Maschinelles Lernen und Datenanalyse in der Mess- und Prüftechnik

PD Stefan Bosse

1. Inhalt

2. Überblick

2.1. Motivation

Dieser Online Kurs mit interaktiven Übungen soll:

- Einen **anwendungsorientierten Einstieg** in die Datenanalyse und Interpretation mit Verfahren des **Maschinellen Lernens** bieten;
- Einen **Überblick** über gängige und weniger gängige **Verfahren** geben;
- **Interaktive Tutorials und Übungen mit zielgruppenorientierten Fallbeispielen** sollen Verfahren begreifbar und erfahrbar machen!

2.2. Tandemkurs

- Dieser Kurs adressiert zwei primäre **Zielgruppen**:
  - FB 4: Produktionstechniker und Materialwissenschaftler (und SysEngs)
  - FB 8: Soziologen (und Psychos)
- Dabei gibt es zwei **Inhaltsstränge**:
  - Einen gemeinsamen Strang mit Grundlagen und Verfahren
  - Getrennte Stränge für Anwendungsbeispiele

2.3. Ontologie der Inhalte

- Die Ontologie des Kurses besteht aus den Bausteinklassen:
  - **Modelle**
  - **Verfahren** (Training, Test, Inferenz)
  - Überwachtes Training
  - Nicht überwachtes Training
- Weiterhin aus den Anwendungs- und **Datenklassen**:
  - Sensorische und experimentelle Daten (Mess- und Prüftechnik)
  - Erhebungs- und Umfragedaten (Soziologie)
  - Metrische und Kategorische Variablen

Die Grenzen der Datenklassen sind fließend! Der Mensch als Sensor!

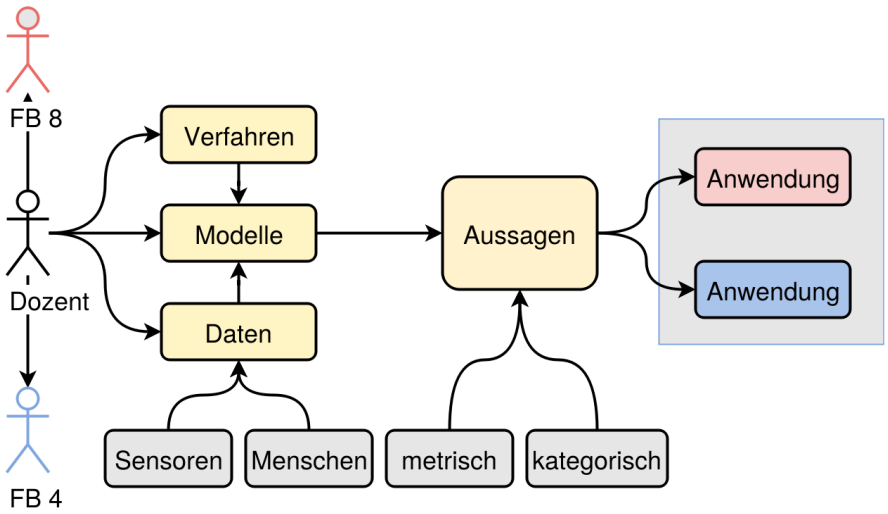


Abb. 1. Gemeinsame Verfahren und Modelle → Unterschiedliche Daten, Aussagen, Anwendungen

2.4. Ontologie der Veranstaltung

1. **Synchrone Vorlesungen mit Livestream** (experimentell!)
   - ➤ Studenten können über einen Chat/Eingabefeld Fragen stellen
   - ➤ Aufzeichnung der Vorlesung → 2.
2. **Asynchrone Video Vorlesungen und Tutorials** (alternativ)
   - ➤ Auch offline seh- und hörbar
3. **Gemeinsame Treffen mit** Videokonferenz (Zoom)
4. **Interaktive Tutorials** und Übungen mit *NoteBook* und ggfs. *WorkBook* im WEB Browser!
   - ➤ Offline ausführbar (evtl. werden Daten von einem Server geladen)
5. Texte und Folien
   - ➤ Vorlesungsskripte (am Anfang: für jedes Modul/jede Einheit) im PDF

- ➤ Das vorlesungsskript gibt die Folieninhalte 1:1 wieder (nur anderes Layout)
- ➤ Alle Folien im HTML Format (auch offline lesbar)
- ➤ Begleitende Literatur (Bücher im PDF)

2.5. Ontologie der Services

1. WEB Service: Informationen, Dokumente, Folien, Videos:http://edu-9.de/Lehre/ml2k
2. Dokuwiki: **News**, Informationen und Links, **Chats**, **Videostreams**: http://ag-0.de/dokuwiki
   - ➤ Registrierung und Login erforderlich
   - ➤ Interaktiv!
3. SAS: Student Assignment System (TODO): http://edu-9.de/cas
   - ➤ Registrierung und Login erforderlich
4. VIDEO: (Video Opencast Server http://ag-0.de)

2.6. Prüfungsleistungen

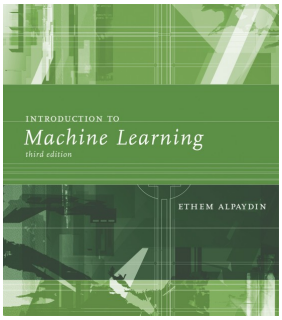
1. Eine mündliche Abschlussprüfung (20 Minuten); **oder alternativ** 2.
2. Eine schriftliche Seminararbeit (Experimentelle Arbeit oder Literaturrecherche)
   - ➤ 15-20 Seiten PDF
3. Bearbeitung und Abgabe der digitalen Übungen (JSON Dateien)

2.7. Literatur

- ➤ Zur Vertiefung!

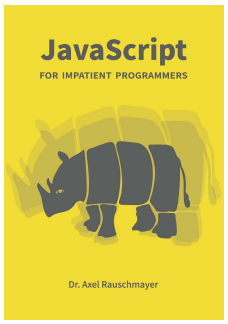
S. Richter, Statistisches und maschinelles Lernen. Springer Spektrum, 2019.

[www.pinterest.com] E. Alpaydın, Introduction to Machine Learning. MIT Press, 2010.

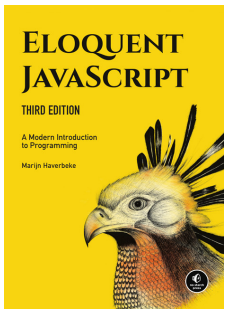


Programmierung

Axel Rauschmayer, JavaScript For Impatient Programmers.

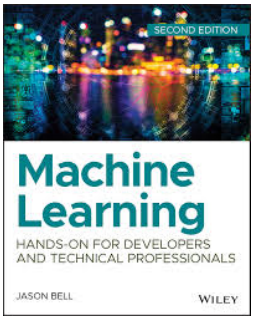


M. Haverbeke, Eloquent JavaScript. 2018.

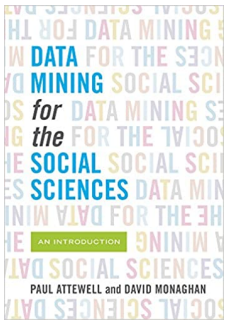


Domainspezifische Literatur

J. Bell, Machine Learning - Hands-On for Developers and Technical Professionals. John Wiley & Sons, Ltd, 2015.



P. Attewell and D. B. Monaghan, Data mining for the social sciences : an introduction. University of California Press, 2015.



2.8. Software

NoteBook

- ➤ Interaktive vorwiegend praktische Übungen werden rein digital im WEB Browser mit den *NoteBooks* durchgeführt
- ➤ Ein digitale Übung (oder Turorial) besteht aus:
  - ❑ Textabschnitten
  - ❑ Informationsblöcken
  - ❑ Aufgaben (mit Lösungen)
  - ❑ Editoren für Programmcode
  - ❑ Ausführungsterminals für Programmcode
  - ❑ uvm.

Abb. 2. Ein NoteBook im WEB Browser

NoteBook Konzept

- ➤ Top-down Bearbeitungsfluß
- ➤ Statische Struktur mit dynamischen Inhalten

- Alle dynamischen Inhalte können in einer JSON Datei gespeichert und wieder geladen werden
- Es können Notizzettel überall im NoteBook angeheftet werden (werden auch gespeichert)
- Musterlösungen (dynamische Inhalte) können eingebettet und mit einem Schlüssel freigeschaltet werden

2.9. WorkBook

- Dynamische Struktur mit dynamischen Inhalten
- Ein WorkBook besteht aus
  - Textabschnitten (MarkDown)
  - Codesnippets mit Editoren und Ausgabekonsolen
  - Speziellen Snippets wie editierbare Tabellen oder allg. Formulare
- Programmierung in JavaScript
- Alle dynamischen Inhalte und Daten können im JSON Format gespeichert und wieder geladen werden

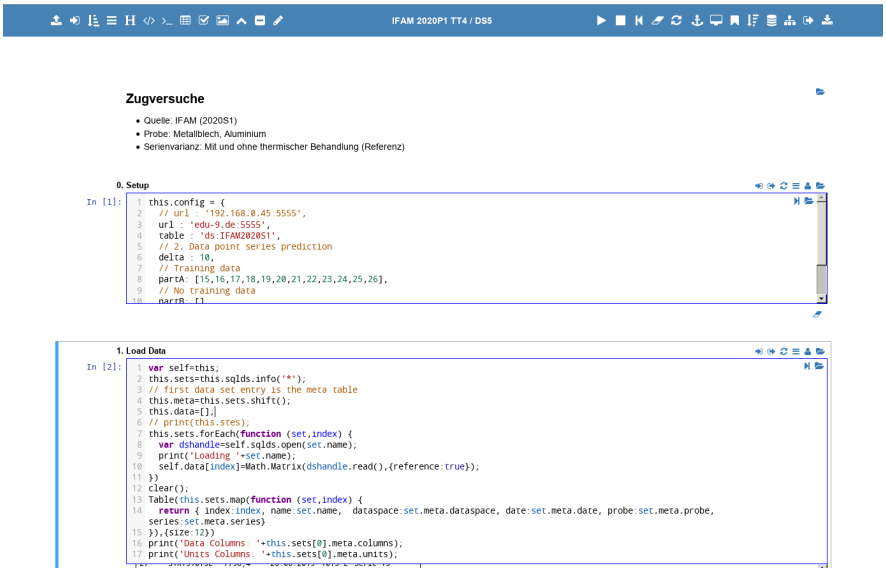


Abb. 3. Ein WorkBook Beispiel

2.10. Machinelles Lernen

Schlüsselwörter und Begriffe

Welche Begriffe werden häufig bei ML genannt:

Anwendungsgebiete

Welche Anwendungsgebiete gibt es:

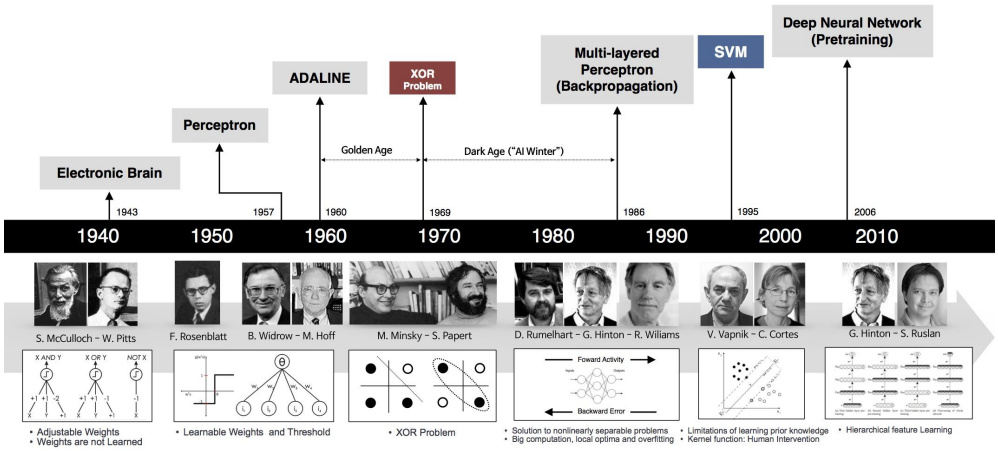
Fragestellungen

Welche Fragestellungen (zu lösende Probleme) gibt es:

2.11. Inhalte

1. **Eingabe** x: Daten (Attribute) und Eigenschaften (Analyse)
2. **Sensoren**: Erfassung von Daten, $S(welt)$: $welt \rightarrow x$
3. **Ausgabe** y: Numerische und kategorische Werte
4. **Metriken und Taxonomie**: Grundlagen des Maschinellen Lernens
5. **Algorithmen und Modelle**: $f(x)$: $x \rightarrow y$
6. **Training, Lernen, Prädiktion, Test** $M(<x,y>)$: $<x,y> \rightarrow f$
7. **Anwendungen**

2.12. Geschichte



[www.pinterest.com]

3. Daten und Sensoren

Metriken von Daten
Metriken von Aussagen
Sensoren als Datenquellen

3.1. Daten

- ➤ Daten sind die Grundlage für die Modellbildung und Modelltestung
- ➤ Daten können aus einer Vielzahl von Quellen stammen
  - ❏ Experiment
  - ❏ Simulation
  - ❏ Feldstudie
  - ❏ Abgeleitet aus anderen Datensätzen: **MapAndReduce**(D): $D \rightarrow D$'
- ➤ Allgemein kann man Daten und deren Werte unterteilen in:
  - ❏ Skalare Werte, wie Temperatur, Alter, usw.

- ❏ Serien von Skalaren Werten, wie Zeitserien
- ❏ Vektorielle Werte wie Bilder
- ❏ Zusammengesetzte Daten, also Datenrecords

➤ Daten haben daher eine Dimensionalität ^{N}, wobei die Wertemenge einer Dimension aus den ganzen , reelen , der Zeit oder kategorischen Wertemengen bestehen kann (oder Untermengen davon).

3.2. Datenreduktion

➤ Ziel der Datenanalyse ist die Reduktion von Eingabedaten bezüglich Größe und Dimensionalität:

$$P(X^N) : X^N \rightarrow Y^M$$
$$|Y| < |X|, M < N$$

```
1: function isRaining(temp,sunrad,moisture) =
2:   temp < 0 ? → false
3:   temp > 40 ? → false
4:   (sunrad-moisture) > 30? → false
5:   true
```

Beispiel einer Datenreduktionsfunktion $^{3} \rightarrow$

Datenklassen

Numerische und Metrische Werte

Das sind Werte die abzählbar sind und wo man Relationen (wie kleiner oder größer) sinnvoll definieren kann, also alle reellen und ganzen Zahlen.

➤ Beispiele: Temperatur, Länge, Ort, Zeit

Kategorische Werte

Das sind symbolische Werte für die entweder keine (sinnvolle) Ordnungsrelation existiert oder wo sich wenigstens keine Differenzen bilden lassen.

➤ Beispiele: Staatsangehörigkeit, Farbennamen (rot < gelb???), Schadenstyp

Skalierung der numerischen Werte

Intervallskaliert

Für diese Art von Attributen sind nur Unterschiede (Addition oder Subtraktion) sinnvoll. Beispielsweise wird die in °C oder °F gemessene Temperatur intervallskaliert. Wenn es 20 °C an einem Tag und 10 °C am folgenden Tag ist, ist es sinnvoll, über einen Temperaturabfall von 10 °C zu sprechen, aber es ist nicht sinnvoll zu sagen, dass es doppelt so kalt ist wie am Vortag.

Verhältnisskaliert

Hier kann man sowohl Differenzen als auch Verhältnisse zwischen Werten berechnen. Zum Beispiel kann man für das Alter sagen, dass jemand, der 20 Jahre alt ist, doppelt so alt ist wie jemand, der 10 Jahre alt ist.

Ordnungsrelationen

Nominal

Die Attributwerte in der Domäne sind ungeordnet und somit nur Gleichheitsvergleiche sinnvoll. Das heißt, wir können nur überprüfen, ob der Wert des Attributs für zwei bestimmte Instanzen gleich ist oder nicht. Zum Beispiel ist Geschlecht ein nominales Attribut.

Ordinal

Die Attributwerte sind geordnet und somit Gleichheitsvergleiche (ist ein Wert gleich einem anderen?) und relationale Vergleiche (ist ein Wert kleiner oder größer als ein anderer?) sind erlaubt, obwohl es möglicherweise nicht möglich ist, die Differenz zwischen den Werten zu quantifizieren!

3.3. Daten

Datensätze als Matrizen

➤ Ein Menge von Daten kann in **Matrizenform** als Matrix ***D*** dargestellt werden (Analogie zur Tabellenform) [1]:

$$\mathbf{D} = \left(\begin{array}{c|cccc} & X_1 & X_2 & \cdots & X_d \\ \hline \mathbf{x}_1 & x_{11} & x_{12} & \cdots & x_{1d} \\ \mathbf{x}_2 & x_{21} & x_{22} & \cdots & x_{2d} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{x}_n & x_{n1} & x_{n2} & \cdots & x_{nd} \end{array}\right)$$

- Die Zeilen sind Rekords der Variablenmenge $\{X_\mathrm{i}|\mathrm{i}=1,d\}$ und geben als d-stelliges Tupel je nach Anwendung und Zielsetzung einzelne Beispiele, Instanzen, Experimente, Entitäten, Objekte, und Eigenschaftsvektoren wieder

$$\tilde{x_i} = (x_{i1}, x_{i2}, .., x_{id})$$

- Der Vektor $\boldsymbol{X}$ ist die Menge aller Variablen (Sensoren) und die Spalten der Matrix $\boldsymbol{D}$:

$$\tilde{X} = (x_1, x_2, .., x_d)$$

```
type row = { X1:number, X2:number, .., Xd:number }
type table = row array;
```

3.4. Eingabe- und Ausgabevariablen

- Sensoren sind typischerweise Eingabevariablen x
- Aussagen sind Ausgabevariablen y, also Ergebnisse die sich aus den Eingangsvariablen ableiten lassen können (durch eine Funktion F):

$$\tilde{X} = (x_1, x_2, .., x_u, y_1, y_2, .., y_v)$$
$$\tilde{x_i} = (x_{i1}, x_{i2}, .., x_{iu}, y_{i1}, y_{i2}, .., y_{iv})$$
$$F(\tilde{x'}) : \tilde{x'} \rightarrow \tilde{y},$$

mit $u+v=d$.

3.5. Beispiel einer Datenmatrix

- Botanischer Datensatz mit geometrischen (numerischen) Eigenschaften einer Pflanze und kategorischer Klassifikation:

	Sepal length X_1	**Sepal width** X_2	**Petal length** X_3	**Petal width** X_4	**Class** X_5
$\mathbf{x}_1$	5.9	3.0	4.2	1.5	Iris-versicolor
$\mathbf{x}_2$	6.9	3.1	4.9	1.5	Iris-versicolor
$\mathbf{x}_3$	6.6	2.9	4.6	1.3	Iris-versicolor
$\mathbf{x}_4$	4.6	3.2	1.4	0.2	Iris-setosa
$\mathbf{x}_5$	6.0	2.2	4.0	1.0	Iris-versicolor
$\mathbf{x}_6$	4.7	3.2	1.3	0.2	Iris-setosa
$\mathbf{x}_7$	6.5	3.0	5.8	2.2	Iris-virginica
$\mathbf{x}_8$	5.8	2.7	5.1	1.9	Iris-virginica
⋮	⋮	⋮	⋮	⋮	⋮
$\mathbf{x}_{149}$	7.7	3.8	6.7	2.2	Iris-virginica
$\mathbf{x}_{150}$	5.1	3.4	1.5	0.2	Iris-setosa

[1]

Attribute

- Die gemessenen Variablen X_1 bis X_4 sind metrische Datenvariablen, die Variable $X_5{=}y$ ist eine kategorische Variable!
- Die gemessenen Variablen X_1 bis X_4 (also Sensoren) nennt man **Attribute**, da sie Eigenschaften und beschreibende Variablen der Zielvariablen y sind

3.6. Sensoren

Welche Sensoren und Messdaten kennt ihr:

- Messtechnik
  - Physikalische Größen wie Temperatur, Dehnung, Spannung, Zeit
  - Fusionierte Umfragevariablen (z.B. Ensemblemittelwerte)
- Bei der Messung mit Sensoren unterscheidet man:
  - Einmalige bzw. einzelne Messungen (single shot)
  - Wiederholte Messungen der gleichen physikalischen Größe (Mittelwertbildung..)

- ❏ Serien von Messwerten, vor allem zeitaufgelöste Datenserien: $\mathbf{D} = \{d_1,d_2,..,d_n\}$, wobei i.A. $\Delta t(d_i,d_{i+1})$=constant

3.7. Sensormodell

- ➤ Ein Sensor ist ein Messwandler, auch in der Soziologie (Indikator für eine Eigenschaft die nicht direkt messbar ist)
- ➤ Ein Sensor bildet daher eine i.A. physikalische Größe x auf eine andere Größe y ab:

$$S(x) : x \to y, K : correct(x \to y)$$

- ➤ Es gibt i.A. eine Kalibrierungsfunktion $K(f, x, y)$
- ➤ Beispiele: Druck → Spannung, Strahlung → Strom, usw.

3.8. Sensordaten

- ➤ Sensoren S sind Datenquellen d von physikalischen, soziologischen oder sonstigen natürlichen nicht direkt erfassbaren Größen x
- ➤ Die Datenwerte (numerisch) werden in einem definierbaren Intervall liegen
  - ❏ Die Kenntnis des Werteintervalls ist wichtig für spätere Datenverarbeitung, Analyse, und Maschinelles Lernen!
  - ❏ Kategorische Werte werden ebenfalls durch eine Menge definiert

$$S(x) : x \to d$$
$$d \in [a, b] \Rightarrow \{v_0, v_1, .., v_i\}$$

4. Mess- und Sensorische Systeme

Der Ursprung der Daten für Analyse und Maschinelles Lernen!
Ein Sensor kommt selten allein.

4.1. Sensoraggregation

Sensorklassen

Physische Sensoren

Physische Sensoren messen direkt eine Größe mit einem Messinstrument (kann auch die Auswertung einer Frage in einem Fragebogen sein), Smartphone

Virtuelle Sensoren

Verwenden Daten (von physischen und anderen virtuellen Sensoren) um neue sensorische Werte zu berechnen (kein Messinstrument) → **Aggregatoren**!!

Schichtenmodell von Sensorischen Systemen

In sensorischen Systemen werden Sensordaten in verschiedenen Ebenen verarbeitet:

- **Vertikale Ebenen** repräsentieren die sensorischen Domainen und die Sensorklassen;
- **Horizontale Ebenen** repräsentieren die Datenverarbeitung.

Vertikale Ebenen

Perzeption

Hier findet die Akquisition der rohen Sensordaten statt. Die Sensoren sind räumlich verteilt und werden lokal vorverarbeitet.

Aggregation

Einzelne Sensordaten werden zeitlich und räumlich zusammengeführt und gesammelt (Sensorfusion)

Applikation

Die gesammelten Daten werden nutzbar gemacht: Weitere Datenverarbeitung, Aufbereitung, Eigenschaftsselektion, Informationsgewinnung, Visualisierung

Horizontale Ebenen

- Die horizontalen Ebenen durchziehen alle vertikalen Ebenen:

1. Sicherheit
2. Datenverarbeitung
3. Kommunikation
4. Datenspeicherung
5. Nachrichtenvermittlung
6. Management

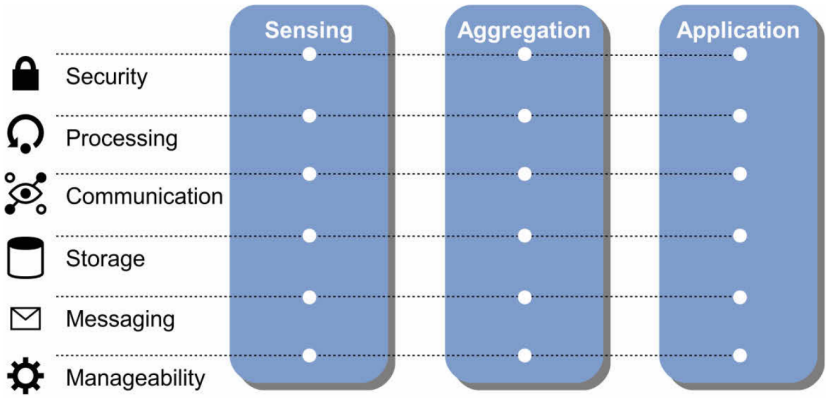


Abb. 4. Grundlegender Zusammenhang der horizontalen und vertikalen Ebenen in Sensorischen Systemen

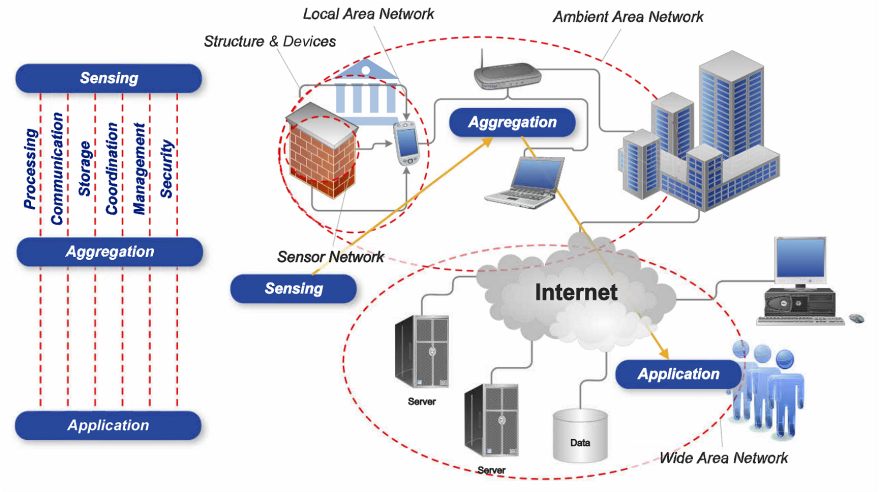


Abb. 5. Räumliche Abbildung der vertikalen Ebenen auf Cloud Computing

4.2. Sensoren in den Ebenen

Perzeption

Vorwiegend physische Sensoren

Aggregation

Virtuelle Sensoren, Datenreduktion (Größe und Dimensionalität)

Applikation

Datenanalyse und Modellbildung, Inferenz von Information, Maschinelles Lernen

4.3. Messfehler und Vertrauen

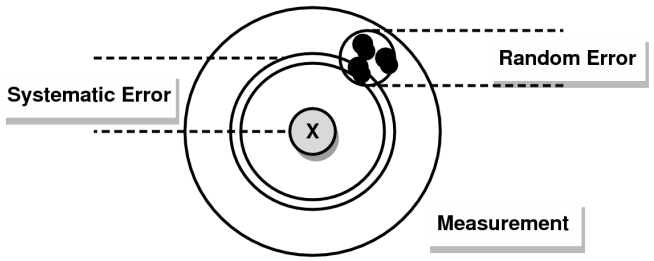
- Die Messgrößen können statisch (zeitlich konstant) oder dynamisch (zeitlich veränderlich) sein. Die Wandlung dieser Messgrößen ergeben dann entsprechend Gleich- und Wechselsignale.
- Auch eine prinzipiell zeitlich unveränderliche Messgröße (bezogen auf die Messung in einem vorgegeben Zeitinterval) erzeugt kein konstantes Signal. Ursache: Rauschen
- Wiederholt man daher eine Messung N-mal unter gleichen Bedingungen, so wird man eine Reihe von verschiedenen Messwerten $\{s_1, s_2, \ldots, s_n\}$ erhalten.
- Es gibt systematische und zufällige Fehler bei der Messung, die sich überlagern.

Systematische Abweichung (systematischer Fehler)

- Abweichung wird durch den Sensor verursacht
- z.B.: falsche Eichung, dauernd vorhandene Störungen wie Reibung
- lässt sich nur durch sorgfältiges Untersuchen der Fehlerquelle beseitigen

Zufällige Abweichung (zufälliger oder statistischer Fehler)

- Abweichung wird durch unvermeidbare, regellose Störungen verursacht
- bei wiederholter Messung weichen Einzelergebnisse voneinander ab
- Einzelergebnisse schwanken um einen Mittelwert



[9]

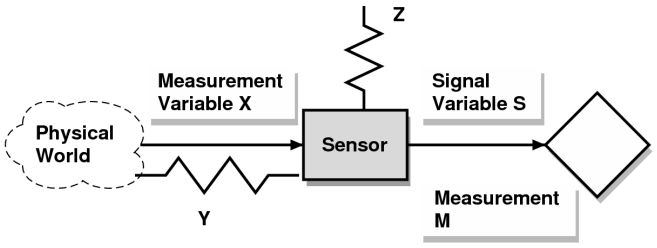
Abb. 6. Offset und Präzision bei der Messung einer Variable X

Systematische Fehler

- Eine Messgröße X ist meistens durch störende Messgrößen Y,Z,... usw. überlagert:

$$K(X,Y,Z) \; : \; X \times Y \times Z \rightarrow S, K(x,y,z) \approx \sum_{n=0}^{m} a_n x^n + \sum_{n=0}^{m} b_n y^n + \sum_{n=0}^{m} c_n z^n$$

- So kann z.B. bei einer Messung einer Kraft oder einer Dehnung die umgebende Temperatur T oder Strukturschwingungen Einfluss auf den Sensor und dessen Übertragungsfunktion und somit auf das Messsignal S haben.

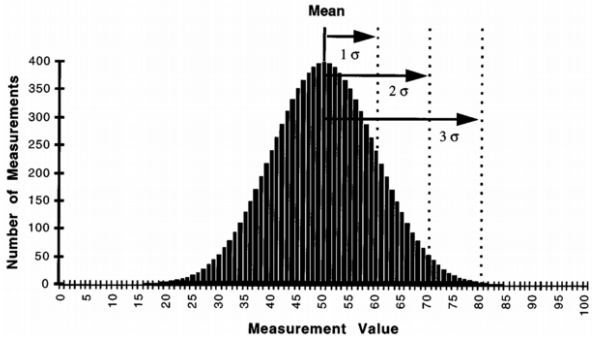


[9]

- Systematische Fehler verfälschen die Kalibrierungsfunktion (z. B. bei Geraden den Offset und Steigung). Sind sie bekannt, können sie kompensiert (rausgerechnet) werden.
- Systematische Fehler können aber auch während der Signalverarbeitung entstehen, so z. B. Offsetspannungen und zeitlicher Drift von Parametern (Verstärkungsfaktor).

Zufällige Fehler - Streuung

- Zufällige Fehler beeinflussen die Genauigkeit einer Messung (Rauschen).
- Wiederholt man eine Messung einer Größe X die durch reine zufälligen Fehler verfälscht wird, so ist die Häufigkeitsverteilung der Messwerte $S=\{s_1,s_2,...,s_n\}$ um einen Mittelwert $\bar{S}$ durch eine Gaussverteilung gegeben (dabei muss die Anzahl der Messungen N groß sein).



[9]

Abb. 7. Häufigkeitsverteilung nach Gauss von Messwerten um einen Mittelwert

- Der Mittelwert S repräsentiert die Abschätzung des wahren/wirklichen Wertes Σ der Messgröße X (oder S):

$$\bar{S} = \frac{1}{N} \sum_{i=1}^{N} s_i$$

- Die Standardabweichung ist ein Maß für die Zuverlässigkeit (Präzision) der einzelnen Messwerte einer Messreihe $\{s_1,s_2,...,s_n\}$:

$$\sigma = \sqrt{\frac{1}{N-1} \sum_{i=1}^{N} (s_i - \bar{S})^2}$$

| Eine Vergrößerung der Anzahl N der Messungen (unter gleichen Bedingungen!) führt zu einer Verbesserung des Mittelwertes $\bar{S}$ (Grenzfall $N \to \infty$), nicht aber zu einer wesentlichen Verkleinerung der Standardabweichung , da die Genauigkeit nicht steigt!

➤ Der wirkliche Mittelwert Σ ist nicht bekannt (nur im Grenzfall $N \to \infty$ ist $\bar{S}$=Σ) - Es gibt aber ein Vertrauensintervall mit einer Wahrscheinlichkeit P dass dieser darin enthalten ist:

Σ $[\bar{S}\text{-} , \bar{S}+]$ mit 68.3%

Σ $[\bar{S}\text{-}2 , \bar{S}+2]$ mit 95.4%

Σ $[\bar{S}\text{-}3 , \bar{S}+3]$ mit 99.73%

➤ Auch in der Soziologie!

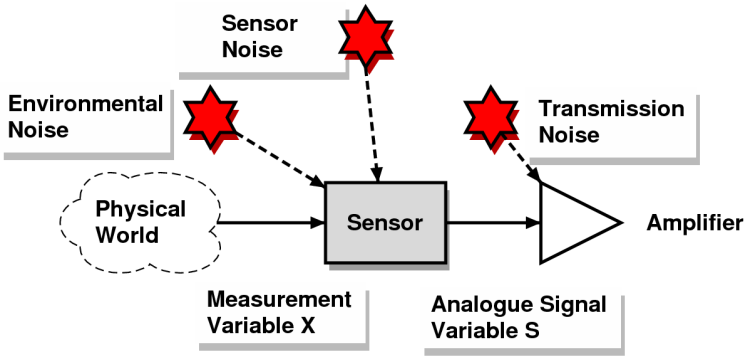


Abb. 8. Rauschquellen bei einer Messung

4.4. Beispiele

/webwork

Plot

```
Plot(Array.random(200),{width:'70%',size:20})
```

Analysis

```
Math.statistics.analysis(Array.random(200))
```

5. Datenanalyse und Eigenschaftsselektion

Häufig sind die rohen sensorischen Daten(variablen) zu hochdimensional und abhängig voneinander
Reduktion auf wesentliche Merkmale kann ML Qualität deutlich verbessern
Häufig besitzen einzelne Sensorvariablen keine oder nur geringe Aussagekraft (geringe Entscheidbarkeitsqualität)

5.1. Datenqualität

➤ Die Daten $\boldsymbol{D}$ werden durch vier wesentliche Eigenschaften beschrieben, die auch mit statistischer Analyse quantifiziert werden können:

Rauschen. Rauschen ist die Verzerrung der Daten. Diese Verzerrung muss entfernt oder Ihre nachteiligen Auswirkungen vermindert werden, bevor ML Algorithmen ausgeführt werden, da die Leistung und Qualität der Algorithmen beeinträchtigen kwerden ann.

Es gibt eine Vielzahl von Filteralgorithmen um den Effekt von Rauschen zu vermindern.

Ausreißer. Ausreißer sind Instanzen, die sich erheblich von anderen Instanzen im Datensatz unterscheiden. Beispiel: Durchschnittliche Anzahl der Follower von Nutzern auf Twitter. Eine Berühmtheit mit vielen Followern kann die Durchschnittliche Anzahl von Followern pro Person leicht verzerren. Da die prominenten Ausreißer sind, müssen Sie aus der Gruppe der Personen entfernt werden, um die Durchschnittliche Anzahl der Follower genau zu Messen.

Aber: Ausreißer können in besonderen Fällen nützliche Muster darstellen und die Entscheidung, sie zu entfernen, hängt vom Kontext und Fragestellung ab.

Fehlende Werte. Fehlende Werte sind Funktionswerte, die in Instanzen fehlen. Zum Beispiel, Einzelpersonen können es vermeiden, Profilinformationen auf social-media-Websites zu melden, wie Ihr Alter, Standort, oder Hobbys. Um dieses Problem zu lösen, können wir (1) Instanzen mit fehlenden Werten entfernen, (2) fehlende Werte schätzen (Z. B. durch den gängigsten Wert ersetzen) oder (3) fehlende Werte ignorieren, wenn data mining-algorithmen ausgeführt werden.

Duplikate. Doppelte Daten treten auf, wenn mehrere Instanzen mit genau denselben Funktionswerten vorhanden sind. Doppelte blog-posts, doppelte tweets oder Profile auf Social-media-Websites mit doppelten Informationen sind Beispiele für dieses Phänomen. Je nach Kontext können diese Instanzen entweder entfernt oder beibehalten werden. Wenn Instanzen beispielsweise eindeutig sein müssen, sollten doppelte Instanzen entfernt werden.

5.2. Statistische Analyse

- ➤ Statistische Analysen von Mess- und Sensordaten können neue Datenvariablen erzeugen und Informationen über die Daten liefern:
  - ❑ Eigenschaftsselektion (Feature Selection) für ML und Informationsgewinnung
  - ❑ Variablentransformation mit Datenreduktion
- ➤ Statistische Analyse liefert eine Reihe von Kennzahlen über Datenvariablen, das können Eigenschaften für die Weiterverabeitung sein:

$$stat(\tilde{x}) : \tilde{x} \rightarrow \tilde{p},$$
$$\tilde{p} = \{mean, \sigma, ..\}$$

Welche statistische Größen gibt es? Was können statistische Größen über Daten aussagen?

5.3. Statistische Funktionen

Peak amplitude (y_{peak})	$y_{peak} = \max \lvert y_i \rvert$
Mean ($\overline{y}$)	$\overline{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$
Mean square ($\overline{y}_{sq}$)	$\overline{y}_{sq} = \frac{1}{n}\sum_{i=1}^{n} (y_i)^2$
Root-mean-square (rms)	$rms = \sqrt{\frac{1}{n}\sum_{i=1}^{n} y_i^2}$
Variance (σ^2)[a]	$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n} (y_i - \overline{y})^2$

[2:173]

Standard deviation (σ)[a]	$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n} (y_i - \overline{y})^2}$
Skewness (dimensionless) (γ)[a]	$\gamma = \frac{\frac{1}{n}\sum_{i=1}^{n} (y_i - \overline{y})^3}{\sigma^3}$
Kurtosis (dimensionless) (κ)[a]	$\kappa = \frac{\frac{1}{n}\sum_{i=1}^{n} (y_i - \overline{y})^4}{\sigma^4}$
Crest factor (X_{cf})	$X_{CF} = y_{peak}/rms$
K-factor (X_k)	$X_{CF} = (y_{peak})(rms)$

[2:173]

5.4. Korrelation von Datenvariablen

- Variablen **X** sollten möglichst (linear) unabängig sein,
  - Um eine geeignete, robuste und genaue Modellsynthese (also ML) zu ermöglichen, d.h.
  - Es sollte möglichst keine Zusammenhänge der Form *correlation*(X_i,X_j) geben!
  - Um den Modellsyntheseprozess zu beschleunigen (also das Training); Rechenzeit reduzieren
  - Um Modelle klein und kompakt zu halten
- Abhängige Variablen sollten identifiziert und in unabhängige "transformiert" werden!

Beispiel Prozessanalyse

- Eine Datentabelle ***D*** mit experimentellen Messgrößen und Fertigungsparametern (Prozessparameter) von additiv gefertigten Bauteilen hatte zunächst 7 Variablen (numerisch):
  - X_1: Hatchabstand [mm]
  - X_2: Scangeschwindigkeit [mm/s]
  - X_3: Laserleistung [W]
  - X_4: Schichtstärke [mm]
  - X_5: Volumenenergiedichte [J/mm^3]
  - X_6: Bauplatten Position x
  - X_7: Bauplatten Position y
  - Y_1: Dichte (%)
- ***D*** bestand aus 61 Experimenten mit unterschiedlichen Fertigungsserien
- Mit einer Principle Component Analysis (PCA) konnte die ganze Tabelle auf die Variablen PC_1 und Y_1reduziert werden!
  - Die Genauigkeit der mit ML synthetitisierten Funktion $M(\boldsymbol{X})$: $\boldsymbol{X} \rightarrow$ *Dichte* konnte ohne signifikanten Genauigkeitsverlust nur aus PC_1 abgeleitet werden, d.h. $M(PC1)$: $PC1 \rightarrow$ *Dichte*

- ❏ Aber: Für die Inferenz (Applikation) von M muss die PCA für die Eingabedaten $\boldsymbol{X}$ wiederholt werden bzw. die Datentransformation durchgeführt werden!

5.5. Analyse Kategorischer Variablen

➤ Die Analyse von kategorischen Variablen vereint die Konzepte:

- ❏ Mengenlehre
- ❏ Kodierung/Dekodierung
- ❏ Verteilung (Wahrscheinlichkeit des Auftretens)

Attribut	Wertemenge
Aussicht	sonnig, regnerisch, bewölkt
Temperatur	kalt, mild, heiß
Luftfeuchtigkeit	hoch, normal
Windig?	ja, nein

[5]

Messdaten

Beispiel	Aussicht	Temperatur	Luftfeuchtigk.	Windig?	Klasse
1	sonnig	heiß	hoch	nein	N
2	sonnig	heiß	hoch	ja	N
3	bewölkt	heiß	hoch	nein	P
4	regnerisch	mild	hoch	nein	P
5	regnerisch	kalt	normal	nein	P
6	regnerisch	kalt	normal	ja	N
7	bewölkt	kalt	normal	ja	P
8	sonnig	mild	hoch	nein	N
9	sonnig	kalt	normal	nein	P
10	regnerisch	mild	normal	nein	P
11	sonnig	mild	normal	ja	P
12	bewölkt	mild	hoch	ja	P
13	bewölkt	heiß	normal	nein	P
14	regnerisch	mild	hoch	ja	N

[5:53]

Abb. 9. Beispiel einer rein kategorischen Datentabelle ***D***. Die Zielvariable *Klasse* mit den Werten {N,P} ist ebenfalls kategorisch, z.B. Klasse=P Sportliche Aktivität

Gemischte Variablenklassen

Outlook	Temperature	Humidity	Windy	Play-time
Sunny	85	85	False	5
Sunny	80	90	True	0
Overcast	83	86	False	55
Rainy	70	96	False	40
Rainy	68	80	False	65
Rainy	65	70	True	45
Overcast	64	65	True	60
Sunny	72	95	False	0
Sunny	69	70	False	70
Rainy	75	80	False	45
Sunny	75	70	True	50
Overcast	72	90	True	55
Overcast	81	75	False	75
Rainy	71	91	True	10

[7:47]

Abb. 10. Einige Datenvariablen wurden mit numerischen/metrischen Werten ersetzt (Klasse → Play-time)

Kann aus der vorherigen Datentabelle mit numerischen Variablen noch ein Zusammenhang aus $\boldsymbol{X}$ *zu* Y *hergestellt werden?*
Reicht die Anzahl der Experimente im Vergleich zu der rein kategorischen Datentabelle?
Wo liegen die Probleme?

Weiteres Beispiel

Age	Spectacle Prescription	Astigmatism	Tear Production Rate	Recommended Lenses
Young	Myope	No	Reduced	None
Young	Myope	No	Normal	Soft
Young	Myope	Yes	Reduced	None
Young	Myope	Yes	Normal	Hard
Young	Hypermetrope	No	Reduced	None
Young	Hypermetrope	No	Normal	Soft
Young	Hypermetrope	Yes	Reduced	None
Young	Hypermetrope	Yes	Normal	Hard
Prepresbyopic	Myope	No	Reduced	None
Prepresbyopic	Myope	No	Normal	Soft
Prepresbyopic	Myope	Yes	Reduced	None
Prepresbyopic	Myope	Yes	Normal	Hard
Prepresbyopic	Hypermetrope	No	Reduced	None
Prepresbyopic	Hypermetrope	No	Normal	Soft
Prepresbyopic	Hypermetrope	Yes	Reduced	None
Prepresbyopic	Hypermetrope	Yes	Normal	None
Presbyopic	Myope	No	Reduced	None
Presbyopic	Myope	No	Normal	None
Presbyopic	Myope	Yes	Reduced	None
Presbyopic	Myope	Yes	Normal	Hard
Presbyopic	Hypermetrope	No	Reduced	None
Presbyopic	Hypermetrope	No	Normal	Soft
Presbyopic	Hypermetrope	Yes	Reduced	None
Presbyopic	Hypermetrope	Yes	Normal	None

[7:7]

5.6. Entropie und Informationsgehalt

- Sensorvariablen können unterschiedlichen *Informationsgehalt* besitzen
  - Nur auf den Dateninhalt (Werte) der Variable X_i bezogen (inherenter Informationsgehalt)
  - Oder zusätzlich bezogen auf die Zielvariable Y (abhängiger Informationsgehalt)
- Der *Informationsgehalt* einer Menge X aus Elementen der Menge C wird durch die *Entropie* $E(X)$ gegeben:

$$E(X) = -\sum_{i=1,k} p_i \log_2(p_i), p_i = \frac{count(c_i, X)}{N}, X = \{c|c \in C\}$$

- Dabie ist k die Anzahl der unterscheidbaren Elemente/Klassen $Val(X) \quad C$ in der Datenmenge X (z.B. die Spalte einer Tabelle) und p_i die Häufigkeit des Auftretens eines Elements $c_\mathrm{i} \quad C$ in X.
- Beispiele:

```
1: X1={A, C, B, C, B, C} → Val(X1)=C={A, B, C},N=6
2:
E(X1)=-(1/6)log(1/6)-(2/6)log(2/6)-(3/6)log(3/6)=1.46
3: X2={A, B, C} → Val(X2)=C={A, B, C},N=3
4:
E(X2)=-(1/3)log(1/3)-(1/3)log(1/3)-(1/3)log(1/3)=1.58
5:
X3={A, A, A, A, B, B} → Val(X3)={A,B}   C={A, B, C},N=6
6:
E(X3)=-(4/6)log(4/6)-(2/6)log(2/6)-(0/6)log(0/6)=0.92
7: X4={A, A, A, A, B, B} → Val(X4)=C={A, B},N=6
8: E(X4)=-(4/6)log(4/6)-(2/6)log(2/6)=0.92
```

- Die Entropie ist Null wenn die Datenmenge X "rein" ist, d.h., nur Elemente einer einzigen Attributklasse $c_1 \quad C$ enthält, z.B. X={A,A,A}.
- Die Entropie ist $log_2(|C|)$ wenn alle Werte gleichverteilt vorkommen, wenn nicht dann kleiner (nicht gleichverteilt).
- Die Entropie reicht allein zur Bewertung des Informationsgehaltes nicht aus:

X1	**X2**	**Y**
A	C	P
B	C	P
A	D	N
B	D	N

- E(X1)=1, E(X2)=1 !! Welche Variable X ist für die Entscheidung der Zielvariable Y geeignet?

5.7. Informationsgewinn (Gain)

- Ansatz: Die Datenmenge Y wird nach den möglichen Werten von X partitioniert, also je eine Partition pro $c_i \quad Val(X)$.

$$G(Y|X) = E(Y) - \sum_{v \in Val(X)} \frac{|Y_v|}{|Y|} E(Y_v)$$

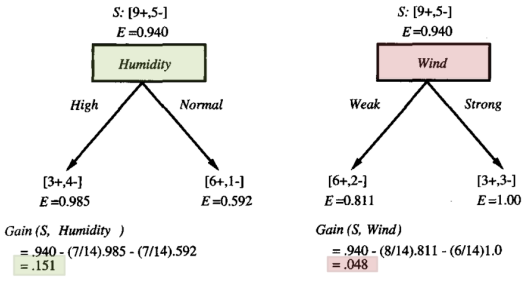
- Die Menge Y_v enthält nur Werte für die X=v ist!
- Ein **Verteilungsvektor** ist dann $Dist(X)$=[$|v_1|$,$|v_2|$,..] und bedeutet wie häufig der bestimmte Wert $v_i \quad Val(X)$ in X auftaucht!

➤ Ein **Verteilungsvektor** ist dann $Dist(Y_v|X)=[|u_1|,|u_2|,..]$ und bedeutet wie häufig der bestimmte Wert $u \quad Val(Y)$ in Y_v auftaucht!

Beispiele

Beispiel	Aussicht	Temperatur	Luftfeuchtigk.	Windig?	Klasse
1	sonnig	heiß	hoch	nein	N
2	sonnig	heiß	hoch	ja	N
3	bewölkt	heiß	hoch	nein	P
4	regnerisch	mild	hoch	nein	P
5	regnerisch	kalt	normal	nein	P
6	regnerisch	kalt	normal	ja	N
7	bewölkt	kalt	normal	ja	P
8	sonnig	mild	hoch	nein	N
9	sonnig	kalt	normal	nein	P
10	regnerisch	mild	normal	nein	P
11	sonnig	mild	normal	ja	P
12	bewölkt	mild	hoch	ja	P
13	bewölkt	heiß	normal	nein	P
14	regnerisch	mild	hoch	ja	N

[12]



5.8. Principle Component Analysis

➤ PCA: Klassische Methode zur unüberwachten linearen Dimensionsreduktion → Analyse der Hauptkomponenten

➤ Reduktion von Redundanz in den Attributen $\boldsymbol{X}$

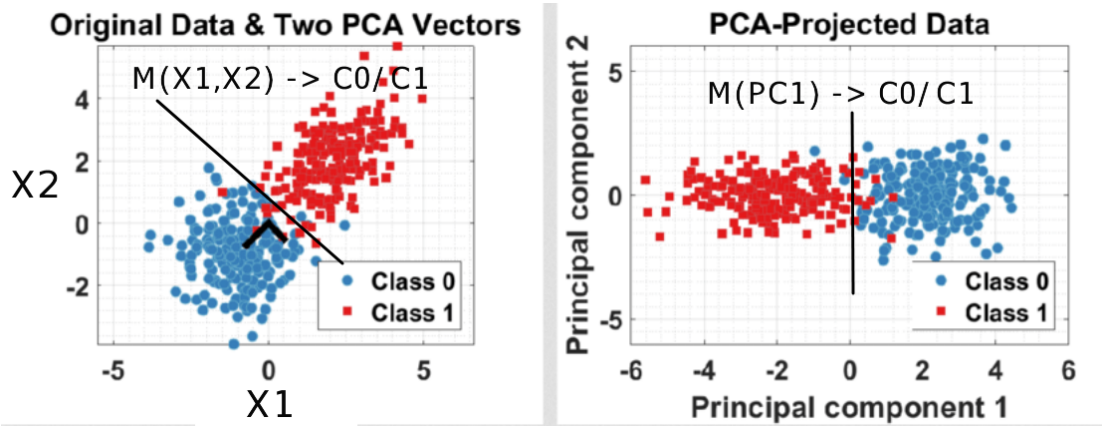
➤ Bessere Trennung bei der Inferenz von kategorischen (und ggfs. auch numerischen) Zielvariablen

➤ Weitere Verfahren:

- ❑ Lineare Diskriminanzanalyse (LDA)
- ❑ Singuläre Wertzerlegung (SVD)

Beispiel

- ***D***=[X_1,X_2,Y], mit $Val(Y)$={Class1,Class2}
- "Rotation" des zweidimensionalen Attributraums führt zu einer reduzierten Datentabelle ***D'***=[PC_1,Y] (PC_2 kann weg gelassen werden)



[Czarnek, RG]

5.9. Spektrale Analyse

Diskerete Fourieranalyse

- (DFT) und schnelle FA (FFT) → (Frequenzspektrum)
  - Spektrum der DFT/FFT hat nur $N/2$ Punkte (Datenreduktion)
  - Aber Problem bei DFT ist: Wenn Anzahl Datenpunkte N klein ist gibt es schlechte Auflösung des Spektrums
  - Vor allem bei "kontinuierlichen" Datenströmen mit fliessenden Fensterverfahren ein Problem (Fensterbreite N_{win} ist klein)
- Es gilt: Das aus der DFT erhaltene Frequenspektrum hat die höchste Frequenz:

$$f_s = 1/\Delta t, \Delta t = t_{i+1} - t_i, f_{max} = f_s/2$$

Diskrete Fouriertransformation

- Eine zeitaufgelöste Variable X wird in eine frequenzaufgelöste Variable F transformiert:

$$DFT(X) : F(k) = \sum_{n=0}^{N-1} X_n e^{-i2\pi nk/N}$$

für k = 0,1,2, … , N–1

- Dabei ist X eine komplexwertige Datenserie { $re(x_\mathrm{i})$,$im(x_\mathrm{i})$ |i=0,1,..,N-1}, und
- Das Ergebnis F ist ebenfalls eine komplexwertige Datenserie { $re(f_\mathrm{i})$,$im(f_\mathrm{i})$ |i=0,1,..,N-1}
- Der Imaginärteil von X wird i.A. auf Null gesetzt (Sensoren sind i.A. reellwertig).

Amplitudenspektrum

$$Mag(X) = \frac{\sqrt{[re(DFT(X))]^2 + [im(DFT(X))]^2}}{N}$$

Phasenspektrum

$$Pha(X) = \arctan\left(\frac{im(DFT(X))}{re(DFT(X))}\right)$$

Leistungsspektrum

$$Power(X) = \frac{[re(DFT(X))]^2 + [im(DFT(X))]^2}{N}$$

5.10. Beispiel für Spektrale Eigenschaftsselektion

Sinus- und Cosinusschwingungen

/webwork

title

SinWaves

```
data1=Array.init(128,Math.sin);
data2=Array.init(128,Math.cos);
data3=Array.init(128,function (x) { return Math.sin(x/2) });
Plot(data1.merge(data2,'c').merge(data3,'c'),{type:'line'});
```

FFT

```
N=128; // data points
fft1=Math.FFT.FFT(N);
re=data1.slice();
im=Array.init(re.length);
fft1.fft1d(re,im);
spectrum=fft1.spectrum(re,im).slice(0,N/2);
Plot(spectrum,{})
print(spectrum.max(true /*position in array*/));
```

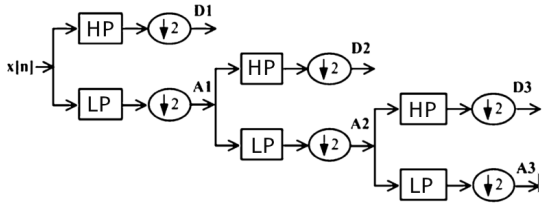
Waveletanalyse

- ➤ Diskrete Wavelet Transformation (DWT)
  - ❏ DFT liefert nur Informationen im Frequenzraum
  - ❏ DWT liefert Informationen aus Zeit- und Frequenzraum
  - ❏ Höherer Informationsgehalt

DWT verwendet lange Zeitfenster für niedrige Frequenzen und kurze Zeitfenster für höhere Frequenzen, was zu einer guten zeitfrequenzanalyse führt.

DWT kann mit digitalen Filterkaskaden aufgebaut werden:

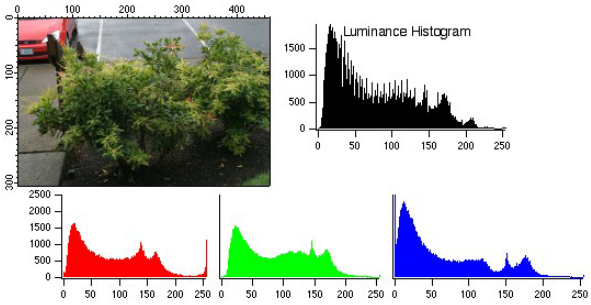
- ➤ Jede Ebene der Filterkaskade besteht aus einem Hoch- und einem Tiefpassfilter
- ➤ Der Hochpassfilter liefert die Details, der Tiefpassfilter die Approximation der DWT auf der n-ten Ebene
- ➤ Die Approximation der Ebene n ist das Eingangssignal für die Ebene $n+1$
- ➤ In jeder Ebene wird der Eingangsvektor $|\mathbf{x}_i|=N$ auf $N/2$ reduziert, d.h. $|\mathbf{x}_{i+1}|=N/2$



5.11. Histogrammanalyse

➤ Ein Histogramm gibt die (intervallbasierte) Verteilung von unterscheidbaren Elementen in einer Datenmenge X an

❑ Beispiel sind Histogramme von Bildern die die Verteilung der Farb-/Grauwerte mit den Werten == Kanälen {0,1,..,255} wiedergeben.

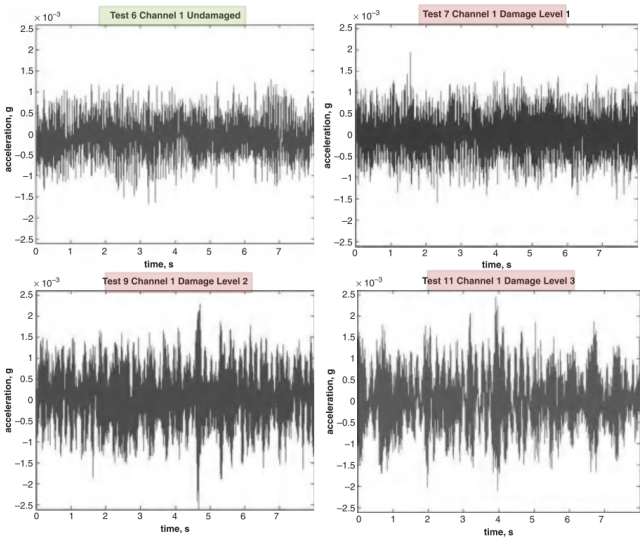


[wavemetrics]

5.12. Analyse von Datenserien

Welcher Sensoren können bei der Bauteilprüfung und Schadensüberwachung Zeit- oder Datenserien erzeugen..

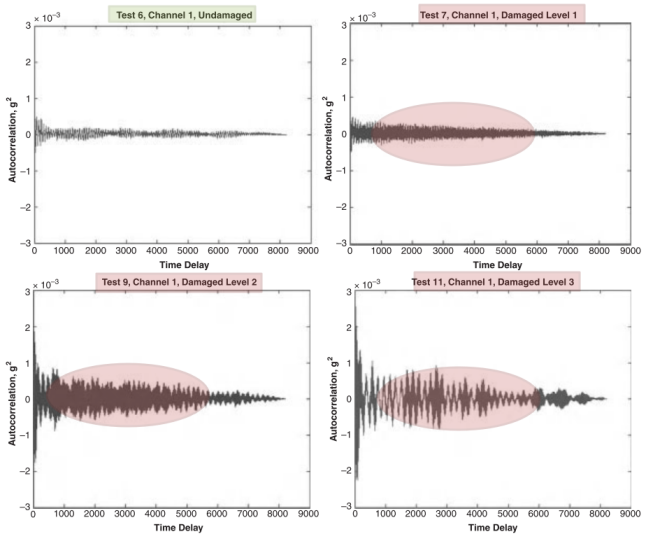
Messdaten



[2:164]

Abb. 11. Zeitaufgelöste Sensordaten $s(t)$ eines Beschleunigungssensors einer Maschine ohne und mit Schäden

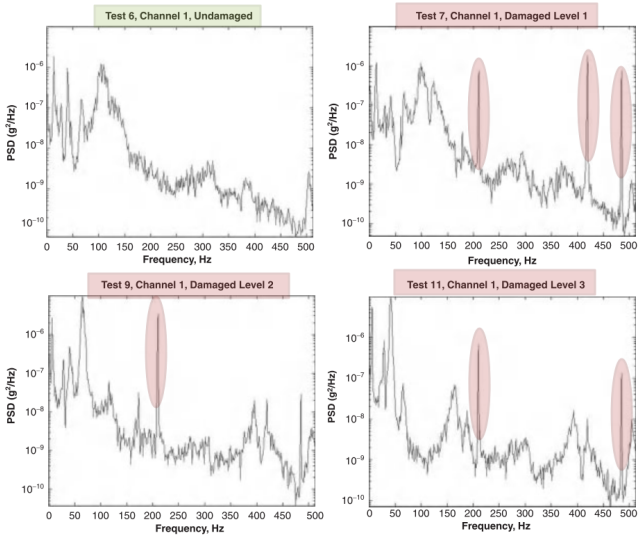
Korrelationsanalyse



[2:164]

Abb. 12. Autokorrlelation der zeitaufgelösten Sensordaten $s(t)$ eines Beschleunigungssensors einer Maschine ohne und mit Schäden

Spektralanalyse



[2:167]

Abb. 13. Spektralanalyse der zeitaufgelösten Sensordaten $s(t)$ eines Beschleunigungssensors einer Maschine ohne und mit Schäden

5.13. Merkmalsselektion

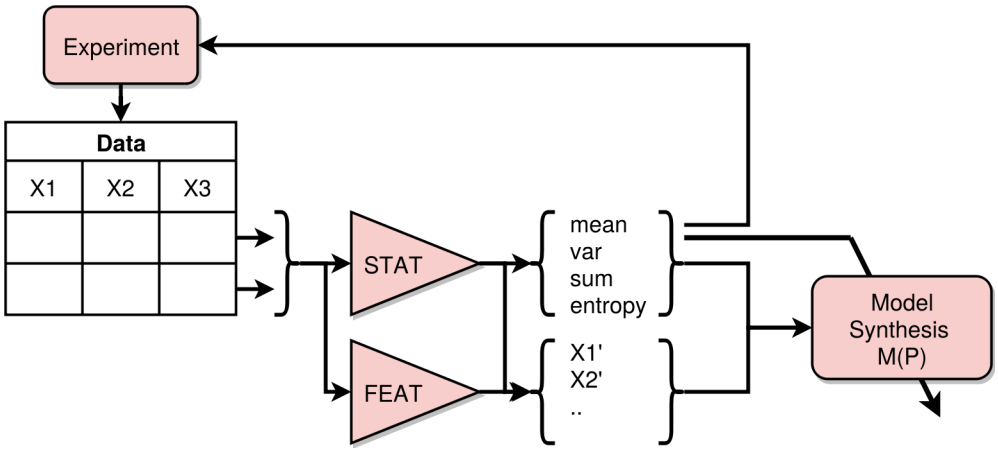


Abb. 14. Die statistische und weitere Analysen können die Eingabe für ML liefern, aber auch die Modellsynthese parametrisieren bzw. beeinflussen

5.14. Zusammenfassung

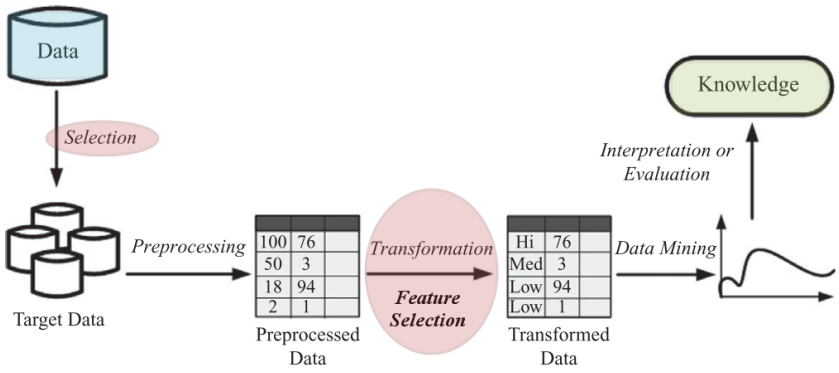
- ➤ Die statistische Analyse von Datentabellen liefert wichtige Informationen über die Qualität der Daten
- ➤ Die Merkmalsselektion transformiert die Rohdaten auf neue möglichst linear unabhängige Attribute
  - ❏ Datenreduktion → Dimensionalität
  - ❏ Datenreduktion → Datengröße
  - ❏ Datenqualitätserhöhung
- ➤ Es werden Verfahren für kategorische und numerische Datenvariablen unterschieden

6. Taxonomie des Maschinellen Lernens

***Zielvariablen**: Kategorische Klassifikation, Numerische Prädiktorfunktionen, Gruppierung*
***Modellfunktionen**: Mit welchen Daten- und Programmarchitekturen können Eingabevariablen auf Zielvariablen abgebildet werden?*
***Training und Algorithmen**: Wie können die Modellfunktionen an das Problem angepasst werden?*
Überwachtes, nicht überwachtes und Agentenlernen

6.1. Datenverarbeitung

- ➤ Die Daten die als Grundlage für die Induktion (Lernen) und die Deduktion (Applikation/Inferenz der Zielvariablen) müssen i.A. vorverabeitet werden → **Merkmalsselektion**



[6]

Abb. 15. Maschinelles Lernen ist ein Werkzeug der Datenanalyse und des Data Minings

6.2. Die Modellfunktion

➤ Die Modellfunktion soll möglichst genau und effizient die Eingabedaten $\boldsymbol{X}$ auf die Zielvariablen $\boldsymbol{Y}$ abbilden:

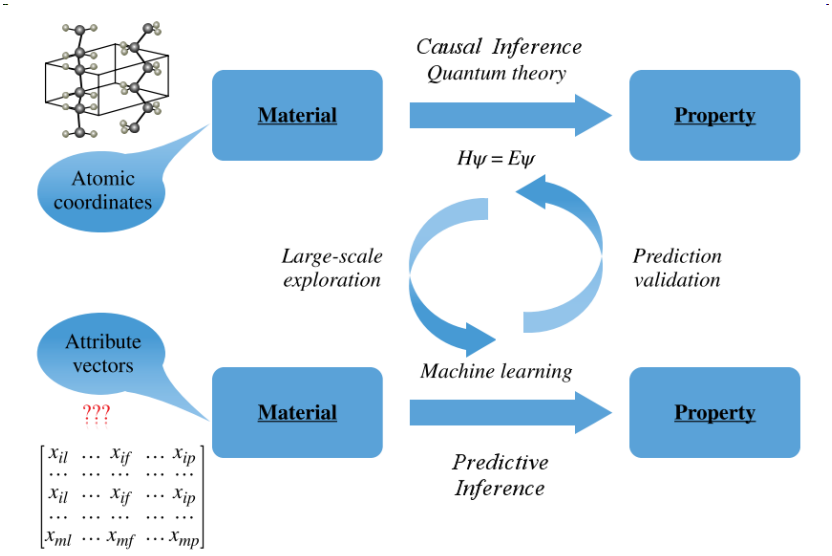
$$M(\tilde{X}) : \tilde{X} \rightarrow \tilde{Y},$$

$$X = \begin{cases} \text{diskrete kategorische Werte} \\ \text{numerische Werte } \mathbb{N},\mathbb{R} \end{cases},$$

$$Y = \begin{cases} \text{diskrete kategorische Werte} \\ \text{numerische Werte } \mathbb{N},\mathbb{R} \\ \text{Gruppen(X), Netzwerke} \end{cases}$$

➤ Die Modellfunktion M **approximiert** eine i.A. nicht bekannte Funktion F, d.h. eine axiomatisch oder analytisch abgeleitete Modellfunktion (z.B. phys. Gesetze) $\rightarrow M$ ist **Hypothese** von F!

Beispiel



[100]

Abb. 16. Kausale vs. Prädiktive Modellbildung und Physikalische Modelle versa algorithmisch bestimmte Modelle (Hypothesen)

6.3. Lernen

Lernen bedeutet die gewünschte Modellfunktion M möglichst genau zu approximieren so dass min error(|Y_0-Y|) für alle (X, Y_0) Paare gilt (Y_0: Referenzdaten).

- ➤ I.a. ist M eine parametrisierbare Funktion $f(\boldsymbol{P})$ oder eine parametrisierbare Datenstruktur
  - ❑ Der Parametersatz $\boldsymbol{P}$=$\{p_1,p_2,..,p_i\}$ bestimmt sowohl Funktion als auch Struktur (z.B. eines Entscheidungsbaumes)
- ➤ Es gibt nicht eine Modellfunktion $\boldsymbol{M}$, sondern eine große Menge möglicher Funktionen, genannt **Hypothesen** .

Lernen bedeutet also die bestmögliche Anpassung des Parametersatzes $\boldsymbol{P}$ *um den Fehler zu minimieren und eine geeignet Hypothesenfunktion zu finden.*

➤ Man unterscheidet bekannte Referenzwerte der Zielvariablen (und Beziehung zu X) Y_0, auch **Labels** genannt, und prognostische Werte Y die als Ergebnis von $M(X)$ geliefert werden (Inferenzwerte), d.h. bei der Applikation ist der wahre Wert Y_t unbekannt (**Schätzung** von Y_t)

$$\begin{aligned} H(\tilde{X}) &: \tilde{X} \rightarrow \tilde{Y}, \\ &= \{M_1^{P1}, M_2^{P2}, .., M_k^{Pk}\}, \\ error(X, Y_0, M) &= |M(X) - Y_0| \end{aligned}$$

Beispiele

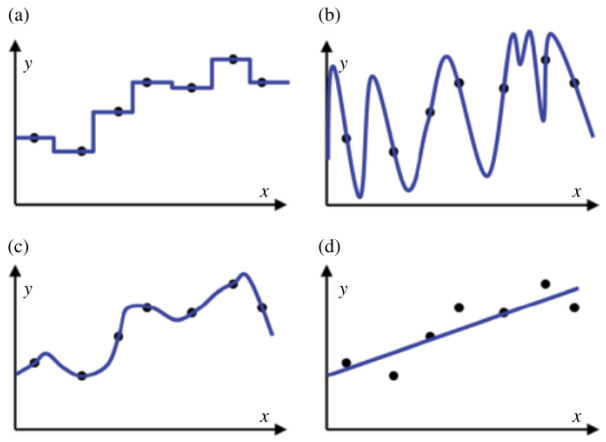
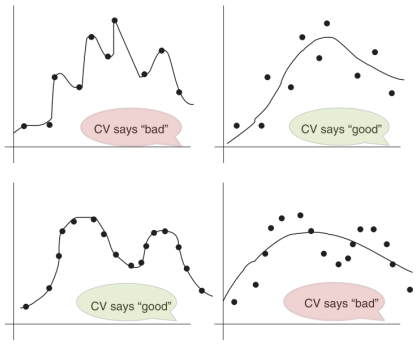


Abb. 17. Verschiedene Modellfunktionen M die die (Trainings) Daten repräsentieren

6.4. Kreuzvalidierung

➤ Beim Training wird ein Inferenzfehler zunächst aus Trainingsdaten bestimmt → Trugschluß!

➤ Stattdessen müssen auch unabhängigige Testdaten für eine Kreuzvalidierung herangezogen werden, und dann …



[13]

6.5. Fehler (Verlust)

Jede Hypothesenfunktion M führt zu einem Informationsverlust durch Appoximation der tatsächlichen und unbekannten Modellfunktione F.

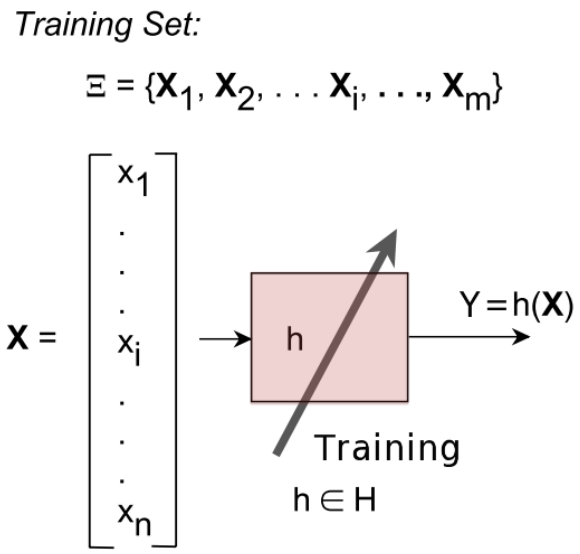
➤ Es gilt also:

$$M(x) : x \rightarrow y = F(x) + E(x)$$

mit E als eine Fehlerfunktion (i.A. zufälliger Fehler) und $\hat{E}$ als mittlerer Prädiktionsfehler.

➤ Die Hypothesenmenge ist also tatsächlich eine Approximation eines unbekannten "exakten" Modells (Modellfunktion) M_{F}, die z.B. mittels physikalischer oder soziologischer Modelle ableitbar wäre.

➤ Genauso wie eine Sensor eine physikalische Größe nur approximieren kann, der tatsächliche Wert der zu messenden Größe ist nicht bekannt



[11]

Abb. 18. Training als Anpassung von Hypothesen für die Abbildungfunktion $X \rightarrow Y$ mit Trainingsdaten

6.6. Parametrisierung

Die Paramater in dem Parametersatz $\boldsymbol{P}$ bestehen aus zwei Klassen:

Statische Parameter $\boldsymbol{P}_{\mathrm{s}}$

Parameter die die Modellimplementierung (Funktion, Datenstruktur, usw.) festlegen und i.A. während des Trainings und der Applikation unverändert bleiben. (Ausnahme: Evolutionäre Algorithmen) → **Konfiguration**

Dynamische Parameter $\boldsymbol{P}_{\mathrm{d}}$

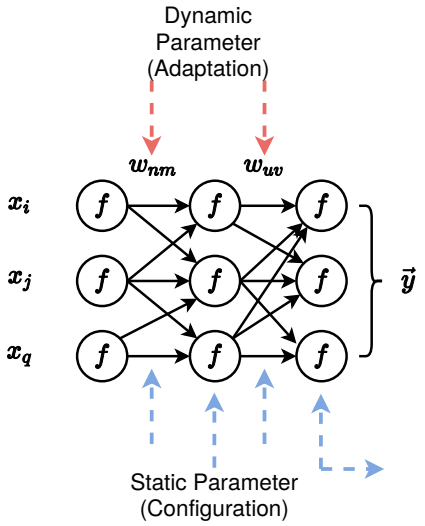
Parameter die während des Trainings verändert (angepasst) werden. Z.B. Funktionsparameter oder Kantengewichte von neuronalen Netzen → **Adaption**

Beispiele

1. Numerische Prädiktorfunktionen (T: Temperatur, S: Satisfaction) → Regression

$$f(T): T \rightarrow S = a + bT + cT^2 + dT^3,$$
$$P_s = \{degr: 3\}, P_d = \{a, b, c, d\}, S = [0, 1]$$
$$f(T): T \rightarrow S = a + bT + cT^c + dT^e,$$
$$P_s = \{terms: 4, lin: 2, exp: 2\}, P_d = \{a, b, c, d, e\}, S = [0, 1]$$

2. Künstliches Neuronales Netzwerk



6.7. Daten

Trainingsdaten $\boldsymbol{D}_{\text{train}}$

Datentabellen die aus Zeilen mit einer bekannten Beziehung (X, Y) bestehen und verwendet werden die Modellfunktion M durch Veränderung von $\boldsymbol{P}$ zu approximieren

Testdaten $\boldsymbol{D}_{\text{test}}$

Datentabellen die aus Zeilen mit einer bekannten Beziehung (X, Y) bestehen und verwendet werden die Modellfunktion M auf Genauigkeit und Fehler zu testen. Man spricht auch von einer Kreuzvalidierung da $\boldsymbol{D}_{\text{test}}$ $\boldsymbol{D}_{\text{train}}$ = sein sollte.

Inferenzdaten $\boldsymbol{D}_{\text{inf}}$

Datentabellen die nur aus Zeilen $\boldsymbol{X}$ bestehen (Y ist unbekannt)

Es gilt: $\boldsymbol{D}_{\text{train}}$ $\boldsymbol{D}_{\text{all}}$, $\boldsymbol{D}_{\text{test}}$ $\boldsymbol{D}_{\text{all}}$, $\boldsymbol{D}_{\text{inf}}$ $\boldsymbol{D}_{\text{all}}$ aber $\boldsymbol{D}_{\text{train}}$ $\boldsymbol{D}_{\text{test}}$ = und $\boldsymbol{D}_{\text{train}}$ $\boldsymbol{D}_{\text{test}}$ $\boldsymbol{D}_{\text{inf}}$ (Idealfall!)

Die großen Probleme beim algorthmischen/trainierten Modellieren:

- ➤ Die Trainingsdaten sind nicht repräsentativ (Umfang, Varianz, Qualität)
- ➤ Die Testdaten sind nicht repräsentativ (Umfang, Varianz, Qualität)
- ➤ Die Trainingsdaten enthalten schwache Variablen die nicht entfernt wurden (Inkonsistenz und geringer Informationsgewinn)

***Generalisierung**. Das gelernte Modell M bildet alle drei Datenmengen gleichermaßen gut ab!*

- ➤ Ergänzung:

Bewertungsdaten

Beim Einsatz eines gelernten Modells kann eine Evaluierung bezüglich Qualität / Genauigkeit stattfinden. Diese Daten können dann ggfs. für eine Adaption des Modells und dessen Parametersatz $\boldsymbol{P}$ verwendet werden.

D.h. bei der Anwendung des Modells können somit auch neue Trainingsdaten gewonnen werden, z.B. im Rahmen eines Produktlebenszyklusmanagements!

6.8. Lernverfahren

Überwachtes Lernen

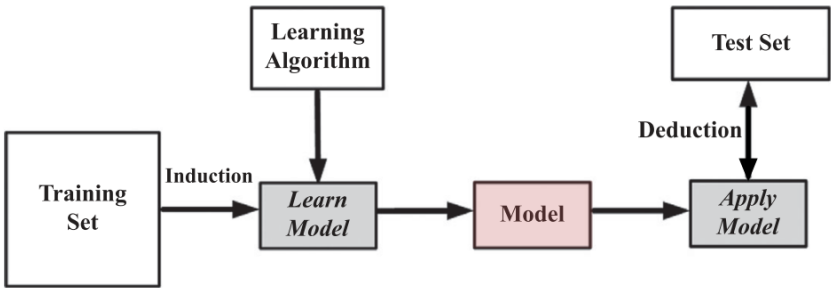
Es gibt Trainingsdaten mit bekannten Beziehungen (X, Y) die verwendet werden um die Modellfunktion mit minimalen Fehler anzupassen. Überwachugn benötigt i.A. einen Experten der die Beziehungen (X, Y) erstellt und analytisch den Fehler bewertet.

Unüberwachtes Lernen

Es gibt Trainingsdaten ohne bekannte beziehung (X, Y), d.h., schon das Lernen führt zu einer automatischen Inferenz der zielvariablen Y, die aber in diesem Fall i.A. nur durch Gruppenmengen bestehen. Eine Gruppenmenge Q=$\{X_i\}$ bringt verschiedene Eingabewerte in Beziehung. D.h. Y .

Belohnungs- und Agentenlernen

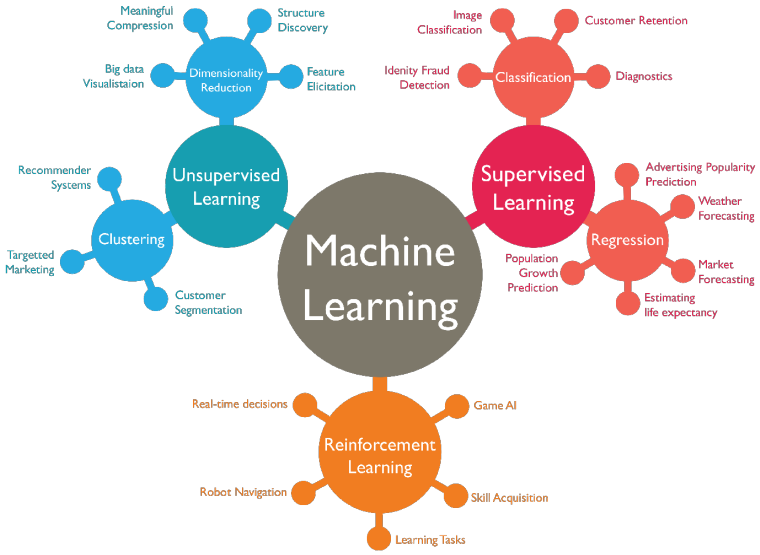
Die Abbildungsfunktion $f(X)$: $X \rightarrow Y$ wird schrittweise durch eine Evaluierung des inferrierten Y mit einem Belohnungswert r=[0,1] gelernt. Training und Inferenz findet gleichzeitig statt.



[6]

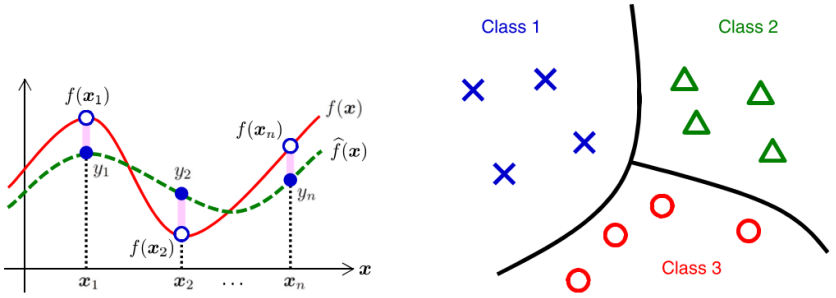
Abb. 19. Ablauf Überwachtes Lernen mit Trainings- (Induktion) und Applikationsphasen (Deduktion)

6.9. Taxonomie der Verfahren



[Abdul Rahid,www.wordstream.com]

6.10. Überwachte Lernverfahren - Unterklassen



(a) Regression (b) Classification

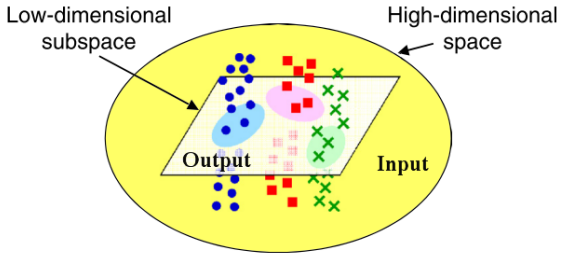
[4]

Abb. 20. Zwei wichtige Unterklassen von überwachten Lernen: Regression (Numerische Zielvariablen) und Klassifikation (Kategorische Zielvariablen)

6.11. Dimensionalitätsreduktion

➤ ML kann auch für die Reduktion von Datendimensionalität eingesetzt werden (Informationen sind reduzierte Daten)

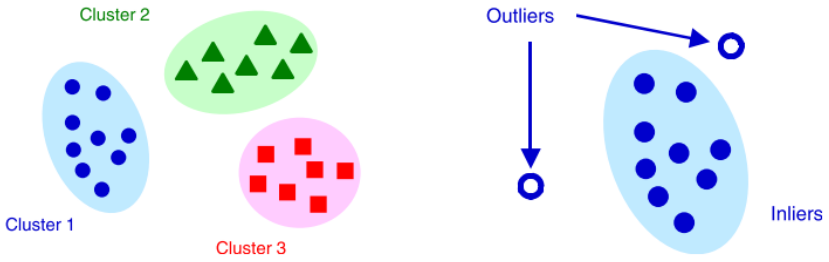
❑ Beispiele: Principle Component Analysis, Single Value Decomposition, ..



[4]

Abb. 21. Abbildung von hochdimensionale Daten X^{n} auf niederdimensionale X^{m} mit $m < n$

6.12. Unüberwachtes Lernen - Unterklassen



(a) Clustering **(b) Ausreißer (Anomalien)**

[4]

Abb. 22. Zwei wichtige Unterklassen von nicht überwachten Lernen: Clustering (Gruppenbildung) und Ausreißerdetektion

6.13. Training

- Das Training einer Modellfunktion M kann
  - **monolitisch** (alle Dateninstanzen werden "parallel" verwendet), oder
  - **stapelbasiert** (d.h. Gruppen von Instanzen werden "parallel" verarbeitet), oder
  - **iterativ** (Dateninstanzen werden "sequenziell" verwendet), und
  - **inkrementell** (iterativ mit neuen Daten).
- Inkrementelle Trainings- und Anpassungsverfahren könnne alte Datensätze verwerfen → Stromldatenlernen!
- Nicht jede Modellimplementierung ist geeignet:
  - Graphen (Bäume) können i.A. nur monolotisch trainiert = erzeugt werden!
  - Regression von math. Funktionen kann monolitisch und/oder iterativ erfolgen;
  - Neuronale Netze können monolitisch, stapelbasiert, iterativ, und vor allem inkrementell trainiert werden.

6.14. Modellimplementierungen

Es gibt im wesentlichen vier verschiedene Architekturen die Modelle M zu implementieren:

Funktionen

Die Struktur einer mathematischen Funktion wird durch ihre Terme gebildet (Berechnungsknoten), z.B. $ax+bx^2$. Zu jedem Term gehört ein dynamischer Parameter der beim Training angepasst wird um den Fehler zu minimieren. Das Ergebnis ist die Zielvariable y.

Gerichtete Graphen

Gerichtete Graphen (oder Entscheidungsbäume) bestehen aus Knoten und Kanten. Die Knoten repräsentieren eine Eingabevariable (Attribute) x $\boldsymbol{X}$. Die Kanten beschreiben die Entwicklung eine Graphens beginnend vom Wurzelknoten hin zu den Blättern. Die Blätter enthalten die Werte der Zielvariable(n) y. Der dynamische Paremetersatz ist der Graph (dessen Struktur).

Funktionale Graphen

Hybrid aus gerichteten Graph und Funktion → Künstliche Neuronale Netze. Die Knoten repräsentieren Berechnungsfunktionen, die Kanten verbinden Ausgänge von Funktionen mit Eingängen. Es gibt Eingangsknoten die mit den Eingabevariablen $\boldsymbol{X}$ verbunden sind, und Ausgangsknoten die mit den Ausgangsvariablen $\boldsymbol{Y}$ verbunden sind.

Ungerichtete Graphen

Hier repräsentieren die Knoten Dateninstanzen X, und die Kanten verbinden die nächsten Nachbarn miteinander. Hier geht es um Gruppenbildung (k nächste Nachbarn/kNN Problem).

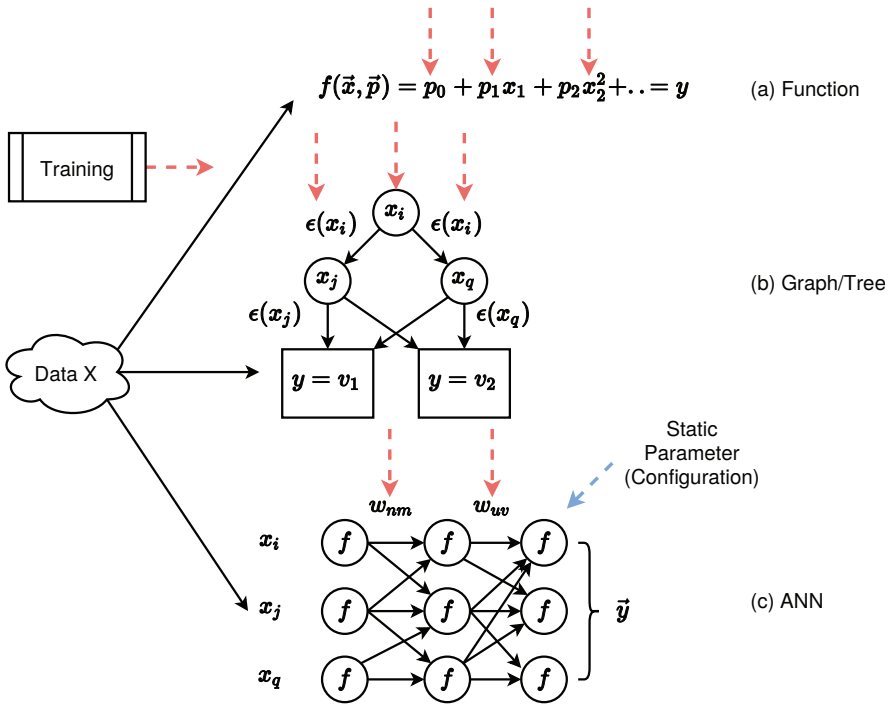


Abb. 23. Verschiedene Modellimplementierungen

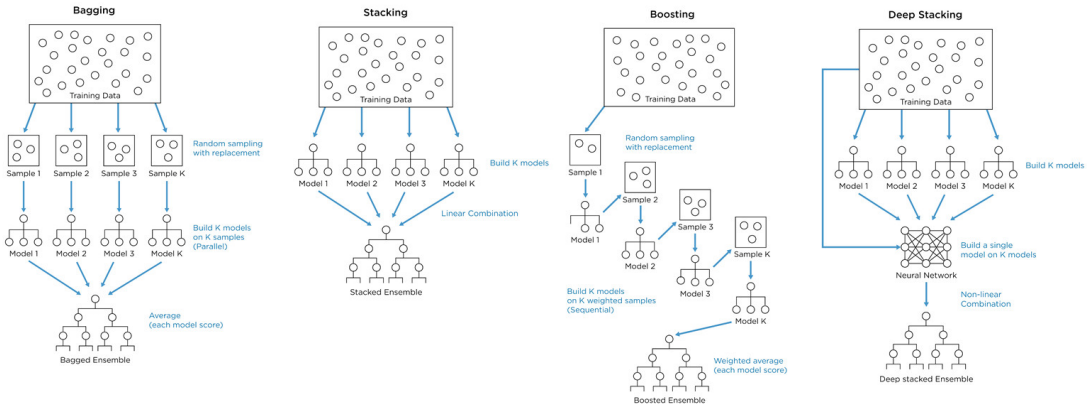
6.15. Hybride Modelle

Multiinstanz Modelle

- Ensemblelernen vereint multiple Modelle (gleicher Klasse oder unterschiedlich)

$$M(\tilde{X}) : \tilde{X} \rightarrow \tilde{Y} = \Phi(\{M_1(X), M_2(X), .., M_n(X)\})$$

- Die einzelnen Modellinstanzen arbeiten mit gleichen oder verschiedenen Dateninstanzen
- Es gibt eine Split- und eine Join Schicht (Kombinierer, Modellfusion)



[Jay Budzik, www.thetalkingmachines.com]

Abb. 24. Verschiedene Architekturen für Multiinstanz Lernen und Inferenz

6.16. Instanzklassifikation

SLSP

Einzelinstanz Lernen (auf allen Daten) und Einzelinstanz Prädiktion (Inferenz auf allen Daten)

SLMP

Einzelinstanz Lernen (auf allen Daten) und replizierte Multiinstanz Prädiktion (Inferenz auf Teildaten mit Modellfusion)

MLSP

Multiinstanz Lernen (auf Teildaten) mit Modellfusion und Einzeninstanz Prädiktion (Inferenz auf allen Daten)

MLMP

Multiinstanz Lernen und Multiinstanz Prädiktion (Modellfusion)

6.17. Ablauf und Phasen von ML

0. Statistische Analyse und Bewertung der Daten
1. Merkmalsselektion
2. Aufteilung der Daten in Trainings- und Testdaten (i.A. randomisiert) $\boldsymbol{D}=\boldsymbol{D}_{\text{train}} \quad \boldsymbol{D}_{\text{test}}$
3. Training einer Modellfunktion M mit bekannten (gelabelten bei ÜL) Trainingsdaten $\boldsymbol{D}_{\text{train}}$ unter Bewertung des Modellfehlers $E(X)$

4. Test und Bewertung von M mit bekannten Daten $\boldsymbol{D}_{\text{test}}$
5. Applikation (Inferenz) von M auf unbekannten Daten $\boldsymbol{D}$

6.18. Zusammenfassung

Maschinelles Lernen besteht aus:

1. Modellimplementierungen:
   - ➤ Funktionen, Gerichtete Graphen, Funktionalen Graphen, Ungerichtete Graphen, also mit/für
   - ➤ Regression, Entscheidungsbäume, Neuronale Netze, Clustering (kNN)
2. Aufgaben
   - ➤ Regression, Klassifikation, Gruppierung (Clustering), Prognostik
3. Methoden und Verfahren
   - ➤ Überwachtes, nicht überwachtes, und rückgekoppeltes Belohnungslernen
   - ➤ Monolitisches, stapelbasiertes, iteratives, und inkrementelles Lernen
   - ➤ Einzel- versa Multiinstanzlernen
   - ➤ Enscheidungsbaumlernen (Konstruktion), Support Vector Machines (Regression), Backpropagation in Neuronalen Netze, usw.
4. ML besteht aus mehreren Phasen:
   - ➤ Datenerhebung, Datenvorverarbeitung, Statistische Bewertung, Merkmalsselektion, Modellertsellung, Training, Test und Analyse (Kreuzvalidierung), Anwendung/Inferenz
5. Daten werden unterteilt in:
   - ➤ Trainingsdaten , Testdaten, Anwendungsdaten
   - ➤ Trainings- und Testdaten bei ÜL mit (x,y) Beziehungen (Labelling)

7. Klassifikation mit Entscheidungsbäumen

***Zielvariablen**: Kategorische Variablen*
***Eigenschaftsvariablen**: Kategorische und Numerische Variablen*
***Modell**: Gerichteter azyklischer Graph (Baumstruktur)*
***Training und Algorithmen**: C4.5, ID3, INN*
***Klasse**: Überwachtes Lernen*

7.1. Entscheidungsbäume

- Ein Entscheidungsbaum ist ein gerichteter azyklischer Graph bestehend aus einer Menge von Knoten $\boldsymbol{N}$ die mit den Eingabevariablen x verknüpft sind und Kanten $\boldsymbol{E}$ die die Knoten verbinden
- Die Endknoten sind Blätter und enthalten Werte der Zielvariablen y (daher kann y nur eine kategorische Variable sein, oder eine intervallkategorisierte)
- Die Kanten bestimmen die Evaluierung des Entscheidungsbaum beginnend von dem Wurzelknoten bis zu einem Blattknoten
  - Jede Kante hat eine Evaluierungsbedingung (x) der Variable des ausgehenden Knotens x
- Zusammengefasst ausgedrückt:

$$
\begin{aligned}
&M(X) : X \rightarrow Y, X = \{x_i\}, Y = \{y_j\} \\
&DT = \langle N_x, N_y, E \rangle \\
&N_x = \{n_i : n_i \leftrightarrow x_j\}, N_y = \{n_i : n_i \leftrightarrow val(y_j)\} \\
&E = \{e_{ij} : n_i \mapsto n_j | \epsilon_{ij}\}
\end{aligned}
$$

- Entscheidungsbäume können neben dem Graphen auch funktional dargestellt werden:

$$
M(X) = \begin{cases}
x_i = v_1, \begin{cases} x_j = v_1, val(y_i) \\ x_j = v_2, val(y_i) \\ x_j = v_3, \{.. \end{cases} \\
x_i = v_2, \begin{cases} x_k = v_1, \{.. \\ x_k = v_2, \{.. \\ x_k = v_3, \{.. \end{cases} \\
x_i = v_3, \begin{cases} x_l = v_1, \{.. \\ x_l = v_2, \{.. \\ x_l = v_3, \{.. \end{cases}
\end{cases}
$$

Baumklassen

Man unterscheidet:

- **Binäre Bäume**. Jeder Knoten hat genau (oder maximal) zwei ausgehende Kanten (Verzweigungen). Der Test der Variable x kann daher nur $x < v$, $x > v$, $x \quad v$, oder $x \quad v$ sein! Wird vor allem bei numerischen Variablen eingesetzt.
- **Bereichs- und Mehrfachbäume**. Jeder Knoten hat $1..k$ ausgehende Kanten (Knotengrad k). Der Test der Variable x kann auf einen bestimmten Wert $x \quad V$ oder auf ein Intervall $[a,b]$ erfolgen! Wird vor allem bei kategorischen Variablen eingesetzt.

Baumstruktur

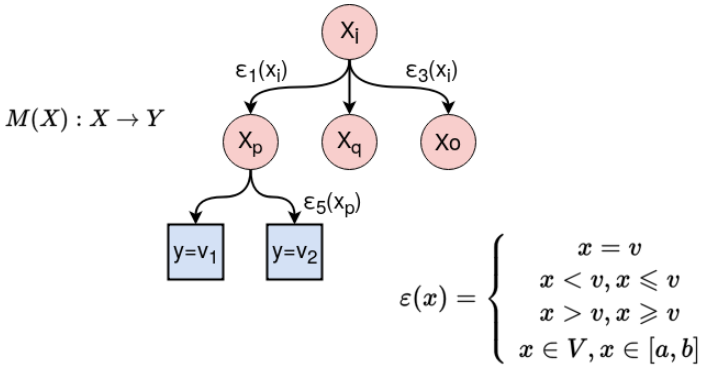


Abb. 25. Grundlegende Struktur eines Entscheidungbaumes

Vorteile

Entscheidungsbäume sind einfach aufgebaut und können mit einfachen Algorithmen erzeugt werden. Entscheidungsbäume als inferriertes Modell erlauben eine **Erklärbarkeit** des Modells, also die Antwort auf die Frage wie sich ein y aus einem x ergibt. Weiterhin ist eine Ableitung eines **inversen Problems** möglich, d.h. welche Werte x für gegebenes y sind möglich?

Nachteile

Entscheidungsbäume können schnell **spezialisieren**, d.h. es fehlt an **Generalisierung**. Theoretisch kann mit einem Entscheidungsbaum jede Trainingsdatentabelle mit einer Trefferquote von 100% abgebildet werden. Der Test mit nicht trainierten Daten ergibt aber Prädiktion in der Größenordnung der Ratewahrscheinlichkeit!

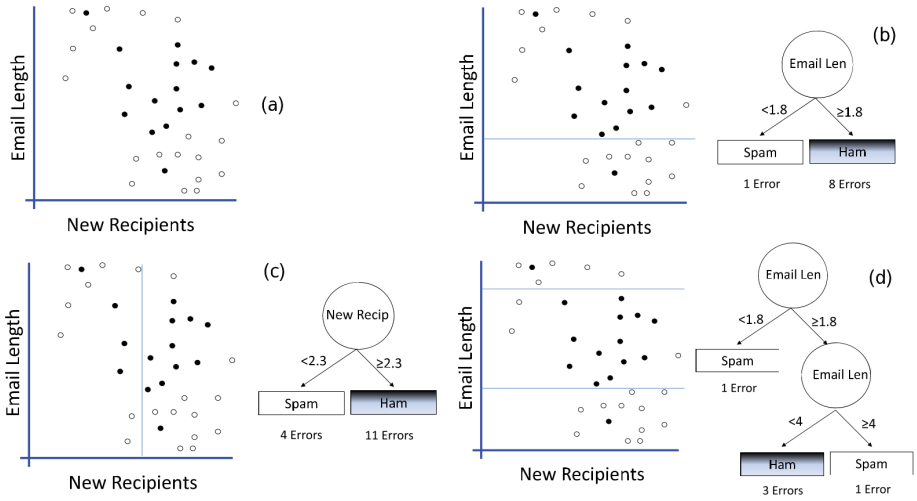
7.2. Training

➤ Das Training mit Trainingsdaten $\boldsymbol{D}_{\text{train}}$ erzeugt den Baum *schrittweise*:

- ❏ Es werden geeignete Variablen x $\boldsymbol{X}$ ausgewählt die einen *Knoten* im Baum erzeugen
- ❏ Jeder hinzugefügte Knoten erzeugt neue Teilbäume (durch Verzweigungen)
- ❏ Die *Verzweigungsbedingungen* (Kanten) werden ebenfalls vom Trainer anhand der Werte der Variable x in Abhängigkeit von der Zielvariablen y gewählt/berechnet.

➤ Die Auswahl der Variablen und die Verzweigungsbedingungen können je nach Algorithmus und Baumklasse variieren!

7.3. Beispiel



[10]

Abb. 26. Schrittweise Erzeugung des Entscheidungsbaums aus den Eingabedaten (a) erst mit einer Variable (b,c), dann mit zwei (d) unter Beachtung des Klassifikationsfehlers

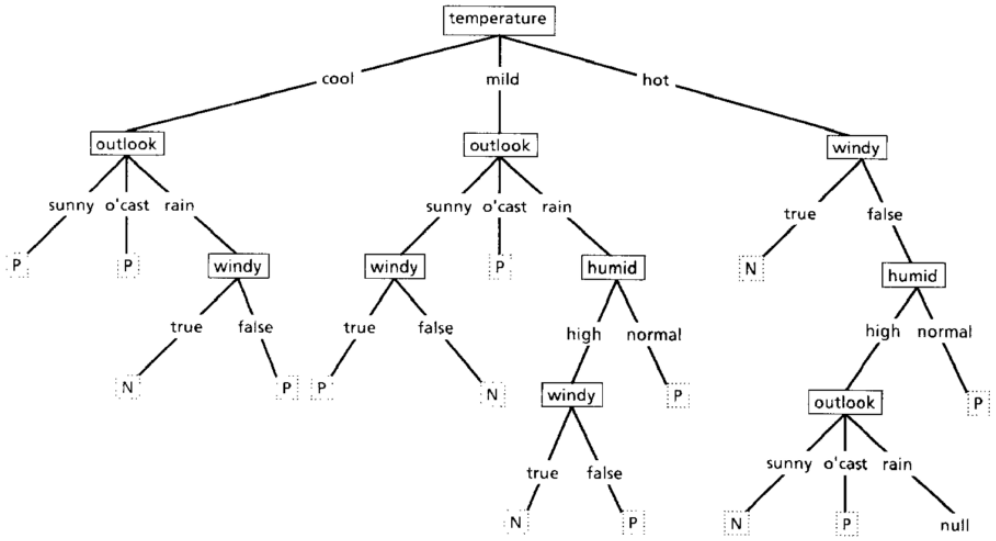
Jeder Knoten in einem binären Baum stellt eine lineare Separation des Eingabedatenraums dar.

Probleme bei Mehrbereichsbäumen

➤ Wenn die Wertemenge $val(x)$ groß ist gibt es entsprechend auch viele Verzweigungen im Baum!

- ❏ Die Größe des Baums wächst an (Speicher)
- ❏ Die Rechenzeit für das Training (Induktion) aber auch die Anwendung (Inferenz, Deduktion) wächst
- ❏ Die Entropie kann als Maß der Varianz der Wertemenge gesehen werden.

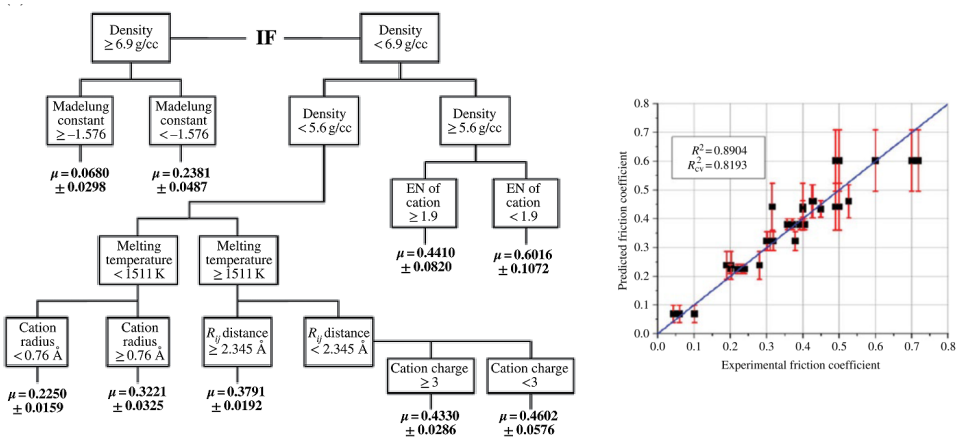
Das "NP" Problembeispiel



[14]

Abb. 27. k-stelliger Entscheidungsbaum für kategorische Variablen

Beispiel Materialeigenschaften



[100]

Abb. 28. (Links) Entscheidungsbaum für die Vorhersage von Reibungskoeffizienten von Materialien auf der Grundlage von sechs grundlegenden Materialmerkmalen (Rechts) Vergleich der vorhergesagten und experimentellen Reibungskoeffizienten

Trainingsalgorithmen

- ➤ Es gibt verschiedene Trainingsverfahren (für verschiedene Baumklassen):
  - ❑ **ID3**. Der Klassiker (Iterative DiChaudomiser 3, Ross Quinlan, 1975-1986) für kategorische Variablen (k-stelliger Baum)
  - ❑ **C4.5**. Der Klassiker (Ross Quinlan 1988-1993) für numerische (und kategorische) Variablen (Binär- und k-stelliger Baum) als Erweiterung des ID3 Verfahrens.
  - ❑ **INN**. Die Eigenkreation (ICE, Stefan Bosse, 2016) für numerische Werte mit Intervallarithmetik für unsichere verrauschte Sensorwerte (also im Prinzip mit Intervallkategorisierung und Kantenbedingungen sind $x \quad [a,b]$), basierend auf C4.5 und ID3

7.4. Vergleich ID3 - C4.5

- ➤ Der ID3-Algorithmus wählt das beste Attribut basierend auf dem Konzept der Entropie und dem Informationsgewinn für die Entwicklung des

Baumes.

➤ Der C4.5-Algorithmus verhält sich ähnlich wie ID3, verbessert jedoch einige ID3-Verhaltensweisen:

- ❏ Möglichkeit, numerische (kont.) Daten zu verarbeiten.
- ❏ Verarbeitung unbekannter (fehlender) Werte
- ❏ Möglichkeit, Attribute mit unterschiedlichen Gewichten zu verwenden.
- ❏ Beschneiden des Baumes nach der Erstellung (**Modellkompaktierung**).
- ❏ Vorhersage der Fehler
- ❏ Hervorhebung und Extraktion von Teilbäumen

7.5. ID3 Verfahren

[1] J. R. Quinlan, "Induction of Decision Trees," in Machine Learning, Kluwer Academic Publishers, Boston, 1986.

➤ Ausgangspunkt für die Konstruktion des Entscheidungsbaums ist die (Shannon) Entropie einer Spalte X der Datentabelle (mit der Variable x):

$$E(X) = -\sum_{i=1,k} p_i \log_2(p_i), p_i = \frac{count(c_i, X)}{N}, X = \{c|c \in C\}$$

➤ Dann der Informationsgewinn einer Spalte X hinsichtlich der Zielvariablenspalte Y:

$$G(Y|X) = E(Y) - \sum_{v \in Val(X)} \frac{|Y_v|}{|Y|} E(Y_v)$$

➤ Der Informationsgewinn, der durch Auswahl des Attributs x und der Spalte X erzielt wird, errechnet sich dann als Differenz der Entropie von Y und der erwarteten/durchschnittlichen Entropie von Y bei Fixierung von x.

7.6. Algorithmus

```
 1:
0. Starte mit leeren Baum, allen Eingansattributen X, der Zielvariablen Y,
 2:    und der vollständigen Datentabelle D(X,Y).
 3:
1. Berechne den Informationsgewinn für jede Attributevariable x   X.
 4:
2. Wenn nicht alle Zeilen zum selben Zielvariablenwert gehören,
 5:
   wird der Datensatz D in Teilmengen D'~xbest,v1~, D'xbest,v(2~~\), usw.
 6:
   aufgeteilt für das Attribut xbest   X mit dem größten Informationsgewinn.
 7:
3. Es wird ein Knoten mit der Attributevariable xbest erstellt.
 8:
4. Wenn alle Zeilen zur selben Klasse gehören, wird ein Blattknoten
 9:    mit dem Wert der Zielvariable erstellt.
10:
5. Wiederholung von 1-4 für die verbleibenden Attribute X'=X / xbest,
11:
   allen Teilbäumen (Verzweigungen von aktuellen Knoten) mit jeweiligen D',
12:    bis alle Attribute verwendet wurden,
13:    oder der Entscheidungsbaum alle Blattknoten enthält.
```

7.7. C4.5 Verfahren

[1] J. R. Quinlan, "C4.5: Programs For Machine Learning". Morgan Kaufmann, 1988.

- Wie ID3 werden die Daten und Attribute an jedem Knoten des Baums bewertet um das beste Teilungsattribut zu bestimmen.
- Aber C4.5 verwendet die Methode der "gain ratio impurity", um das Teilungsattribut zu bewerten (Quinlan, 1993).
- Entscheidungsbäume werden in C4.5 mithilfe eines Satzes von Trainingsdaten oder Datensätzen wie in ID3 erstellt.
- An jedem Knoten des Baums wählt C4.5 ein Attribut der Daten aus, das seinen Satz von Samples am effektivsten in Teilmengen aufteilt, die in der einen oder anderen Klasse verteilt sind.
- Das Kriterium ist der **normalisierte Informationsgewinn**:

- ❏ Verhältnis des Informationsgewinns G (Gain) zu einer sog. Teilungsqualität (Split Info SI), die sich aus der Zielvariable Y zum Aufteilen nach den Y Werten der Daten ergibt.
- ❏ Das Attribut mit dem höchsten Verhältnis GR (Gain Ratio) wird ausgewählt, um die Entscheidung für die Teilung zu treffen.

$$G(Y|X) = E(Y) - \sum_{v \in Val(X)} \frac{|Y_v|}{|Y|} E(Y_v)$$

$$SI(Y) = \sum_{c \in Val(Y)} -\frac{|Y_c|}{|Y|} \log_2 \frac{|Y_c|}{|Y|}$$

$$GR = \frac{G(Y|X)}{SI(Y)}$$

7.8. Teilung von kategorischen und numerischen Variablen

- ➤ Bei kategorischen Variablen bestimmen die Werte $Val(X)$ einer Spalte der Datentabelle einer Variablen x die Aufteilung eines Entscheidungbaums (**Partitionierung**).
- ➤ Bei nuemrischen Variablen muss ein Wert als Teilungspunkt aus der Werteverteilung bestimmt!
  - ❏ Nicht trivial; Welches Kriterium?
  - ❏ Intervallkategorisierung und Wertepartitionierung kann helfen!
  - ❏ D.h. mit intervallkategorisierten diskrete Werter wird die Spalte X entsprechend der Zielvariable Y partitioniert.
  - ❏ Und diese Partitionen werden bewertet und der Teilungspunkt x_{split} X bestimmt (z.B. über Mittelwerte der Intervalle)

Vertiefung

L. Rokach and O. Maimon, Data Mining with Decision Trees - Theory and Applications. World Scientific Publishing, 2015.

7.9. Intervallkodierung

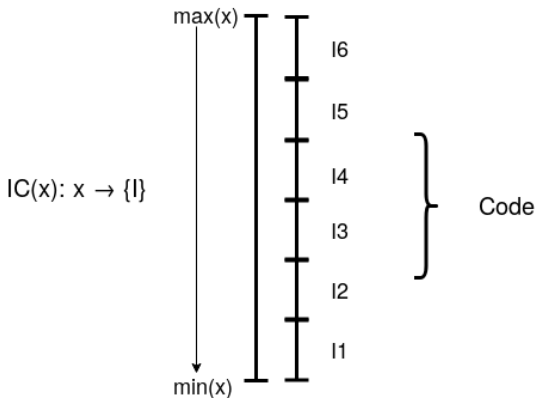


Abb. 29. Einteilung von kontinuierlichen Werteverteilungen in Intervall und Abbildung auf kategorische (diskrete) Werte

7.10. Unvollständige Trainingsdaten

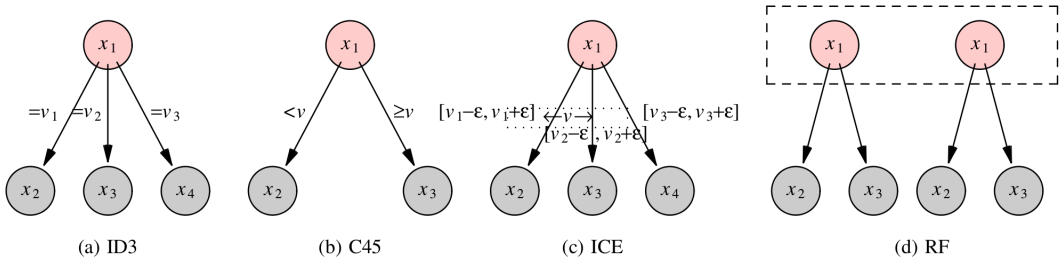
- Es kommt vor allem in der Soziologie aber auch in der Mess- und Prüftechnik vor, dass nicht alle Werte der Attributvariablen $\boldsymbol{X}$ für alle Trainingssätze bekannt sind.
  - Die Behandlung fehlender Attributwerte in den Zeilen der Datentabellen ist schwierig
- Es gibt keine Universallösung für den Umgang mit ? Werten. Möglichkeiten:
  - Ersetzen des fehlenden Werts mit einem Standardwert
  - Ersetzen des fehlenden Werts mit einem probalistisch über Verzeilungshäufigkeiten bestimmten Wert (auch unter Einbeziehung des gesamten Datensamples)
  - Attributevariablen mit fehlenden Werten nicht verwenden

7.11. Intervallkategorisierte Entscheidungsbäume (INN/ICE)

- Bisherige Entscheidungsbäume (C4.5/ID3) wurden entweder mit einer diskreten Anzahl von kategorischen Werten verzweigt oder mittels binärer Relationen!

- Aber Sensoren (sowohl in der Mess-und Prüftechnik als auch ind er Soziologie) sind fehlerbehaftet, d.h. es gibt bei jedem x-Wert ein Unsicherheitsintervall $[x\text{-}, x+] \rightarrow$ **Rauschen**
- Damit können Entscheidngsbäume (anders als Neuronale Netze oder Regressionslerner) nicht umgehen.
  - Wenn der Split mit $x<50$ und $x\ 50$ an einem Knoten mit x erfolgt würde bei Werten um 50 und überlagerten Rauschen ein Entscheidungsproblem entstehen!
- Lösung: k-stellige Knoten mit Intervallverzweigungen, also:

$$M(X) = \begin{cases} x_i \in [v_1 - \varepsilon_i, v_1 + \varepsilon_i], \{\cdots \\ x_i \in [v_1 - \varepsilon_i, v_1 + \varepsilon_i], \{\cdots \\ \cdots \\ x_i \in [v_n - \varepsilon_, v_n + \varepsilon_i], \{\cdots \end{cases}$$



(a) ID3 (b) C45 (c) ICE (d) RF

Abb. 30. Vergleich der verschiedenen Baumarten und Knotenverzweigungen

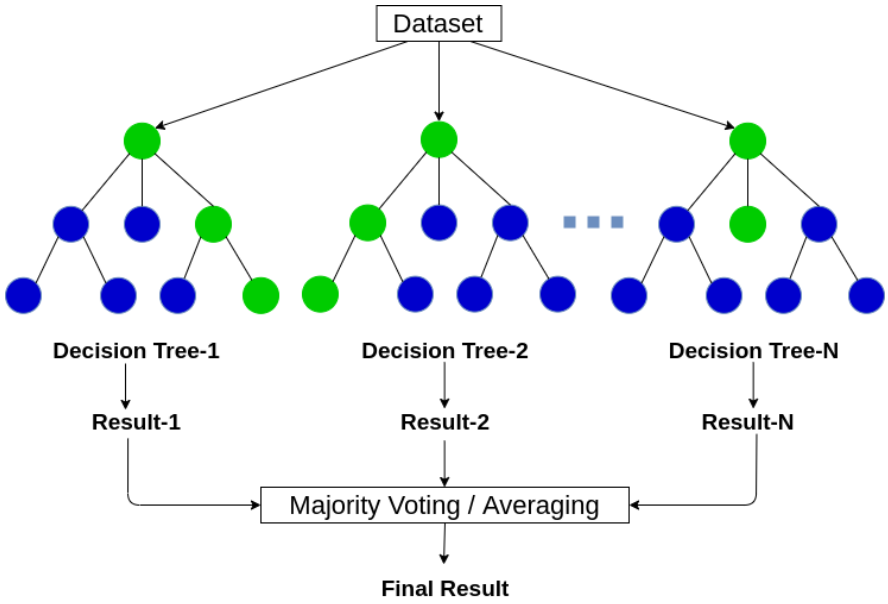
- Bei der Konstruktion des Entscheidungsbaums werden wieder nach Informationsgewinn bzw. Gewinnverhältnis Attributvariablen und Spalten der Datentabelle ausgewählt.
- Die numerischen Werte werden sowohl beim Training als auch bei der Inferenz durch Intervalle ersetzt → Ersetzung von diskreter mit **Intervallarithmetik**
- Entropien usw. werden durch kategorisierte Intervalle bestimmt
- Das große Problem: Für jede Variable muss ein abgeschätzt werden → Statistisches Modell erforderlich.
- Und was bedeuten jetzt überschneidende Intervalle?
  - Überschneidungen bedeuten Ununterscheidbarkeit!

Inferenz mit NN Suche

- ➤ Jeder Knoten x_i hat ausgehende Kanten mit annotierten Intervallen [v_j-,v_j+]
- ➤ Bei einem neuen zu testenden Variablenwert v wird einseits auch ein Intervall [v- ,v+] gebildet und mit den Kantenintervallen verglichen, andererseits wird das nächstliegende Intervall gesucht

7.12. Random Forest Trees

- ➤ Multiinstanzmodell
  - ❑ Es werden m Entscheidungsbäume DT=$\{dt_1,..,dt_m\}$ getrennt gelernt und erzeugt
  - ❑ "Random": Die Aufteilung der Daten in Teilungsvariablen erfolgt randomisiert!
  - ❑ Eingabedaten werden zur Inferenz an alle Teilbäume dt_i DT gegeben
  - ❑ Alle Ausgabevariablen der Teilbäume werden fusioniert
- ➤ Fusion:
  - ❑ Mittelwert (bei intervallkodierten oder intervallskalierbaren kat. Zielvariablen durch Dekodierung in numerische Werte)
  - ❑ Mehrheitsentscheid
  - ❑ Konsensfindung (Verhandlung)
- ➤ Parametersatz:
  - ❑ Stelligkeit eines Knotens (Anzahl der ausgehenden Kanten)
  - ❑ **Anzahl der Teilbäume**
  - ❑ **Partitionierung** des Eingaberaums (d.h. ein bestimmter Baum verwendet nur eine Teilmenge der Spalten aus $\boldsymbol{D}$)
  - ❑ Fusionsmodell und Algorithmus



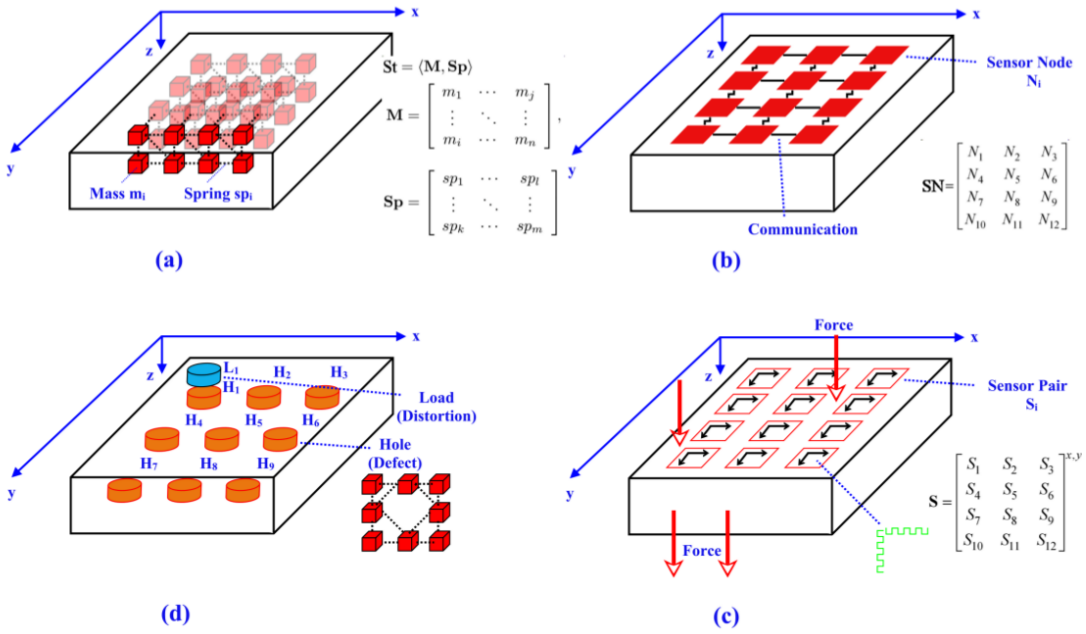
[Abhishek Sharma, 2020, www.analyticsvidhya.com]

Abb. 31. Grundprinzip von Multibaumklassifikatoren

7.13. Beispiel

Experiment

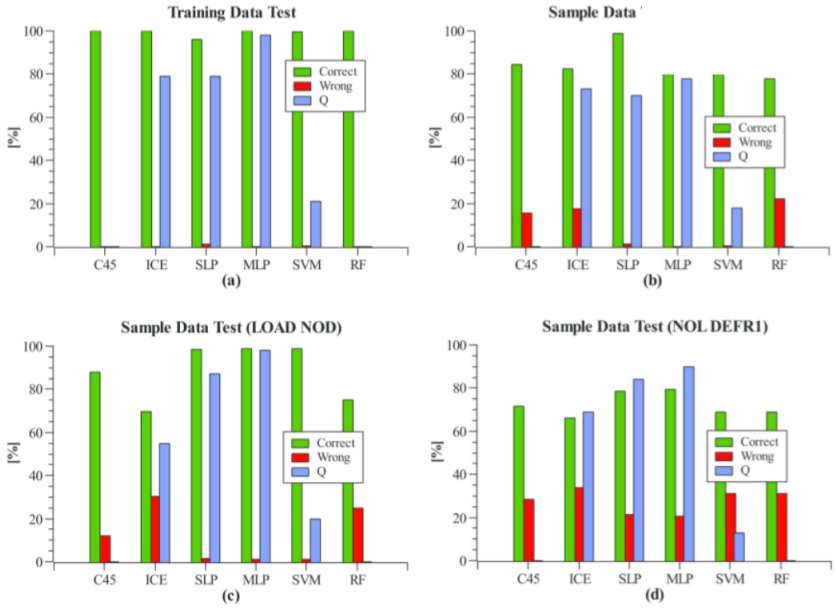
- ➤ Sensornetzwerk von (3 × 4) Dehnungssensoren
- ➤ Stimulus: Bauteilschwingung
- ➤ Varianz: Bauteilschäden (Defekte)
- ➤ Zielvariable: Schadensklassifikation (9 Positionen)
- ➤ Merkmalsvektor: Downgesampletes zeitaufeglöstes Sensorsignal einer



Ressourcen

ML	Parameter	Learning Time	Modelsize (Bytes)
C45	-	8s	4k
ICE	ϵ=0.01	100ms	16k
SLP	*iter*=1000	1s	190k
MLP[1]	*iter*=1000, $layers_{\text{hidden}} = [5]$	2s	210k
MLP[2]	*iter*=20000, $layers_{\text{hidden}} = [5]$	22s	210k
SVM	*iter*=1000, *kernel*={*type*: *rbf*, *C*:0.5, σ:0.1}	90s	260k
RF	$depth_{\text{max}} = 10$, $trees = 5$	150ms	1.2M

Genauigkeit



7.14. Zusammenfassung

- ➤ Entscheidungsbäume sind für die Klassifikation von kategorischen Zielvariablen geeignet
- ➤ Numerische Zielvariablen müssen intervallkodiert werden.
- ➤ ID3/C4.5 Lerner können numerische und kategorische Eingabevariablen (Attribute) verwenden
  - ❑ Ein Attributariable ist ein Teilungspunkt
- ➤ Rauschen auf Sensordaten muss durch “Unsicherheitsintervall” und Intervallarithmetik behandelt werden
- ➤ Vergleich mit anderen Lernverfahren zeigt gute Ergebnisse (je nach Problem)

8. Klassifikation mit Künstlichen Neuronale Netze

***Zielvariablen**: Numerische Variablen*
***Eigenschaftsvariablen**: Numerische Variablen*
***Modell**: Gerichteter Graph (zyklisch oder azyklisch)*
***Training und Algorithmen**: Backpropagation*
***Klasse**: Überwachtes Lernen*

8.1. Künstliche Neuronale Netze

➤ Ein Künstliches Neuronales Netz (KNN) ist ein gerichteter Graph bestehend aus einer Menge von Knoten $\boldsymbol{N}$ und Kanten $\boldsymbol{E}$ die die Knoten verbinden

- ❏ Knoten: Neuron oder Perzeptron mit einem oder mehreren Eingängen $\boldsymbol{I}$ und einem Ausgang o; Berechnungsfunktion $g(\boldsymbol{I})$: $\boldsymbol{I} \rightarrow o$
- ❏ Kanten: Gewichteter Datenfluß vom Ausgang eines Neurons zum Eingang eines anderen (oder des selben) Neurons

Ein KNN ist eine Komposition aus einer Vielzahl von Abbildungsfunktionen $\boldsymbol{G}=(g_1,g_2,..,g_m)$. Es gibt Parallelen zu Regressionsverfahren mit Funktionen.

➤ Zusammengefasst ausgedrückt:

$$M(X) : X \rightarrow Y, X = \{x_i\}, Y = \{y_j\}$$
$$KNN = \langle N_x, N_d, N_y, E \rangle$$
$$N_x = \{n_i : n_i \leftrightarrow \{x_j\}\}, N_d = \{n_d\}, N_y = \{n_k : n_k \leftrightarrow y_k\}$$
$$n = g(\tilde{p}, \tilde{w}, b) : \tilde{p} \rightarrow o = f(\sum_i w_i p_i + b)$$
$$E = \{e_{ij} : n_i \mapsto n_j w_{ij}\}$$

➤ f ist eine Transferfunktion die die akkumulierten Eingangswerte auf den Ausgangswert o abbildet, und g ist dann die gewichtete und akkumulative Transferfunktion

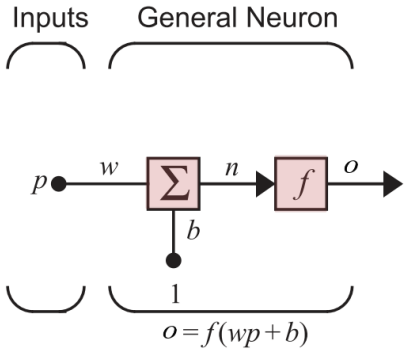
➤ Unterschied (künstliches) Neuron und Perzeptron:

- ❏ Ein Neuron ist immer eine Elementarzelle
- ❏ Ein Perzeptron kann ein einzelnes Neuron oder ein Netzwerk aus Neuronen beschreiben

➤ Daher gibt es:

- ❏ Single Layer Perceptron (SLP) → Nur Eingangs- $\boldsymbol{N}_{\mathrm{x}}$ und Ausgangsneuronen $\boldsymbol{N}_{\mathrm{y}}$
- ❏ Multi Layer Perceptron (MLP) → + Innere Neuronen $\boldsymbol{N}_{\mathrm{d}}$

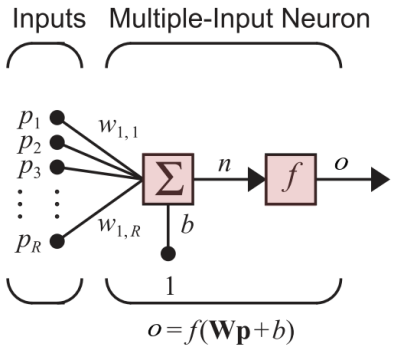
8.2. Das Neuron



[15]

Abb. 32. Ein einzelnes Neuron mit einem einzelnen Eingang p und einem Ausgang o. w ist ein Gewichtungsfaktor (ein Gewicht für eingehendes p) und b ist ein Bias (Offset)

8.3. Das Mehreingangsneuron

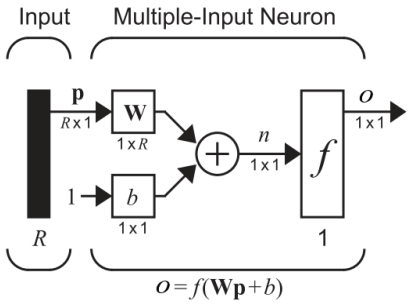


[15]

Abb. 33. Ein einzelnes Neuron mit einem Eingangsvektor $\boldsymbol{p}$ und einem skalaren Ausgang o. $\boldsymbol{w}$ ist ein Gewichtungsfaktorvektor (ein Gewicht für eingehendes p) und b ist ein Bias (Offset)

8.4. Neuronale Netze und Matrizen

➤ Neuronale Netze werden durch eine Graphenstruktur (statische Paremeter) und mathematisch durch Matrizen (dynamische Parameter) beschrieben:



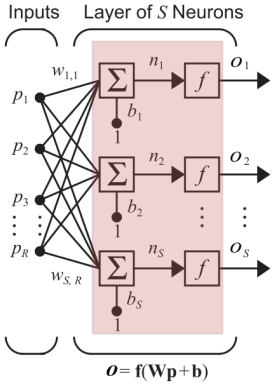
[15]

Abb. 34. Ein einzelnes Neuron mit einem Eingangsvektor $\boldsymbol{p}$ und einem skalaren Ausgang o. $\boldsymbol{w}$ ist ein Gewichtungsfaktorvektor (ein Gewicht für eingehendes p) und b ist ein Bias (Offset); jetzt in Matrizenform (Annotation)

8.5. Schichten von Neuronalen Netzen

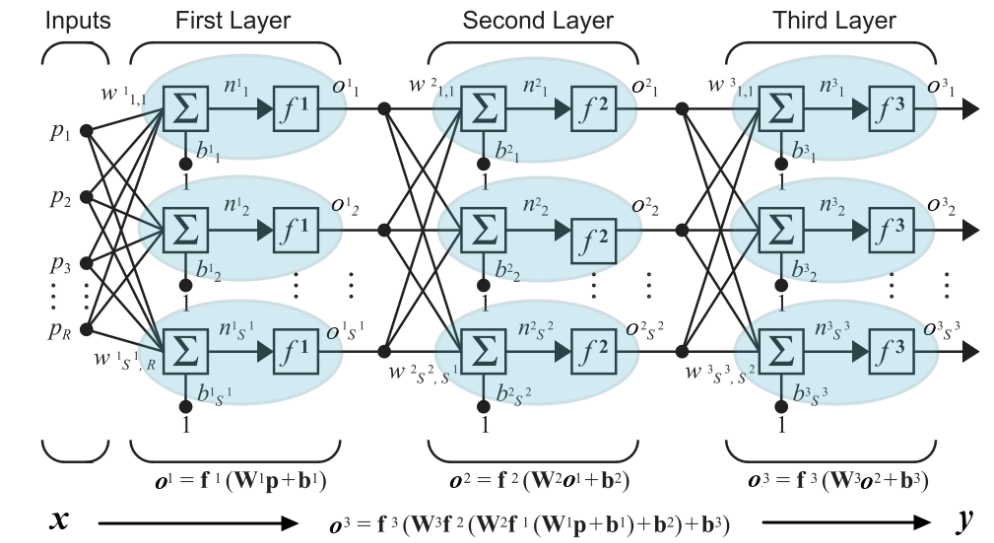
➤ I.A. werden Neuronen von neuronalen Netzen in Schichten (Layern) angeordnet und gruppiert

- ❏ Günstig für Matrixalgebra
- ❏ Aber nicht notwendig!



[15]

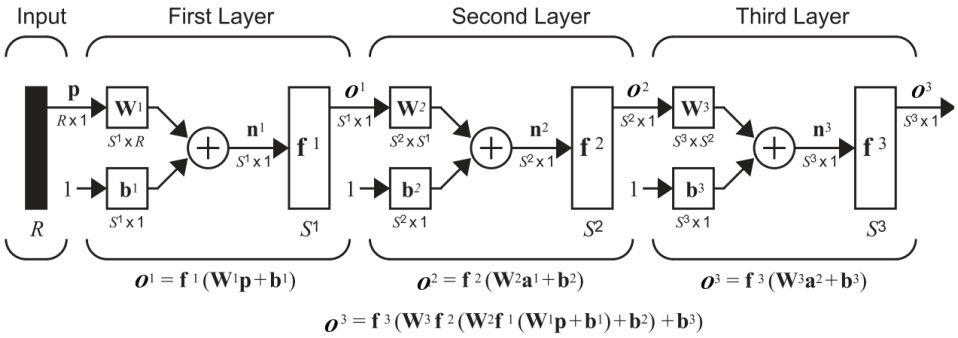
8.6. Struktur eines KNN



[15]

Abb. 35. Grundlegende Struktur eines KNN mit Matrizen (blaue Ellipse=1 Neuron)

8.7. Vereinfachte Form eines KNN



[15]

Abb. 36. Vereinfachte Struktur eines KNN mit Matrizen

8.8. Klassen von KNN

Forwärtsgekoppelte Netzwerke

Azyklischer gerichteter Graph, d.h. es gibt nur eine Vorwärtspropagation von einer Schicht zur nächsten (keine Rückkopplung).

➤ Diese Netzwerke können rein funktional beschrieben und berechnet werden.

➤ Es gibt keinen Zustand!

➤ D.h. die aktuellen Ausgangswerte hängen nur von den aktuellen Eingangswerten ab!

Rückgekoppelte Netzwerke

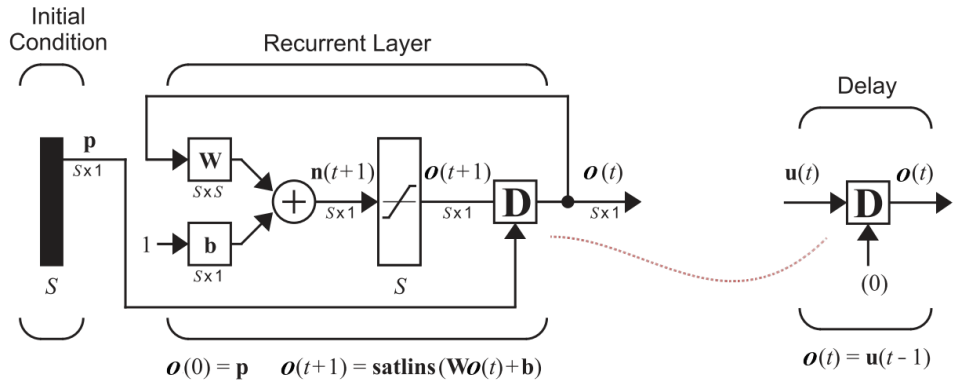
Zyklischer gerichteter Graph, d.h. es gibt Rückkopplungen (Ausgang eines Neurons geht in Eingänge der aktuellen oder vorherigen Schichten).

➤ Diese Netzwerk können nicht rein funktional beschrieben und berechntet werden!

➤ Sie besitzen einen Zustand, d.h. der Ausgangswert hängt von der Historie vergangener Eingabewerte ab!

8.9. Rückgekoppelte Netzwerke

➤ Geeignet für Prädiktion auf zeit- und Datenserien $D(\mathrm{t})=d_0,d_1,...,d_\mathrm{t}$



[15]

Abb. 37. Rückgekoppeltes und zustandsbehaftetes KNN mit einer Verzögerungsfunktion (Speicher)

8.10. Transferfunktion

➤ Auch Aktivierungsfunktion genannt (in Anlehnung an biologische Vorbild)

- ❏ Biologisch: Häufig eine Schwellwertfunktion
- ❏ Künstlich / ML: Auch lineare Übertragunsfunktionen!

➤ Es gibt eine Vielzahl verschiedener Funktionen

- ❏ Die einfachste wäre (wenn auch wenig in Gebrauch): $f(a) = a$

Warum ist eine solche Übertragungsfunktion ungeeignet bzw. problematisch?

➤ Welche mathematischen **Eigenschaften** (Übertragungskurve) sollte wohl eine Transferfunktion besitzen?

- ❏ Zur Erinnerung: Wir nehmen an dass der Wertebereich von einem x [-1,1] ist. Ebenso für ein y [-1,1].

Transferfunktionen besitzen häufig begrenzende Eigenschaften (Sättigungsverhalten), und nicht lineares Übertragungsverhalten

Name	Input/Output Relation	Icon	Function
Hard Limit	$a = 0 \quad n < 0$ $a = 1 \quad n \geq 0$		hardlim
Symmetrical Hard Limit	$a = -1 \quad n < 0$ $a = +1 \quad n \geq 0$		hardlims
Linear	$a = n$		purelin
Saturating Linear	$a = 0 \quad n < 0$ $a = n \quad 0 \leq n \leq 1$ $a = 1 \quad n > 1$		satlin
Symmetric Saturating Linear	$a = -1 \quad n < -1$ $a = n \quad -1 \leq n \leq 1$ $a = 1 \quad n > 1$		satlins
Log-Sigmoid	$a = \frac{1}{1+e^{-n}}$		logsig
Hyperbolic Tangent Sigmoid	$a = \frac{e^{n}-e^{-n}}{e^{n}+e^{-n}}$		tansig
Positive Linear	$a = 0 \quad n < 0$ $a = n \quad 0 \leq n$		poslin
Competitive	$a = 1$ neuron with max n $a = 0$ all other neurons	C	compet

[15]

Abb. 38. Verschiedene gebräuchliche Transferfunktionen $f(a)$

8.11. Ein einfaches Neuron - Funktional

$$f_{sigmoid}(a) = \frac{1}{1+e^{-a}}$$

$$g(x_1, x_2, x_3) = f_{sigmoid}(b + \sum w_i x_i)$$

/webwork

Neuron

```
function sigmoid(x) {
  return 1/(1+Math.exp(-x))
}
print(sigmoid(0))
var data = [
  {x1:1,  x2:2,  y:1.0},
  {x1:-1, x2:2,  y:0.0},
  {x1:0,  x2:-1, y:0.0},
]
function neuron(x1,x2,w,b) {
  var accu = x1*w[0]+x2*w[1];
  return sigmoid(accu+b)
}
var w = [1.0,1.0], b=0;
data.forEach(function (row) { print(neuron(row.x1,row.x2,w,b),row.y) });
```

8.12. Parametersatz des KNN

Statische Parameter

- Anzahl der Eingangsneuronen (verbunden mit $\mathbf{x}$), abhängig von der Anzahl der Eingabevariablen $|\mathbf{x}|$ und der Kodierung (numerisch vs. kategorisch)

➤ Anzahl der Ausgangsneuronen (abhängig von der Kodierung). Bei numerischen Zielvariablen $\mathbf{y}$ gilt also: $|N_\mathrm{y}|=|\mathbf{y}|$

➤ Anzahl der inneren verdeckten Neuronen $|N_\mathrm{d}|$ und deren Anordnung in Schichten

➤ D.h. die **Konfiguration** des Netwerkes ist $[c_1,c_2,..,c_\mathrm{m}]$ bei m Schichten und c_i Neuronen pro Schicht

➤ Bei vollständig verbundenen Schichten ist keine Angabe der Vernetzung notwendig

Dynamische Parameter

- Im wesentlichen die Gewichtematrix $\mathbf{W}_\mathrm{i}$ (Schicht i):

$$W_i = \begin{bmatrix} w_{1,1} & amp; w_{1,2} & amp; \cdots & amp; w_{1,R} \\ w_{2,1} & amp; w_{2,2} & amp; \cdots & amp; w_{2,R} \\ \vdots & amp; \vdots & amp; & amp; \vdots \\ w_{S,1} & amp; w_{S,2} & amp; \cdots & amp; w_{S,R} \end{bmatrix}, B_i = \begin{bmatrix} b_1 \\ \vdots \\ b_S \end{bmatrix}$$

Mit S: Anzahl der Neuronen in der Schicht, R: Anzahl der Eingangsvariablen (oder Neuronen der vorherigen Schicht)

- Der Ausgangswert eines Neurons n_j ist dann gegeben durch einen Wert aus B und die j-te Zeile von $\boldsymbol{W}$:

$$o(\tilde{p}) = f({}_jW^T\tilde{p} + b_i)$$

- Bei mehrschichtigen Netzwerken hat man eine Menge von Gewichtematrizen, die zu einem Tensor zusammengefasst werden können.

8.13. Training von KNN

- Wie bei allen überwachten Lernproblemen gilt es eine Fehlerfunktion zu minimieren:

$$M(\tilde{x}) : \tilde{x} \rightarrow \tilde{y}$$
$$\underset{W}{\operatorname{argmin}}\ err(M) = |y(\tilde{x}) - y_0(\tilde{x})|, \forall(x, y_0) \in D$$

Ziel ist die Minimierung des Fehlers unserer Modellhypothese $M(\mathbf{x})$ durch Anpassung der Gewichtematrix $\boldsymbol{W}$ und evtl. (wenn vorhanden) des Offsetvektors $\boldsymbol{B}$
Es ist leicht zu erkennen dass das Training einen hochdimensionalen Parametersatz anpassen muss. Es ist nicht unmittelbar klar wie ein optimales $\boldsymbol{W}$ abgeleitet werden kann!

Erklärbarkeit

- Der Zusammenhang von y und x ($x \rightarrow y$) ist schon bei einem einschichtigen Netzwerk nur noch schwer nachvollziehbar!
- Eine Invertierung (inverses Problem $y \rightarrow x$) ist ebenso nur schwer möglich

➤ Eigentlich ist nur ein einzelnes Neuron erklärbar und verständlich

❑ Dort ist die Anpassung (des Gewichtevektors **w**) noch durch multivariate Regression möglich

Beispiel

➤ "Gradient Descent" Verfahren

➤ Problem: $\mathbf{x}=(a,b)$, y

➤ Netzwerk: Ein Neuron, Sigmoid Transferfunktion

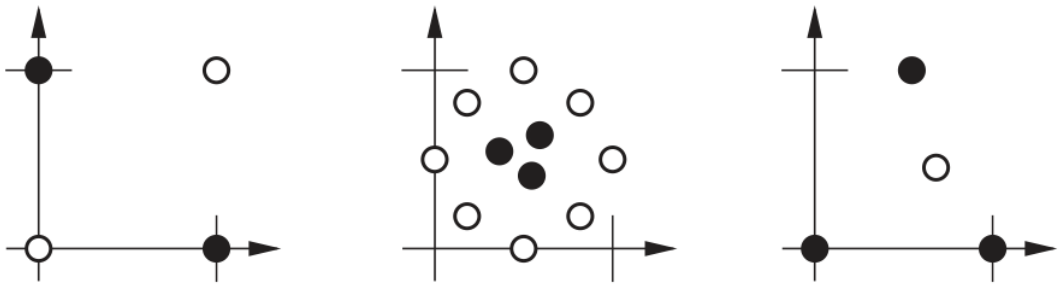
/webwork

Neuron

```
function sigmoid(x) {
  return 1/(1+Math.exp(-x))
}
print(sigmoid(0))
var data = [
  {x1:1,  x2:2,  y:1.0},
  {x1:-1, x2:2,  y:0.0},
  {x1:0,  x2:-1, y:0.0},
]
function neuron(x1,x2,w,b) {
  var accu = x1*w[0]+x2*w[1];
  return sigmoid(accu+b)
}
var w = [1.0,1.0], b=0, sets=[0,1,2], rate=0.1, errors=[];
for(var run=0;run<10000;run++) {
  var set=Math.random.select(sets),
      row=data[set];
  var y=neuron(row.x1,row.x2,w,b), err=y-row.y;
  if ((run % 100)==0) { errors.push(Math.abs(err)); print(Math.abs(err))}
  w[0]=w[0]-rate*err*row.x1;
  w[1]=w[1]-rate*err*row.x2;
}
print(w)
data.forEach(function (row) { print(neuron(row.x1,row.x2,w,b),row.y) });
```

8.14. Nicht lineare Probleme

SLP können nur lineare Probleme separieren.



[15]

Abb. 39. Nicht linear separierbare Probleme - nur mit MLP klassifizierbar

/webwork

Neuron

```
function sigmoid(x) {
  return 1/(1+Math.exp(-x))
}
print(sigmoid(0))
var data = [
  {x1:0,  x2:0,  y:0},
  {x1:1,  x2:0,  y:1},
  {x1:0,  x2:1,  y:1},
  {x1:1,  x2:1,  y:0},
]
function neuron(x1,x2,w,b) {
  var accu = x1*w[0]+x2*w[1];
  return sigmoid(accu+b)
}
var w = [1.0,1.0], b=0, sets=[0,1,2], rate=0.1, errors=[];
for(var run=0;run<10000;run++) {
  var set=Math.random.select(sets),
      row=data[set];
  var y=neuron(row.x1,row.x2,w,b), err=y-row.y;
  if ((run % 100)==0) { errors.push(Math.abs(err)); print(Math.abs(err))}
  w[0]=w[0]-rate*err*row.x1;
  w[1]=w[1]-rate*err*row.x2;
}
print(w)
data.forEach(function (row) { print(neuron(row.x1,row.x2,w,b),row.y) });
```

8.15. Backpropagation Verfahren

➤ Bekanntes und gängiges Verfahren

https://hmkcode.com/ai/backpropagation-step-by-step

Gradientenverfahren

➤ Baut auf dem Minimierungsansatz “Gradient Descent” (GD) auf (Absteigender Gradient)

➤ Beim GD Verfahren wird eine Funktion, z.B. $f(x,w)$: $x \rightarrow y$ derart über den Parameter w angepasst wird dass der Fehler $err=|y\text{-}y_0|$ minimal wird

➤ Es wird nun die Änderung des Fehlers beobachtet Δerr und der (oder später die) Parameter w mit der Ableitung des Fehlerwerts $err/\ w$ zu der Änderung des Parameters korrigiert:

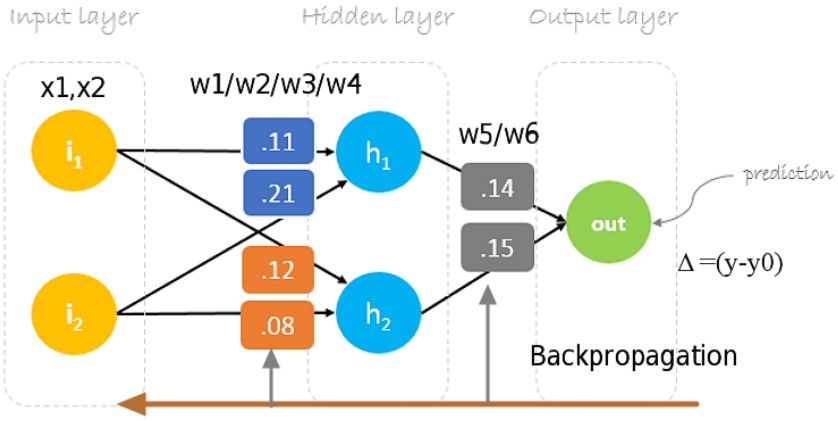
$$w' = w - \alpha \frac{\partial err}{\partial w}$$

➤ Vereinfacht gilt:

$$\frac{\partial err}{\partial w} \sim x(y - y_0)$$

➤ Jetzt wird ein neuronales Netzwerk betrachtet, wo die Neuronen ebenfalls Funktionen mit Eingangsvariablen und Ausgangsvariablen sind

➤ Bei zusammengesetzten Funktionen (z.B. auch Neuronen in inneren Schichten) müssen die Gewichte schrittweise von hinten nach vorne angepasst werden



[hmkcode.com/ai/backpropagation-step-by-step]

Abb. 40. Beispiel eines ANN mit Kantengewichten und dem Ansatz der Backpropagation

➤ Die Gewichte werden nun Schicht für Schicht unter Einbeziehung der gewichteten Fehlerpropagation fleichermaßen angepasst

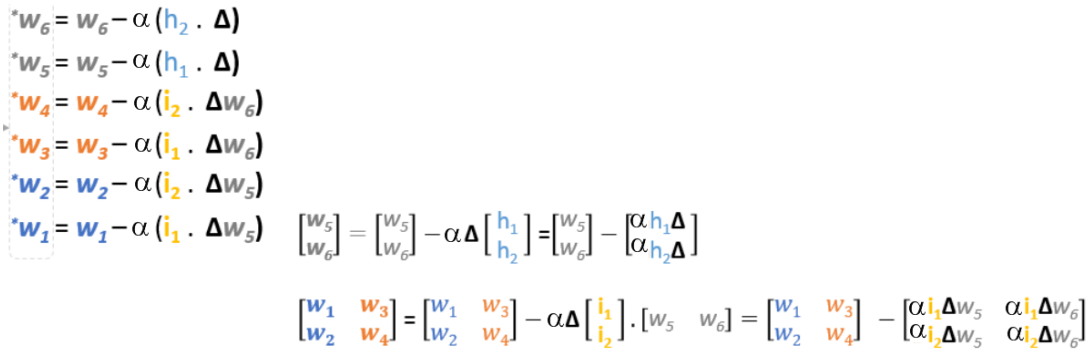


Abb. 41. Backpropagation des Fehlers zu den Eingängen des Beispielnetzwerkes

8.16. Kategorische Multiklassen Probleme

➤ Wenn die Ergbnisvariable vom kategorischen Typ ist dann gibt es zwei Möglichkeiten:

One-Hot Kodierung

Jedes Klassensymbol (also ein diskreter Wert v_i der Zielvariable y) wird durch ein Ausgangsneuron repräsentiert

Multi-level Kodierung

Jedes Klassensymbol wird durch einen Wert aus dem Wertebereich eines Ausgangsneurons repräsentiert

➤ Problem: Nicht lineare Transferfunktion und Sättigungsverhalten

❏ Die gleichen Verfahren sind auch auf kategorische Eingabevariablen anwendbar

8.17. Numerische Prädiktorfunktionen

➤ Neben der Klassfikation lassen sich mit ANN auch numerische (kontinuierliche) Funktionen lernen

➤ Damit wird **Funktionsapproximation** wie bei den Regressionsverfahren möglich

❏ Unterschied: Bei der Regression ist die funktionale Struktur von $f(x)$: $x \rightarrow y$ bereits fest und muss vorgegeben sein

- ❑ Die Verwendung eines ANN bietet da auch noch indirekt das Lernen der funktionalen Strukturen neben der Anpassung der Parameter

8.18. Literatur zur Vertiefung

[1] M. T. Hagan, howard B. Demuth, M. H. Beale, and O. D. Jesus, *Neural Network Design*. https://hagan.okstate.edu/nnd.html

8.19. Zusammenfassung

- ➤ Neuronale Netze bestehen aus Neuronen
- ➤ Neuronen sind Funktionen
- ➤ Die Kanten verbinden Ausgänge von Neuronen mit den Eingängen nachfolgender Neuronen mit einer Multiplikation eines Gewichtfaktors
- ➤ Alle Eingänge eines Neurons werden summiert, das Ergebnis einer Transfer/Aktivierungsfunktion übergeben
- ➤ Training bedeutet Anpassung der Gewichte um den Ausgangsfehler zu minimieren
  - ❑ Gängiges Verfahren: Fehlerrückpropagation

9. Ein- und Ausgabeschnittstellen von Prädiktorfunktionen

Kodierung und Dekodierung von Variablen für kont. Prädiktorfunktionen
Normalisierung und Skalierung von Daten

9.1. Kagetorische Variablen

- ➤ Kategorische Variablen müssen für die Verwendung in Prädiktorfunktionen in numerische Werte kodiert werden
  - ❑ Multi-level Kodierung: Abbildung von allen kategorischen Werten auf unterschiedlich numerische Werte (skalare und vektorielle Werte)

- ❏ One-hot Kodierung: Abbildung von allen kategorischen Werten auf 01 Vektoren

*Aber: Anders als die Intervallkodierung von numerischen Variablen (z.B. für Entsch.bäume) muss im umgekehrten Fall der Kodierung einer kat. Variable in **eine** numerische Variable eine Intervall- und Verhältnisskalierung existieren!!*

Beispiele

- ➤ Farben **C**={rot,grün,blau,braun}
  - ❏ Kodierung in skalaren kont. Variable aber mit diskreten Werten
  - ❏ Kodierung in vektorielle Variable mit kont. Werten → RGB

$$z_1(C) = \begin{cases} 1, C = \text{rot} \\ 2, C = \text{gruen} \\ 3, C = \text{blau} \\ 4, C = \text{braun} \end{cases}, z_2(C) = \begin{cases} (1.0, 0.0, 0.0), C = \text{rot} \\ (0.0, 1.0, 0.0), C = \text{gruen} \\ (0.0, 0.0, 1.0), C = \text{blau} \\ (0.5, 0.0, 0.5), C = \text{braun} \end{cases}$$

Welche Probleme ergeben sich bei der Multi-level Kodierung (skalar) von Zielvariablen bei typischen Sigmoid Transferfunktionen von Neuronen? Nichtlinearität an den Rändern [0,1]

9.2. Ein- und Ausgabeschnittstellen

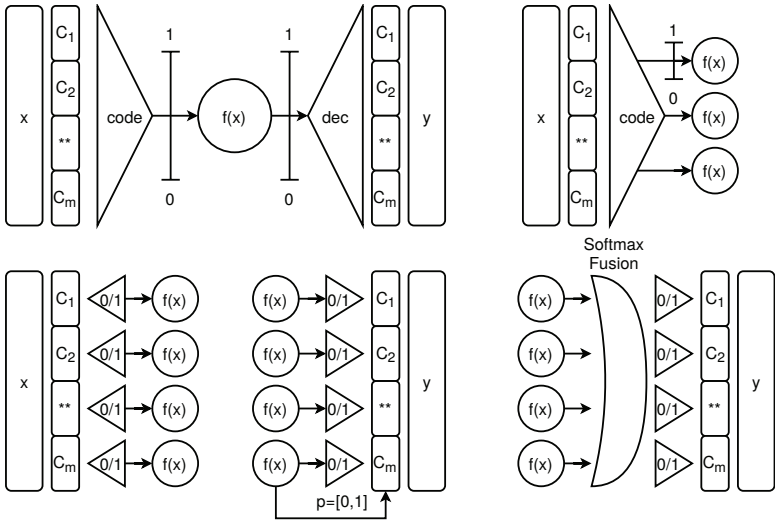


Abb. 42. Verschiedene Kodierungen von ein-/ausg. kat. Variablen

- ➤ Die One-hot Kodierung von Eingabevariablen erfordert die Berechnung eines diskretwertigen Vektors
- ➤ Aber die One-hot Kodierung von Ausgabevariablen ergibt einen kontinuierlichen Vektor der noch diskretisiert werden muss:
  - ❏ Schwellwertfunktion → Es kann zu Mehrdeutigkeiten kommen (Unentscheidbarkeit)
  - ❏ Der kont. Ausgangswert y_i (C_i) wenn im Bereich [0,1] kann als ein Maß für die Wahkrscheinlichkeit des Auftretens eines Klassensymbols C_i verwendet werden!
  - ❏ Softmax Funktion: Diese bildet einen n-dimensionalen Vektor y mit y_i [0,1] und $max\ (\Sigma\ y_\mathrm{i}) = |\mathrm{y}|$ auf einen n-dimensionalen Vektor ab mit $max\ (\Sigma\ y_\mathrm{i}) = 1$!

Softmax Funktion

$$softmax(Y) = \begin{pmatrix} \frac{e^{y_1}}{weight} \\ ... \\ \frac{e^{y_n}}{weight} \end{pmatrix}, weight = \sum e^{y_k}, y_i \in Y, Y = \begin{pmatrix} y_1 \\ ... \\ y_n \end{pmatrix},$$
$$\sum softmax(Y) \in [0, 1]$$

10. Fehleranalyse und Kostenfunktionen

Bewertung der Qualität eines Klassifikators oder von Prädiktorfunktionen
Aufteilung von Datensätzen
Künstliche Erweiterung von Datensätzen

10.1. Fehlerfunktionen

1. Ziel ist ein aussagekräftiger Fehlerwert um die Qualität des Trainings und des erzeugten Modells $M(x)$:$x \rightarrow y$ bewerten zu können
2. Fehlerwerte bei kategorischen Zielvariablen vergleichen direkt die Übereinstimmung der inferrierten und vorgegebenen Werte der Zielvariablen

   ➤ Gesamtfehler (falsche Klassifikation)

   ➤ Falsch-positiver Fehler (binärer Klassifikator)

   ➤ Falsch-negativer Fehler (binärer Klassifikator)

   ➤ Falsch-C Fehler (Multiklassifikator)

   ➤ Klassenspezifischer Fehler (Multiklassifikator)

10.2. Fehlerberechnung

Daten

➤ Es sei $\boldsymbol{D}$ die Gesamtmenge der Dateninstanzen $\boldsymbol{D}$

➤ Es sei $\boldsymbol{D}_{\text{test}}$ die Testdatenmenge $\boldsymbol{D}_{\text{test}} \quad \boldsymbol{D}$

Klassifikationsfehler

$$err(Y^0, Y^p) = \frac{1}{N} \sum_{i \in D} \begin{cases} 1, Y_i^0 \neq Y_i^p \\ 0 \end{cases}, Y^0 = D_{test}(Y), N = |D_{test}|$$

➤ Dabei sind Y^0 die vorgegebenen Werte der Zielvariablen und Y^{p} die aus dem trainierten Modell berechneten (inferrierten) Werte

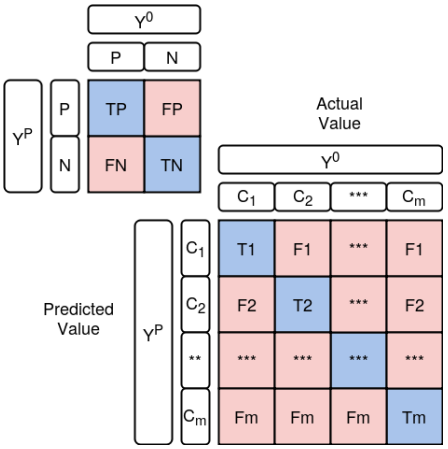
➤ Der Inferenzfehler liegt dann im Bereich [0,1]

➤ Bei binärer Klassifikation sollten die Falsch-positiv und Falsch-negativ Fehler zusätzlich getrennt bestimmt werden:

$$err_C(Y^0, Y^P) = \frac{1}{N} \sum_{j \in D(Y=C)} \begin{cases} 1, Y_j^0 \neq Y_j^P \\ 0 \end{cases}, N = |D_{test}(Y = C)|$$

10.3. Konfusionsmatrix

➤ Bei Multiklassifikation bietet sich die Konfusionsmatrix an um eine Bewertung der Klassifikationsqualität mit falsch-C Bewertungen einer Klasse C und somit err_C zu erfassen



10.4. Kreuzentropie

➤ Bei kontinuierlichen Ausgabevariablen (KNN, SVM) aber evtl. kategorischen Zielvariablen ist die Berechnung der Kreutentropie zwischen Y^0 und Y^P aussagekräftiger

➤ Eine Kreuzentropie $\rightarrow 0$ bedeutet vollständige Übereinstimmung, Werte > 0 bedeuten Abweichungen

➤ Je größer die Kreuzentropie zwischen zwei Vektoren/Matrizen ist, desto größer ist die Abweichung

Berechnung der Kreuzentropie zweier Matrizen

➤ Eingabewerte: Matrix U und V mit Zellenwerten im Bereich [0,1]

➤ Ergebnis: Skalarer Wert

➤ Problem (siehe unten): log(0) → -∞

$$
\begin{aligned}
&E_{cross}(U,V) = -mean(sumRows(A+B)),\\
&A = (A_{ij}), A_{ij} = U_{ij}\log(\max(V_{ij}, 1^{-9})),\\
&B = (B_{ij}), B_{ij} = (1-U_{ij})\log(1-\min(V_{ij}, 1-1^{-9})),\\
&mean(X) = \sum x_i / |X|,\\
&sumRows(X) = (S_i), S_i = \sum X_{ij}
\end{aligned}
$$

10.5. Beispiele

/webwork

confmat

```
var confmat = [
 [1,0,0],
 [0,0.3,0.2],
 [0,0.7,0.8]
]
Plot(confmat,
     {type:'confusion',
      title:'Machine Learning Results',
      labels:{data:['A','B','C']}})
```

crossEntr

```
U=[[0,1],[1,0],[0,1],[0,1]]
V=[[0.1,0.99],[0.6,0.4],[0.01,0.99],[0.1,0.95]]
var cross=ML.statistics.crossEntropy(U,V)
var error= ML.math.meanMat(
  ML.math.activateTwoMat(U,V,
    function (u,v) { return Math.pow(u-v,2) }))
print('Cross Entropy  = '+cross);
print('Mean Sq. Error = '+(error*100));
```

11. Netzwerkkonfiguration

Konfiguration von ein- und mehrschichtigen neuronalen Netzwerken
Festlegung der Anzahl der Schichten, Knoten pro Schicht, und Vernetzung

11.1. Neuronale Netze

- ➤ Ein neuronales Netz ist ein gerichteter Funktionsgraph
- ➤ Einzelne Funktionsknoten können zu Schichten zusammengefasst werden

Eingabeschicht

Die Anzahl der Eingabeneuronen wird durch den Eingangsvektor $\boldsymbol{x}$ und der Kodierung bestimmt! Jedes Element von $\boldsymbol{x}$ ist mit einem Eingang eines Eingangsneurons verbunden

Ausgabeschicht

Die Anzahl der Ausgabeneuronen wird durch den Ausgangsvektor $\boldsymbol{y}$ und der Kodierung bestimmt! Jedes Element von $\boldsymbol{y}$ ist mit einem Ausgang eines Ausgangsneurons verbunden

Innere Schichten

Neuronen die weder mit den Ein- noch den Ausgängen direkt verbunden sind.

Konfiguration

- ➤ Wenn von vollständig verbundenen Schichten ausgegangen wird kann die Konfiguration eines KNN mit einem Vektor/Array angegeben werden. Jedes Element gibt die Anzahl der Neuronen pro Schicht an.
- ➤ Beispiele:

```
layers=[1] → SLP
layers=[2,1] → SLP
layers=[2,3,1] → MLP
layers=[1,4,3,1] → MLP
```

Verbindungen

1:1

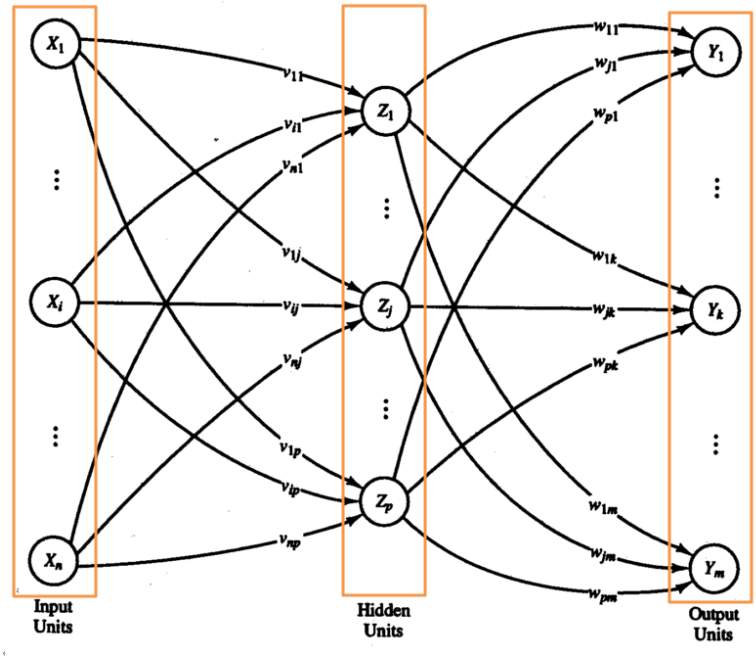
Der Ausgang eines Neurons einer Schicht i (oder einer Eingabevariable x) wird mit dem Eingang eines folgenden Neurons der Schicht j verbunden

1:n

Die Ausgänge aller Neuronen einer Schicht i (oder alle Eingabevariablen $\boldsymbol{x}$) werden mit den Eingängen jedes folgenden Neurons der Schicht j verbunden → **Vollständig verbundene Schicht**

0/1:k

Nur ein Teil der Ausgänge werden mit den Eingängen von folgenden Neuronen verbunden → **Unvollständig und irregulär verbundene Schichten** (eher Sonderfall)



[16]

Abb. 43. Vollständig verbundenes Netzwerk

➤ Die statische Festlegung der Konfiguration der inneren Schichten ist schwierig!

➤ Es gilt:

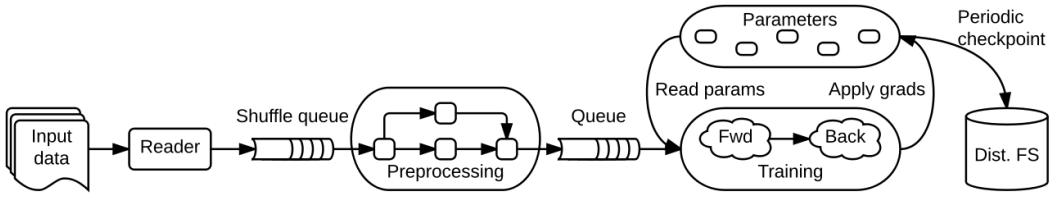
- ❏ Linear separierbare Probleme verwenden keine inneren Schichten!
- ❏ Nichtlinear separierbare Probleme können innere Schichten verwenden
- ❏ Beim Training wirken innere Schichten zunächst wie eine “Mauer”, die Wirkung auf den Ausgang wird kleiner
- ❏ Bei inneren Schichten muss die Trainingsdatenvarianz und Anzahl der Instanzen groß sein
- ❏ Je mehr innere Schichten und Neuronen um so mehr Trainingsinstanzen und Anstieg der Rechenzeit!

12. ML Frameworks

Tensorflow
Neataptic
Torch
ML

12.1. Tensorflow

- ➤ Datenverarbeitung mit tiefen Künstlichen Neuronalen Netzwerken (“Deep Learning”) → “Google Brain”
- ➤ Strombasierte Datenverarbeitung mit Pipelines (Queues)
  - ❏ Datenverarbeitungsgraph (Datenflussdiagram)



[103]

- TensorFlow verwendet ein einheitliches Datenflussdiagramm, um sowohl die Berechnung in einem Algorithmus als auch den Zustand darzustellen, in dem der Algorithmus arbeitet.
- Im Gegensatz zu herkömmlichen Datenflusssystemen, bei denen Knoten funktionale Berechnungen für unveränderliche Daten darstellen, ermöglicht TensorFlow Kanten mit Berechnungen darzustellen, die einen veränderlichen Zustand besitzen oder aktualisieren [103].
- https://www.tensorflow.org
- Durch die Vereinheitlichung des Berechnungs- und Zustandsmanagements in einem einzigen Programmiermodell ermöglicht TensorFlow Programmierern, mit verschiedenen Parallelisierungsschemata zu experimentieren, die beispielsweise die Berechnung auf Servern oder GPUs erlauben.
- Eine typische TensorFlow-Anwendung hat zwei verschiedene Phasen:
- Die erste Phase definiert das Programm (Z. B. ein zu trainierendes neuronales Netzwerk und die Aktualisierungsregeln) als Symbolisches Datenflussdiagramm mit Platzhaltern für die Eingabedaten und Variablen, die den Status darstellen.
- Die zweite phase führt eine optimierte Version des Programms auf einer Menge verfügbarer Geräte aus (z.B. Server, CPU, GPU).

Verabeitungseinheiten

- Das "Programm" ist zunächst unabhängig von der verwendeten Verarbeitungsarchitektur
- Abstraktion für heterogene Beschleuniger: Neben Allzweckgeräten wie multicore-CPUs und GPUs können spezielle Beschleuniger für Deep Learning signifikante Leistungsverbesserungen und Energieeinsparungen erzielen.

Verabeitungsmodell und Parallelität

TensorFlow unterscheidet sich von batchbasierten Datenflusssystemen in zweierlei Hinsicht:

- Das Modell unterstützt mehrere gleichzeitige Ausführungen auf überlappenden Teilgraphen des Gesamtgraphen.
- Das Modell unterstützt verteilte Berechnung

➤ Einzelne Knoten des Graphes können einen veränderlichen Zustand haben, der zwischen verschiedenen Ausführungen des Diagramms geteilt werden kann.

Elemente des Datenflussgraphens

➤ In einem TensorFlow-Diagramm stellt jeder Knoten eine Einheit der lokalen Berechnung dar, und jede Kante repräsentiert die Ausgabe von oder Eingabe in einen Knoten.

❑ Die Berechnung findet an Eckpunkten mit Operationen auf Werten statt, die entlang der Kanten als Tensoren fließen.

Tensoren

Alle Daten werden als monosortige Tensoren modelliert (n-dimensionale Arrays), wobei die Elemente vom primitiver Typ wie int32, float32 oder string sind (wobei string Binärdaten darstellen kann).

Operatoren

Eine Operation nimmt m 0 Tensoren als Eingangswerte und erzeugt n 0 Tensoren als Ausgangswerte.

Zustansbasierte Operationen

Eine Operation kann einen veränderlichen zustand besitzen, der bei jeder Ausführung gelesen und/oder geschrieben wird. D.h. das Ergebnis einer Operation hängt von vorherigen Berechnungen ab!

Queues

TensorFlow unerstützt mehrere Warteschlangenimplementierungen, die Koordination von Berechnungen unterstützen.

12.2. tensforflow.js

➤ Tensorflow ist primär in C++ implementiert

❑ Ausführung auf speziellen Verarbeitungseinheiten wie GPUs erfordert eine Kodeübersetzung zur Laufzeit

➤ Es gibt Python Anbindungen

➤ tensorflow.js ist eine Implementierung von Tensorflow rein in JavaScript und kann in Browsern ausgeführt werden

- ❏ GPU Nutzung z.B. über Open/WEBGL

➤ https://www.tensorflow.org/js

12.3. Nachteile von Tensorflow

➤ Durch die Trennung von Daten, Kode, und Verarbeitungseinheiten schlechte Performanz bei "kleinen" Problemen (hohe Initialisierungszeit und Overhead)

- ❏ Beispiel Lernen des EXOR Problems: Tensorflow.js benötigt ca 3-5 Minuten für das Training, Neataptic.js nur 100ms!

➤ Durch primäre Matrixalgebra nicht flexibel anpassbar

- ❏ Z.B. kaum Möglichkeit der Verwendung von evolutionären Algorithmen (Dynamische Restrukturierung des KNN)

➤ Lernkurve ist am Anfang flach

12.4. Neataptic

➤ Ebenfalls ein KNN Framework,

- ❏ Reine JavaScript Implementierung
- ❏ Basiert NICHT auf Matrixalgebra; die Neuronen werden einzeln berechnet

➤ Aber mit (optionalen) evolutionären Algorithmen

- ❏ Das Training kann neben Kantengewichten auch die Struktur ändern: Knoten und Kanten können getauscht, entfernt, oder hinzugefügt werden!

➤ Für kleine Probleme gute Perfomanz

➤ Lernkurve ist "steil"

➤ Bietet eine Vielzahl von Berechnungsfunktionen und Strukturen

- ❏ Struktur kann beliebig und irregulär programmiert werden
- ❏ Es gibt "Architekten" für viele gängige Architekturen
- ❏ Forwärtsgekoppelte und rückgekoppelte Architekturen

- https://wagenaartje.github.io/neataptic/

12.5. Torch

- Torch ist eine open-source-Bibliothek für maschinelles lernen, ein scientific computing framework und eine Skriptsprache, die auf der Programmiersprache Lua basiert.
  - Daher kann mit einer einfach zu erlernden Programmiersprache direkt auf Torch gearbeitet werden
- http://torch.ch
- Torch gibt es auch für *R*!
  - https://torch.mlverse.org
- Lernkurve ist "steil"

12.6. ML

- Eigenes Framework (Stefan Bosse) dass eine Vielzahl von Lernalgorithmen und Modellen zusammenfasst:
  - Entscheidungsbäume (C4.5, ID3, ICE, usw.)
  - KNN (ANN → Neataptic, MLP → SMO, usw.)
  - Belohnungslernen (Reinforcement und Agenten L.)
  - kNN
  - Clustering (SOM)
  - Textanalyse
  - ...
- Reine JavaScript Implementierung
- Trennung von Daten und Algorithmen (d.h. prozedurale Progarmmierung)
  - Viele Modelle sind portabel, d.h. können direkt mit `JSON.stringify` und `JSON.parse` serialisiert und deseralisiert werden
- Enthalten in bzw. verfügbar für verschiedene Software Frameworks:

- ❏ WorkBook
- ❏ NoteBook (digitale Übungen)
- ❏ JAM (JavaScript Agent Machine)

12.7. Beispiele

/webwork

data

```
var x = [
  [0,0],
  [0,1],
  [1,0],
  [1,1]
]
var y = [
  [0],
  [1],
  [1],
  [0]
]
ML.log(function () { print(inspect(arguments)) })
```

model

```
model=ML.learner({
  algorithm: ML.ML.MLP,
  layers : [2,4,1],
  verbose: 1,
});
```

train

```
ML.train(model,{
  x:x,
  y:y,
  epochs:1000
});
```

test

```
var results=ML.predict(model,x);
print(results)
print(ML.statistics.crossEntropy(ML.math.vec2Mat(results),y));
```

12.8. Zusammenfassung

- ➤ Es wird zwischen Klassifikatoren (kategorische Zielvariablen) und Prädiktorfunktionen (numerische Zielvariablen) unterschieden
  - ❑ Klassifikation mit Prädiktorfunktionen erfordert Kodierung und Dekodierung!
- ➤ Die Fehleranalyse während und nach dem Training kann mit den statistischen Größen "Mean Squared Error" und Kreuzentropie erfolgen
- ➤ Die Konfiguration der inneren Schichten von KNN ist schwierig und häufig ein iterativer Prozess

13. Daten- und Dimensionalitätsreduktion

Datenreduktion ist ein wichtiger Schritt in der Datenvorverarbeitung für ML
Ziel: Reduktion der Datenvariablen (Attribute) → Dimensionalitätsreduktion pro Instanz
Ziel: Reduktion der Dateninstanzen (durch kleine Anzahl von Repräsentanteninstanzen) → Datenvolumenreduktion

13.1. Motivation für Datenreduktion

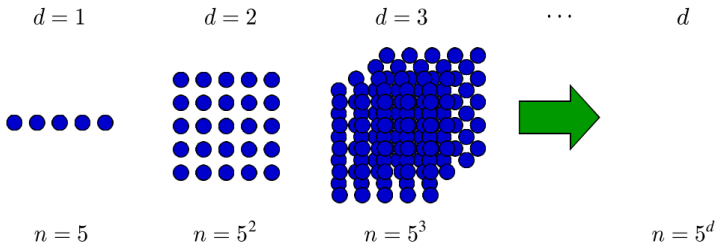
1. Die Daten sind sowohl hinsichtlich Dimensionalität des Eingabevektors $\boldsymbol{x}$ als auch hinsichtlich der Anzahl von Dateninstanzen $|\mathbf{D}|$ sehr groß
2. Es gibt Redundanzen
   - ➤ a. Von Datenvariablen (lineare Abhängigkeit)
   - ➤ b. Von Dateninstanzen (Redundanz und Überlappung)
   - ➤ Aber: Reduktion bei b. kann die geforderte Datenvarianz verschlechtern!

3. Trennung von wenig aussagekräftigen (schwachen) von aussagekräftigen (starken) Variablen

Wenn die Dimensionalität der Eingabedaten $\boldsymbol{x}$ zunimmt, wird jedes Lernproblem immer schwieriger und rechenintensiver!

➤ Beispielsweise werden in regelmäßigen Abständen 5 Punkte von [0,1] abgetastet.

❑ Das sammeln von Proben auf die gleiche Weise im d-dimensionalen Raum erfordert $5d$-Punkte, die exponentiell in Bezug auf d wachsen.



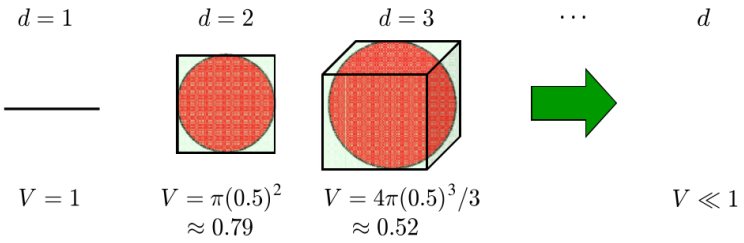
[4]

Ein weiteres Problem bei hochdimensionalen Daten ist, dass unsere geometrische Interpretation von Daten irreführend sein kann.

➤ Betrachtet man zum Beispiel die ausfüllende Hypersphäre des Einheishyperwürfels im d-dimensionalen Raum

❑ Wenn $d = 1$ ist, ist das Volumen einer eingepassten Hypersphäre 1, was dem Hyperwürfel entspricht.

❑ Wenn d auf 2 und 3 erhöht wird, ist das Volumen des Hyperkubus immer noch 1, aber das Volumen der Hypersphäre ist ungefähr 0,79 bzw. 0,52.



[4]

➤ Unsere geometrische Intuition ist falsch,

- ❏ da obwohl die eingepasste Hypersphäre kleiner als der Hyperwürfel ist, ist die Hypersphäre nicht extrem klein.
- ❏ Wenn jedoch d weiter erhöht wird, ist das Volumen des Hyperkubus immer noch 1, aber das Volumen der ausfüllenden Hypersphäre neigt zu 0.
- ❏ Dies bedeutet, dass, wenn d groß ist, die Hypersphäre fast vernachlässigbar ist.

13.2. Verfahrem und Methoden

Lineare Algebra

➤ Bestimmung von Eigenvektoren und Hauptkomponenten mit der **Principle Component Analysis**

- ❏ Abbildung des Eingabedatenraums auf einen niedrigdimensionaleren Datenraum **unter Beibehaltung der Datenrepräsentation**
- ❏ Ziel: Reduktion der Attribute, Beibehaltung der Dateninstanzen

Clustering

➤ Bestimmung von Gruppen von Dateninstanzen (mit geometrischer Gemeinsamkeit durch "Nähe") durch dichtebasierte Clusteringverfahren (**DBSCAN**)

- ❏ Ziel: Reduktion der Instanzmenge durch wenige repräsentative Instanzen; Beibehaltung der Datenvariablen
- ❏ Repräsentative Instanzen können (aber müssen nicht) mehrheitlich die Instanzen der Gruppe mit einem bestimmten Wert der Zielvariable (sofern kategortisch oder intervallkodiert) verknüpfen
- ❏ Hohe Zielvariablenwertdiversität in Gruppen zeigt schwache Korrelation von $\boldsymbol{x}$ mit $\boldsymbol{y}$!

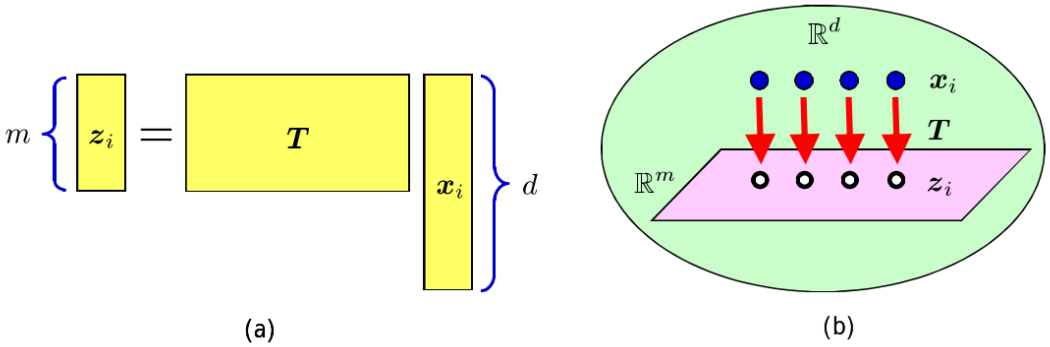
13.3. Lineare Dimensionalitätsreduktion

Lineare Dimensionalitätsreduktion transformiert die ursprünglichen d-dimensionalen Dateninstanzen $\{x_\mathrm{i}\}^\mathrm{n}$ in nieder m-dimensionale Ausdrücke $\{z_\mathrm{i}\}^\mathrm{n}$

durch eine lineare Transformation $\boldsymbol{T}^{m \times d}$

$$z_i = \mathbf{T} x_i$$

➤ $\boldsymbol{T}$ ist die Transformationsmatrix, z der reduzierte Datenvektor (pro Dateninstanz i) und x der ursprüngliche Datenvektor.



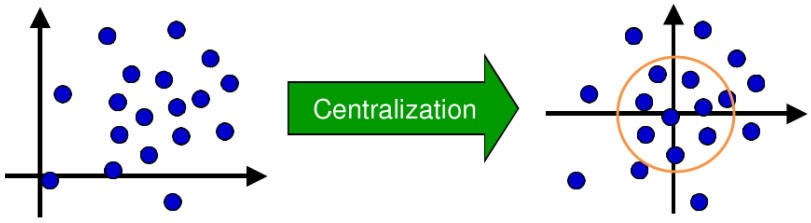
[4]

Abb. 44. (a) Lineare Dimensionalitätsreduktion (b) Projektion von Daten auf einen linearen Subraum

Zentrierung von Daten

➤ Verschiebung der Daten in Richtung "Koordinatenursprung"

$$x_i \leftarrow x_i - 1/n \sum_j x_j$$

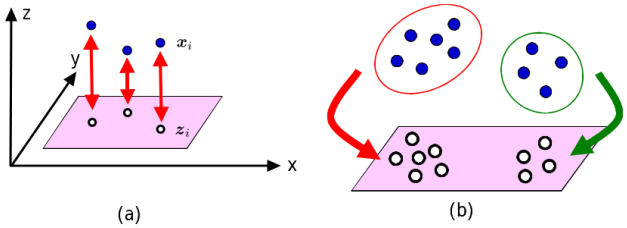


[4]

13.4. Unüberwachte Dimensionsreduktion

Hauptkomponentenanalyse (PCA)

➤ Reduktion der Datendimesionalität unter Beibehaltung der intrinsischen Information der Daten und der Datenstruktur



[4]

Abb. 45. PCA: (a) Reduktion der Dimensionalität $3 \rightarrow 2$ unter Beibehaltung der ursprünglichen geometrischen Eigenschaften der Punkte zueinander (Position) und der Gruppenzugehörigkeit (b)

➤ Das bedeutet dass $\boldsymbol{z}_\mathrm{i}$ eine **orthogonale Projektion** von $\boldsymbol{x}_\mathrm{i}$ ist

❏ D.h. $\boldsymbol{T}\ \boldsymbol{T}^\mathrm{T} = \boldsymbol{I}_\mathrm{m}$ mit $\boldsymbol{I}_\mathrm{m}$ als Identitätsmatrix und a^T die transponierte Matrix

➤ Die "Distanz" zwischen $\boldsymbol{x}$ und $\boldsymbol{z}$ kann aber (als Fehler) nicht unmittelbar bestimmmt werden

❏ Daher wird $\boldsymbol{z}$ wieder in den ursprünglich d-dimensionalen Datenraum durch $\boldsymbol{T}^\mathrm{T}$ zurück transformiert

➤ Die euklidische Distanz ist dann gegeben durch die totale statistische Streumatrix $\boldsymbol{C}$ und der Spur einer Matrix tr:

$$\sum_j ||\mathbf{T}^T\mathbf{T}x_i - x_i||^2 = -tr(\mathbf{TCT}^T) + tr(\mathbf{C})$$

➤ mit:

$$\mathbf{C} = \sum_j x_j x_j^T$$

➤ D.h. PCA ist ein Optimierungsproblem mit:

$$\max_{\mathbf{T}} tr(\mathbf{TCT}^T)$$

- Es gibt eine globale Lösung:

$$\mathbf{T} = (\mathbf{e}_1, ..., \mathbf{e}_m)^T$$

- mit e_i als Eigenvektor der Matrix $\boldsymbol{C}$ mit den Eigenwerten $_1 \quad _2 \quad .. \quad _d$ 0.

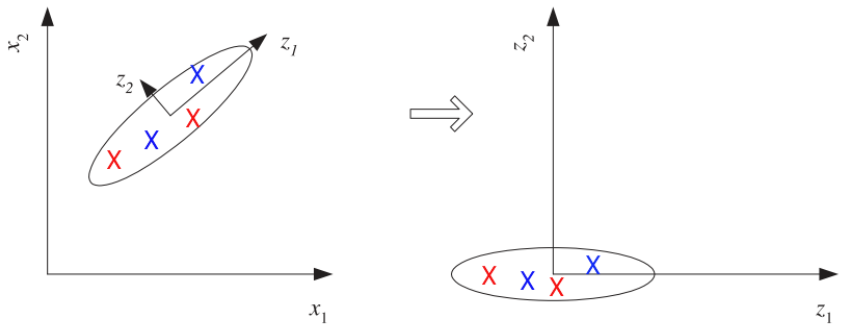
- D.h. es gilt:

$$\mathbf{Ce} = \lambda \mathbf{e}$$

- Es wird "kleine" und "große" Eigenvektoren geben

Die Transformationsmatrix $\boldsymbol{T}$ der PCA ist also eine orthogonale Projektion in einen Subraum der durch die "großen" Eigenvektoren aufgespannt wird,

- Die kleinen Eigenvektoren werden daher entfernt
- Und: PCA transformierte Variablen sind unkorreliert!
- PCA liefert aber i.A. keine Struktureigenschaften wie Cluster



[17]

Abb. 46. Weiteres Beispiel von PCA: Die Analyse der Hauptkomponenten zentriert die Dateninstanzen und dreht dann die Achsen, um Sie mit den Richtungen der höchsten Varianz in Einklang zu bringen. Wenn die Varianz auf z_2 klein ist, kann Sie ignoriert werden und wir haben eine Dimensionalitätsreduktion von zwei auf eins.

13.5. ML.pca

- ➤ Das ML Framework stellt ein PCA Modul zur Verfügung:
- ➤ Die Daten $\boldsymbol{D}$ müssen in Matrixform vorliegen (keine Recordtabellen)
- ➤ Eigenvektoren berechnen

```
ML.pca.getEigenVectors = function(data:number [][]) → {
  eigenvalue: number, vector: number []
} [];
```

2. Transformation der Datentabelle berechnen

```
ML.pca.computeAdjustedData =
  function (data:number [][],
            eigenVector1:{},eigenVector2?:{},..)
→ {
  adjustedData: number [][],
  formattedAdjustedData: number [][],
  avgData: number [][],
  selectedVectors: number [][]
}
```

- ➤ Achtung: Das Datenformat von *adjustedData* ist transponiert und muss für eine Tabelle rekonstruiert werden

3. Rekonstruktion der Datentabelle aus der reduzierten Tabelle

```
ML.pca.computeOriginalData = function(
  formattedAdjustedData,
  selectedVectors,
  avgData) → {
  formattedOriginalData,
}
```

13.6. PCA Beispiel

/webwork

data

```
data = [
  {
    "length":5.1,
    "width":3.5,
    "petal_length":1.4,
    "petal_width":0.2,
    "species":"setosa"
  },
  {
    "length":4.7,
    "width":3.2,
    "petal_length":1.3,
    "petal_width":0.2,
    "species":"setosa"
  },
  {
    "length":4.6,
    "width":3.1,
    "petal_length":1.5,
    "petal_width":0.2,
    "species":"setosa"
  },
  {
    "length":5,
    "width":3.6,
    "petal_length":1.4,
    "petal_width":0.2,
    "species":"setosa"
  },
  {
...
```

eigen

```
dataX=ML.preprocess(data,'m',
 {features:
    ['length','width','petal_length','petal_width']
}).data;
eigenX=ML.pca.getEigenVectors(dataX);
```

adjust

```
dataZ=ML.pca.computeAdjustedData(dataX,eigenX[0],eigenX[1]).adjustedData;
```

reconstruct

```
dataXR=ML.pca.computeOriginalData(
  dataZ.formattedAdjustedData,
  dataZ.selectedVectors,
  dataZ.avgData).formattedOriginalData;
```

13.7. Lokalitätsbewahrende Projektion

➤ Ähnlichkeit zwischen Instanzen $\boldsymbol{x}_\mathrm{i}$ und $\boldsymbol{x}_\mathrm{j}$ wird untersucht und mit einem Wert 0 $W_{\mathrm{i,j}}$ 1 ausgedrückt.

Ähnlichkeitsfunktionen

Gaußfunktion

$$W_{i,j} = exp(-\frac{||x_i - x_j||^2}{2t^2})$$

(t ist ein Anpassungsparameter)

k-Nächster Nachbar (kNN)

$$W_{ij} = \begin{cases} 1, (x_i \in N_k(x_j) \vee x_j \in N_k(x_i)) \\ 0 \end{cases}$$

➤ $N_\mathrm{k}(\boldsymbol{x})$ ist die Menge aller Nachbarschaftsinstanzen (k ist die Anzahl der Menge)

➤ k ist ein Anpassungsparameter der die Lokalität einstellt (Reichweite)

Transformation

$$\min_T \sum_{i,j} W_{ij} ||\mathbf{T}x_i - \mathbf{T}x_j||^2$$

➤ Problem: $\boldsymbol{T}$=0 ist eine Lösung, aber nutzlos!

➤ Daher weitere Randbedingung für die Minimierung (Anpassung von $\boldsymbol{T}$):

$$\mathbf{TXDX}^T\mathbf{T}^T = \mathbf{I}_m$$

$$\mathbf{X} = (x_1, .., x_n), \mathbf{D}_{ij} = \begin{cases} \sum_k W_{ik}, (i = j) \\ 0, (i \neq j) \end{cases}$$

13.8. Dichtebasiertes Clustering

- ➤ kNN kann ebenso als ein ML Clusterer mit Inferenz auf unbekannten eingesetzt werden
- ➤ Dichtebasierte Clusteringmethoden können primär für die Datenreduktion eingesetzt werden
  - ❑ Reduktion von einer Menge von Dateninstanzen auf Gruppen (wobei die Instanzen erhalten bleiben, aber in jeder Gruppe reduziert werden können)

DBSCAN

Dichtebasiertes Räumliches Clustering mit Rauschen (DBSCAN) ist ein Basisalgorithmus. Es kann Cluster unterschiedlicher Formen und Größen aus einer großen Datenmenge entdecken, die Rauschen und Ausreißer enthält.

Algorithmische Schritte für DBSCAN Clustering

- ➤ Der Algorithmus iteriert über alle Punkte indem er willkürlich einen Punkt im Datensatz aufnimmt (bis alle Punkte besucht wurden)
- ➤ Wenn es mindestens *minPoint* - Punkte in einem radius von bis zu dem Punkt gibt, werden alle diese Punkte als Teil desselben Clusters betrachet.
- ➤ Die Cluster werden dann erweitert, indem die Nachbarschaftsberechnung für jeden Nachbarpunkt rekursiv wiederholt wird.
- ➤ Eventuell gibt es Beschränkungen bezüglich der Dimension der Dateninstanzen (typisch 2: Clustering von Bildpunkten)

Wahl der Parameter

Jede Data Mining Aufgabe hat das Problem der Parameterwahl.

➤ Jeder Parameter beeinflusst den Algorithmus auf bestimmte Weise. Für DBSCAN werden die Parameter und *minPts* benötigt.

minPts

Das Minimum von *minPts* kann aus der Dimension des Datensatzes ***D** abgeleitet werden, so dass minPts D + 1 gilt.

Der Wert für kann dann unter Verwendung eines k-Entfernungsgraphen ausgewählt werden, der den Abstand zum $k = minPts$-1 nächsten Nachbarn zeichnet, der vom größten zum kleinsten Wert geordnet ist.

➤ Bei guten Werten von gibt es eine ausgewogene Verteilung der Dateninstanzen in Gruppen

- ❏ Wenn viel zu klein gewählt wird, wird ein großer Teil der Daten nicht gruppiert, während für
- ❏ einen zu hohen Wert von Cluster zusammengeführt werden und sich die Mehrheit der Objekte im selben Cluster befinden.

Aufgrund minPts Randbedingung werden u.U. nicht alle Instanzen gruppiert!

Beispiel DBSCAN

/webwork

data

```
data = [
  {
    "length":5.1,
    "width":3.5,
    "petal_length":1.4,
    "petal_width":0.2,
    "species":"setosa"
  },
  {
    "length":4.7,
    "width":3.2,
    "petal_length":1.3,
    "petal_width":0.2,
    "species":"setosa"
  },
  {
    "length":4.6,
    "width":3.1,
    "petal_length":1.5,
    "petal_width":0.2,
    "species":"setosa"
  },
  {
    "length":5,
    "width":3.6,
    "petal_length":1.4,
    "petal_width":0.2,
    "species":"setosa"
  },
  {
...
```

pca1

```
dataX=ML.preprocess(data,'m',
 {features:
   ['length','width','petal_length','petal_width']
}).data;
eigenX=ML.pca.getEigenVectors(dataX);
```

pca2

```
dataZ=ML.pca.computeAdjustedData(dataX,eigenX[0],eigenX[1]).adjustedData;
```

clust1

```
dataZ2=dataZ[0].merge(dataZ[1],'c')
dbscan=new ML.DBCLUST.DBSCAN();
```

clust2

```
eps=0.3;
clusters=dbscan.run(dataZ2,eps,10);
```

13.9. Zusammenfassung

- ➤ Datenreduktion ist ein wichtiger Schritt in der Datenvorverarbeitung
  - ❏ Datenreduktion bedeutet die Reduktion der Datenmenge und/oder ihrer Dimensionalität
  - ❏ Datenreduktion ist bereits eine Merkmalsselektion (Reduktion auf wesentliche Attribute und Werte)
- ➤ PCA ist geeignet um numerische Eingabedaten in ihrer Dimensionalitä zu reduzieren (Reduktion der Datenvariablen und Ersatz mit transformierten)
- ➤ DBSCAN ist geeignet um Gruppen von Dateninstanzen zu finden um schliesslich repräsentative Instanzen daraus zu ermitteln

14. Support Vector Machines

SVM gehören zu den Regressionsverfahren
SVM nutzen aber bei der Parameteranpassung (Training) eine andere Fehlerfunktion (Loss) als bei anderen gängigen Regressionsverfahren (z.B. LS Minimierung)
SVM können primär kategorische und weniger numerische Zielvariablen verwenden
SVM sind (zunächst) lineare Klassifikatoren!

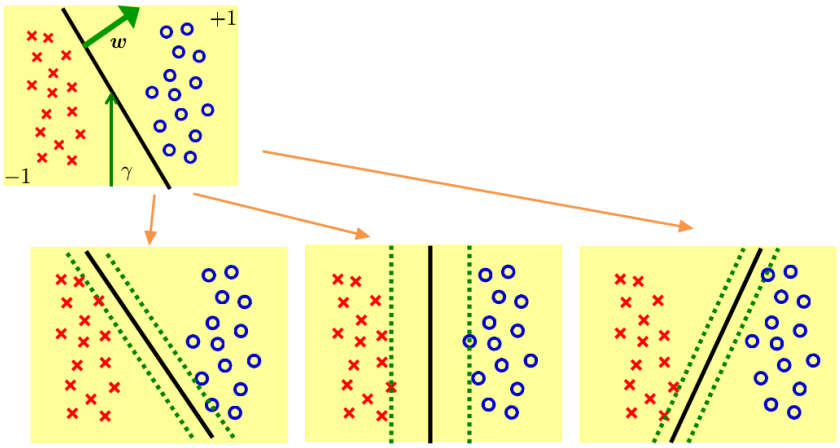
14.1. Binärer Klassifikator

➤ Ein binärer Klassifikator soll durch eine **lineare Funktion** repräsentiert werden (y={-1,+1}):

$$f(\tilde{x}) : \tilde{x} \rightarrow y = \tilde{w}^T \tilde{x} + \gamma$$

Dabei sind $\boldsymbol{w}$ und die Parameter des Modells die durch das Training an das Problem angepasst werden müssen.

➤ $\boldsymbol{w}$ ist ein Normalenvektor der bei einem binären Klassifikationsproblem die beiden Instanzklassen trennt



[4]

Abb. 47. (Oben) $\boldsymbol{w}$ ist der Normalenvektor und die Verschiebung der Trennungsgrenze für zwei Klasseninstanzen (Unten) Verschiedene $\boldsymbol{w}$/ Varianten der Trennungsgrenze mit unterschiedlichen Rändern (Sicherheitsbereichen)

14.2. Training

➤ Das Lernen von $\boldsymbol{w}$ und erfordert die Berechnung des Abstandes von allen Dateninstanzen von der Trenngrenze. Die Abstände müssen positiv sein:

$$f(\tilde{x}_i)y_i = (\tilde{w}^T \tilde{x}_i + \gamma)y_i > 0, \forall i$$

für alle Dateninstanzen $\boldsymbol{D}$={$(\boldsymbol{x}_\mathrm{i}, y_\mathrm{i})$}$^\mathrm{n}$.

➤ Da $\boldsymbol{w}$ und beliebig gewählt werden können, kann die Randbedingung auch mit (..)y_i 1 gewählt werden.

➤ Weiterhin kann es sinnvoll sein alle Dateninstanzen um den Ursprung des Koordinatensystems zu **zentrieren**

Wichtig: Die Werte für y liegen im Intervall [-1,1]!

➤ Alle Probleme die (..)y_i 1 erfüllen mit einem ($\boldsymbol{w}$,) sind linear separierbar.

➤ Es gibt unendlich viele Lösungen (also Entscheidungsgrenzen)

➤ Man wählt das ($\boldsymbol{w}$,) aus bei der alle Dateninstanzen die größte Trennung besitzen (breitester Trennbereich, siehe Abb.)

➤ Der Abstand der Dateninstanzen $\boldsymbol{D}$ ist definiert als das Minimum des normalisierten Abstandes:

$$m_i = (\tilde{w}^T \tilde{x}_i + \gamma) y_i / ||\tilde{w}||$$

➤ D.h. SVM zu trainieren ist das Minimierungsproblem:

$$\min_i \frac{(\tilde{w}^T \tilde{x}_i + \gamma) y_i}{||w||} = \frac{1}{||w||}$$

Harter Trennungsbereich (Hard SVM)

➤ Bei einer "harten" Trennung einer SVM gilt dann:

$$\min_i 1/2||w||^2, (\tilde{w}^T \tilde{x}_i + \gamma) y_i \geqslant 1, \forall i$$

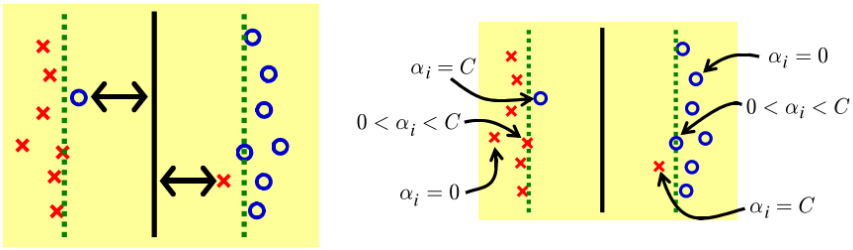
14.3. Weicher Trennungsbereich (Soft SVM)

➤ Die SVM mit harten Trennungsbereich erfordert lineare Separierbarkeit der Dateninstanzen

❑ Ist in der Realität aber nicht immer oder eher selten gegeben

➤ Die weiche Trennung durch eine SVM führt einen Fehlerparametervektor $=\{\ _i\}^n$ für die Bestimmung des Trennbereichs ein:

$$\min_{\forall i: w, \xi, \gamma} \left[1/2||w||^2 + C \sum_i \xi_i \right],$$
$$(\tilde{w}^T \tilde{x}_i + \gamma) y_i \geqslant 1 - \xi_i, \xi_i \geqslant 0, \forall i$$



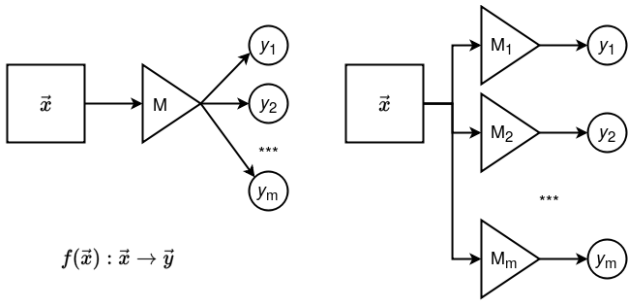
[4]

Abb. 48. Weicher Trennbereich einer SVM (Soft margin SVM). Durch werden kleine Klassifikationsfehlerbereiche erlaubt.

14.4. Multiklassen SVM

Jedes Multiklassenproblem mit m verschiedenen (diskreten) Klassenwerten kann auf m binäre Klassifikationsprobleme transformiert werden

➤ Anders als bei ANN ist bei SVMs aber nur eine One-hot Kodierung möglich (ggfs. mit Softmaxfunktion).



14.5. ML Framework API für SVM

➤ Higher precision tol : number, // default : 20. Higher max_passes –> Higher precision max_passes : number, // default : 1e-5. Higher alpha_tolerance –> Higher precision alpha_tol : number, kernel : { type: 'rbf', sigma: 0.5 } // { type: "polynomial", c: 1, d: 5} }) → model '

14.6. Beispiel

/webwork

data

```
data = [
  {
    "length":5.1,
    "width":3.5,
    "petal_length":1.4,
    "petal_width":0.2,
    "species":"setosa"
  },
  {
    "length":4.7,
    "width":3.2,
    "petal_length":1.3,
    "petal_width":0.2,
    "species":"setosa"
  },
  {
    "length":4.6,
    "width":3.1,
    "petal_length":1.5,
    "petal_width":0.2,
    "species":"setosa"
  },
  {
    "length":5,
    "width":3.6,
    "petal_length":1.4,
    "petal_width":0.2,
    "species":"setosa"
  },
  {
...
```

prepro

```
// {}[] -> x:number[][], y:string [][]
dataN = ML.preprocess(
  data,
  'xmy',
  {
    features:['length','width','petal_length','petal_width'],
    target:['species']
  }
)
// Kodierung der y-Variable
dataN.y = dataN.y.map(function (row) {
  switch (row[0]) {
    case 'setosa': return [-1,-1,1];
    case 'versicolor': return [-1,1,-1];
    case 'virginica': return [1,-1,-1];
  }
});
models=[]
results=[]
```

train

```
var target = 0; // 0,1,2
models[target] = ML.learn({
    algorithm:ML.ML.SVM,
    x:dataN.x,
    y:dataN.y.pluck(target),
    threshold:false,  // no threshold function on output; highest value of svms is winner
    C : 15.0, // default : 1.0. C in SVM.
    tol : 1e-5, // default : 1e-4. Higher tolerance --> Higher precision
    max_passes : 200, // default : 20. Higher max_passes --> Higher precision
    alpha_tol : 1e-5, // default : 1e-5. Higher alpha_tolerance --> Higher precision
    kernel : { type: 'rbf', sigma: 0.5 } // { type: "polynomial", c: 1, d: 5}
})
```

test

```
var target = 0; // 0,1,2
results[target]=ML.predict(models[target],dataN.x).merge(dataN.y.pluck(target),'c')
```

[Sugiyama, ItSML, pp 303]

14.7. Zusammenfassung

- ➤ Eine SVM wird als binärer Klassifikator verwendet und wird i.A. durch eine lineare Funktion (Kernel) repräsentiert
  - ❑ Das Problem sollte dann linear separierbar sein!
- ➤ Multiklassenprobleme werden auf Multi-SVMs zurückgeführt
  - ❑ Verwendung einer Softmax Funktion für eindeutige Klassentrennung
- ➤ Das Training einer SVM ist ein Minimierungsproblem dass den Trennbereich maximiert und den Fehler minimiert

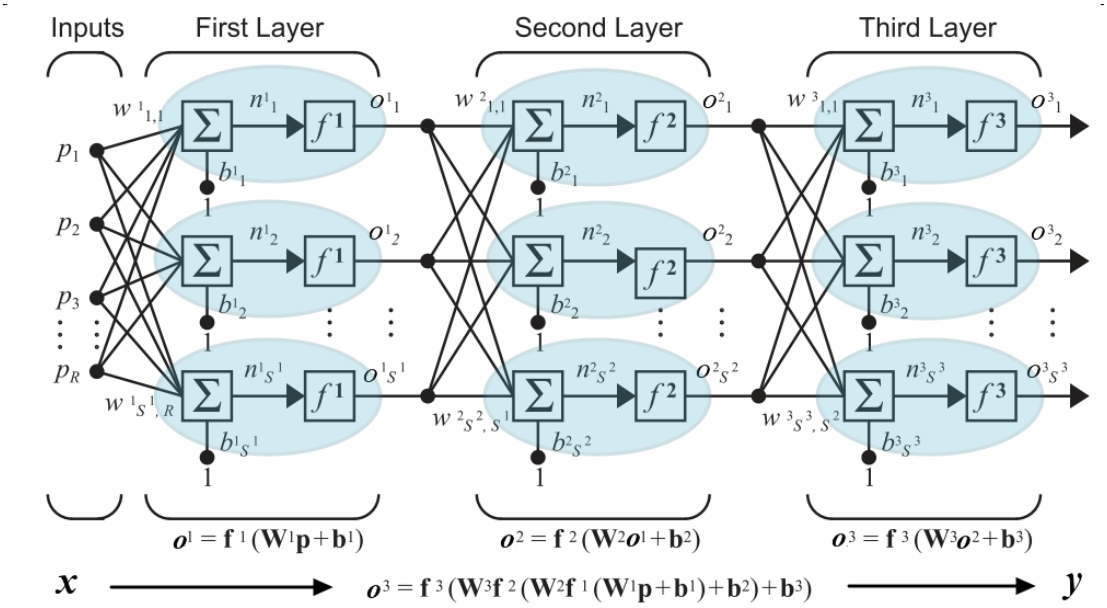
15. Zustandsbasierte Netze

Bisher wurden nur "vorwärtsgerichtete" ANN betrachtet (Feed forward/ FF-ANN)
Die Ausgänge von FF-ANN hängen nur von aktuellen Eingängen ab!
Rückgekoppelte ANN (Recurrent / RNN) besitzen Zusatnd und Gedächtnis!
Eignung für Daten- und Zeitserienprädiktion!

15.1. Wiederholung: FF-ANN

- ➤ Ein vorwärtsgekoppeltes ANN (Feed forward) besteht aus Neuronen die jeweils durch Funktionen $f(\boldsymbol{i})$:$\boldsymbol{i} \rightarrow o$ repräsentiert werden können.
- ➤ Die Funktionen f werden als azyklischer gerichteter Graph dargestellt (also das NN), d.h., kein Neuron hat ein Eingangssignal von nachfolgenden Neuronen (die das Ausgangssignal dieses Neurons enthalten)

D.h: Das aktuelle Ausgangssignal $\boldsymbol{y}$ eine FF-ANN hängt nur von den aktuellen Eingangssignalen $\boldsymbol{x}$ ab!



15.2. Recurrent ANN: Die Rückkopplung

- Ausgänge von Ausgangs- oder inneren Neuron werden auf Eingänge von vorherigen Neuronen gekoppelt
- Dadurch werden die Netzwerke zustandsbasiert, d.h., die aktuellen Ausgänge hängen neben den aktuellen Werten der Eingänge von der Historie der Werte von Ein- und Ausgängen ab!
- Recurrent ANN (RNN) sind durch hohe "Instabilität" schwierig mit gradientenbasierten Verfahren zu trainieren!
- Daher im Laufe der Zeit verbesserte Architekturen wie LSTM oder GRU Netzwerke

15.3. LSTM Netzwerke

- Long-short-term Memory (LSTM) Netzwerke bilden eine bekannte Architektur die:
  - Für die Daten- und zeitserienprädiktion verwendbar sind
  - Mit gradientenbasierten Trainingsverfahren einigermaßen stabil (konvergent) trainiert werden können

Daten- und Zeitserien

➤ Es sei $\{x_n\}^t$ eine Serie von Eingabedaten (z.B. zeitaufgelöstes Sensorsignal einer Ultraschallmessung), d.h. $\{x_n\}=\{x_1,x_2,..,x_t\}$

➤ Die einzelnen Werte (skalar oder vektoriell) sind aufeinanderfolgend → **Ordnungsrelation**

$$f_\delta(\{s_i\}_i^n, n) : s_{n-m}, s_{n-m+1}, ..s_n \rightarrow s_{n+\delta}$$

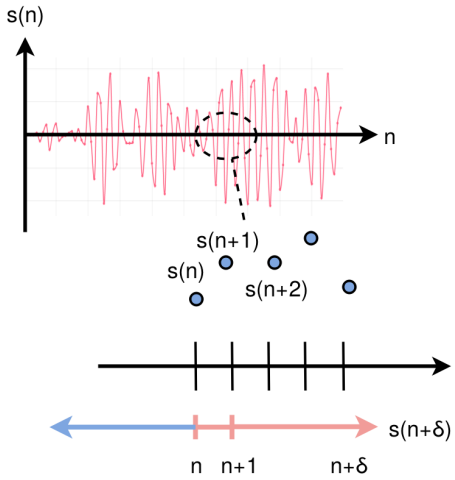


Abb. 49. Datenserie und Prädiktion $s(n+\ )$

➤ Neben der Prädiktion von "zukünftigen" Entwicklungen einer Variable x_i kann auch eine zukünftige Prognostik einer anderen Zielvariable y_i erfolgen:

$$f_\delta(\{x_i\}_i^n, n) : \{x_{n-m}, x_{n-m+1}, ..x_n\} \rightarrow y_{n+\delta}$$

➤ Etwaige Zielvariablen könnten aus dem Signal abgeleitete Eigenschaften (Merkmale) sein:

- ❏ Schadensinformation
- ❏ Änderung einer zweiten Sensorvariable
- ❏ Zustandsvariable
- ❏ Veränderung von Betriebsbedingungen
- ❏ Prozessparameter

15.4. Netzwerkarchitekturen

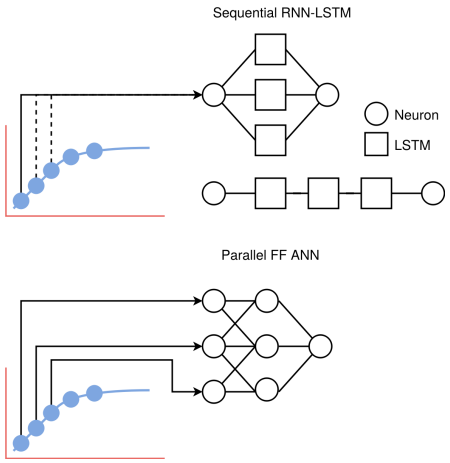


Abb. 50. (Oben) Die Datenserie kann sequentiell (in ein RNN/LSTM) oder auch (Unten) parallel in ein FNN eingegeben werden

15.5. LSTM Zelle

- Eine LST Zelle besteht aus:
  - Einer "Speicherzelle"
  - Mehreren Gattern die aktiv Verbindungen (Kanten) zwischen Neuronen steuern, d.h., effektiv das Kantengewicht in Abhängigkeit von Ausgangswerten von Neuronen verändern /Ventile/
- Ein wichtige Rolle bei der Speicherung vergangener Daten (Gedächnis) ist das Vergessen gesteuert durch das "Forget Gate"

Beispiel für eine "gedämpfte" Gedächnisfunktion: Ein Tiefpassfilter 1. Ordnung

$$f(x,n) = f(n-1)\cdot\alpha + \beta\cdot x$$

- Der Parameter =(0,1) bestimmt den Einfluß neuer Werte x und der Parameter =(0,1) bestimmt den Einfluß alter Werte (Vergessen); i.A. =1-

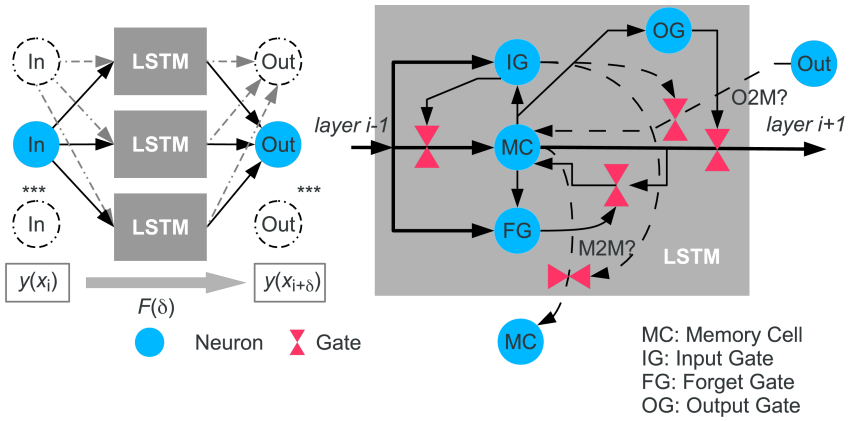


Abb. 51. Aufbau einer LSTM Zelle mit zentraler Speicherzelle (Neuron) und den Gattern

15.6. LSTM Demo

- Lernen einer Prädiktorfunktion für Datenserien nach der Sinusfunktion mit LSTM Netzwerk:

$$f_\delta(x_n) : x_n \rightarrow x_{n+\delta}$$

- Für jedes $=\{1,2,..,m\}$ muss eine eigene Prädiktorfunktion f erzeugt und trainiert werden
- Konfiguration des Netzwerks: $[1,l,1]$, wobei $l=\{1,2,..\}$ die Anzahl der LSTM Zellen in der inneren Schicht ist.

15.7. Zusammenfassung

- Diskrete Daten- und Zeitserien $x(i)$ kommen in der Mess- und Prüftechnik als Sensorsignal häufig vor, aber auch bei
  - Wirtschaftsdaten (Aktienkurse), Wetterdaten, Klimadaten, usw.
- Die Vorhersage von unbekannten zukünftigen Datenwerten aus historischen bekannten Datenwerten erfordert **zustandsbasierte Prädiktorfunktionen** sofern es
  - keinen ausgewiesenen Start- und Endpunkt gibt (also **sequentielle Aktivierung** von M).

- ❑ Gibt es ausgewiesene Start- und Endpunkte der Serie kann auch eine zustandslose Prädiktorfunktion verwendet werden (**parallele Aktivierung** einer Auswahl von Datenpunkten)

- ➤ Die Long Short Term Memory Zelle ist eine bekannte Architektur für eine RNN PF mit Zustand (Speicher)
- ➤ Problem beim Trainieren von RNN ist Instabilität und der Zustandsspeicher (wirkt sich auf Fehlerberechnung ungünstig aus)

16. Rechnerarchitekturen für ML

Die Software für Implementierung der Modelle und Algorithmen ist stark verknüpft mit der Rechnerarchitektur
Performanzprobleme vor allem bei Training, aber auch bei Inferenz!
Parallelisierung kann die Performanz steigern
***Effizienz**: Optimierung von Rechenzeit + Speicherbedarf + Energie!*

16.1. Rechnerklassifikation

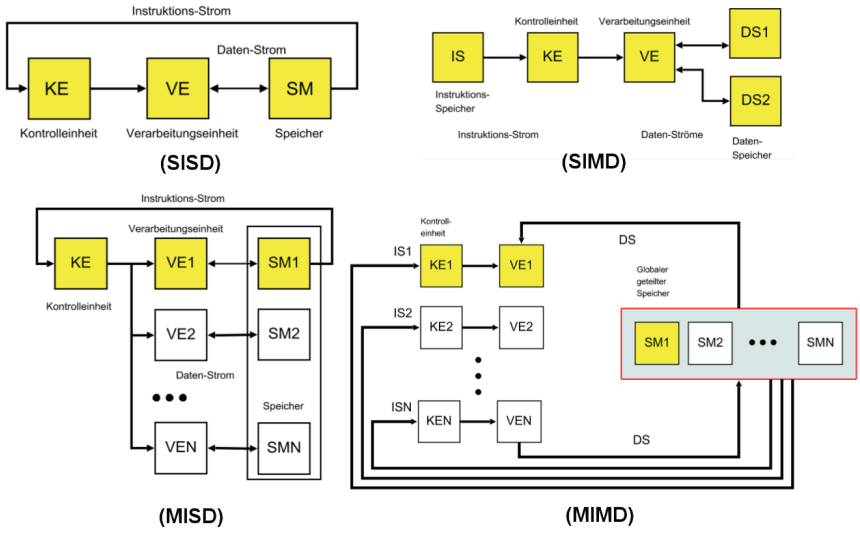
- ➤ Datenverarbeitung bedeutet die Verarbeitung eines Daten- und eines Instruktionsstromes (DS/IS)
- ➤ Eine Datenverarbeitungsanlage enthält:

VE
: Verarbeitungseinheit für Daten

SM
: Speichermodul → (Static/Dynamic) RAM, Registerbank, Cache

KE
: Kontrolleinheit für die Ablaufsteuerung, kann in VE enthalten sein.

- ➤ Dabei sind diese drei Einheiten über **Instruktionsströme** und **Datenströme** gekoppelt:



➤ Man kann grob Rechnerarchitekturen nach der Art der Verarbeitung des DS/IS und möglicher Parallelisierung klassifizieren **Flynn** Klassifikation

SISD

Single-Instruction Single-Data Stream

SIMD

Single-Instruction Multiple-Data Stream

MISD

Multiple-Instruction Single-Data Stream

MIMD

Multiple-Instruction Multiple-Data Stream

16.2. Funktionale Dekomposition

➤ Die Modelle lassen sich je nach Architektur und Aufbau funktional darstellen → Funktionale Komposition mit Datenfluss

Neuronales Netzwerk

Jedes Neuron N besteht aus zwei gekoppelten Funktionen Σ als Summierer und f als Transferfunktion:

$$N(x_1, x_2, .., x_n) = f(\Sigma_j w_j x_j)$$

Das Netzwerk kann als eine große Funktion $M(\mathbf{x})$ aus Funktionen (Neuronen

einer Ebene sind zu einer Funktion L zusammengefasst) gebaut werden:

$$M(\tilde{x}) = L_m(L_{m-1}(...(L_0))), L_i = \{N_{i,1}, N_{i,2}, ..\}$$

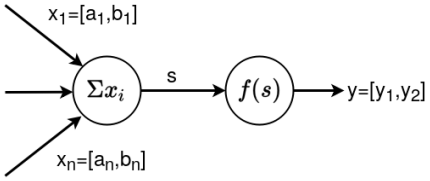


Abb. 52. Funktionale Dekomposition eines SLP (Neuron N)

Die Berechnung aller $\{N_i\}$ Funktionen in einer Ebene L_i kann parallel erfolgen!

Entscheidungsbäume

Auch ein Entscheidungsbaum kann funktional dargestellt werden durch geschachtelte konditionale Ausdrücke:

$$N(x_i) = \begin{cases} v_1, x_i < \varepsilon \\ v_2, x_i \geqslant \varepsilon \end{cases}$$

$$M(\tilde{x}) = N(N_{x1}(N_{x3}(..), N_{x4}(..)), N_{x2}(..))$$

16.3. Matrixalgebra

- ➤ Parallelisierung ist gut auf Probleme mit “Schleifeniterationen” anwendbar
- ➤ Matrixalgebra zeigt inherente Parallelität in Matrixoperationen (häufig sequenziell durch Schleifeniteration verabeitet)

Neuronale Netze (i.A. mit regulärer Struktur) können durch Matrixalgebra abgebildet werden

16.4. Parallelisierung

Parallelisierung erfolgt auf der Daten- und Kontrollpfadebene
Kontrollpfadparallelität *bedeutet Paritionierung einer Berechnung in separate Prozesse → Multiprocessing & Multithreading → Mehrprozessorechner (MCPU)*
Datenpfadparallelität *bedeutet die Belegung mehrerer Verarbeitungseinheiten (Rechenwerke) → GPU, FPGA, (CPU:MMX)*

Inferenz

Bei Entscheidungsbäumen kann der Baum nur sequenziell berechntet werden (höchstens spekulativ mit parallelen Teilbäumen). Aber es gibt auch Multibäume (Random Forest). Diese können parallel berechnet werden.Bei neuronalen Netzen können alle Neuronenfunktionen einer Ebene parallel berechnet werden.

Training

Bei Entscheidungsbäumen (Monobäumen) ist nur bei der Datenanalyse eine Parallelisierung möglich. Aber es gibt auch Multibäume (Random Forest). Diese können parallel erzeugt werden (mit gemeinsamer Datenanalyse).Bei neuronalen Netzen ist Parallelisierung auf die Matrixalgebra anwendbar.

16.5. Berechnungskomplexität

Inferenz

Bei Entscheidungsbäumen hängt die Berechnungskomplexität nur noch von der Höhe des Baums (Anzahl der Ebenen) ab → Selektiv aktivierbare Teilberechnung (trivial, Berechnung einfacher Ausdrücke).Bei neuronalen Netzen müssen i.A. alle Funktionen berechntet werden (Komplexität ist noch klein) und die Berechnung muss Datenabhängigkeiten beachten.

Training

Bei Entscheidungsbäumen müssen die Datenspalten vs. Parititionierung der Zielvariablen analysiert werden (also Rechenzeit |D||X|).Bei neuronalen Netzen findet zunächst eine vollständig Berechnung aller Neuronenfunktionen statt (für alle Datensätze also Rechenzeit |D||NN|) und anschliessend werden alle Gewichte mit Fehlerfunktionen neu berechnet!|D|: Anzahl der Dateninstanzen, |X|: Anzahl der Eingabevariablen, |NN|: Anzahl der Neuronen im KNN

16.6. Approximation

➤ Tatsächlich ist jede Berechnung mit einem Digitalrechner eine Approximation!

- ❏ Reelle Zahlen können nicht mit Digitalrechnern verarbeitet werden
- ❏ Digitalrechner können nur ganze Zahle verarbeiten
- ❏ Fliesskommaarithmetik ist tatsächlich Granzzahlrechnung mit einem "verschobenen" Komma

➤ Unterschiedliche Datentypen und Rechenwerke benötigen unterschiedlichen Speicherbedarf

- ❏ *Daten*: Float64 → 64 Bit / 8 Byte, *Rechenwerk (ALU)*: > 1M Logikgatter
- ❏ *Daten*: Int16 → 16 Bit / 2 Byte, *Rechenwerk (ALU)*: 10k Logikgatter

➤ Bei der Rechenzeit / Operation gab es früher großen Unterschied t_{op}(Float) t_{op}(Integer), heute gilt in modernen CPUs aber t_{op}(Float) t_{op}(Integer)

➤ Bei der Berechnung, z.B. Matrixalgebra, spielt aber in heutigen Rechnerarchitekturen der Datenfluss die Hauptrolle → Speicherzugriff ist langsam!

Parallelisierungsgrad P

Die maximalze Anzahl von binären Stellen (bits) pro Zeiteinheit (Taktzyklus) die von einer Datenverarbeitungsanlage verarbeitet werden kann.

➤ Es gilt: $P = W \bullet B$

Wortlänge W

Die Wortlänge oder Wortbreite gibt die Anzahl der Bits eines Datenpfades an.

Bitslice-Länge B

Die Bitslice-Länge setzt sich zusammen aus der Anzahl von Verarbeitungseinheiten VE, die parallel ausgeführt werden können, und der Anzahl der Pipeline-Stufen einer VE.

➤ Es gilt: $B = N_{\text{VE}} \bullet N_{\text{Stages}}$

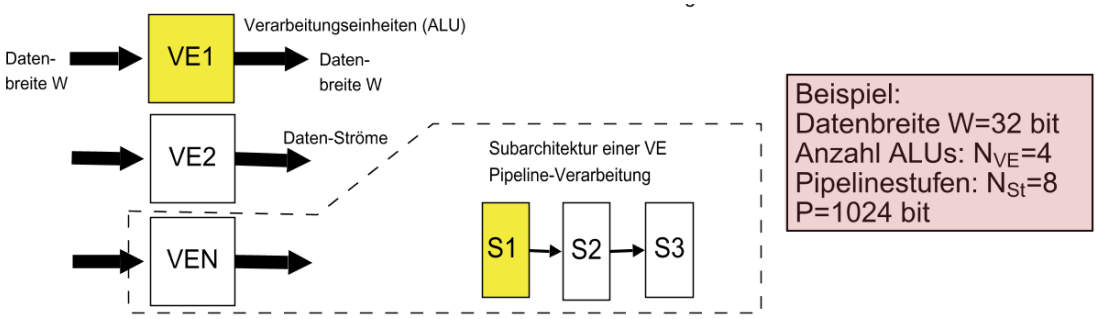
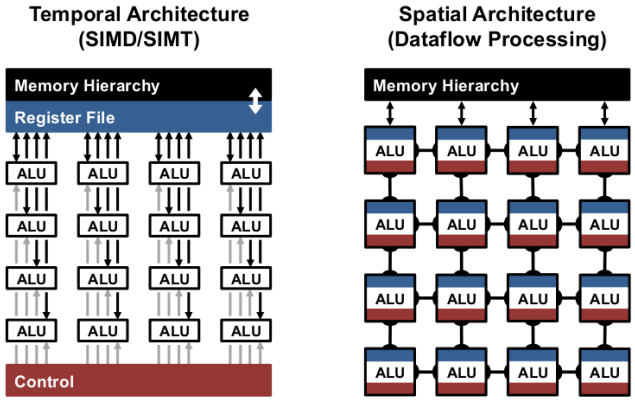


Abb. 53. Illustration Parallelisierungsgrad und Beispiel

Die Reduktion der Datenwortbreite bei Berechnungen bringt eine Steigerung der Verarbeitungsgeschwindigkeit wenn Datendurchsatz oder Ressourcen limitierende Faktoren sind!
Aber Reduktion der Datenwortbreite bedeutet approximatives Rechnen.

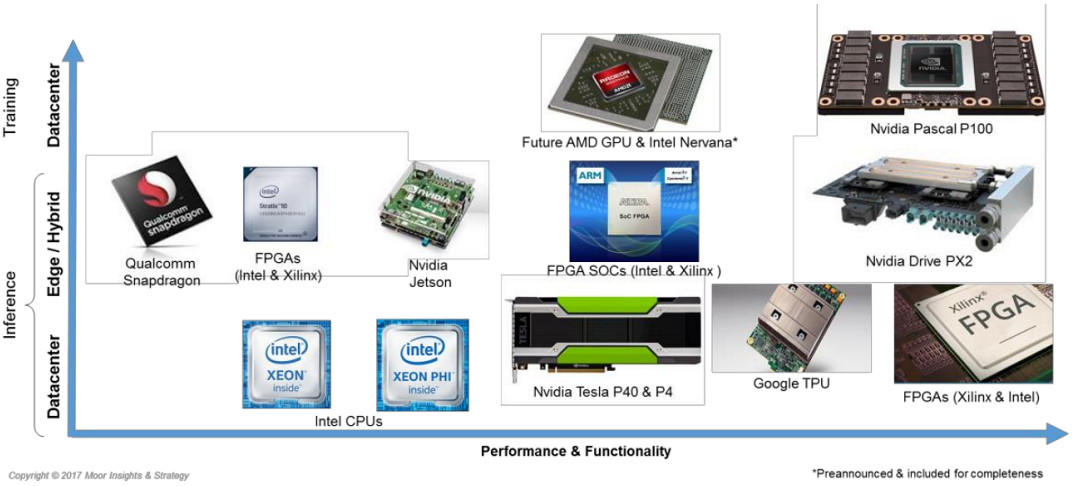
16.7. Hochparallele Verarbeitungsarchitekturen



[Sze, Hardware for Machine Learning: Challenges and Opportunities]

Abb. 54. Fein granulierte parallele DV auf Datenpfadebene mit Multi-ALU Architekturen

16.8. MACHINE LEARNING COMPUTE PLATFORMS



(Source: Moor Insights & Strategy)

[Moor I&S, A MACHINE LEARNING APPLICATION LANDSCAPE]

Abb. 55. Metrik der Berechnungsplattformen für ML (CPU, GPU, DSP, FPGA, ASIC)

16.9. Generische Prozessoren und Rechner

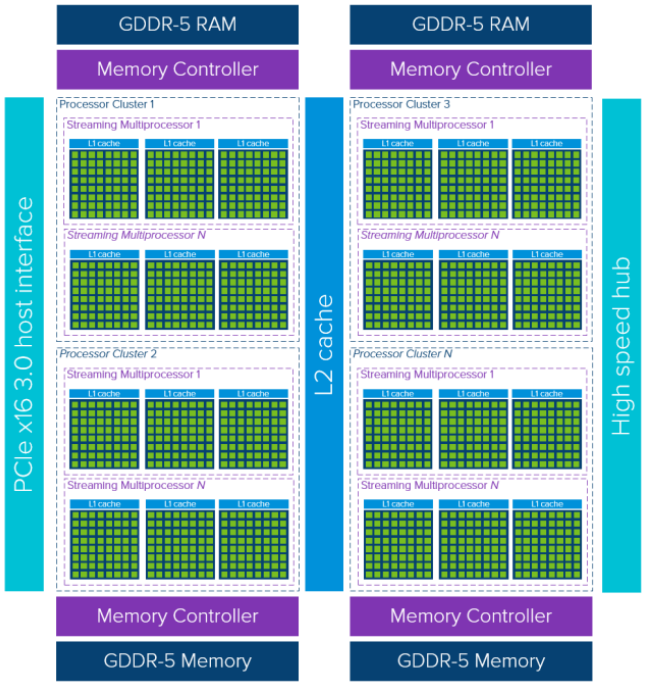
- ➤ Berechnungen finden sequenziell statt:
  - ❑ Es gibt i.A. nur eine ALU (Rechenwerk)
  - ❑ Parallelität auf Kontrollpfadebene bei Mehrkern- oder Mehrprozessorrechnern möglich
  - ❑ Skalierbarkeit im Bereich $[1.5,0.8N]$ bei N 2 Prozessoren/Kernen
- ➤ Wichtig: Mehrkernrechner nutzen einen geteilten Speicher (DRAM). Speicherhierarchie beachten!
  - ❑ Der L2/L1 Cache ist wesentlich für die Rechenleistung (Ops/s)

16.10. Grafikprozessoren

- ➤ Eine GPU entspricht der MIMD Klasse.

- ❏ Aber: GPUs sind keine generischen Rechneranlagen sondern sind auf Grafikoperation (häufig Matrixalgebra) spezialisiert! (Aber: Übergang zu GPGPU Arch.)

Eine GPU-Einheit besteht aus: + mehreren **Prozessorclustern** (PC), die + mehrere **Streaming-Multiprozessoren** (SM) enthalten. - Jede SM hat eine Layer-1-Befehls-Cache-Schicht mit den zugehörigen Kernen. - In der Regel verwendet ein SM einen L1-Cache und einen gemeinsam genutzten L2-Cache - Seine Architektur ist tolerant gegenüber Speicherlatenz. - Häufig reduzierte Rechenauflösung ("Float16")



[nielshagoort.com/2019/03/12/exploring-the-gpu-architecture/]

16.11. Vergleich CPU mit GPU Architektur

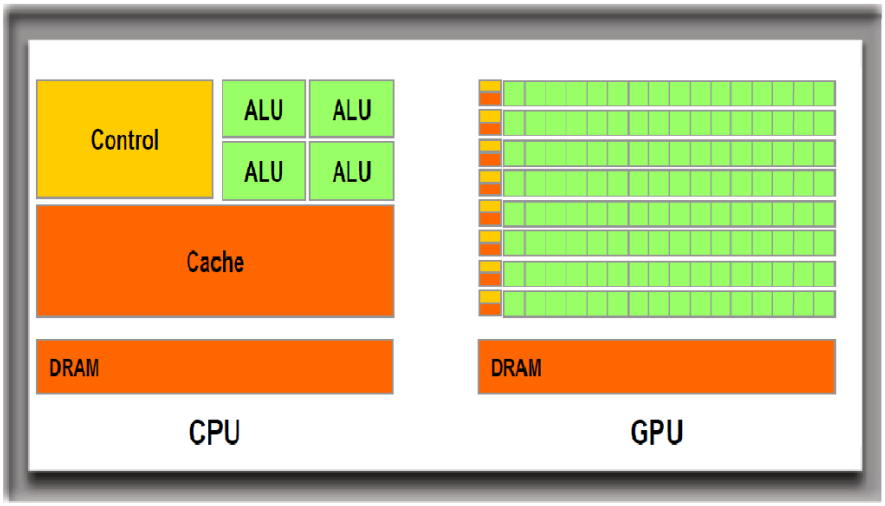


Abb. 56. CPU sind kontrollpfadorientier, GPU sind datenpfadorientiert

16.12. Digitallogik und FPGA

FPGA: Field Programmable (Logic) Gate Array

Funktionale Systeme können 1:1 unter Ausnutzung maximaler Parallelität auf Digitallogiksysteme abgebildet werden

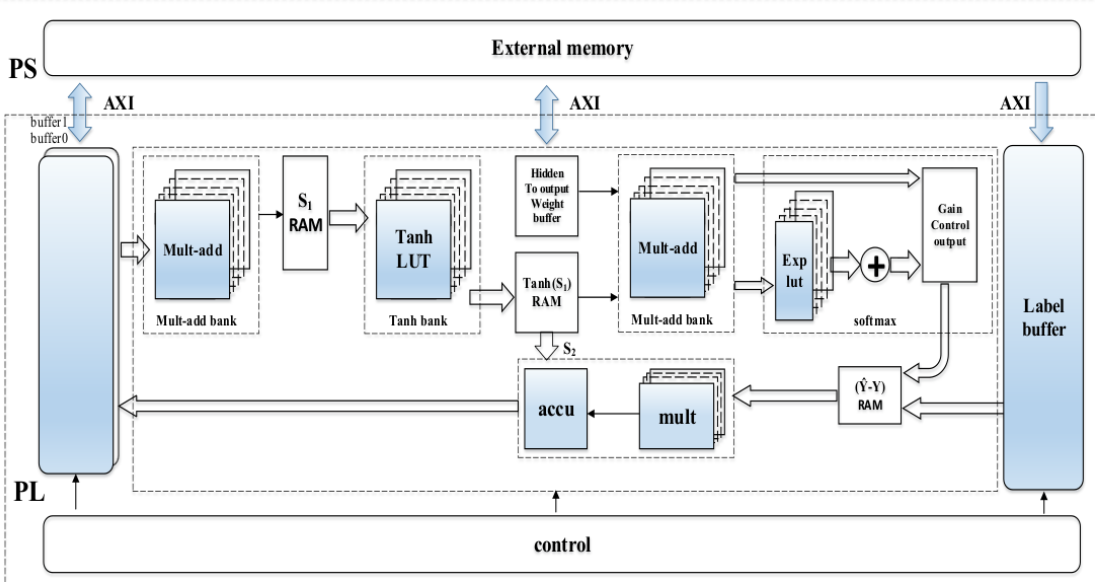
- ➤ Aber Fliesskommarithmetik erfordert großen Ressourcenbedarf (Multi-ALU Architekturen nur begrenzt möglich)
- ➤ Daher Übergang auf Ganzzahlarithmetik und Intervallartihmetik (bzw. Festpunktarithmetik)!
- ➤ Ein FPGA kann (1) Algorithmen und (2) funktionale Modelle implementieren

1. **Abbildung von Algorithmen** (wie z.B. Gleichungslöser, Regler, FFT, usw.) ermöglicht beschleunigte Berechnung des Algorithmus → Lernalgrotihmen werden i.A. nicht auf FPGAs übertragen!
2. **Abbildung von Funktionsmodellen** hier das ANN und deren Berechnung (Vorwärtspropagation) auf FPGA
   - ➤ Problem: Training mit Backpropagation außerhalb i.A. mit CPU

- ➤ Daher besser Übertragung eines in Software trainierten Modells auf die Hardware (d.h. nur für Inferenz)
- ➤ Aber: Ein mit "Float64" trainiertes Modell muss auf Ganzzahlarithmetik "Int16" transformiert werden und kann u.U. schlechte Ergebnisse ergeben!
- ➤ Daher schon Training mit Zieldatenformat und Einführung von speziellen approximativen Berechnungen (d.h. Übersetzung)

16.13. Generische KNN-FPGA Architekturen

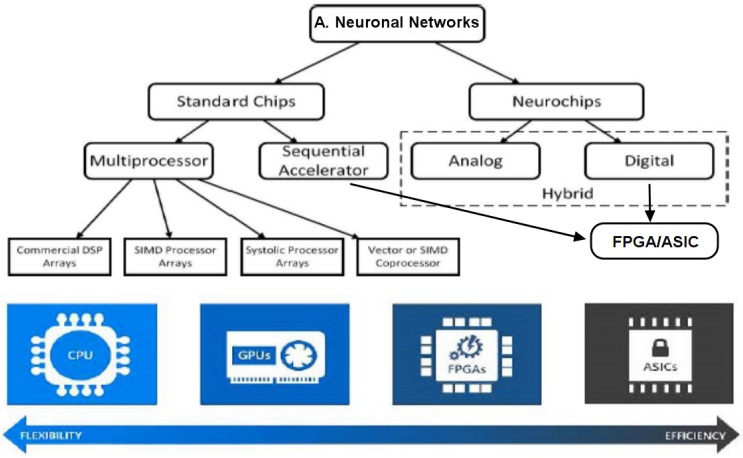
- ➤ Bisher sind die Implementierungen von KNN in FPGA problemspezifisch
- ➤ Generische Architekturen wären problemspezifisch konfigurierbar
- ➤ Dazu müssen die Elementaroperationen und Verbindungselemente eines KNN in einem FPGA frei konfigurierbar implementiert werden
- ➤ Ein spezifisches KNN wird dann im FPGA durch Verbindung und Konfiguration der Elementarblöcke individuell angepasst



[Hao, A General Neural Network Hardware Architecture on FPGA]

Abb. 57. Die FPGA Blöcke bilden Elementaroperationen von KNN ab

16.14. Zusammenfassung



[Jawandhiya, HARDWARE DESIGN FOR MACHINE LEARNING]

Abb. 58. Taxonomie der Hardwarearchitekturen für KNN: Fexibilität vs. Effizienz

17. Referenzen

17.1. Bücher

1. M. J. Zaki and W. Meira, Data Mining and Machine Learning - Fundamental Concepts and Algorithms. Cambridge University Press, 2020.
2. C. R. Farrar and K. Worden, Structural Health Monitoring: A Machine Learning Perspective. Wiley-Interscience, 2013.
3. X.-S. Yang, Introduction to Algorithms for Data Mining and Machine Learning. Elsevier, 2019.
4. M. Sugiyama, Introduction to Statistical Machine Learning. 2016.
5. J. Herrmann, Maschinelles Lernen und Wissensbasierte Systeme. Springer, 1997.
6. R. Zafarani, M. A. Abbasi, and H. Liu, Social Data Mining. 2014.
7. I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal, Data Mining Practical Machine Learning Tools and Techniques. Morgan Kaufmann.

8. C. Borcea, M. Talasila, and R. Curtmola, Mobile Crowdsensing. CRC Press, 2017.

9. J. G. Webster, Measurement, Instrumentation and Sensors Handbook, no. 0. 1999.

10. L. Rokach and O. Maimon, Data Mining with Decision Trees - Theory and Applications. World Scientific Publishing, 2015.

11. N. J. Nilsson, Introduction To Machine Learning. 1996.

12. T. M. Mitchel, Machine Learning. McGraw Hill, 1997.

13. P. Attewell and D. B. Monaghan, Data mining for the social sciences : an introduction. University of California Press, 2015.

14. J. R. Quinlan, "Induction of Decision Trees," in Machine Learning, Kluwer Academic Publishers, Boston, 1986

15. M. T. Hagan, howard B. Demuth, M. H. Beale, and O. D. Jesus, Neural Network Design.

16. L. Fausett, Fundamentals of Neural Networks. 1994.

17. E. Alpaydın, Introduction to Machine Learning. MIT Press, 2010.

17.2. Artikel

100. T. Mueller, A. G. Kusne, and R. Ramprasad, "Machine learning in materials science: Recent progress and emerging applications," Reviews in Computational Chemistry, Volume 29, First Edition., 2016.

101. N.-C. Chen et al., "Challenges of Applying Machine Learning to Qualitative Coding."

102. J. Radford, K. Joseph, "Theory In, Theory Out: The Uses of Social Theory in Machine Learning for Social Science", Front. Big Data

103. Abadi et al., "TensorFlow: A system for large-scale machine learning", 2th USENIX Symposium on Operating Systems Design and Implementation, USENIX Association