (X) | |
|---|---|---|---|---|
| Tatsächlich P | | **TP** $TP = 10$ **True Positive Rate (Recall, Sensitivity):** $TPR = \frac{10}{13} \approx 0{,}77$ | **FN** $FN = 3$ **False Negative Rate (Miss Rate):** $FNR = \frac{3}{13} \approx 0{,}23$ | $\sum$: $TP + FN = 13$ (alle positiven) $TPR + FNR = \frac{10+3}{13} = 1$ |
| Tatsächlich N | | **FP** $FP = 4$ **False Positive Rate (Fall Out):** $FPR = \frac{4}{16} = 0{,}25$ | **TN** $TN = 12$ **True Negative Rate (Specificity, Selectivity):** $TNR = \frac{12}{16} = 0{,}75$ | $\sum$: $FP + TN = 16$ (alle negativen) $FPR + TNR = \frac{4+12}{16} = 1$ |
| | | $\sum$: $TP + FP = 14$ alle positiv vorhergesagten | $\sum$: $FN + TN = 15$ alle negativ vorhergesagten | |

# 2.2 Konfusionsmatrix und Receiver Operating Characteristic (ROC)

## Konfusionsmatrix Beispiel

- Erkennung von Ziffern, am Beispiel des *USPS Digit Dataset* (US Postal Service).
  - Enthält nur Ziffern von 0 bis 9. Das allgemeine Problem der Zeichenerkennung inkl. Buchstaben ist auch unter dem Begriff *Optical Character Recognition* (OCR) bekannt.
  - Der Datensatz enthält knapp 10.000 Bilder von handgeschriebenen Ziffern mit 16x16 Pixeln Auflösung (256 Klassifikatoreingänge).
- Das Bild zeigt Beispiele für die Erkennung oder Fehlerkennung der Ziffer 5:
  - **True Positives** (TP), **True Negatives** (TN): Der Klassifikator hat korrekt erkannt, das eine 5 (Positive) bzw. **keine** 5 (Negative) vorliegt.
  - **False Negatives** (FN): Der Klassifikator hat eine 5 als etwas anderes erkannt (z.B. 8, 9).
  - **False Positives** (FP): Der Klassifikator hat etwas anderes als eine 5 erkannt (z.B. 3, 8).

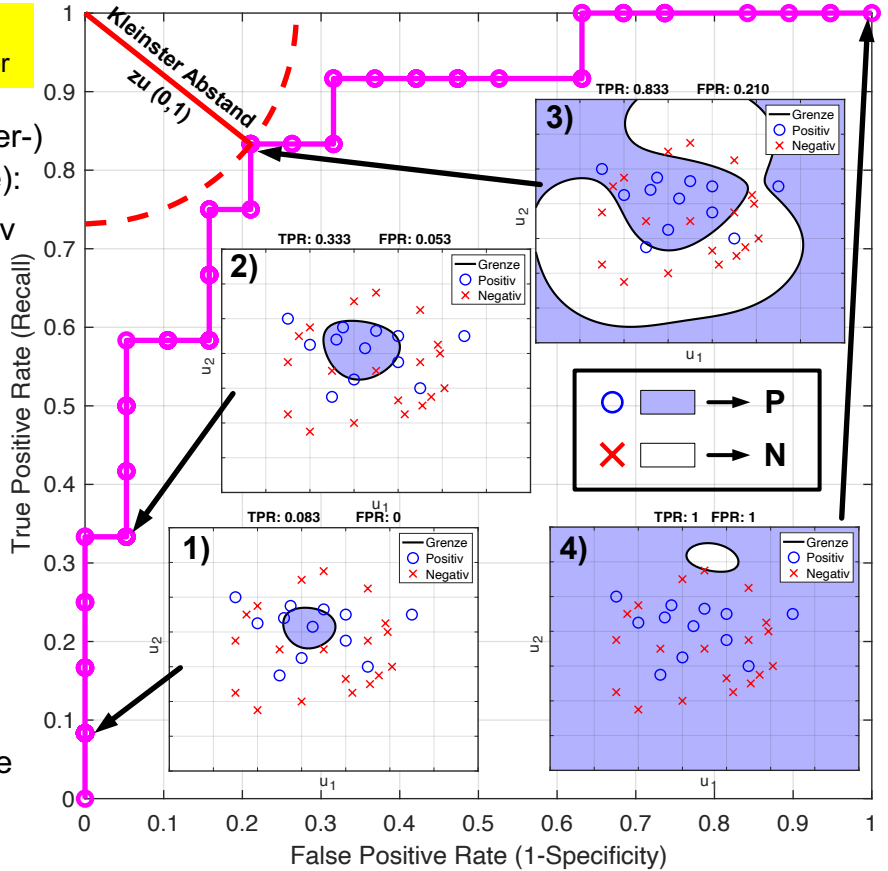


## Receiver Operator Characteristic (ROC)

Beispiel:
RBF-Klassifikator

Die ROC stellt die Änderung der True Positive Rate (TPR) über der False Positive Rate (FPR) bei Änderung eines (Hyper-) Parameters des Klassifikators dar (i.d.R. der Klassifikationsschwelle):

1. Im Koordinatenursprung (0,0) werden alle Datenpunkte als negativ klassifiziert. Daher sind alle 19 Kreuze (negativ) richtig (FPR=0), aber alle 12 Kreise (positiv) falsch (TPR = 0). Einen Schritt über dem Ursprung wird nun ein Kreis richtig klassifiziert (1 Kreis im blauen Bereich, TPR=1/12=0,083). Die Kreuze sind immer noch alle richtig (FPR = 0).
2. Durch die Ausdehnung der Klassifikationsgrenze werden nun 4 Kreise richtig klassifiziert (TPR = 4/12 = 0,333), allerdings ist nun auch ein Kreuz innerhalb der Klassifikationsgrenze (FPR = 1/19 = 0,053).
3. Dies ist der Fall, der dem Idealpunkt (0,1) am nächsten liegt. Hier werden nur 2 Kreise und 4 Kreuze falsch klassifiziert.
4. Am anderen Ende der Kurve liegt der Punkt (1,1). Nahezu die gesamte Fläche wird positiv klassifiziert. Damit sind alle Kreise richtig (TPR = 1), aber auch alle Kreuze falsch (FPR = 1) klassifiziert.



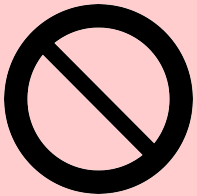
# 2.2 Konfusionsmatrix und Receiver Operating Characteristic (ROC)

## Bedeutung von True Positive (TP) und False Positive (FP) Werten

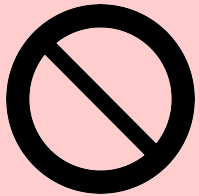
- Je nach Anwendungsgebiet ist es wichtiger möglichst **viele TP** (ROC oben rechts, Schwelle sehr niedrig) oder möglichst **wenige FP** (ROC unten links, Schwelle sehr hoch) zu erzielen.
- Beides kann nicht gleichzeitig erreicht werden − ideal wäre, man könnte bei der ROC den Punkt oben links erreichen. Eine Verbesserung des einen, verschlechtert aber das andere. → **Kompromiss ist nötig!**
- Ein Beispiel für **unterschiedliche Präferenzen** bei dieser Abwägung sind die Anwendungsfälle „SPAM-Mail Erkennung“ und „Jugendschutz“ (Erkennung nicht kindgerechter Inhalte im Internet):
  - **SPAM:** Erkennen von *keinem SPAM* (P) als *SPAM* (N) unerwünscht: **TP wichtiger!** Möglichst alle Mails die kein SPAM sind erkennen und dafür gelegentlich SPAM-Mails fälschlicherweise durchlassen. Löschen wichtiger Mails vermeiden.
  - **Jugendschutz:** Erkennen *ungeeigneter Inhalte* (N) als *geeignet* (P) unerwünscht: **FP wichtiger!** Möglichst keine Inhalte zulassen die ungeeignet sind und dafür lieber einzelne geeignete ablehnen.
- Die Abwägung findet oft zwischen Sicherheit und Komfort statt, z.B.:
  - **Erkennung von Störungen** an technischen Anlagen (Störung P, keine Störung N): Möglichst keine Störung übersehen (viele TP wichtig), möglichst selten grundlos alarmieren (wenige FP wichtig).
  - **Zugangskontrolle** (erlaubt P, verboten N), z.B. mit Fingerabdruck- oder Retinasensor: Möglichst keinen unberechtigten Zugang gewähren (wenige FP wichtig), aber häufiges Ausprobieren oder gar Abweisen eines Berechtigten vermeiden (viele TP wichtig). Kompromiss hier von der Höhe des Sicherheitsrisikos abhängig!
- Der Zusammenhang zwischen TP- und FP-Rate (Verhältnis von TP zu allen positiven, bzw. von FP zu allen negativen) wird anschaulich mit der **Receiver Operating Characteristic (ROC)** dargestellt (siehe vorherige Folie).

# 2.2 Konfusionsmatrix und Receiver Operating Characteristic (ROC)

## SPAM-Erkennung

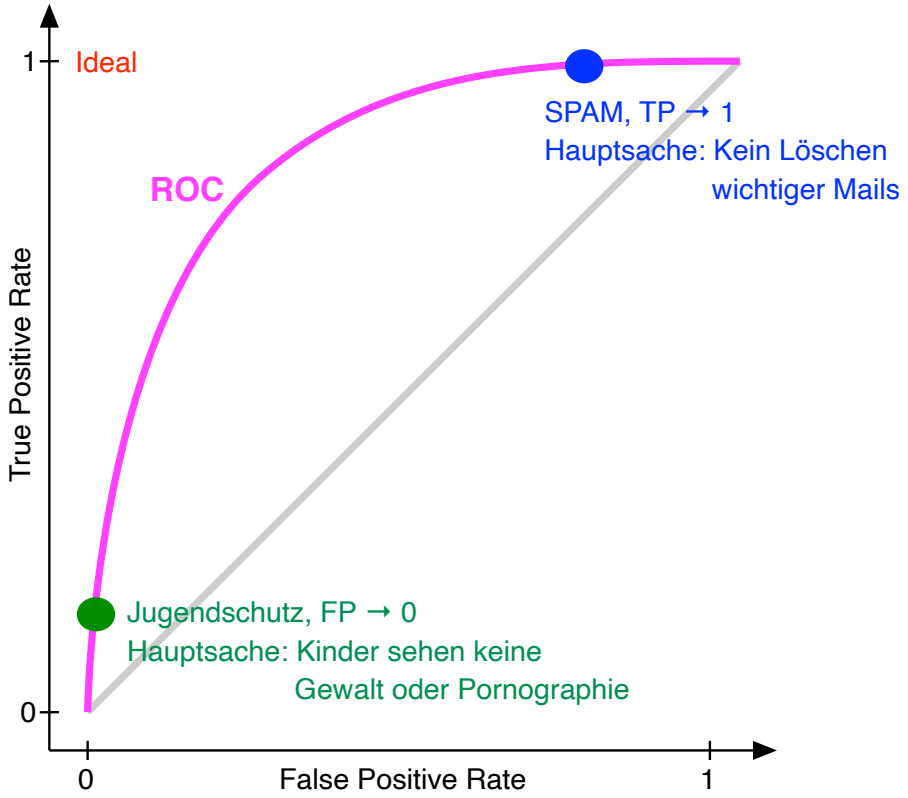
| Tatsächlich \ Vorhergesagt | P (kein SPAM) | N (SPAM) |
|---|---|---|
| P | TP  | FN Löschen wichtiger Mails  darf nicht passieren! |
| N | FP Durchlassen von SPAM  nicht so schlimm! | TN  |

## Jugendschutz

| Tatsächlich \ Vorhergesagt | P (geeignete Inhalte) | N (ungeeignete Inhalte) |
|---|---|---|
| P | TP  | FN Geeignete Inhalte werden gesperrt  nicht so schlimm |
| N | FP Kinder sehen Gewalt oder Pornografie  darf nicht passieren! | TN  |

## Bedeutung von True Positive (TP) und False Positive (FP) Werten

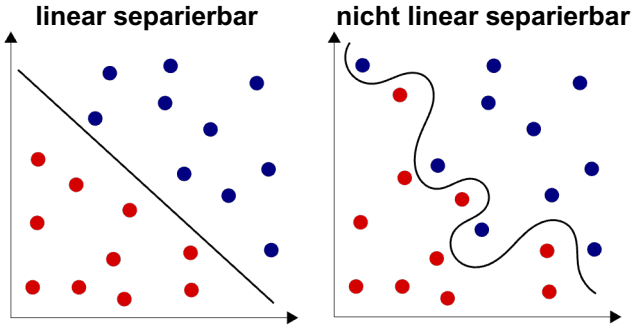
- SPAM und Jugendschutz sind 2 **extreme Beispiele**, bei denen man sich durch Einstellen der Schwelle nahe eines Eckpunktes der ROC legen möchte:
  - SPAM: Oben rechts
  - Jugendschutz: Unten links
- Bei den **meisten Anwendungen** wird man eher einen ausgleichenden Kompromiss suchen, weil beide Fehler (FP und FN) ähnliche schlimme Auswirkungen haben – die Situation ist dann symmetrischer.
- Diesen **ausgleichenden** Kompromiss findet man, indem man dem Bereich oben links nahe (Ideal) kommt.
- Z.B. kann man den Punkt auf der ROC mit dem **kleinsten Abstand** zum Ideal suchen.
- Man kann **verschiedene Klassifikatoren** in den ROC-Plot einzeichnen. Je weiter man nach oben links (Ideal) kommt, desto besser!
- Die graue, diagonale Gerade kennzeichnet einen **Zufallsklassifikator**.



# 2.3 Lineare Separierbarkeit

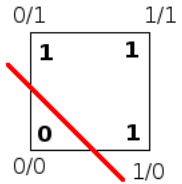
## Lineare Separierbarkeit = Trennbarkeit durch eine lineare Funktion

- Eine Trennung der Klassen durch eine **lineare Funktion** ist besonders **einfach**
  → Es lohnt sich viel Aufwand zu treiben, um durch eine **geschickte Wahl** der **Eingänge/Features** diese Eigenschaft zu erzeugen
- Lineare Funktion: 1D = Konstante, 2D = Gerade, 3D = Ebene, 3+D = Hyper-Ebene
- Dann kann ein **linearer Klassifikator** eingesetzt werden
  → Standardproblem der Statistik (**logistische Regression**)
  → Moderne Verfahren wie **lineare Support Vector Machine**
- Vorteile: Robust, wenig Rechenaufwand, wenige Hyperparameter, wenig Probieren

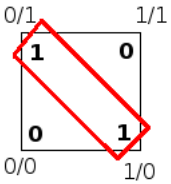



## Keine lineare Separierbarkeit

- In komplexen Problemen ist lineare Separierbarkeit nicht erreichbar.
  Es zwei mögliche Gründe warum dies nicht möglich ist:
  1. Wird durch Rauschen verhindert (Bild rechts oben)
     → Rauschen ignorieren und nur halbwegs korrekte Klassifikation akzeptieren
  2. Ist strukturell unmöglich (Bild rechts unten)
     → Klassifikationsgrenze nichtlinear machen mit leistungsfähigeren Klassifikatoren
- Problem: Ob 1 oder 2 zutrifft, ist unbekannt!



OR-Funktion:
Linear separierbar



XOR-Funktion:
Nicht linear separierbar

Quelle: https://de.wikipedia.org/wiki/Lineare_Separierbarkeit

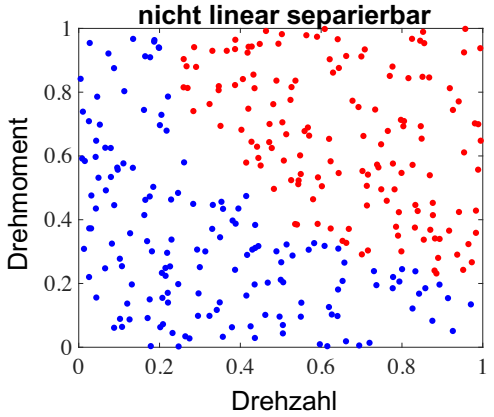
## Transformation der Eingänge/Features

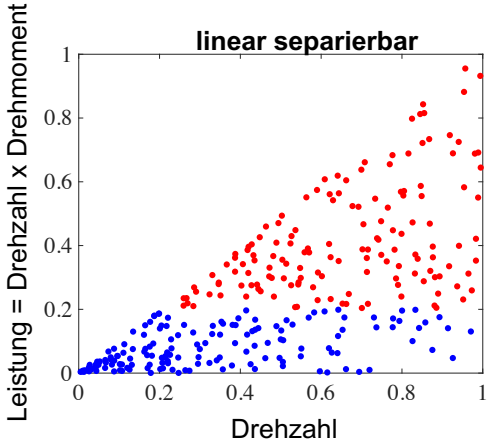
- Durch geschickte Wahl der **Eingänge/Features** kann manchmal das Problem stark vereinfacht werden
- Oft sind aus der Physik des Problems bestimmte nichtlineare Zusammenhänge bekannt, die in die Eingänge/Features integriert werden können, z.B.
  - Leistungsgrenzen, z.B. Maximalleistung eines Motors
    a) Darstellung Drehmoment = *f*(Drehzahl) führt zu einer Grenzkurve in **Hyperbel-Form**
    b) Darstellung Leistung = *f*(Drehzahl) führt zu einer **linearen** Grenzkurve
  - Bei einer Arbeitsmaschine hängen viele Effekte an der Leistung, nicht an den einzelnen Signalen, z.B. Elektromotor
    a) Modell = *f*(Strom, Spannung)
    b) Modell = *f*(Leistung) mit Leistung = Strom x Spannung
- Die geschickte Wahl, Manipulation, Transformation der Eingänge/Features nennt man

  **Feature Engineering**

  und ist traditionell eine Kernaufgabe eines Ingenieurs / Fachexperten
- Bei sehr komplexen Problemen (insb. Bild- und Sprachverarbeitung) wird das manuelle Feature Engineering zunehmend durch automatisches Lernen abgelöst → **Deep Learning.**

  Nachteil: Benötigt sehr **viele Daten** und **eliminiert** jegliche **Transparenz**.



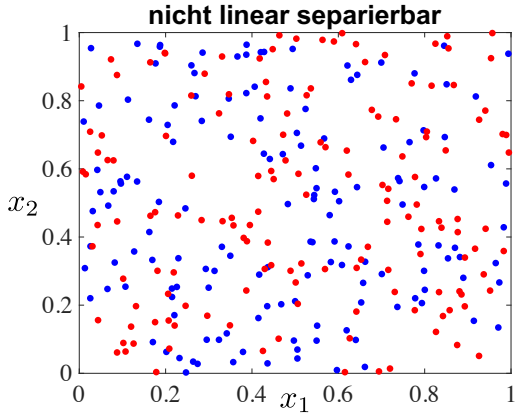



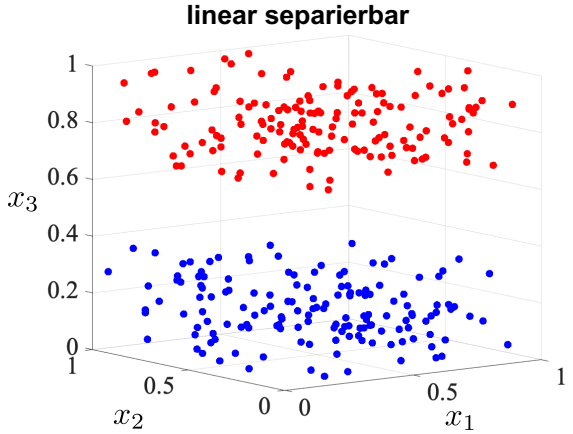
## Dimensionserhöhung

- Meist ist man am Gegenteil interessiert: **Dimensionsreduktion**, die macht das Problem
  - einfacher
  - übersichtlicher
  - schneller zu rechnen
  - visualisierbarer
- Je **mehr informative Eingänge/Features** zur Klassifikation herangezogen werden, umso **leichter** wird die Klassifikationsaufgabe.
- Auf der anderen Seite wird der Klassifikator mit **jeder** zusätzlichen **Dimension komplexer** (mehr Parameter).

→ Ein zusätzlicher Eingang/Feature sollte mehr Informationen bringen als es durch zusätzliches Overfitting schadet.

- Im Beispiel rechts können die Klassen in 2D $x_1$-$x_2$ nicht separiert werden. Mit der zusätzlichen Dimension $x_3$ wird es leicht möglich.
- Es wäre allerdings viel besser gewesen $x_3$ als erste Dimension zu wählen, da sie offensichtlich die meiste Information über die Klassenzugehörigkeiten enthält.
- Allerdings ist dies oft nicht vorab bekannt und muss erst ermittelt werden.



# 2.5 Robustheit der Klassifikationsgrenze

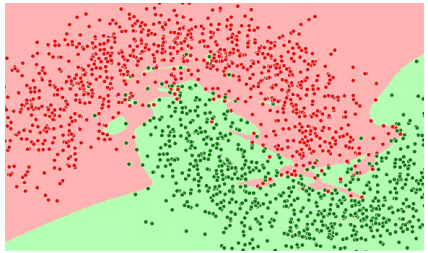
## Was ist Robustheit der Klassifikationsgrenze?

- Nach dem Training eines Klassifikators wird die Klassifikationsgrenze typischerweise zur Trennung der Klassen (für die Trainingsdaten) gut geformt sein. Aber wird sie auch gut generalisieren, d.h. für neue Daten gut funktionieren. Es gibt viele Versuche, diese Generalisierungsfähigkeit zu bewerten:
  - Performance auf Validierungsdaten: Standardvorgehen. Aber sind die Validierungsdaten wirklich repräsentativ, für alles, was in Zukunft noch kommen wird...?
  - Flexibilität des Klassifikators bzw. Komplexität der Grenze
  - Margin oder Boundary Thickness

Modell- und Dateneigenschaften

Modelleigenschaften

**Komplexe Klassifikationsgrenze Overfitting**



## Gewünschte Eigenschaften für Robustheit der Klassifikationsgrenze?

- Die Grenze sollte **möglichst einfach** sein (z.B. linear oder zumindest wenig "wellig/schnörkelig") – **je komplexer** die Grenze, desto mehr Parameter/Freiheitsgrade braucht man für deren Beschreibung und desto mehr neigt der Klassifikator zu **Overfitting**.
- Der Abstand zwischen den Klassen (*margin* → Kapitel 6 SVM oder *boundary thickness*) sollte möglichst groß sein.

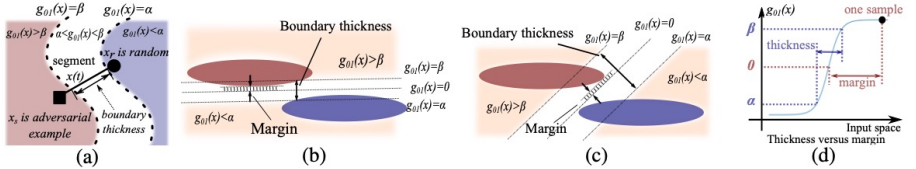



Figure 1: **Main intuition behind boundary thickness.** (a) Boundary thickness measures the gap between two level sets $g_{01}(x) = \alpha$ and $g_{01}(x) = \beta$ along the adversarial direction. (b) A thin boundary easily fits into the narrow space between two different classes, but it is not robust. (c) A thick boundary is harder to achieve a small loss, but it achieves higher robustness. In these two toy settings, the margin is the same, but choosing a thicker boundary leads to better robustness. (d) Illustration of the relationship between boundary thickness and margin.

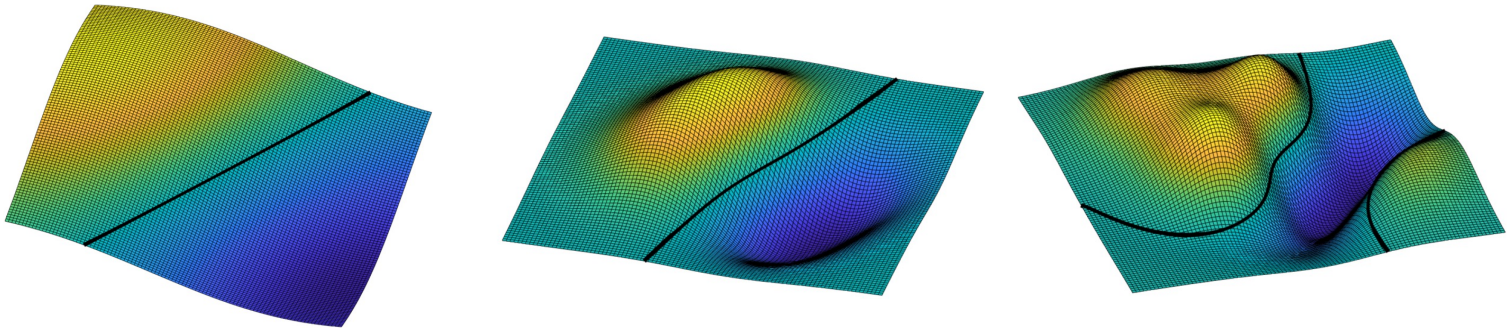
Quelle: Yaoqing Yang, Rajiv Khanna, Yaodong Yu, Amir Gholami, Kurt Keutzer, Joseph E. Gonzalez, Kannan Ramchandran, Michael W. Mahoney: „Boundary thickness and robustness in learning models", *Advances in Neural Information Processing Systems* 33 (2020): 6223-6234

# 2.5 Robustheit der Klassifikationsgrenze

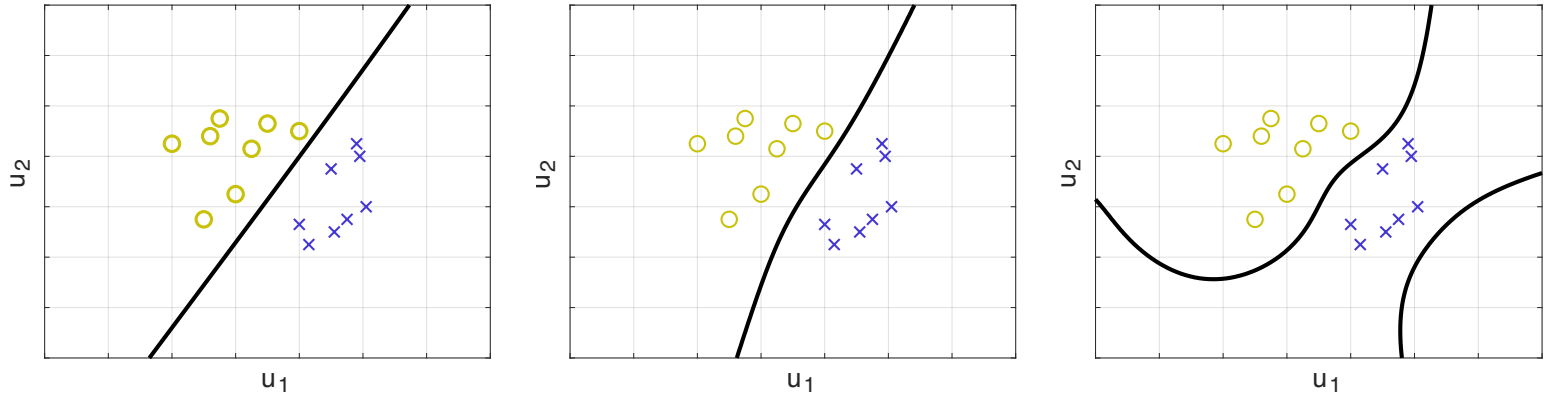
Bei einem Klassifikator kann die Klassifikationsgrenze durch die Wahl von Hyperparametern unterschiedlich glatt eingestellt werden. Ist die Grenze aber flexibler als es der Klassifikationsaufgabe angemessen ist (Overfitting), kann es bei der Anwendung des Klassifikators auf neue Daten leichter zu Fehlklassifikationen kommen:

**Klassifikations-funktion:**

$\geq 0$: Kreise
$< 0$: Kreuze



**Lage der Daten-punkte:**
Rechte Variante deutlich zu flexibel für Klassifikations-problem



## Warum mehrere Datensätze?

- Es werden minimal 2 Datensätze benötigt, oft finden sogar 3 Datensätze Verwendung
- **Datensätze**:
  - **Training**: Hiermit werden die Modelle **gelernt/trainiert**
    Danach „kennt“ das Modell diese Daten, hat sich auf deren Rauschrealisierung und an deren Verteilung angepasst.
    Dieser Anpassungseffekt (**Overfitting**) ist unerwünscht und umso stärker, je flexibler das Modell ist.
    Er kann aber nicht vollständig vermieden werden. Die **Qualität** des Modells auf **Trainingsdaten** ist immer **überoptimistisch**!
  - **Test**: Hiermit wird das Modell **getestet**
    Diese Daten sind zuvor „weggeschlossen“ und dem Modell unbekannt. Daher kann objektiv die Qualität des Modells auf neuen Daten (Generalisierung) beurteilt werden.
  - **Validierung (optional)**: Hiermit kann während des Lernens die Qualität des Modell objektiv eingeschätzt werden.
    Diese Einschätzung wird genutzt, um die Struktur und Flexibilität des Modells (nahezu) optimal einzustellen:
    - Auswahl der Eingänge/Features
    - Anzahl der Schichten, Neuronen o.ä.
    - Regularisierungsstärke(n)

**WICHTIG:** Das lässt sich *nicht* anhand der Trainingsdaten einstellen, da wegen des Overfittings auf Trainingsdaten flexiblere Modelle *immer* besser aussehen – auf neuen Daten (wie Validierungsdaten) aber nicht sind. Nur so lässt sich eine gute/optimale Modellkomplexität/flexibilität bestimmen.

- Die Gesamtdaten werden typischerweise ungefähr in folgenden Verhältnissen **zufällig** aufgeteilt:
  - Training : Test: 70 : 30 oder 80 : 20
  - Training : Validierung : Test: 70 : 15 : 15 oder 60 : 20 : 20

# 2.6 Datensätze für Training, Validierung, Test

## Alternativen zu Validierungsdaten

- Wenn man auf Validierungsdaten verzichten kann, können diese den Trainingsdaten zugeschlagen werden → mehr Trainingsdaten!
- Kreuzvalidierung (*cross validation*)
- Informationskriterium (*information criterion, IC*) wie Akaike IC (AIC) für $\rho = 2$ oder Bayesian IC (BIC) für $\rho = \ln N$ :

$$\mathrm{IC}(\rho) = N \ln(I(\underline{\theta})) + \rho\, n$$

Bestrafung der Flexibilität des Modells

$N$ = Anzahl Datenpunkte
$n$ = Anzahl Parameter

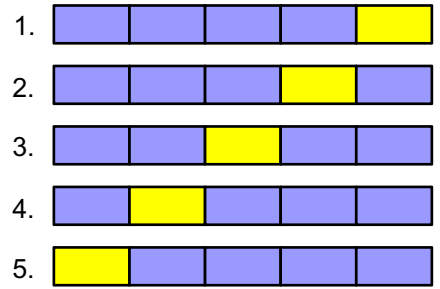
$$I(\underline{\theta}) = \frac{1}{N}\sum_{i=1}^{N} e^2(i)$$

Verlustfunktion

## Kreuzvalidierung

- Aufteilung der Trainingsdaten in $K$ gleichgroße Teile (*folds*)
- Typisch ist $K = 5$ oder $K = 10$
- Davon wird 1 Teil zur **Validierung** verwendet und $K$-1 Teile zum **Training**
- Das Ganze wird für alle $K$ Aufteilungen wiederholt
- Der Kreuzvalidierungsfehler ist die Summe aller $K$ Validierungsfehler
- Am Ende wird mit dem ganzen Datensatz trainiert. Die Prozedur dient lediglich der Ermittlung eines Validierungsfehlers, ohne Validierungsdaten abzusplitten.

**5-fach Kreuzvalidierung (*5-fold cross validation*)**




$N$ Datenpunkte

## Leave-One-Out (LOO) und Generalized Cross-Validation (GCV) Fehler

- Die „extremste“ Form der Kreuzvaldierung mit *K* = *N*, d.h. der (gelbe) Teil zur Validierung besteht immer nur aus **einem Punkt**, der für das jeweilige Training weggelassen wurde – daher der Name
- Im Allgemeinen: **Immenser Rechenaufwand**, da die Anzahl an Datenpunkten *N* meist (sehr) groß ist
  → typischerweise praktisch nicht machbar!
- **ABER**
  - Für Probleme, die **linear-in-den-Parametern** sind (lineare Regressionsprobleme), existiert eine recht einfache geschlossene (analytische) Formel zur Berechnung des LOO-Fehlers
  - Beispiele für solche linearen Regressionsprobleme sind Schätzung von
    - Polynomkeffizienten
    - Ausgangsgewichten eines RBF-Netzes oder eines Gaußprozessmodells
    - ...

Sollte viel mehr genutzt werden!
Ein sehr **mächtiges** Tool!

- **LOO-Formeln**

Modell: $\underline{\hat{y}} = \underline{X}\,\hat{\underline{\theta}}$

Least-Squares-Schätzung

Optimale Parameter: $\hat{\underline{\theta}} = \left(\underline{X}^T\underline{X}\right)^{-1}\underline{X}^T\underline{y}$

Zusammenhang zwischen *Modell*ausgang und *Prozess*ausgang:

$$\underline{\hat{y}} = \underbrace{\underline{X}\left(\underline{X}^T\underline{X}\right)^{-1}\underline{X}^T}_{\underline{S}}\underline{y}$$

**Smoothing-Matrix**
oder Hat-Matrix

- LOO-Fehler: $e^{(\mathrm{LOO})}(i) = \dfrac{e(i)}{1 - s_{ii}}$

Diagonalelemente von $\underline{S}$

GCV-Fehler: $e^{(\mathrm{GCV})}(i) = \dfrac{e(i)}{1 - \mathrm{tr}(\underline{S})/n}$

# 2.7 Einfacher Bayes Klassifikator (*Naive Bayes*)

## Annahme

- Alle Eingänge/Features sind **unabhängig**
- D.h. die **Korrelationen** zwischen den Eingängen/Features $x_i$, $i$ = 1, 2, ..., $n$ werden **ignoriert** bzw. vernachlässigt
- Daher, dass diese Korrelationen in der Realität praktisch immer auftreten und sogar groß sind, rührt der Name (***Naive***)

Ein super Erklärvideo zum **Satz von Bayes** „Bayes theorem, the geometry of changing beliefs“ von 3Blue1Brown findet sich unter: https://www.youtube.com/watch?v=HZGCoVF3YvM

## Herleitung

- Der **Satz von Bayes** wird genutzt.
- Hier 2 Klassen, d.h. entweder Klasse $k$ = 1 oder $k$ =2

**Satz von Bayes**

$$p(\text{Hypothese}\,|\,\text{Daten}) = \frac{p(\text{Hypothese}) \cdot p(\text{Daten}\,|\,\text{Hypothese})}{p(\text{Daten})}$$

- **Gegeben:** Wahrscheinlichkeit der **Klassen** (vor Datenerhebung – a priori (*prior*)): $p(C_k)$
  Wahrscheinlichkeitsdichte der **Daten** insgesamt: $p(\underline{x})$
  Wahrscheinlichkeitsdichte der **Daten, wenn Klasse *k* vorliegt**: $p(\underline{x}\,|\,C_k)$
- **Gesucht:** Die Wahrscheinlichkeiten für Klasse 1 $p(C_1\,|\,\underline{x})$ und für Klasse 2 $p(C_2\,|\,\underline{x})$, **nachdem die Daten** $\underline{x}$ erhoben wurden:

$$\overbrace{p(C_k|\underline{x})}^{\text{posterior}} = \frac{\overbrace{p(C_k)}^{\text{prior}} \cdot \overbrace{p(\underline{x}|C_k)}^{\text{likelihood}}}{\underbrace{p(\underline{x})}_{\text{evidence}}}$$

**Klassifikator**: **Maximum A Posteriori (MAP)**, also:
Wähle für jedes $\underline{x}$ die Klasse $k$, mit der höchsten Wahrscheinlichkeit

Ohne Berücksichtigung des Priors $p(C_k)$: **Maximum Likelihood**

# 2.7 Einfacher Bayes Klassifikator (*Naive Bayes*)

## Vereinfachung durch Annahme

- Durch die (angenommene) **Unabhängigkeit** der Eingänge/Features, **vereinfacht** sich die Mathematik erheblich
- Im Allgemeinen ist die **Likelihood** eine ***n*-dimensionale Funktion**, wenn *n* Eingänge/Features vorliegen. Durch die Unabhängigkeit dürfen die Randwahrscheinlichkeitsdichten einfach multipliziert werden, d.h. es ergibt sich ein Produkt aus *n* 1-dimensionalen Funktionen:

wegen Unabhängigkeit der Eingänge/Features $x_i$, $i = 1, 2, ..., n$

$$\overbrace{p(\underline{x}|C_k)}^{\text{likelihood}} = p(x_1, x_2, \ldots, x_n|C_k) = p(x_1|C_k) \cdot p(x_2|C_k) \cdot \ldots \cdot p(x_n|C_k) = \prod_{i=1}^{n} p(x_i|C_k)$$

## Anmerkungen

- Meist wird eine Normalverteilung für die Wahrscheinlichkeitsdichten angenommen. Verallgemeinerung → 7. Dichteschätzung
- Trotz ihrer Einfachheit funktionieren diese Klassifikatoren überraschend gut – dafür gibt es auch theoretische Begründungen...
- Der Unterschied zwischen einer Posterior-Betrachtung über den Satz von Bayes und einer Likelihood-Betrachtung ist, dass bei einer Posterior-Betrachtung mit den Klassenwahrscheinlichkeiten gewichtet wird. Das kann einen sehr großen Unterschied machen. In vielen Fällen sind die $p(C_k)$ sehr unterschiedlich (oft andere Größenordnungen), z.B.: Wahrscheinlichkeiten für
  - Fehler / In Ordnung
  - Krank / Gesund
  - Betrug / Normaler Kunde
- Für mehr Details zu Bayes Klassifikatoren siehe z.B. https://en.wikipedia.org/wiki/Naive_Bayes_classifier

# 2.7 Einfacher Bayes Klassifikator (*Naive Bayes*)

## Beispiel: Bayes-Klassifikator mit Normalverteilungen

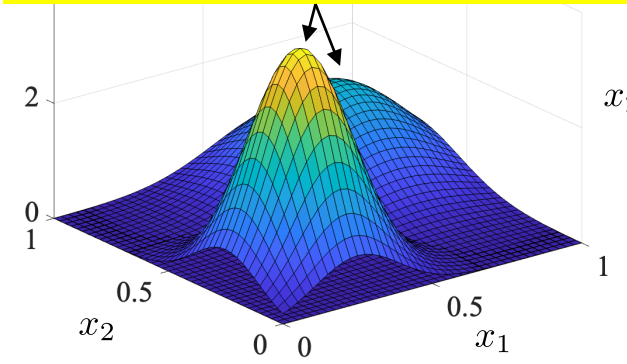
- Einfaches 2-D-Beispiel
- Eine Klasse $C_1$ (rot = 0): unten links
- Andere Klasse $C_2$ (blau = 1): sonst
- 140 **verrauschte** Datenpunkte zufällig in $[0\ \ 1]^2$ verteilt

$p(C_1) = 1/4$
$p(C_2) = 3/4$

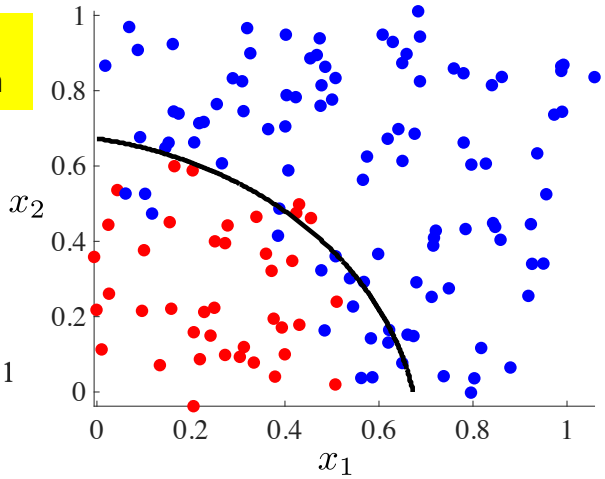
weil 3x so viele blaue Punkte wie rote!

**Wahrscheinlichkeitsdichten $p(\underline{x}\,|\,C_1), p(\underline{x}\,|\,C_2)$**

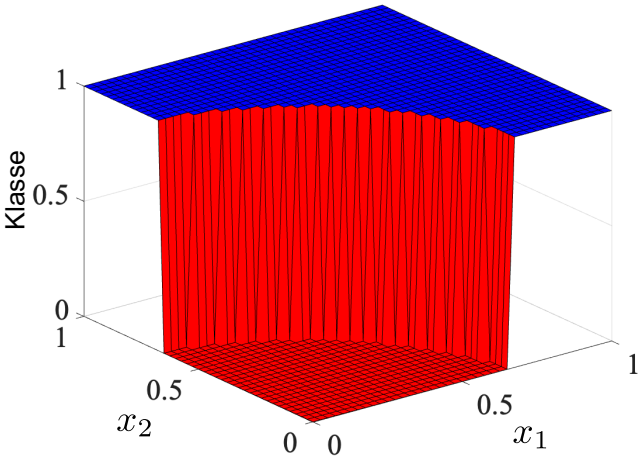
beste Gaußglocken zur Beschreibung der roten und blauen Punkteverteilungen




**Daten und Klassifikationsgrenze**




**Bayes-Klassifikator**



# 2.8 Mehrklassen-Klassifikation

## 2 Klassen

- Aufgabe: Binäre Klassifikation
- Fehlererkennung trennt typischerweise 2 Klassen: In Ordnung / Fehler (defekt)

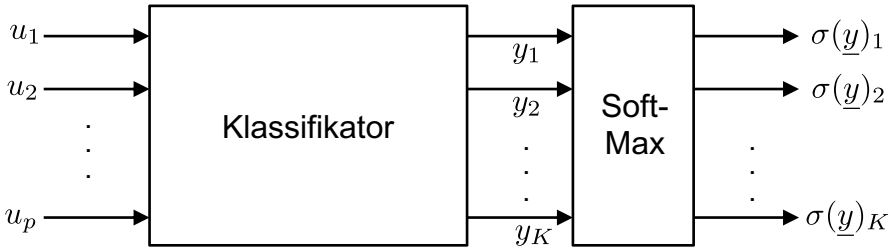
## Mehrere Klassen

- Fehlerdiagnose soll typischerweise $K=q+1$ Klassen trennen: In Ordnung / Fehler 1 / Fehler 2 / ... / Fehler $q$
- Drei verschiede Ansätze für Klassifikator

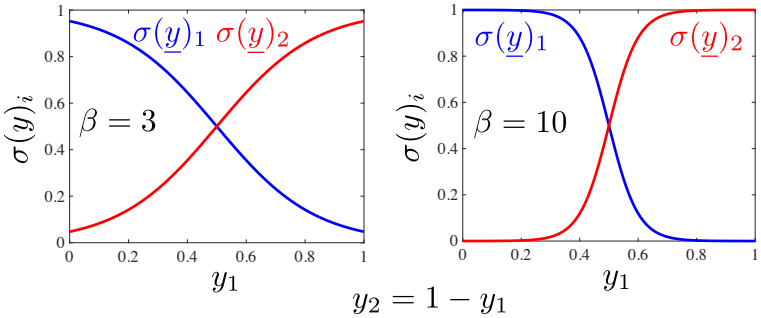
**1. Erweiterung der Struktur des binären Klassifikators (abhängig von Architektur!)**

Z.B. ein Ausgangsneuron pro Klasse → $K$ Ausgangsneuronen

**Soft-Max**: Alle Ausgänge addieren sich zu 1 und können als **Wahrscheinlichkeiten** für die zugehörige Klasse interpretiert werden
**Entscheidung** für Klasse mit höchster Wahrscheinlichkeit




**Soft-Max-Funktion**



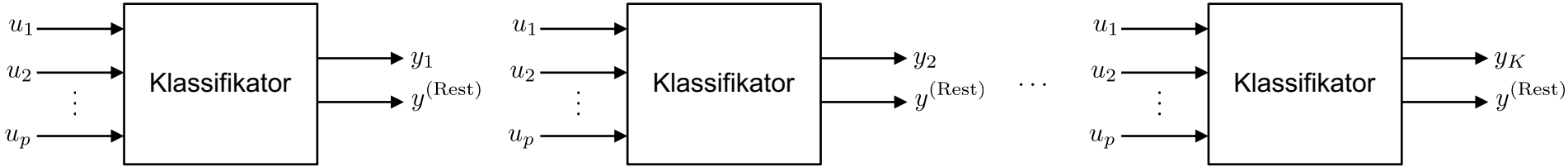

$$y_2 = 1 - y_1$$

$$\sigma(\underline{y})_i = \frac{\mathrm{e}^{-\beta y_i}}{\mathrm{e}^{-\beta y_1} + \mathrm{e}^{-\beta y_2} + \ldots + \mathrm{e}^{-\beta y_K}}, \quad i = 1, 2, \ldots, K$$

## Nutzung binärer Klassifikatoren

### 2. Einer-gegen-Alle (*one-vs.-rest* oder *one-vs.-all*)

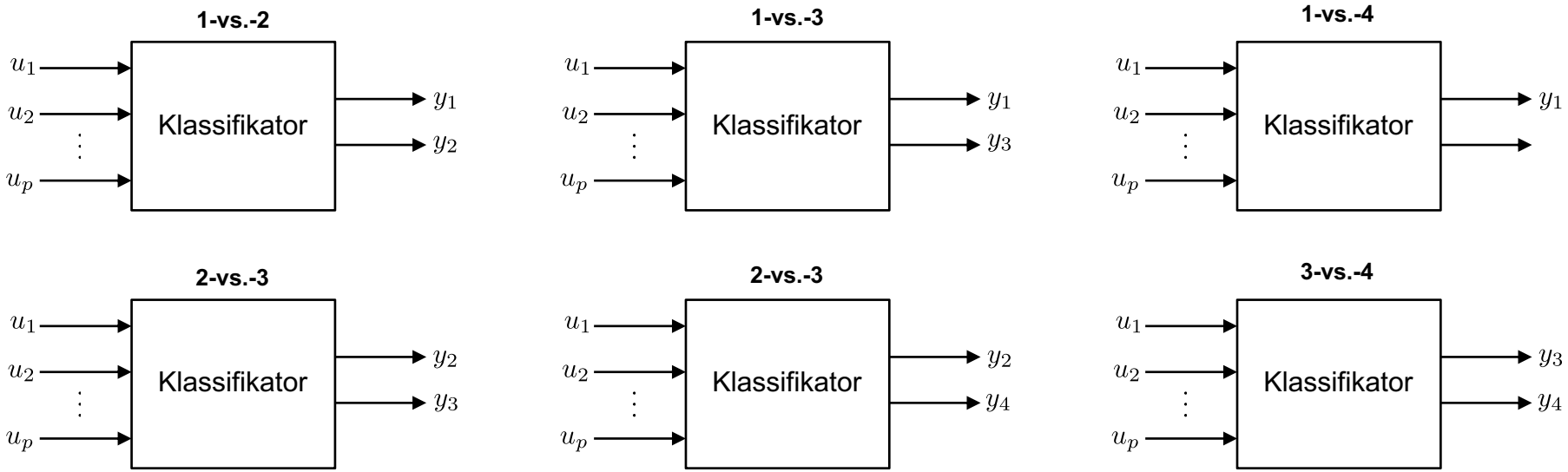
- *K* Klassifikatoren
- Gewinner-Klasse: Wo $y_i$ (Konfidenz) am größten
- Problem: Unterschiedlich große Klassen
- Problem: Selbst wenn Trainingsdaten „balanced“, d.h. alle Klassen gleich häufig vertreten, Ungleichheit in der Punktehäufigkeit, da $y_i$ weniger Punkte hat als $y^{(\text{Rest})}$



## Nutzung binärer Klassifikatoren

### 3. Einer-gegen-Einen (*one-vs.-one*)

- Alle Kombinationen aus 2 Klassen: $K(K\text{-}1)/2$ Klassifikatoren
- Gewinner-Klasse: Voting: Welche Klasse gewinnt am häufigsten? Bei Unentschieden entscheiden die $y_i$ (Konfidenzen)
- Beispiel für $K$=4:

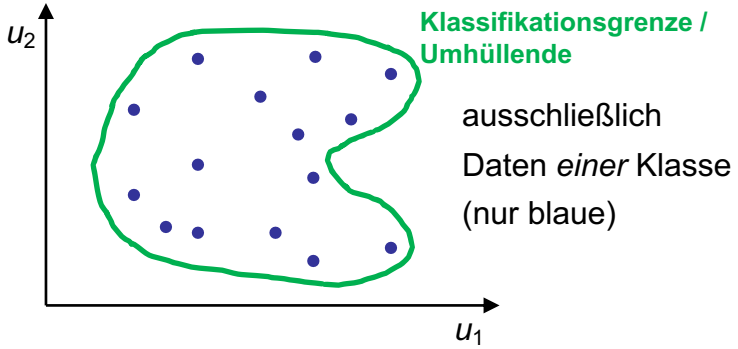


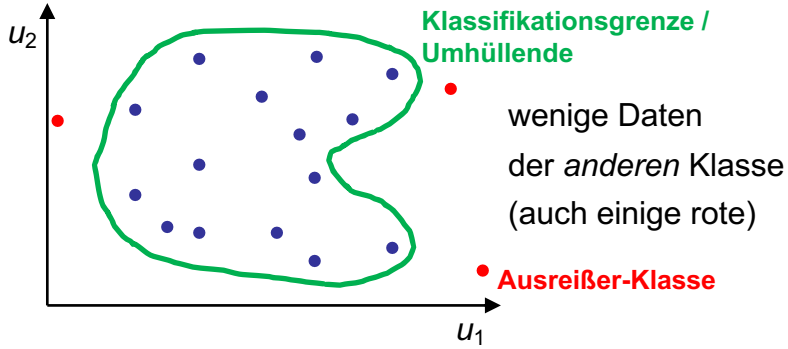
# 3. 1-Klassen-Klassifikation

# Inhalt

## Problemstellung

- Es gibt (fast) ausschließlich Daten **einer Klasse** (z.B. nur positive Beispiele bzw. nur Daten aus fehlerfreiem Betrieb)




## Viele *Namen* für eine Sache: 1-Klassen-Klassifikation

- Je nach **Disziplin** und **Anwendungszweck** existieren viele **synonym** verwendete Bezeichnungen:
- Ein-Klassen-Klassifikation *(One-Class Classification, OCC)*
- Anomalie-Erkennung *(Anomaly Detection)*
- Neuigkeits-Erkennung *(Novelty Detection)*
- Ausreißer-Erkennung (*Outlier Detection*)
- *Unary Classification* oder *Concept Learning*

## Diskussion des Problems

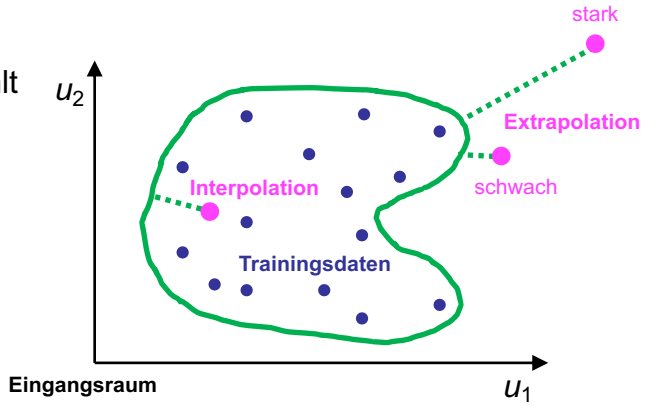
- Bei 2 Klassen muss die **Klassifikationsgrenze zwischen beiden** liegen – am besten mittig.
  Bei 1 Klasse wird die Klassifikationsgrenze nur von Daten auf **einer Seite** gestützt
- Mittels 1-Klassen-Klassifikation kann man unterscheiden:
  - liegt ein (neuer) Punkt **in** der **Punktwolke** oder **außerhalb**?
  - handelt es sich bei einem (neuen) Punkt um ein **typisches** oder **untypisches** Beispiel?
- Es wird eine **geschlossen** Kurve oder Umhüllende der blauen Punkte gesucht

- Im Kontext der **Fehlerdiagnose** hat man meist folgende Situation
  - **viele** Daten für das **gesunde** System (blau), **wenige** oder keine Daten für die **Fehlerfälle** (rot)
  - die Begriffe Anomalie oder Neuigkeits-Erkennung passen hier besonders gut

- Das Problem ist **schlecht gestellt** (*ill-posed*), insb. im Fall ausschließlicher Daten einer Klasse
- Man sucht eine **Umhüllende** (grün) für eine Punktwolke aber das ganze Raum wäre eine Umhüllende
- Daher muss man das Problem **weiter spezifizieren**:
  - **wie weit** darf/soll die Umhüllende von der Punktwolke **entfernt** sein?
  - **wie viele Punkte** (bzw. deren Anteil) dürfen **außerhalb** der Umhüllenden liegen (rote)?

## Anwendungsfelder: Erkennung von

1. **Ausreißern**
   - Ausreißer entstehen durch **Fehler** in den Daten (z.B. Kippen eines Bits im A/D-Wandler) oder durch **Störung/Rauschen** mit einer langsam abklingenden Wahrscheinlichkeitsdichte (sog. ***fat tail distribution***)
   - viele anspruchsvolle Methoden reagieren **empfindlich** auf Ausreißer → diese sollten zuvor entfernt werden
   - 1-Klassen-Klassifikation kann Ausreißer erkennen (rote Datenpunkte); anschließend können sie entfernt werden
2. **Extrapolation**
   - hier gibt es typischerweise nur eine Klasse, die Trainingsdatenpunkte
   - bei datengetriebenen Verfahren möchte man zwischen **Interpolation** und **Extrapolation** unterscheiden können
   - **Interpolation**: Methode/Modell meist zuverlässig
   - **Extrapolation**: Methode/Modell typischerweise unzuverlässig, da Information fehlt
   - mit dem 1-Klassen-Klassifikator kann man das Gebiet beschreiben, in dem die **Trainingsdaten** lagen
   - damit lässt sich ein **Extrapolation-Detektor** bauen
   - die Entfernung zur Klassengrenze gibt an, **wie weit** ein (neuer) Punkt im Interpolations- bzw. Extrapolationsbereich liegt



## Anwendungsfelder: Erkennung von

3. **Fehlern**
   - in der Fehlerdiagnose sind meist die Daten für den **fehlerfreien** Fall **reichlich** vorhanden und **billig** zu erheben
   - für die **Fehlerfälle** sind Daten sehr **knapp** und **teuer**, manchmal **gar nicht** zu erheben
   - daher liegt oft ein **1-Klassen-Klassifikationsproblem** vor:
     Punkte im Inneren der Umhüllenden → fehlerfrei
     Punkte im außerhalb der Umhüllenden → Fehler

     Eigentlich ist das falsch. Außerhalb sag erstmal nur: Das ist neu – bislang nicht vorgekommen! → Es könnte ein Fehler sein.
   - wenn **Trainingsdaten** für die **Fehlerfälle** vorliegen, kann man um **jeden Fehler** einen **eigenen** 1-Klassen-Klassifikator / Umhüllende legen und somit die verschiedenen Fehler unterscheiden und diagnostizieren
4. **Charakteristik eines Datensatzes**
   - oft ist man in Machine Learning an einem **Vergleich zwischen 2 Datensätzen** interessiert, z.B. den Trainings- und Validierungsdaten oder gemessene Trainingsdaten 1 von heute und Trainingsdaten 2 von gestern
   - um einen dieser Datensätze kann man einen 1-Klassen-Klassifikator legen und dann prüfen wie viele Datenpunkte des zweiten Datensatzes darin liegen
   - damit kann man Ähnlichkeiten in der Raumabdeckung herausfinden und damit prognostizieren,
     - wie gut und repräsentativ der Validierungsfehler die Modellqualität beschreibt
     - wie ähnlich sich Modelle sein werden, die mit Trainingsdaten 1 oder Trainingsdaten 2 trainiert wurden

## Verschiedene Ideen zur 1-Klassen-Klassifikation

- **2-Klassen-Klassifikation** (→ Kapitel 2) mit normalen Daten (blau, Klasse 1) und (künstlichen) Ausreißer-Daten (rot, Klasse 2)
  - es werden künstlich Ausreißer-Daten erzeugt, indem die Originaldaten „verrauscht“ werden
  - künstliche Daten oft sehr wenig typisch für echte Ausreißer (andere Verteilung)
  - sehr unrobust, sehr empfindlich bzgl. Fluch der Dimensionalität (*curse of dimensionality*)
- **Generative Modelle** beschreiben mittels Vorwissen die Wahrscheinlichkeitsdichte der normalen Daten (blau, Klasse 1) und der Ausreißer-Daten (rot, Klasse 2)
  - die generativen Modelle werden mit Hilfe der Messdaten abgeglichen / trainiert
- **Dichteschätzer** (→ Kapitel 7) schätzt die Wahrscheinlichkeitsdichte der normalen Daten (blau, Klasse 1) und der Ausreißer-Daten (rot, Klasse 2)
  - es können a-priori-Wahrscheinlichkeiten für beide Klassen angenommen werden, was die nach dem Satz von Bayes die Grenze/Schwelle zwischen beiden Klassen beeinflusst
  - Dichteschätzung funktioniert nur im Niederdimensionalen zuverlässig und benötigt relativ viele Daten – diese sind für die Ausreißer-Klasse meist nicht vorhanden
- **Randmethoden** schätzen nicht die gesamte Wahrscheinlichkeitsdichte sondern fokussieren sich auf die Randbereiche (Außen)
  - z.B. ***K*-Zentren-Methode**: Es werden *K* Kugeln mit identischem Radius so platziert (Zentren werden optimiert), dass die maximale Distanz der Nächste-Nachbar-Entfernung von Daten zu jeder Kugel minimiert wird, d.h. die Kugeln die Datenwolke möglichst kompakt (mit minimalem Gesamt-Volumen) abdecken

## Optimierung der Hyperparameter mittels Leave-One-Out-Fehler

- Um die **Hyperparameter** (z.B. Standardabweichung $\sigma$ und/oder Regularisierungsstärke $\lambda$) gut einzustellen bzw. zu **optimieren** gibt es 2 Möglichkeiten (→ Kapitel 2.6)
  - **Validierungsdaten** → zusätzliche Daten notwendig
  - **Kreuzvaldierung** → zusätzlicher Rechenaufwand notwendig

  Wegen des **Overfitting-Effekts** kann man die **Trainingsdaten nicht** dazu heranziehen; auf Trainingsdaten sähe immer das flexibelste Modell wie das „beste“ aus, weil der Varianz-Fehler nicht berücksichtigt wird!
- Die extremste Form der Kreuzvalidierung ist die **Leave-One-Out**-Methode
- Für die **meisten** Klassifikatoren muss Leave-One-Out wirklich so durchgeführt werden:
  - lasse Datenpunkt $i$ aus den Trainingsdaten weg und trainiere den Klassifikator mit den übrigen $N$-1 Datenpunkten
  - berechne den Fehler auf dem weggelassenen Datenpunkt $i$ (entspricht einer Validierung auf diesem Datenpunkt!)
  - wiederhole diesen Vorgang für $i$ = 1, 2, ..., $N$ und summiere die Fehlerquadrate $e^2(i)$ oder -beträge $|e(i)|$ o.ä.

  Diese Prozedur ist **super rechenaufwändig**, insb. für große $N$
- Für Klassifikatoren, die **linear-in-den-Parametern** sind, wie z.B. lineare oder polynomiale Modelle, RBF-Netze, Gauß-Prozess-Modelle, Least-Squares Support Vector Machines, können die Parameter mit **Least-Squares** geschätzt werden. Dann gilt für den **Leave-One-Out-Fehler** oder den Generalized Cross Validation Fehler:
  - es gibt eine **geschlossen analytische** Lösung (Formel!) → **Kapitel 2.6**, obige Prozedur ist nicht notwendig
  - dies ermöglicht ein **effizientes** und **mächtiges Tuning** der **Hyperparameter**!

## Beispiel RBF-Klassifikator mit 2 Hyperparametern (Sigma $\sigma$ und Lambda $\lambda$)

**Funktionsweise des Klassifikators (weitere Details siehe „Neuronale Netze“-Vorlesung):**

- Auf **jedem** der *N* Datenpunkte wird eine Gauß-Glocke mit **Standardabweichung** $\sigma$ platziert.
- Der Wert der **Klassifikationsfunktion** *y* berechnet sich aus der **gewichteten Summe** der Funktionswerte aller Gauß-Glocken.
- Die Gewichte *w* werden beim **Training mit regularisiertem Least-Squares** (Ridge-Regression) bestimmt. Dies ist nötig, weil die Zahl der Gewichte gleich der Zahl der Datenpunkte ist und daher Overfitting-Gefahr besteht, bzw. numerische Probleme auftreten können. Die **Regularisierungsstärke** wird mit dem Parameter $\lambda$ eingestellt.
- Für alle Trainingsdatenpunkte soll der Wert der Klassifikationsfunktion 1 betragen.
- Die **Klassifikationsgrenze** $y_{lim}$ wird etwas kleiner als 1 gewählt (z.B. 0,95).
  **Gute Strategie für Wahl der Hyperparameter:**
  - durch die Least Squares Schätzung kann der **Leave-One-Out-Fehler (LOOE)** mit wenig zusätzlichem Rechenaufwand ermittelt werden.
  - je kleiner der LOOE, um so größer ist die Robustheit des Klassifikators: Klassifikationsgrenze hängt nicht von jedem einzelnen Datenpunkt ab (bei LOOE = 0).
  - es wird das **kleinste $\sigma$ und größte $\lambda$** gesucht für den der kleinste LOOE (möglichst Null) auftritt. Damit erhält man das engste (kleines $\sigma$) und numerisch robusteste (größtes $\lambda$) Klassifikationsergebnis mit minimalem LOOE.

$$\hat{y}(l) = \sum_{i=1}^{N} w_i \cdot e^{-\frac{(\underline{u}(l)-\underline{u}(i))^T(\underline{u}(l)-\underline{u}(i))}{2\sigma^2}}$$

$$\underline{w} = \left(\underline{X}^T\underline{X} + \lambda\underline{I}\right)^{-1}\underline{X}^T\underline{y}$$

$$X_{ij} = e^{-\frac{(\underline{u}(i)-\underline{u}(j))^T(\underline{u}(i)-\underline{u}(j))}{2\sigma^2}}$$

$$e_{\mathrm{LOO}}(i) = \frac{y(i)-\hat{y}(i)}{1-s_{ii}}$$

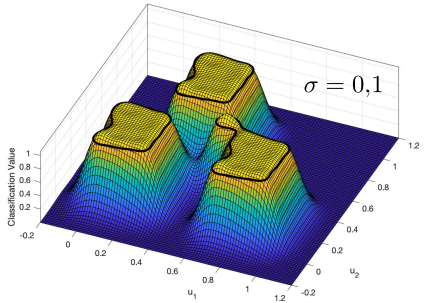
$$\underline{S} = \underline{X}\left(\underline{X}^T\underline{X} + \lambda\underline{I}\right)^{-1}\underline{X}^T$$

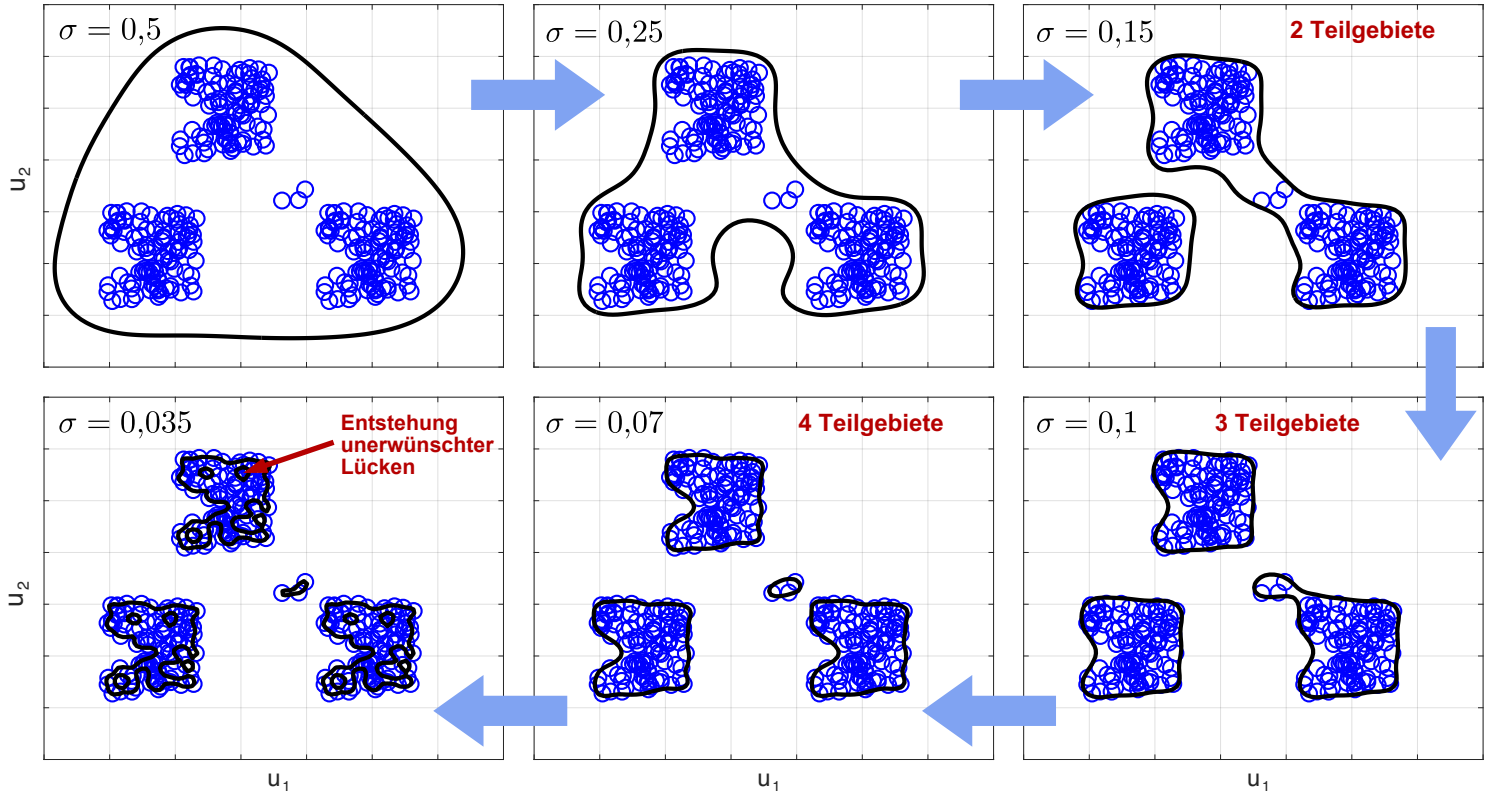
$$\hat{y}_{\mathrm{LOO}}(i) = \hat{y}(i) - e_{\mathrm{LOO}}(i)$$

## Beispiel RBF-Klassifikator mit 2 Hyperparametern – Einfluss von $\sigma$ ($\lambda = 10^{-3}$, konstant)

**Abnehmendes $\sigma$:**

- Flexiblere Grenze.
- Grenze näher an Randpunkten.
- Zunehmende Zahl an Teilgebieten.
- Unerwünschte Lücken.
- Zunehmende LOO Fehler (geringere Robustheit).

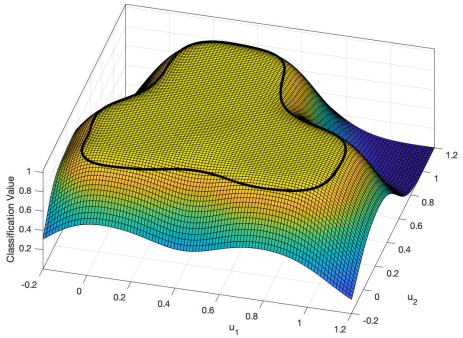


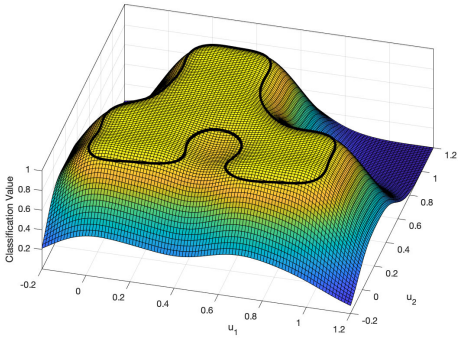



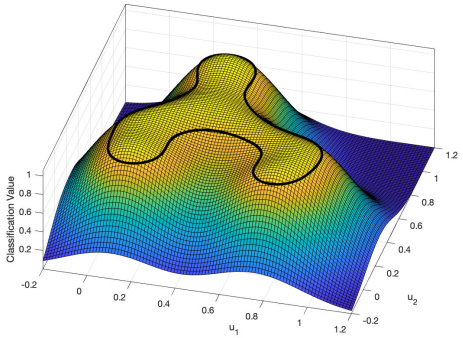
## Beispiel RBF-Klassifikator mit 2 Hyperparametern – Einfluss von $\lambda$ ($\sigma$ = 0,25, konstant)

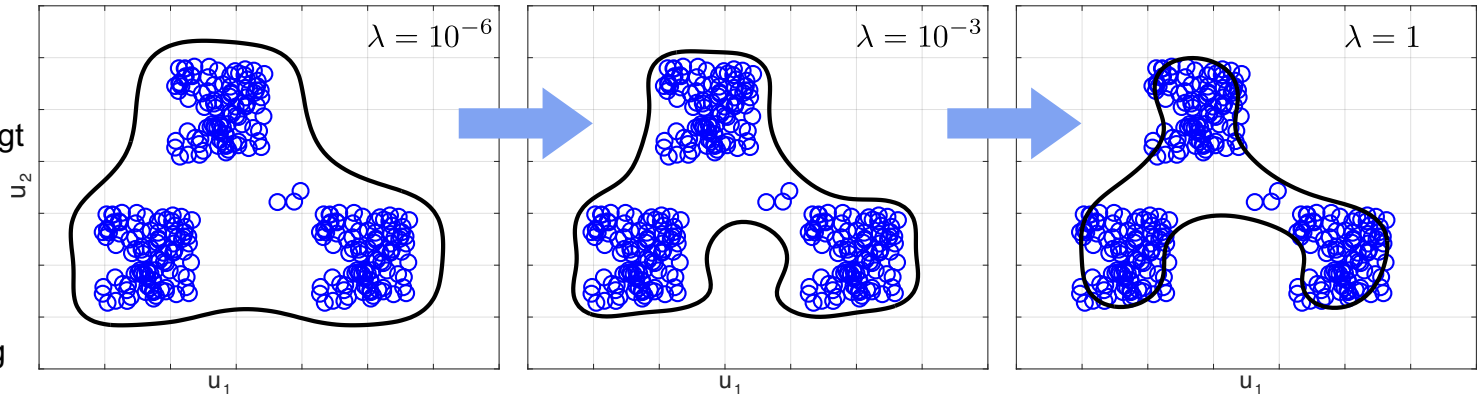
**Größeres $\lambda$:**

- Besser konditionierte Matrix bei Least Squares Berechnung.
- Gewichte *w* gehen zunehmend gegen Null, d.h.:
  - Bereiche ohne Punkte fallen zunehmend unter Klassifikations-grenze (keine Auswirkung auf Trainings-fehler). Grenze schmiegt sich an.
  - Bei großen $\lambda$ kommt es zu Trainingsfehlern.
- Das optimale $\lambda$ für minimalen LOOE ist abhängig von $\sigma$.



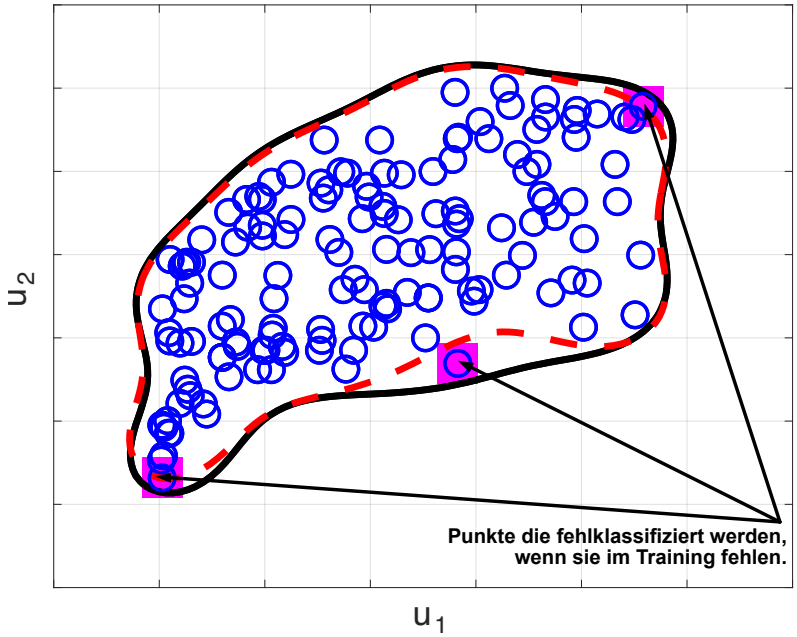



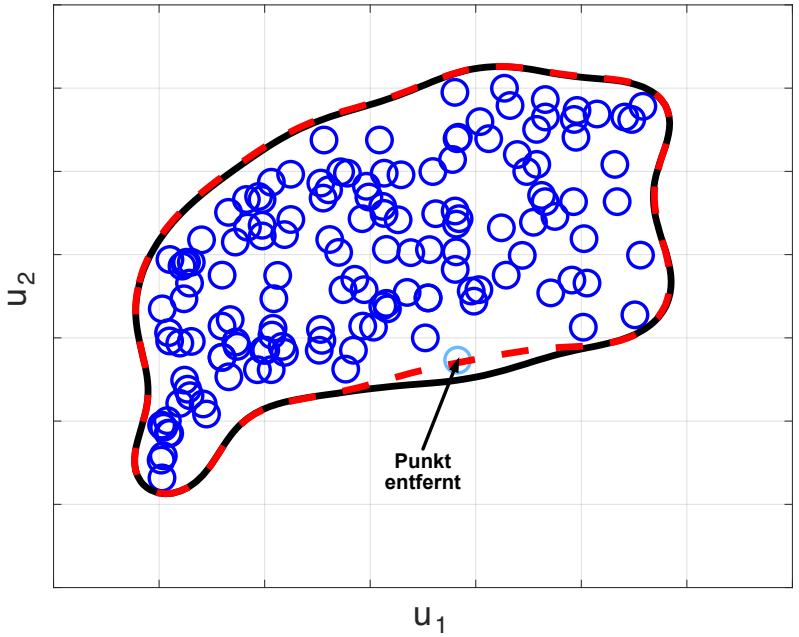
## Beispiel RBF-Klassifikator mit 2 Hyperparametern – Optimale Parameterwahl (Leave-One-Out-Fehler = Null)

- Rote gestrichelte Linie: Klassifikationsgrenze, wenn magenta markierte Punkte aus Trainingsdaten entfernt würden.
- Bei optimaler Parameterwahl kann jeder beliebige Punkt im Training entfernt werden und würde dennoch korrekt klassifiziert.

**Nicht optimale Wahl der Hyperparameter LOOE = 3**




**Optimale Wahl der Hyperparameter LOOE = 0**



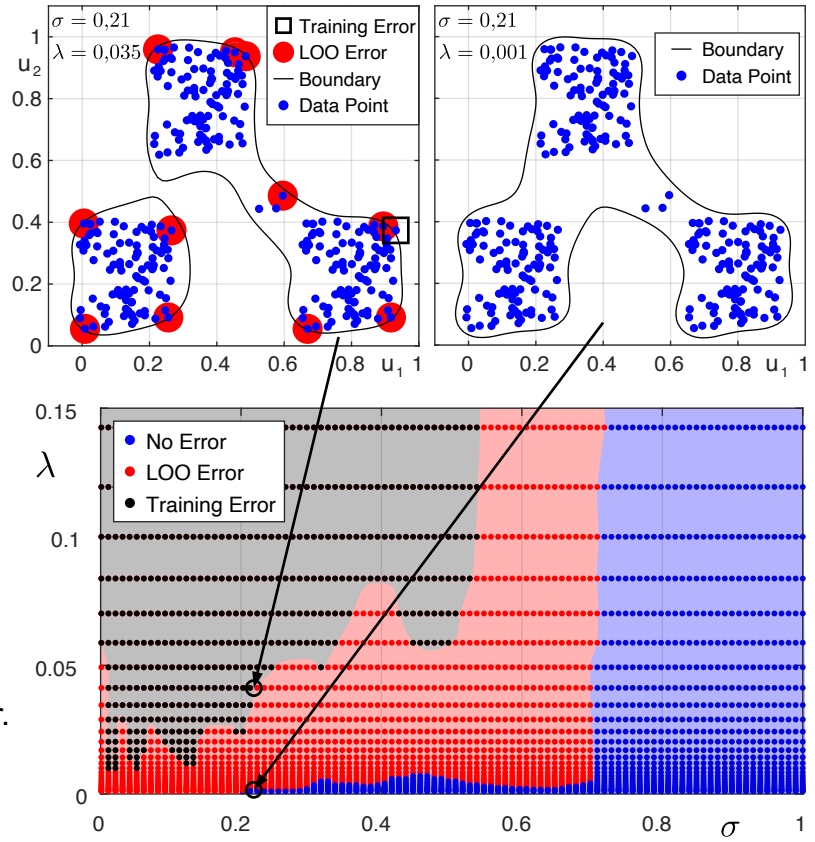
## Beispiel RBF-Klassifikator mit 2 Hyperparametern – Strategie der LOO-Optimierung

**Vorgehensweise:**

1. Trainieren einer moderaten Zahl von Klassifikatoren mit unterschiedlichen ($\sigma$, $\lambda$)-Kombinationen (i.d.R. < 5000)
   ⇒ **Rechenaufwand** minimieren.
2. Auswahl aller Ergebnisse ohne (oder mit minimalem) Trainings- und LOO-Fehler ⇒ **robuste** Klassifikationsgrenze.
3. Von den Ergebnissen aus 2, Auswahl derjenigen, mit dem **kleinsten $\sigma$**.
   ⇒ möglichst **enge** Klassifikationsgrenze.
4. Von den Ergebnissen aus 3, Auswahl desjenigen, mit dem **größten $\lambda$**.
   ⇒ **numerisch** möglichst gutes Ergebnis ohne **Overfitting**.

**Beispiel (Bilder rechts):**

- Fehlerarten für Kombinationen von 150 $\sigma$ und 30 $\lambda$-Werten (Bild unten).
- Optimales Ergebnis nach obigen Kriterien im Bild oben rechts.
- Bild oben links zeigt ein Beispiel mit 11 LOO- und einem Trainingsfehler.
- Bei großem $\sigma$ ist die Wahl von $\lambda$ für eine robuste Klassifikation (blauer Bereich) quasi egal, allerdings ist die Klassifikationsgrenze in diesem Fällen sehr weit von den Punkten entfernt.



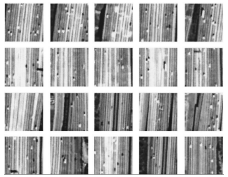
# 4. Nearest Neighbor

# Inhalt

## Nearest Neighbor Methoden (NN Methoden)

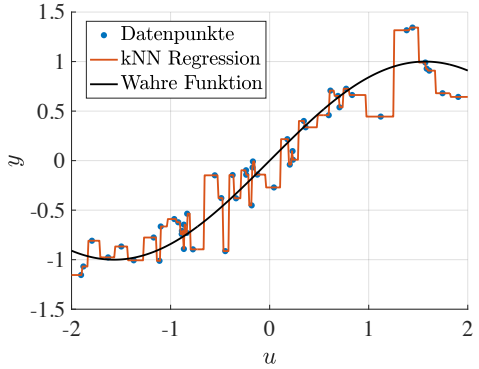
- Nearest Neighbor Methoden sind sowohl im **überwachten** als auch im **unüberwachten** Lernen weit verbreitet
- Sie werden für viele verschiedene Klassifikations- und Regressionsprobleme erfolgreich eingesetzt:
  - Mustererkennung, Textkategorisierung, Objekterkennung, etc.
  - Festlegen von **geometrischen Größen** (Kernelbreite) für RBF-Netze
  - Datensatz: **Abstand** jedes Punktes zu seinem Nearest Neighbor → Kapitel 8
  - Interpolation bei Kennlinien mit einer Konstanten kann ebenfalls als Nearest Neighbor Vorgehen angesehen werden
- Abhängig von der Aufgabenstellung können Daten des **Eingangsraums** oder des **Eingangsraums + Ausgangsdaten (*product space*)** verwendet werden
- Sehr guter Überblick über verschiedenste Nearest Neighbor Methoden:
  N. Bhatia: Survey of Nearest Neighbor Techniques,
  IJCSIS Vol. 8, No.2, 2010

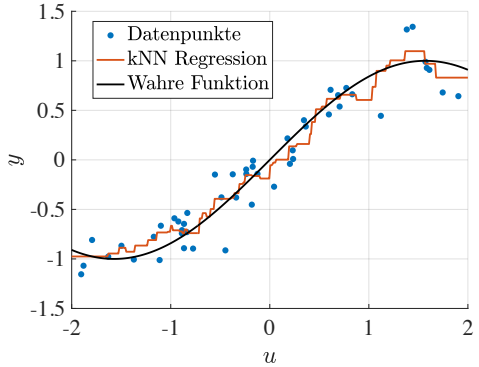
Handgeschriebene Zahlen



Satellitenbilder

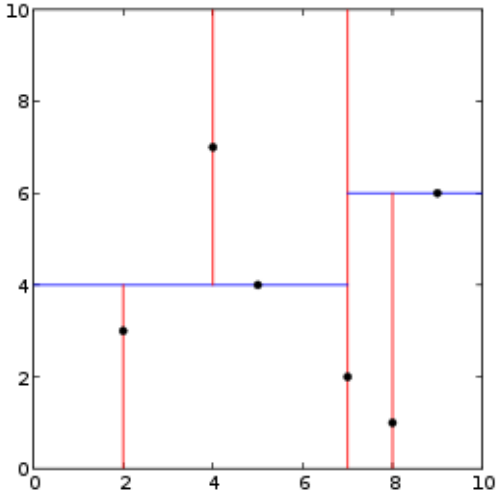
**Beispiel**: Regression mit $k = 1$ (oben) und $k = 5$ (unten) nächsten Nachbarn



## Training und Verwendung von *k*-NN Methoden

- **„Training“**: Nur die **Abspeicherung** der Daten. Im Gegensatz zu den meisten Modellen werden keine Parameter geschätzt bzw. optimiert
- Durch eine geschickte Organisation der Trainingsdaten in sog. **k-d-Bäumen** (Bild) lässt sich die Auswertegeschwindigkeit (Abfrage (*query*)) steigern
- Mittels **Validierung** muss ein guter Wert für die ***Anzahl* der berücksichtigten Nachbarn *k*** gefunden werden:
  *k* = 1: Nur der nächste Nachbar, rausch- und overfittingempfindlich
  *k* mittel: **Balanciert** Bias (*k* zu groß) und Varianz (*k* zu klein)
  *k* sehr groß: Auflösung stark reduziert, lokale Charakteristik geht verloren



Quelle: https://de.wikipedia.org/wiki/K-d-Baum

## MATLAB Funktion

- `[idx, d] = knnsearch(X,Y)`
  Findet die Nachbarn in Daten `X` bezüglich der Anfragedaten in `Y` und gibt sie im Indexvektor `idx` zurück; Distanzen sind in `d`
- `knnsearch(X,Y, 'k',3, 'distance', 'minkowski', 'p',5)`
  Findet die 3 nächsten Nachbarn. Benutzt die Minkowski-Distanz zwischen je 2 Datenpunkten, die in `X` und `Y` stehen:

$$d(\underline{x}(i), \underline{y}(j)) = \left( \sum_{d=1}^{n} |x_d(i) - y_d(j)|^p \right)^{\frac{1}{p}}$$

## Nearest Neighbor Methoden

- Nearest Neighbor Methoden basieren auf der Auswertung von **benachbarten Datenpunkten**
- Es werden die Datenpunkte ausgewertet, die den minimalen Abstand zu einem **angefragten Punkt** (***query***) besitzen
  - Die **Anzahl** der ausgewerteten Datenpunkte ***k*** wird vom Nutzer vorgegeben
  - Es können verschiedene **Abstandsmaße** verwendet werden (z.B. euklidischer Abstand, Manhattan-Abstand)
- Datenpunkte: $\underline{u}(i), i = 1, 2, ..., N$

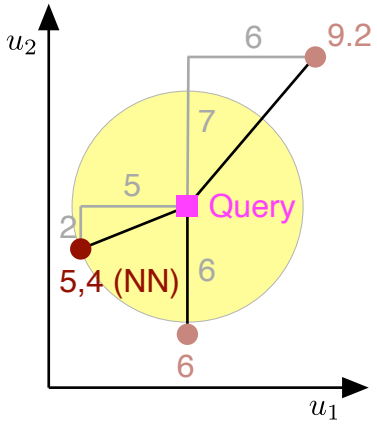
mit $\underline{u}(i) = \begin{bmatrix} u_1(i) \\ u_2(i) \\ \vdots \\ u_n(i) \end{bmatrix}$

$$d_{\text{Euklid}}(\underline{u}(i), \underline{u}(j)) = \sqrt{\sum_{d=1}^{n} (u_d(i) - u_d(j))^2}$$

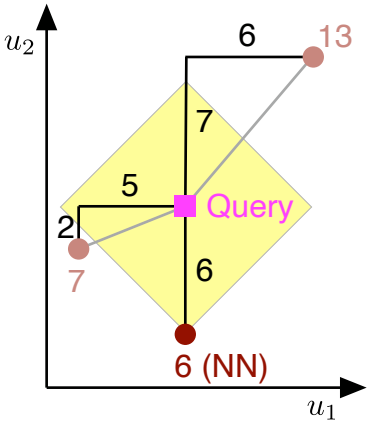
$$d_{\text{Manhattan}}(\underline{u}(i), \underline{u}(j)) = \sum_{d=1}^{n} \underbrace{|u_d(i) - u_d(j)|}_{\text{alle Punktpaare}}$$

läuft über alle Dimensionen

Euklidische Distanz




Manhattan-Distanz



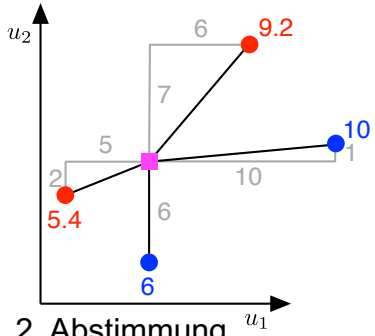
## Grundidee Nearest Neighbor Klassifikation

- Instanzbasiertes Lernen (direkt aus Daten, **keine Schätzung**): Es wird kein Modell gelernt
- Die Klasse eines neuen Datenpunkts wird bestimmt, indem:
  1. die *k* nächsten Datenpunkte, deren Klassen bekannt sind, mit einem Distanzmaß bestimmt werden, und
  2. die am häufigsten vertretene Klasse der *k* nächsten Datenpunkte für den neuen Datenpunkt ausgewählt wird.
- Als **Distanzmaß** wird in den meisten Fällen der euklidische Abstand oder der Manhattan-Abstand gewählt
- Da nahe Nachbarn relevanter sind als entfernte, kann man auch eine **Gewichtung der Nachbarn** vornehmen, z.B. antiproportional zum Abstand
- Das Grundprinzip beruht auf der Auswertung einer vorher festgelegte Anzahl an Datenpunkten *k*

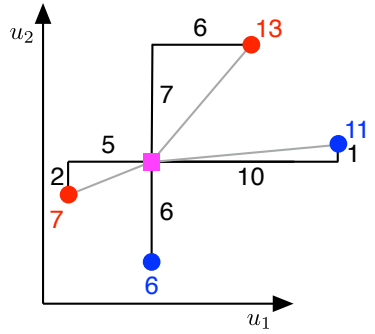
**Beispiel für Anzahl der Nachbarn *k* = 3**

1. Distanz

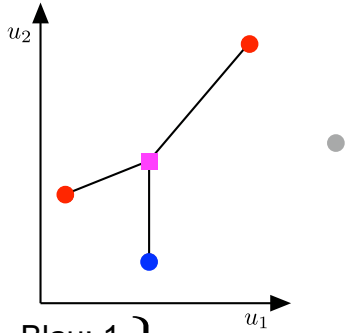
Euklidische Distanz



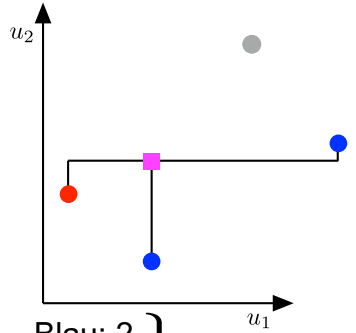

Alternativ: Manhattan-Distanz




2. Abstimmung




Blau: 1
Rot: 2
} Neuer Punkt: Rot



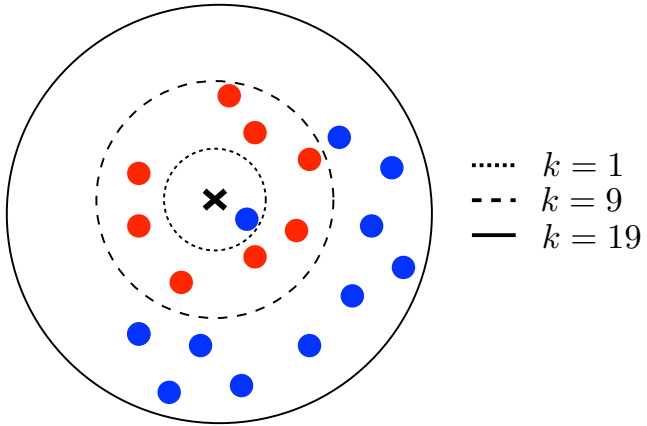

Blau: 2
Rot: 1
} Neuer Punkt: Blau

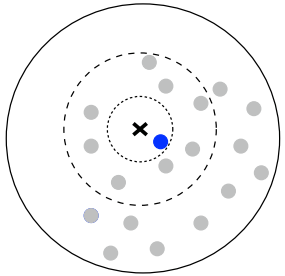
# 4.3 Anzahl an Nachbarn *k*

## Wie groß sollte die Anzahl an Nachbarn *k* sein?

- ***k* zu klein**: hohe Empfindlichkeit bei Ausreißern
  → Zu lokal, problematisch bei verrauschten Daten
- ***k* zu groß**: zu viele Nachbarn werden in den Entscheidungsprozess einbezogen
  → Zu global, lokales Verhalten verschwimmt
- Ein **mittleres *k*** sorgt für die höchste Klassifikationsgüte

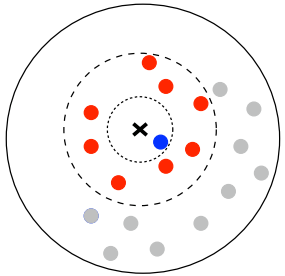
**Beispiel**: Nächste Nachbarn für verschiedene Werte von *k*



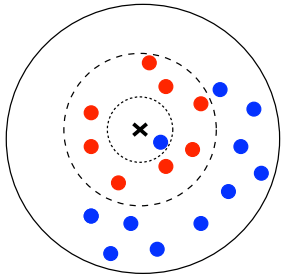
*k* = 1

Blau: 1
Rot: 0
} Neuer Punkt: Blau



*k* = 9

Blau: 1
Rot: 8
} Neuer Punkt: Rot



*k* = 19

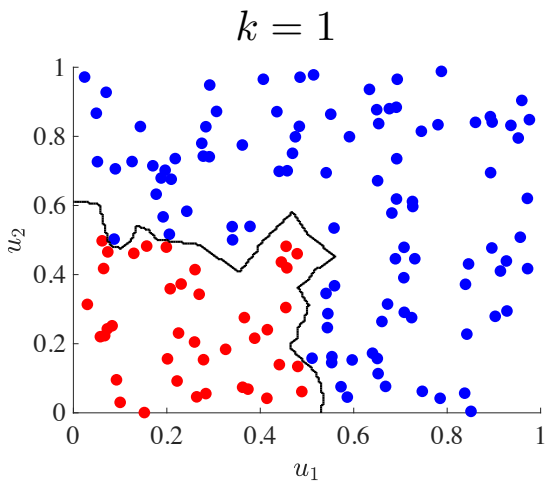
Blau: 11
Rot: 8
} Neuer Punkt: Blau

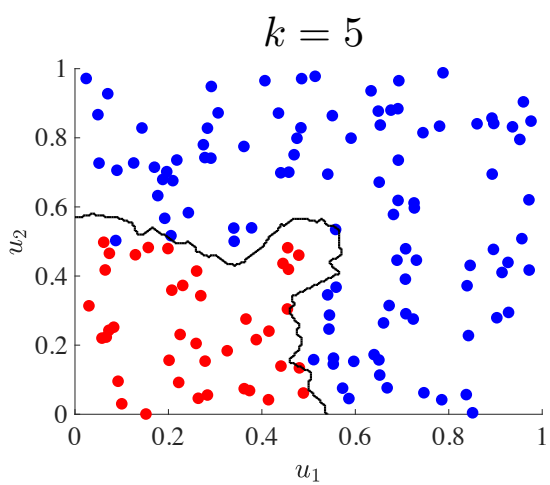
## *k*-NN mit unverrauschten Daten

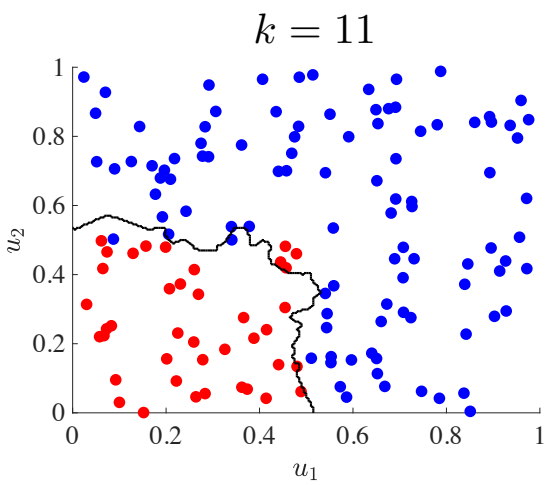
- Beispiel: *N* = 140 Punkte, alle Punkte im linken unteren Quadranten gehören zu Klasse 1 (rot), der Rest zu Klasse 2 (blau)
- Bei unverrauschten Daten ergeben alle NN-Klassifikatoren ähnlich gute Ergebnisse
- Je **größer *k*** wird, desto **weicher** (***smoother***) wird die Klassifikationsgrenze

**Schwarze Linie** = Klassifikationsgrenze

$\sigma_{\text{noise}} = 0$



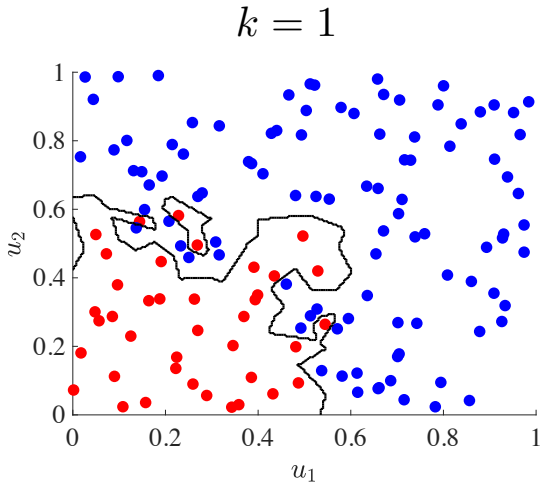



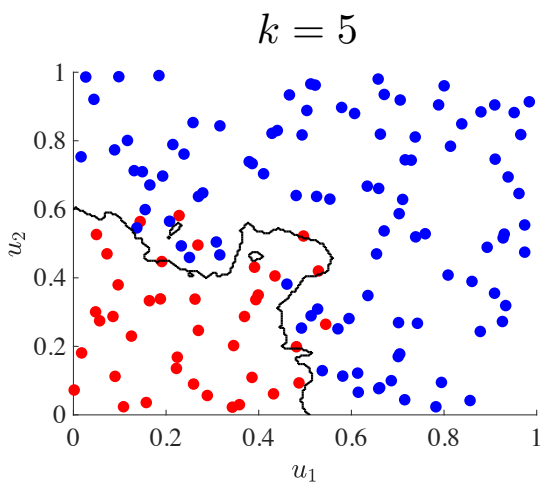
## *k*-NN mit verrauschten Daten

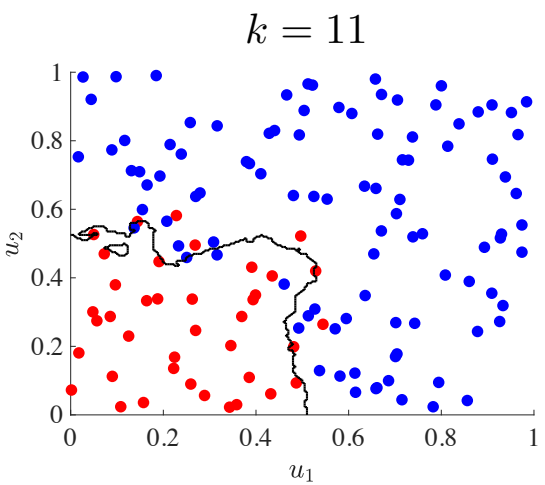
- Beispiel: *N* = 140 Punkte, alle Punkte im linken unteren Quadranten gehören zu Klasse 1, der Rest zu Klasse 2
- Bei verrauschten Daten reagieren die NN-Klassifikatoren mit zu kleinem *k* empfindlich
- Wegen des Rausches ist der Fall *k* = 1 unbrauchbar (Overfitting!)

**Wie werden die Daten erstellt?**
Gleichverteilte Datenpunkte sampeln
Punkte $u_1 < 0.5$ und $u_1 < 0.5$ → Rot
Punkte $u_1 > 0.5$ und $u_1 > 0.5$ → Blau
Anschließend Datenpunkte verrauschen
Rauschen: Normalverteilt mit $\sigma_{\text{noise}}$

$\sigma_{\text{noise}} = 0.05$



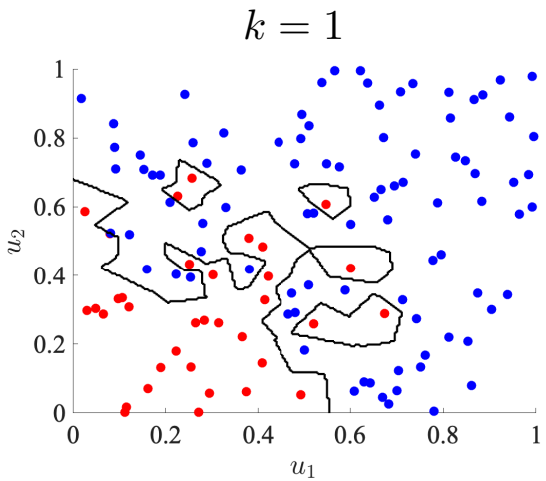



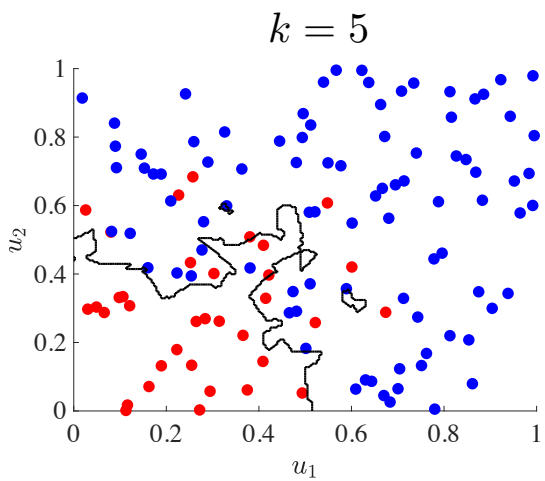



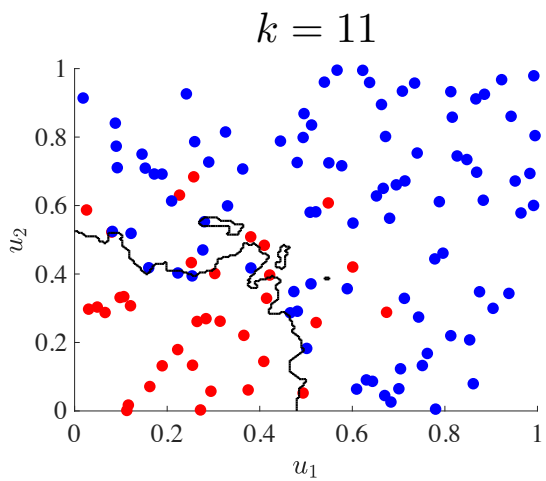
## *k*-NN mit verrauschten Daten

- Beispiel: *N* = 140 Punkte, alle Punkte im linken unteren Quadranten gehören zu Klasse 1, der Rest zu Klasse 2
- Bei verrauschten Daten reagieren die NN-Klassifikatoren mit zu kleinem *k* empfindlich
- Wegen des starken Rausches ist der Fall *k* = 1 und *k* = 5 unbrauchbar (Overfitting!); selbst *k* =11 ist schlecht

$\sigma_{\text{noise}} = 0.1$



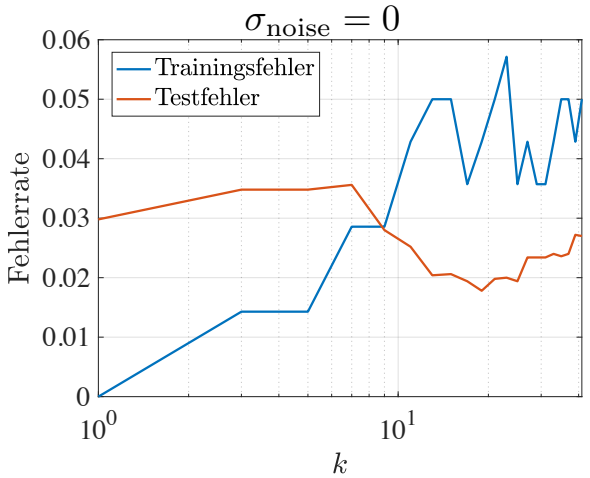



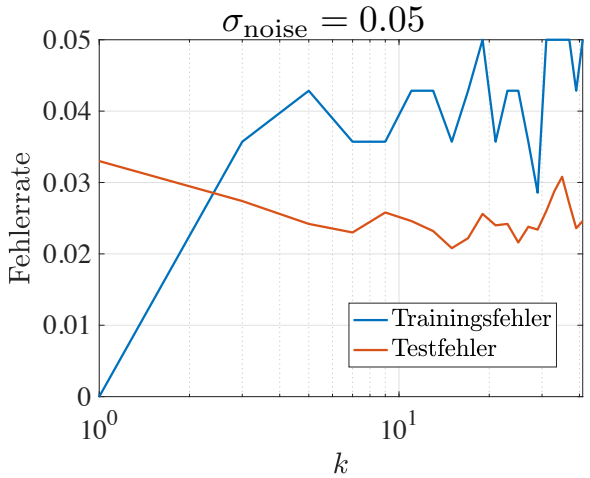



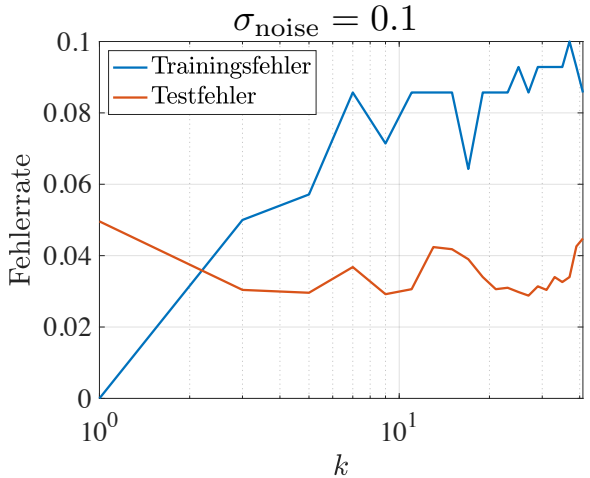
## *k*-NN mit verrauschten Daten

- Beispiel: *N* = 140 Punkte, verrauscht mit verschiedenen Rauschniveaus (0, 0.05, 0.1)
- **Fehlerrate über Anzahl der Nachbarn** *k* = 1, 2, ..., 40:
  Wie entwickeln sich Training- und Testfehler?
- Je **stärker** das **Rauschen**, desto besser die Performance von NN-Klassifikatoren mit **größerem *k*** auf den Testdaten
- Trainingsfehler: kein guter Prädiktor für Performance des Modells auf Testdaten

**Auswertung**
Trainingsdatensatz: *N* = 140 Datenpunkte
Testen der Klassifikatoren auf *N* = 5.000 Testdatenpunkten
Auswertungskriterium: **Fehlerrate** = Verhältnis falsch klassifizierter Daten zur Gesamtzahl der Daten *N*



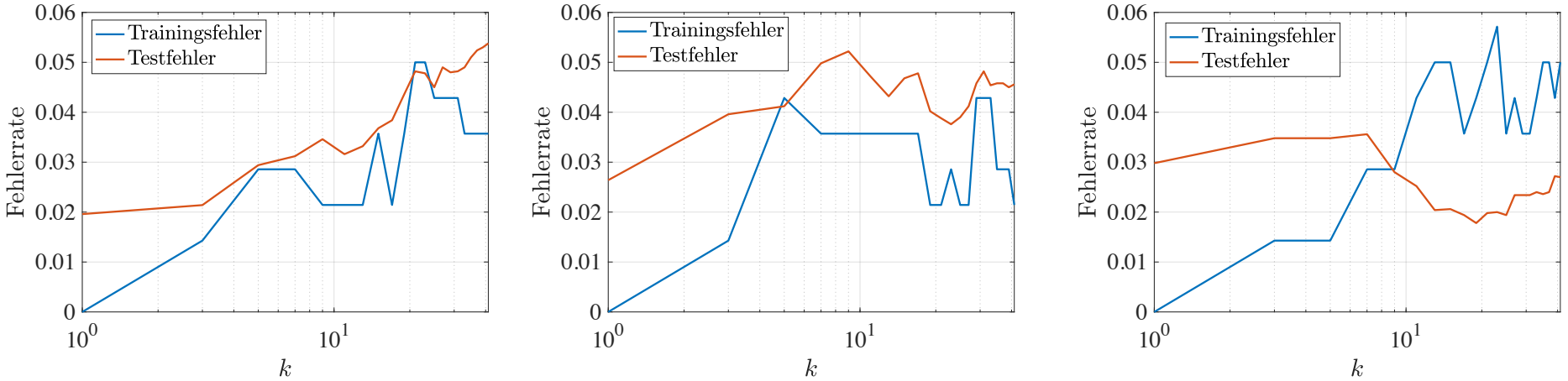



## Beispiele für *k*-NN mit unverrauschten Daten

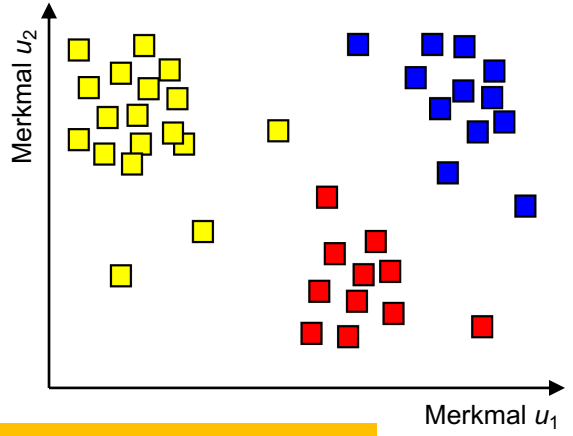
- 3 unterschiedliche, zufällige Punktverteilungen (andere exakte Lage der Punkte aber immer in den korrekten Quadranten)
- Verhalten des Trainingsfehlers ist ähnlich
- Verhalten des **Testfehlers** kann sehr **unterschiedlich** sein.
  Je nach Abstand der einzelnen Datenpunkte zur Klassifikationsgrenze und der Punktdichte kann $k = 1$ optimal sein (links) aber auch stärkere Generalisierung ($k = 20$) vorteilhaft sein (rechts).

$\sigma_{\text{noise}} = 0$



## Was ist Clustering?

- Methode aus dem **unüberwachten Lernen**
  → Keine Information über Klassen bzw. Ausgangswert der Datenpunkte
- "**Clustering**" heißt „**Gruppenerkennung**", d.h. man versucht Datenpunkte „gleicher Art" jeweils einer Gruppe (Cluster) zuzuordnen.
- „Gleiche Art" bezieht sich typischerweise auf **geometrische Merkmale**, man sucht z.B. nach
  - Kreise/Ellipsen bzw. Hyper-Sphären (höherdim.) [**Standard**]
  - Linien bzw. Hyper-Ebenen
- **Ziel**:
  - Punkte **innerhalb** eines Clusters: **Nahe** beisammen
  - Punkte in **unterschiedlichen** Clustern: **Weit** entfernt
- Was „nahe" und „weit" bedeutet wird über das **Abstandsmaß** festgelegt, z.B.
  - Euklidische Abstand (L2-Norm) $d = \sqrt{u_1^2 + u_2^2 + \ldots + u_n^2}$
  - Mahalanobis-Abstand (verallgemeinerte L2-Norm) $d = \sqrt{\underline{u}^T \underline{M}\, \underline{u}}$ $\quad \underline{M} = \underline{\Sigma}^{-1}$
  - Manhattan-Abstand (L1-Norm) $d = |u_1| + |u_2| + \ldots + |u_n|$




**Häufig verwendete Clusteringverfahren:**
- *K*-Means Clustering (Kreise/Ellipsen)
- Gustafson-Kessel (schräg liegende Ellipsen)
- DBScan (dichtebasiert, Ausreißer möglich)

inverse Kovarianzmatrix

beschreibt die Form einer Ellipse

$$\underline{u}^T = [u_1 \quad u_2 \quad \cdots \quad u_n]$$

# 4.4 Nearest Neighbor Clustering

## Grundidee Nearest Neighbor Clustering

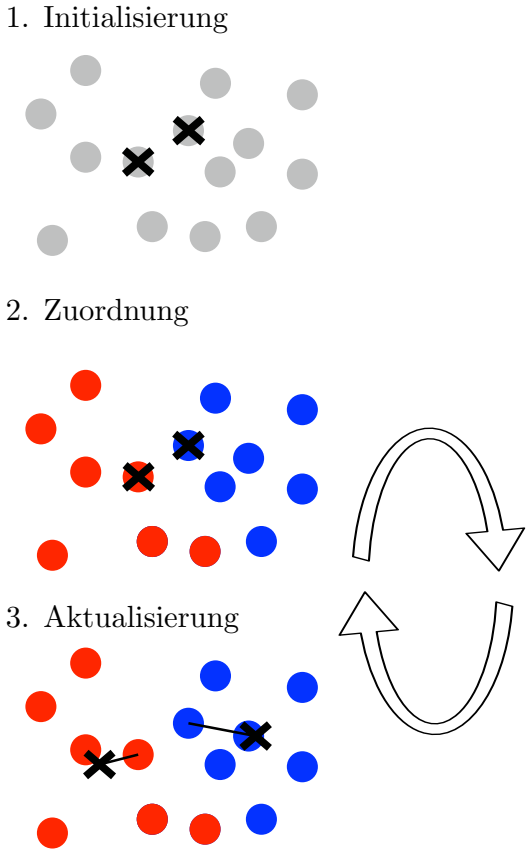
- **Ziel:** *N* Datenpunkte in *K* Cluster aufzuteilen, wobei jeder Datenpunkt dem Cluster mit dem nächsten Clusterzentrum zugeordnet wird
- Generell bei allen Clusteringverfahren zu beachten
  - Skalierung der Achsen bzw. Dimensionen
  - Durch unterschiedliche Skalierungen je Achse, kann jede Achse ein anderes Gewicht bekommen

wird in vielen Clusteringverfahren genutzt, z.B. K-means

- Typisches Nearest Neighbor Clusteringverfahren: **Lloyd-Algorithmus**:

  1. **Initialisierung**
     Wähle *K* zufällige Datenpunkte aus Datensatz als Clusterzentren aus
  2. **Zuordnung**
     Ordne jeden Datenpunkt dem Cluster zu, bei dem die **Cluster-Varianz** (d.h. die Varianz aller Datenpunkte in einem Cluster) **am wenigsten erhöht** wird. Das ist der Cluster, zu dem der Datenpunkt „am besten passt“.
  3. **Aktualisierung**
     Berechne die **Zentren** der Cluster als **Mittelwert/Schwerpunkt** der zugeordneten Datenpunkte **neu**.

- Schritte 2-3 werden so oft wiederholt, bis sich an der Zuordnung nichts mehr ändert.



## Vor- und Nachteile von Nearest Neighbor Methoden

### Pro

- **Intuitiv** und **einfach** zu verstehen und zu implementieren
- Es sind **keine Strukturannahmen** nötig
- Kann für Klassifikations- und Regressionsprobleme genutzt werden
- Nur **ein Hyperparameter**: Die Anzahl der Nachbarn ***k***
- Es werden nur die Datenpunkte benötigt und direkt ausgewertet:
  → **Kein zeitaufwändiges Training** erforderlich
- In vielen Anwendungen: Hohe Klassifikationsgenauigkeit

### Kontra

- **Rechenintensiv bei Auswertung**: Anfrage eines neuen Datenpunktes erfordert Nachbarn finden von allen Datenpunkten
- Deutlicher **Anstieg der Komplexität** mit steigender Anzahl an Eingängen/**Dimensionen**
- Alle **Datenpunkte** müssen **gespeichert** werden
- **Wahl von *k* problematisch** bei sehr unterschiedlichen Klassengrößen (unbalancierter Datensatz)

# 5. Klassifikationsbäume (CART)

# Inhalt



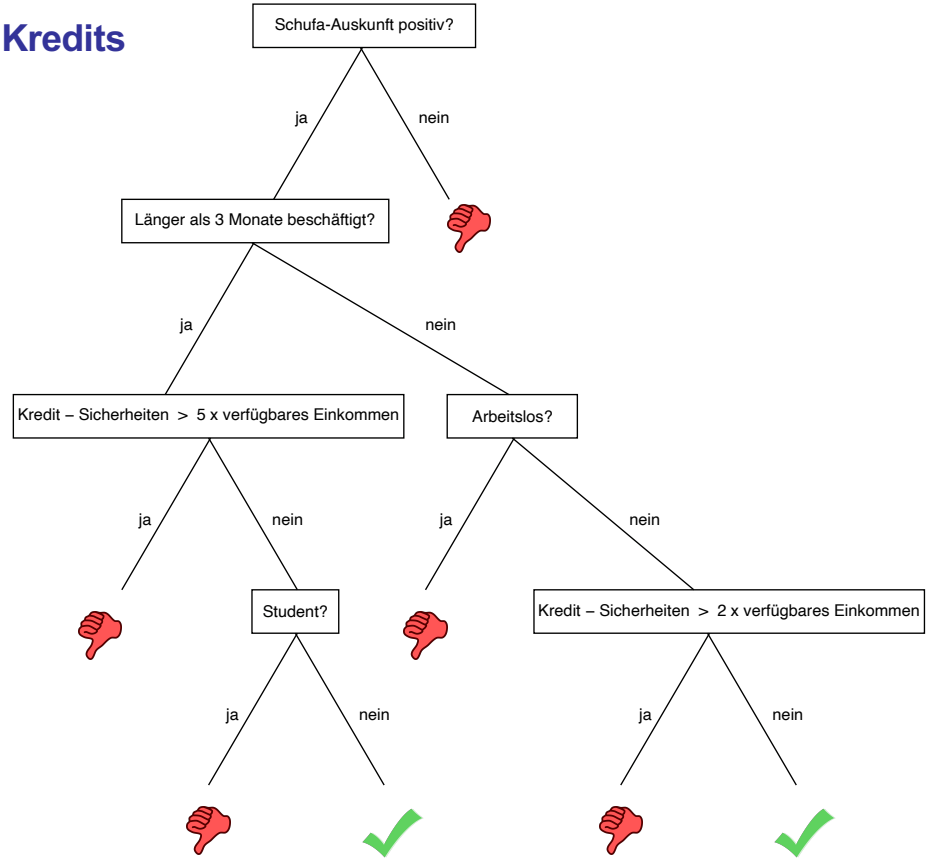
**Empfohlene Videos zum Thema Klassifikationsbäume:**

- Videos zu Statistical Learning von T. Hastie und R. Tibshirani (Stanford)
- https://www.youtube.com/watch?v=XvdLKUOldkE&list=PLAOUn-KLSAVOqj5TG8E1HTb8Txwxe6OtV

## Beispiel: Klassifikationsbaum zur Bewilligung eines Kredits

- Nachvollziehbar.
- Keine Black-Box!
- Gute Schnittstelle zum fachlichen Experten.
- Liefert Klassifikation plus Begründung:
  „Abgelehnt, weil weniger als 3 Monate beschäftigt
  und Kredit − Sicherheiten > 2 x verfügbares Einkommen“
- Können aus Daten gelernt werden → z.B. CART.

- Baum besteht aus Knoten:
  - Wurzel = 1. Knoten (oben),
  - Blätter bzw. Terminalknoten (unten)
    mit Zuordnung der Klasse




Beispiel entnommen aus https://www.cs.uni-potsdam.de/ml/teaching/ws12/ml/Entscheidungsbaeume.pdf
„Maschinelles Lernen Entscheidungsbäume“
Universität Potsdam, Institut für Informatik
Lehrstuhl Maschinelles Lernen

# 5.1 Idee der Bäume

## Bezeichnungen

- Es gibt mehrere ähnliche Algorithmen zur Konstruktion von **Klassifikationsbäumen** (oder **Entscheidungsbäumen**) aus Daten.
- In der Statistik etabliert hat sich:
  **CART** = Classification and Regression Tree von *Leo Breiman*
  Buch: Breiman, L., Friedman, J., Olshen, R., Stone, C. (1984). CART. Classification and Regression Trees, Taylor & Francis, Routledge.
- Aus der Informatik stammen:
  **ID3** = Iterative Dichotomiser (to separate into two parts) 3, **C4.5**, **C5.0** von *Ross Quinlan*
  Buch: Quinlan, R. (1993). C4.5: Programs for Machine Learning, Morgan Kaufmann, San Mateo.

## Grundidee: Teile und Herrsche (*Divide and Conquer*)

- Teile das ursprüngliche Klassifikationsproblem in 2 kleinere Klassifikationsprobleme und löse beide unabhängig voneinander und führe diese Ergebnisse zusammen.
- Diese Teilungsstrategie kann auf beide Teil-Klassifikationsprobleme wieder angewandt werden (Rekursion!). Auf diese Art wird das Problem rekursiv in immer kleinere Teilprobleme zerlegt.
- Diese Unterteilung findet spätestens dann ein Ende (Terminierungskriterium), wenn die gewünschte Anzahl an maximalen Teilungen erreicht ist oder wenn die Lösung trivial ist. Das ist z.B. der Fall, wenn:
  - Das Teilproblem enthält nur noch Datenpunkte aus einer Klasse → Klassifiziere diese Klasse
  - Das Teilproblem enthält keine Datenpunkte mehr → Stoppe schon zuvor

# 5.1 Idee der Bäume

## Problembeschreibung

- Jedes Merkmal/Feature/Eingang entspricht einer Dimension/Achse.
- Jedem Datenpunkt ist eine Klasse zugeordnet.
- Typischerweise existieren 2 Klassen, z.B.
  - defekt / in Ordnung
  - kritisch / unkritisch
- Alle Datenpunkte liegen dann in einer Box bzw. einem (Hyper)-Quader, der oft auf den Einheitswürfel $[0\ 1]^n$ normiert wird, mit $n$ = Dimension.
- Ziel ist es, ein Gebiet (oder mehrere Gebiete) zu finden, das je eine Klasse repräsentiert → **Klassifikator.**
- Typischerweise entsteht eine Grenze (**Klassifikationsgrenze**) zwischen dem Gebiet für Klasse 1 und dem Gebiet für Klasse 2.

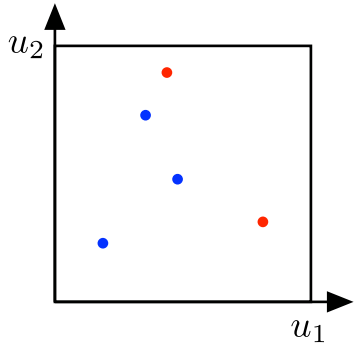
## Leistungsfähigkeit

- Wichtig sind „gute“ Merkmale, in denen möglichst viel Klasseninformation steckt.
- Bäume erlauben die Konstruktion eines besonders einfachen und gut **interpretierbaren** Klassifikators.
- Ein Baum kann auch in Form einer Hierarchie von **Regeln** aufgeschrieben werden.

## Vorgehensweise

- Der Merkmalsraum soll durch **Splits** (Schnitte) so unterteilt werden, dass die Datenpunkte unterschiedlicher Klassen getrennt werden.
- In jedem Schritt oder **Iteration** der **Baumkonstruktion** wird nur ein Split durchgeführt (**binärer Baum**).
- Die sich ergebenden Gebiete werden immer weiter **rekursiv** verfeinert. D.h. die Vorgehensweise für das **Ganze**, wird immer wieder auf seine **Teile** (Gebiete) angewandt.
- Typischerweise beschränkt man sich auf **achsenorthogonale** Splits, d.h. es wird eine Dimension/Achse unterteilt.
- Es gibt **2 Freiheitsgrade** für einen Split:
  - Dimension
  - Position/Stelle
- Generell möchte man für den **Split** wählen
  - schlechtestes Gebiet, weil dort eine Verbesserung am wichtigsten ist
  - beste Dimension
  - beste Position
- Das ist ein gieriger (*greedy*) Ansatz, weil in jedem Schritt das **kurzfristig** beste gemacht wird, ohne das langfristige **Endergebnis** zu betrachten.

Beispiel (künstliche Daten)




2 Dimensionen, 5 Datenpunkte

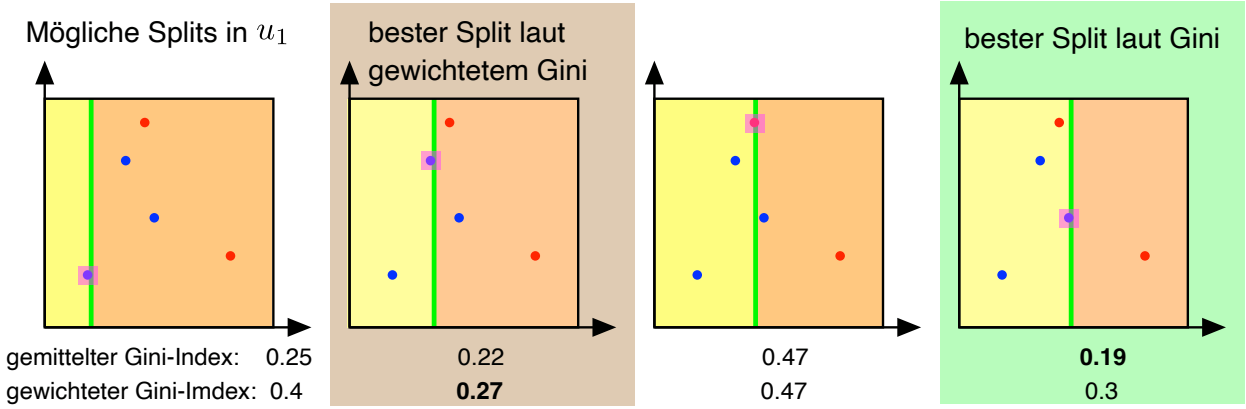
- 3 x Klasse 1
- 2 x Klasse 2

Split

Gini-Index und gemittelter/gewichteter Gini-Index berechnen die Qualität eines Splits → Abschnitt 5.2

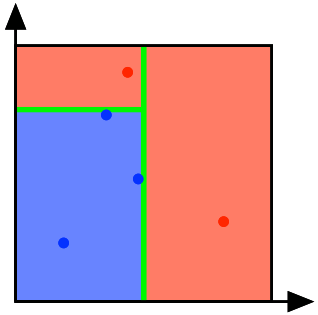
## 1. Split

- Split bei den **Koordinaten** der Datenpunkte ■
- Das zu splittende Gebiet wird in 2 Teile gesplittet
- Hier: 5 Datenpunkte
  → 4 mögliche Splits pro Dimension, (beidseitig müssen Punkte liegen)
- Es ergeben sich folgende mögliche Aufteilungen der 5 Datenpunkte (gelb/orange):
  1-4
  2-3
  3-2
  4-1
- Ein Split alleine kann das Klassifikationsproblem nicht lösen
  → Besten Split durchführen und weitermachen...

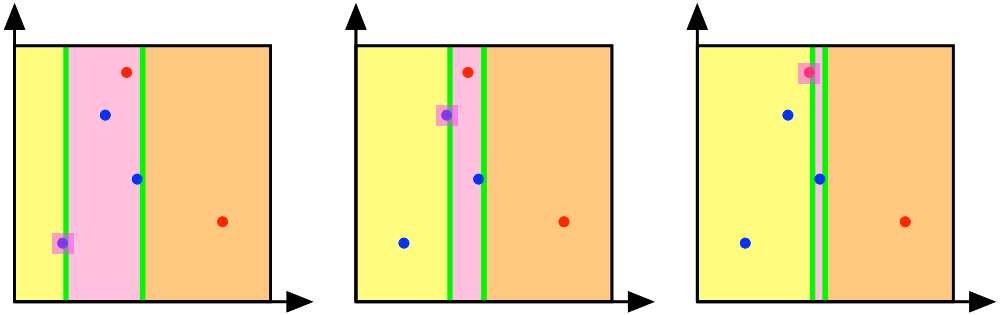


## Auswahl Split 4. oben (grün): Jetzt 2. Split

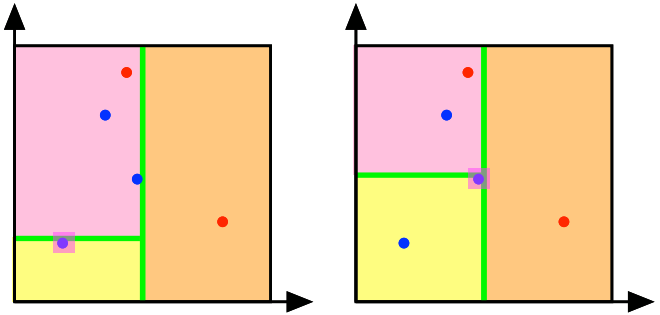
- Rechtes (oranges) Gebiet ist schon perfekt oder rein (*pure*). Es enthält nur Punkte von einer Klasse (rote). → Teilung dieses Gebietes ist beendet.
- Linkes (gelbes) Gebiet enthält noch Punkte beider Klassen. Es muss weiter geteilt werden.
- Diese Vorgehensweise wird fortgeführt (weitere Splits) bis nur noch reine Gebiete existieren.
- Dann werden alle Gebiete entweder Klasse 1 (blau) oder Klasse 2 (rot) zugeordnet.
- In diesem Beispiel ergibt sich schon nach 2 Splits ein Klassifikator mit Trainingsfehler = 0:



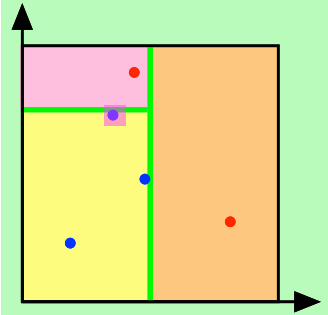
Mögliche Splits in $u_1$



Mögliche Splits in $u_2$



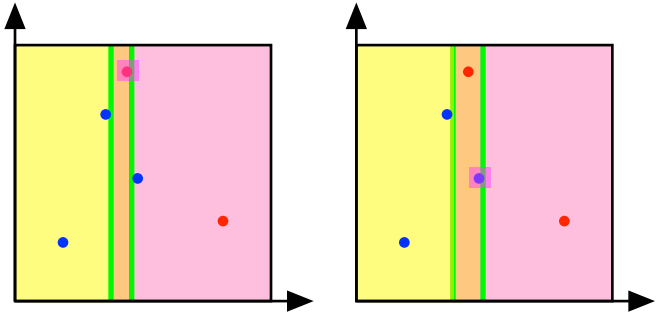
bester Split



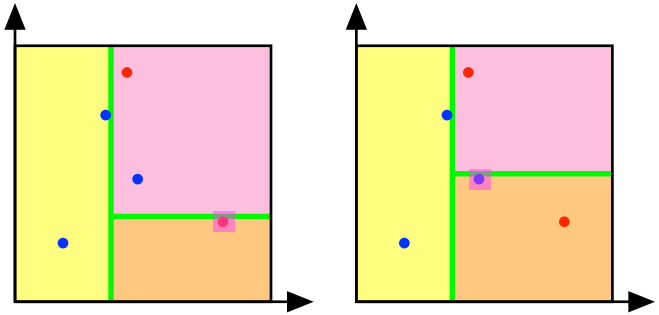
## Oder Auswahl Split 2. oben (braun): Jetzt 2. Split

- Linkes (gelbes) Gebiet ist schon perfekt oder rein (*pure*). Es enthält nur Punkte von einer Klasse (blaue). → Teilung dieses Gebietes ist beendet.
- Rechtes (oranges) Gebiet enthält noch Punkte beider Klassen. Es muss weiter geteilt werden.
- Diese Vorgehensweise wird fortgeführt (weitere Splits) bis nur noch reine Gebiete existieren.
- In diesem Beispiel ergibt sich nach 2 Splits noch kein Klassifikator mit Trainingsfehler = 0.
- Es muss weiter geteilt werden...

Mögliche Splits in $u_1$

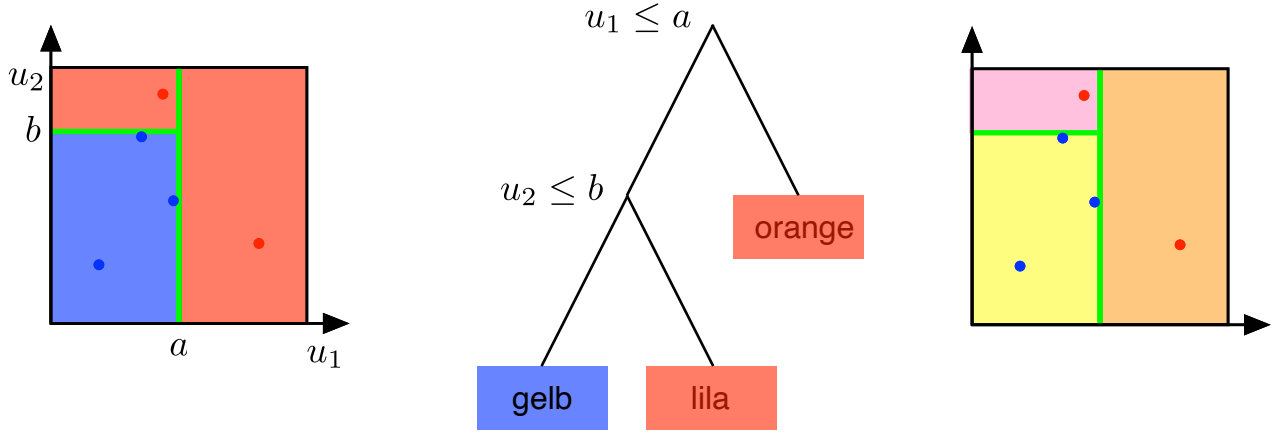


Mögliche Splits in $u_2$



## Interpretation in Form von Regeln oder als Baum

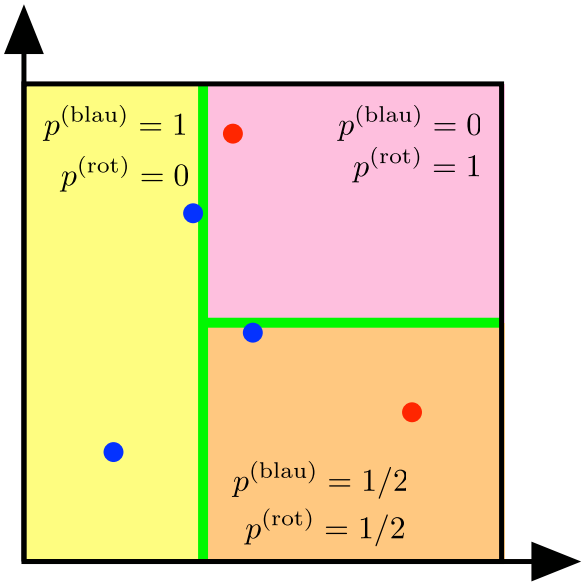
- Ein solcher Klassifikator kann als **Hierarchie von Regeln** oder als **Baum** interpretiert werden.
- Zwischen dem Klassifikator, den Regeln und dem Baum existiert eine **1:1-Beziehung**.
- Die Regeln und der Baum sind insbesondere sehr hilfreich, wenn mehr als 2-3 Dimensionen/Merkmale vorliegen. Dann kann die **Partitionierung** (Verteilung der blauen und roten Gebiete) nicht mehr einfach visualisiert werden.
- Dies ist eine sehr natürliche Art, Zusammenhänge darzustellen, wie sie z.B. bei Ärzten, Juristen oder auch Kfz-Mechatronikern üblich ist oder auch in Expertensystemen kodiert wird.



## Blätter des Baums

- Jedem Blatt muss eine Klasse zugeordnet werden
- 2 Möglichkeiten:
  1. Blatt ist rein (pure), d.h. enthält nur Datenpunkte *einer* Klasse
     → diese Klasse wird dem Blatt zugeordnet bzw.
     für diese Klasse ist die Wahrscheinlichkeit $p = 1$, für die andere Klasse $p = 0$
  2. Blatt ist nicht rein, d.h. enthält Datenpunkte *beider* Klassen,
     z.B. $N_1$ Punkte der Klasse 1, $N_2$ Punkte der Klasse 2
     → die Mehrheits-Klasse (mit mehr Datenpunkten) wird dem Blatt zugeordnet bzw.
     es ergeben sich die Wahrscheinlichkeiten, siehe Bild rechts:

$$p_1 = \frac{N_1}{N_1 + N_2} \qquad p_2 = \frac{N_2}{N_1 + N_2} = 1 - p_1$$

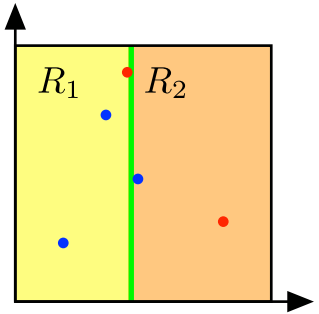



- Wenn die Bäume ausgewachsen sind, sind alle Blätter vom Typ 1.
- Wenn die Bäume nicht ausgewachsen sind, existieren auch Blätter vom Typ 2; oft sind die meisten Blätter von Typ 2. Dann ist es sehr sinnvoll, den Blättern Kassen-Wahrscheinlichkeiten zuzuordnen.
  → Bei Ensemble-Methoden (Aggregation mehrerer Bäume, siehe Abschnitt 5.6) werden diese Wahrscheinlichkeiten von Klassifikationsbäumen genauso behandelt, wie die reellen Funktionswerte bei Regressionsbäumen, d.h. sie können gemittelt werden.

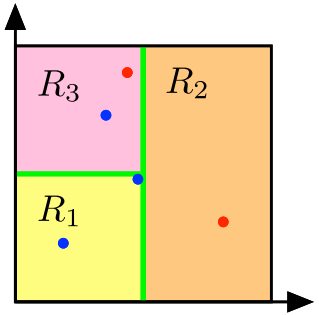
## Wie entscheidet man *genau*, was der „beste“ Split ist?

- Wir brauchen eine exakte mathematische Formulierung für die Qualität eines Splits, um anhand eines numerischen Werts den besten aussuchen zu können.
- Es gibt 2 Arten von Bäumen:
  - Regressionsbäume: Eine Möglichkeit, Funktionen zu approximieren. Typische Maße für die Qualität sind: Summe der Fehlerquadrate oder –beträge o.ä., wobei der Fehler die Differenz zwischen Prozess und Modell ist.
  - Klassifikationsbäume: Darum geht es hier.
- Bei Klassifikationsbäumen berechnet man zunächst für jedes Gebiet/Hyper-Quader $R_i$ den Anteil/Prozentsatz der Datenpunkte einer Klasse:

$$p_i^{(\text{blau})} = \text{Anteil der blauen Datenpunkte in Region } R_i$$
$$p_i^{(\text{rot})} = \text{Anteil der roten Datenpunkte in Region } R_i$$




$$p_1^{(\text{blau})} = 2/3 \qquad p_1^{(\text{rot})} = 1/3$$
$$p_2^{(\text{blau})} = 1/2 \qquad p_2^{(\text{rot})} = 1/2$$




$$p_1^{(\text{blau})} = 1 \qquad p_1^{(\text{rot})} = 0$$
$$p_2^{(\text{blau})} = 0 \qquad p_2^{(\text{rot})} = 1$$
$$p_3^{(\text{blau})} = 1/2 \qquad p_3^{(\text{rot})} = 1/2$$

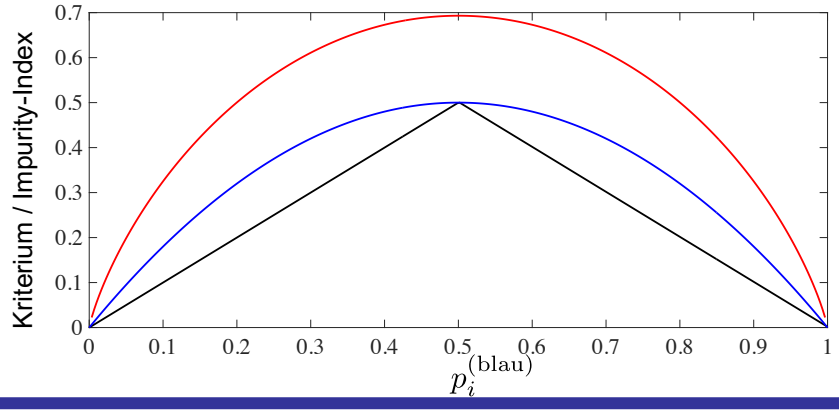
# 5.2 Klassifikationskriterien

**Reine Gebiete $R_i$ mit $p_i = 0$ und $p_i = 1$ werden nicht weiter geteilt**

**Diese Anteile $p^{(blau)}$ und $p^{(rot)}$ werden zu einem Skalar zur eindeutigen Bewertung zusammengefasst. Für Klassifikationsbäume ist eines der folgenden Kriterien üblich:**

- **Missklassifikationsfehler**: $1 - \max(p_i^{(\text{blau})}, p_i^{(\text{rot})}) = 1 - \max(p_i^{(\text{blau})}, 1 - p_i^{(\text{blau})}) = 1 - \max(p_i^{(\text{rot})}, 1 - p_i^{(\text{rot})})$
- **Gini-Index**: $2p_i^{(\text{blau})} p_i^{(\text{rot})} = 2p_i^{(\text{blau})}(1 - p_i^{(\text{blau})})$
- **Kreuzentropie**: $-p_i^{(\text{blau})} \log\, p_i^{(\text{blau})} - p_i^{(\text{rot})} \log\, p_i^{(\text{rot})} = -p_i^{(\text{blau})} \log\, p_i^{(\text{blau})} (1 - p_i^{(\text{blau})}) \log\, (1 - p_i^{(\text{blau})})$

- Diese Kriterien können als ein **Maß für Unordnung (*impurity*)** verstanden werden.
  Für reine Gebiete sind sie = 0,
  für komplette Vermischung ½:½ sind sie maximal.
- Gini-Index und Kreuzentropie sind **differenzierbar** und eignen sich daher auch gut für **kontinuierliche** numerische Optimierung.
- All das kann auch auf mehr als 2 Klassen erweitert werden!



## Entscheidung für besten Split

- Der beste Split soll das gewählte **Kriterium** (Missklassifikationsfehler, Gini-Index, Kreuzentropie) am meisten verbessern (= reduzieren), weil damit **maximal Ordnung** erzeugt wird, also blaue und rote Datenpunkte bestmöglich getrennt werden.
- Für die Gesamtordnung ist aber nicht nur das gewählte Kriterium für beide Kindknoten nach dem Split wichtig, sondern auch **wie viele Datenpunkte** in der jeweiligen Region überhaupt betroffen sind.
- Daher wird für jeden potentiellen Split berechnet:

Kriterium für Kindknoten 1: $J_1$ Größe (Anz. Datenpunkte) des Kindknoten 1: $N_1$

Kriterium für Kindknoten 2: $J_2$ Größe (Anz. Datenpunkte) des Kindknoten 2: $N_2$

**Kriterium für den Split:** **Mittleres Kriterium:**

$$J = \frac{J_1 + J_2}{2}$$

**Gewichtetes Kriterium:**

$$J = \frac{N_1 J_1 + N_2 J_2}{N_1 + N_2}$$

- Durch die Gewichtung mit der Knotengröße wird das Maß der Unordnung $J_i$ mit dessen Bedeutung für die Region verknüpft. Ein reiner Knoten trägt ja nur wenig für den Klassifikationserfolg bei, wenn nur bspw. 2 Datenpunkte darin enthalten sind. Hingegen kann schon ein bisschen mehr Ordnung (etwas bessere Trennung von blau und rot) sehr bedeutsam sein, wenn bspw. 100.000 Punkte betroffen sind.

# 5.3 Pruning

## Optimale Baumgröße bzw. -komplexität

- Bis jetzt: Bei der Konstruktion des Baums wird so lange gesplittet bis jeder Knoten rein (pure) ist. Dann wird dieser Knoten zu einem **Blatt** und das Wachstum (*growing*) findet in diesem Unterzweig ein Ende.
- Der dabei konstruierte Baum erreicht immer Trainingsfehler = 0, ist typischerweise aber **riesig** (sehr viele Knoten).
- Aus zwei Gründen möchte man solche Bäume meist nicht verwenden:
  - Sie sind schwer zu verstehen/interpretieren, aufwändig zu verarbeiten (hohe Rechnerressourcen) und unhandlich.
  - Sie stellen meist ein starkes **Overfitting** dar. → Siehe Nelles: „Neuronale Netze und Fuzzy-Systeme“.

  Der erste Grund ist eher praktischer Natur. Der zweite Grund ist sehr viel grundsätzlicher und zerstört meist die Qualität des Klassifikators. Die Klassifikationsgrenze wird meist sehr „ausgefranst“ und sehr empfindlich bzgl. verrauschter Daten.

## Finden der optimalen Komplexität

- Das ist wahrscheinlich das wichtigste Problem aller datengetriebenen Verfahren.
- Es wird durch das Bias/Varianz-Dilemma beschrieben.
- Es gibt ein ganzes Arsenal an Methoden, dieses Problem anzugehen.

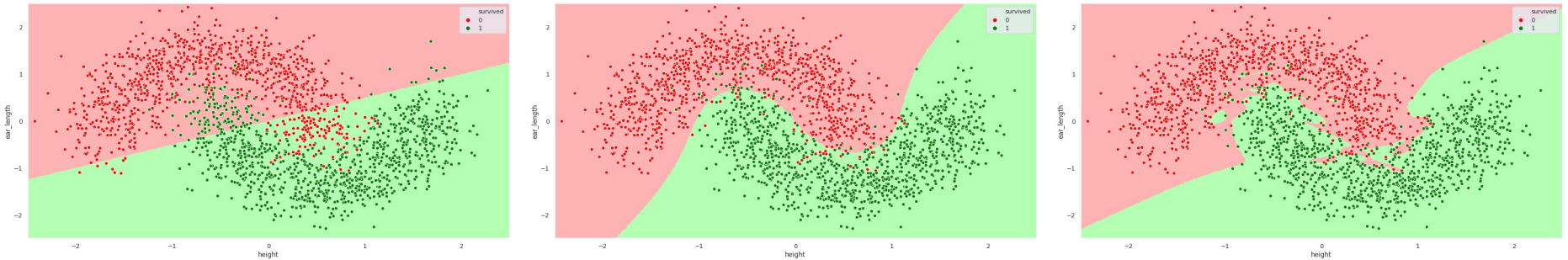
## Regularisierung

- Verfahren, welche die Flexibilität einschränken und damit die **Varianz reduzieren**, dabei aber die Struktur unverändert lassen.

## Overfitting, Underfitting und ein guter Tradeoff

- Um Overfitting zu vermeiden/reduzieren, werden voll **ausgewachsene** Bäume in der Regel **nicht verwendet**.
- Stattdessen versucht man einen Baum geeigneter Komplexität zu finden, also einen guten **Mittelweg** zwischen
  - zu einfach (hoher Bias, niedrige Varianz): kann die Klassen nicht gut trennen, zu wenig flexible Klassifikationsgrenze
  - zu komplex (niedriger Bias, hohe Varianz): erzielt auf Trainingsdaten gute Ergebnisse, versagt aber bei neuen (zuvor ungesehenen Testdaten) wegen der zu flexiblen Klassifikationsgrenze, die schnörkelig ist und unrobust auf Rauschen/Ausreißer reagiert

**Underfitting** | **guter Tradeoff** | **Overfitting**




Bilder aus https://anarthal.github.io/kernel/posts/underfitting-overfitting/

## Guter Komplexitäts-Tradeoff bei Bäumen

- Regularisierung (**implizite** Varianzreduktion) kommt typischerweise mittels der Kombination vieler Bäume zum Einsatz → Abschnitt 5.5.
- Für einzelne Bäume (hier) wird typischerweise versucht, die optimale Anzahl an Knoten zu realisieren, d.h. statt Regularisierung zu nutzen versucht man die Komplexität der Baumstruktur **explizit** vorzugeben.
- Dies kann auf zwei Arten geschehen:
  - Das Wachsen (***Growing***) wird vorzeitig terminiert, also nicht bis zu den reinen Knoten durchgeführt.
    Das Problem: Man weiß nie, was durch weiteres Splitten noch gewonnen werden könnte und wie viele/wenige Splits bis zum finalen, ausgewachsenen Klassifikator noch nötig sein werden.
  - Der komplette, ausgewachsene Baum wird konstruiert bis alle Blätter rein sind.
    Dann wir der Baum wieder zurückgeschnitten (***Pruning***).

  Diese zwei Alternativen (Growing und Pruning) tauchen an sehr vielen Stellen in der modernen Statistik auf. Z.B. auch bei Auswahl der Regressoren bei einem linearen Regressionsproblem:
  - **Growing** = Forward Selection: Starte mit einem **leeren** Modell und **selektiere** zusätzliche Regressoren, Stück für Stück.
  - **Pruning** = Backward Elimination: Starte mit einem **vollen** Modell und **werfe** Regressoren **raus**, Stück für Stück.

- Typischerweise ist **Growing deutlich weniger aufwändig** aber **Pruning deutlich besser**, weil wichtige Informationen über die Wechselwirkungen der verschiedenen Teile (Knoten oder Regressoren) im vollen Baum/Modell vorhanden sind, die beim Growing fehlen.

## Vorgehensweise beim Pruning

**Knotengröße** = Anzahl der Datenpunkte in einem Knoten

- Ein großer Baum $T_0$ wird konstruiert, entweder
  - bis er ausgewachsen ist (also alle Blätter rein sind) oder
  - bis eine minimale **Knotengröße** erreicht ist, z.B. $N_i = 5$
- Dann wird dieser Baum mit sog. **Cost-Complexity Pruning** auf kleinere Bäume $T$ zurückgeschnitten.

## Cost-Complexity Pruning

- Ähnlich wie bei Informationskriterien (AIC, BIC, ...) wird ein Strafterm zum Klassifikationskriterium addiert:

Finde den Baum $T$, der minimiert: $C_\alpha(T) = \sum_{i=1}^{|T|} N_i J_i + \alpha |T|$

Strafterm für Baumkomplexität

Unordnung aller Blätter im Baum, z.B. Gini-Index, gewichtet mit Anzahl enthaltener Datenpunkte

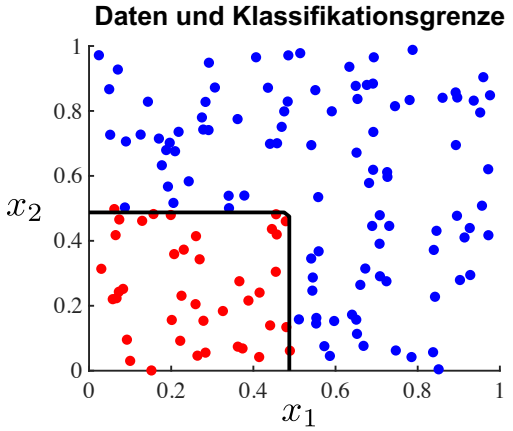
$T_0$: Original-Baum
$T$: Geprunter Baum
$|T|$: Anzahl Blätter in Baum $T$

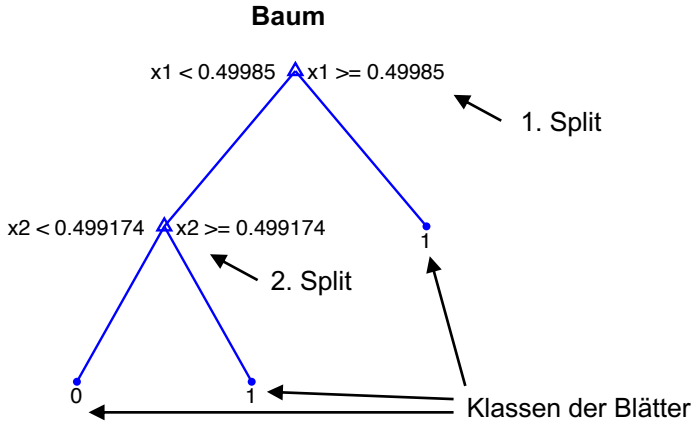
- Einstellung der Stärke der Komplexitätsstrafe $\alpha$ erlaubt einen Tradeoff zwischen hoher Klassifikationsleistung ($\alpha$ klein) und einfachen Bäumen ($\alpha$ groß). Jedem Wert von $\alpha$ entspricht ein geprunter Baum.
- $\alpha$ wird von 0 ausgehend immer weiter erhöht, es fallen immer mehr Knoten weg, bis am Ende nur noch die Wurzel übrig bleibt.
- Ein guter Zahlenwert für $\alpha$ kann über Kreuzvalidierung oder separate Validierungsdaten gefunden werden.

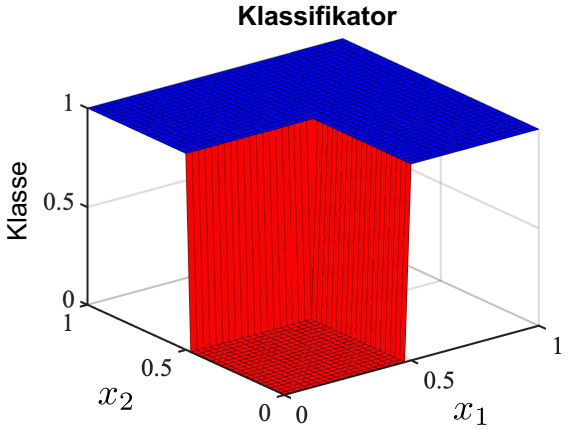
# 5.4 Beispiel CART und Pruning

## Beispiel CART

- Einfaches 2-D-Beispiel
- Eine Klasse (rot = 0): unten links
- Andere Klasse (blau = 1): sonst
- 140 **unverrauschte** Datenpunkte zufällig in $[0\ \ 1]^2$ verteilt
- Mit 2 Splits kann ein **perfekter Klassifikator** konstruiert werden
  → Baum mit 5 Knoten bzw. 3 Blättern
- Die wahren Grenzen bei ($x_1 = 0.5$ / $x_2 = 0.5$) werden sehr genau gefunden



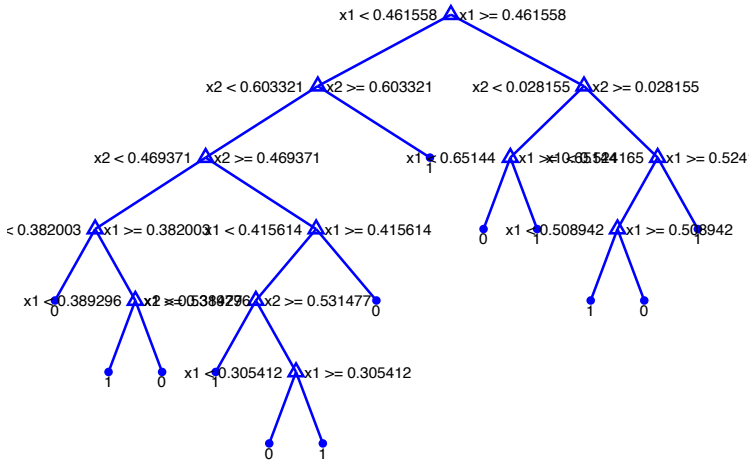
# 5.4 Beispiel CART und Pruning

## Beispiel CART

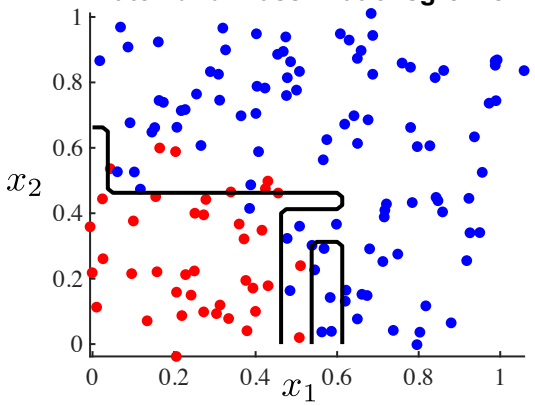
- Dasselbe nochmal mit **verrauschten** Daten
- Es entsteht ...
  → Baum mit 25 Knoten bzw. 13 Blättern
- Sehr **komplexe** und **unrobuste Klassifikationsgrenze**
- Starkes **Overfitting**

→ **Schlechter** Klassifikator!

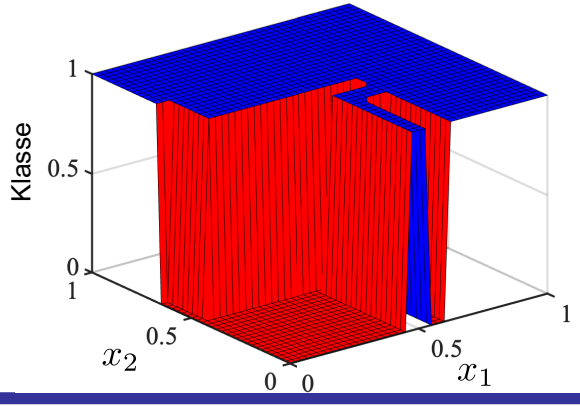
**Baum**




**Daten und Klassifikationsgrenze**

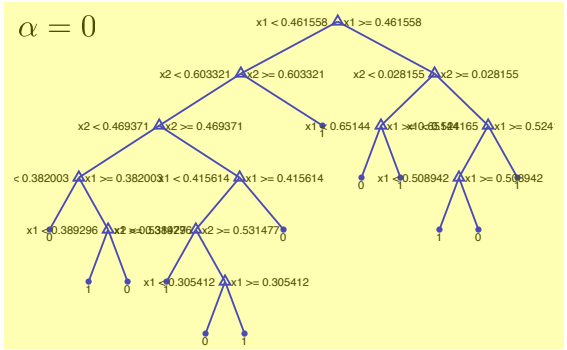


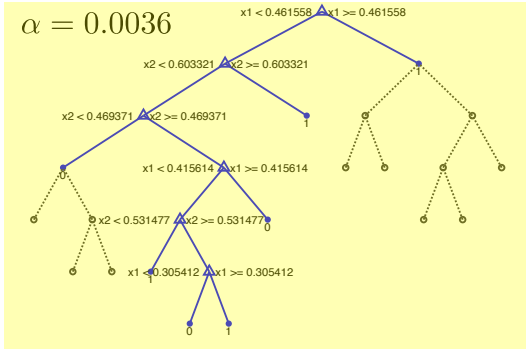

**Klassifikator**

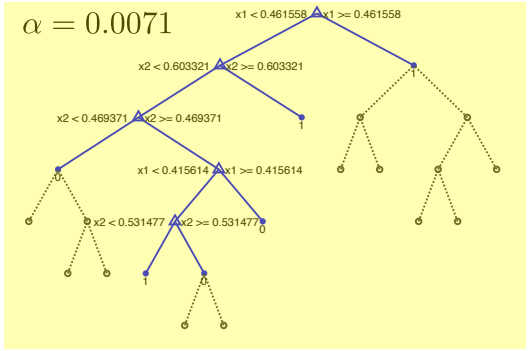


## Beispiel CART

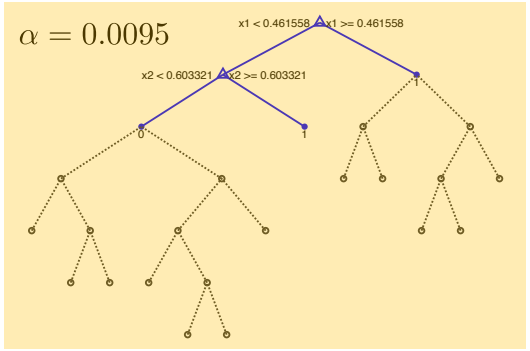
- Es wird Pruning benötigt, um den Baum zu vereinfachen und damit Overfitting zu reduzieren.

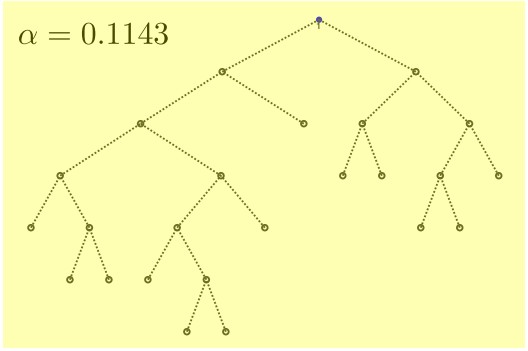



Struktur identisch zu Baum für unverrauschte Daten!



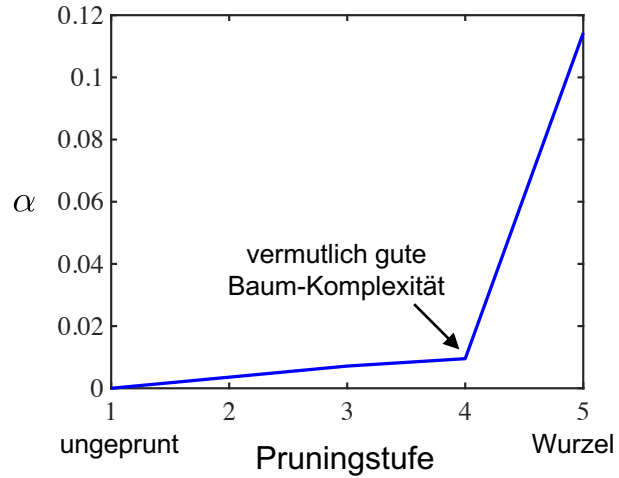

Alles weggeprunt bis zur Wurzel

## Beispiel CART

- Wie findet man ein **gutes *α*** ?
- ***α*** wird stetig **erhöht**

- Standardvorgehen:
  - Fehler auf separaten Validierungdaten
  - **Kreuzvalidierung** (*cross validation*) typischerweise 5-fach oder 10-fach

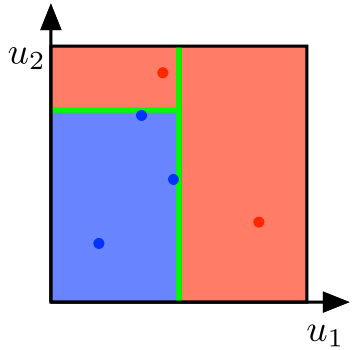
- Als grober Anhaltspunkt kann auch dienen:
  - Verlauf von *α* über Pruningstufe
  - wieviel mehr Bestrafung *α* ist notwendig, damit weitere Baumteile wegfallen?
  - *α* vor **großen Sprüngen** (hier: 4) sind vielversprechende Bestrafungswerte, da der **Klassifikationsfehler** offensichtlich **sehr stark ansteigt**, wenn man weiter prunt



# 5.5 Orthogonale oder schräge Splits?

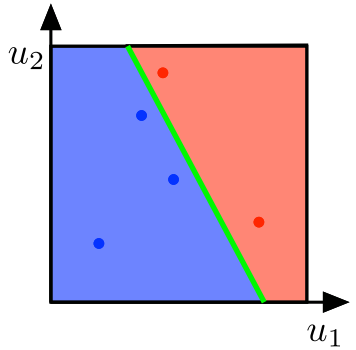
## Stärke achsenorthogonaler Splits

- **Einfach**
- **Endliche** Anzahl an Möglichkeiten, d.h. Optimierung = beste Alternative wählen.
- Gut **interpretierbar**, insb. auch im **Hochdimensionalen**, da man nur für jede Achse individuell denken muss.
- Vorgehensweise wie bei **Fuzzy-Logik** üblich, nur dass dort statt harter Splits weiche Übergänge durch Zugehörigkeitsfunktionen verwendet werden.




## Schwächen achsenorthogonaler Splits

- Teilweise werden **sehr viele Splits** für eine gute Beschreibung der Klassifikationsgrenze gebraucht
- Schräge Klassifikationsgrenzen müssen durch „**Treppenstufen**“ (Serie von Splits) angenähert werden
- Jede **Rotation** der Achsen (z.B. durch PCA) verändert dramatisch die Problemstellung und Lösung




## Eigenschaften schräger (*oblique*) Splits

- **Kompliziert** zu finden. Wie genau?
- **Schwer zu interpretieren**
- Sehr **flexibel** und leistungsfähig → viel weniger Splits/Gebiete notwendig.
- **Vorteil** schräger Splits **wächst** mit zunehmender **Dimension.**

# 5.6 Eigenschaften von Klassifikationsbäumen

## *Vorteile* von CART oder anderen (einzelnen) Bäumen

- Bäume sind sehr gut **interpretierbar**, nachvollziehbar und bilden eine exzellente **Schnittstelle** zum menschlichen **Experten.**
- Bäume sind **einfach** und **schnell** (Training und Auswertung).

## *Nachteile* von CART oder anderen (einzelnen) Bäumen

- Bäume, insb. ausgewachsene (nicht geprunte), neigen stark zu **Overfitting** (kleiner Bias, hohe Varianz), d.h. sie sind auch sehr empfindlich auf Änderungen bzw. Verrauschung einzelner Datenpunkte.
- Bäume sind **instabil/fragil** (nicht im dynamischen Sinne gemeint), d.h. kleine Änderungen in den Daten (ein zusätzlicher Datenpunkt, etwas anderes Rauschen, ...) können dramatische Änderungen des Baums zur Folge haben.
- Bäume **schalten** an den Splits **hart** um (*non-smooth*). Bei Klassifikationsbäumen ist das typischerweise ok. Bei Regressionsbäumen (hier nicht behandelt) stört dies sehr, da dort Stetigkeit und oft sogar Differenzierbarkeit gewünscht wird.

## Generelle Einordnung im Vergleich zu alternativen Methoden des Machine Learning

- Hohe Geschwindigkeit, da keine kontinuierliche numerische Optimierung nötig
- Gute Interpretierbarkeit
- Niedrige oder mittlere Performance
- Niedriger Bias, hohe Varianz

# 5.7 Bagging und Random Forests

## Ensemble-Methoden

- **ZIEL: Reduktion des Varianzfehlers**. Daher insb. sinnvoll bei Modellarchitekturen mit hoher Varianz wie Bäumen.
- Manchmal spricht man auch von einem **Komitee** (*committee*) statt einem Ensemble.
- Statt einem Modell (hier einem Baum) wird ein **Ensemble von vielen Modellen (hier Bäumen)** aus Daten gelernt.
- Diese vielen Modelle (hier Bäume) werden **aggregiert**, also zusammengefasst, so dass sich ein **Gesamtmodell** ergibt.
- Es existieren verschiedene Methoden der **Aggregation**:
  - Mittelung (mit Abstand am weitesten verbreitet bei Regression)
  - gewichtete Mittelung: Die Gewichte können z.B. der inversen Fehlervarianz jedes Einzelmodells entsprechen
  - Voting (bei Klassifikation): Jedes Modell klassifiziert und das Gesamtmodell richtet sich nach der Mehrheit

## Eigenschaften von Ensembles

- Die Einzelmodelle müssen **möglichst unterschiedlich** sein, damit Ensembles große Verbesserungen bringen können.
- **Ensembles reduzieren** eine große Schwäche von Bäumen (großer **Varianzfehler**).
- Es gibt verschiedene **Strategien**, wie sich die Einzelmodelle **unterscheiden** können → nächste Folie.
- Oft setzen sich Ensembles aus **100** oder **1000 Einzelmodellen** zusammen. Dementsprechend höher ist der Aufwand bei Training und Auswertung bzgl. Rechenaufwand und Speicher.
- Weiterer Nachteil dadurch: **Einfachheit** und **Interpretierbarkeit** von Einzelbäumen geht **verloren**.

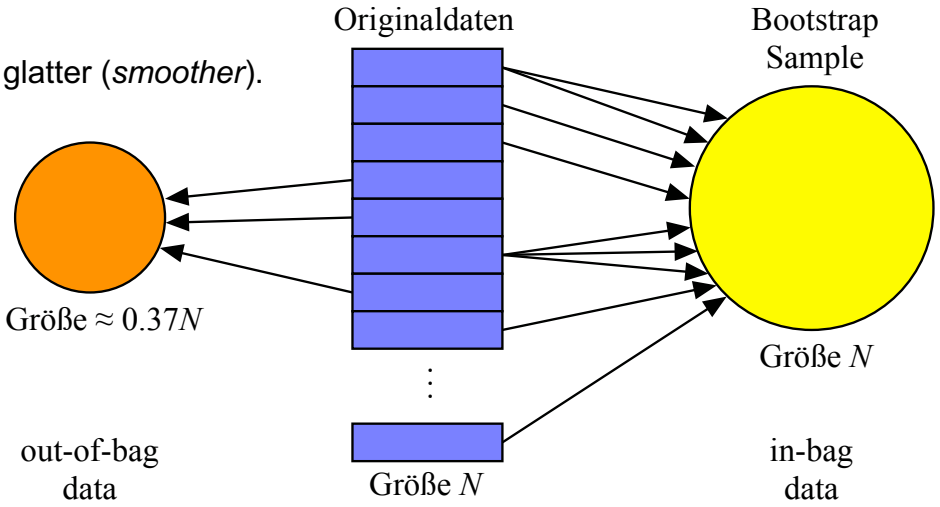
# 5.7 Bagging und Random Forests

## Unterscheidung der Einzelmodelle (einzelnen Bäume) in einem Ensemble

- Die Einzelbäume sollen in unterschiedlichen Regionen gut bzw. schlecht sein und unterschiedliche (lokale) Komplexitäten aufweisen. Durch die Aggregation entsteht dann ein besseres, robusteres Gesamtmodell – ähnlich wie beim Mitteln verrauschter Daten.
- Zwei Möglichkeiten für Unterschiede
  - Daten, typischerweise mit Bootstrapping
  - Modellkonstruktion, typischerweise mit Einschränkungen für erlaubte Dimensionen für Splits
- Die Klassifikationsgrenze wird durch das Ensemble feiner und glatter (*smoother*).

## Bootstrap Sample

- Bootstrapping ist ein Standard-Statistikverfahren.
- Es werden viele Bootstrap Samples aus den Originaldaten gezogen (jeweils mit Zurücklegen).
- Manche Daten darin mehrfach, manche fehlen (≈37%).
- Mit jedem Bootstrap Sample wird ein Modell trainiert.
- So kann man beliebig viele Datensätze erzeugen zur Bewertung von Statistiken (z.B. Konfidenzintervalle).



# 5.7 Bagging und Random Forests

## Bagging

- Bagging steht für **Bootstrap Aggregation.**
- Ziehe $N_{\text{bag}}$ Bootstrap Samples aus den Originaldaten.
- Trainiere für jedes Bootstrap Sample (identische Größe wie Originaldaten) einen CART $\hat{f}_i$ , $i = 1, 2, ..., N_{\text{bag}}$ .
- Typische Aggregation
  - bei Klassifikationsbäumen Voting: (alternativ Mittelung der Wahrscheinlichkeiten)
    $$\hat{f}_{\text{bag}} = \arg\max \hat{f}_i$$
  - bei Regressionsbäumen Mittelung:
    $$\hat{f}_{\text{bag}} = \frac{1}{N_{\text{bag}}} \sum_{i=1}^{N_{\text{bag}}} \hat{f}_i$$

**Random Forests** sind sehr leistungsfähige State-of-the-Art-Klassifikatoren, die oft sogar Support Vector Machines in ihrer Performance übertreffen.

**Anwendung** z.B. in der Gestenerkennung bei Kinect (Xbox).

## Random Forest

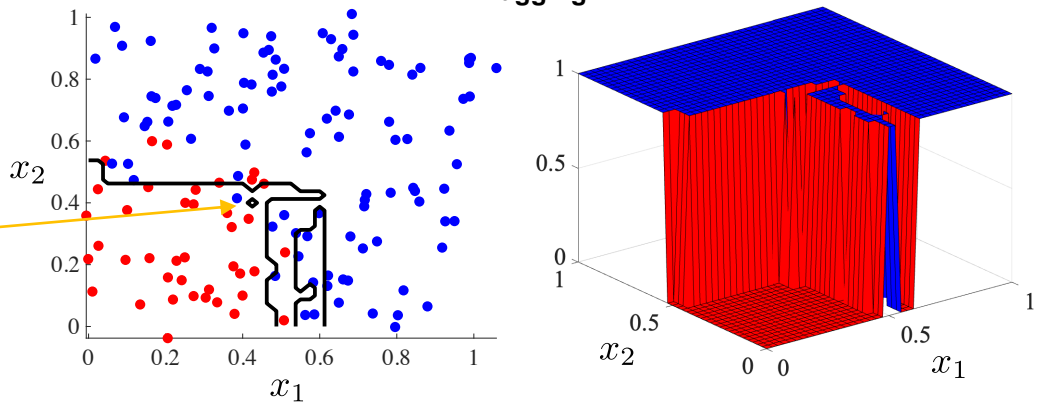
- Noch mehr Randomisierung und Mittelung, um den Ensemble-Effekt weiter zu verstärken.
- Bagging + bei jedem Split werden nicht alle Dimensionen für das Teilen zugelassen.
  Es werden nur zufällig ausgewählte Dimensionen für das Splitting bereit gestellt – bei jedem Split wieder neu „ausgewürfelt“.
- Typischerweise werden bei *n* Dimensionen nur ca. $\sqrt{n}$ Dimensionen zufällig fürs Splitting zur Verfügung gestellt.
- Dadurch wird das Overfitting stark reduziert und das Gesamtmodell sehr robust, weil
  - die besten Dimension oft gar nicht für das Splitten verfügbar sind
  - Bagging diese Bäume mittelt

## Beispiel Bagging und Random Forest

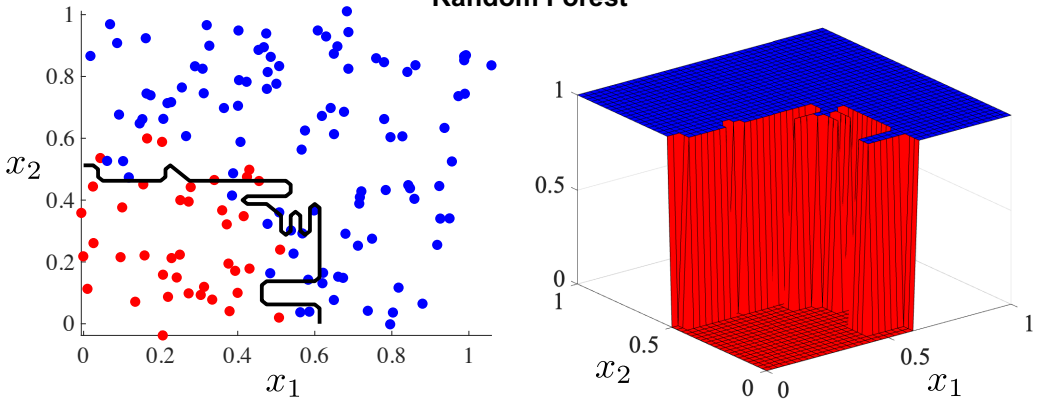
- Ensemble mit 30 Bäumen
- Deutlich flexiblere Klassifikationsgrenzen (mehr Rundungen und Schnörkel möglich)
- Beim Bagging-Beispiel werden sogar einzelne Punkte isoliert: Ist das sinnvoll?
- Random Forest ist noch flexibler als Bagging
- Anzahl der Bäume im Ensemble in Praxis:
  - Bagging: ~100
  - Random Forest: ~1000

- Hier: Leider sehr hohes Overfitting
- Für größere und vor allem höherdimensionalere Probleme mit mehr Bäumen im Ensemble sollte der Varianzfehler (das Overfitting) deutlich reduziert sein!

**Bagging**




**Random Forest**



# 6. Support Vector Machines (SVM)

# Inhalt

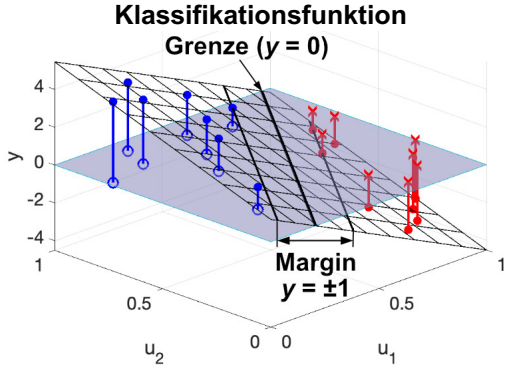


## Empfohlene Videos zum Thema Support Vector Machines:

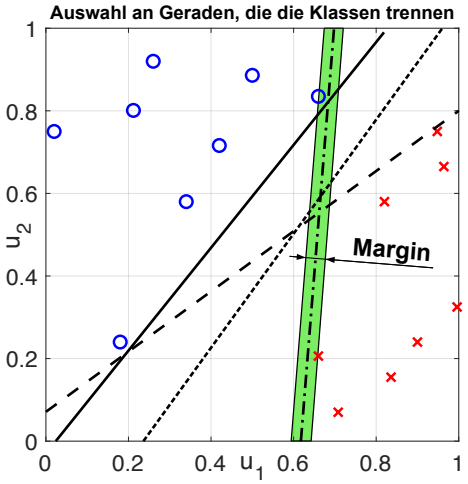
- Videos zu Statistical Learning von T. Hastie und R. Tibshirani (Stanford)
- https://www.youtube.com/watch?v=m59UOo5jAFU&list=PLAOUn-KLSAVOf4Uk-WbLGPUDFjMSyytkw

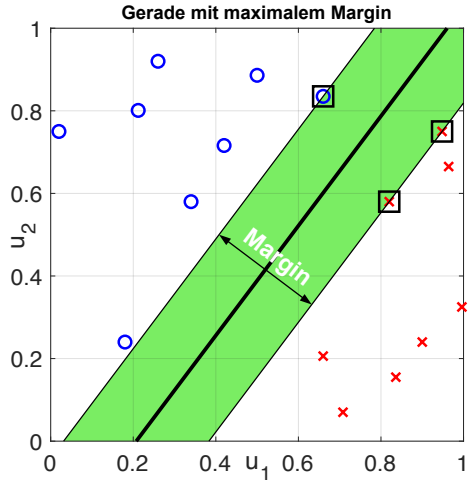
# 6.1 Idee der Support Vector Machine (SVM)

## Separation der Datenpunkte mit maximalem Trennbereich (Margin)

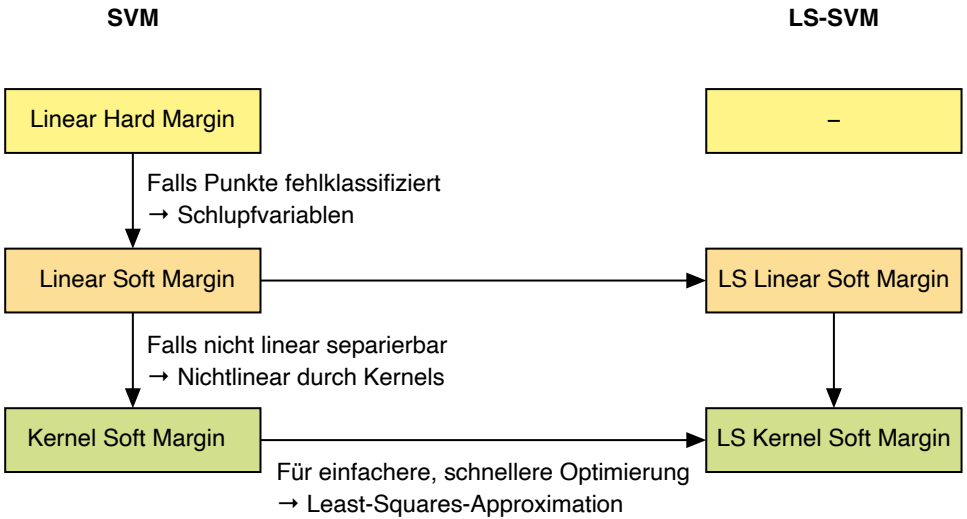



- **Ziel:** Trennung von 2 Klassen mit maximaler Robustheit.
- Lineare Support Vector Machine (hier: Hard Margin SVM, d.h. linear separierbare Daten)
  - Klassifikationsfunktion: Ebene (2 Eingänge), bzw. Hyperebene (>2 Eingänge).
  - Klassifikationsgrenze ist die Höhenline der Funktion beim Wert Null.
  - die Höhenlinie soll so liegen, dass die nächsten Punkte einen maximalen Abstand von der Klassifikationsgrenze haben.
  - wenn der Trennbereich möglichst breit ist, ist die Wahrscheinlichkeit einer Fehlklassifikation von neuen Daten gering.
- SVM-Lösung (Bild unten rechts):
  - Größtmöglicher Margin.
  - Punkte, die auf den Margin-Rändern liegen (mit Quadraten gekennzeichnet), heißen **Support Vektoren**.
  - zur Berechnung der Klassifikationsfunktion werden nur diese Support Vektoren benötigt. Andere Verfahren (z.B. RBF-Netze) benötigten in der Regel alle Trainingsdatenpunkte ⇒ Bei vielen Datenpunkten ist der Rechen- und Speicheraufwand bei SVM kleiner.



## Übersicht der verschiedenen SVM-Varianten




- **Hard Margin**
  - Möglichst großer Abstand (Margin) zwischen den Klassen
- **Soft Margin**
  - Erlauben von fehlklassifizierten Punkten
    → Erhöht die Praxistauglichkeit und Robustheit bzgl. Rauschen, Ausreißern, Fehlern, Ungenauigkeiten
  - Einführung eines Tuningparameters $C$ (auch $\gamma$ oder $\lambda$ bzw. $1/\gamma$, $1/\lambda$) zur Bestrafung des Abstands zum Margin
- **Nichtlineare Kernels**
  - Erlaubt beliebig geformte Klassifikationsgrenzen
  - Hyperparameter ermöglichen das Tuning der Glattheit (*smoothness*)
- **Least-Squares-Approximation**
  - Macht alle Datenpunkte zu Support Vektoren; alle Datenpunkte tragen zur Lage der Klassifikationsgrenze bei
  - Alle Vorteile von Least-Squares können ausgenutzt werden
    - effizient, schnell, robust, numerisch ausgereift, analytische Lösung
    - rekursive (d.h. onlinefähige) Algorithmen
    - analytische Berechnung des Leave-one-out-Fehlers

# 6.2 Herleitung des Optimierungsproblems

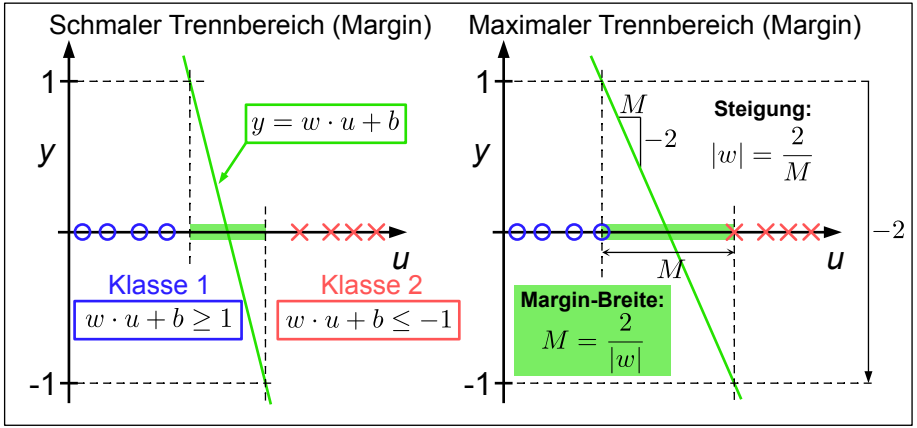
## Bestimmung der Margin-Breite

Um die Margin-Breite *M* maximieren zu können, muss sie zunächst berechnet werden:

- Bei der SVM wird definiert, dass alle Punkte auf dem Margin-Rand, je nach Klasse, entweder den **Wert +1 oder -1** haben.
- Ein Eingang bzw. Merkmal *u* (*m* = 1) ⇒ Geradengleichung:
  - Im Bild rechts kann man leicht sehen, dass die Margin-Breite *M* in diesem Fall 2 geteilt durch den Betrag der Steigung |*w*| der Klassifikationsfunktion *y* ist.
- Mehrere Eingänge $u_i$ (*m* > 1) ⇒ (Hyper-)Ebenengleichung:
  - Es gibt nun eine Steigung $w_i$ für jeden der *m* Eingänge, die in dem liegenden Vektor $\underline{w}^T$ zusammengefasst werden.
  - Für die Berechnung der Margin-Breite wird nun die Länge des Vektors $\underline{w}$ benötigt (2-Norm):

$$M = \frac{2}{||\underline{w}||_2}$$

$$||\underline{w}||_2 = \sqrt{\underline{w}^T \cdot \underline{w}} = \sqrt{w_1^2 + w_2^2 + \ldots + w_m^2}$$




**(Hyper-)Ebenengleichung:**

$$y = \underline{w}^T \cdot \underline{u} + b, \quad \underline{w}^T = [w_1, w_2 \ldots w_m], \quad \underline{u} = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_m \end{bmatrix}$$

$$y = \sum_i^m w_i \cdot u_i + b = w_1 \cdot u_1 + w_2 \cdot u_1 + \ldots + w_m \cdot u_m + b$$

## Verlustfunktion und Nebenbedingungen des Optimierungsproblems der SVM

- Zu minimierende Größe (quadratische Verlustfunktion):
  - Statt die **Margin-Breite** zu **maximieren**, wird der Kehrwert zum Quadrat **minimiert**. Dies führt auf das gleiche Optimum, hat aber numerische Vorteile. Dabei werden die optimalen Parameter $\underline{w}$ und $b$ gesucht.
- Nebenbedingungen (alle linear):
  - für jeden der $N$ Datenpunkte erhält das Optimierungsproblem eine Nebenbedingung, die von der Klasse abhängig ist, denn die Punkte sollen je nach Klasse einen Klassifikationswert $y$ von +1 oder mehr, bzw. -1 oder weniger haben.
  - wenn man für die Trainingsdaten die $y_i$ Werte der Datenpunkte je nach Klassenzugehörigkeit als +1 bzw. -1 wählt, kann man beide Arten von Nebenbedingungen zu einer zusammenfassen.

Verlustfunktion:

$$\min_{\underline{w},b} \frac{||\underline{w}||_2^2}{2}$$

Nebenbedingungen:

$\underline{w}^T \cdot \underline{u}_i + b \geq +1, \text{ wenn } y_i = +1$ Klasse 1 ○

$\underline{w}^T \cdot \underline{u}_i + b \leq -1, \text{ wenn } y_i = -1$ Klasse 2 ×

Zusammengefasste Nebenbedingungen:

$$y_i \cdot (\underline{w}^T \cdot \underline{u}_i + b) \geq 1$$

- Lösung des **quadratischen Optimierungsproblems** (quadratische Verlustfunktion und lineare Nebenbedingungen):
  - Quadratic Programming (QP) ⇒ Effiziente Algorithmen vorhanden, z.B. MATLAB `quadprog`.
  - $N$ Nebenbedingungen (eine für jeden Datenpunkt), $m$+1 optimale Parameter werden gesucht (alle $w_i$ und $b$).
  - führt nicht auf die SVM-Lösung mit Support Vektoren ⇒ **Optimierung des Dualen Problems nötig!**
  - **einfaches Beispiel zur Herleitung eines Duales Problems ⇒ nächste Seite.**

## Beispiel: Minimum von $x^2$ unter einer Nebenbedingung

- Die Nebenbedingung wird, geeignet umgeformt, mit dem Lagrange-Multiplikator $\alpha$ (≥0) multipliziert und zur eigentlich zu minimierenden Größe hinzuaddiert, siehe Schritt 1 im Kasten rechts. Man erhält die Lagrange-Funktion $L$:
  - der Nebenbedingungs-Term wird so gewählt, dass $L$ **größer** wird, wenn die Nebenbedingung verletzt wird ($x$<1), er also der Minimierung entgegen wirkt. Es ergibt sich somit ein mit $\alpha$ gewichteter „Strafterm“. Der Wert des Multiplikators $\alpha$ wird in Schritt 3 bestimmt.
  - $L$ wird nun nach $x$ abgeleitet, die Ableitung gleich Null gesetzt. Daraus ergibt sich das Minimum bezüglich $x$. Die Position des Minimums hängt nun nur noch von $\alpha$ ab. Es gibt also für verschiedene $\alpha$-Werte **unterschiedliche Minima**.
- Das Ergebnis wird nun in $L$ eingesetzt und diesmal die Ableitung nach $\alpha$ berechnet (Schritte 2, 3 und 4). Dabei wird nun aber das **Maximum** gesucht (insgesamt also das Maximum aller Minima):
  - dies nennt man auch das **Duale Optimierungsproblem**.
  - warum sich bei diesem Vorgehen das Minimum des Optimierungsproblems mit Nebenbedingung ergibt, wird auf der nächsten Folie mit einer Grafik für dieses Beispiel verdeutlicht.

Gesucht ist das $\min_x x^2$ unter der Nebenbedingung $x \geq 1$:

1) Mit dem Lagrange-Multiplikator $\alpha$ ergibt sich:

$$L = x^2 - \alpha \cdot (x-1), \ \alpha \geq 0$$

$$\frac{dL}{dx} = 2x - \alpha = 0 \Leftrightarrow \boxed{x = \frac{\alpha}{2}}, \ \frac{d^2L}{dx^2} = 2 > 0 \Rightarrow \text{Minimum}$$

2) Einsetzen von $x$ in die Lagrange-Funktion:

$$L = \frac{\alpha^2}{4} - \alpha \cdot \left(\frac{\alpha}{2} - 1\right) = -\frac{\alpha^2}{4} + \alpha$$

Sattelpunkt!

3) Berechnen von $\max_\alpha -\frac{\alpha^2}{4} + \alpha$ :

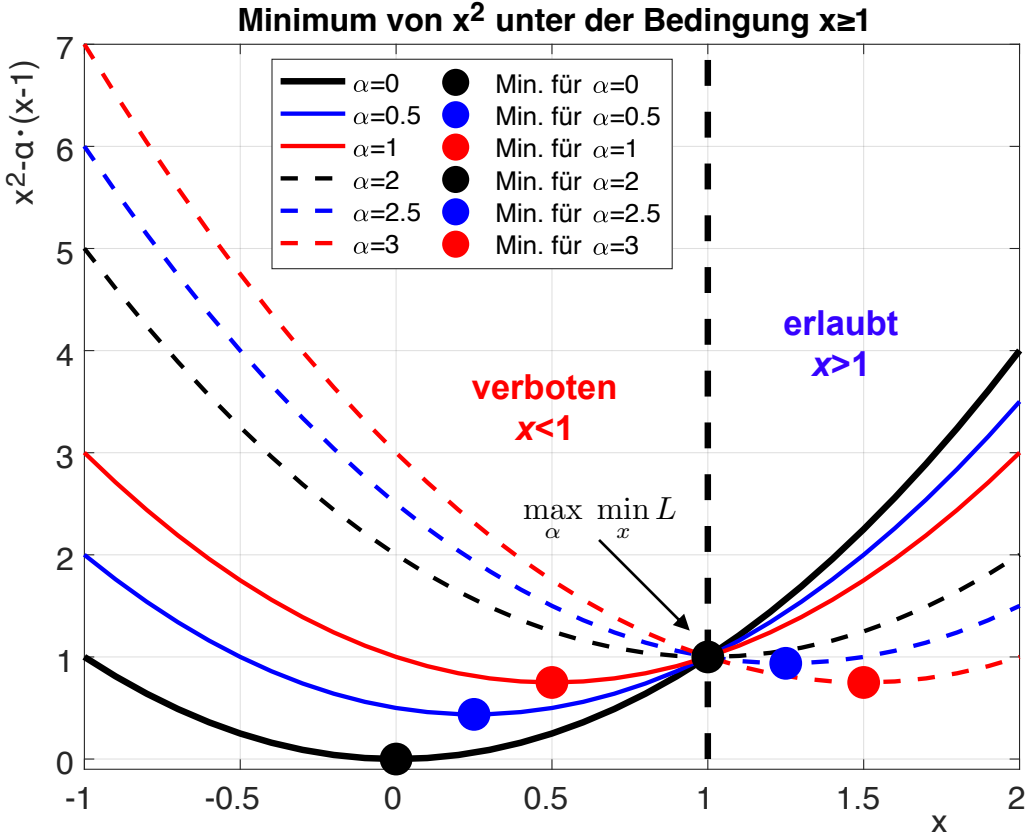
$$\frac{dL}{d\alpha} = -\frac{\alpha}{2} + 1 = 0 \Leftrightarrow \boxed{\alpha = 2}, \ \frac{d^2L}{d\alpha^2} = -\frac{1}{2} < 0 \Rightarrow \text{Maximum}$$

4) Die Lage des Minimums unter der Nebenbedingung $x \geq 1$ lautet daher :

$$x = \frac{\alpha}{2} = \frac{2}{2} \Rightarrow \boxed{x = 1}$$

## Beispiel (fortgesetzt): Optimaler Wert für $\alpha$

- Im rechten Bild ist die Lagrange-Funktion $L$ für verschiedenen Werte von $\alpha$ dargestellt.
- Die schwarze nicht gestrichelte Line stellt die zu minimierende Funktion ($x^2$) ohne „Strafterm“ ($\alpha = 0$) dar und hat ihr Minimum im schwarzen Punkt bei $x = 0$.
- Mit zunehmendem Wert für $\alpha$ bewegen sich die Minima der Kurven (blaue, rote Linie) nach rechts, da $L$ durch den „Strafterm“ links von $x = 1$ angehoben und rechts von $x = 1$ abgesenkt wird.
- Bei der schwarzen gestrichelten Line erreicht das Minimum (schwarzer Punkt, $\alpha = 2$) den Maximalwert, bei weiterer Erhöhung (blaue, rote gestrichelte Linie) sinken die Minima wieder ab.
- Das Minimum mit dem maximalen Wert liegt, wie berechnet, bei $x = 1$ und entspricht der Lösung des Optimierungsproblems mit Nebenbedingung.
- Es gibt Fälle, bei denen das duale Problem nicht die exakt gleiche Lösung hat wie das originale. Dies tritt bei den hier gezeigten Anwendungen aber nicht auf.



## Herleitung der Dualen Formulierung der Hard Margin SVM

- Herleitung der Lagrange-Funktion:
  - zu minimierende Größe: Kehrwert des quadratischen Margins.
  - Ungleichheitsnebenbedingungen für jeden der *N* Datenpunkte.
    ⇒ Lagrange-Funktion *L* mit $\alpha_i$ wobei $\alpha_i \geq 0$ sein muss, *i* = 1, ..., *N*.
- Ableitungen von *L* nach $\underline{w}$ und *b* gleich Null setzen:
  - man erhält eine Gleichung für die Berechnung von $\underline{w}$,
  - und ***N*+1** Nebenbedingung (NB 1: $\alpha_i \geq 0$, NB 2: Summe $\alpha_i y_i = 0$).
- Einsetzen von $\underline{w}$ und der NB 2 in die Lagrange-Gleichung:
  - Quadratisches Optimierungsproblem, zu **maximieren** bzgl. $\alpha_i$ .
  - Bestimmen der $\alpha_i$ z.B. mit `quadprog` (MATLAB) ⇒ nächste Folie.
- Die Klassifikationsfunktion *y* für einen neuen Punkt $\underline{u}_{neu}$ wird mit den $\alpha_i$ , den Datenpunkten ($y_i$, $\underline{u}_i$) und *b* berechnet.
- Die Lösung des Optimierungsproblems zeigt, dass $\alpha_i$ **für alle Punkte $\underline{u}_i$ Null** wird, die **nicht** auf der **Margin-Grenze** liegen
  ⇒ Irrelevant für Optimierungsproblem ($\alpha_i = 0$)!
  ⇒ Die übrigen sind die Support Vektoren ($\alpha_i > 0$)!

$$\min_{\underline{w},b} \frac{1}{2}||\underline{w}||_2^2 \quad \text{mit NB: } y_i \cdot (\underline{w}^T \cdot \underline{u}_i + b) \geq 1, \; i = 1, 2, \ldots N$$

$$L = \frac{1}{2}||\underline{w}||_2^2 - \sum_{i=1}^{N} \alpha_i \left[ y_i \cdot (\underline{w}^T \cdot \underline{u}_i + b) - 1 \right], \text{ mit } \boxed{\alpha_i \geq 0}$$

NB 1

$$\frac{\partial L}{\partial \underline{w}} = \underline{w} - \sum_{i=1}^{N} \alpha_i y_i \underline{u}_i = 0 \Leftrightarrow \boxed{\underline{w} = \sum_{i=1}^{N} \alpha_i y_i \underline{u}_i}$$

$$\frac{\partial L}{\partial b} = -\sum_{i=1}^{N} \alpha_i y_i = 0 \Leftrightarrow \boxed{\sum_{i=1}^{N} \alpha_i y_i = 0}$$

NB 2

$$\max_{\alpha_i} \left[ \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \left( \underline{u}_i^T \underline{u}_j \right) \right]$$

$\underline{w}$ und NB 2 in *L*

$$\text{Klassifikationsfunktion: } y = \sum_{i=1}^{N} \alpha_i y_i \left( \underline{u}_i^T \underline{u}_{neu} \right) + b$$

$$b = y_{SV} - \sum_{i=1}^{N} \alpha_i y_i \left( \underline{u}_i^T \underline{u}_{SV} \right)$$

$\underline{u}_{SV}$, $y_{SV}$ beliebiger Support Vektor ($\alpha_{SV} \neq 0$)

- Das Problem ist nur lösbar für **linear separierbare** Fälle (Hard Margin SVM) ⇒ **Schlupfvariablen** einführen (Soft Margin)!

## Lösung des quadratischen Optimierungsproblems mit MATLAB (Befehl `quadprog`)

- Da der MATLAB-Befehl **minimiert**, aber beim Dualen Problem aber das **Maximum** gesucht wird, **muss die Gleichung mit -1 multipliziert werden**. Ebenso muss beachtet werden, das die **Ungleichheitsnebenbedingungen** als **≤** definiert sind.
- **MATLAB-Befehl**: `x = quadprog(H,f,A,c,Aeq,ceq)`
  - *H*: Matrix mit Koeffizienten für den quadratischen Teil der Verlustfunktion.
  - *f*: Vektor mit Koeffizienten für den linearen Teil der Verlustfunktion.
  - *Aeq, ceq*: Matrix, bzw. Vektor mit Koeffizienten für die Gleichheitsnebenbedingungen.
  - *A, c*: Matrix, bzw. Vektor mit Koeffizienten für die Ungleichheitsnebenbedingungen in der Form ≤.
- Die Elemente $H_{ij}$ der Matrix $\underline{H}$ errechnen sich sich aus den $y_i$ und $\underline{u}_i$ Werten der i=1,2,…*N* Datenpunkte.
- *f* ist ein Vektor der Länge N mit -1 Einträgen, da hier die negative Summe aller $\alpha_i$ gebildet werden muss.
- *Aeq* ist ein Vektor (nur eine Gleichheits-NB) mit allen $y_i$ und *ceq* ist gleich 0 (Skalar).
- *A* ist eine negative Einheitsmatrix (*N* x *N*). Negativ, weil die Bedingung in MATLAB ≤ statt ≥ ist. c ist ein Nullvektor (*N*x1).

**Allgemein (MATLAB)**

$$\min_x \frac{1}{2} \cdot \underline{x}^T \cdot \underline{H} \cdot \underline{x} + \underline{f}^T \cdot \underline{x}$$

mit den Nebenbedingungen:

$$\underline{A}_{eq} \cdot \underline{x} = \underline{c}_{eq}$$

$$\underline{A} \cdot \underline{x} \leq \underline{c}$$

$$\underline{x} = [\alpha_1, \alpha_2, \ldots, \alpha_N]^T$$

$$H_{ij} = y_i y_j \left(\underline{u}_i^T \underline{u}_j\right)$$

$$\underline{f}^T = -[1, 1, \ldots, 1]$$

$$\underline{A}_{eq} = \underline{a}_{eq}^T = [y_1, y_2, \ldots, y_N]$$

$$c_{eq} = 0$$

$$\underline{A} = -\underline{I},\ A_{ij} = \begin{cases} -1 \text{ für } i = j \\ 0 \text{ für } i \neq j \end{cases}$$

$$\underline{c} = [0, 0, \ldots, 0]^T$$

**Hard Margin SVM (Maximierung)**

$$\max_{\alpha_i} \left[ \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \left(\underline{u}_i^T \underline{u}_j\right) \right]$$

$$\cdot(-1)$$

$$\sum_{i=1}^{N} \alpha_i y_i = 0$$

$$\alpha_i \geq 0$$

**Hard Margin SVM (Minimierung)**

$$\min_{\alpha_i} \left[ \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \left(\underline{u}_i^T \underline{u}_j\right) - \sum_{i=1}^{N} \alpha_i \right]$$

$$\sum_{i=1}^{N} \alpha_i y_i = 0$$

$$\alpha_i \geq 0$$

Für alle Punkte (Quadrate darum) im Margin oder fehlklassifiziert ist $\alpha_i = C$.

## Einführung von Schlupfvariablen (*slack variables*)

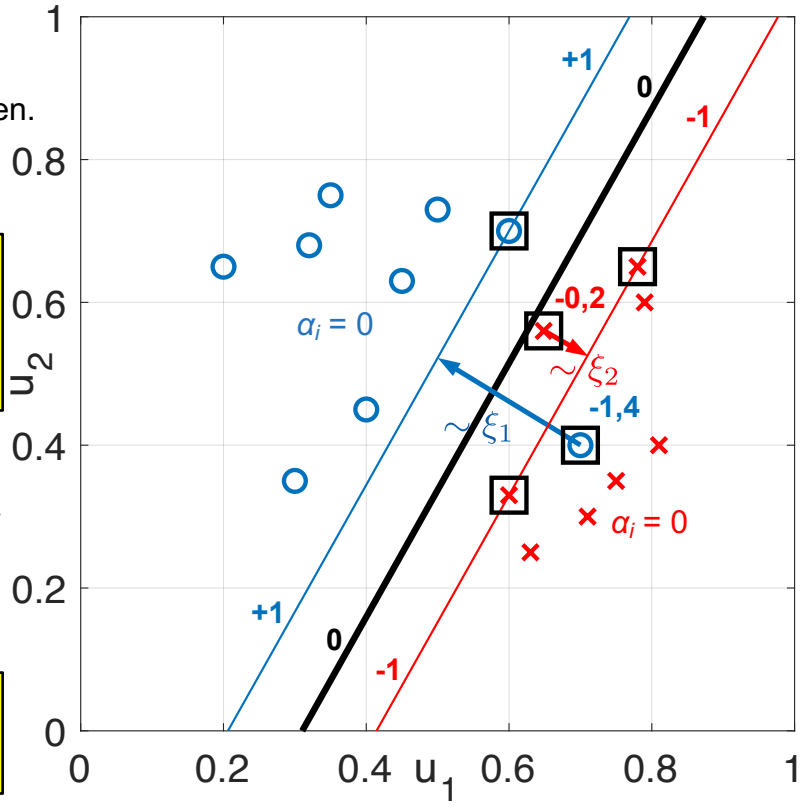
**Näherungsweise** Lösung von **nicht linear separierbaren** Problemen.
Fehlklassifizierte Punkte und Punkte im Margin werden zu Support Vektoren.

1. Einführung einer **Korrektur $\xi_i$** (Schlupfvariable) für jeden Datenpunkt $i = 1, ..., N$, damit die **Nebenbedingungen** erfüllt werden können:

$$y_i(\underline{w}^T \underline{u}_i + b) \geq 1 - \xi_i, \text{ mit } \xi_i \geq 0 \text{ (Schlupfvariable)}$$
$$+1 \cdot (-1{,}4) \geq 1 - \xi_i \Rightarrow \xi_i \geq 2{,}4 \quad \text{Kreis (soll: } y_i \geq +1, \text{ ist: } y_i = -1{,}4)$$
$$(-1) \cdot (-0{,}2) \geq 1 - \xi_i \Rightarrow \xi_i \geq 0{,}8 \quad \text{Kreuz (soll: } y_i \leq -1, \text{ ist: } y_i = -0{,}2)$$

2. Suche nach der **kleinstmöglichen Korrektur**, die alle Nebenbedingungen erfüllt:
   ⇒ Erweiterung der Verlustfunktion um einen Term, der die Größe dieser Korrektur bestraft. Mit dem Parameter $C$ muss ein **Kompromiss** zwischen maximalem Margin und korrekter Klassifizierung eingestellt werden ($C = 0$ ⇒ Hard Margin SVM):

$$\min_{\underline{w},b,\xi_i} \frac{1}{2}||\underline{w}||_2^2 + C\sum_{i=1}^{N}\xi_i \quad \text{mit } C \geq 0$$



# 6.4 Soft Margin SVM

## Duales Problem der Soft Margin SVM

- Zur Ermittlung des Dualen Problems müssen nun zusätzlich auch die Ableitungen nach den Schlupfvariablen berücksichtigt werden:
  - es ergibt sich jedoch nahezu das gleiche Duale Problem, lediglich die **Ungleichheitsnebenbedingungen ändern sich**: ⇒ Die Lagrange-Multiplikatoren müssen nun nicht nur größer gleich 0, sondern auch kleiner gleich dem Straffaktor $C$ sein.
  - Berechnung von $b$: Nur Support Vektoren mit $\alpha_i < C$ erlaubt.
- MATLAB-Funktion `quadprog`:
  - Änderung der Matrix $\underline{A}$: Verdopplung der Zeilenzahl. Die ersten $N$ Zeilen behalten die negative Einheitsmatrix die übrigen erhalten eine positive.
  - Änderung des Vektors $\underline{c}$: Verdoppelung der Länge auf $2N$. Die ersten $N$ Einträge bleiben 0, die übrigen enthalten den gewählten Straffaktor $C$.
- Beispiele (Bilder rechts): Mit zunehmendem $C$ wird der Margin immer schmaler, damit möglichst wenige Punkte innerhalb liegen.

nicht die Quadrate, weil $\alpha_i = C$ !

$0 \le \alpha_i \le C$

d.h. $\alpha_i \ge 0$ und $\alpha_i \le C$

**MATLAB**

$\underline{A} \cdot \underline{x} \le \underline{c}$

$\underline{A}$ ist eine $2N \times N$ Matrix

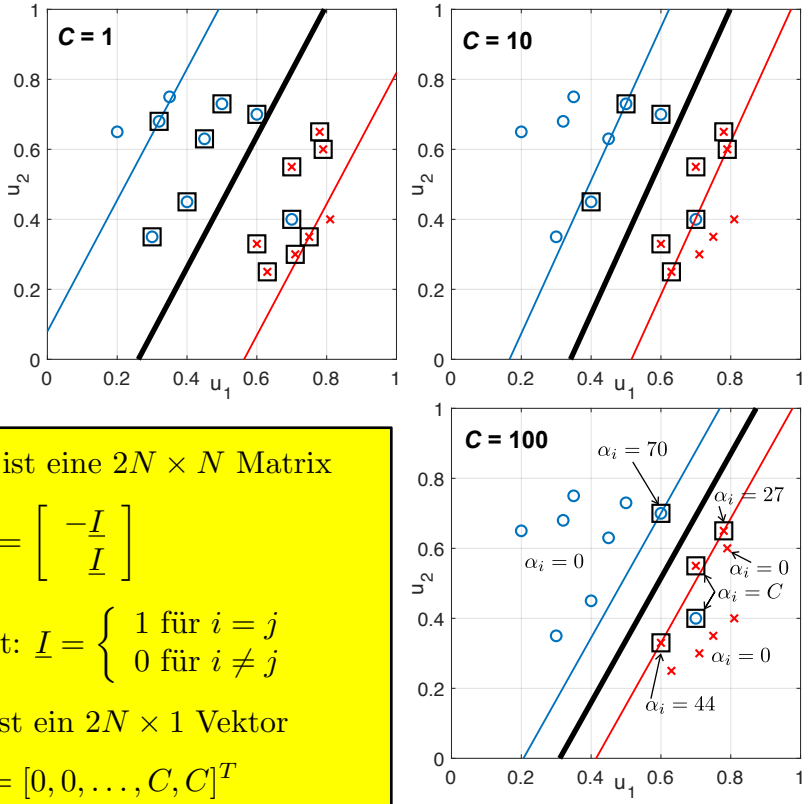
$$\underline{A} = \begin{bmatrix} -\underline{I} \\ \underline{I} \end{bmatrix}$$

mit: $\underline{I} = \begin{cases} 1 \text{ für } i = j \\ 0 \text{ für } i \neq j \end{cases}$

$\underline{c}$ ist ein $2N \times 1$ Vektor

$\underline{c} = [0, 0, \ldots, C, C]^T$

$C$ klein: Großer Margin aber wenige Punkte robust richtig klassifiziert




$C$ groß: Kleiner Margin aber viele Punkte robust richtig klassifiziert

# 6.5 Kernels für Nichtlineare Separierbarkeit

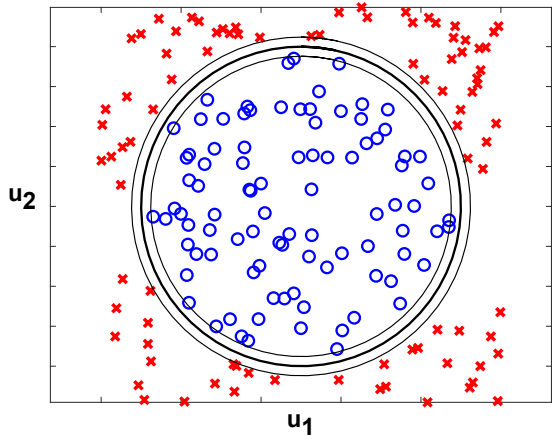
## Idee der Verwendung von Kernel-Funktionen

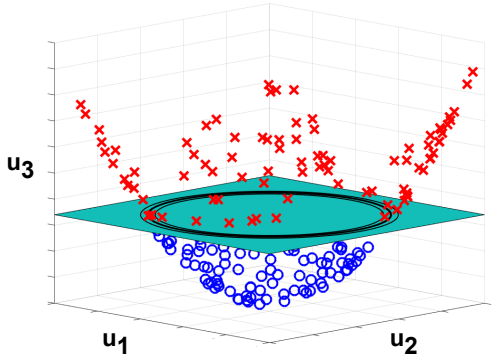
- Im Gegensatz zur linearen Soft Margin SVM:
  **Exakte** Klassifizierung von nicht linear separierbaren Daten.
- Transformation der Daten in einen höherdimensionalen Raum mit Hilfe einer vektoriellen Funktion:
  - lineare Trennung der Daten im höherdimensionalen Raum durch eine (Hyper-)Ebene wieder möglich!
  - Rücktransformation des Schnittes der linearen Klassifikationsgrenze mit der Transformationsfunktion ergibt eine nichtlineare Grenze.

$$\begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \rightarrow \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} \quad \text{z.B.} \quad \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \rightarrow \begin{bmatrix} u_1 \\ u_2 \\ u_1^2 + u_2^2 \end{bmatrix} \quad \text{(Bilder rechts)}$$

$$\underline{u}_+ = \underline{\phi}(\underline{u}) \quad \text{mit: } \underline{u}_+ \text{ vergrößerter Vektor, } \underline{\phi} \text{ Vektorfunktion}$$

- In den SVM-Gleichungen kommt der Eingangsvektor $\underline{u}_i$ allerdings immer nur in Form eines **Skalarprodukts** zweier Punkte $\underline{u}_i^T \underline{u}_j$ vor, nie alleine!
  ⇒ Keine explizite Vektortransformation nötig, es genügt eine Funktion mit bestimmten Eigenschaften (Mercer-Bedingungen), die einen Skalar zurückliefert (**Kerneltrick**):
  **Kernelfunktion** $K(\underline{u}_i, \underline{u}_j) = \underline{\phi}^T(\underline{u}_i) \cdot \underline{\phi}(\underline{u}_j)$



# 6.5 Kernels für Nichtlineare Separierbarkeit

## Kernel-SVM Gleichungen und Beispiele

- **Mercer-Bedingungen:** Eine Funktion muss bestimmte Bedingungen erfüllen, um als Kernel verwendet werden zu können:
  - Symmetrie:
  - positiv semidefinite Kernelmatrix $\underline{K}$ mit den Elementen $K_{ij}$:

$$K(\underline{u}_i, \underline{u}_j) = K(\underline{u}_j, \underline{u}_i)$$

$$\underline{K} \text{ mit } K_{ij} = K(\underline{u}_i, \underline{u}_j)$$

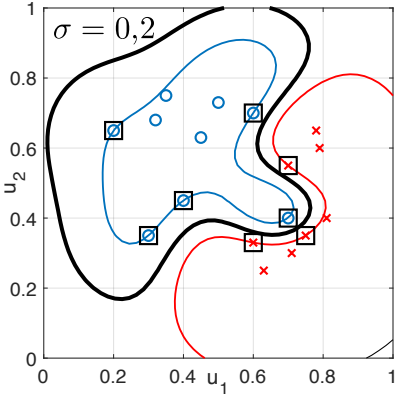
- Übliche **Kerneltypen** sind z.B.:
  - linearer Kernel (wie bisher):
  - polynomialer Kernel (Grad $d$):
  - Gauß-Kernel (Parameter $\sigma$):

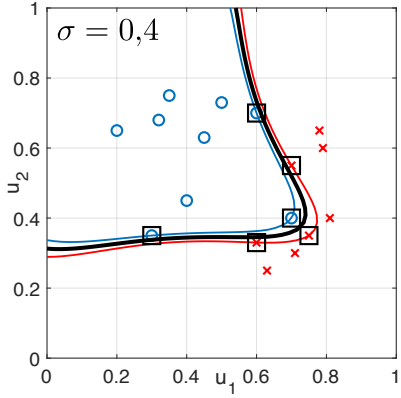
$$K(\underline{u}_i, \underline{u}_j) = \underline{u}_i^T \cdot \underline{u}_j$$

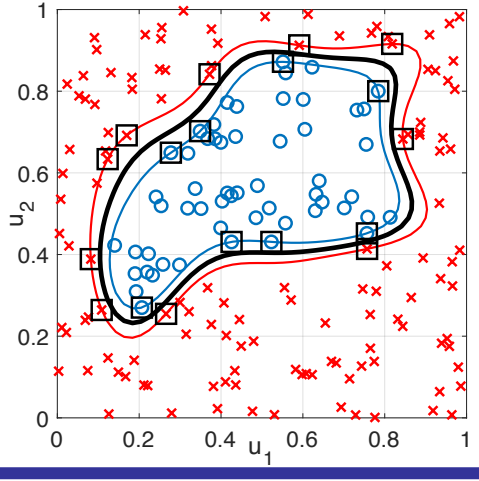
$$K(\underline{u}_i, \underline{u}_j) = (\underline{u}_i^T \cdot \underline{u}_j + 1)^d$$

$$K(\underline{u}_i, \underline{u}_j) = e^{-\frac{(\underline{u}_i - \underline{u}_j)^T \cdot (\underline{u}_i - \underline{u}_j)}{2\sigma^2}}$$

- Je nach Kernel: Transformation in Raum mit unendlich vielen Dimensionen, daher Separation immer möglich (z.B. Gauß-Kernel, Reihenentwicklung der e-Funktion).
- **Klassifikationsbeispiele mit Gauß-Kernel** (Bilder rechts):
  - **Oben:** Nicht separierbares Beispiel von zuvor (wie bei linearer Soft Margin SVM). Mit Gauß-Kernel separierbar, Wahl der Standardabweichung $\sigma$ beeinflusst das Ergebnis, bei größerem $\sigma$ wird die Grenze weniger flexibel.
  - **Unten:** Beispiel mit umschlossener Klasse. Mit Gauß-Kernel separierbar. Es ist deutlich zu erkennen, dass sich die Margin-Breite, der örtlichen Punktdichte anpasst.



## Approximation einer SVM durch eine Least Squares SVM (LS-SVM)

- Nachteile Optimierung SVM gegenüber Least Squares Problemen:
  - Quadratisches Optimierungsproblem (insbesondere mit vielen Nebenbedingungen) ist aufwändiger zu lösen.
  - Leave-One-Out Fehler zur Ermittlung von Hyperparametern (z.B. $\sigma$ oder $d$ bei Kernel-SVM) ist aufwändiger zu bestimmen.
- Vorgehen zur Überführung der SVM in ein LS-Problem:
  - Ausgangspunkt ist die Soft Margin SVM mit Schlupfvariablen.
  - Schlupfvariablen (und auch die Lagrange-Multiplikatoren $\alpha_i$) dürfen nun auch negativ werden ⇒ Jetzt Bestrafung des Quadrats der Schlupfvariablen in der Verlustfunktion nötig.
  - Alle weiteren Ungleichheitsnebenbedingungen werden durch Gleichheitsnebenbedingungen ersetzt.
- Folgen dieser Vorgehensweise:
  - Wenn der Kernel flexibel genug ist und $C$ groß, liegen alle Datenpunkte nun **auf den +1 oder -1 Höhenlinien!**
  - Alle Punkte sind Supportvektoren mit $\alpha_i > 0$ (keine Sparsity, d.h. keine Reduktion auf wenige Support Vektoren).
  - **Aber Vorteil:** Nur noch ein **lineares Gleichungssystem** zu lösen!

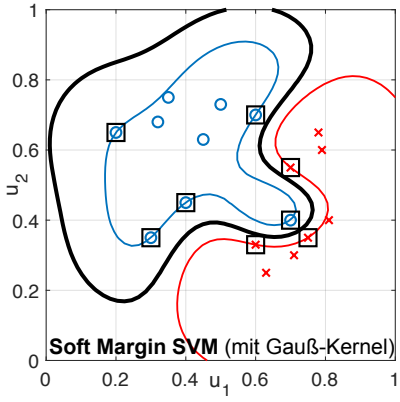
$$\left[\begin{array}{c|c} 0 & \underline{y}^T \\ \hline \underline{y} & \underline{\Omega} + \underline{I}/C \end{array}\right] \cdot \left[\begin{array}{c} b \\ \hline \underline{\alpha} \end{array}\right] = \left[\begin{array}{c} 0 \\ \hline \underline{1} \end{array}\right]$$

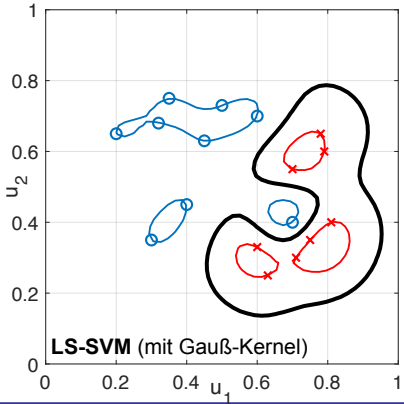
Elemente der Matrix $\underline{\Omega}$: $\Omega_{ij} = y_i y_j K(\underline{u}_i, \underline{u}_j)$

$\underline{1} = [1 \;\; 1 \;\; \cdots \;\; 1]^T$

$C$ : Straffaktor wie bei Soft Margin SVM

Klassifikationsfkt.: $y(\underline{u}_{neu}) = \sum_{i=1}^{N} \alpha_i y_i K(\underline{u}_{neu}, \underline{u}_i) + b$



# 6.7 Unterschiede zu RBF-Netz

## Vergleich nichtlinearer Kernel SVM bzw. LS-SVM mit RBF-Netz

| | SVM | LS-SVM | RBF-Netz |
|---|---|---|---|
| Parameter | Gewichte + Offset | Gewichte + Offset | Gewichte (kein Offset) |
| Optimierung | QP | Ridge Regression | LS oder Ridge Regression |
| Numerik / Robustheit der Opt. | gut | besser | am besten |
| Margin Maximierung | ja | ja | nein |
| Komplexität des Klassifikators | Anzahl der Support Vektoren | Anzahl der Datenpunkte + 1 | Anzahl der Datenpunkte |

## Weitere Bemerkungen

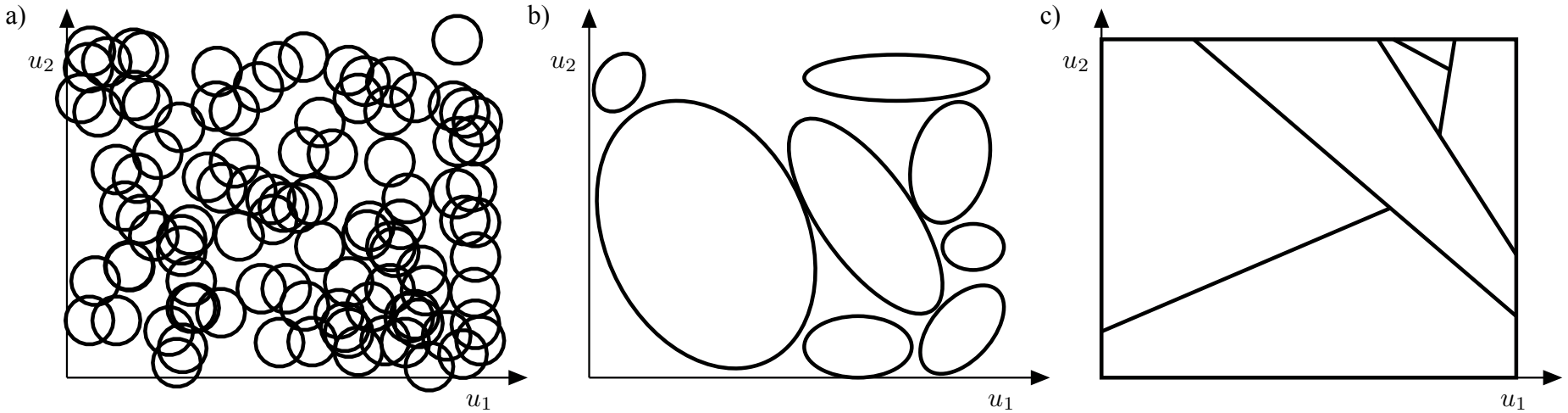
- In SVMs entsteht eine **sparse Lösung**. Nur die Support Vektoren beeinflussen die Klassifikationsgrenze. Dadurch entsteht eine große Robustheit der Klassifizierung und eine **sehr geringe Gefahr für Overfitting**.
- Ähnliche Effekte (Sparsity) können bei LS-SVM und RBF-Netzen durch aufwändige **Strukurselektionsverfahren** erzeugt werden, bei denen mittels Forward Selection, Backward Elimination oder eine Kombination daraus (Stagewise Selection) die Kernels nur auf wenige Datenpunkte gelegt werden. Alternativ können die Kernel-Positionen auch optimiert werden.
- **Numerisch** hat das **RBF-Netz** die **günstigsten Eigenschaften**, da die Matrizen symmetrisch sind (kein Offset).
- Alle 3 Ansätze sind **gut für hochdimensionale Probleme** aber **schlecht für sehr viele Daten** geeignet.

# 6.8 Modellierungsparadigmen

a) Kernel-basiert: Auf jedem Datenpunkt liegt ein Kernel oder Basisfunktion → SVM

b) Cluster-basiert: Zu jedem Cluster gehört eine Einheit, z.B. lokale (lineare) Modelle

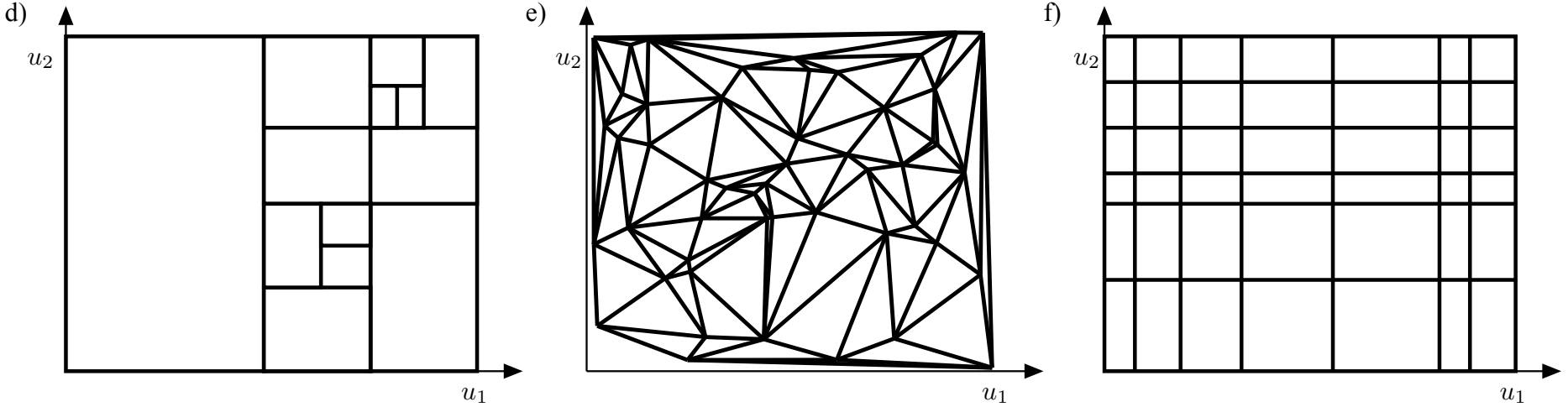
c) Achsen-schräge Partitionierung: Der Raum wird durch schräge Schnitte in Regionen zerlegt; zu jeder Region gehört eine Einheit



**d) Achsen-orthogonale Partitionierung: Der Raum wird durch orthogonale Schnitte in Regionen zerlegt; zu jeder Region gehört eine Einheit**

**e) Delaunay-Triangulierung: Der Raum wird in kompakte Dreiecke (2D), Tetraeder (3D), ... zerlegt mit *n*+1 Ecken bei *n* Dimensionen; zu jedem Dreieck gehört eine Einheit**

**f) Gitter: Jeder Eingang/Dimension wird in Levels eingeteilt; diese werden alle miteinander kombiniert**



# 7. Dichteschätzung

# Inhalt



Meist wird in diesem Skript als Eingangsgröße die Variable *u* verwendet.
Bei manchen Bildern aus externen Quellen kann es aber auch die Variable *x* sein!

## Bedeutung der Dichteschätzung?

- Leistungsfähigkeit datengetrieben Methoden hängt entscheidend von der Qualität der Daten ab.
- Ein entscheidender Faktor für die **Qualität** der Daten ist deren räumliche **Verteilung** (und bei dynamischen Modellen auch deren zeitliche Verteilung).
- Zwei Wege der **Datenakquisition**:
  - aktiv: im Laborversuch oft möglich, in der realen Anwendung nur manchmal: Versuchsplanung (*design of experiments*)
  - passiv: im laufenden Betrieb wird beobachtet und gemessen, was passiert
  - gemischt: erst passiv, dann aktive Auswahl von vielversprechenden Anteilen der durchgeführten Messungen

→ **Dichteschätzung** erlaubt eine **qualitative Beurteilung** der Datenverteilung

## Gewünschte Verteilung der Daten?

- **Wenig Vorwissen** → Universeller Ansatz: **Gleichverteilung**, d.h. Daten überall (wo zulässig) mit identischer Dichte.
- **Vorwissen** über
  - wichtige oder kritische Regionen
  - systematische regionale Unterschiede der Störungen bzw. des Rauschens
  - räumlich variierende Genaugigkeitsanforderungen

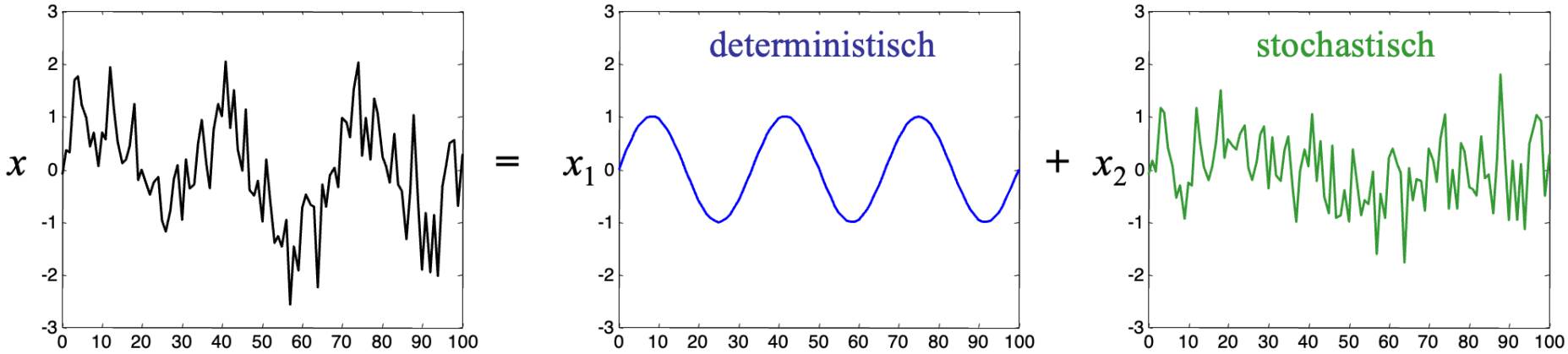
→ Gewünschte **Über-** und **Unterrepräsentation** bestimmter **Regionen.**

**Beispiel:** Wie wichtig ist für die Kalibrierung einer bestimmten Kfz-Fahrfunktion Stadt-, Landstraßen- und Autobahn-Verkehr?

## Deterministische oder stochastische Beschreibung?

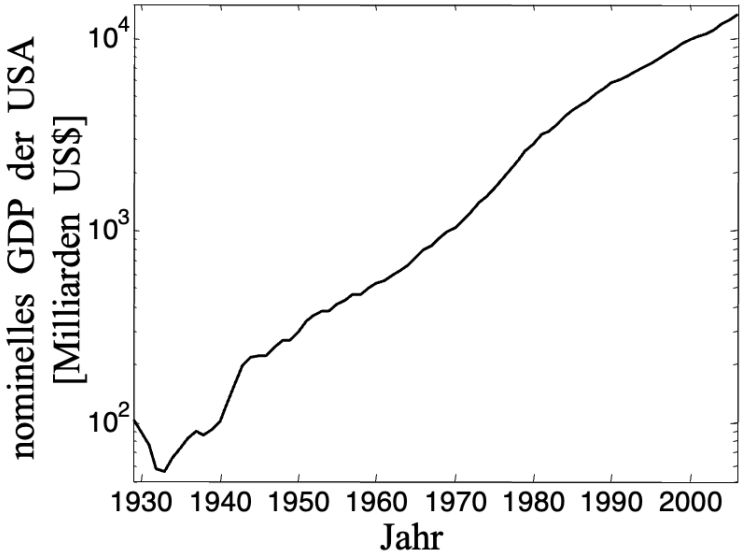
Wenn sich eine Größe zufällig oder scheinbar zufällig verhält, dann ist es sinnvoll, diese Größe stochastisch, d.h. mittels einer *Zufallsvariablen* zu beschreiben. Dabei spielt es keine Rolle, ob das Verhalten *echten* zufälligen Ursprungs ist, wie z.B. radioaktiver Zerfall, oder ob es durch Verknüpfung unüberschaubar vieler komplexer Einflüsse verschiedenster Herkunft „nur“ zufällig schwankend *wirkt*, wie z.B. Börsenkurse, deren Ursprung sicherlich rein deterministisch ist (niemand kauft oder verkauft Aktien zufällig).

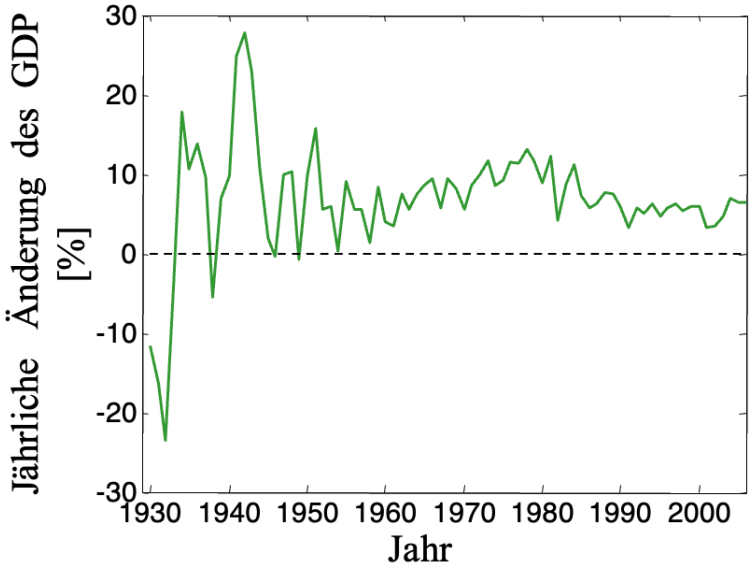
Oft kommt es auch vor, dass eine Größe zu einem Teil deterministisch und zu einem weiteren Teil stochastisch beschrieben wird. Typisch ist die Beschreibung einer Größe $x = x_1 + x_2$ als Summe eines deterministischen Anteils $x_1$ und eines zufälligen Anteils $x_2$.



## *Beispiel:* Bruttoinlandsprodukt (BIP)

Das Bruttoinlandsprodukt (BIP) als Wohlstandsmaß für Nationen weist in allen entwickelten Ländern einen klaren exponentiellen Wachstumstrend (linear in logarithmischer Skalierung) auf. Dieser Trend lässt sich rein deterministisch beschreiben. Diesem Trend überlagert sind Einflüsse durch Konjunkturschwankungen (zyklisch), Börsencrashs, Ölkrisen oder Kriege (Schocks) u.ä. Diese lassen sich (teilweise) stochastisch beschreiben. Ein Modell für das BIP setzt sich also einem deterministischen und einem stochastischen Anteil zusammen.



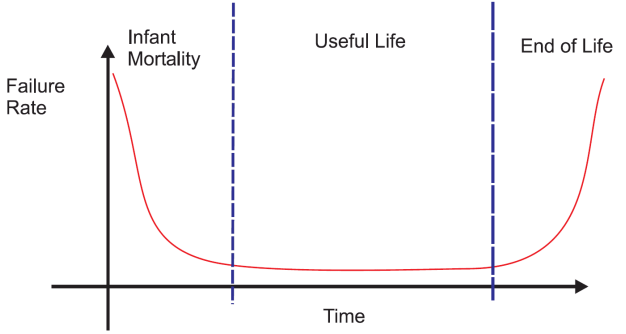



# 7.1 Wahrscheinlichkeitsdichten

## Zufallsvariable

Eine Zufallsvariable kann eine diskrete oder eine kontinuierliche Größe sein:

- Diskrete Zufallsvariable können nur eine *endliche* Zahl an möglichen Ereignissen annehmen, z.B. „Kopf“ oder „Zahl“ beim Münzwurf oder 1, 2, 3, 4, 5 oder 6 beim Würfeln.
- Kontinuierliche Zufallsvariablen können einen kontinuierlichen Zahlbereich annehmen, z.B. alle reellen Zahlen von 0 bis ∞ bei der Modellierung einer mittleren Ausfallzeit (*mean-time-to-failure*) oder alle reelle Zahlen zwischen $-\infty$ und ∞ bei bei der Modellierung eines Störsignals oder alle reellen Zahlen zwischen $-q/2$ und $q/2$ mit einem Quantisierungsintervall der Breite $q$ zur Modellierung des Quantisierungsrauschens.




Quelle: `https://www.electrontest.com/mtbf-calculation/`

## Zufallsprozess

Ist eine Zufallsvariable eine Funktion der kontinuierlichen Zeit $t$ oder der diskreten Zeit $k$, dann spricht man von einem *Zufallsprozess*.

Wir fokussieren uns auf kontinuierliche Zufallsvariablen, die evtl. von der diskreten Zeit abhängen können.

## Herleitung über Relative Häufigkeiten / Histogramme

Wenn wir eine Zufallsgröße *N* mal in identischer Weise messen, wird jede dieser Messungen wegen der zufälligen Fluktuationen ein anderes Ergebnis zeitigen. Um einen guten Überblich über die Qualität und Reproduzierbarkeit dieser Größe zu bekommen, ist es sinnvoll und äußerst hilfreich, die Messungen in einem **Histogramm** aufzutragen.

Hierzu werden die Messungen in Intervalle der Größe $\Delta x$ aufgeteilt.
Die Anzahl der Messungen, die in jedes Intervall *i* fallen werden **absolute Häufigkeiten $H_i$** genannt. Jede Messung fällt exakt in ein Intervall (mit $n_I$ Intervallen insgesamt):

$$\sum_{i=1}^{n_I} H_i = N$$

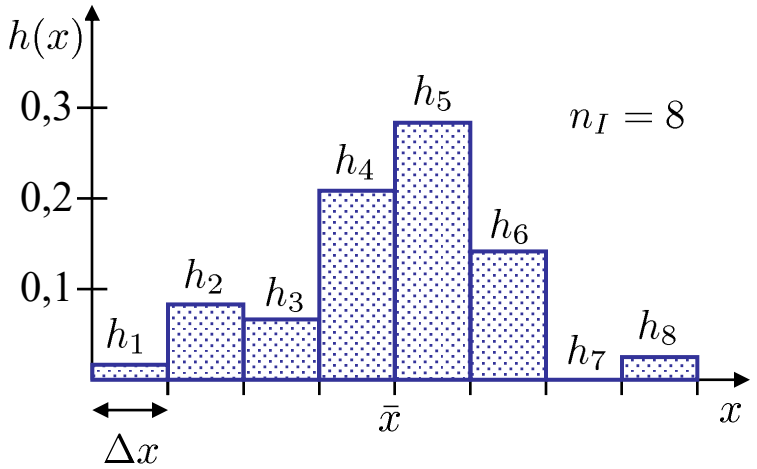
Empfehlung für Anzahl der Intervalle: $n_I \approx \sqrt{N}$

Die **relative Häufigkeit $h_i$** beschreibt den jeweiligen **Anteil** von $H_i$, der in das Intervall *i* fällt:

$$h_i = \frac{H_i}{N}$$

Die relativen Häufigkeiten summieren sich zu 100%: $\sum_{i=1}^{n_I} h_i = 1 = 100\%$

Histogramm



## Wahrscheinlichkeitsverteilungsdichte (*probability density function, pdf*)

Mit Hilfe des Histogramms bekommt man leicht eine gute Übersicht über die Verteilung der Messungen, d.h. z.B. wie stark sie um ihren Mittelwert $\bar{x}$ streuen. Wenn wir die Anzahl an Messungen *N vergrößern* und gleichzeitig die Größe $\Delta x$ der Intervalle *verkleinern* (also die Auflösung verbessern), dann konvergiert das **Histogramm** gegen die **Wahrscheinlichkeitsverteilungs-dichte**.

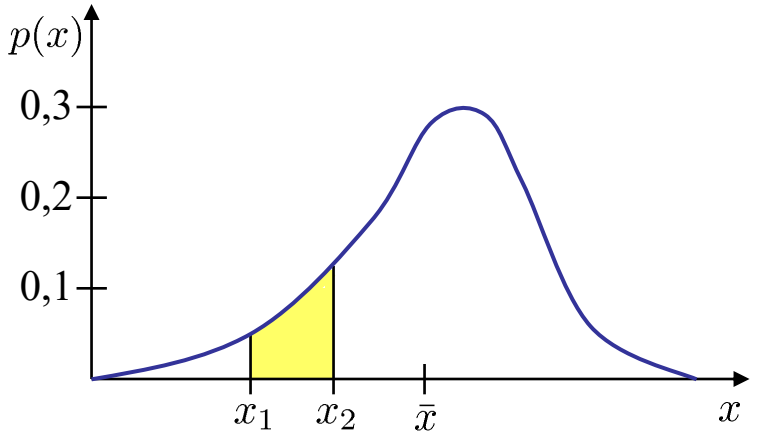
$$p(x) = \lim_{\Delta x \to 0} \left( \lim_{N \to \infty} h(x) \right)$$

Erinnerung: $\int_{-\infty}^{\infty} p(x)dx = 1$

Die Dichte $p(x)$ ist eine kontinuierliche Funktion ohne Sprünge.
Wir können mit ihr die Wahrscheinlichkeit berechnen, dass eine Messung in ein bestimmtes Intervall ($x_1$ $x_2$] fällt:

$$P(x_1 < x \leq x_2) = \int_{x_1}^{x_2} p(x)dx$$

Die wahre Dichte $p(x)$, der Messungen zugrunde liegen, ist normalerweise unbekannt. Typischerweise werden anwendungsabhängig realistische Annahmen über diese Dichte anhand von Vorwissen und der Histogramme getroffen. In vielen Fällen wird eine **Normalverteilung** angenommen.
In einigen Folien erfahren wir, warum?! Stichwort: Zentraler Grenzwertsatz.



## Herleitung über Wahrscheinlichkeitsverteilungsfunktion (kurz: Verteilung oder *cdf*)

Betrachten wir eine kontinuierliche Zufallsvariable $x$ mit dem Wertebereich von $-\infty$ bis $\infty$. Wie wahrscheinlich ist welcher Zahlenwert? Diese Information steckt in der **Wahrscheinlichkeitsverteilungsfunktion** (*cumulative density function, cdf*) $F(x)$. Diese gibt an, mit welcher Wahrscheinlichkeit ein Wert $x \leq x_0$ von der Zufallsvariablen angenommen wird. D.h. die Wahrscheinlichkeitsverteilungsfunktion ist definiert als

$$F(x_0) = p(x \leq x_0)$$

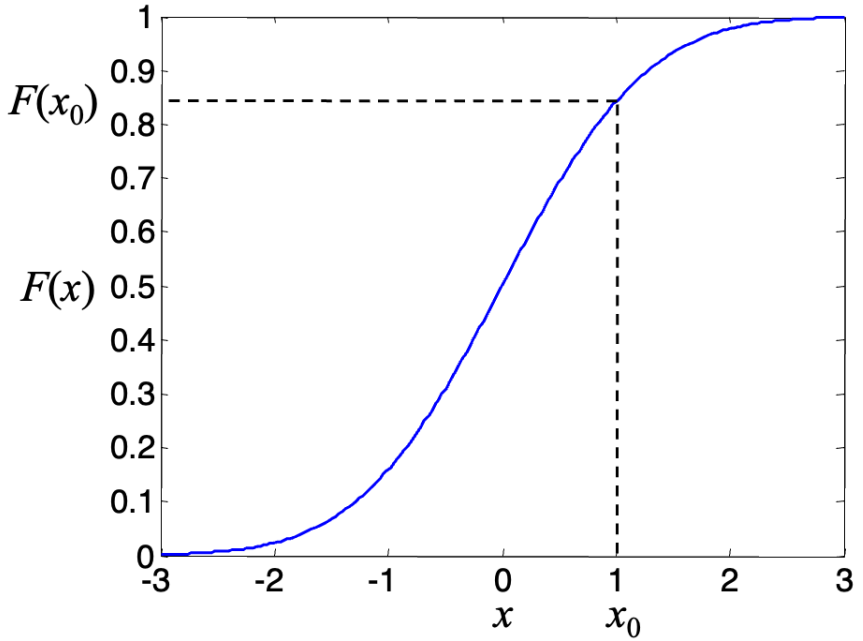
Da kein Wert kleiner sein kann als $-\infty$, gilt:

$F(-\infty) = 0$.

Da alle Werte kleiner sein müssen als $\infty$, gilt:

$F(\infty) = 1$.

Zwischen $-\infty$ und $\infty$ ist $F(x)$ eine *monoton steigende* Funktion, da die Wahrscheinlichkeit dafür, dass die Zufallsvariable eine Zahl aus einem gegebenen Zahlenbereich annimmt mit der Größe des Zahlenbereichs größer werden muss.



## Wahrscheinlichkeitsverteilungsdichtefunktion (kurz: Dichte oder *pdf*)

Die Ableitung der *cdf* ist die **Wahrscheinlichkeitsverteilungsdichtefunktion** (*probability density function, pdf*) $f(x)$.
Die **Fläche** unterhalb der Dichte von $x = a$ bis $x = b$ entspricht der Wahrscheinlichkeit, dass die Zufallsvariable $x$ zwischen $a$ und $b$ liegt.

$$\int_a^b f(x)dx = p(a \leq x \leq b)$$

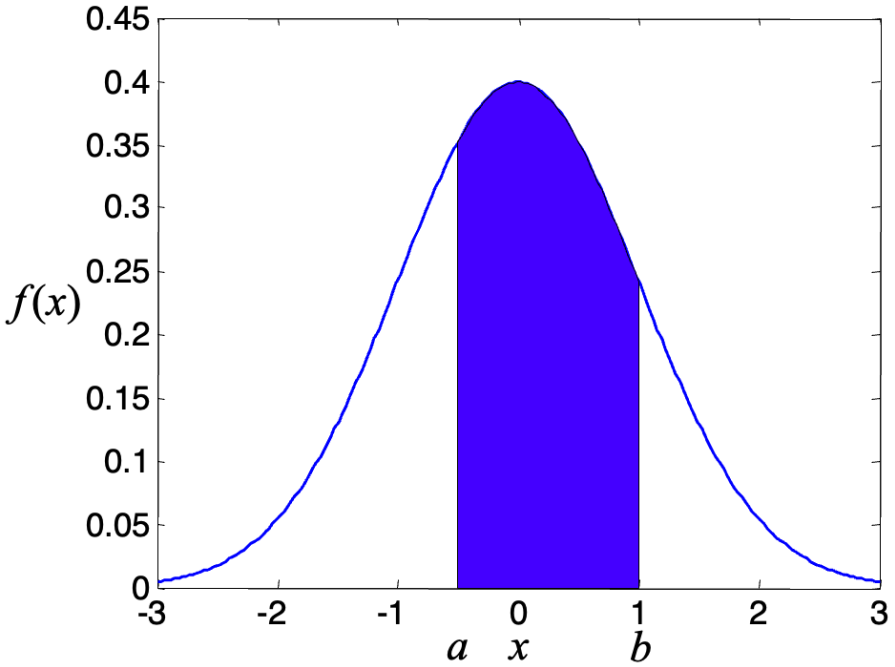
Damit ergeben sich folgende Beziehungen zwischen Dichte und Verteilung:

$$f(x) = \frac{dF(x)}{dx} \qquad F(x_0) = \int_{-\infty}^{x_0} f(x)dx$$

Außerdem gilt natürlich:

$$F(\infty) = \int_{-\infty}^{\infty} f(x)dx = 1$$

Für Wertebereiche $[c, d]$, welche die Zufallsvariable nicht annehmen kann, gilt $f(x) = 0$, $x \in [c, d]$.



***Beispiel 1:*** Wie wahrscheinlich ist es, dass eine Zufallsvariable mit obiger Verteilung einen Wert zwischen $-\infty$ und 1 annimmt?

*Antwort:* $p = F(1) = 0.84$.

***Beispiel 2:*** Wie wahrscheinlich ist nach obiger Verteilung das Auftreten einer Zahl zwischen –1 und 1?

*Antwort:* Die Wahrscheinlichkeit von Zahlen kleiner gleich 1 abzüglich der Wahrscheinlichkeit von Zahlen kleiner gleich –1: $p = F(1) - F(-1) = 0.84 - 0.16 = 0.68$.

***Beispiel 3:*** Wie wahrscheinlich ist nach obiger Dichte das Auftreten einer Zahl zwischen –0.5 und 1?

*Antwort:* $$p = \int_{-0.5}^{1} f(x)dx = F(1) - F(-0.5) = 0.84 - 0.31 = 0.53$$

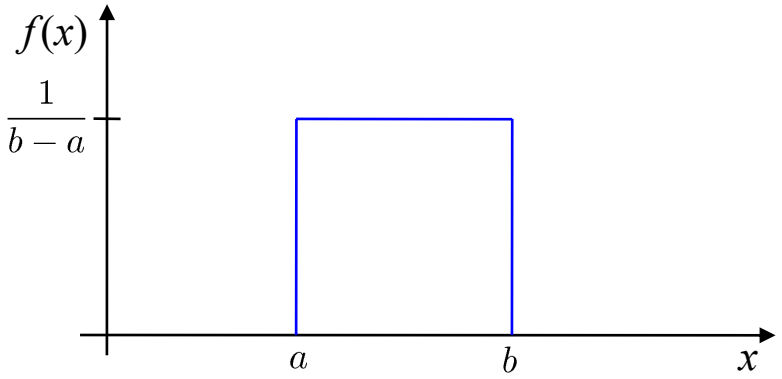
$$F(-0.5) = \int_{-\infty}^{-0.5} f(x)dx \qquad F(1) = \int_{-\infty}^{1} f(x)dx$$

## Wichtige Dichten

*Gleichverteilung*

$$f(x) = \begin{cases} \frac{1}{b-a} & a \le x \le b \\ 0 & \text{sonst} \end{cases}$$

- Alle Zahlen zwischen *a* und *b* sind gleich wahrscheinlich; alle anderen kommen nicht vor.

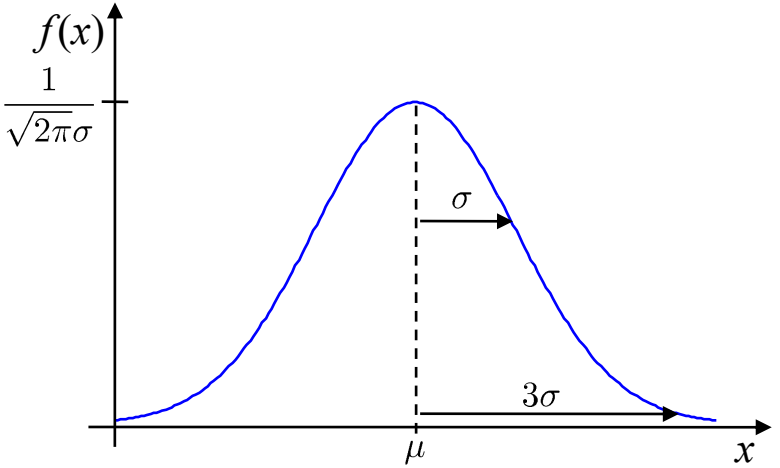



*Gauß- oder Normalverteilung*

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} \mathrm{e}^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

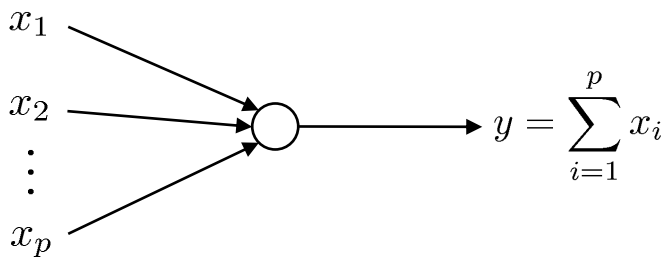
Carl Friedrich Gauß, 1777-1855
(www.wikepedia.org)

- Sehr viele Zufallsvariablen sind so verteilt.
- Ergibt sich als Grenzfall aus vielen anderen Dichten (Binomialverteilung, t-Verteilung).
- Summe aus vielen irgendwie verteilten Zufallsvariablen ist näherungsweise normalverteilt.



## Relevanz der Gauß-Verteilung

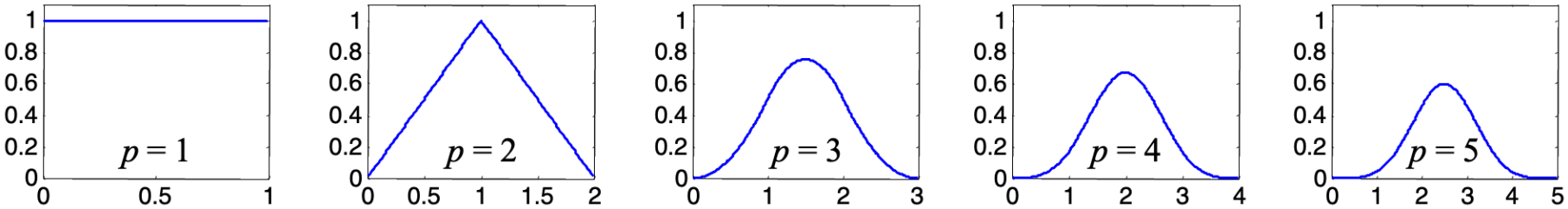
Nach dem *zentralen* **Grenzwertsatz der Statistik** ist die Summe mehrerer *unabhängiger*, irgendwie verteilter Zufallsvariablen $x_i$ näherungsweise **normalverteilt**. Die Näherung gilt um so besser, je mehr Zufallsvariablen addiert werden, und für $p \to \infty$ folgt $y$ einer Gauß-Verteilung.




Dabei dürfen die $x_i$ unterschiedlich und fast beliebig verteilt sein, nur einige exotische Verteilungen (z.B. Cauchy-Verteilung) sind ausgeschlossen.

Da viele stochastisch modellierte Größen aus einer Summe unüberschaubar vieler kleiner Einzeleffekte herrühren, ist es also nicht verwunderlich sondern nahezu zwangsläufig, dass die Gauß-Verteilung so häufig vorkommt.

Folgende Graphiken zeigen die Dichten von $y$ für verschiedene $p$, wenn die $x_i$ alle zwischen 0 und 1 gleichverteilt sind:



## Idee der Nichtparametrischen Dichteschätzung

- Es kommt nur auf die Verteilung der Punkte im ***Eingang*sraum** an; der Ausgang ist typischerweise irrelevant.
- Wo viele Datenpunkte liegen, muss die Dichte hoch sein.
- Wo wenige bzw. keine Datenpunkte liegen, muss die Dichte niedrig bzw. 0 sein.
- Es wird eine Kernel-Funktion **auf jeden Datenpunkt** gelegt.
- Dieser Kernel ist selbst eine **Wahrscheinlichkeitsdichte**, d.h. nichtnegativ und mit der Eigenschaft:

Die Matlab-Funktion `mvksdensity` führt eine Kerndichteschätzung durch

$$\int_{\min}^{\max} f(u)du = 1$$

- Außerdem soll der Kernel **lokal** sein, d.h. seinen größten Wert auf dem Datenpunkt aufweisen und **Abklingen**, je weiter man sich davon entfernt.
- Alle diese Kernels werden gemittelt.
- Daraus ergibt sich der **Kerndichteschätzer** (*Kernel Density Estimator*) oft auch ***Parzen***-Fenster-Methode genannt:

$$\hat{f}(u) = \frac{1}{N}\sum_{i=1}^{N} K(u - u(i))$$

mit dem Kernel $K(.)$ und den $N$ Datenpunkten, die bei $u(i)$, $i = 1, 2, ..., N$ liegen.

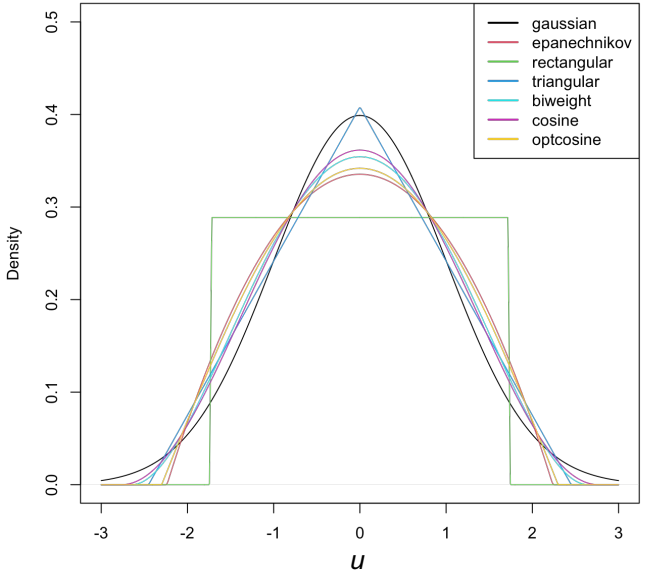
# 7.2 Kernel Density Estimation

## Beliebte Kernels

- Es gibt eine Vielzahl von möglichen Kernel-Funktionen. Beliebt sind u.a.:

Gauß $\frac{1}{\sqrt{2\pi}} \mathrm{e}^{-\frac{1}{2}u^2}$ Cauchy $\frac{1}{\pi(1+u^2)}$

Picard $\frac{1}{2} \mathrm{e}^{-|u|}$ Epanechnikov $\begin{cases} \frac{3}{4}(1-u^2) & \text{wenn } -1 \le u \le 1 \\ 0 & \text{sonst} \end{cases}$




Quelle: `https://bookdown.org/egarpor/NP-UC3M/kde-i-kde.html`

## ACHTUNG!

- In der Statistik-Literatur wird der Dichteschätzer oft in einer anderen Form geschrieben, nämlich als

$$\hat{f}(u) = \frac{1}{N \cdot h} \sum_{i=1}^{N} K_h(u - u(i))$$

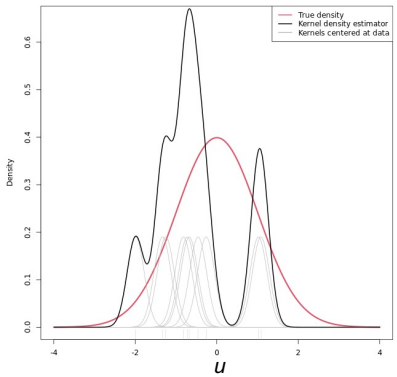
wobei *h* für die Breite / Bandbreite (*bandwidth*) steht. Hierbei sind die $K_h(.)$ keine Wahrscheinlichkeitsdichten, weil sie (je nach Wahl von *h*) keine Fläche / Integral = 1 abdecken. Teilweise sieht man die Formel auch ganz ohne normierenden Vorfaktor. Im Ergebnis (Kernels ausgeschrieben) sind all diese Formulierungen aber identisch.

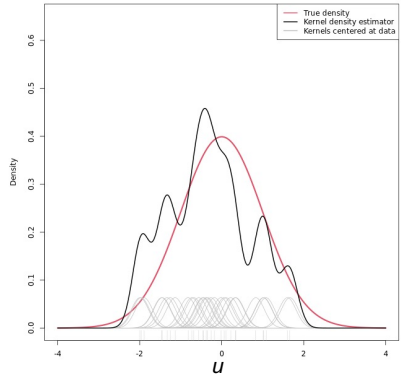
## Beispiele für $N = 10, 30, 100$ Datenpunkte

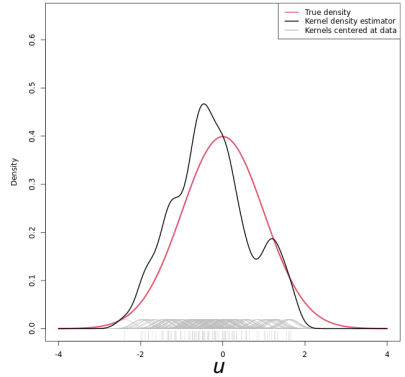
Unter `https://bookdown.org/egarpor/NP-UC3M/kde-i-kde.html` lässt sich schön und interaktiv mit Dichteschätzern spielen!

Gauß-Kernel
Rot = Original-Dichte
Schwarz = Dichteschätzung

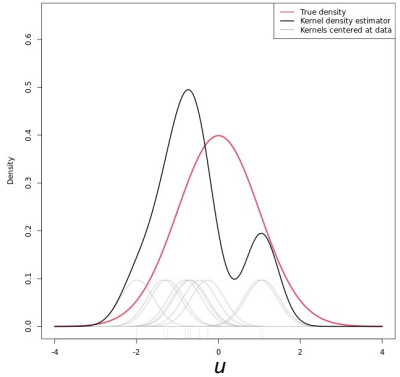
Kleine Kernel-Breite (Standardabweichung):

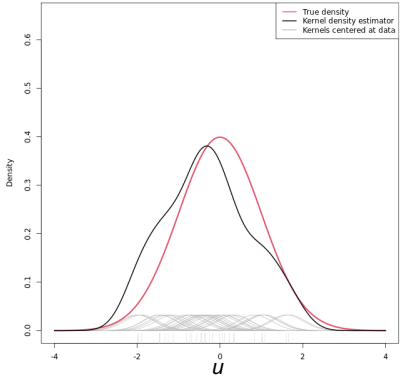


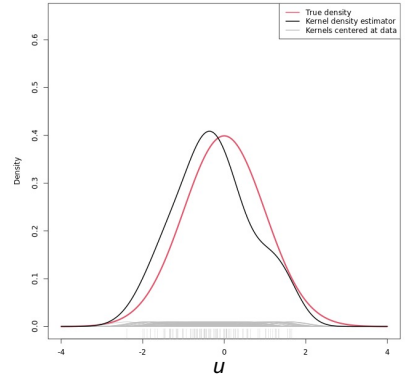




Mittlere Kernel-Breite (Standardabweichung):

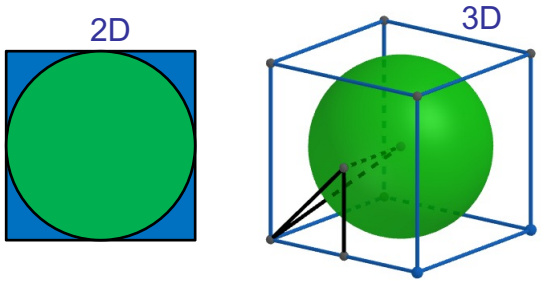


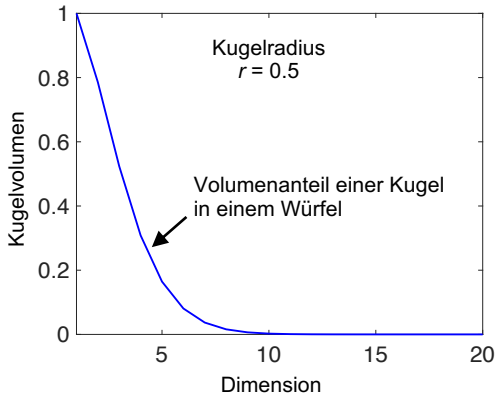



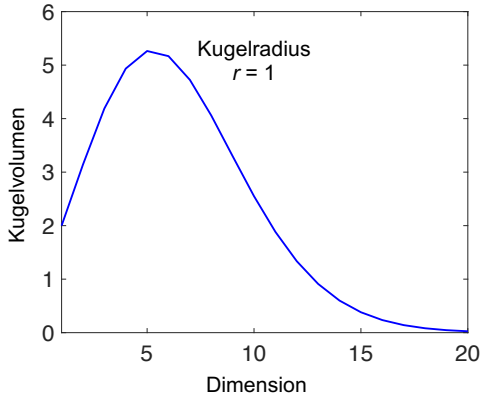
## Eigenschaften von Kerndichteschätzern

- Nach dem Satz von *Nadaraya* **konvergieren** die Kerndichteschätzer bei zunehmender Anzahl von Datenpunkten *N* (und damit auch zunehmender Anzahl von Kernels) gegen die **wahre Wahrscheinlichkeitsdichte** der Daten.
- Sie werden vorwiegend für relativ niedrig-dimensionale Probleme mit *n* = 1-6 Eingangsgrößen eingesetzt.
- Bei hochdimensionalen Problemen sind die Daten im Eingangsraum immer dünn verteilt. Das ist eine Manifestation des Fluchs der Dimensionalität (***curse of dimensionality***). Das bedeutet auch, dass im Hochdimensionalen alle Punkte von allen anderen weit entfernt sind. Es gibt dann keine sinnvoll interpretierbare Nachbarschaft. Für solche Probleme funktionieren Kerndichteschätzer nicht mehr gut.




Quelle: `https://www.geogebra.org/m/fefnqhga`



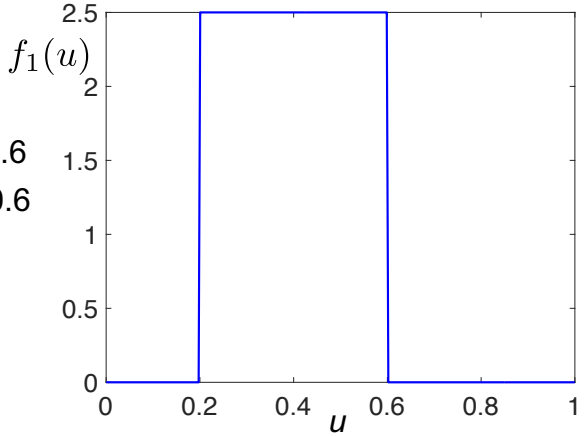



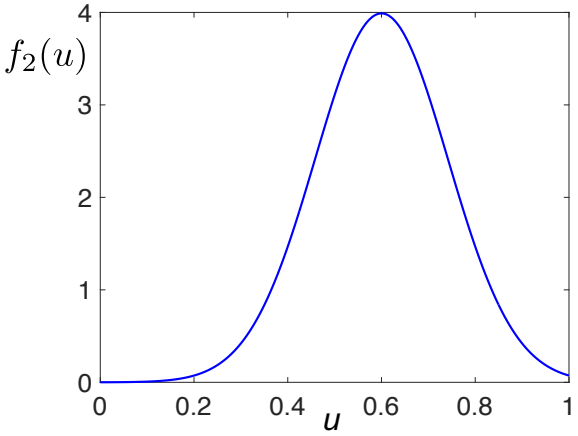

- Im Hochdimensionalen muss auf andere komplexere und leistungsfähigere Verfahren basierend auf neuronalen Netzen zurückgegriffen werden. Die Basis sind meist überwachte Verfahren der Dimensionsreduktion (arbeiten oft mit Projektionen), die mit dem Fluch der Dimensionalität besser umgehen können.

## Beispiel Wahrscheinlichkeitsdichte

- Mittelung aus 2 Zufallsvariablen
  - 1) Gleichverteilung zwischen $u$ = 0.2 und 0.6
  - 2) Normalverteilung mit Mittelwert bei $c$ = 0.6 und Standardabweichung $\sigma$ = 0.1:

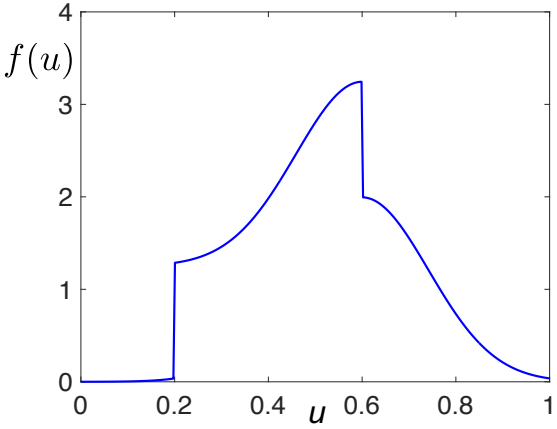
$$f_2(u) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2}\frac{(u-c)^2}{\sigma^2}\right)$$



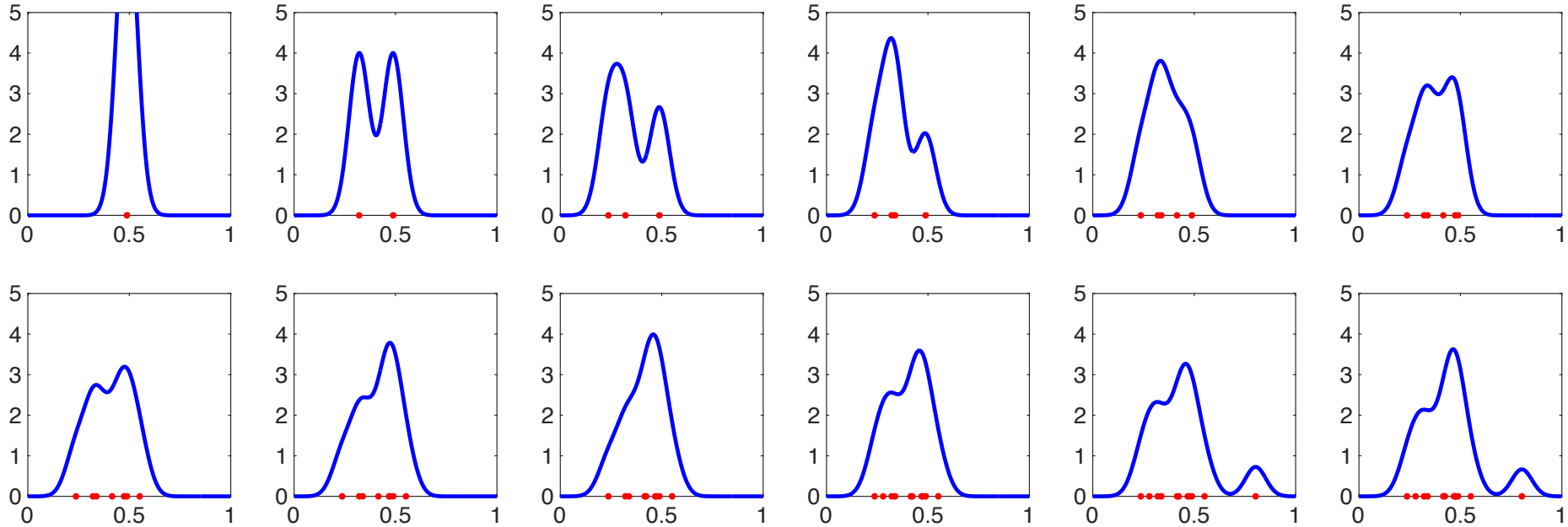

## Daten

- In den Beispielen werden aus dieser Verteilung Daten gezogen.
- Dies geschieht mittels den Matlab-Funktionen
  rand: Gleichverteilung
  randn: Normalverteilung
- `z1 = 0.2 + rand*0.4`
- `z2 = 0.6 + 0.1*randn`
- Mit Wahrscheinlichkeit von je ½ wird $z1$ oder $z2$ gezogen.



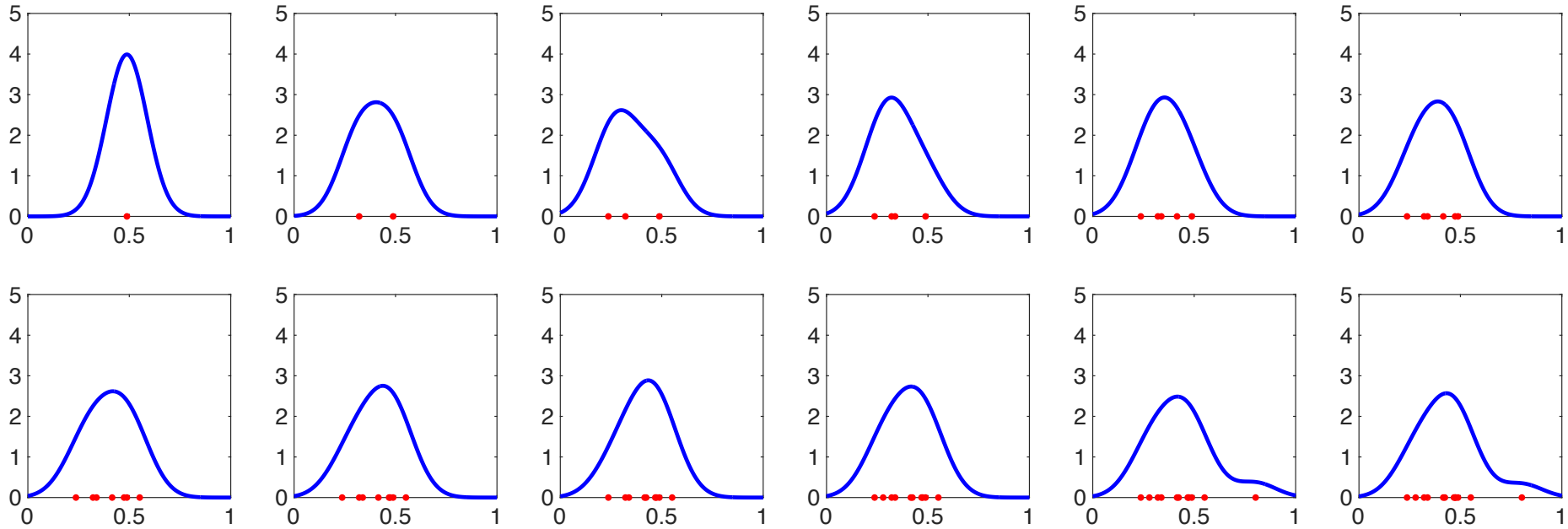
## Kernel Density Estimator für 1 ... 12 Datenpunkte

- Gauß-Kernel
- Breite / Standardabweichung $\sigma$ = 0.05 (gut)



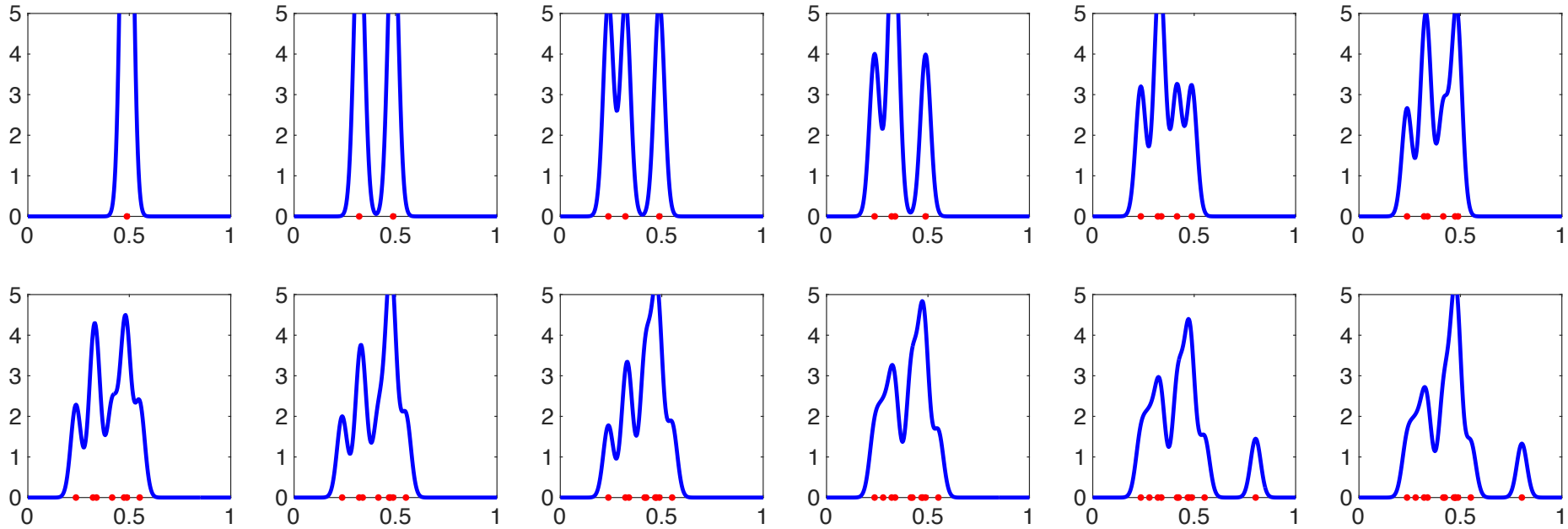
## Kernel Density Estimator für 1 ... 12 Datenpunkte

- Gauß-Kernel
- Breite / Standardabweichung $\sigma$ = 0.1 (zu glatt)



## Kernel Density Estimator für 1 ... 12 Datenpunkte

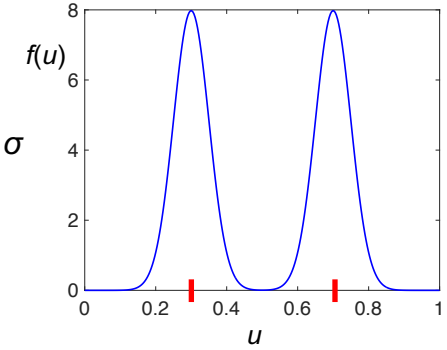
- Gauß-Kernel
- Breite / Standardabweichung $\sigma$ = 0.025 (zu rau)



# 7.3 Parametrierung der Kernel

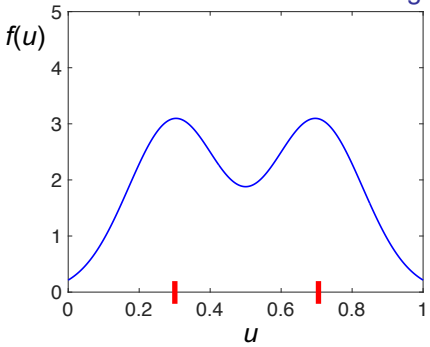
## Wichtigster Parameter der Kernels: Breite

- Oft Bandbreite (*bandwidth*) genannt
- Bei Gauß-Kernels ist Breite = Standardabweichung $\sigma$
- Meist der einziger Parameter
- Möglichkeiten:
  - eine Breite für alle Dimensionen / Eingänge
  - individuelle Breite je Dimension / Eingang




$\sigma$ = 0.05: Breite zu klein
hohe Auflösung
fast keine Generalisierung
*Undersmoothing*

$\sigma$ = 0.13: Breite gut
mittlere Auflösung
mittlere Generalisierung

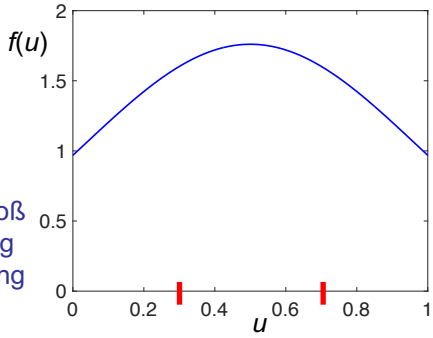



## Ziel: "Gute" Überlappung der Kernels

- Zu klein: Geschätzte Dichte zu rau (*rough*). Kaum Generalisierung.
- Zu groß: Geschätzte Dichte zu glatt (*smooth*). Lokale Unterschiede werden glattgebügelt.

## Wahl der Breite

- Heuristik bzw. Faustformel (*rule of thumb*) — *bandwidth selection*
- Optimierung — *bandwidth optimization*




$\sigma$ = 0.4: Breite zu groß
zu niedrige Auflösung
große Generalisierung
*Oversmoothing*

# 7.3 Parametrierung der Kernel

## Grundsätzliche Überlegungen zur Breite der Kernels

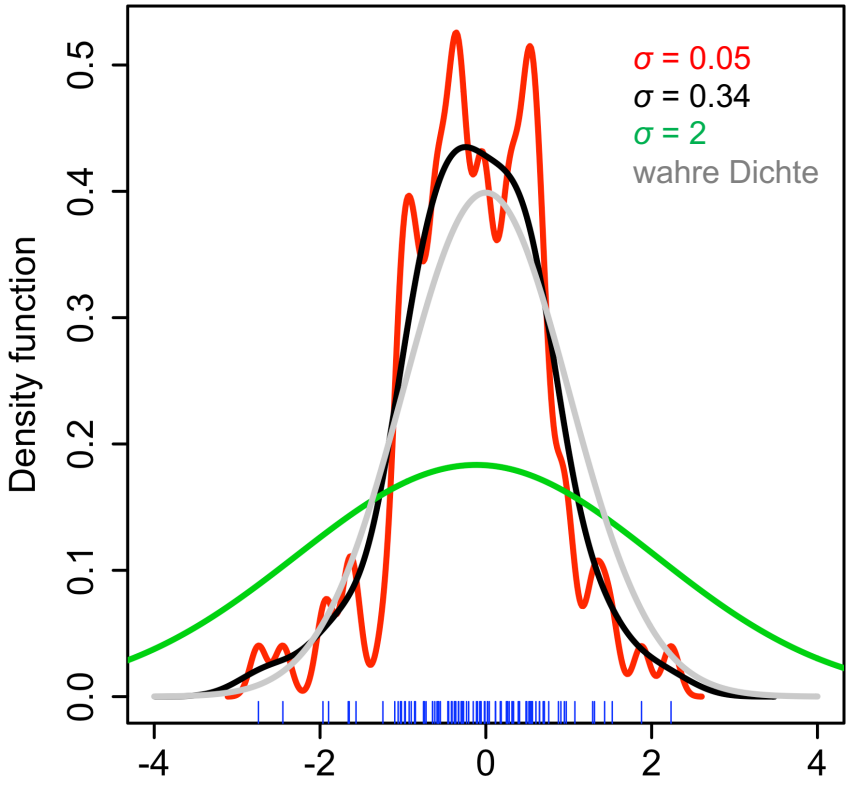
- Je mehr Datenpunkte *N* zur Schätzung der Dichte zur Verfügung stehen, desto kleiner sollte die Breite / Standardabweichung der Kernel sein. Die Auflösung wird damit höher.
- Je höherdimensional die Dichte ist, d.h. je mehr Eingänge, desto größer sollte die Breite / Standardabweichung der Kernel sein. Die Auflösung wird damit niedriger.

## Regel nach *Silverman*

- Für jede Dimension wird die Standardabweichung $\sigma$ separat berechnet:

$$\sigma = \left(\frac{4}{3N}\right)^{\frac{1}{5}} \cdot \sigma_{\text{data}}$$

- $\sigma_{\text{data}}$ ist die Streuung der Daten
- Der Kernel hat damit in jeder Dimension eine eigene Breite.




Quelle:
https://en.wikipedia.org/wiki/Kernel_density_estimation#/media/File:Comparison_of_1D_bandwidth_selectors.png

# 7.3 Parametrierung der Kernel

## Regel nach *Scott*

- Nur für näherungsweise **Normalverteilungen**!
- Für *n* Dimensionen wird eine voll besetzte **Kovarianzmatrix** des Kernels berechnet.

$$\underline{\Sigma} = \frac{1}{N^{n+4}} \cdot \underline{\Sigma}_{\text{data}}$$

- Die Einträge in $\underline{\Sigma}_{\text{data}}$ berechnen sich aus:

$$\hat{\sigma}_{ij} = \frac{1}{N-1} \sum_{k=1}^{N} (x_{ki} - \bar{x}_i)(x_{kj} - \bar{x}_j)$$

$$\underline{\Sigma} = \begin{pmatrix} \hat{\sigma}_{11} & \hat{\sigma}_{12} & \cdots & \hat{\sigma}_{1n} \\ \hat{\sigma}_{21} & \hat{\sigma}_{22} & \cdots & \hat{\sigma}_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \hat{\sigma}_{n1} & \hat{\sigma}_{n2} & \cdots & \hat{\sigma}_{nn} \end{pmatrix}$$

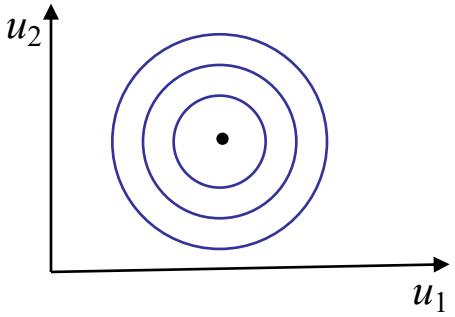
Normalverteilung:

$$\sim \mathrm{e}^{-\frac{1}{2}\underline{x}^T \underline{\Sigma}^{-1} \underline{x}}$$

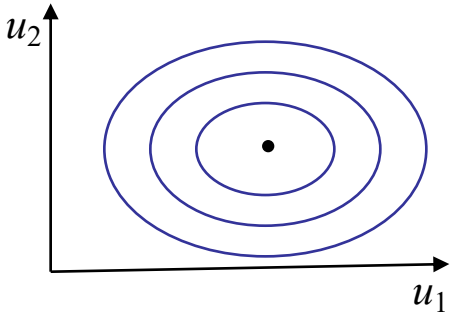
Für die Kovarianzmatrix $\underline{\Sigma}$ des Kernels existieren drei Fälle:
1. eine Standardabweichung in alle Dimensionen
2. individuelle Standardabweichungen je Dimension
3. voll besetzte Kovarianzmatrix
   (symmetrisch, postitiv definit)

Linien mit gleichem Mahalanobis-Abstand

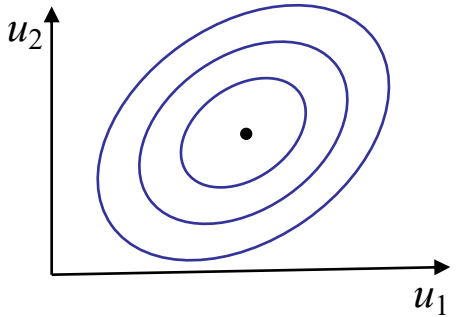
**1. $\underline{\Sigma}$ ~ Einheitsmatrix**




**2. $\underline{\Sigma}$ = Diagonalmatrix**




**3. $\underline{\Sigma}$ = sym. pos. def. Matrix**



## Optimierung der Breite der Kernels

- Optimierung ist **besser** aber deutlich aufwändiger als Faustformeln.
- Insbesondere besser bei **multimodalen** Dichten.
- Insbesondere wichtig für multivariate Dichten. Je **höherdimensionaler**, desto **wichtiger**!

- Die Optimierung der Kernel-Parameter muss auf neuen Daten erfolgen, weil die **Generalisierungsperformance** bewertet werden muss. 2 Möglichkeiten hierfür:
  - auf separatem **Validierungsdatensatz** → große Datenmengen notwendig
  - auf **Leave-One-Out**-Fehler, siehe Kapitel 3 → effizienter Umgang mit Daten aber rechenaufwändig
- Oft existieren **lokale Optima**. 2 Klassen von Optimierungsverfahren:
  - lokal, z.B. **Gradienten**-basiert → effizient und exakt, aber bleibt in lokalen Optima stecken
  - global, z.B. **Gitter**-basiert → einfach aber grob, bleibt aber nicht stecken
  - beides kann auch **kombiniert** werden, z.B. erst Gitter für guten Initialwert, dann Gradientenverfahren für Genauigkeit

- Es gibt eine große Anzahl an weiteren Methoden, die weit über die Diskussion in diesem Skript hinausgehen.
- Allerdings werden nichtparametrische Dichteschätzer für höherdimensionale Probleme sehr unzuverlässig, ca. für $n > 5-8$.

# 7.4 Multivariate Dichteschätzung

## Eigenschaften multivariater (mehrdimensionaler) Dichteschätzung

- Rechenaufwand steigt näherungsweise **linear** mit der **Dimension** (Euklidische Abstandsberechnung).
- Rechenaufwand steigt **linear** mit der Anzahl der Kernels / **Datenpunkte.**
- ABER: Der Raum wird **exponentiell** größer mit der **Dimension** („Fluch der Dimensionalität“).
- Daher müsste für eine gleiche **Punktabdeckung** / Punktdichte die Anzahl der Datenpunkte auch **exponentiell** steigen.
- In der Praxis ist das unrealistisch → **Qualität** (Auflösung und Robustheit) **sinkt** erheblich mit der **Dimension.**

## *Eine* Kernel-Breite

- Die Wahl *einer* (global identischen) Breite / Bandbreite der Kernel kann sehr **suboptimal** werden.
- Daten können in bestimmten Regionen dicht verteilt → impliziert Breite sollte klein sein
  und in anderen Regionen dünn verteilt sein → impliziert Breite sollte groß sein.
- Die Wahl *einer* Breite / Bandbreite wird immer **suboptimaler**, je **höherdimensionaler** das Problem ist.

## Variable Kernel-Breite

- Idee: Wahl der Kernel-Breite **antiproportional** zu geschätzten **Punktdichte.**
- Matlab-Funktion **`akde`**: Zdravko Botev (2021). Kernel Density Estimator for High Dimensions (https://www.mathworks.com/matlabcentral/fileexchange/58312-kernel-density-estimator-for-high-dimensions), MATLAB Central File Exchange. Retrieved November 10, 2021.

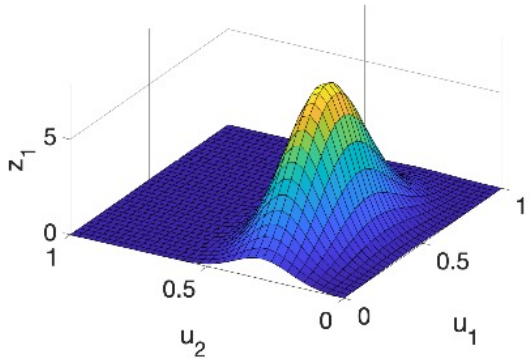
## Beispiel 2D-Wahrscheinlichkeitsdichte: Daten

- In den Beispielen werden aus dieser Verteilungen Daten gezogen.
- Dies geschieht mittels der Matlab-Funktion `randn`
- Eine multivariate Normalverteilung mit Kovarianzmatrix $\underline{\Sigma}$ kann über die Cholesky-Faktorisierung berechnet werden. Diese berechnet die „Wurzel“ der Kovarianzmatrix, quasi eine Standardabweichung im Mehrdimensionalen:

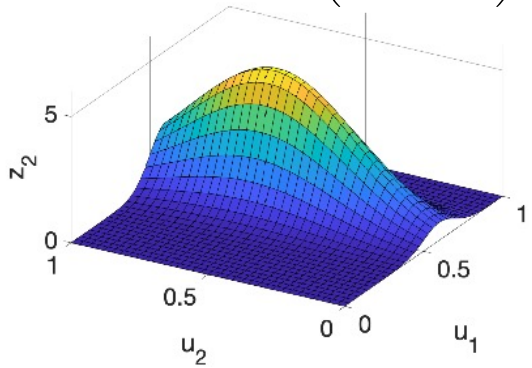
```
R = chol(Sigma)      % Kovarianzmatrix Sigma
z = mu + randn*R     % Mittelwert mu
```

- Für *z*: Mit Wahrscheinlichkeit von je ½ wird *z*1 oder *z*2 gezogen. Anzahl an Datenpunkten *N* = 50.

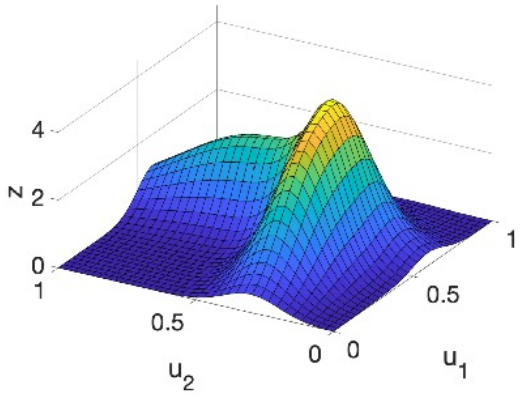
$$\underline{\Sigma}_1 = \begin{pmatrix} 0.2^2 & 0 \\ 0 & 0.1^2 \end{pmatrix}$$



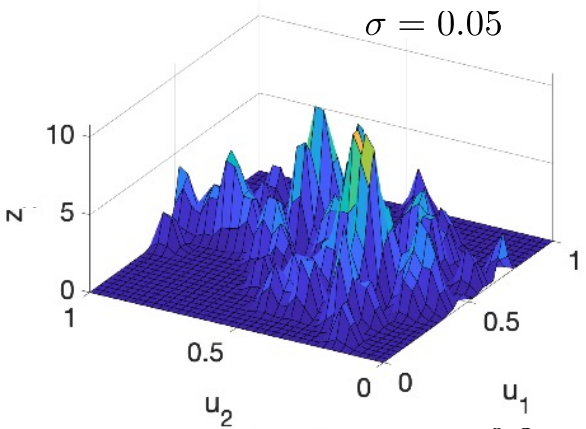

$$\underline{\Sigma}_2 = \begin{pmatrix} 0.1^2 & 0 \\ 0 & 0.3^2 \end{pmatrix}$$

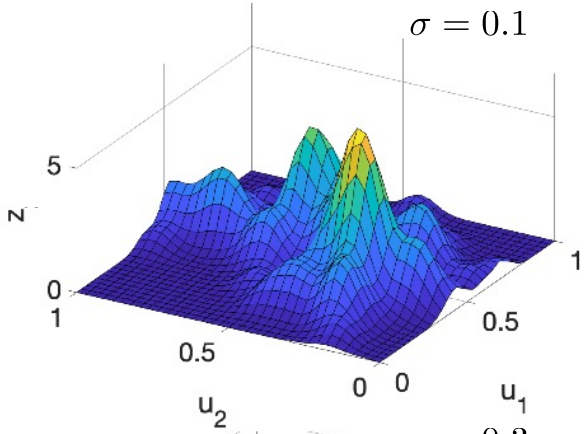


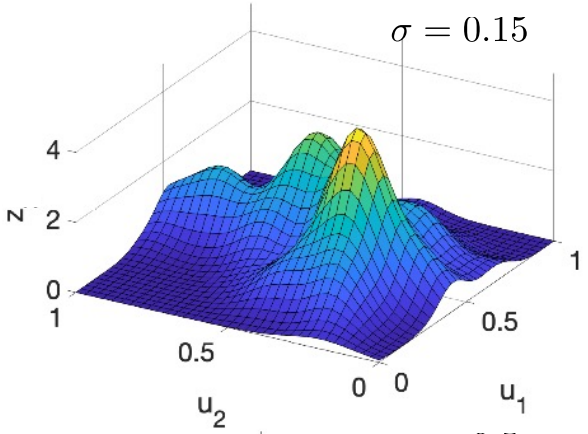

kombiniert:

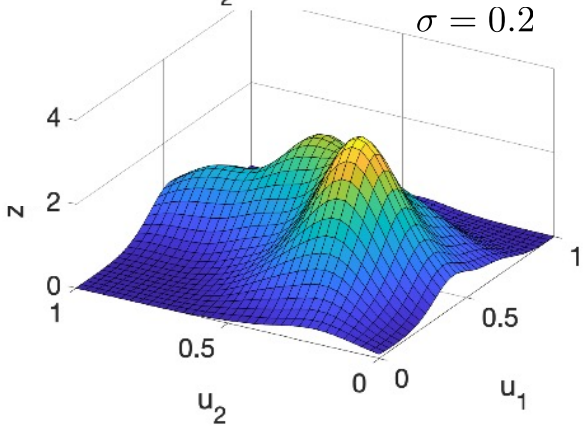


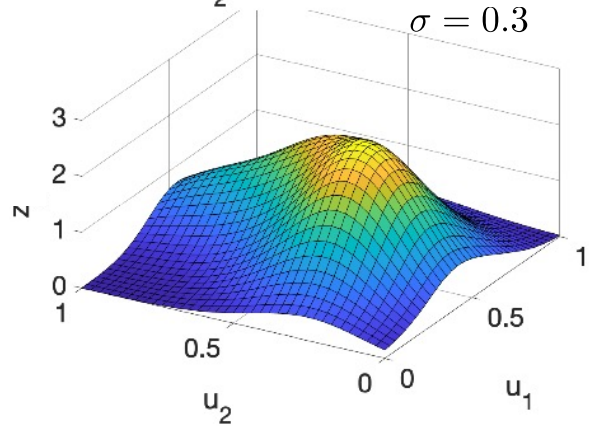
# 7.4 Multivariate Dichteschätzung

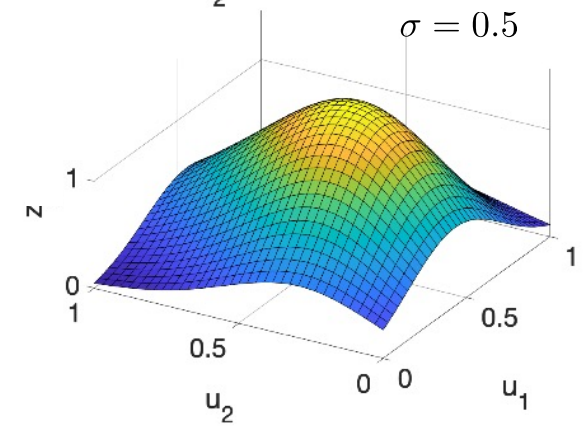


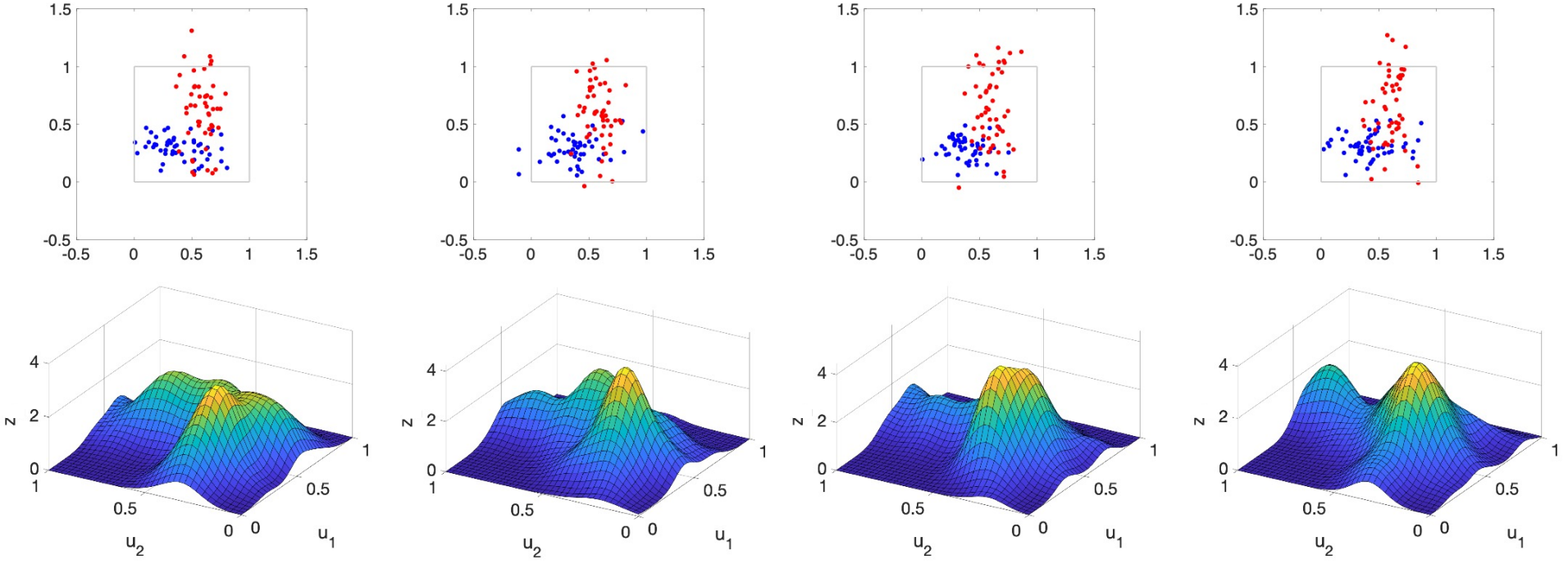



# 7.4 Multivariate Dichteschätzung

## Beispiele für verschiedene Realisierungen von jeweils 100 gezogenen Datenpunkten aus der Originaldichte



# 7.5 Randkorrektur

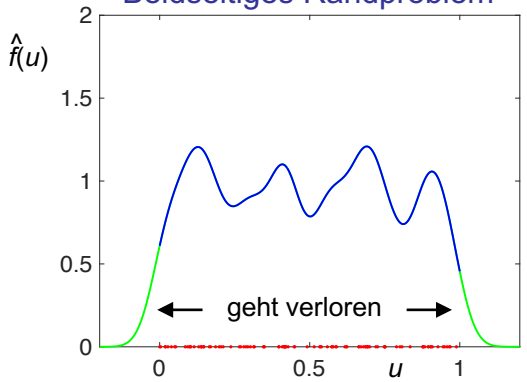
## Problem

- In vielen Anwendungen liegen einige oder alle Dimensionen / Eingänge nur in einem **begrenzten Intervall** [$u_{min}$ $u_{max}$].
- Beispiele sind
  - Fahrpedal oder Ventil oder Wirkungsgrad [0% 100%]
  - Drehzahl Verbrennungsmotor [$n_{min}$ $n_{max}$]
  - Drehzahl Elektromotor [$-n_{max}$ $n_{max}$]
  - Betriebsstunden oder Lebensdauer [0 ∞) (einseitig)
- Bei Dichteschätzung für eine solche begrenzte Größe, wird typischerweise "Wahrscheinlichkeitsmasse" über die Grenzen in den physikalisch unmöglichen Bereich hinausragen. Dadurch summiert / integriert sich die Dichte in dem Intervall zu einer Gesamtwahrscheinlichkeit < 1. D.h. es kommt zu einer systematischen **Unterschätzung** der Dichte.

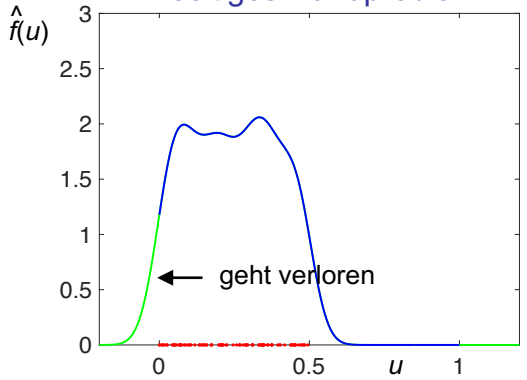
## Lösung

- Dieser systematische Fehler muss korrigiert werden, indem die verlorenen Dichteanteile außerhalb des Intervalls auf die Dichte in dem Intervall zugeschlagen werden. Es gibt verschiedene Verfahren eine solche **Randkorrektur** (*boundary correction*) vorzunehmen.

Beidseitiges Randproblem




Einseitiges Randproblem



## Idee zur Randkorrektur

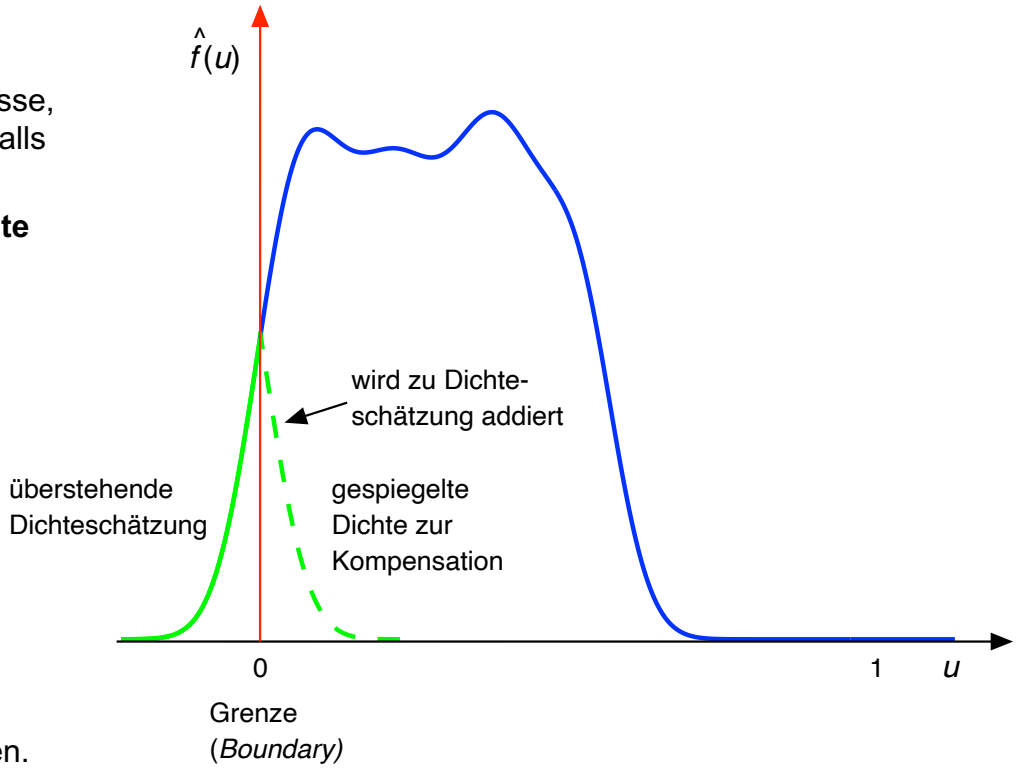
- Es existieren viele Alternativen mit spezifischen Vor- und Nachteilen.
- Grundidee ist es die „verlorene“ Wahrscheinlichkeitsmasse, die über den Rand hinausragt, für das Innere des Intervalls zurückzugewinnen.
- Dies stellt sicher, dass die **integrierte geschätzte Dichte** über das physikalisch sinnvolle Intervall **= 1** ist.

$$\int_{\min}^{\max} \hat{f}(u)du = 1$$

## Methoden zur Randkorrektur

- **Spiegelung**: Siehe rechts.
- log: Konvertiert die begrenzten Daten durch log in einen unbegrenzten Zahlenbereich, macht dort die „normale“ Dichteaschätzung und transformiert zurück.
- **Matlab** bietet ‚`reflection`‘ und ‚`log`‘ als Optionen.
- Es gibt es Vielzahl weiterer, weniger populärer Methoden.

Methode der Spiegelung (*Reflection*)

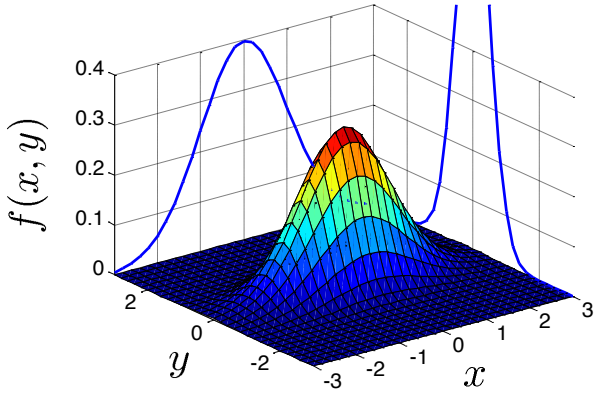


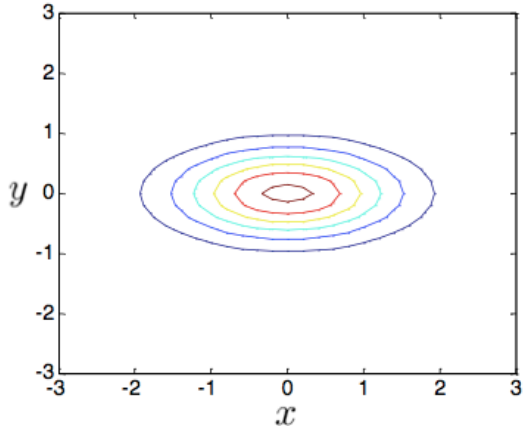
## Ergänzungen

- Im multivariaten Fall (mehrere Dimensionen) kann die Randkorrektor **in jeder Dimension einzeln** durchgeführt werden. Die Kernel-Funktion ergeben sich dann aus Multiplikation der 1-dimensionalen Kernel-Funktionen. Beispiel für Gauß-Kernel mit 2 Dimensionen *x* und *y*:

$$f(x)f(y) = \frac{1}{\sqrt{2\pi}\sigma_x} \mathrm{e}^{-\frac{1}{2}\left(\frac{x-\mu_x}{\sigma_x}\right)^2} \frac{1}{\sqrt{2\pi}\sigma_y} \mathrm{e}^{-\frac{1}{2}\left(\frac{y-\mu_y}{\sigma_y}\right)^2} = \frac{1}{2\pi\sigma_x\sigma_y} \mathrm{e}^{-\frac{1}{2}\left[\left(\frac{x-\mu_x}{\sigma_x}\right)^2 + \left(\frac{y-\mu_y}{\sigma_y}\right)^2\right]} = f(x,y)$$

***x* und *y* unabhängig:**



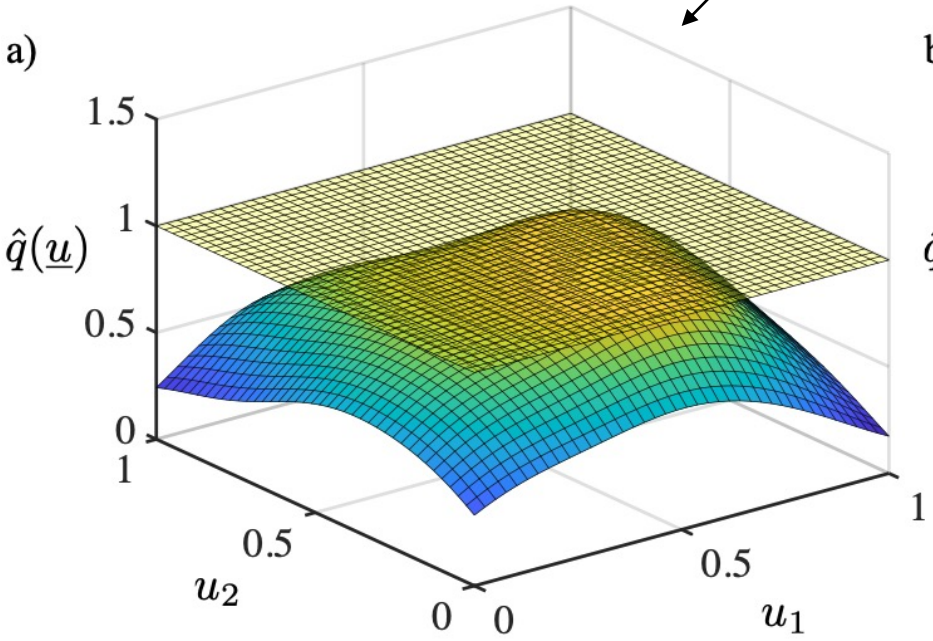
## Randkorrektur in 2D

- Gleichverteilte 30 Datenpunkte gezogen mit `rand`

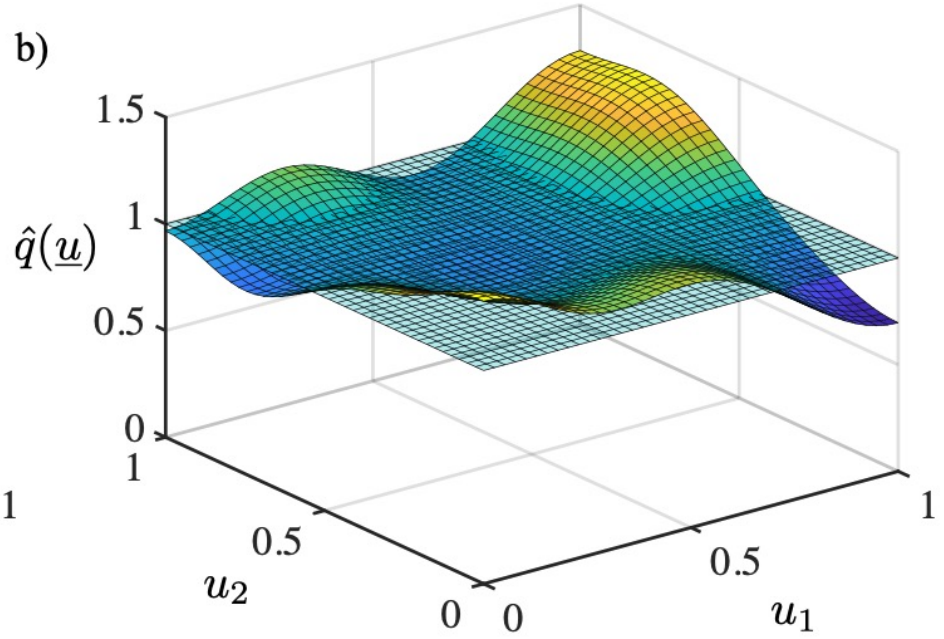
Ohne Randkorrektur entsteht im begrenzten Intervall gar keine gültige Dichte (Integral < 1)!
Durch einen Korrekturfaktor müsste dies behoben werden.

Besser aber Randkorrektur:

Ohne Randkorrektur




Mit Randkorrektur



## Beispiele der nächsten Folien

Mehr über KL-Divergenz in Kapitel 8

- Die gewünschte Verteilung ist eine Gleichverteilung, d.h. die Wunschdichte ist $p(\underline{u}) = 1$.
- Die KL-Divergenz misst, wie gut die Datenpunktverteilung diesen Wunsch erfüllen kann.
- Dazu wird die Wahrscheinlichkeitsverteilungsdichte $\hat{q}(u)$ geschätzt und über die KL-Divergenz mit $p(\underline{u})$ verglichen.

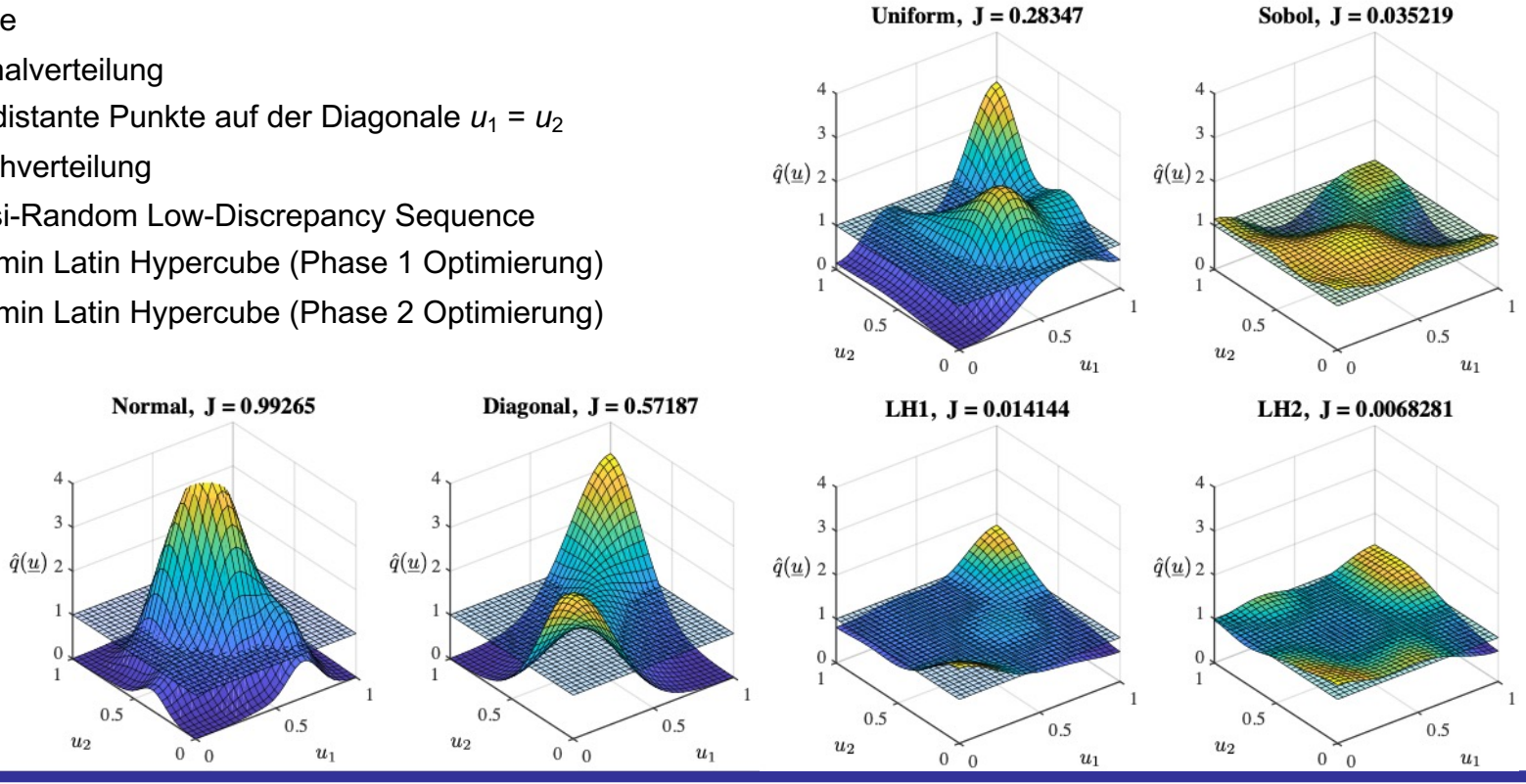
## Welche Verteilung wünscht man sich?

- Wenn kein Vorwissen über die Charakteristik des Prozesses vorliegt, ist beste allgemeingültige Wahl eine **Gleichverteilung** der Daten.
- Anwendungsspezifische, besser geeignete Verteilungen sind evtl. möglich, erfordern aber **Vorwissen** (*prior knowledge*). Dieses Vorwissen könnte sein:
  - wo soll das Modell besonders gut sein oder wo ist die Performance weniger wichtig?
  - in welchen Arbeitsbereichen wird das Modell häufiger genutzt, wo weniger oft?
  - wo verursachen Modellfehler hohe Kosten, wo sind Modellfehler weniger entscheidend?
- Eine Gleichverteilung wird durch `rand` meist unbefriedigend umgesetzt, da sich beim Zufall die Punkte in bestimmten Regionen häufen können (und damit andere Regionen dünn besetzt oder leer bleiben).
- Daher ist es üblich, deterministische, gleichverteilte Daten durch sog. **raumfüllende Sequenzen** (***space-filling sequences***) nach bestimmten Prinzipien zu erzeugen. Dies stellt i.A. sicher, dass nicht zufällig schlechte Datenverteilungen entstehen können.

## Beispiele geschätzter Wahrscheinlichkeitsdichten für verschiedene Datenverteilungen

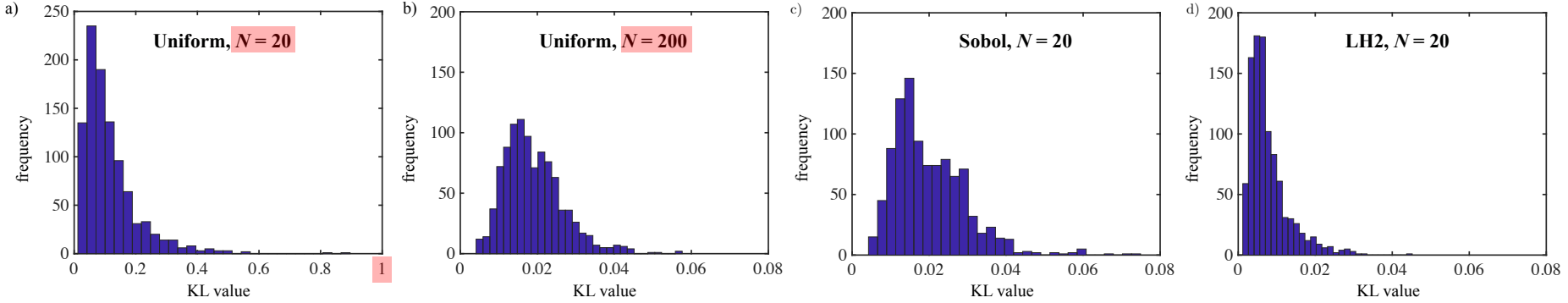
- $N$ = 20 Datenpunkte
- Normal = Normalverteilung
- Diagonal = Äquidistante Punkte auf der Diagonale $u_1 = u_2$
- Uniform = Gleichverteilung
- Sobol = Quasi-Random Low-Discrepancy Sequence
- LH1 = Maximin Latin Hypercube (Phase 1 Optimierung)
- LH2 = Maximin Latin Hypercube (Phase 2 Optimierung)

- $J$ = KL-Güte (kleiner ist besser)
- $\hat{q}$ = geschätzte Dichte

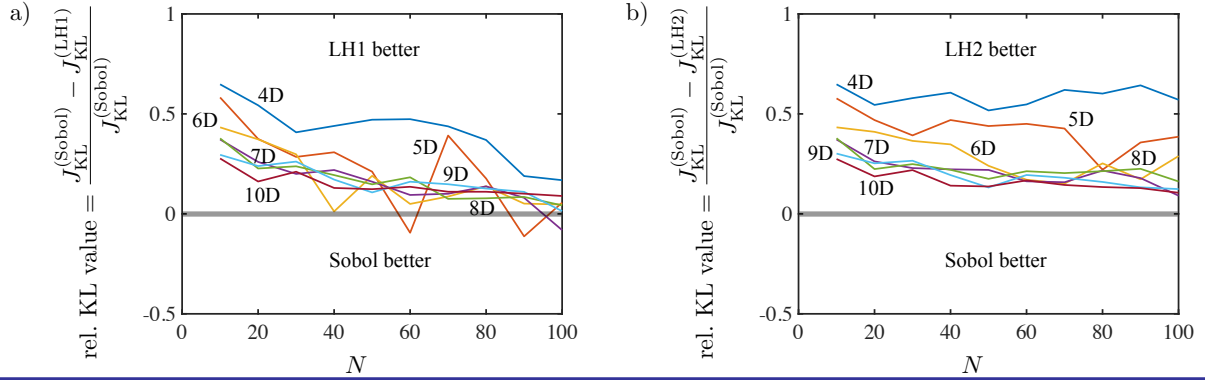


# 7.6 Kullback-Leibler-Divergenz

## Kullback-Leibler-Divergenz für 1000 Realisierungen (Histogramm)



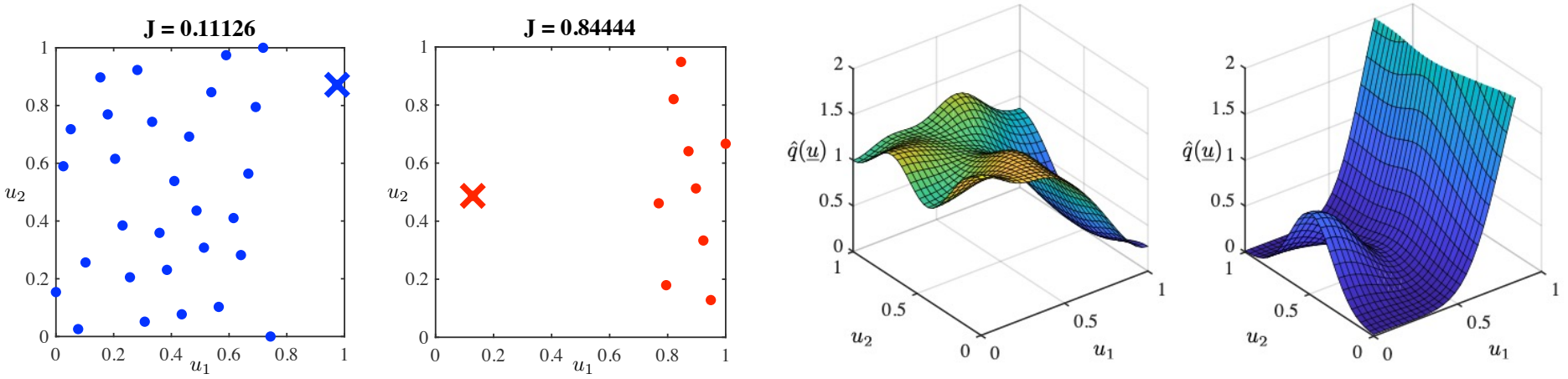

## Raumfüllende Eigenschaften von Maximin Latin Hypercubes vs. Sobol Sequenzen




Achtung! Verschiedene Datenmengen und Achsenskalen beachten!

## Erzeugung von Gleichverteilung mittels Punkttausch zw. schlechten Trainings- und Validierungsdaten

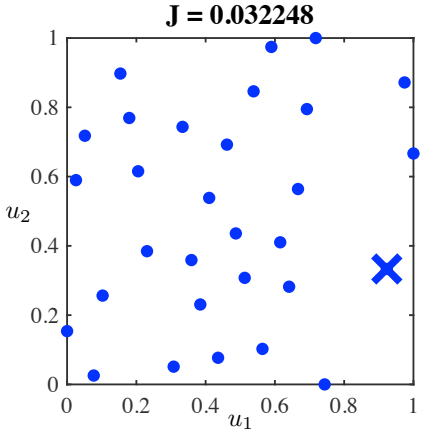
### Iteration 1

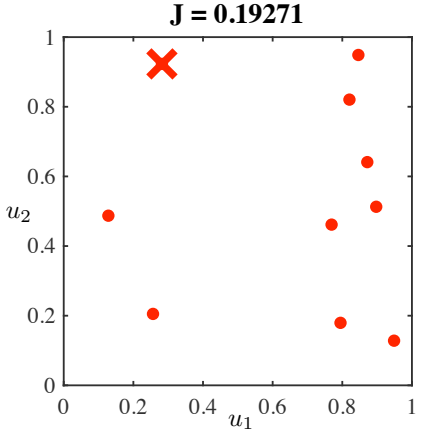


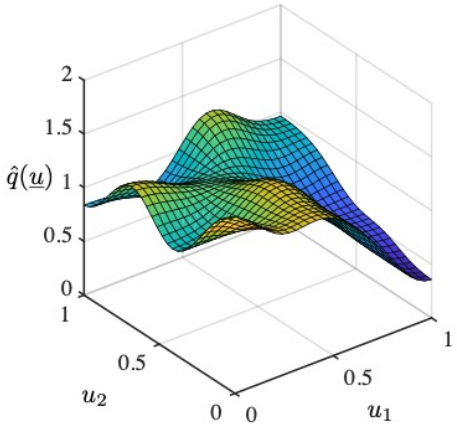

- Extrembeispiel von schlecht verteilten blauen Trainingsdaten (alle Punkte links) und roten Validierungsdaten (alle Punkte rechts).
- Optimierungsalgorithmus tauscht Punkte beider Datensätze, um die KL-Divergenz beider geschätzten Dichten zu verbessern.

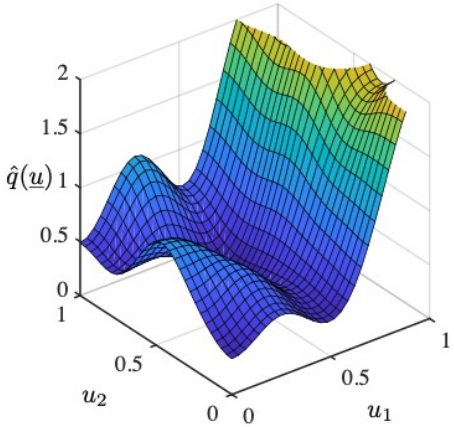
## Erzeugung von Gleichverteilung mittels Punkttausch zw. schlechten Trainings- und Validierungsdaten

### Iteration 3



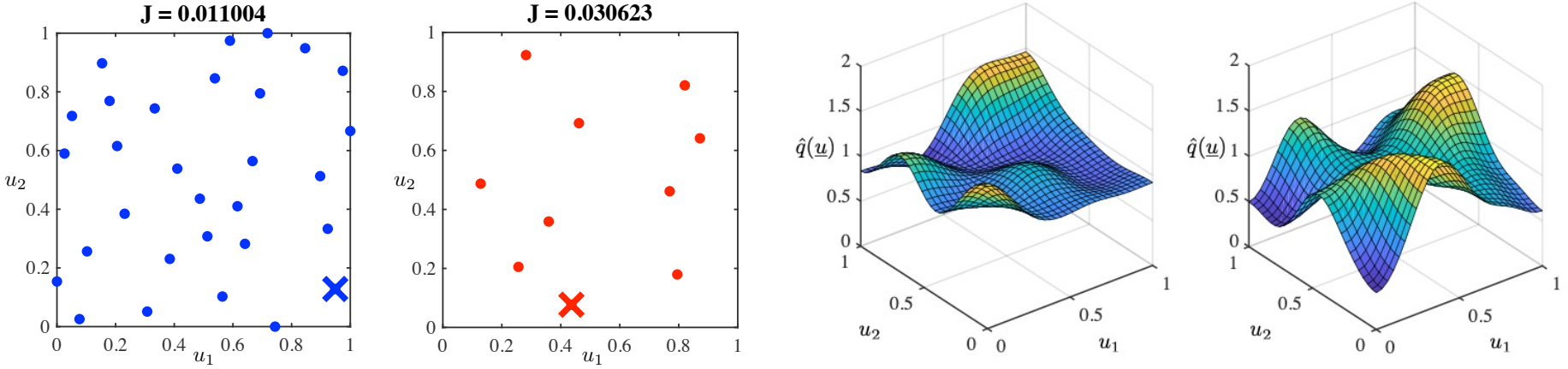



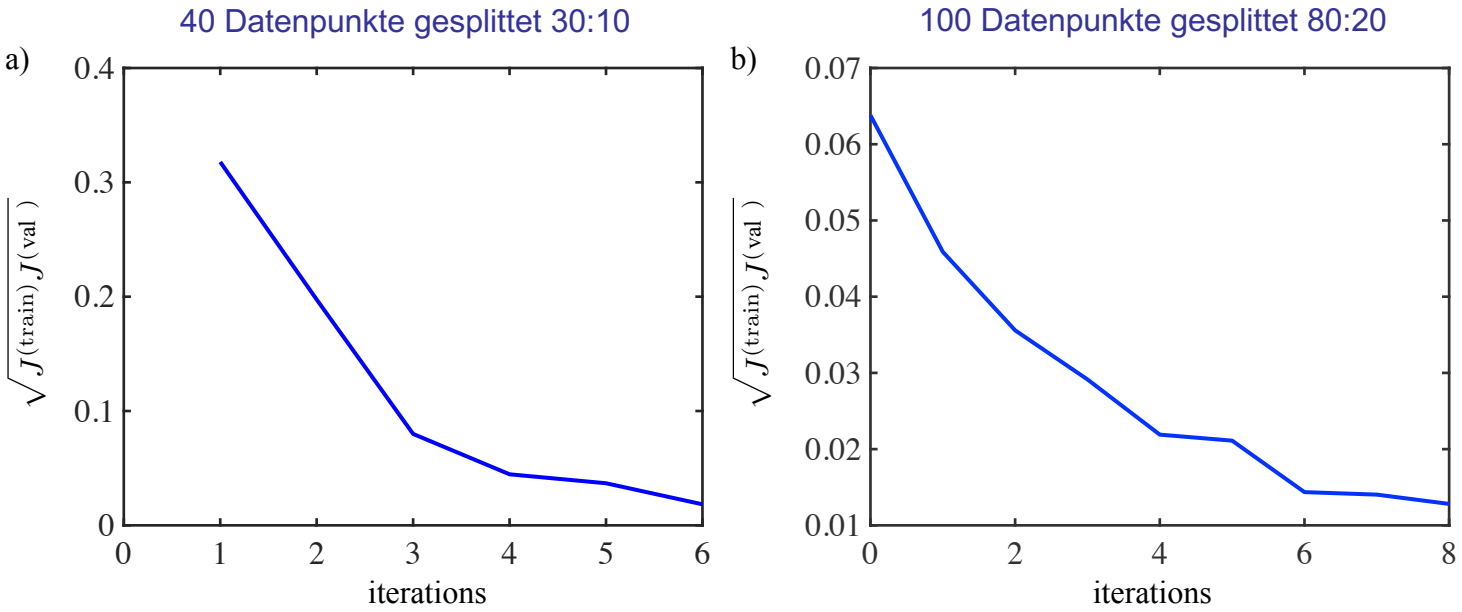
## Erzeugung von Gleichverteilung mittels Punkttausch zw. schlechten Trainings- und Validierungsdaten

### Iteration 6



## Konvergenz der Tauschoptimierung

- Teil-Verlustfunktionen = KL-Divergenzen der geschätzten Dichten für Trainings- und Validierungsdaten.
- Aggregierte Verlustfunktion ist das **geometrische Mittel** der Verlustfunktionen auf Trainingsdaten und Validierungsdaten.



# 8. Verteilung der Datenpunkte

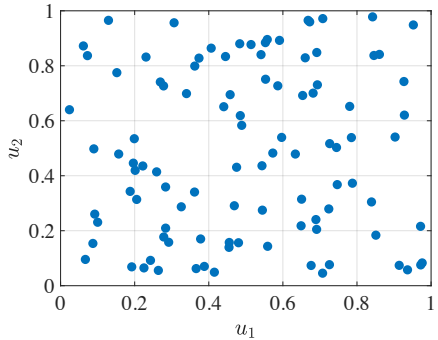
# 8. Verteilung der Datenpunkte

# 8.1 Bedeutung der Datenpunktverteilung

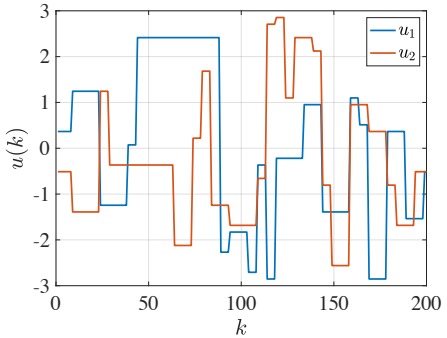
## Motivation

- Die **Qualität** von datenbasierenden Modellen hängt stark von der Verteilung der genutzten Trainingsdaten ab.
- In der Anwendung können **zwei verschiedene Fälle** unterschieden werden:
  - Datensatz ist noch **nicht vorhanden** → 1. Design of Experiments (DoE): Gewünschte Lage der Datenpunkte 2. Messung
  - Datensatz ist **vorhanden** → Daten können direkt für das Training eines Modells verwendet werden
- Hierbei müssen der dynamische und statische Fall getrennt voneinander betrachtet werden.
  → Reihenfolge der Datenpunkte im dynamischen Fall relevant, im statischen nicht.
- Allgemein: Wenn **kein Vorwissen** über den betrachteten Prozess vorhanden ist, sollten bei statischen und dynamischen Prozessen möglichst **gleichverteilte Daten** verwendet werden.

A) Eingangsraum, statisch




B) Eingangssignale, dynamisch

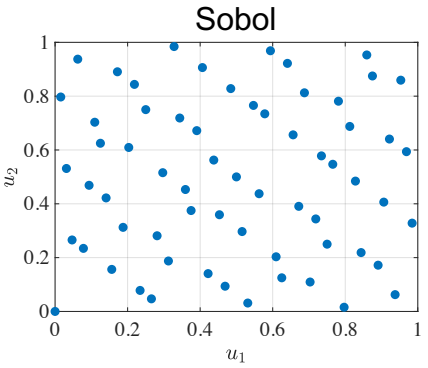


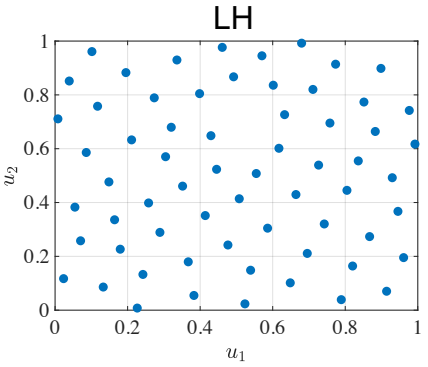
# 8.1 Bedeutung der Datenpunktverteilung

## A) Statischer Fall: Sobol Sequenzen vs. Latin Hypercube Designs

- Im statischen Fall kann man auf 2 unterschiedliche Arten gleichverteilte Datenpunkte erzeugen:
  - stochastisch bzw. zufällig (in MATLAB mit `rand`)
  - deterministisch (quasi-stochastisch)
- Es gibt es **zwei dominierende Verfahren**, um deterministisch gleichverteilte Daten zu erstellen:

1. **Sobol**-Sequenzen
   - quasi-zufällige (keine wirkliche Zufälligkeit) Sequenzen mit geringer Diskrepanz (gleichverteilt)
   - sehr geringer Rechenaufwand für beliebig große Anzahl an Datenpunkten und Dimensionen
   - deutlich verbesserte Raumfüllung im Vergleich zu zufälligen Daten
2. Optimierte **Latin Hypercube (LH)** Designs
   - jeder Eingang / Dimension wird in *N* Levels (meist äquidistant) diskretisiert
   - ein *N* x *N* x ... x *N*-Gitter entsteht, wo jede Gitterline genau einmal mit einem Punkt besetzt ist, d.h. es gibt insgesamt *N* Datenpunkte
   - im Gegensatz zu Sobol-Sequenzen nicht-kollabierendes Design: Punkte fallen bei 1D-Projektion (auf eine Achse) nicht zusammen, d.h. sind unterschiedlich
   - ohne Optimierung typischerweise schlechter raumfüllend als Sobol-Sequenzen
   - durch Optimierung, z.B. maximin: Bessere Raumfüllung als Sobol-Sequenzen möglich



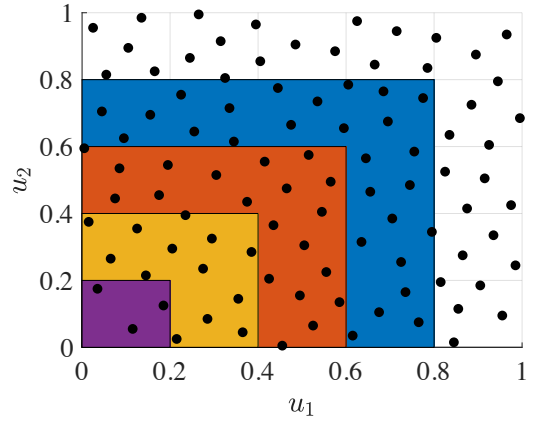



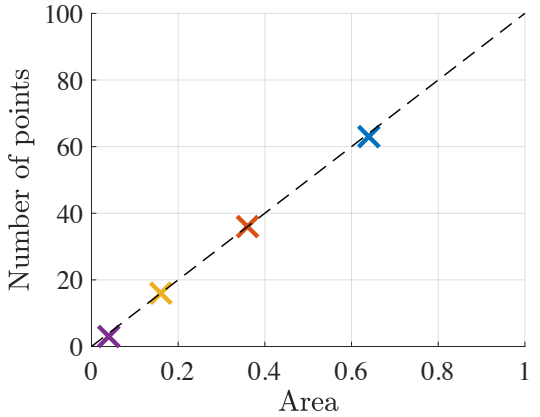
# 8.1 Bedeutung der Datenpunktverteilung

## A) Quasi-zufällige Sequenzen mit niedriger Diskrepanz

- Sequenzen mit niedriger Diskrepanz erfüllen die Eigenschaft, dass **jedes Subset** (Teil-Sequenz) dieser Sequenz eine **niedrige Diskrepanz** besitzt.
- Hier: Subset liegt in einer bestimmten geometrischen **Form/Volumen**, d.h. man betrachtet die Punkte, die in einen (Hyper)kubus, (Hyper)sphäre, etc. fallen.
- **Diskrepanz**: Variation des Verhältnisses zwischen der **Anzahl der Punkte**, die in ein beliebiges Subset fallen und dem **Maß** der geometrische **Form/Volumen.**
- **Niedrige Diskrepanz** = Wenig schwankende **Punktdichte** überall
  → Gleichmäßig raumfüllende Sequenzen.
- Sequenzen mit niedriger Diskrepanz werden ebenfalls **quasi-zufällig** (*quasi-random*) genannt, da sie häufig als Ersatz von gleichverteilten Zufallszahlen verwendet werden.
- Das „**quasi**“ soll hervorheben, dass zwar Eigenschaften mit echtem Zufall geteilt werden, es sich aber nicht um echten Zufall handelt (deterministisch).

**Beispiel rechts:**
100 gleichverteilte Datenpunkte im Raum $[0, 1]^2$.
Der Flächeninhalt (≙ Maß im 2D-Fall) von 4 Subsets wurde ausgewertet (Quadrate lila, gelb, rot, blau) und ein proportionaler Zusammenhang zwischen Fläche und Anzahl der Datenpunkte festgestellt.



# 8.1 Bedeutung der Datenpunktverteilung

## A) Gewünschte Eigenschaften von Latin Hypercube Designs

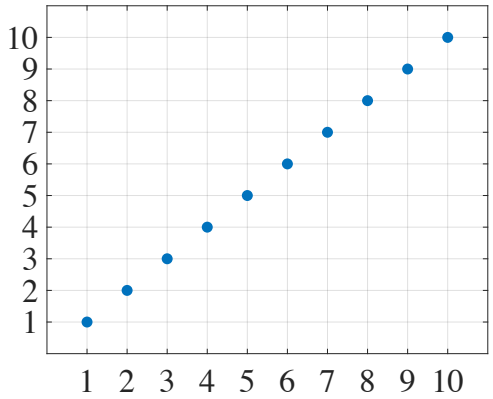
1. Raumfüllung

- Das Design sollte den gesamten Eingangsraum gleichmäßig abdecken
  → LHDs erfüllen diese Eigenschaft erst nach einer **Optimierung**

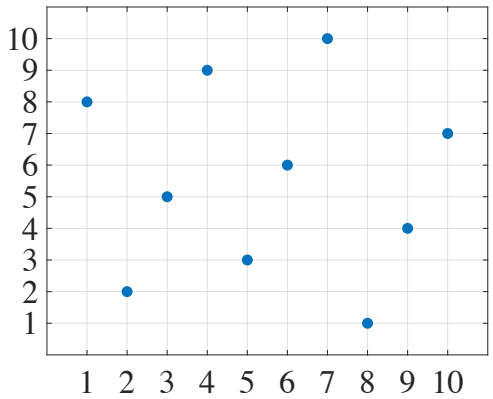
2. Nicht-kollabierend

- Alle Projektionen der Daten auf eine Achse sollen zu unterschiedlichen Werten auf dieser Achse führen.
  → Strukturbedingte Eigenschaft von LHDs

**Latin Hypercube Design 1**




**Latin Hypercube Design 2**

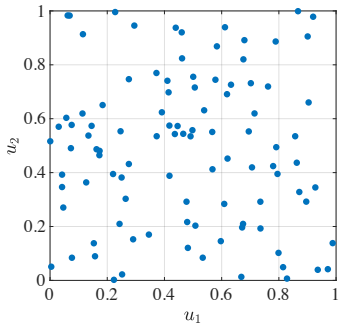


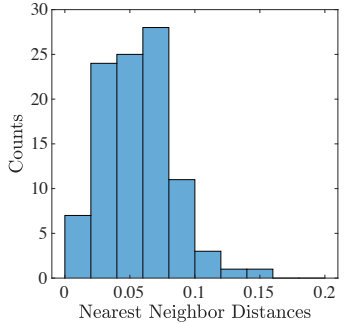

| | LHD 1 | LHD 2 |
|---|---|---|
| **Nicht-Kollabierend** | Ja | Ja |
| **Raumfüllung** | Nein | Ja |

# 8.1 Bedeutung der Datenpunktverteilung

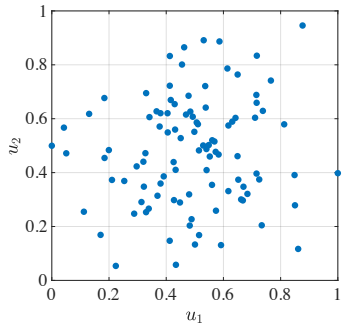
## A) Verschiedene Eingangsräume im statischen Fall

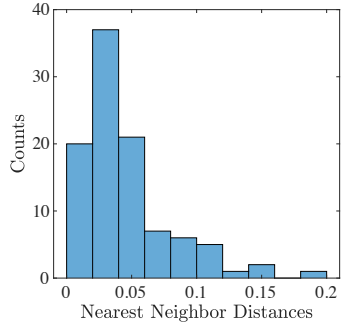
Zufällig gleichverteilt



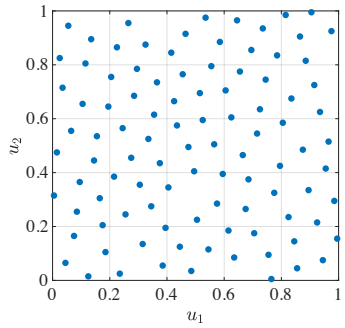



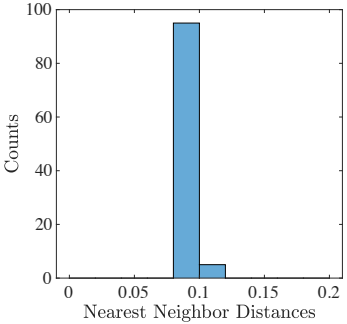

Normalverteilt



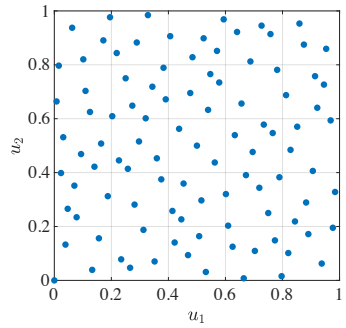



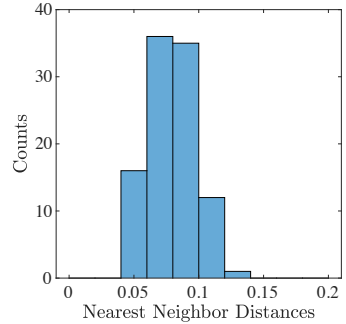

Optimierter LH



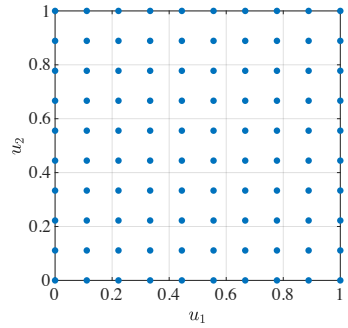



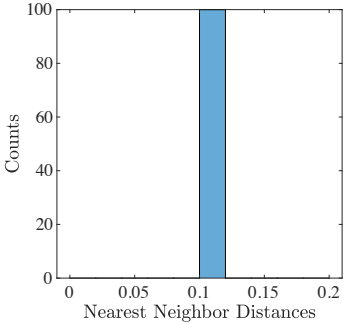

Sobol Sequenz




Gitter

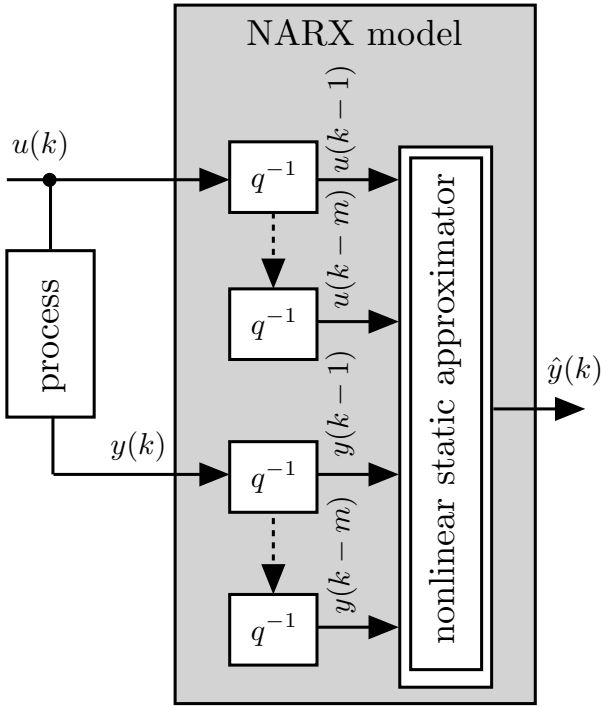


## B) Anregungssignale für nichtlineare dynamische Modelle

$q^{-1}$: Verzögerungsoperator
$m$: dynamische Modellordnung

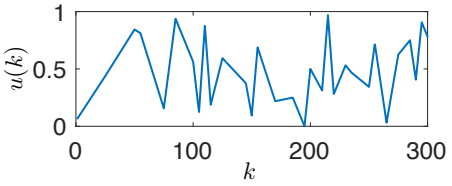
- Im dynamischen Fall muss man unterschieden:
  - **Anregungssignal** $u(k)$ (SISO) bzw. $u_i(k)$, $i = 1, 2, ..., p$ (MISO)
  - **Regressorraum** [ $u(k\text{-}1)$ ... $u(k\text{-}m)$ $y(k\text{-}1)$ ... $y(k\text{-}m)$ ], d.h. Eingangsraum des Modells. Je nach Dynamikrealisierung des Modells ergeben sich unterschiedliche Regressorräume.
- Diverse **populäre** Anregungssignale existieren für dynamische Modelle
  - (A)PRBS, Multisinus, Chirp, ...
- Das Anregungssignal bestimmt im nichtlinearen Fall die Datenverteilung für den **Eingangsraum des nichtlinearen Approximators**.
- **Gleichverteilte** Daten sind zu bevorzugen, besonders wenn kein/wenig Vorwissen über den zu untersuchenden Prozess vorhanden ist. Das ist aber selbst annähernd **sehr schwierig** zu erreichen.
- **Anforderungen** an Anregungssignale:
  - Abdeckung (*coverage*) des **Amplitudenbereichs**
  - Abdeckung (*coverage*) des **Frequenzspektrums**
  - Berücksichtigung der Betriebsbedingungen (z.B. Rate Constraints, Bedingungen für einen sicheren Betrieb).

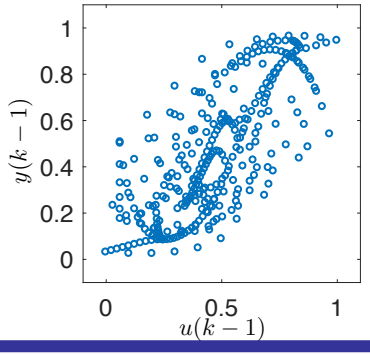


# 8.1 Bedeutung der Datenpunktverteilung

## Rampe

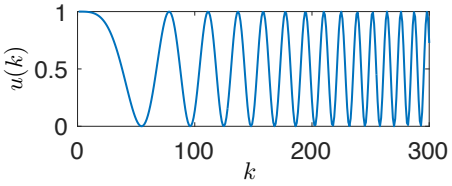
- Signal besitzt größtenteils niedrige Frequenzen
- Datenpunkte konzentriert auf statischem Equilibrium

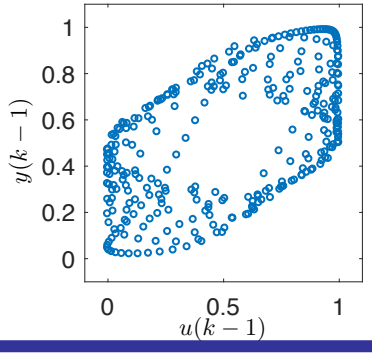



## Chirp

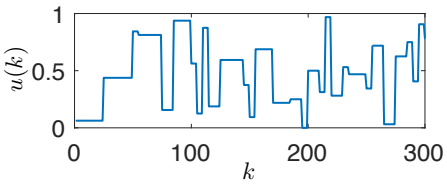
- Niedrig- bis hochfrequentes Signal
- Datenpunkte fehlen im Equilibrium und an den Rändern des Eingangsraums

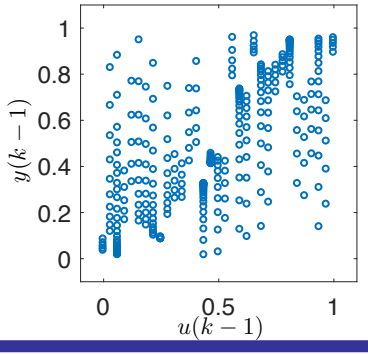



## APRBS

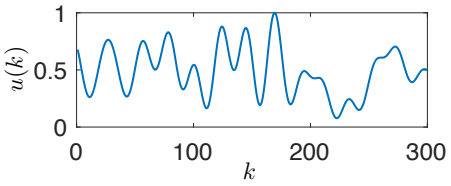
- Sprünge: hochfrequent, konstante Stellen: niedrigfrequent
- Datenpunkte fehlen in „Spalten“ des Eingangsraums

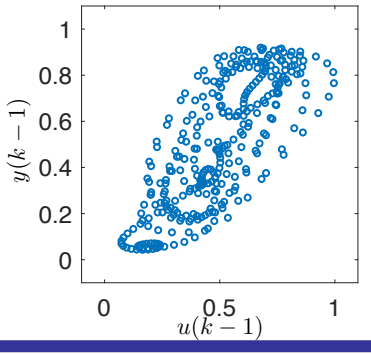



## Multisinus

- Niedrig- bis hochfrequentes Signal
- Datenpunkte fehlen an den Rändern des Eingangsraums



## Grundlagen der Informationstheorie

- Geht auf **Claude Shannon** zurück
- Beschäftigt sich mit Informationsgehalt und Entropie in der Informationsübertragung, Datenkompression, etc.
- Informationsgehalt:

$$I(p_i) = \log_2(1/p_i) = -\log_2(p_i)$$

  - Jedes Zeichen *i* besitzt eine **Auftrittswahrscheinlichkeit $p_i$**
  - Eine zu versendende **Nachricht** besteht aus einer Kette von **Zeichen** eines **Alphabets**
  - **Alphabet**: Z.B. das deutsche Alphabet; **Zeichen**: Buchstaben des Alphabets; **Nachricht**: Ein Wort/Satz/Text
- Informationsgehalt kann allgemein als **Überraschungswert** verstanden werden. Der Informationsgehalt eines Zeichens ist seine **statistische Signifikanz**. Er bezeichnet also die minimale Anzahl von **Bits**, die benötigt werden, um ein Zeichen (also eine Informationseinheit) darzustellen oder zu übertragen. [https://de.wikipedia.org/wiki/Informationsgehalt]
- In der Informationstheorie ist die **Entropie** definiert als der Erwartungswert des Informationsgehalts (Summe der mit Wahrscheinlichkeiten gewichteten Informationengehalte):

$$H = -\sum_i p_i \log_2(p_i)$$

  → Ein Maß für den mittleren Informationsgehalt einer Nachricht pro Zeichen

**Beispiel:** Der Buchstabe *Q* hat im deutschen Alphabet eine sehr geringe Auftrittswahrscheinlichkeit → Hoher Informationsgehalt, da er die Menge der möglichen Wörter stark einschränkt

| | Zuf | Deu | Eng | Nie | Spa | Fra | Ita |
|---|---|---|---|---|---|---|---|
| A | 3,85 | 5,45 | 7,19 | 7,17 | 6,69 | 6,82 | 10,73 |
| B | 3,85 | 1,75 | 1,58 | 1,41 | 0,71 | 0,70 | 0,89 |
| C | 3,85 | 3,37 | 4,05 | 1,78 | 3,52 | 3,30 | 5,05 |
| D | 3,85 | 5,11 | 3,11 | 6,85 | 4,03 | 3,71 | 3,57 |
| E | 3,85 | 16,89 | 13,05 | 18,84 | 15,92 | 15,61 | 13,19 |
| F | 3,85 | 1,28 | 2,42 | 0,78 | 1,10 | 1,13 | 1,31 |
| G | 3,85 | 3,76 | 2,34 | 2,94 | 1,57 | 0,84 | 1,05 |
| H | 3,85 | 5,26 | 4,71 | 2,75 | 1,22 | 0,59 | 1,50 |
| I | 3,85 | 8,51 | 7,71 | 6,87 | 7,32 | 7,11 | 9,80 |
| J | 3,85 | 0,18 | 0,09 | 1,50 | 0,16 | 0,19 | 0,01 |
| K | 3,85 | 1,51 | 0,58 | 1,92 | 0,05 | 0,01 | 0,01 |
| L | 3,85 | 3,77 | 3,72 | 4,15 | 5,31 | 4,85 | 5,76 |
| M | 3,85 | 2,22 | 2,54 | 1,88 | 2,56 | 3,22 | 2,98 |
| N | 3,85 | 10,42 | 7,81 | 9,91 | 7,14 | 9,42 | 7,57 |
| O | 3,85 | 3,11 | 7,52 | 5,85 | 6,01 | 6,08 | 9,66 |
| P | 3,85 | 0,63 | 2,30 | 1,36 | 3,53 | 3,21 | 2,63 |
| Q | 3,85 | 0,01 | 0,10 | 0,02 | 1,36 | 1,74 | 0,69 |
| R | 3,85 | 7,14 | 6,41 | 6,50 | 7,03 | 5,81 | 6,09 |
| S | 3,85 | 6,24 | 6,49 | 4,45 | 9,44 | 9,53 | 5,94 |
| T | 3,85 | 6,08 | 9,22 | 6,02 | 7,31 | 7,32 | 5,90 |
| U | 3,85 | 3,40 | 2,83 | 1,77 | 5,72 | 6,92 | 2,95 |
| V | 3,85 | 0,89 | 0,86 | 2,66 | 1,12 | 1,06 | 1,64 |
| W | 3,85 | 1,64 | 1,07 | 1,40 | 0,05 | 0,01 | 0,01 |
| X | 3,85 | 0,02 | 0,45 | 0,02 | 0,71 | 0,42 | 0,02 |
| Y | 3,85 | 0,07 | 1,73 | 0,09 | 0,36 | 0,36 | 0,01 |
| Z | 3,85 | 1,27 | 0,10 | 1,12 | 0,06 | 0,03 | 1,04 |

## Entropie (kontinuierlich)

- Die **Entropie** einer Wahrscheinlichkeitsdichte $p(x)$ ist:

$$H(P) = -\int_{-\infty}^{\infty} p(x) \cdot \log p(x) dx$$

- Die Entropie kann als durchschnittlicher **Informationsgehalt** der Messung einer Zufallsvariable verstanden werden.
- Alternative Interpretationen sind die durchschnittliche Stärke/Menge an **Überraschung** oder **Unsicherheit**.

## Kreuz-Entropie (kontinuierlich)

- Die **Kreuz-Entropie** kann als ein Vorläufer der KL-Divergenz betrachtet werden. Sie misst auch die Ähnlichkeit zweier Wahrscheinlichkeitsdichten:

$$H(P, Q) = -\int_{-\infty}^{\infty} p(x) \cdot \log q(x) dx$$

- Die Kreuz-Entropie hat aber den großen Nachteil, dass $H(P, P) = H(P) \neq 0$. $P$ und $P$ sind sich aber maximal ähnlich, weil identisch; daher ist diese Eigenschaft sehr wünschenswert. Wenn man aber die **Entropie** von $P$ von der **Kreuz-Entropie** von $P$ und $Q$ abzieht, erhält man die **KL-Divergenz** mit genau dieser Eigenschaft:

$$D_{\mathrm{KL}}(P, Q) = H(P, Q) - H(P)$$

## Kullback-Leibler (KL) Divergenz

- Wie kann man die „Übereinstimmung“ oder den „Unterschied“ zweier Wahrscheinlichkeitsdichten beurteilen?
- Eine sehr weit verbreitetes Maß ist die **Kullback-Leibler-Divergenz** (auch relative Entropie genannt):

Kontinuierliche Zufallsvariablen: $D_{\mathrm{KL}} = \int_{-\infty}^{\infty} p(x) \cdot \log \frac{p(x)}{q(x)} dx$

Diskrete Zufallsvariablen: $D_{\mathrm{KL}} = \sum_{x \in \mathcal{X}} P(x) \cdot \frac{P(x)}{Q(x)}$

- Hierzu wird angenommen:
  - $p(x)$: Wahre Wahrscheinlichkeitsdichte
  - $q(x)$: Approximation der Wahrscheinlichkeitsdichte

  Wie gut beschreibt eine Approximation $q(x)$ die Realität $p(x)$?

$P$, $Q$: Wahrscheinlichkeitsverteilungen
$p$, $q$: Wahrscheinlichkeitsverteilungsdichten

- Die KL-Divergenz wird auch als **Informationsgewinn** verstanden, den man erzielt, wenn statt der Approximation $q(x)$ das wahre $p(x)$ verwendet würde.
- Wird manchmal als $D_{\mathrm{KL}}(P, Q)$, typischerweise aber als $D_{\mathrm{KL}}(P \mid\mid Q)$ geschrieben.
- Wenn $q(x) \rightarrow p(x)$, dann $D_{\mathrm{KL}}(p,q) \rightarrow 0$
- **Abstände** oder **Metriken** sind **symmetrisch**. Z.B. ist der Abstand von Punkt A zu Punkt B gleich groß, wie der Abstand von Punkt B zu Punkt A, also formelmäßig: $d$(A,B) = $d$(B,A) für eine „normale“ Definition der Abstandsfunktion $d$.
- Die KL-Divergenz ist **nicht symmetrisch**! D.h. $D_{\mathrm{KL}}(p,q) \neq D_{\mathrm{KL}}(q,p)$. Daher ist sie auch formal kein Abstand.

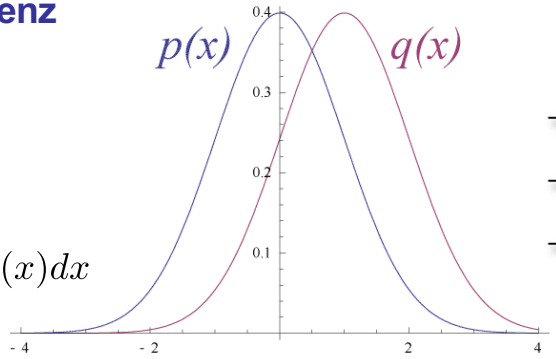
## Beispiele: Kullback-Leibler (KL) Divergenz

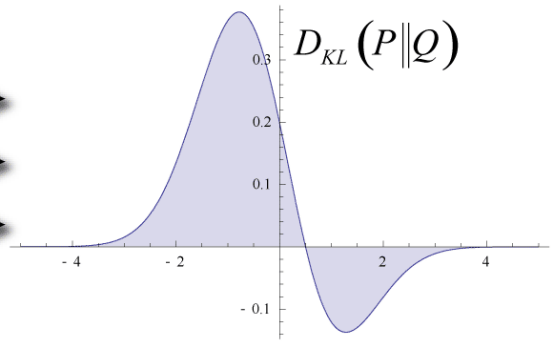
- Beachte:

$$D_{\mathrm{KL}} = \int_{-\infty}^{\infty} p(x) \cdot \log \frac{p(x)}{q(x)} dx =$$

$$\int_{-\infty}^{\infty} p(x) \cdot \log p(x) dx - \int_{-\infty}^{\infty} p(x) \cdot \log q(x) dx$$

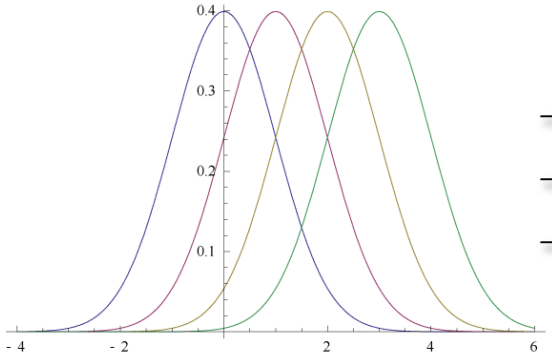
- D.h. die KL-Divergenz entspricht der Differenz der Logarithmen der Dichten, gewichtet mit der wahren Dichte *p*(*x*), integriert über alle Ereignisse *x*.
- Diese Gewichtung betont die häufigen/ wahrscheinlichen Ereignisse während die seltenen/unwahrscheinlichen kaum relevant sind.
- Durch diese Gewichtung mit *p*(*x*) (und nicht mit *q*(*x*)) entsteht die Asymmetrie zwischen *p*(*x*) und *q*(*x*).

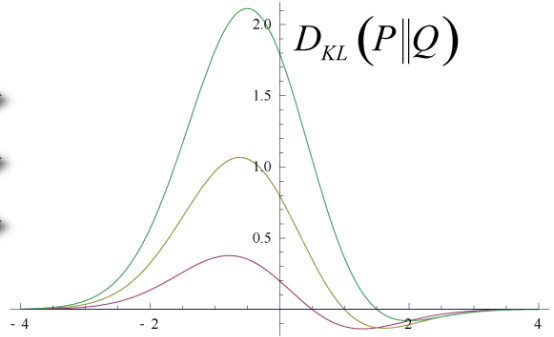



Original Gaussian PDF's

KL Area to be Integrated




Quelle: `https://en.wikipedia.org/wiki/Kullback-Leibler_divergence`

## Bias-Varianz Dilemma

→ Abschnitt 1.4

## Aufteilung der Datensätze für Training, Validierung, Test und verschiedene Ansätze zur Validierung

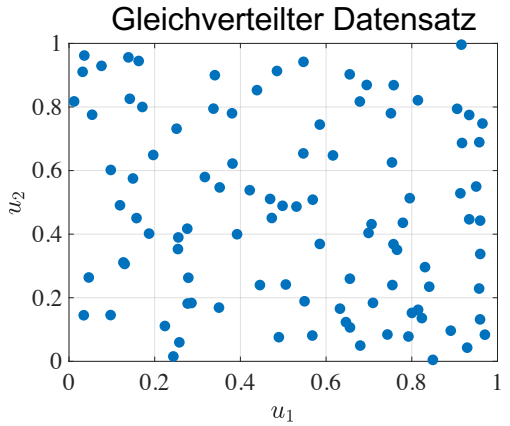
→ Abschnitt 2.6

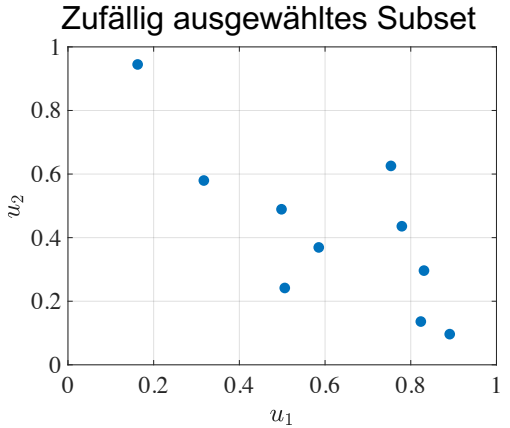
## Datenpunktverteilung der unterschiedlichen Datensätze

- **Trainingsdatensatz** sollte alle Arbeitspunkte und Betriebsmodi enthalten
- **Testdaten**
  - ähnlich verteilt wie Trainingsdaten: Realistische Bewertung der Modellgüte
  - anders verteilt als Trainingsdaten: Testfehler groß! Viel Extrapolation beim Testen!
    Aber keine Aussage zur Modellgüte in Betriebsbereichen der Trainingsdaten
- Was tun, wenn Trainingsdaten anders verteilt sind als bei Modellverwendung?
  - Idee: Teile des Modells (z.B. Struktur) werden aus ursprünglichem Modell (aus Original-Trainingsdaten) übernommen.
    Andere Teile werden an einen neuen Trainingsdatensatz (neue Datenverteilung) angepasst.
  - Methoden: **Transfer Learning** oder **Domain Adaptation**

## Datenpunktverteilung des Trainingsdatensatzes

- Genauigkeit datengetriebener Modelle wird der Datenpunktdichte entsprechen
  - Trainingsdaten dicht → hohe Genauigkeit
  - Trainingsdaten dünn → niedrige Genauigkeit
- Was tun bei ungleicher Punktverteilung?
  - Punktselektion → Abschnitt 8.4
  - Punktgewichtung → Abschnitt 8.5

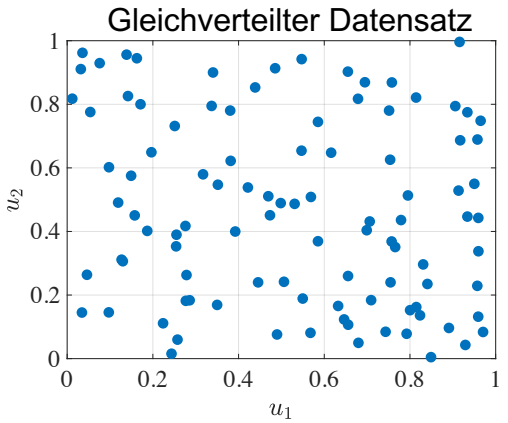


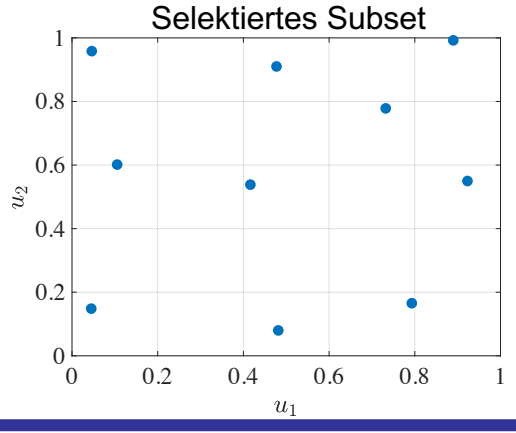



# 8.4 Punktselektion

## Problematik

- Mittels Punktselektion wird aus einem Original-Datensatz ein kleineres Subset an Daten ausgewählt.
- Diese Selektion kann 2 Ziele verfolgen:
  1. **Reduktion** der Datenmenge
  2. Andere, **bessere Verteilung** der Daten des Subsets
- Zu 1.
  - damit Trainingsdaten in den Speicher passen
  - damit sie bei Batch-Verfahren (z.B. Least-Squares) in einem Schritt verarbeitet werden können
- Zu 2.
  - damit alle Arbeitsbereiche im datengetriebenen Modell gleich stark repräsentiert sind
  - damit besonders wichtige Arbeitsbereiche hervorgehoben werden können (bessere Performance auf Kosten der anderen Bereiche)



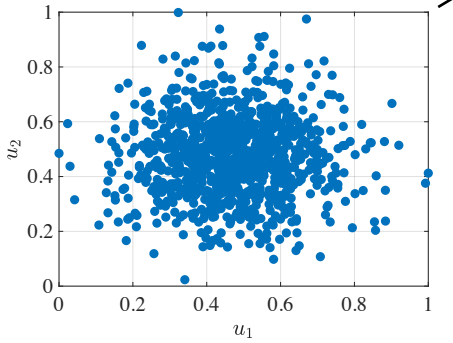



# 8.4 Punktselektion

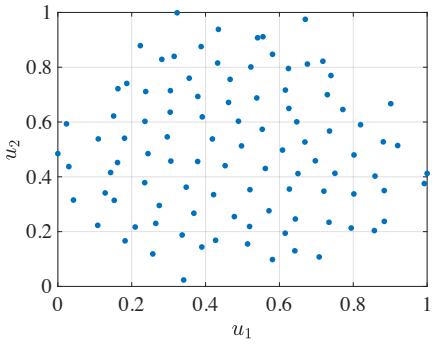
## Punktselektion basierend auf geschätzten Wahrscheinlichkeitsdichten

- In einer aktuellen Veröffentlichung* wird eine Methode vorgestellt, mit der entweder
  1. ein dem originalen Datensatz möglichst ähnliches Subset selektiert werden kann oder
  2. ein Subset selektiert werden kann, welches einer gewünschten Dichte möglichst ähnlich ist
- Besonderheit: Die Auswertung der Dichtewerte geschieht nur an den Datenpunkten des Subsets
  → Deutlich effizienter als die Auswertung mit Monte Carlo Sampling
- Gieriges (*greedy*) Vorgehen: Es werden einem leeren Subset die Datenpunkte aus dem originalen Datensatz hinzugefügt, die den Abstand von zwei geschätzten Wahrscheinlichkeitsdichten minimieren

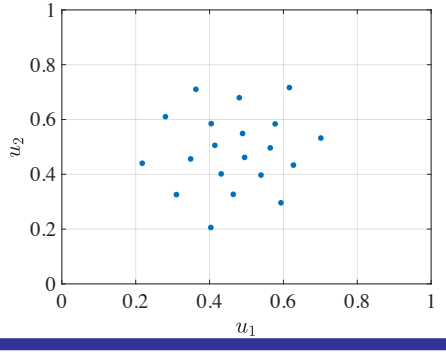
Originaler Datensatz:
Normalverteilt




Selektiert: $N$ = 100 **gleichverteilte** Punkte



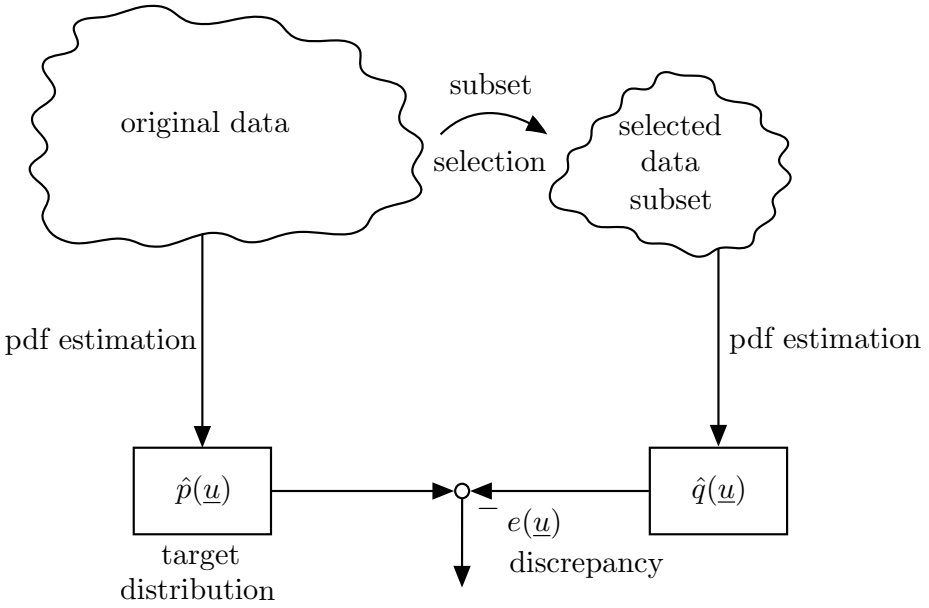

Selektiert: $N$ = 20 **normalverteilte** Punkte




*Quelle: T.J. Peter, O. Nelles: Fast and simple dataset selection for machine learning, at – Automatisierungstechnik 2019; 61(10):833-842.

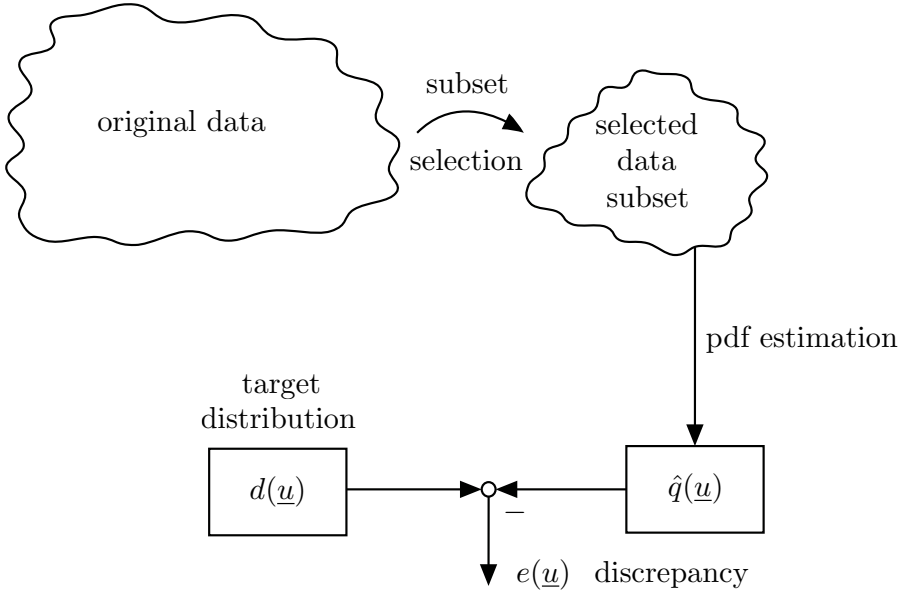
## 1. Reduktion der Datenmenge

- **Ziele des Vorgehens**: Datenmenge reduzieren,
  - damit Trainingsdaten in den Speicher passen,
  - damit sie bei Batch-Verfahren (z.B. Least-Squares) in einem Schritt verarbeitet werden können.
- Einem leeren Subset werden hier nach und nach die Datenpunkte aus dem originalen Datensatz hinzugefügt, die den **Abstand der Wahrscheinlichkeitsdichten** von Subset und originalem Datensatz **minimieren**
- Dabei sind verschiedene Abbruchkriterien denkbar:
  - Das Subset hat enthält eine gewünschte **Anzahl von Datenpunkten**
  - Kriterium auf Basis von anderen Maßen wie z.B. der KL-Divergenz



## 2. Andere, bessere Verteilung der Daten des Subsets

- **Ziele des Vorgehens**: Datenpunkte selektieren,
  - um alle Arbeitsbereiche im datengetriebenen Modell gleich stark zu repräsentieren,
  - um damit besonders wichtige Arbeitsbereiche hervorzuheben (bessere Performance auf Kosten der anderen Bereiche).
- Dem leeren Subset werden Datenpunkte aus dem originalen Datensatz hinzugefügt, die den Abstand der Wahrscheinlichkeitsdichten von Subset und einer **beliebigen, gewünschten Dichte** minimieren
- Sinnvolle Wahl der gewünschten Dichte bei einem sehr ungleichmäßig verteilten originalen Datensatz: **Gleichverteilung**
- Allgemein kann aber **jede beliebige Dichte** als Zielverteilung verwendet werden



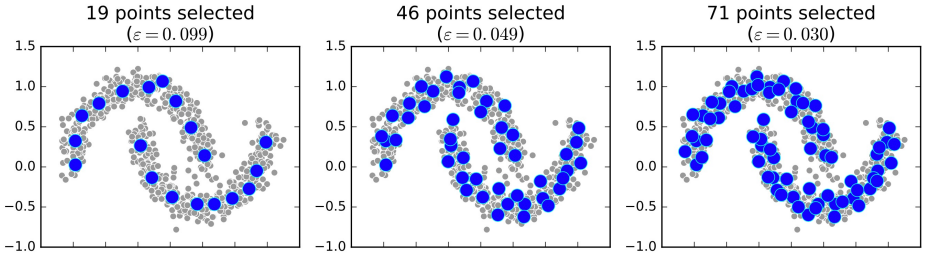
# 8.4 Punktselektion

## Punktselektion basierend auf Informationskriterien

- Alternativer Ansatz*: Selektieren eines Subsets basierend auf **Informationskriterien**
- **Verlustfunktion** basiert auf **zwei Informationskriterien**:

1. **Informationspotential (IP)** des selektierten Subsets wird **minimiert**
   - Subset besitzt dadurch die maximale Entropie
     → Maximiert die Abdeckung des Raums und die Diversität der selektierten Punkte
2. **Kreuzinformationspotential (KIP)** zwischen originalem Datensatz und Subset wird **maximiert**
   - Maximale Ähnlichkeit der pdfs von originalem Datensatz und selektiertem Subset

- Mit der Kombination aus diesen zwei Kriterien wird sichergestellt, dass das Subset sowohl den Raum gut abdeckt als auch dem originalen Datensatz ähnlich ist




Verlustfunktion: $\arg\min_{Z} \left[ \underbrace{\left\| \hat{f}_Z \right\|^2}_{\text{IP}} - 2 \underbrace{\left\langle f_X, \hat{f}_Z \right\rangle}_{\text{KIP}} \right]$

*Quelle: A.R.C. Paiva. Information-theroretic dataset selection for fast kernel learning. International Joint Conference on Neural Networks (IJCNN), Anchorage, USA, 2017.

# 8.5 Punktgewichtung

## Weighted Least Squares (WLS)

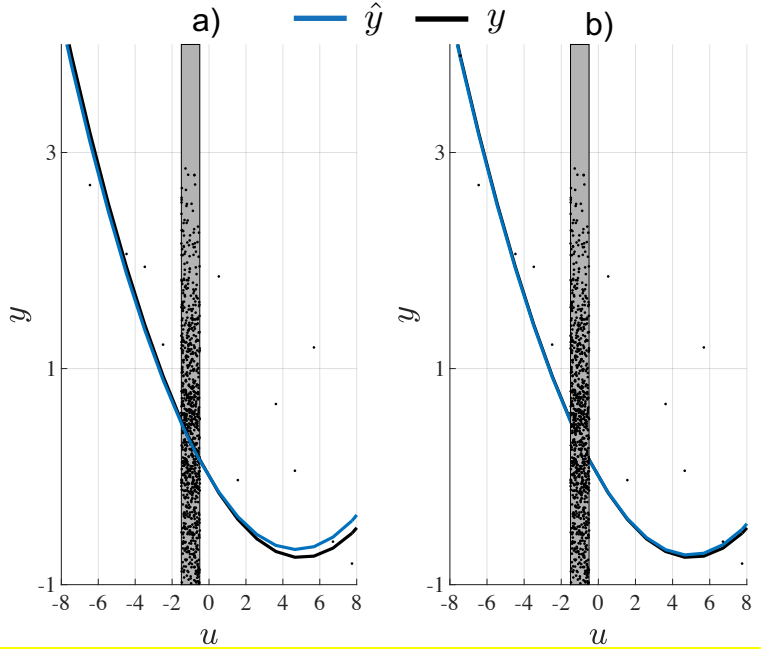
- In der Datenmodellierung sind gewichtete Fehlernormen sehr verbreitet, z.B.:

$$J = \sum_{i=1}^{N} w_i e(i)^2$$

- **Grundidee**: Erweiterung der Verlustfunktion um eine Gewichtungsmatrix $\underline{W}$, mit der jeder Fehlerterm einzeln mit $w_i$ gewichtet werden kann (stehen in der Diagonale von $\underline{W}$):

$$\hat{\underline{\theta}}_{\text{opt}} = \left(\underline{X}^T \underline{W}\, \underline{X}\right)^{-1} \underline{X}^T \underline{W}\, \underline{y}$$

- Durch **ungleichmäßige Datenverteilungen** können Modelle in manchen Bereichen performanter sein als in anderen.
- Falls dieser Einfluss ungewünscht ist, kann dem mit WLS entgegengewirkt werden.
- WLS kann unter anderem genutzt werden für
  - die Reduzierung von Rauschen (z.B. Markow-Schätzung)
  - die Reduzierung des Einflusses von ganzen Datenbereichen,
  - oder die Fokussierung von Datenbereichen.




**Beispiel: Kompensation von schlecht verteilten Daten**
Polynomiales Modell, welches den Großteil der Eingangsdaten im grau markierten Bereich [-0.5,-1.5] vorliegen hat.
a) Schätzung des Modells mit LS
b) Schätzung des Modelles mit WLS

## Least-Squares-Schätzung unter Berücksichtigung der Kovarianz-Matrix des Rauschens

- Ist das Rauschen *n(k)* nicht weiß, sondern korreliert, dann kann man aus der Messung zu einem Zeitpunkt Informationen über das Rauschen zu einem anderen Zeitpunkt gewinnen.
- Diese Informationen stecken in der Kovarianz-Matrix des Rauschprozesses (identisch zur Korrelations-Matrix, da das Rauschen als mittelwertfrei angenommen werden kann).
- Ist diese Kovarianz-Matrix als $\underline{R}_{nn}$ bekannt, so kann diese Information optimal in der LS-Formel verwendet werden.
- Dies führt zur sogenannten **Markow-Schätzung**:

$$\underline{\hat{\theta}}_{\text{opt}} = \left(\underline{X}^T \underline{R}_{nn}^{-1} \underline{X}\right)^{-1} \underline{X}^T \underline{R}_{nn}^{-1} \underline{y} \qquad \text{Also: } \underline{W} = \underline{R}_{nn}^{-1}$$

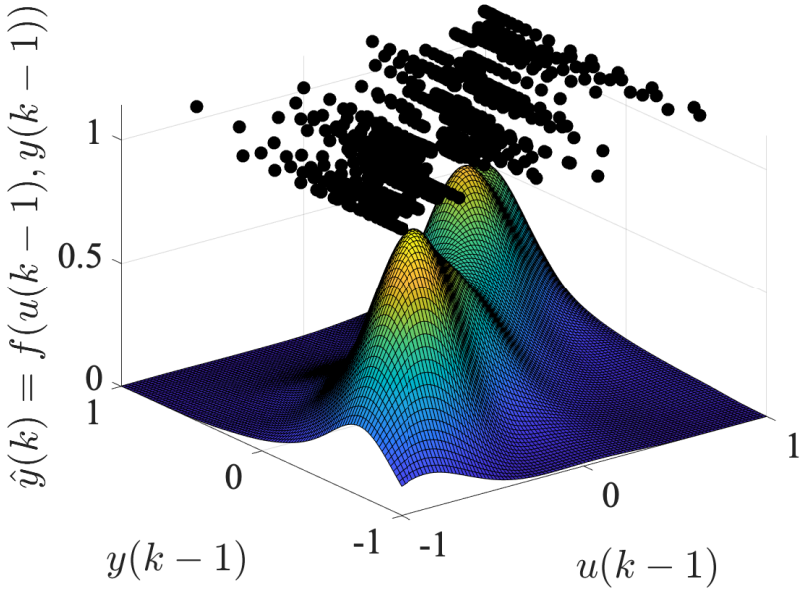
- Durch die Inverse der Kovarianz-Matrix wird das Rauschen dekorreliert und dessen vorhersagbarer Anteil herausgerechnet.
- Ist das Rauschen **statistisch unabhängig** oder gar **weiß**, vereinfacht sich die Kovarianz-Matrix zur Diagonal-Matrix. Dann ist die Markow-Schätzung identisch mit WLS, wobei jeder Messwert mit dem Kehrwert der Rauschvarianz gewichtet wird:

$$\underline{R}_{nn} = \begin{bmatrix} \sigma_n^2(1) & 0 & \dots & 0 \\ 0 & \sigma_n^2(2) & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \sigma_n^2(N) \end{bmatrix} \qquad \underline{R}_{nn}^{-1} = \begin{bmatrix} 1/\sigma_n^2(1) & 0 & \dots & 0 \\ 0 & 1/\sigma_n^2(2) & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1/\sigma_n^2(N) \end{bmatrix}$$

## Anwendung: WLS im Regressorraum

- Im dynamischen Fall: Modell abhängig von zeitverzögerten Ein- und Ausgängen (Regressoren).
- Regressorraum: Abhängig von gewählter Modellstruktur.
  - (N)ARX: $u, y$ oder andere Dynamikrealisierungen
- Je **langsamer** ein Prozess angeregt wird, desto mehr statische Datenpunkte kommen in der Verlustfunktion vor.
  → Zu hohes Gewicht der Statik.
- Idee: Punktgewichtung nutzen, um **unterrepräsentierte** Bereiche **höher** und **überrepräsentierte niedriger** zu gewichten.
- Wie? Gewichtungsmatrix aufbauen mit der Inverse der geschätzten Dichte im Regressorraum.
- Dieses Vorgehen kann auf jedes Identifikationsverfahren angewandt werden, das eine Fehlernorm zur Schätzung der Parameter nutzt.
- Führt zu einer Verbesserung von dynamischen Modellen, welche für den modellierten Prozess zu langsam angeregt wurden.

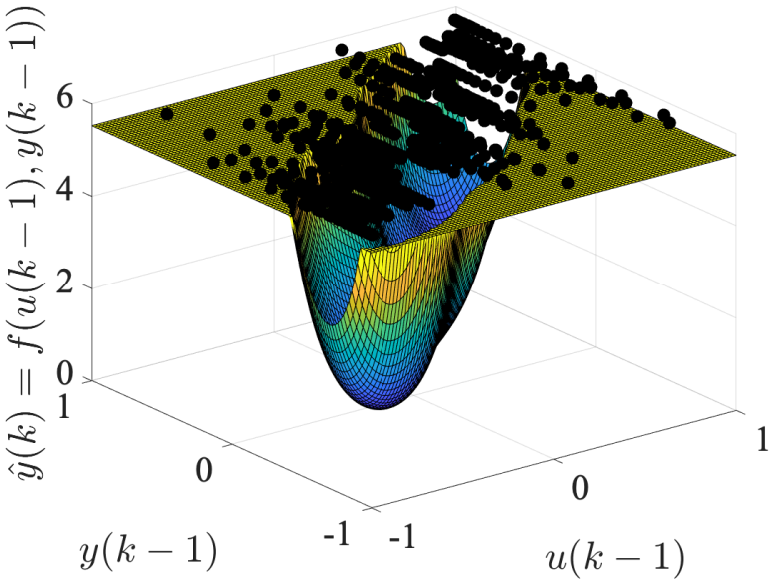
Geschätzte Dichte im Regressorraum



## Anwendung: WLS im Regressorraum

- **Vorgehen:**
  1. Die Dichte für jeden Datenpunkt im Regressorraum wird geschätzt
  2. Die Inverse der Dichte wird berechnet, es kann ein zusätzlicher **Cut-off-Wert $k_d$** eingeführt werden
  3. Die Gewichtungsmatrix $\underline{W}$ wird mit den Werten der Inversen Dichte aufgefüllt
  4. Die Parameter werden mit WLS geschätzt
- Cut-off-Wert: Werte der inversen Dichte werden sehr groß in Bereichen mit wenigen Punkten.
  → Alle Werte größer $k_d$ werden auf $k_d$ gesetzt.
- Der Cut-off-Wert (Maximalwert) soll zu große Differenzen zwischen den Bereichen mit niedriger und hoher Dichte verhindert.
- Guter Erfahrungswert: $k_d = 5$.

Inverse, gedeckelte Dichte im Regressorraum



# 9. Case Study: Fehlerdiagnose Drehgestell

# Automatic system identification for robust fault detection of railway suspensions

Henning JUNG[1], Oliver NELLES[2], Peter KRAEMER[3], Kerstin WEINBERG[4], Geritt KAMPMANN[2], Claus-Peter FRITZEN[1],

[1]Applied Mechanics
Prof. Dr.-Ing. Claus-Peter Fritzen

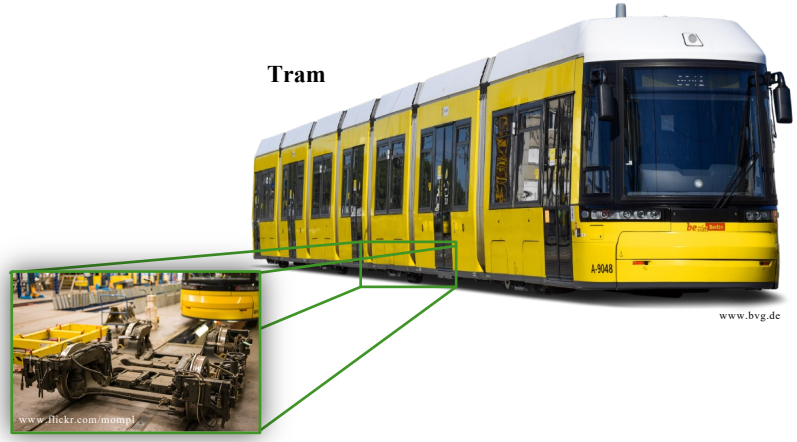
[2]Measurement and Control Engineering
Prof. Dr.-Ing. Oliver Nelles

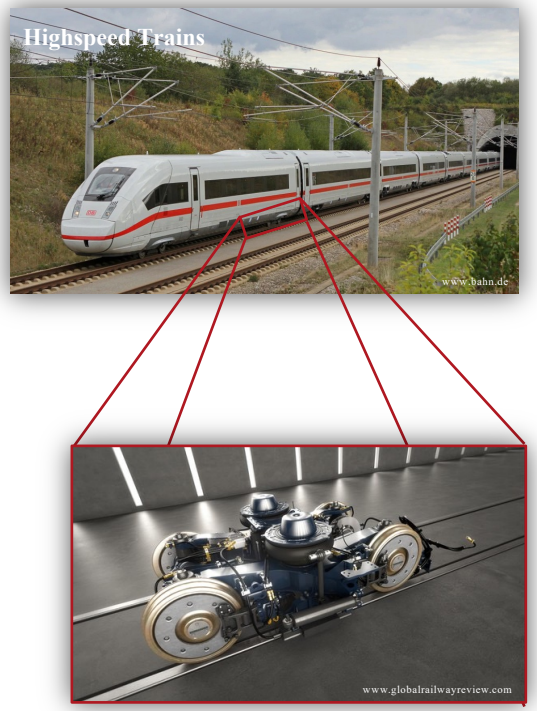
[3]Mechanics, Structural Health Monitoring
Prof. Dr.-Ing. Peter Kraemer

[4]Solid Mechanics
Prof. Dr.-Ing. Kerstin Weinberg

## The Bogie – Core Part of Rail Vehicles

- The bogie's primary function is to carry the structure of the car body and to lead the rail vehicle on the track.
- Failures of bogie components can cause derailments. Broken springs and broken suspensions are major reasons for accidents.
- Therefore the bogie is an absolutely essential element of each rail vehicle and needs to be investigated for a secure rail vehicle concept.

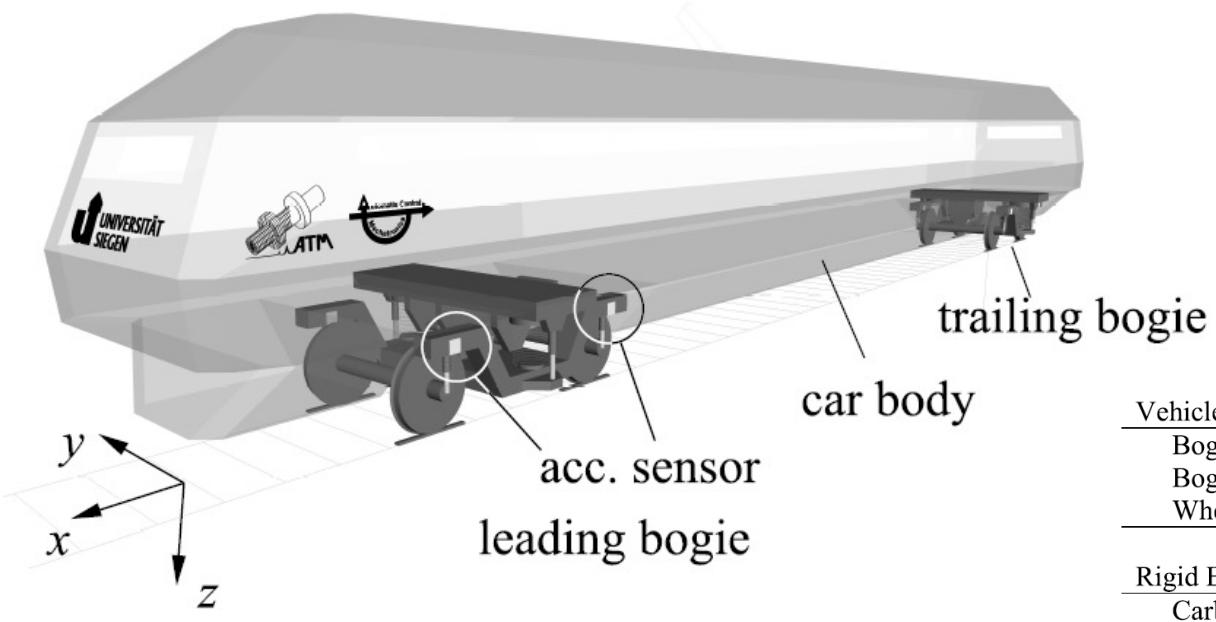


## Outline

## MBS – Model – University Siegen Train

General passenger vehicle based on the Manchester Benchmark[1,2]




| Vehicle Dimensions | |
|---|---|
| Bogie wheelbase | 2.56 m |
| Bogie pivot spacing | 19.00 m |
| Wheel radius | 0.46 m |

| Rigid Body | |
|---|---|
| Carbody | 32 t |
| Bogie | 2.615 t |
| Wheelset | 1.813 t |

[1] S. Iwnick, *The Manchester Benchmarks for Rail Vehicle Simulation*. Sweets & Zeitlinger, 1999.
[2] S. Iwnick, Manchester Benchmark for Rail Vehicle Simulation. *Vehicle System Dynamics*, **30:3-4**, 259-313, 1998.

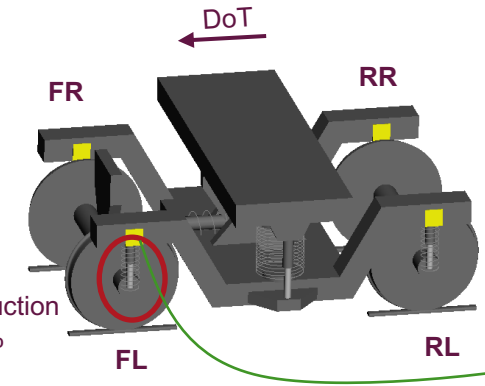
# 9.1 Conventional SSI Procedure for Modal Property Extraction

## Subspace Identification – Data Generation


incl. track irregularities (based on PSDs)

- vertical direction
- lateral direction

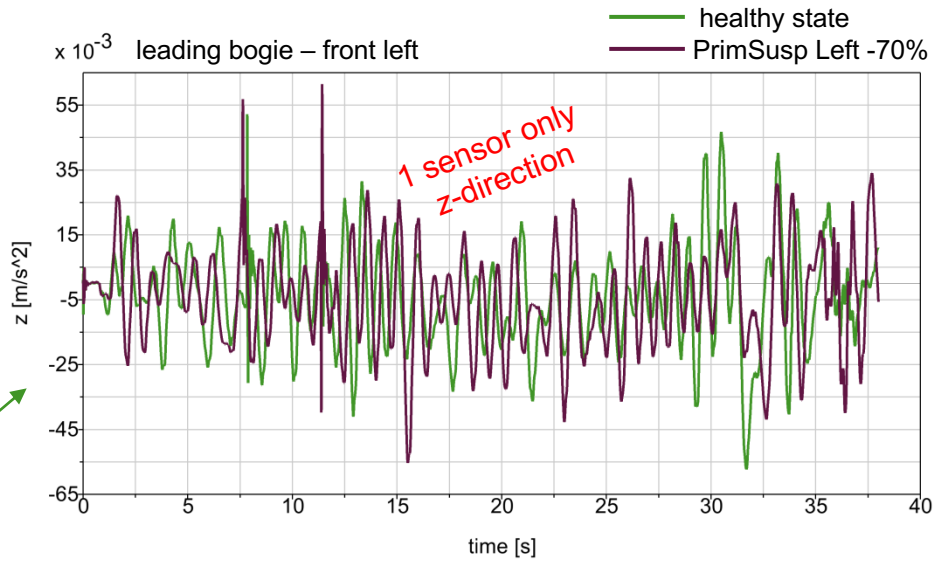
Acceleration Sensor Positions




stiffness reduction by -70%

leading bogie

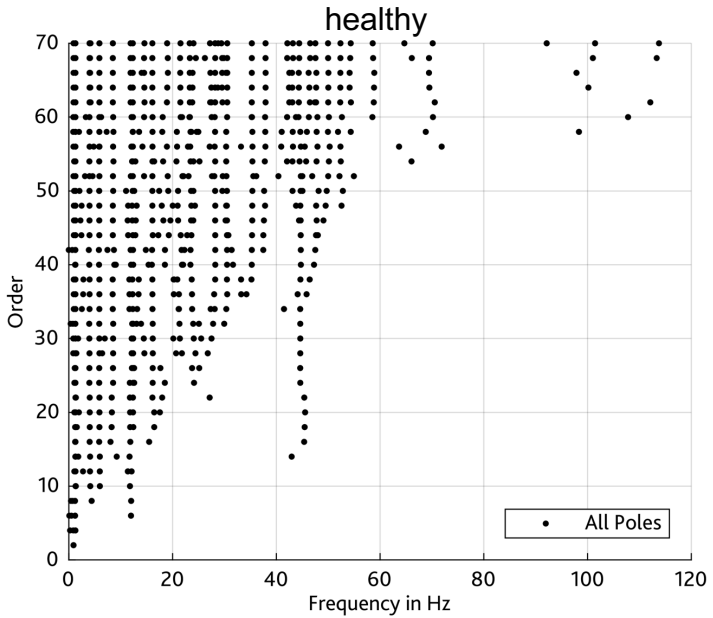
Vertical acc. Signal – *y* in ssi algorithm

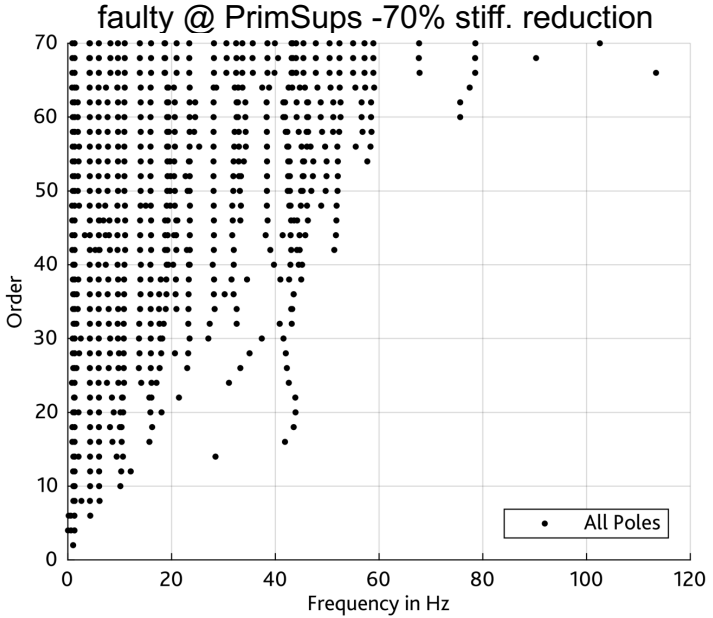


# 9.1 Conventional SSI Procedure for Modal Property Extraction

## Subspace Identification – Extracted Modal Properties

**Stabilization Diagram based on the SSI Algorithm [1] (prediction horizon r = 480, AR-model order s = 480)**



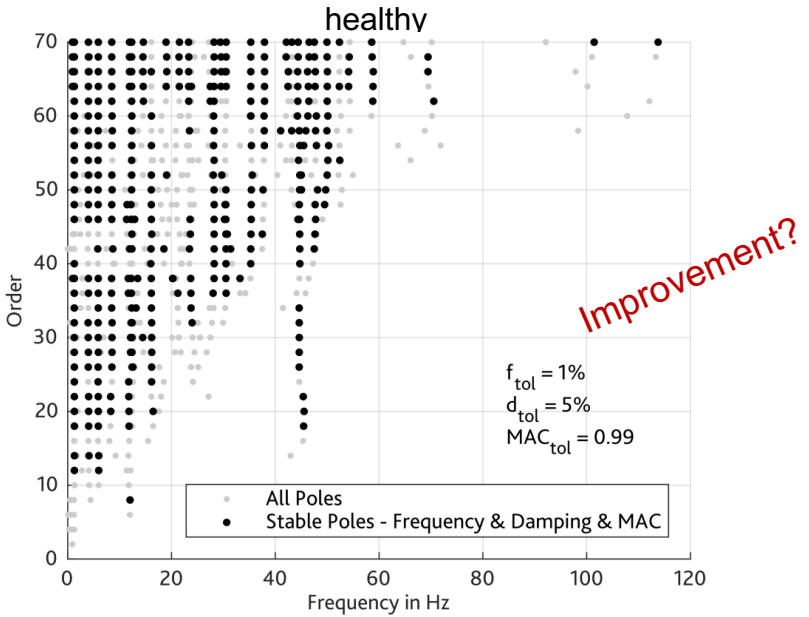

*It is difficult to compare and classify different scenarios based on a general visual inspection.*

[1] H. Jung; T. Münker; G. Kampmann; K. Rave; C.-P. Fritzen; O. Nelles, *A Novel Full Scale Roller Rig Test Bench for SHM Concepts of Railway Vehicles*. In: 8th European Workshop on Structural Health Monitoring. Bilbao, Spain, 2016.

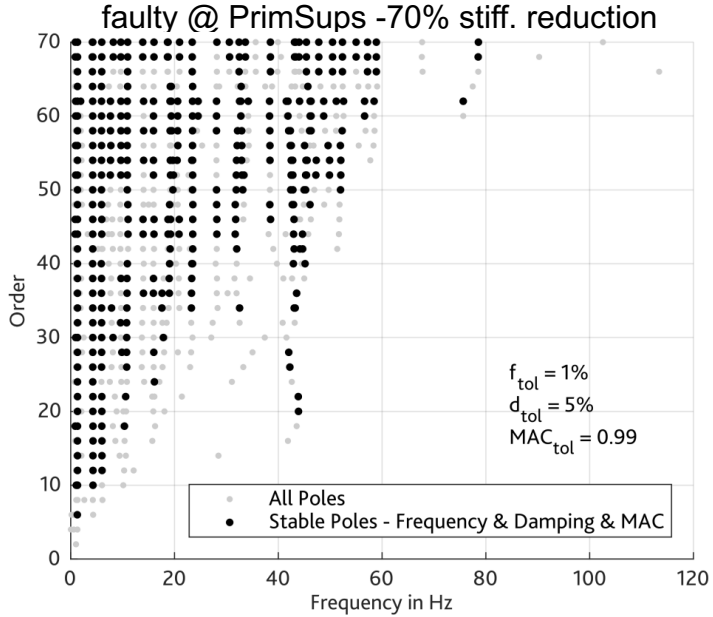
# 9.1 Conventional SSI Procedure for Modal Property Extraction

## Subspace Identification – Extracted Modal Properties

**Stabilization Diagram based on the SSI Algorithm [1] (prediction horizon r = 480, AR-model order s = 480)**




Improvement?




*Also, with the stabilized frequencies it is difficult to compare and classify different scenarios.*

[1] H. Jung; T. Münker; G. Kampmann; K. Rave; C.-P. Fritzen; O. Nelles, *A Novel Full Scale Roller Rig Test Bench for SHM Concepts of Railway Vehicles*. In: 8th European Workshop on Structural Health Monitoring. Bilbao, Spain, 2016.

# 9.1 Conventional SSI Procedure for Modal Property Extraction

## Statement of the problem

### Problem:

The conventional SSI procedure calculates the system frequencies with an unknown uncertainty. The poles can be related to physical poles but also to spurious poles (like numerical poles etc.)

### Assumption:

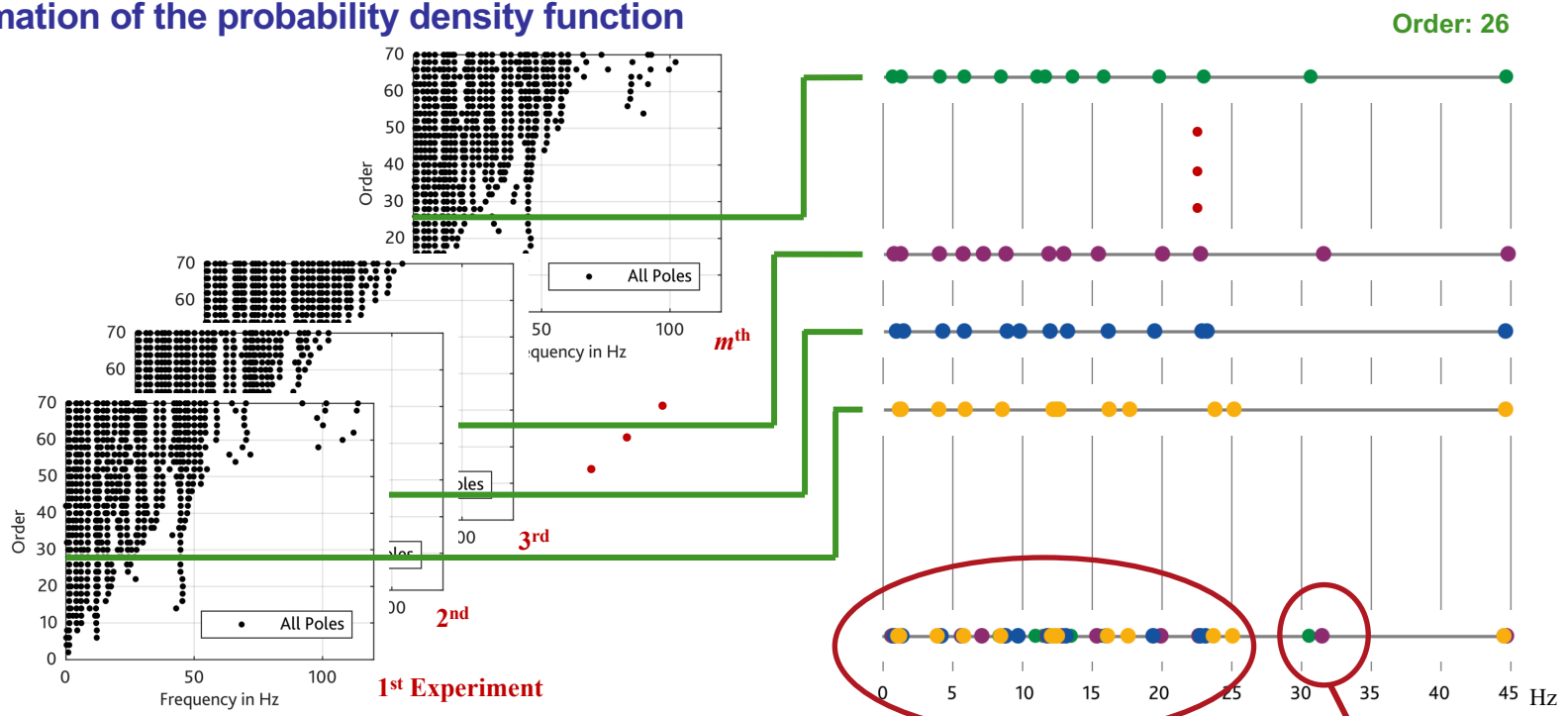
Physical poles will be the dominant poles in a set of different SSI calculations.

### Idea:

The most dominant / probable poles (the physical poles) occur with a high value in the probability density function of the eigenfrequencies.

# 9.2 Development of the Eigenfrequency Density Estimator (EFDE)

## Estimation of the probability density function




The *m* experiments are done with *m* different excitations, but with the same health state.

Region with a high frequency density

Region with a low frequency density

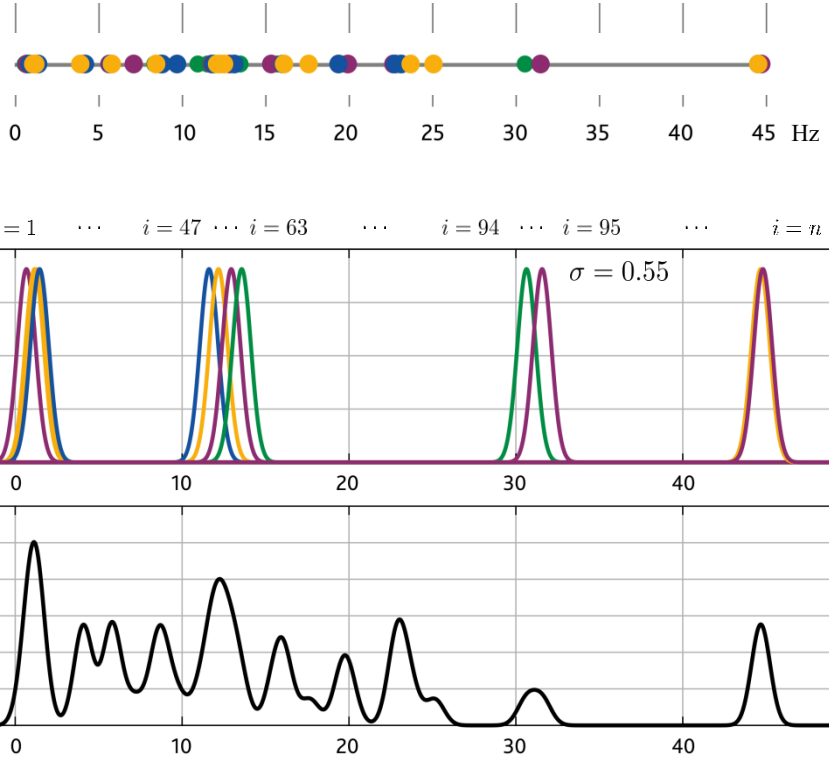
# 9.2 Development of the Eigenfrequency Density Estimator (EFDE)

## Estimation of the probability density function

- Lets denote the collected frequencies f1, f2, … fn as a sample of size n from a random variable with density function p(x).
- **Assumption:** Every frequency is independent identically distributed (i.i.d.).
- Kernel Density Estimator with a Gaussian Kernel:

$$p_n(x) = \frac{1}{n} \frac{1}{\sqrt{2\pi\sigma^2}} \sum_{i=1}^{n} e^{-\frac{(x-f_i)^2}{2\sigma^2}}$$

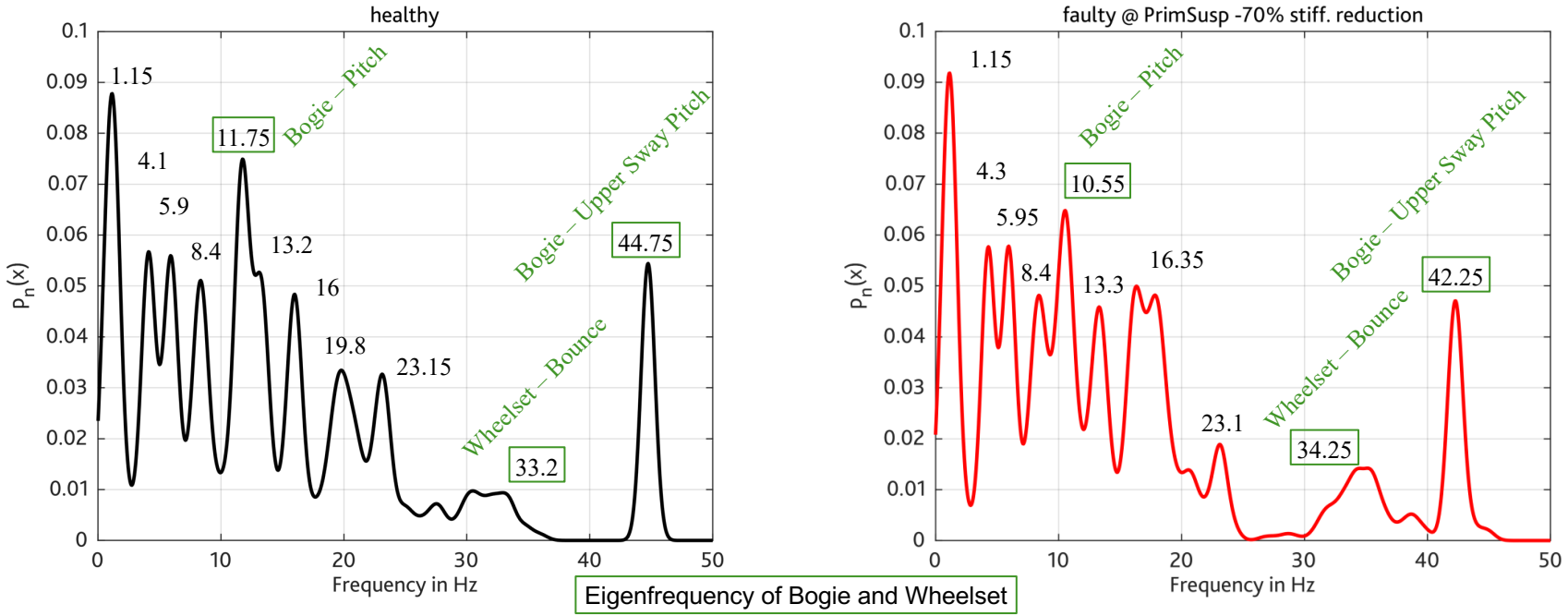
$n$ No. of samples
$\sigma$ Bandwidth resp. standard deviation




[1] Parzen, E. : On Estimation of a Probability Density Function and Mode. In: The Annals of Mathematical Statistics 33 (3), S. 1065–1076. 1962.

## Subspace Identification – Extracted Modal Properties

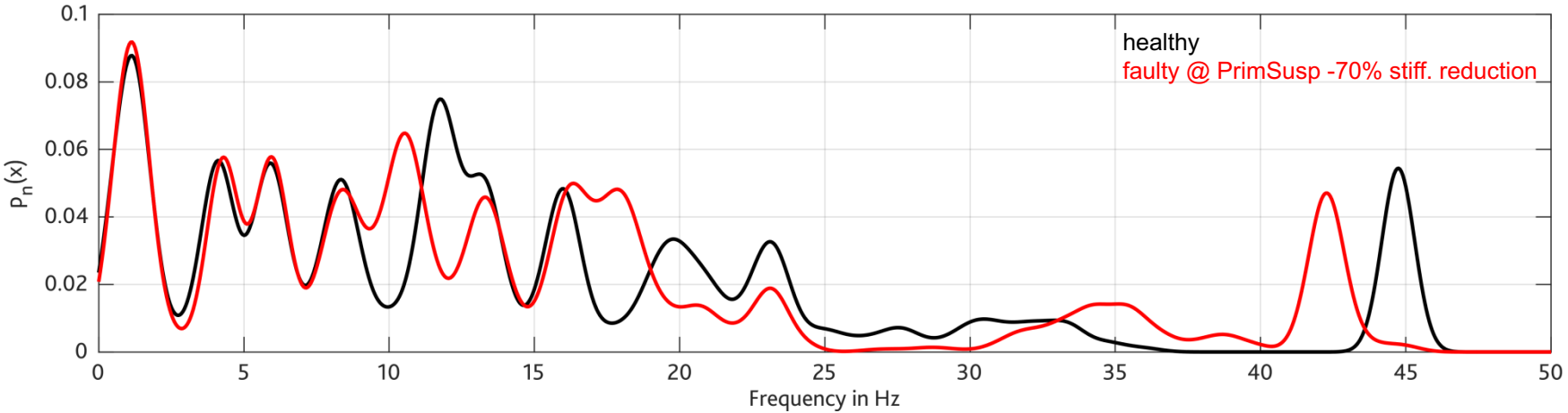
**Modal Parameters based on the SSI & EFDE (SSI: r = s = 480 / EFDE: p = 26, m = 99, σ = 0.55)**




*With the EFDE it is possible to analyze the stabilization plots automatically according to physical poles.*

## Health Assessment

SSI: r = s = 480 / EFDE: p = 26, m = 99, σ = 0.55

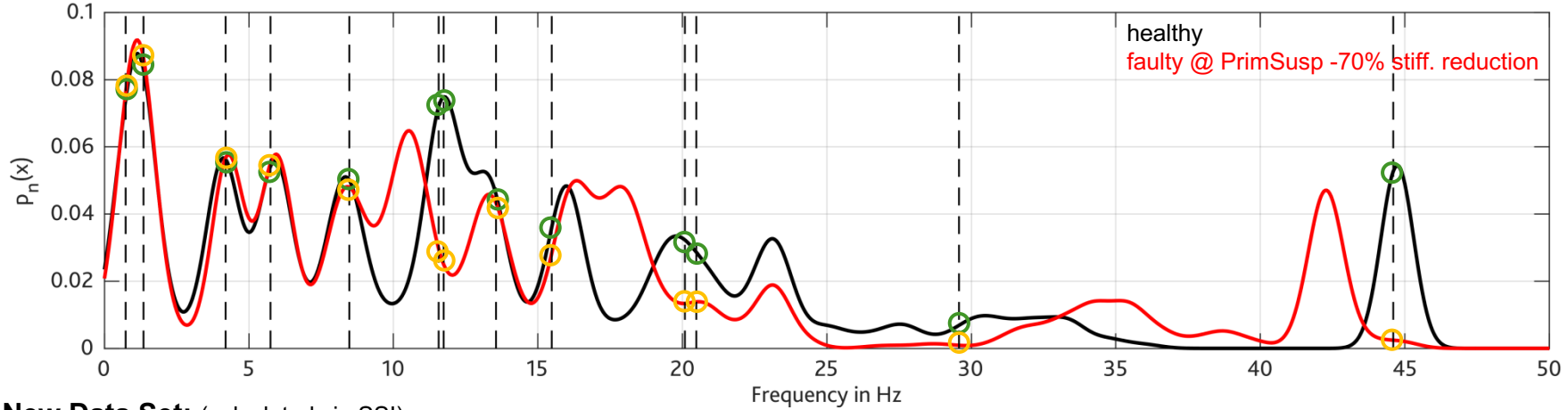



- Comparison shows that there is a significant difference between the healthy and faulty state.
- For a health assessment strategy the probability density functions can be used to distinguish if a new data set characterizes the faulty or healthy case by calculating the probability of the new data set concerning the faulty or healthy case.
- The higher the probability of a new data set with respect to a certain pdf (healthy or faulty), the greater the likelihood that it is this particular health condition.

# 9.3 Health Assessment Strategy with EFDE

## Health Assessment - Case 1 (healthy)

**SSI: r = s = 480 / EFDE: p = 26, m = 99, σ = 0.55**




**New Data Set:** (calculated via SSI)

$f$ = 0.73; 1.35; 4.19; 5.74; 8.47; 11.57; 11.74; 13.55; 15.48; 20.09; 20.48; 29.57; 44.60 Hz

**Calculate the Probability :**

$$P^{\text{healthy}}(f) = \frac{1}{n}\sum_{i=1}^{n} p_n^{\text{healthy}}(f_i) = 0.0514$$

$$P^{\text{faulty}}(f) = \frac{1}{n}\sum_{i=1}^{n} p_n^{\text{faulty}}(f_i) = 0.0371$$

**Decision:**

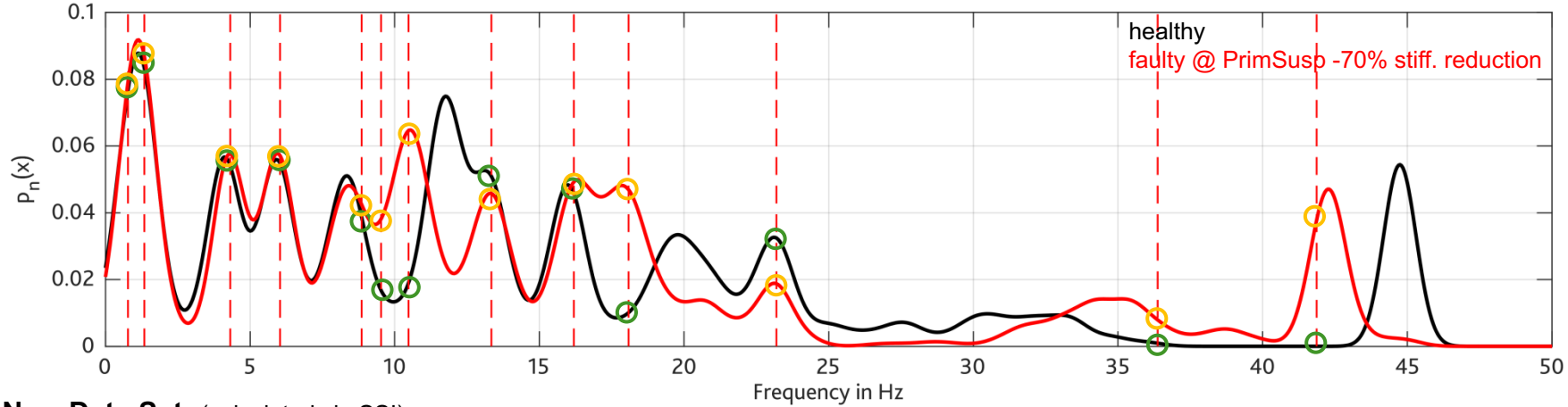
$$P^{\text{healthy}}(f) > P^{\text{faulty}}(f)$$

*With a higher probability the data set indicates a healthy system state.*

# 9.3 Health Assessment Strategy with EFDE

## Health Assessment - Case 2 (faulty)

**SSI: r = s = 480 / EFDE: p = 26, m = 99, σ = 0.55**




**New Data Set:** (calculated via SSI)

$f$ = 0.78; 1.34; 4.31; 6.04; 8.86; 9.53; 10.48; 13.34; 16.19; 18.08; 23.19; 36.37; 41.86 Hz

**Calculate the Probability :**

$$P^{\text{healthy}}(f) = \frac{1}{n}\sum_{i=1}^{n} p_n^{\text{healthy}}(f_i) = 0.0373$$

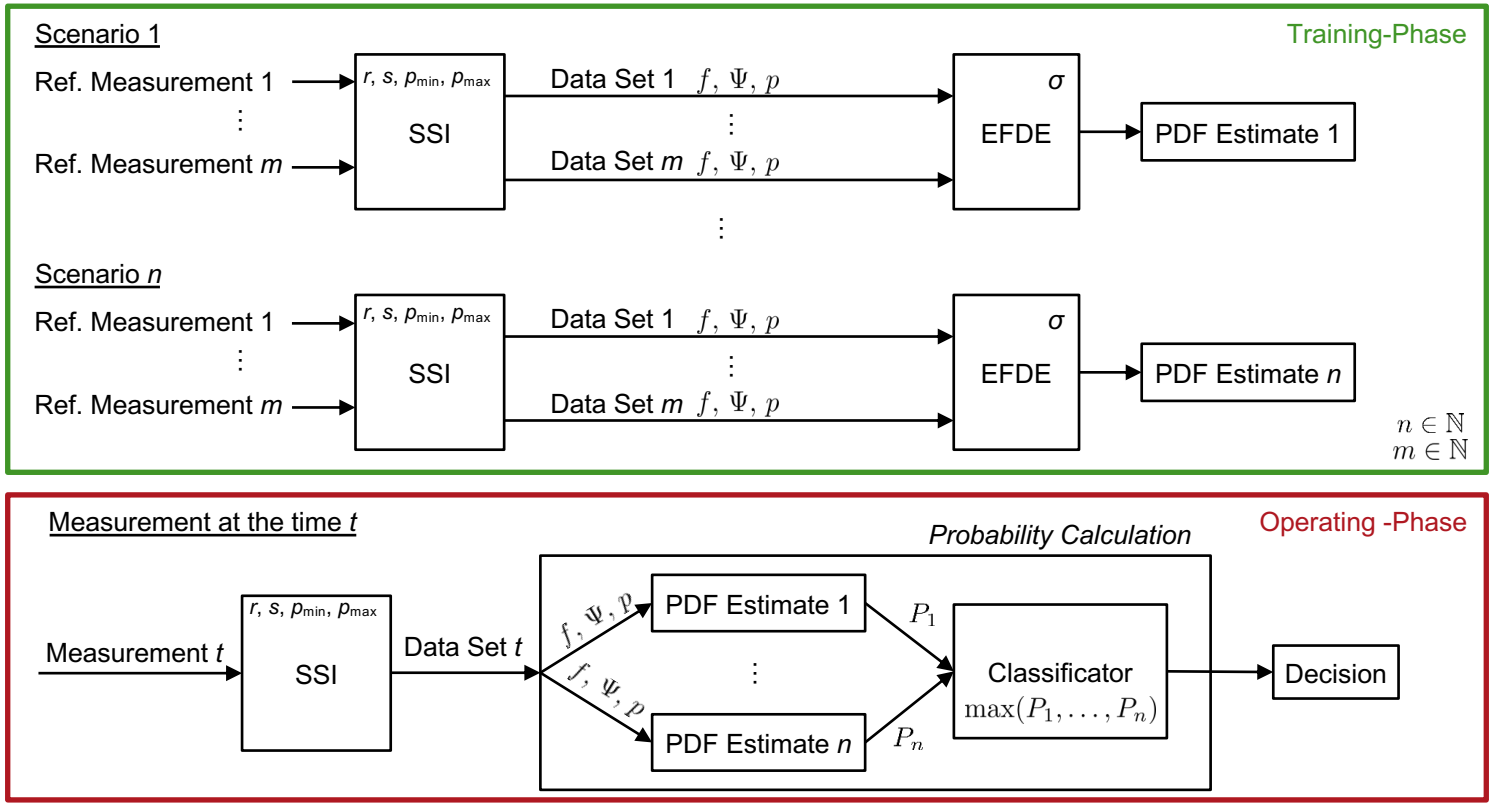
$$P^{\text{faulty}}(f) = \frac{1}{n}\sum_{i=1}^{n} p_n^{\text{faulty}}(f_i) = 0.0487$$

**Decision:**

$$P^{\text{faulty}}(f) > P^{\text{healthy}}(f)$$

*With a higher probability the data set indicates a faulty system state.*

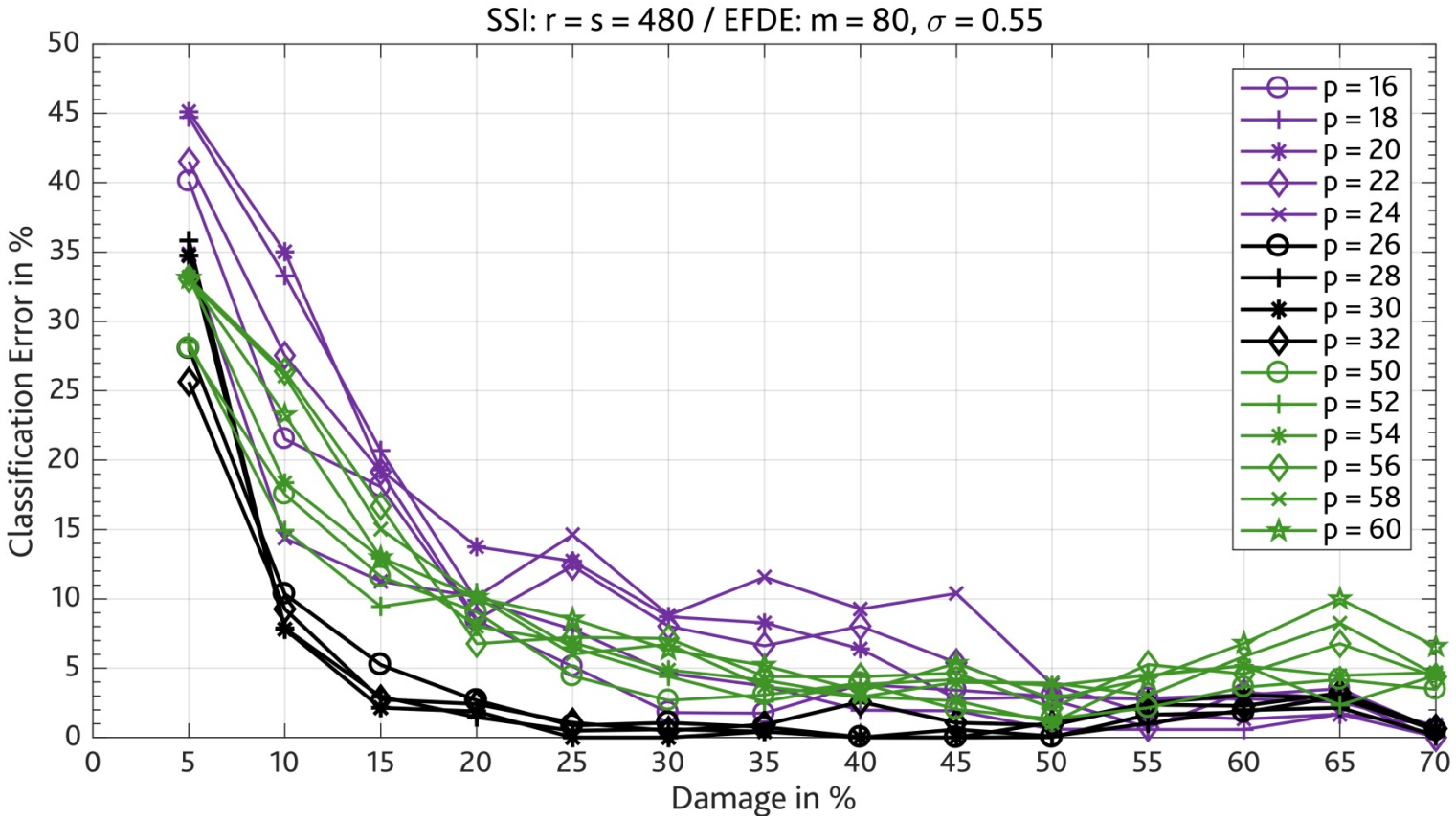
## Health Assessment - General Framework



# 9.3 Health Assessment Strategy with EFDE

## Health Assessment Sensitivity

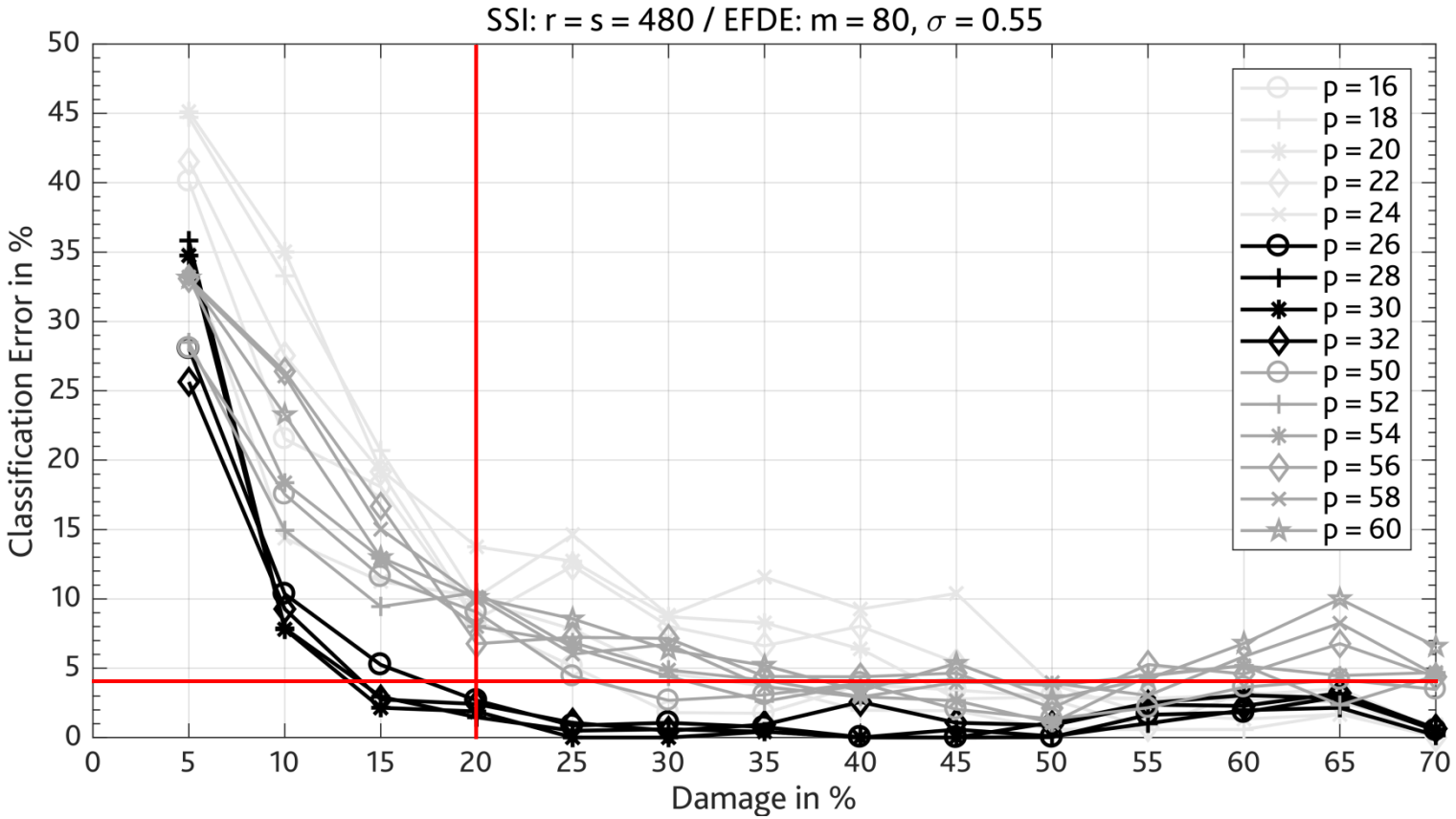
*The convergence is independent of the order.*



# 9.3 Health Assessment Strategy with EFDE

## Health Assessment Sensitivity

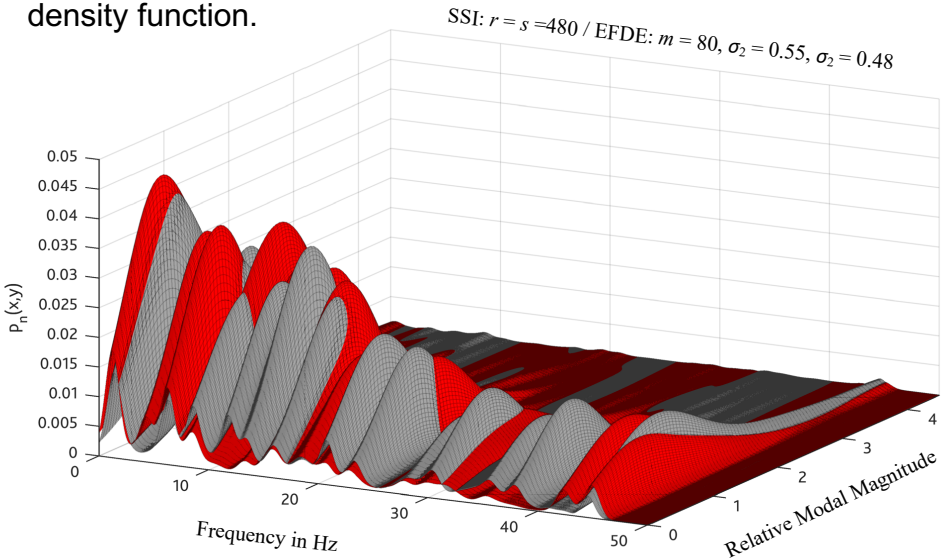
*A stiffness reduction greater then 20 % can be detected (p = 26 … 32) with a classification error less then 4 %.*

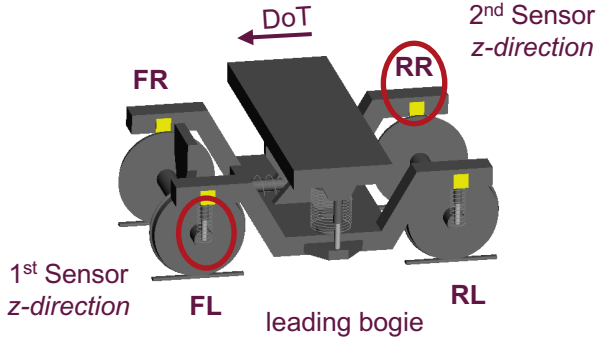


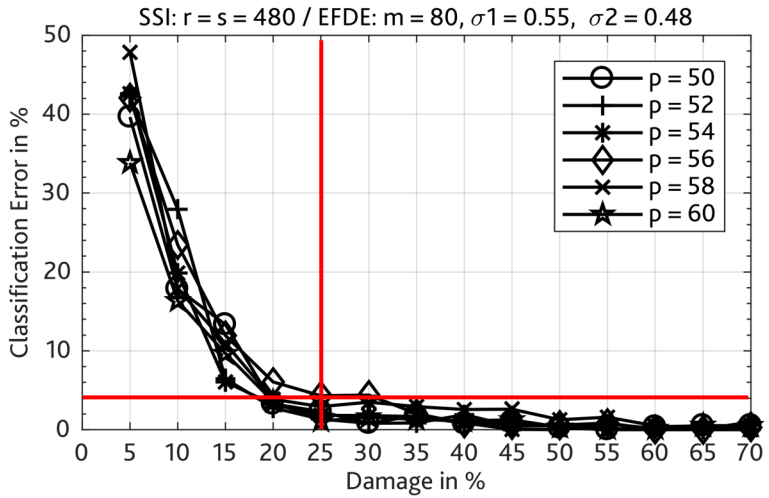
# 9.4 EFDE with Multiple Sensor Signals

## Health Assessment with Multiple Sensor Signals

- The previous results based on one sensor signal only (acc. z-direction Bogie Front Left).
- By using a second sensor signal (e.g. acc. z-direction Bogie Rear Right) a second feature (e.g. Relative Modal Magnitude) can be used for calculating a 2-dim probability density function.



## Conclusion & Outlook

- Bogies represent a big part of the total economical value of a railway vehicle and is an essential element of each rail vehicle that needs to be investigated for a sustainable rail vehicle concept.
- The presented Eigenfrequency Density Estimator and Health Assessment strategy is suitable to detect failures in the bogies suspension system.
- Subject of future research will be the investigation of the influence of the rail-wheel-contact forces, varying masses and velocities.

# 10. Case Study: Fehlerdiagnose Eisenbahnschiene

UNIVERSITÄT SIEGEN

Naturwissenschaftlich Technische Fakultät

# A Structural Health Monitoring Approach for Rail Fastening Systems

Railways 2022 – Montpellier France

Daniel Pak, Oliver Nelles, Peter Kraemer, Geritt Kampmann

## Content



Magnetic inductive sensors at the elevated rail of a bridge (1.2m spacing).

# 10.1 Introduction

## Task

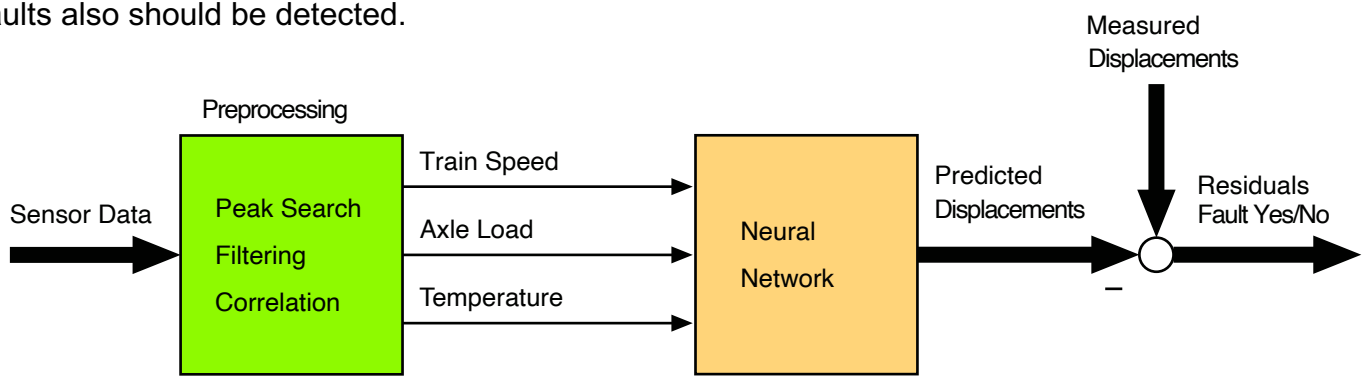
- Automated Structural Health Monitoring (SHM) of rail fastening systems: Early detection of faults, avoidance of derailments.
- Pilot project funded by Deutsche Bahn (DB): Sensors were installed on a railway bridge to measure the (horizontal) deformation of the rail against the derailment guard during train crossings.
- Simple monitoring of the displacement and comparison with a fixed limit value is not sufficient: Faults will then only be detected if a train is heavy (and fast) and the situation may already be close to failure.

## Approach

- Beside the displacement, other quantities (additional features) are considered as well:
  - **Axle Load**: Is not directly measured. Must be estimated from other data.
  - **Train Speed**: I also not directly measured. Must be estimated from other data.
  - **Temperature**: Thermal expansion influences the rail movement. Temperature measurement available.
- The additional measurements are used to:
  - to predict the expected displacement of a fault-free rail fastening as precise as possible (regression problem with load, speed and temperature as inputs)
  - and compare it to the actually measured value of displacement.
- The use of machine learning methods is considered for these approaches (e.g. neural networks, local model networks).

## Data Preprocessing

Most methods rely on data preprocessing. At this stage sensor faults also should be detected.

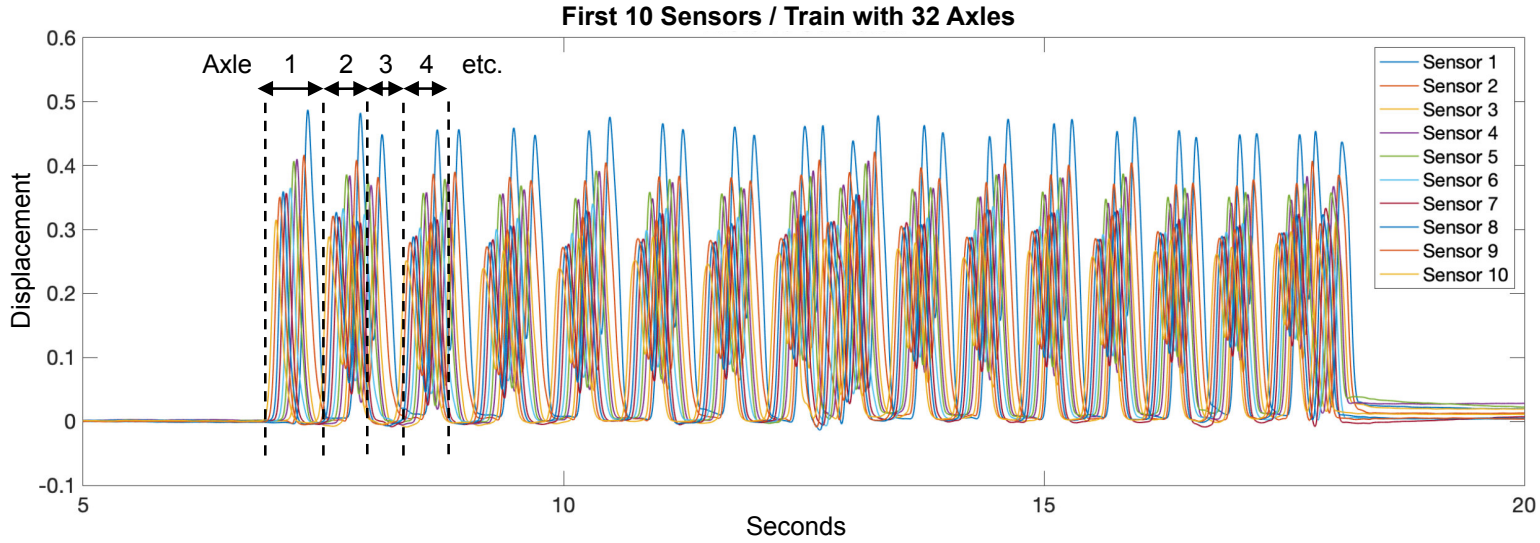



## Regression (Prediction)

- Prediction of displacement of fault-free rail fastenings.
- No data with faulty fastenings necessary, therefore preferred over classifiction approaches.
- Discrepancy between model and measurement (residual)used for fault detection.

# 10.3 Analysis of a Train Crossing

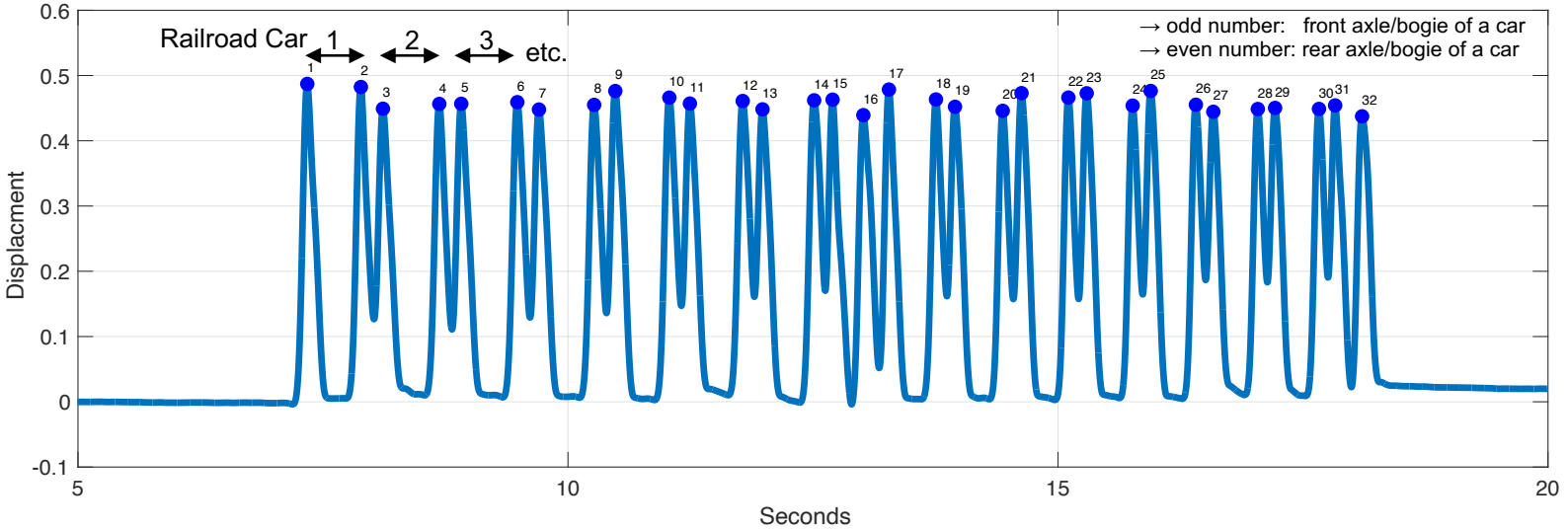
## Data Example for one Typical Train Crossing




- Different maximum values for the **same sensor** (same color) indicate **different axle loads**.
- Different maximum values for the **same axle** at different sensors indicate ...
  - in the **fault-free** case: different sensor calibration (correction necessary).
  - otherwise: a damaged rail fastening near the corresponding sensor.
  - The data (which is collected on a fault-free rail) clearly indicates the necessity of sensor correction.
- The train speed can be determined using the time shift of the sensor signals (e.g. using cross-correlation).

# 10.3 Analysis of a Train Crossing: Maximum Rail Displacement

**Analysis of one Sensor**

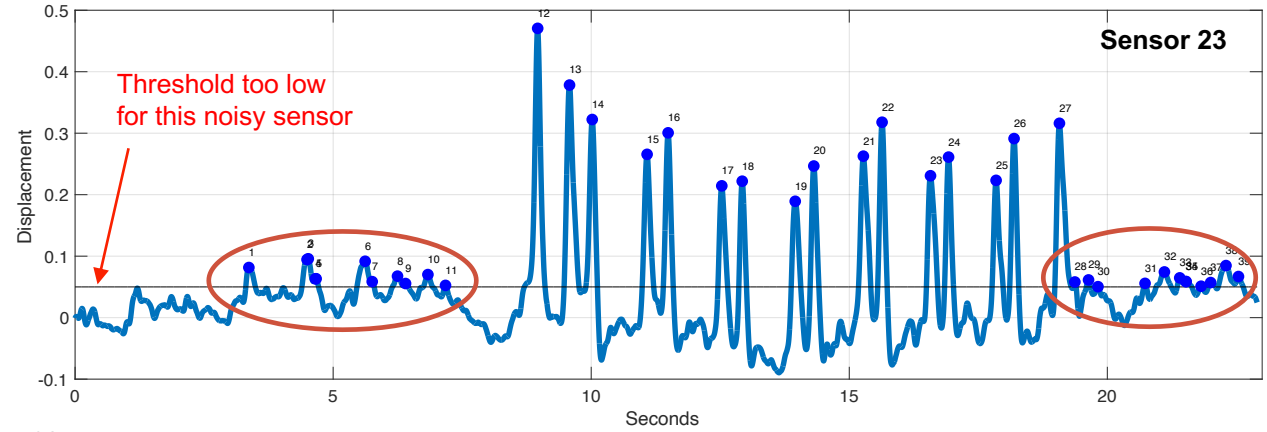


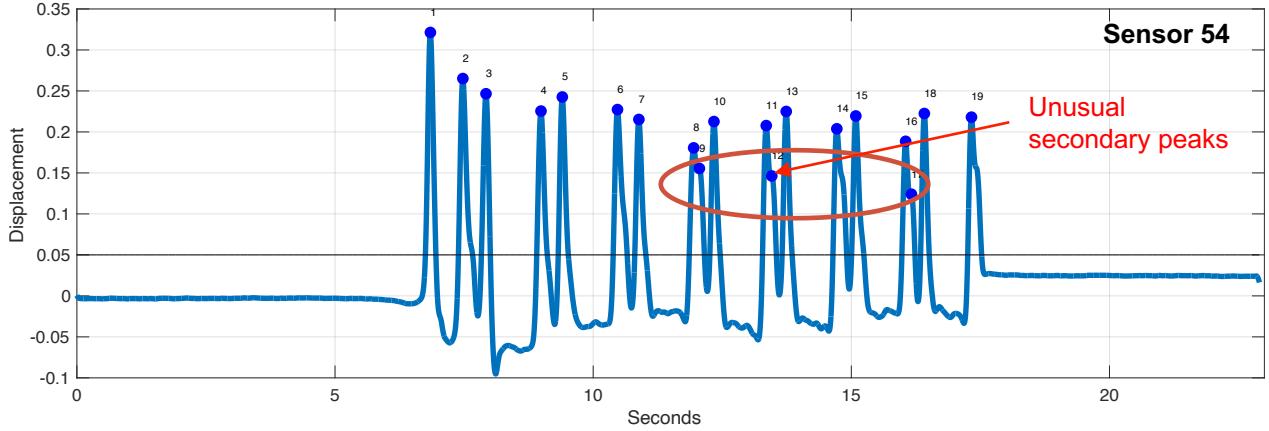

- Data has to be set to zero at the beginning (subtract mean value of a certain number of samples).
- Filtering of noise caused by the power line frequency (16 2/3 Hz) (eg. 60$^{th}$ order moving average filter for 1ms sample rate).
- The above diagram shows the correct identification of all 32 displacement peaks for the 32 axles/bogies of a 16 car train.
- The **method used for peak localization** is shown on the **next slide**.
- **For regular trains, no shared (Jacobs) bogies between 2 cars the peak number has to be even! Detection of an odd number of peaks would mean faulty measurements (sensor defect, strong noise, etc.).**

# 10.3 Analysis of a Train Crossing: Maximum Rail Displacement

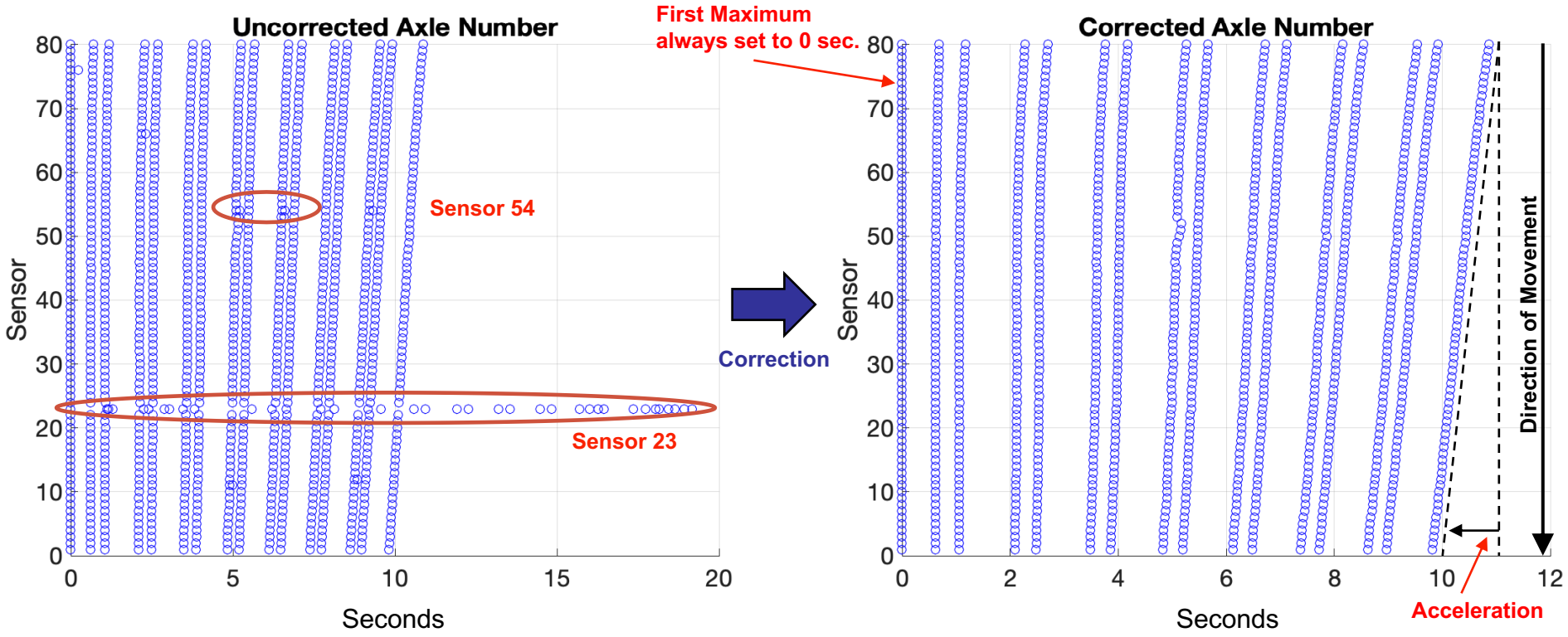
## Method for Peak Localization and Handling of False Peaks

- **Peak localization:** Compare 3 consecutive data points $y(k)>y(k+1)$ and $y(k)>y(k-1)$.
- **Threshold:** Avoid false peaks.
- **Robustness:** Number of maxima for each sensor is determined. If different, the most frequent $N_A$ is used.
- **Correction of false Peaks:** Excess maxima are removed by selecting only the $N_A$ highest values.
  - Top diagram: Unsuitable threshold or unusual noise → false peaks.
  - Lower diagram: Threshold okay, but unusual secondary peaks.
  - If the majority of sensors deliver the correct number of peaks, correction is possible → **next slide**.



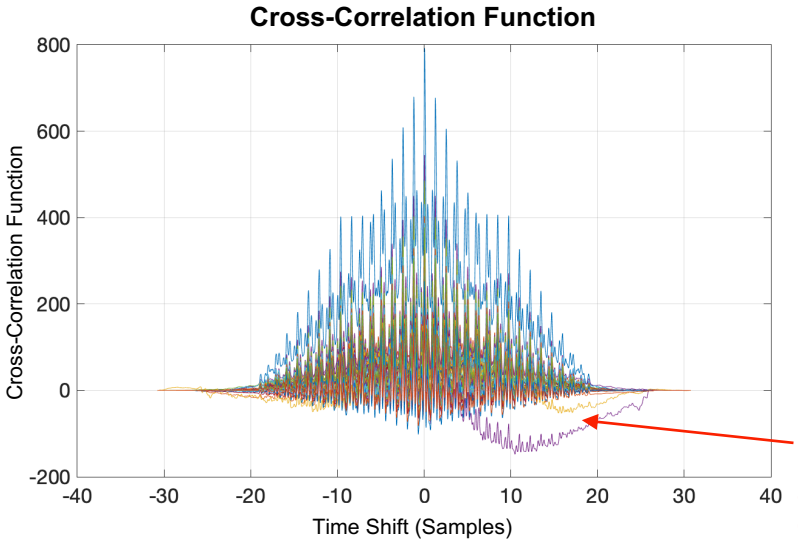



# 10.3 Analysis of a Train Crossing: Maximum Rail Displacement



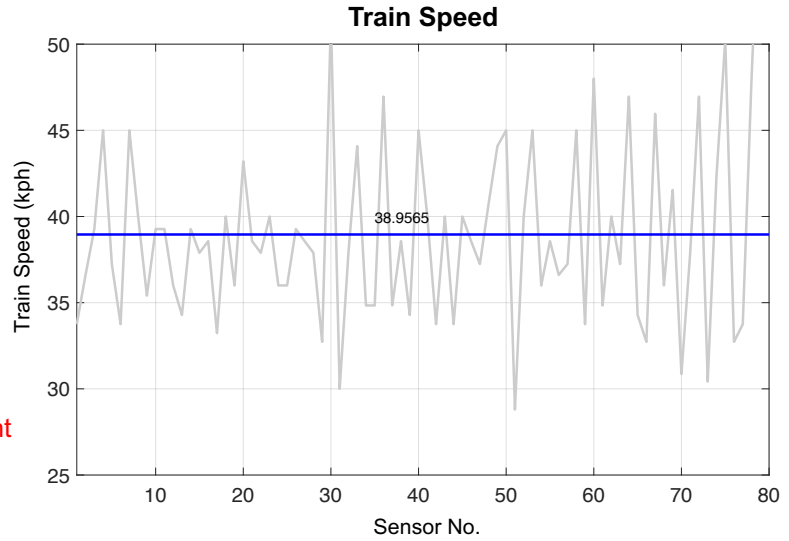

- In most cases the axle number can be corrected to the actual number (depending on the extend of the sensor disturbances).
- **An acceleration of the train during the crossing can easily seen from this diagrams!**

# 10.3 Analysis of a Train Crossing: Speed Estimation




Calculation and Averaging

Faulty measurement (one defect sensor affects two cross-correlations)




- **Left diagram:** Cross-correlation of displacements two adjacent sensors (79 correlations from 80 sensors):
  - For every correlation function: Estimation of speed from position of maximum, known sensor distance and sampling interval → 79 estimations of the train speed (**right diagram**).
- **Right diagram:** Averaging of the speed values leads to robust speed estimation, despite of at least one faulty sensor (left diagram and possibly slightly nonuniformly distributed sensor positions.

# 10.3 Analysis of a Train Crossing: Axle Load Estimation

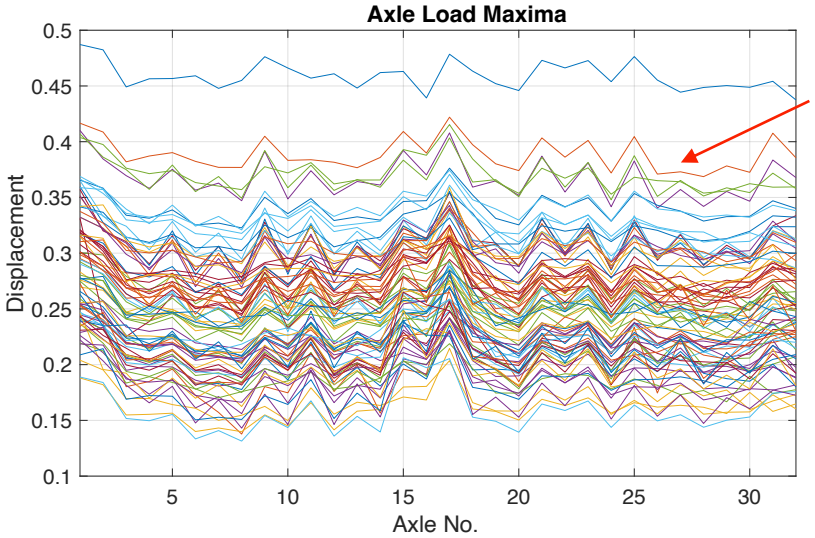
## Different Sensor Calibration

- Same maximum displacement for **same axle at all sensors** if the rail fastening is fault-free:
  - As this is not the case, a **correction factor** must be determined for **each sensor** to ascertain approximately equal displacements.
  - A simple correction factor is used. More complex (nonlinear) correction models are considered if necessary and more data is available.
  - Calibration may drift over time with the used sensors. Recalibration may be necessary.
- The train speed (and possibly accelerations or decelerations) might also have an influence on the correction.

## Different Axle Loads

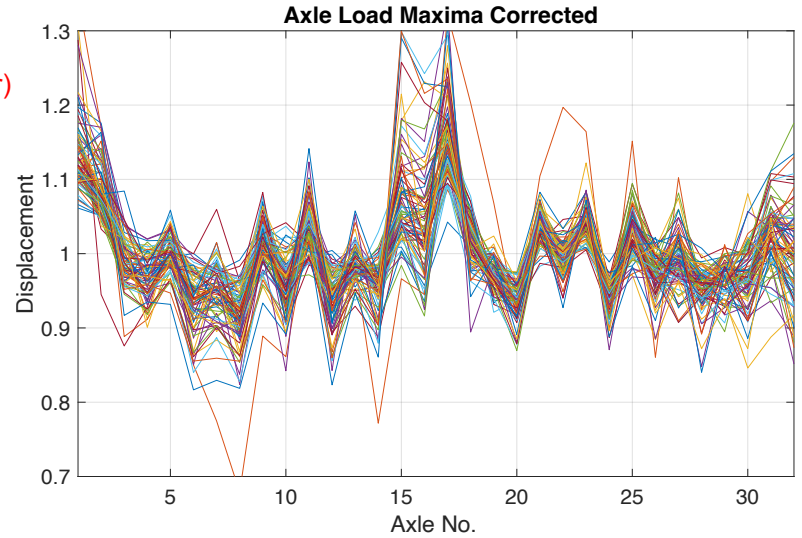
- From the different maximum values of the **different axles** at the **same sensor** different axle loads can be deduced.
- An estimation of the **relative axle load** should therefore be possible:
  - Simple linear model (factor) is used so far.
  - A more complex nonlinear model (and possible dependence on train speed, acceleration) may be investigated if more data is available.

# 10.3 Analysis of a Train Crossing: Axle Load Estimation




Curves (one per sensor) should be identical!

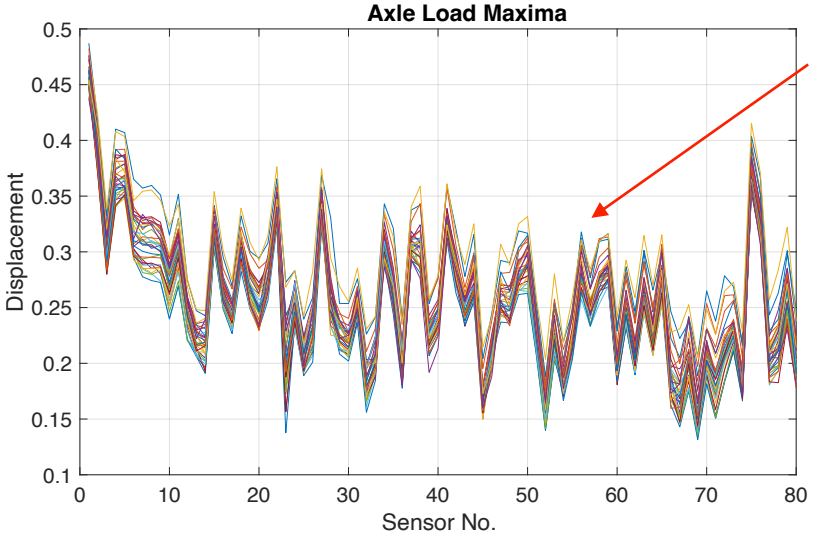
Korrektur




- **Left diagram:** Uncorrected maxima of displacement over all 32 axles at all 80 sensor positions.
  - Ideally (fault-free rail fastenings) all 80 curves should be identical since each axle should cause the same displacement at every sensor positions. → **Necessity of sensor correction clearly recognizable!**
- **Right diagram:** Corrected displacements (simple approach):
  - Calculation of mean displacement over all axles (here 32) for every sensor (here 80).
  - Use of the 80 mean values as correction factor for the 80 sensors.

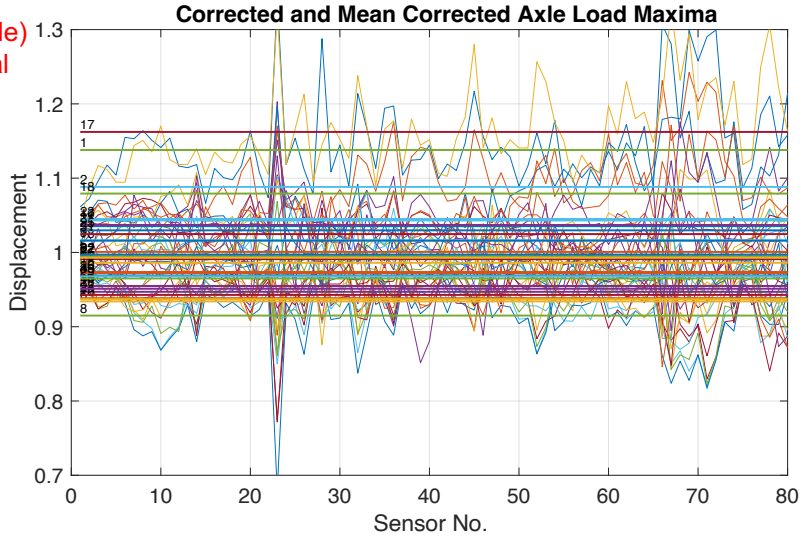
→ Correction: devision of each curve (left) by its mean value

# 10.3 Analysis of a Train Crossing: Estimation of Axle Load




Curves (one per axle) should be horizontal lines

Correction

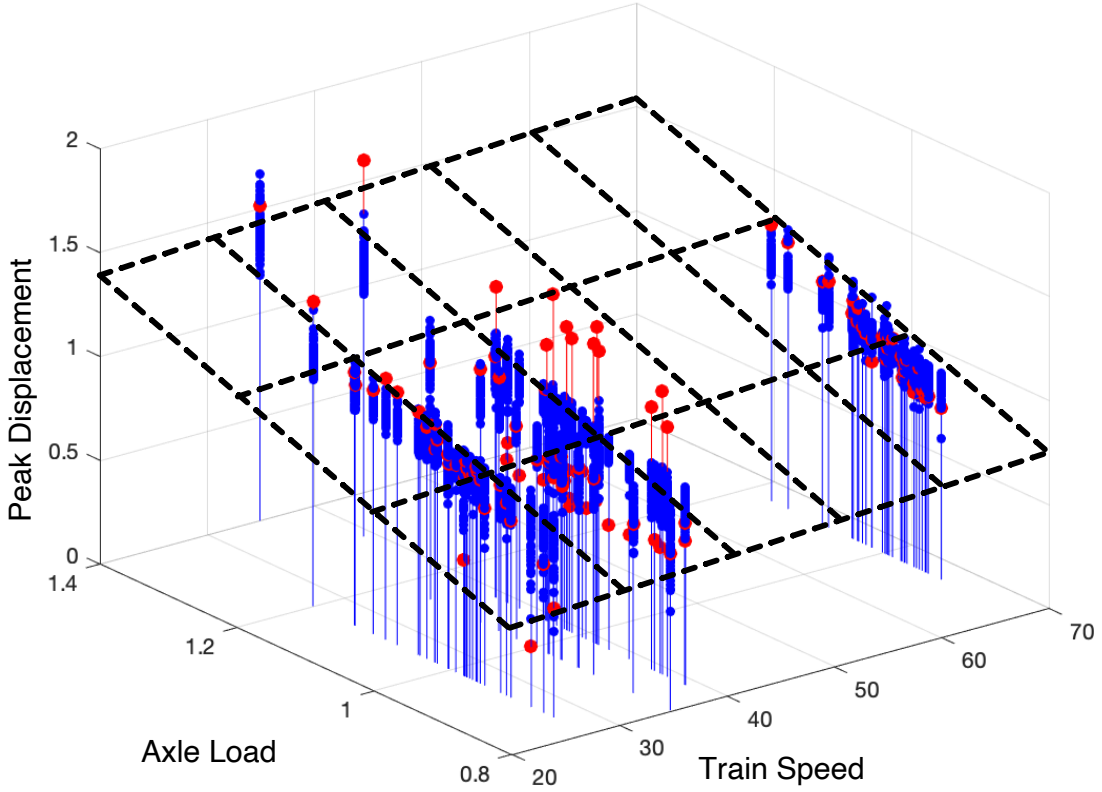



- **Left diagram:** Uncorrected maxima of displacement over all 80 sensor positions for all 32 axles:
  - Maxima should be approximately identical for each axle over all sensors and represent the axle load.
- **Right diagram:** Corrected maxima (with correction factors of previous slide):
  - Ideal constant values for each axle over all sensors can not be obtained.
  - But the mean value for each axle over all sensors is still meaningful: E.g. axles 1/2 und 17/18 are clearly detected as having the largest loads: Probably the power cars of the coupled half trains of an ICE 2.

# 10.4 Fault Detection – Linear Displacement Model

## Fault Detection Using Data From 4 Train Crossings

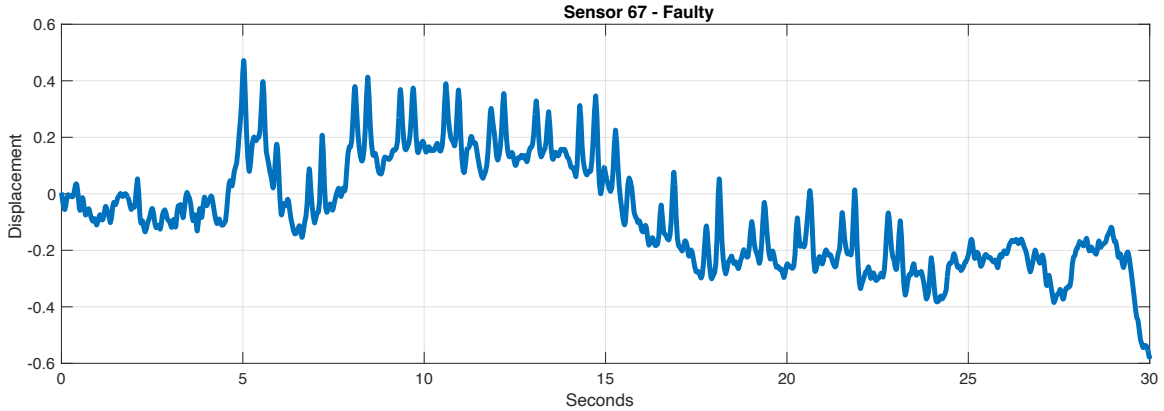
- The the stem plot shows the maximum displacement over axle load and train speed.
- There appears to be a mainly linear relationship between axle load, speed and displacement.
  - More crossings with will have to be analyzed for a more complex (nonlinear) model.
- Red dots show data of a sensor (no. 67) which is known to be unreliable:
  - The faulty sensor can clearly be detected using this simple linear model.
- So far, no data with actual fastening damage available.
- First results show that the chosen approach is suitable for fault detection.

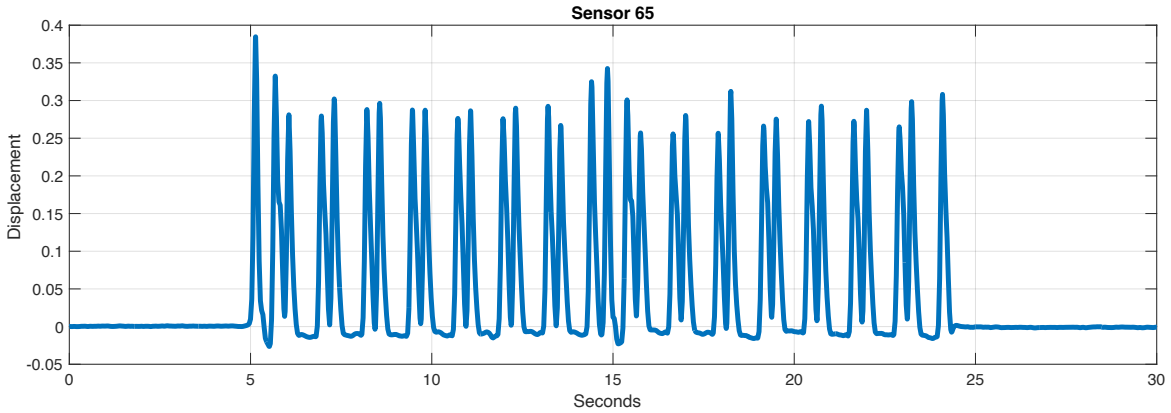


# 10.4 Fault Detection – Faulty Sensors

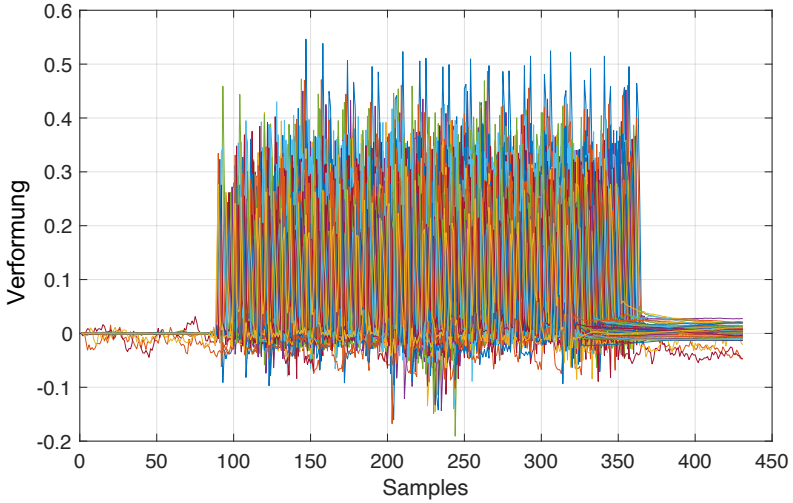
## Comparison Between Adjacent Sensors

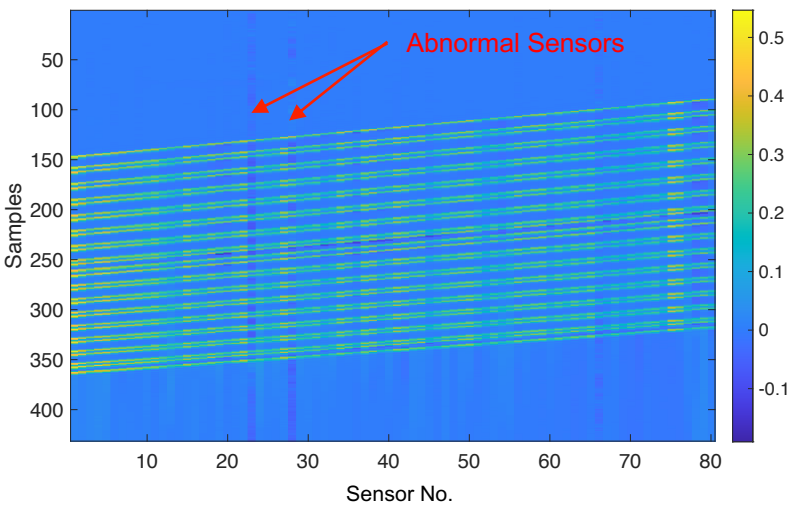
- The top diagram shows exemplary data of the faulty sensor (no. 67).
- The lower diagram shows data from the close-by sensor (no. 65) which shows the typical crossing pattern.
- Sensor faults could be detected by comparing data of adjacent sensors for similarity.
- This sensor was also stood out when the displacement maxima were detected, since it was not possible to find the same axle number compared to the other sensors.



# 10.4 Fault Detection – Visual Representation of a Train Crossing




- **Left diagram:** Measurement data of all 80 sensors of one crossing.
- **Right diagram:** Depiction of the same data as a heat map in two dimensions:
  - Every column of the picture contains the data of one sensor. The displacement is color coded.
  - The individual axis can be recognised as lines, their slope represents the train speed. Acceleration leads to curves.
  - Abnormal measurements may be recognized by color deviations.
  - This representation might be suitable for the use of convolutional neural networks (CNN) for fault detection.

# 10.5 Conclusions & Outlook

## Conclusions

- It has been shown, that it is possible to estimate the most important influences on the rail displacement from the data of a train crossing:
  - Number of axles and maximum displacement values.
  - Estimation of axle loads.
  - Estimation of train speed.
- For now, only 80 of the available sensors were considered.
- First studies to predict the maximum rail displacment have been executed:
  - Displacement should be predicted well using a regression model and faults be detected using residuals.

## Outlook

- Collection and examination of additional measured data.
- Training of Local Model Networks as regression models for the rail displacements:
  - Residual-based fault detection (starting with simulated rail fastening faults until actual data available).
- Investigation of classification methods instead of regression (1-class, 2-class).
- Considering temperature and other quantities (e.g. train acceleration) as additional inputs for regression / classification.