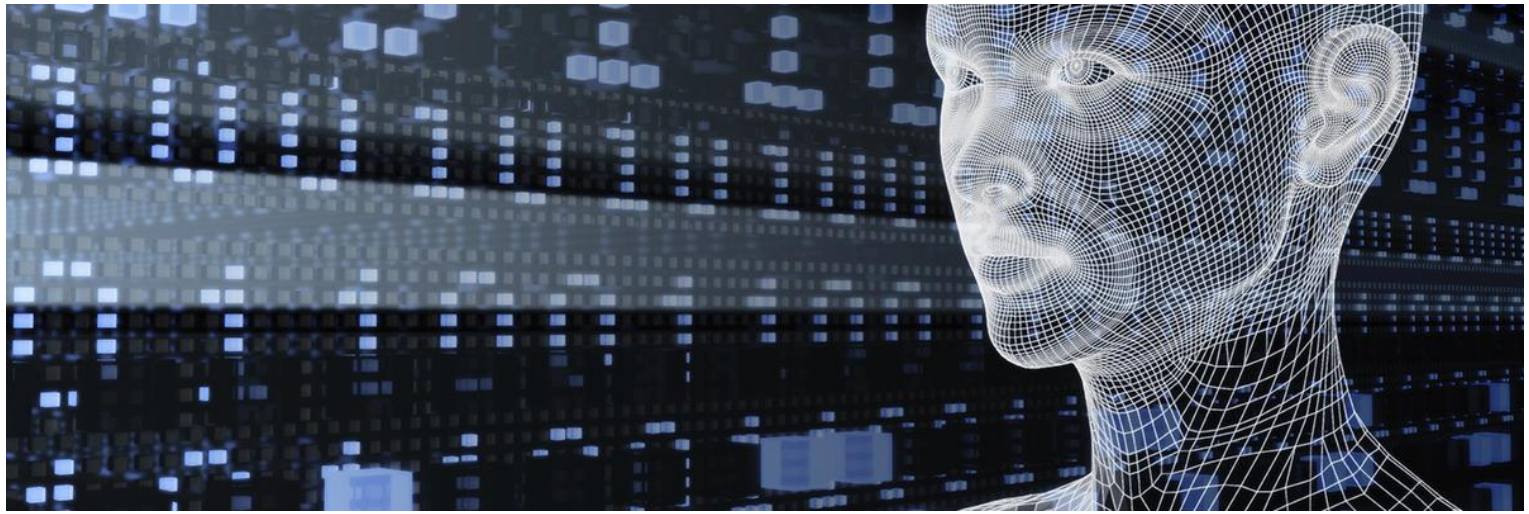
Künstliche Intelligenz

Vorlesung 6 und 7: Einführung in das maschinelle Lernen und Data Mining



Literatur

- Wolfgang Ertel – Grundkurs Künstliche Intelligenz. Eine Praxisorientierte Einführung
- Credits: Slides zum Buch von W. Ertel.

Warum Maschinelles Lernen?

Elaine Rich:

„Artificial Intelligence is the study of how to make computers do things at which, at the moment, people are better.”,

und:

Menschen lernen besser als Computer

=>

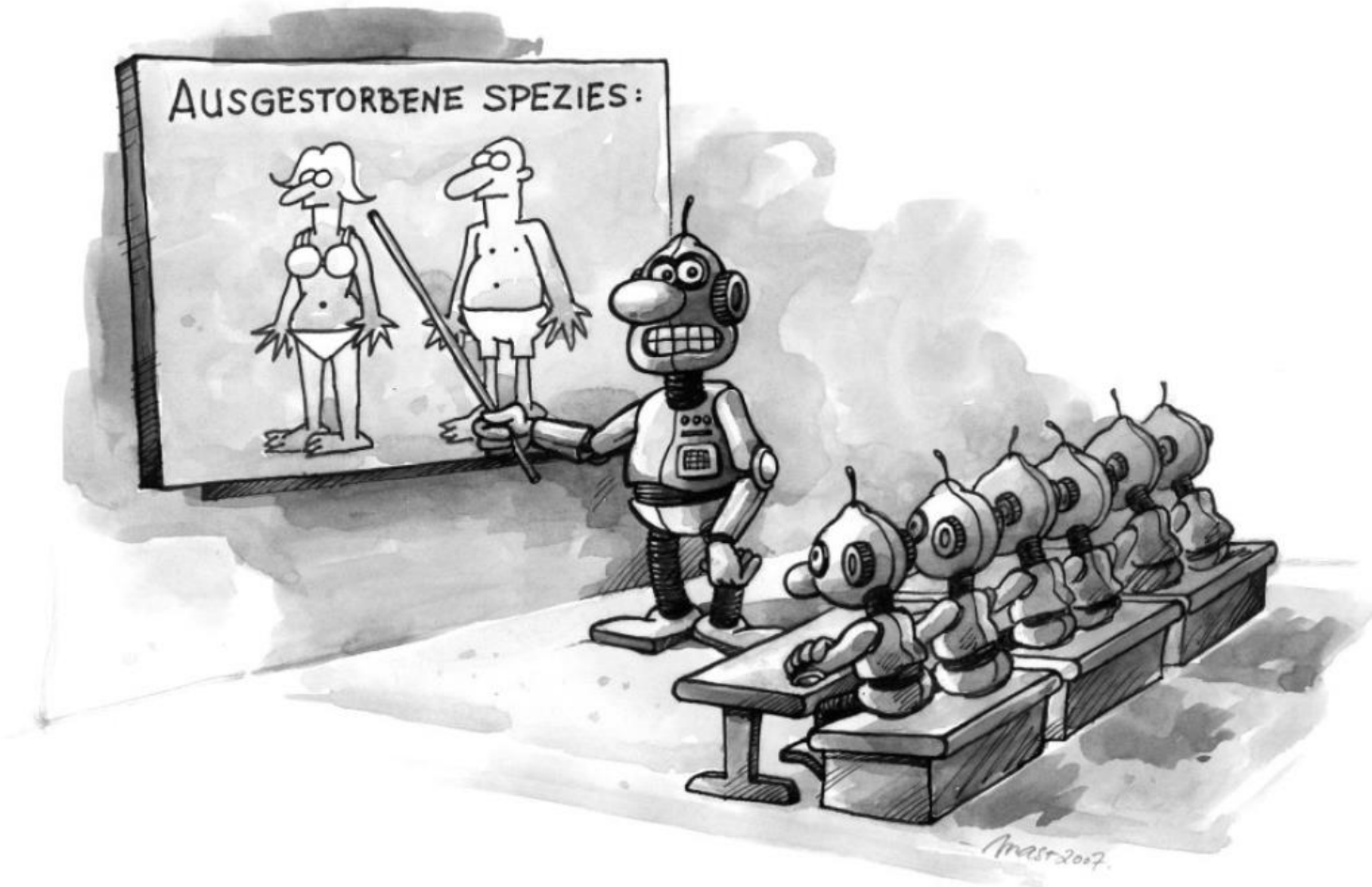
Maschinelles Lernen ist für die KI sehr wichtig

Warum Maschinelles Lernen?

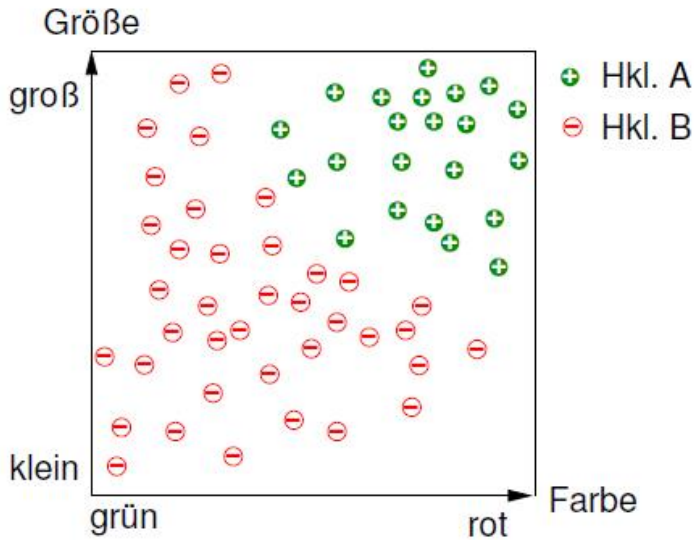
- Komplexität der Software-Entwicklung, z.B.
  - Verhalten eines autonomen Roboters
  - Expertensysteme
  - Spamfilter
- Lösung: hybride Mischung aus programmierten und gelernten Komponenten.

Was ist Lernen?

- Lernen von Vokabeln einer Fremdsprache?
- Auswendiglernen eines Gedichts?
- Lernen mathematischer Fertigkeiten?
- Lernen des Skifahrens?


AUSGESTORBENE SPEZIES:

Sortieranlage für Äpfel



Apfelsortieranlage und einige im Merkmalsraum klassifizierte Äpfel der Handelsklassen A und B.

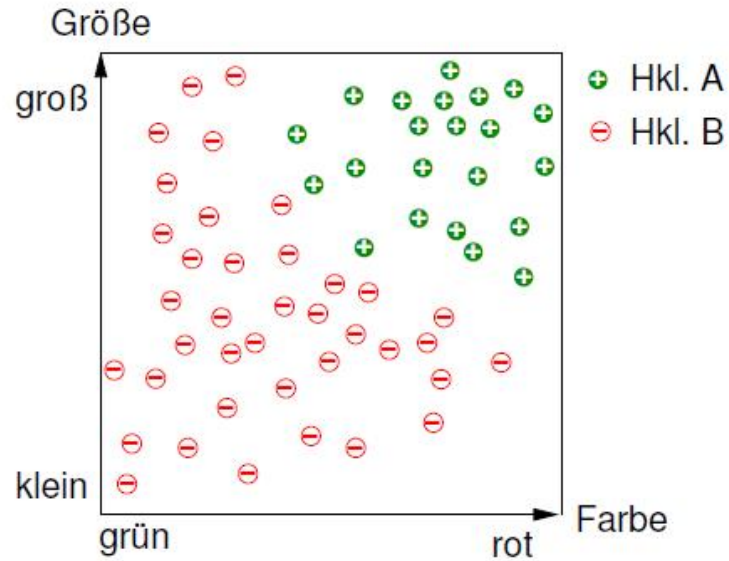
Sortieranlage für Äpfel

Merkmale (engl. features): Größe und Farbe
Klassifikationsaufgabe: Äpfel in HKl. A, bzw. B einteilen

(Classifier)

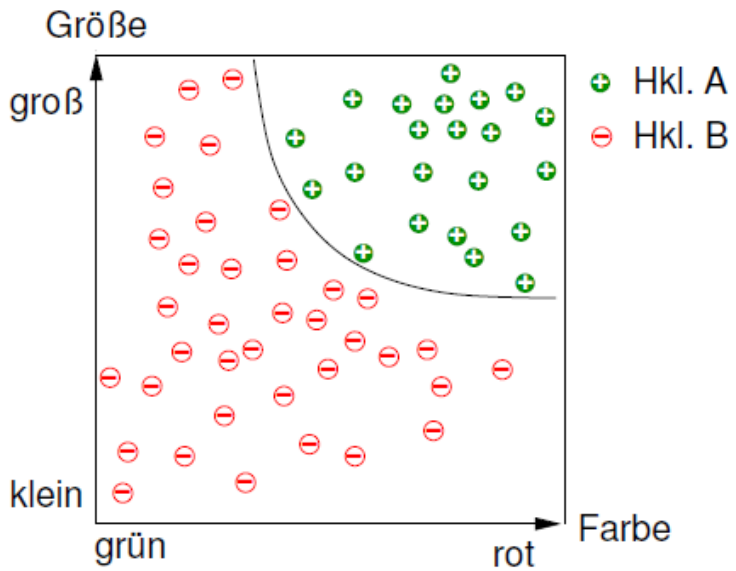
Größe [cm]	8	8	6	3	...
Farbe [0: grün, 1: rot]	0.1	0.3	0.9	0.8	...
Handelsklasse	B	A	A	B	...

Trainingsdaten für den Apfelsortieragenten.



Apfelsortieranlage und einige im Merkmalsraum klassifizierte Äpfel der Handelsklassen A und B.

Kurve trennt die Klassen



In der Praxis: 30 oder mehr Merkmale!

***n* Merkmale:** In n-dimensionalem **Merkmalsraum** ist eine $(n-1)$-dimensionale Hyperfläche zu finden, welche die beiden Klassen möglichst gut trennt.

Begriffe

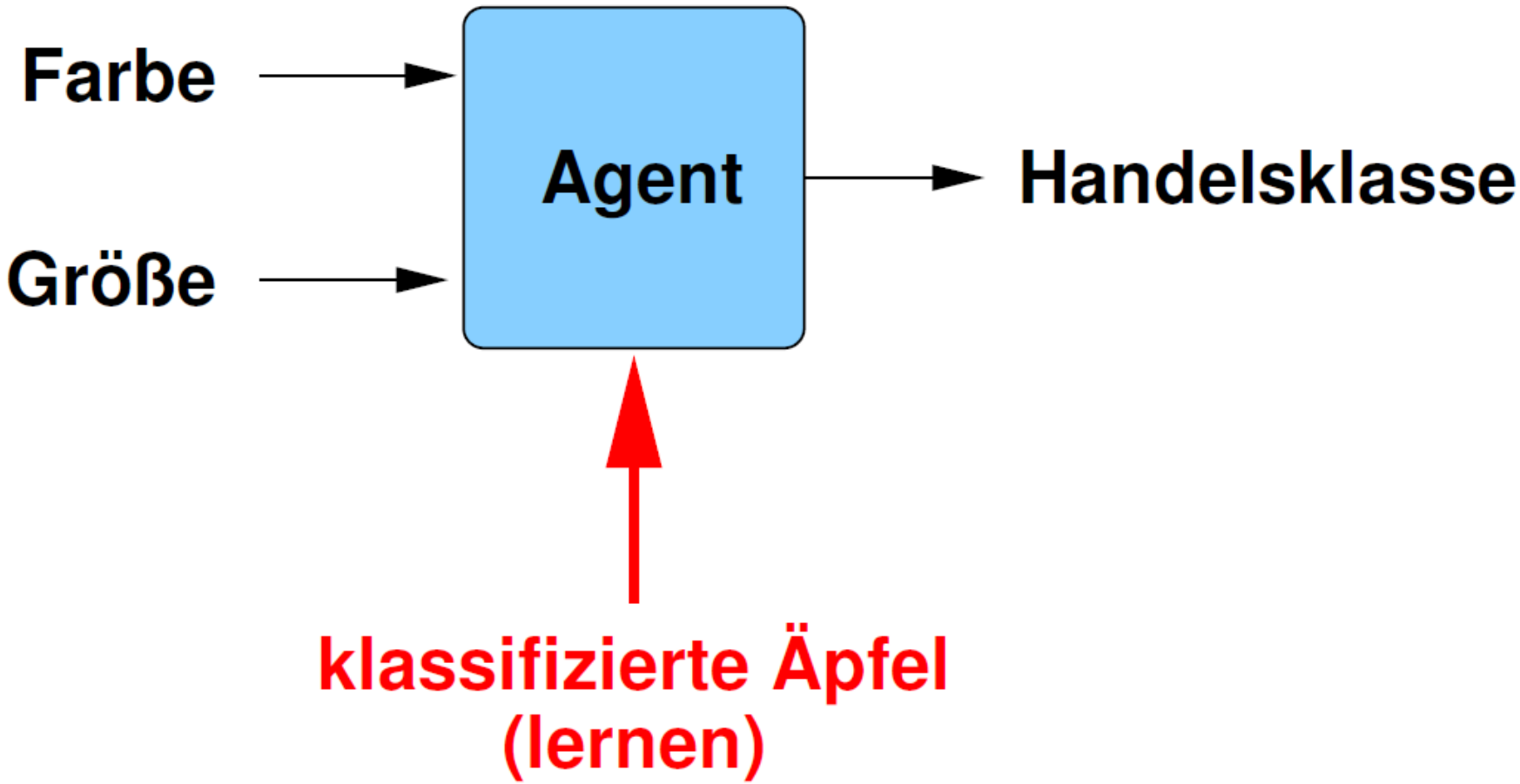
Klassifikation / Classifier: bildet einen Merkmalsvektor auf einen Klassenwert ab. Feste Anzahl von Alternativen.

Beispiel: Apfelsortierung

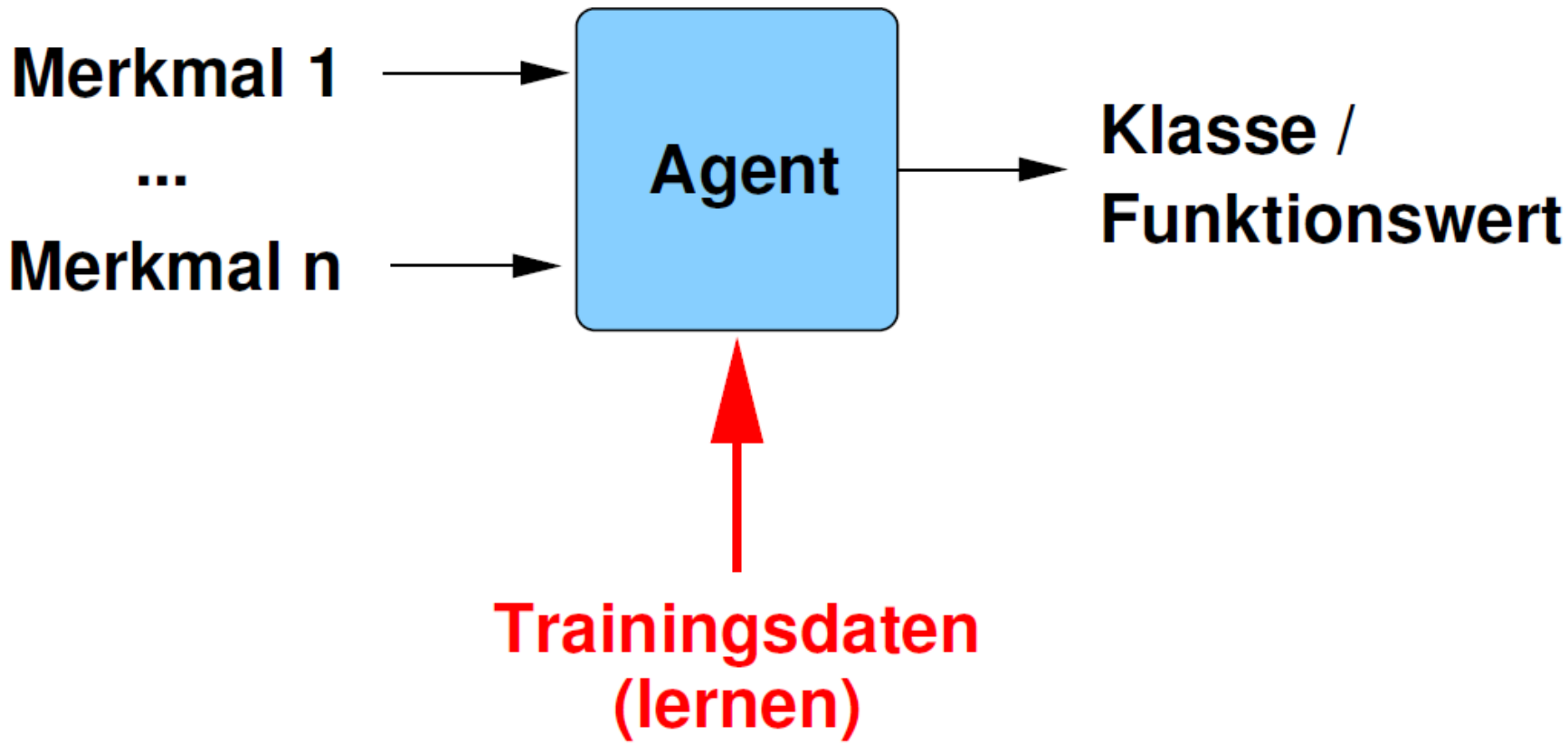
Approximation: bildet einen Merkmalsvektor auf eine reelle Zahl ab

Beispiel: Prognose des Kurses einer Aktie aus gegebenen Merkmalen

Der lernende Agent



Der lernende Agent, allgemein



Definition des Begriffs „Maschinelles Lernen”

Machine Learning is the study of computer algorithms that improve automatically through experience.

Definition

Ein Agent heißt lernfähig, wenn sich seine Leistungsfähigkeit auf neuen, unbekannten Daten, im Laufe der Zeit (nachdem er viele Trainingsbeispiele gesehen hat) verbessert (gemessen auf einem geeigneten Maßstab).

Begriffe

am Beispiel der Apfelsortierung:

Aufgabe: Abbildung von Größe und Farbe eines Apfels auf die Handelsklasse lernen

Performance-Maß: Zahl der korrekt klassifizierten Äpfel

variabler Agent: (genauer eine Klasse von Agenten): das Lernverfahren bestimmt die Klasse aller möglichen Funktionen.

Trainingsdaten: (Erfahrung): Trainingsdaten enthalten das Wissen, welches von dem Lernverfahren extrahiert werden soll.

Testdaten: zeigen, ob der trainierte Agent gut von den gelernten auf neue Daten generalisieren kann.

Was ist Data-Mining?

Was ist Data-Mining?

- Wissen aus Daten extrahieren
- Wissen für Menschen verständlich machen
- Beispiel: Induktion von Entscheidungsbäumen
- Wissensmanagement:
  - Analyse von Kundenwünschen durch Statistik, z.B. in e-Shops
  - gezielte Werbung

Definition

> Der Prozess des Gewinnens von Wissen aus Daten sowie dessen Darstellung und Anwendung wird als **Data Mining** bezeichnet.

Literatur: I. Witten und E. Frank. *Data Mining*. Von den Autoren in Java entwickelte DataMining Programmbibliothek WEKA: (`www.cs.waikato.ac.nz/~ml/weka`). Hanser Verlag München, 2001

Datenanalyse

LEXMED-Daten, $N = 473$ Patienten:

Var.-Nr.	Beschreibung	Werte
1	Alter	stetig
2	Geschlecht (1=männl., 2=weibl.)	1,2
3	Schmerz Quadrant 1	0,1
4	Schmerz Quadrant 2	0,1
5	Schmerz Quadrant 3	0,1
6	Schmerz Quadrant 4	0,1
7	Lokale Abwehrspannung	0,1
8	Generalisierte Abwehrspannung	0,1
9	Schmerz bei Loslassmanoever	0,1
10	Erschuetterung	0,1
11	Schmerz bei rektaler Untersuchung	0,1
12	Temperatur axial	stetig
13	Temperatur rektal	stetig
14	Leukozyten	stetig
15	Diabetes mellitus	0,1

$$\mathbf{x}^1 = (26, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 37.9, 38.8, 23100, 0, 1)$$

$$\mathbf{x}^2 = (17, 2, 0, 0, 1, 0, 1, 0, 1, 1, 0, 36.9, 37.4, 8100, 0, 0)$$

Mittelwert

$$\bar{x}_i := \frac{1}{N} \sum_{p=1}^{N} x_i^p$$

Standardabweichung

$$s_i := \sqrt{\frac{1}{N-1} \sum_{p=1}^{N} (x_i^p - \bar{x}_i)^2}.$$

Kovarianz

$$\sigma_{ij} = \frac{1}{N-1} \sum_{p=1}^{N} (x_i^p - \bar{x}_i)(x_j^p - \bar{x}_j)$$

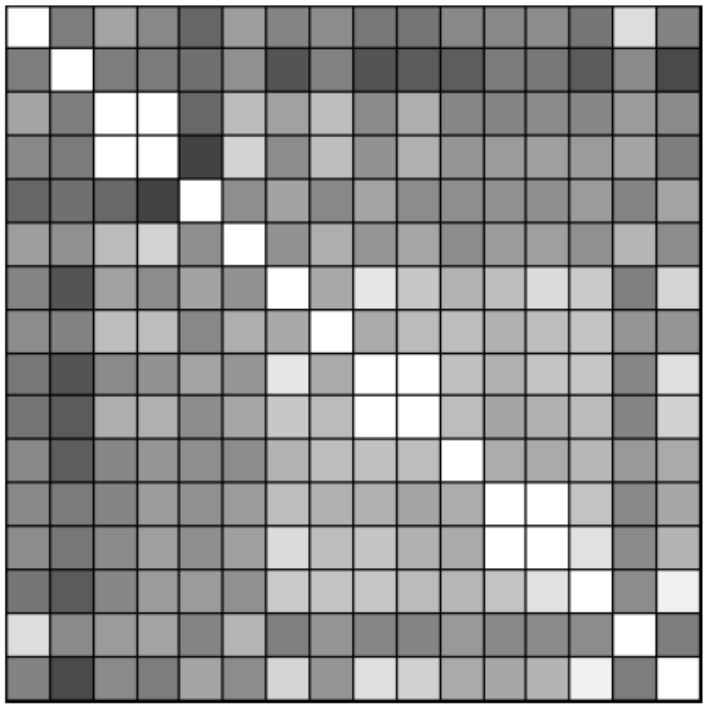
Korrelationskoeffizient:

$$K_{ij} = \frac{\sigma_{ij}}{s_i \cdot s_j}.$$

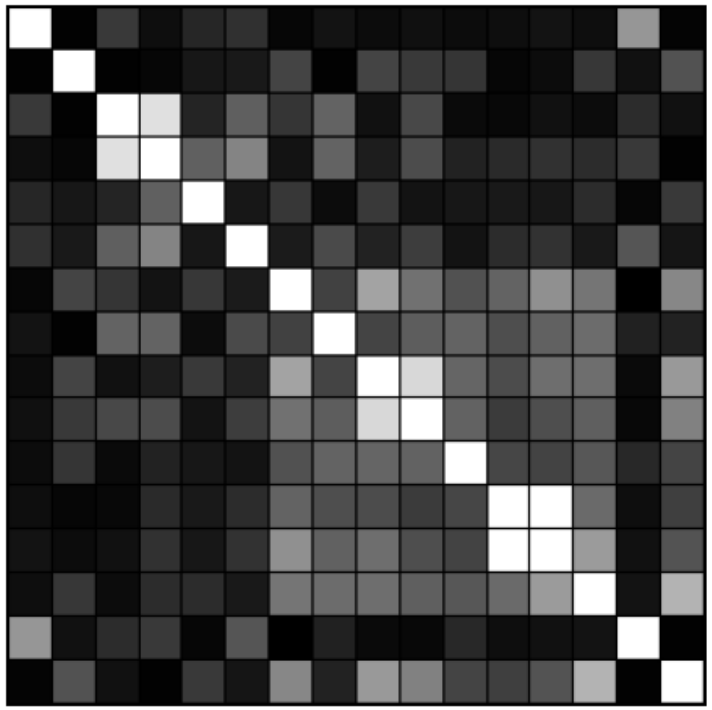
1.	−0.009	0.14	0.037	−0.096	0.12	0.018	0.051	−0.034	−0.041	0.034	0.037	0.05	−0.037	0.37	0.012
−0.009	1.	−0.0074	−0.019	−0.06	0.063	−0.17	0.0084	−0.17	−0.14	−0.13	−0.017	−0.034	−0.14	0.045	−0.2
0.14	−0.0074	1.	0.55	−0.091	0.24	0.13	0.24	0.045	0.18	0.028	0.02	0.045	0.03	0.11	0.045
0.037	−0.019	0.55	1.	−0.24	0.33	0.051	0.25	0.074	0.19	0.087	0.11	0.12	0.11	0.14	−0.0091
−0.096	−0.06	−0.091	−0.24	1.	0.059	0.14	0.034	0.14	0.049	0.057	0.064	0.058	0.11	0.017	0.14
0.12	0.063	0.24	0.33	0.059	1.	0.071	0.19	0.086	0.15	0.048	0.11	0.12	0.063	0.21	0.053
0.018	−0.17	0.13	0.051	0.14	0.071	1.	0.16	0.4	0.28	0.2	0.24	0.36	0.29	−0.0001	0.33
0.051	0.0084	0.24	0.25	0.034	0.19	0.16	1.	0.17	0.23	0.24	0.19	0.24	0.27	0.083	0.084
−0.034	−0.17	0.045	0.074	0.14	0.086	0.4	0.17	1.	0.53	0.25	0.19	0.27	0.27	0.026	0.38
−0.041	−0.14	0.18	0.19	0.049	0.15	0.28	0.23	0.53	1.	0.24	0.15	0.19	0.23	0.02	0.32
0.034	−0.13	0.028	0.087	0.057	0.048	0.2	0.24	0.25	0.24	1.	0.17	0.17	0.22	0.098	0.17
0.037	−0.017	0.02	0.11	0.064	0.11	0.24	0.19	0.19	0.15	0.17	1.	0.72	0.26	0.035	0.15
0.05	−0.034	0.045	0.12	0.058	0.12	0.36	0.24	0.27	0.19	0.17	0.72	1.	0.38	0.044	0.21
−0.037	−0.14	0.03	0.11	0.11	0.063	0.29	0.27	0.27	0.23	0.22	0.26	0.38	1.	0.051	0.44
0.37	0.045	0.11	0.14	0.017	0.21	−0.0001	0.083	0.026	0.02	0.098	0.035	0.044	0.051	1.	−0.0055
0.012	−0.2	0.045	−0.0091	0.14	0.053	0.33	0.084	0.38	0.32	0.17	0.15	0.21	0.44	−0.0055	1.

Korrelationsmatrix für die 16 Appendizitis-Variablen, gemessen auf 473 Fällen.

Korrelationsmatrix als Dichteplot



$K_{ij} = -1$: schwarz, $K_{ij} = 1$: weiß



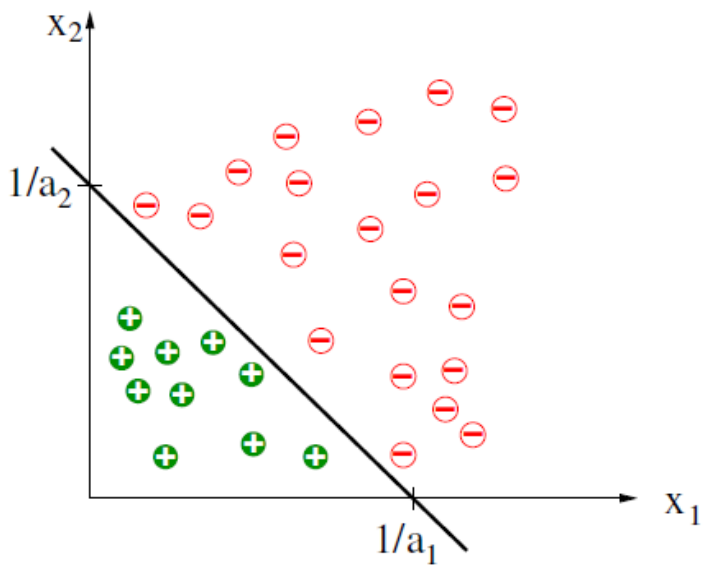
$|K_{ij}| = 0$: schwarz, $|K_{ij}| = 1$: weiß

Das Perzeptron, ein linearer Classifier

Linear separable zweidimensionale Datenmenge.

Trenngerade:

$$a_1 x_1 + a_2 x_2 = 1.$$



Lineare Separabilität

$n-1$-dimensionale Hyperebene im $\mathbb{R}^n$ ist durch

$$\sum_{i=1}^{n} a_i x_i = \theta$$

gegeben, also:

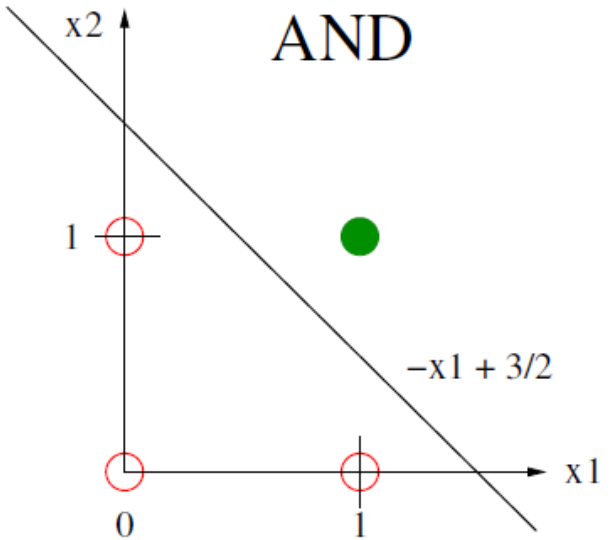
Definition

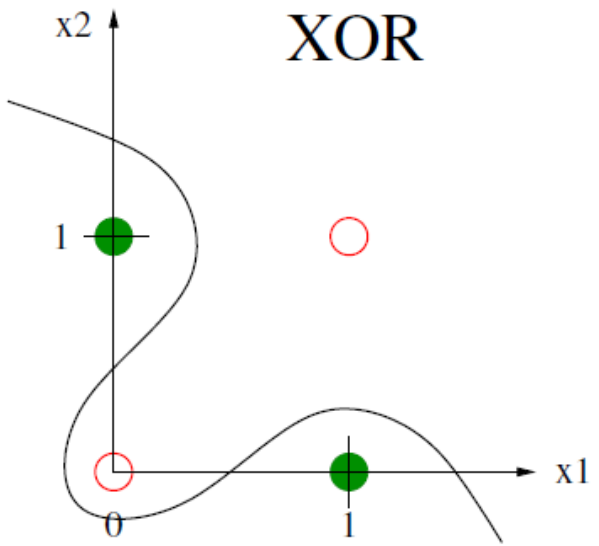
Zwei Mengen $M_1 \subset \mathbb{R}^n$ und $M_2 \subset \mathbb{R}^n$ heißen linear separabel, wenn reelle Zahlen $a_1, \ldots, a_n, \theta$ existieren mit

$$\sum_{i=1}^{n} a_i x_i > \theta \quad \text{für alle } \mathbf{x} \in M_1 \qquad \text{und} \qquad \sum_{i=1}^{n} a_i x_i \leq \theta \quad \text{für alle } \mathbf{x} \in$$

Der Wert θ wird als Schwelle bezeichnet.

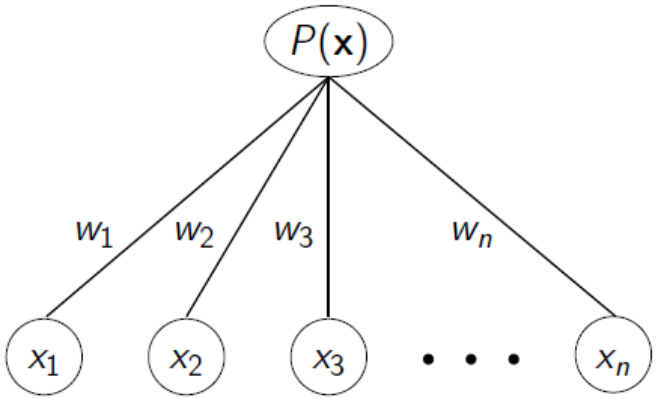
Lineare Separabilität





AND ist linear separabel, XOR nicht. (● $\hat{=}$ *wahr*, ○ $\hat{=}$ *falsch*).

Perzeptron



Definition

Sei $\mathbf{w} = (w_1, \ldots, w_n) \in \mathbb{R}^n$ ein Gewichtsvektor und $\mathbf{x} \in \mathbb{R}^n$ ein Eingabevektor. Ein **Perzeptron** stellt eine Funktion $P : \mathbb{R}^n \to \{0, 1\}$ dar, die folgender Regel entspricht:

$$P(\mathbf{x}) = \begin{cases} 1 & \text{falls} \quad \mathbf{w}\,\mathbf{x} = \sum_{i=1}^{n} w_i x_i > 0 \\ 0 & \text{sonst} \end{cases}$$

Perzeptron

- zweilagiges gerichtetes Neuronales Netzwerk
- Die Eingabevariablen x_i werden als **Merkmale** (engl. features) bezeichnet.
- trennende Hyperebene geht durch den Ursprung
- Punkte $\mathbf{x}$ über der Hyperebene $\sum_{i=1}^{n} w_i x_i = 0$ werden positiv ($P(\mathbf{x}) = 1$) klassifiziert

Die Lernregel

M_+: Menge der positiven Trainingsmuster
M_-: Menge der negativen Trainingsmuster
[15], S. 164 :

PerzeptronLernen(M_+, M_-)

$\mathbf{w}$ = beliebiger Vektor reeller Zahlen ungleich 0
Repeat
  For all $\mathbf{x} \in M_+$
    If $\mathbf{w}\,\mathbf{x} \leq 0$ **Then** $\mathbf{w} = \mathbf{w} + \mathbf{x}$
  For all $\mathbf{x} \in M_-$
    If $\mathbf{w}\,\mathbf{x} > 0$ **Then** $\mathbf{w} = \mathbf{w} - \mathbf{x}$
Until alle $\mathbf{x} \in M_+ \cup M_-$ werden korrekt klassifiziert

[15] M. Minsky und S. Papert. *Perceptrons*. MIT Press, Cambridge, MA, 1969.

PerzeptronLernen(M_+, M_-)

...

For all $\mathbf{x} \in M_+$

If $\mathbf{w}\,\mathbf{x} \leq 0$ Then $\mathbf{w} = \mathbf{w} + \mathbf{x}$

For all $\mathbf{x} \in M_-$

If $\mathbf{w}\,\mathbf{x} > 0$ Then $\mathbf{w} = \mathbf{w} - \mathbf{x}$

...

$$(\mathbf{w} + \mathbf{x}) \cdot \mathbf{x} = \mathbf{w}\,\mathbf{x} + \mathbf{x}^2$$

$\Rightarrow$ $\mathbf{w}\,\mathbf{x}$ wird irgendwann positiv!

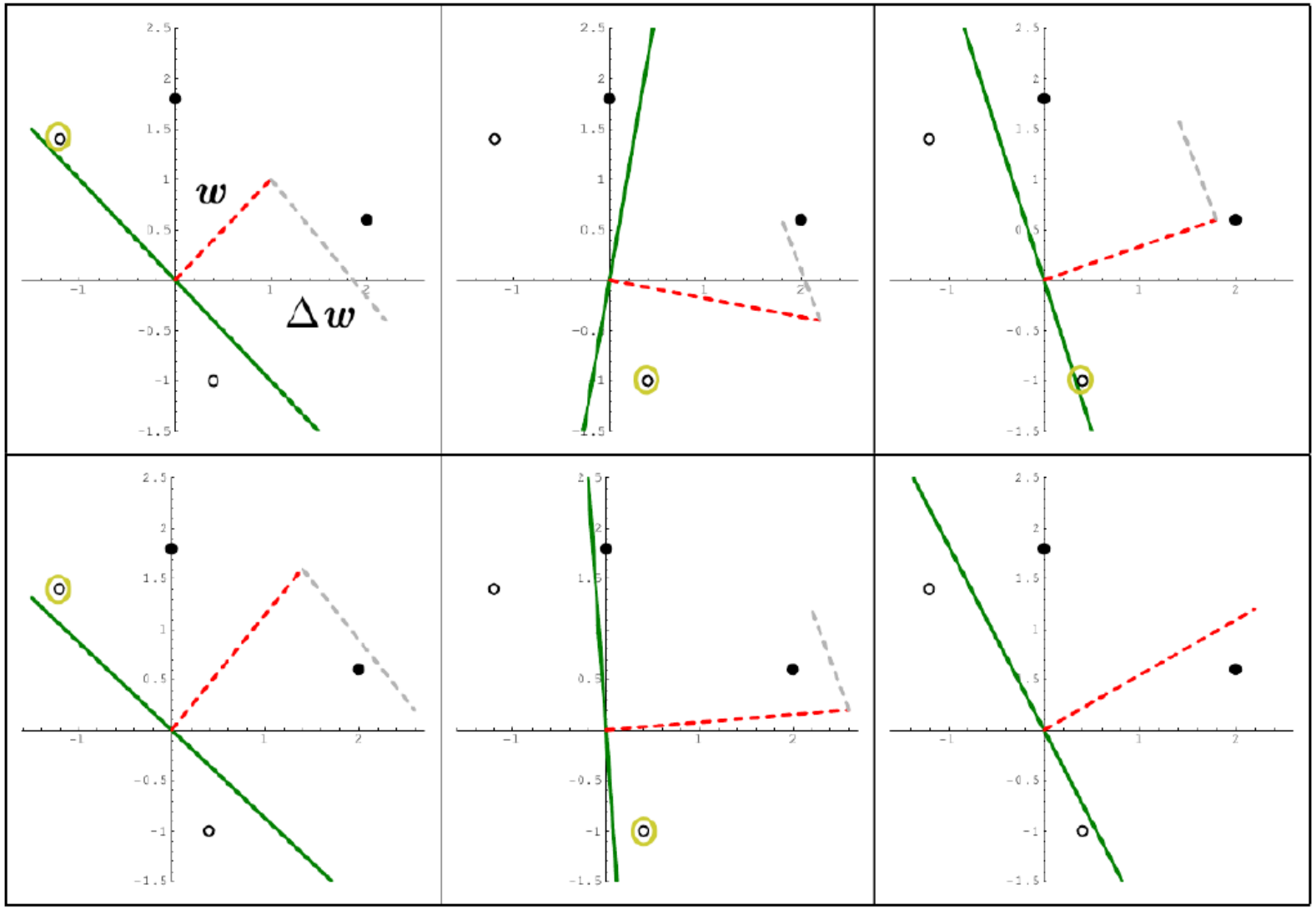
$$(\mathbf{w} - \mathbf{x}) \cdot \mathbf{x} = \mathbf{w}\,\mathbf{x} - \mathbf{x}^2$$

$\Rightarrow$ $\mathbf{w}\,\mathbf{x}$ wird irgendwann negativ![16]

[16] Vorsicht! Dies ist kein Konvergenzbeweis!

Beispiel

- $M_+ = \{(0, 1.8), (2, 0.6)\}$, $M_- = \{(-1.2, 1.4), (0.4, -1)\}$
- Initialer Gewichtsvektor: $\mathbf{w} = (1, 1)$
- Gerade $\mathbf{w}\,\mathbf{x} = x_1 + x_2 = 0$



$(-1.2, 1.4)$ falsch klassifiziert weil $(-1.2, 1.4) \cdot \begin{pmatrix} 1 \\ 1 \end{pmatrix} = 0.2 > 0.$

Also $\mathbf{w} = (1, 1) - (-1.2, 1.4) = (2.2, -0.4)$

Konvergenz

Satz

Es seien die Klassen M_+ und M_- linear separabel durch eine Hyperebene $\mathbf{w}\,\mathbf{x} = 0$. Dann konvergiert die PerzeptronLernen *für jede Initialisierung des Vektors w. Das Perzeptron P mit dem so berechneten Gewichtsvektor trennt die Klassen M_+ und M_-, d.h.*

$$P(\mathbf{x}) = 1 \quad \Leftrightarrow \quad \mathbf{x} \in M_+$$

und

$$P(\mathbf{x}) = 0 \quad \Leftrightarrow \quad \mathbf{x} \in M_-.$$

Lineare Separabilität

- Perzeptron kann nicht beliebige linear separable Mengen trennen
- im $\mathbb{R}^2$ Ursprungsgerade
- im $\mathbb{R}^n$ Hyperebene im Ursprung,
- denn $\sum_{i=1}^{n} w_i x_i = 0$.

Trick

$x_n := 1$
Gewicht $w_n =: -\theta$ wirkt als Schwelle **(bias unit)**, denn

$$\sum_{i=1}^{n} w_i x_i = \sum_{i=1}^{n-1} w_i x_i - \theta > 0 \quad \Leftrightarrow \quad \sum_{i=1}^{n-1} w_i x_i > \theta$$

- Anwendung: an jeden Trainingsdatenvektor ein 1-Bit anhängen!
- auch das Gewicht w_n, bzw. die Schwelle θ wird gelernt!

ein Perzeptron $P_\theta : \mathbb{R}^{n-1} \to \{0, 1\}$

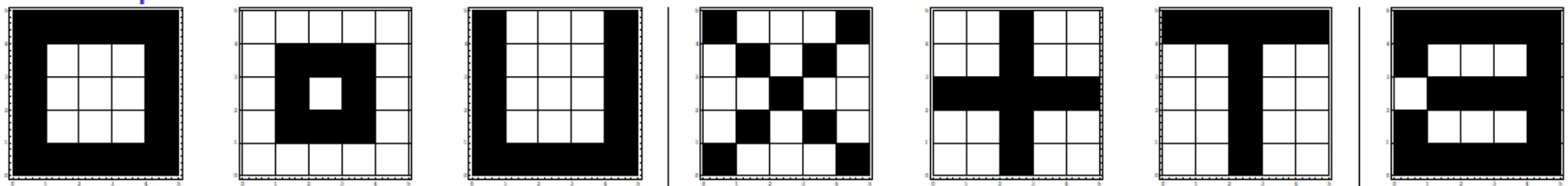
$$P_\theta(x_1, \ldots, x_{n-1}) = \begin{cases} 1 & \text{falls } \sum_{i=1}^{n-1} w_i x_i > \theta \\ 0 & \text{sonst} \end{cases} \tag{27}$$

mit beliebiger Schwelle kann durch ein Perzeptron $P : \mathbb{R}^n \to \{0, 1\}$ mit Schwelle 0 simuliert werden. Also:

Satz

Eine Funktion $f : \mathbb{R}^n \to \{0, 1\}$ kann von einem Perzeptron genau dann dargestellt werden, wenn die beiden Mengen der positiven und negativen Eingabevektoren linear separabel sind.

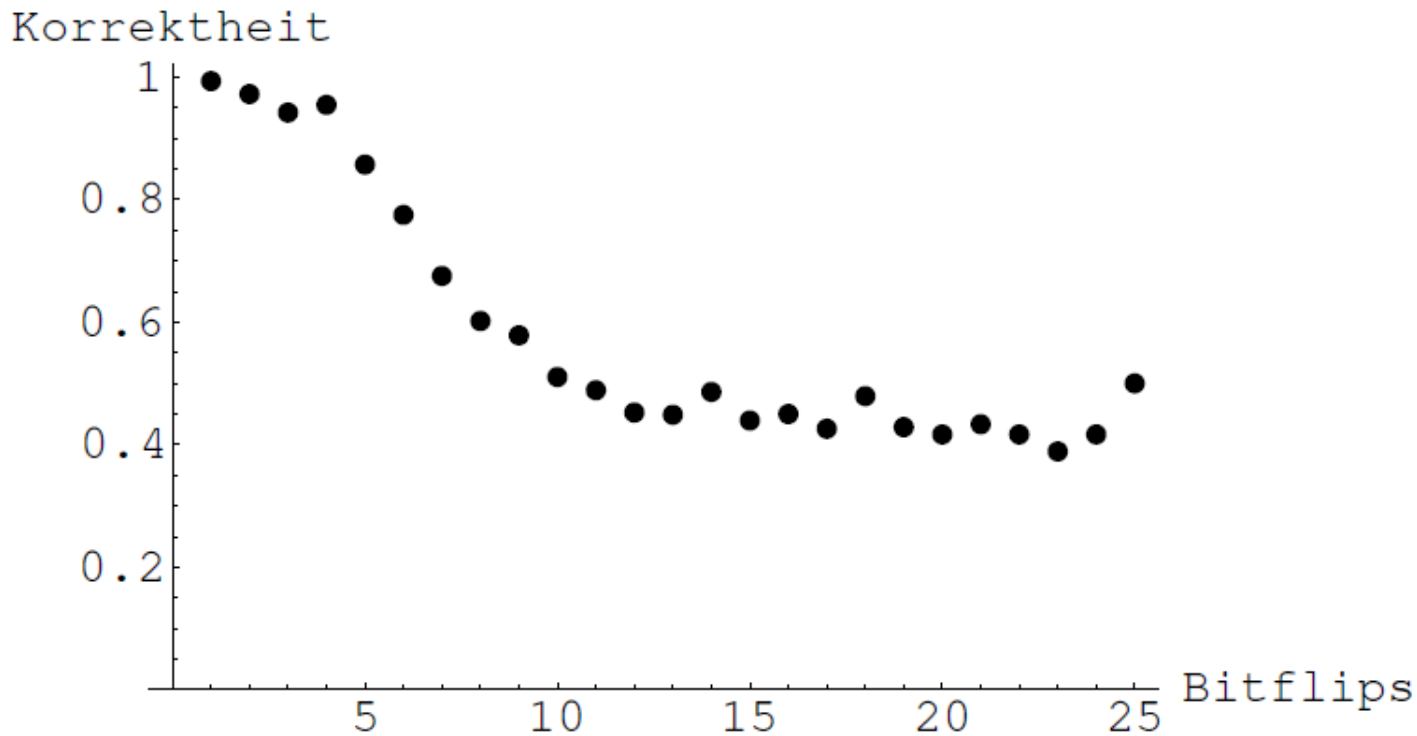
Beispiel



Positive Trainingsbeispiele (M_+) | *Negative Trainingsbeispiele (M_-)* | *Testmuster*

- Trainingsdaten werden in 4 Iterationen über alle Muster gelernt
- Testen der Generalisierungsfähigkeit: verrauschte Muster mit einer variablen Anzahl invertierter (aufeinanderfolgender) Bits.

Relative Korrektheit des Perzeptrons in Abhängigkeit von der Zahl invertierter Bits in den Testdaten.



Optimierung und Ausblick

- langsame Konvergenz
- Beschleunigung durch Normierung der Gewichtsänderungsvektoren: $\mathbf{w} = \mathbf{w} \pm \mathbf{x}/|\mathbf{x}|$.
- Bessere Initialisierung des Vektors $\mathbf{w}$.

$$\mathbf{w_0} = \sum_{x \in M_+} \mathbf{x} - \sum_{x \in M_-} \mathbf{x},$$

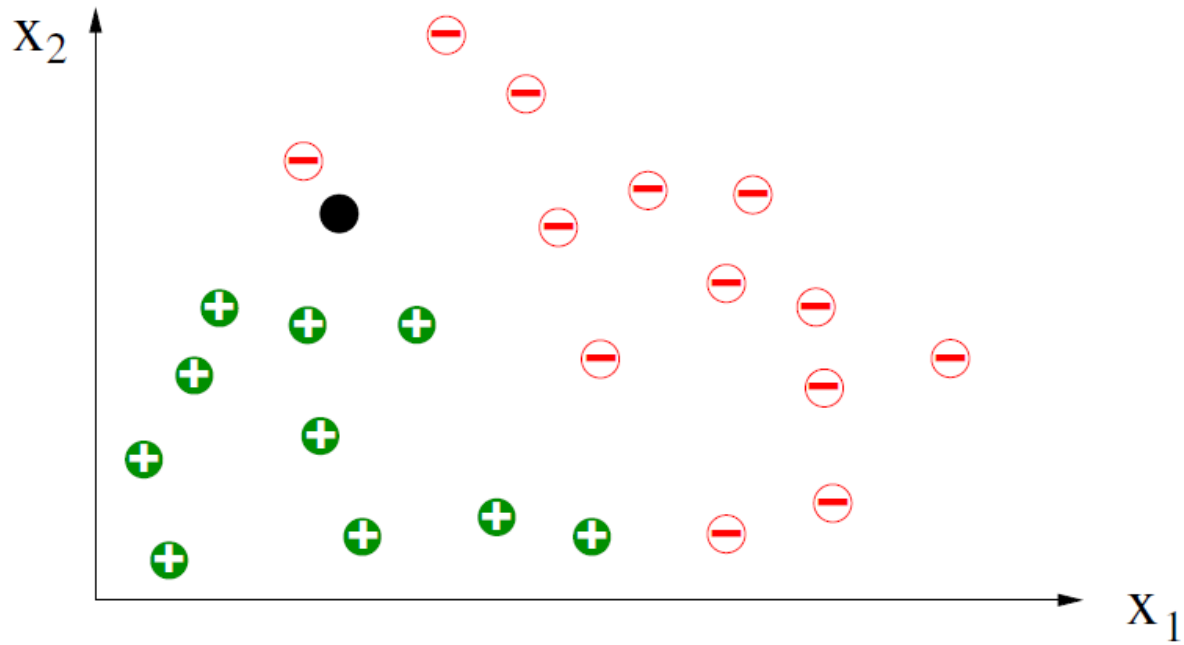
  die in genauer untersucht werden soll.
- mehrlagige Netze, z.B. Backpropagation sind mächtiger

Die Nearest Neighbour Methode

- Auswendiglernen mit Generalisierung
- Arzt merkt sich Fälle
- 1. bei der Diagnose: erinnern an ähnlich gelagerte Fälle aus der Vergangenheit.
- 2. gleiche oder eine ähnliche Diagnose stellen.
- Schwierigkeiten:
  - Arzt muss gutes Gefühl für **Ähnlichkeit** besitzen
  - Ist der gefundene Fall ähnlich genug?

Was heißt Ähnlichkeit?

Zwei Beispiele sind umso ähnlicher, je geringer ihr Abstand im Merkmalsraum ist.



Schwarz markierter Punkt wird negativ klassifiziert.

Abstand

Abstand $d(\mathbf{x}, \mathbf{y})$ zwischen zwei Punkten:
durch die euklidische Norm gegebenen Metrik

$$d(\mathbf{x}, \mathbf{y}) = |\mathbf{x} - \mathbf{y}| = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}.$$

gewichtete Merkmale:

$$d_w(\mathbf{x}, \mathbf{y}) = |\mathbf{x} - \mathbf{y}| = \sqrt{\sum_{i=1}^{n} w_i (x_i - y_i)^2}.$$

Algorithmus

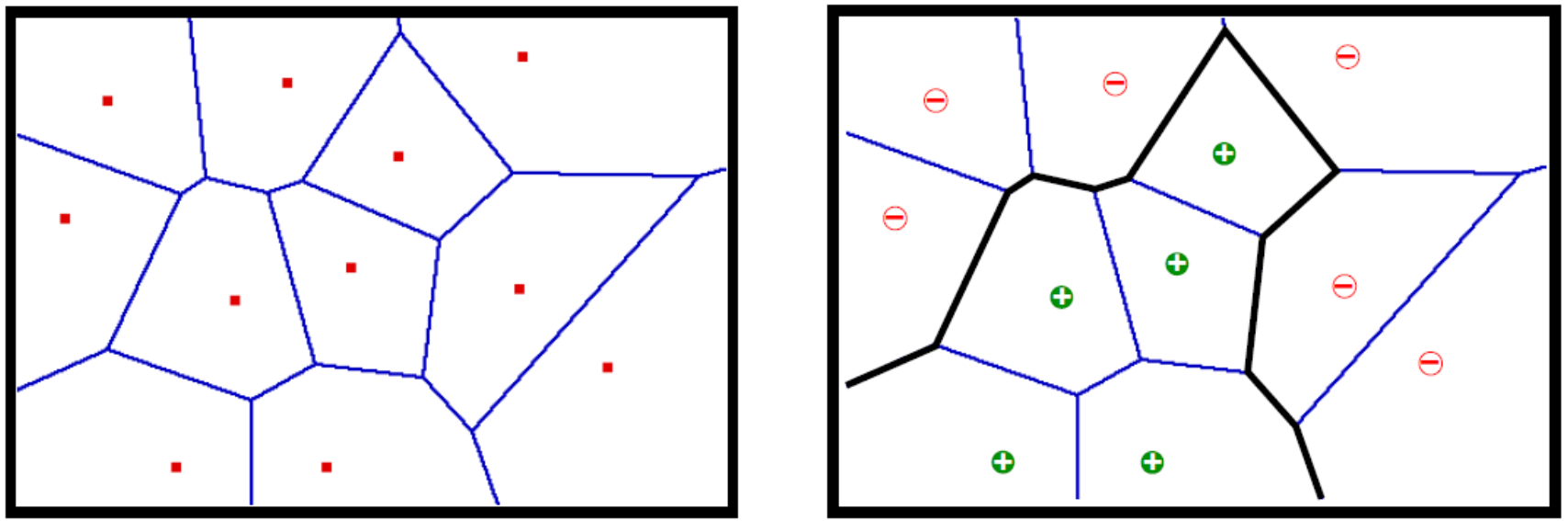
NearestNeighbour($M_+, M_-, \mathbf{s}$)

$\mathbf{t} = \text{argmin}_{\mathbf{x} \in M_+ \cup M_-} \{d(s, x)\}$

If $\mathbf{t} \in M_+$ Then Return("+")

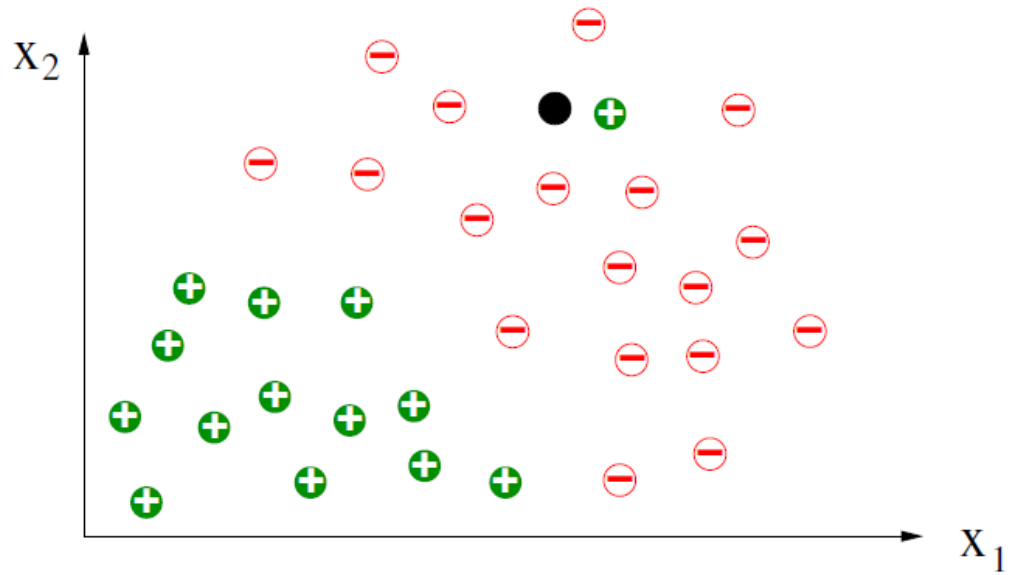
Else Return("−")

Klassentrennung (virtuell)



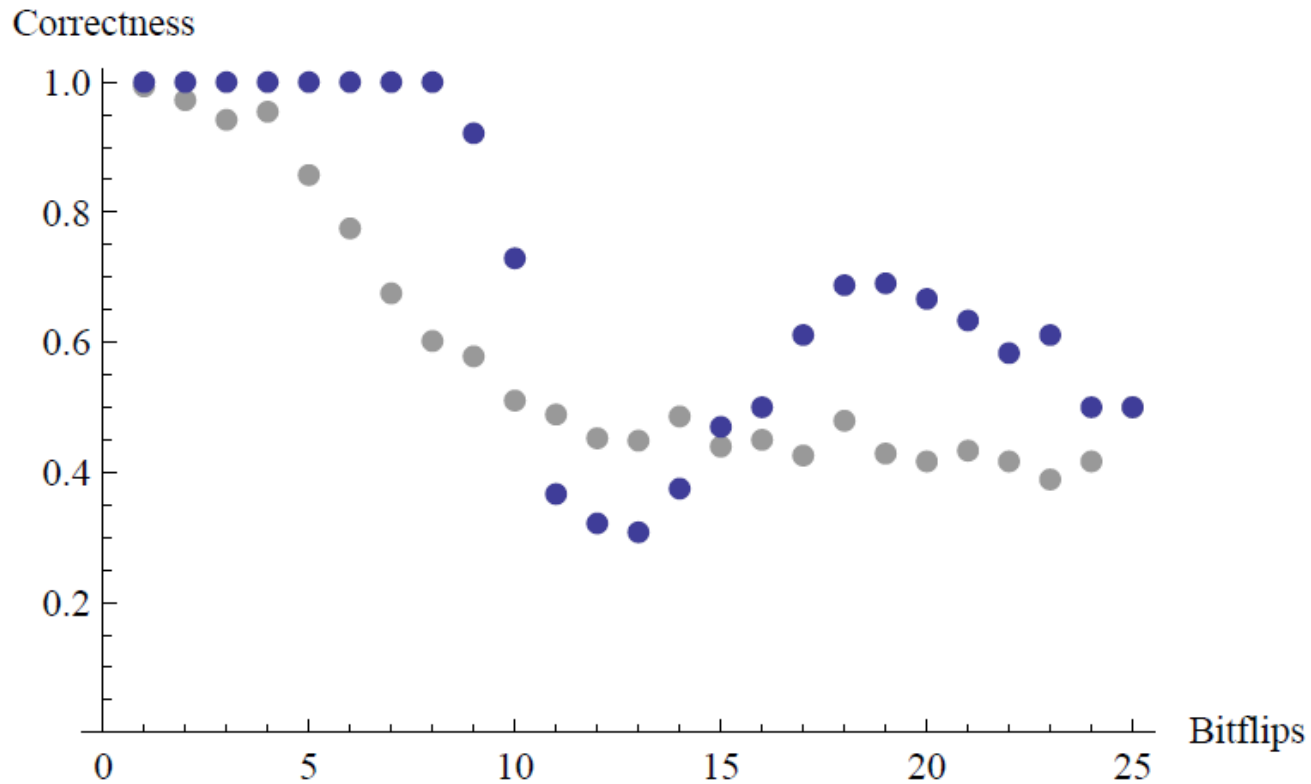
Voronoidiagramm (links) und Klassentrennlinie (rechts).

- Nearest-Neighbour-Methode ist mächtiger als das Prezeptron
- beliebig komplexe Trennlinien (Hyperflächen) möglich



Der schwarze Punkt ist falsch klassifiziert.

- Gefahr durch Ausreißer
- Fehlanpassungen an zufällige Fehler (Rauschen)
- **Überanpassung** (engl. overfitting).



Korrektheit der Nearest-Neighbour-Klassifikation (grau: Perzeptron)

k-Nearest-Neighbour-Methode

Mehrheitsentscheid unter den k nächsten Nachbarn:

Der Algorithmus k-NearestNeighbour.

k-NearestNeighbour($M_+, M_-, \mathbf{s}$)

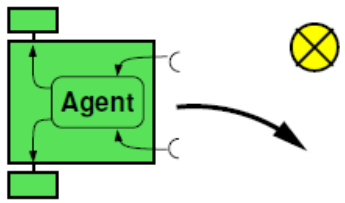
$V = \{k$ nächste Nachbarn in $M_+ \cup M_-\}$
If $|M_+ \cap V| > |M_- \cap V|$ **Return**(„+")
ElseIf $|M_+ \cap V| < |M_- \cap V|$ **Return**(„–")
Else **Return**(Random(„+", „–"))

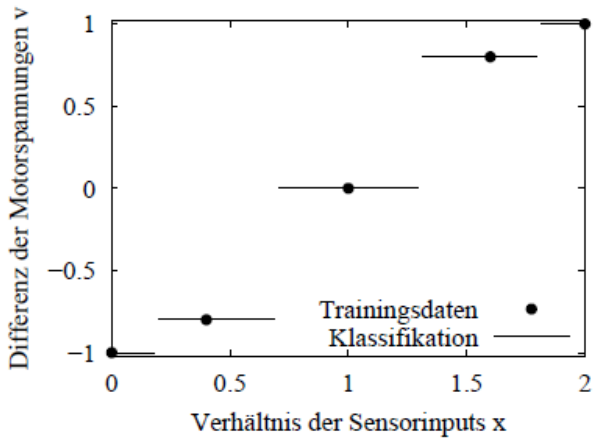
Mehr als zwei Klassen

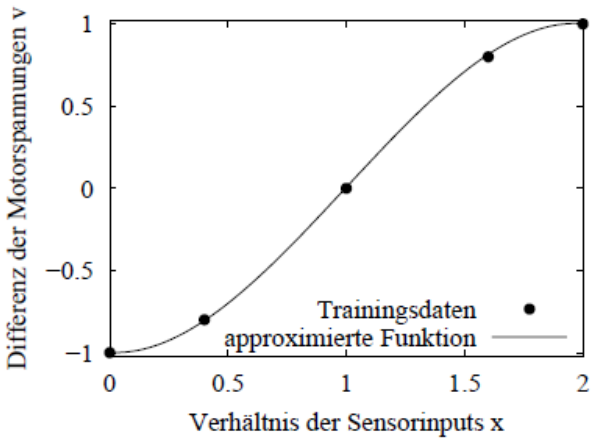
- Nearest-Neighbour-Klassifikation für mehr als zwei Klassen: Klasse des nächsten Nachbarn
- k-Nearest-Neighbour-Methode: Klasse mit Mehrheit unter den k nächsten Nachbarn.
- k-Nearest-Neighbour-Methode nur bei wenigen Klassen, bis etwa 10 Klassen

Klassifikation versus Approximation

Autonomer Roboter soll lernen, sich vom Licht wegzubewegen.







- Sensorsignale s_l des linken und s_r des rechten Sensors, $x = s_r / s_l$
- Aus x soll $v = U_r - U_l$ bestimmt werden.
- Der Agent muss eine Abbildung f finden, die für jeden Wert x den "richtigen" Wert $v = f(x)$ berechnet.

Approximationsmethoden

- Polynominterpolation, Spline-Interpolation, Methode der kleinsten Quadrate
- Problematisch in hohen Dimensionen.
- Schwierigkeit: in der KI werden modellfreie Approximationsmethoden benötigt
- Neuronale Netze, Supportvektormaschinen, Gauß'sche Prozesse, ..., Nearest Neighbour

k-NN für Approximationsprobleme

1. Bestimmung der Menge $V = \{\mathbf{x}_1, \mathbf{x}_2, \ldots \mathbf{x}_k\}$ der k nächsten Nachbarn von $\mathbf{x}$
2. mittlerer Funktionswert

$$\hat{f}(\mathbf{x}) = \frac{1}{k} \sum_{i=1}^{k} f(\mathbf{x}_i) \tag{28}$$

Je größer k wird, desto glatter wird die Funktion $\hat{f}$.

Der Abstand ist relevant

bei großem k:

- viele Nachbarn mit großem Abstand
- wenige Nachbarn mit kleinem Abstand.
- Berechnung von $\hat{f}$ durch weit entfernte Nachbarn dominiert

Lösung: in k-NearestNeighbour werden "Stimmen" gewichtet mit

$$w_i = \frac{1}{1 + \alpha d(\mathbf{x}, \mathbf{x}_i)^2}, \tag{29}$$

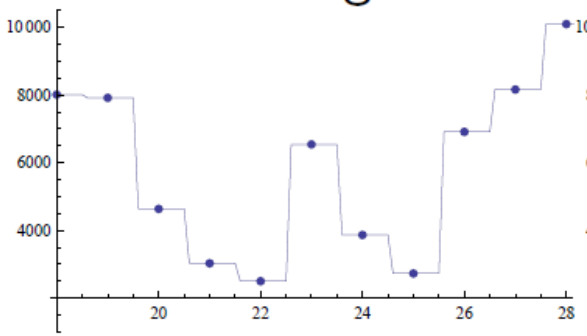
Approximation: Gleichung 28 wird ersetzt durch

$$\hat{f}(\mathbf{x}) = \frac{\sum_{i=1}^{k} w_i f(\mathbf{x}_i)}{\sum_{i=1}^{k} w_i}.$$

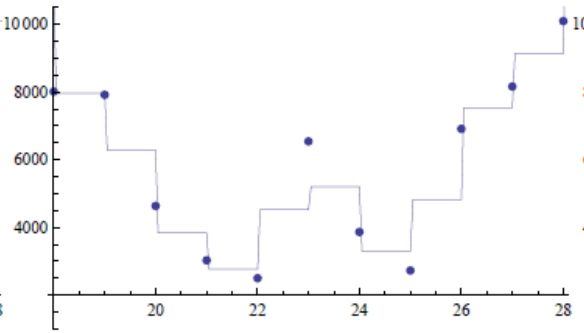
- Einfluß der Punkte für größer werdenden Abstand geht gegen Null.
- alle Trainingsdaten können zur Klassif./Approx. verwentet werden!

k-NN mit/ohne Abstandsgewichtung

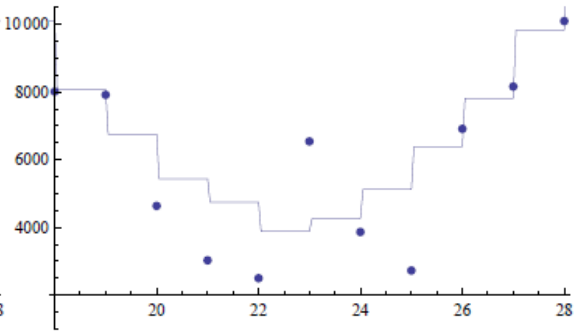
k-Nearest Neighbour



$k = 1$

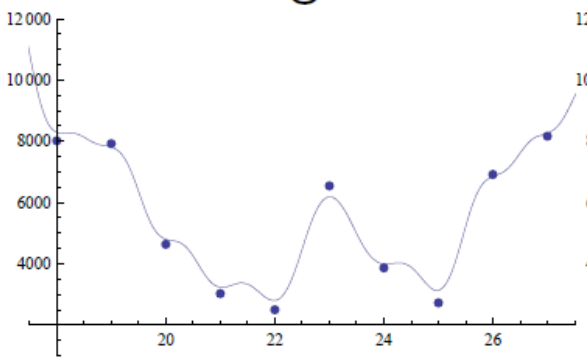


$k = 2$

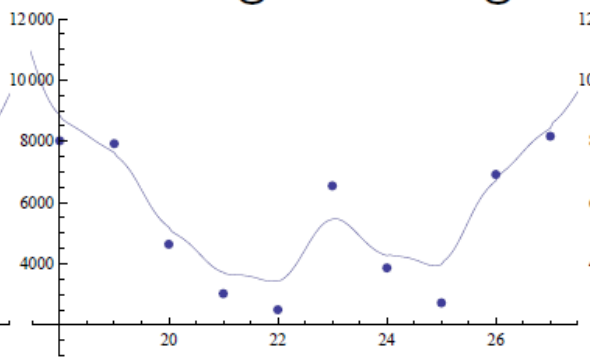


$k = 6$

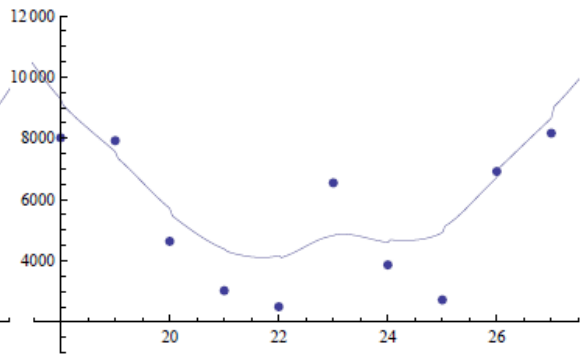
Nearest Neighbour mit Abstandsgewichtung



$\alpha = 20$



$\alpha = 4$



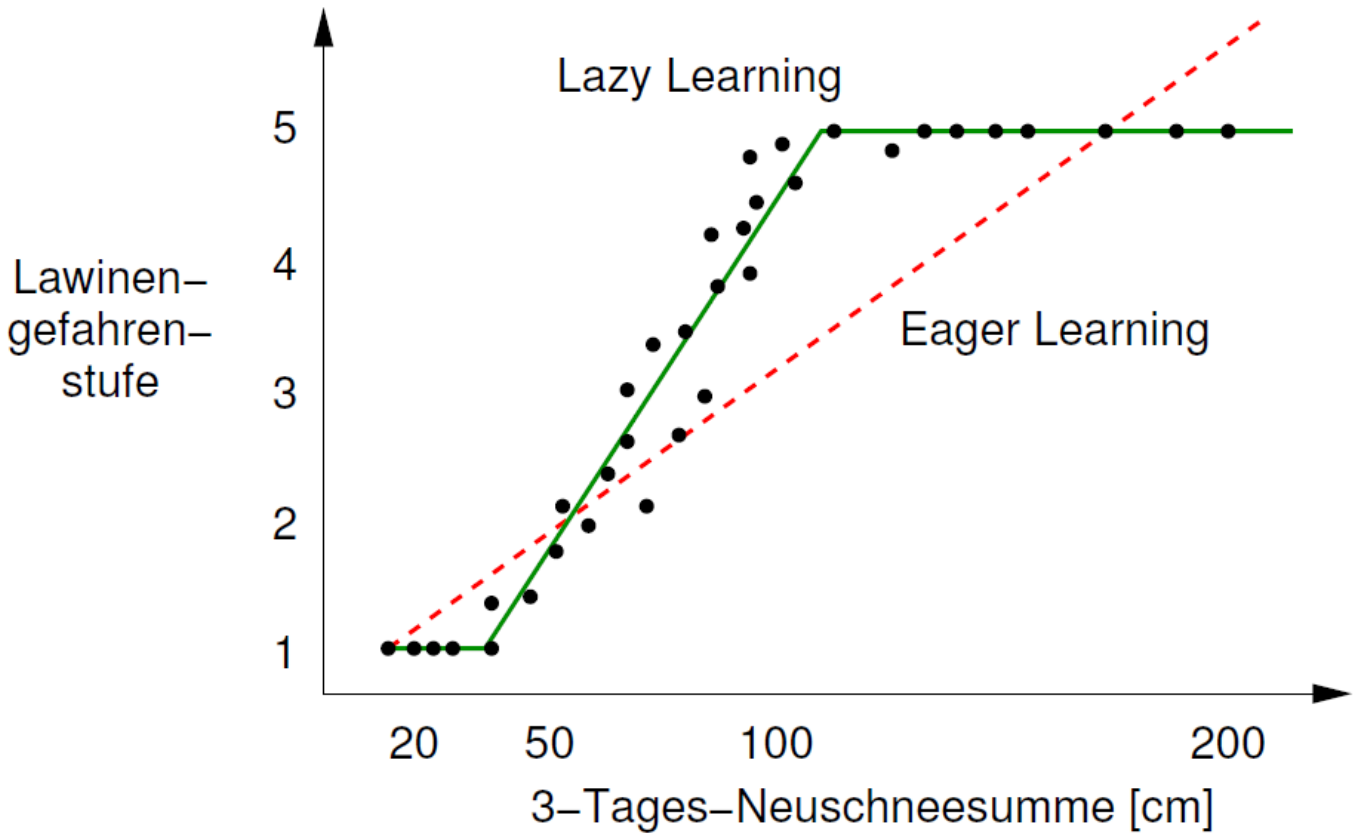
$\alpha = 1$

Rechenzeiten

- Lernen durch einfaches Abspeichern
- Daher sehr schnelles Lernen
- Klassifikation/Approximation eines Vektors $\mathbf{x}$ sehr aufwändig
- Finden der k nächsten Nachbarn bei n Trainingsdaten: $\Theta(n)$
- Klassifikation/Approximation: $\Theta(k)$
- Gesamtrechenzeit:$\Theta(n+k)$
- Nearest-Neighbour-Methoden = **Lazy Learning**

Beispiel: Lawinenprognose

Aufgabe: aus der Neuschneemenge ist die aktuelle Lawinengefahr zu bestimmen.



NN-Methoden: Einsatzgebiete

Nearest-Neighbour-Methoden sind gut geeignet,
wenn eine gute lokale Approximation benötigt wird, die aber keine hohen Anforderungen an die Geschwindigkeit des Systems stellen.

Nearest-Neighbour-Methoden sind schlecht geeignet,
wenn eine für Menschen verständliche Beschreibung des in den Daten enthaltenen Wissens gefordert wird (Data Mining).

Fallbasiertes Schließen

(engl. case-based reasoning), kurz CBR:

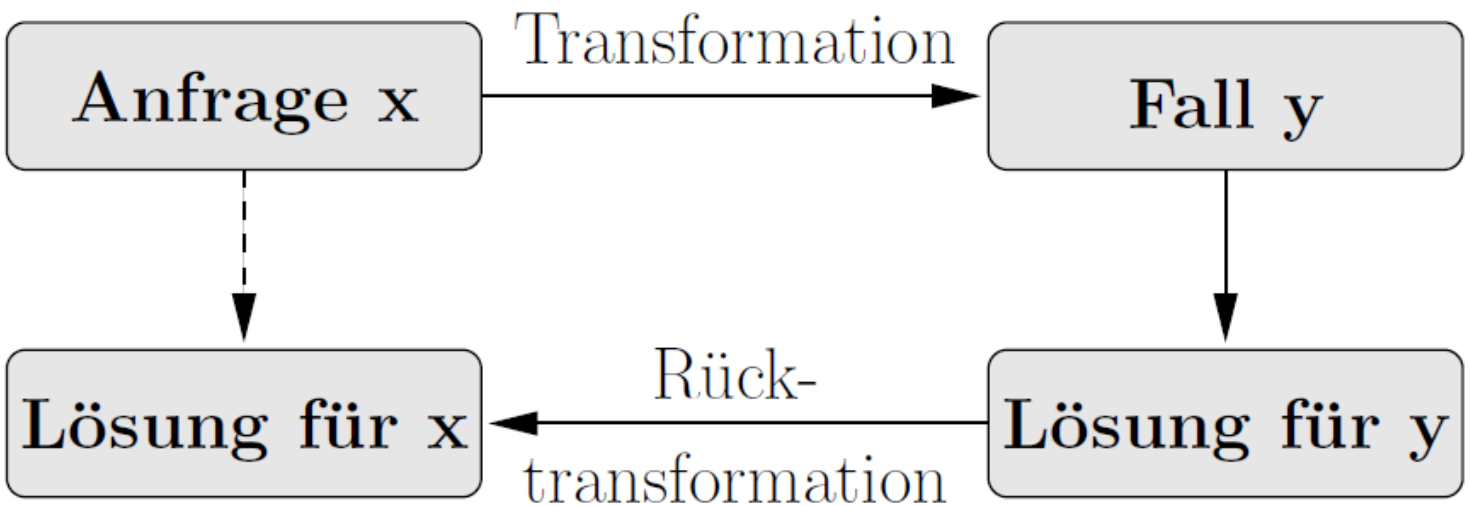
- Erweiterung von NN-Methoden auf symbolische Problembeschreibungen und deren Lösung
- Einsatzgebiete:
  - Diagnose technischer Probleme
  - Telefonhotlines

CBR-Beispiel

Merkmal	**Anfrage**	**Fall aus Fallbasis**
Defektes Teil:	Rücklicht	Vorderlicht
Fahrrad Modell:	Marin Pine Mountain	VSF T400
Baujahr:	1993	2001
Stromquelle:	Batterie	Dynamo
Zustand der Birnen:	ok	ok
Lichtkabelzustand:	?	ok
	Lösung	
Diagnose:	?	Massekontakt vorne fehlt
Reparatur:	?	Stelle Massekontakt vorne her

Transformation: Rücklicht auf Vorderlicht abbilden

CBR-Schema


Anfrage x
Transformation
Fall y
Lösung für x
Rück-
transformation
Lösung für y

CBR: Probleme

Modellierung Frame-Problem: Kann der Entwickler alle möglichen Spezialfälle und Problemvarianten vorhersehen und abbilden?

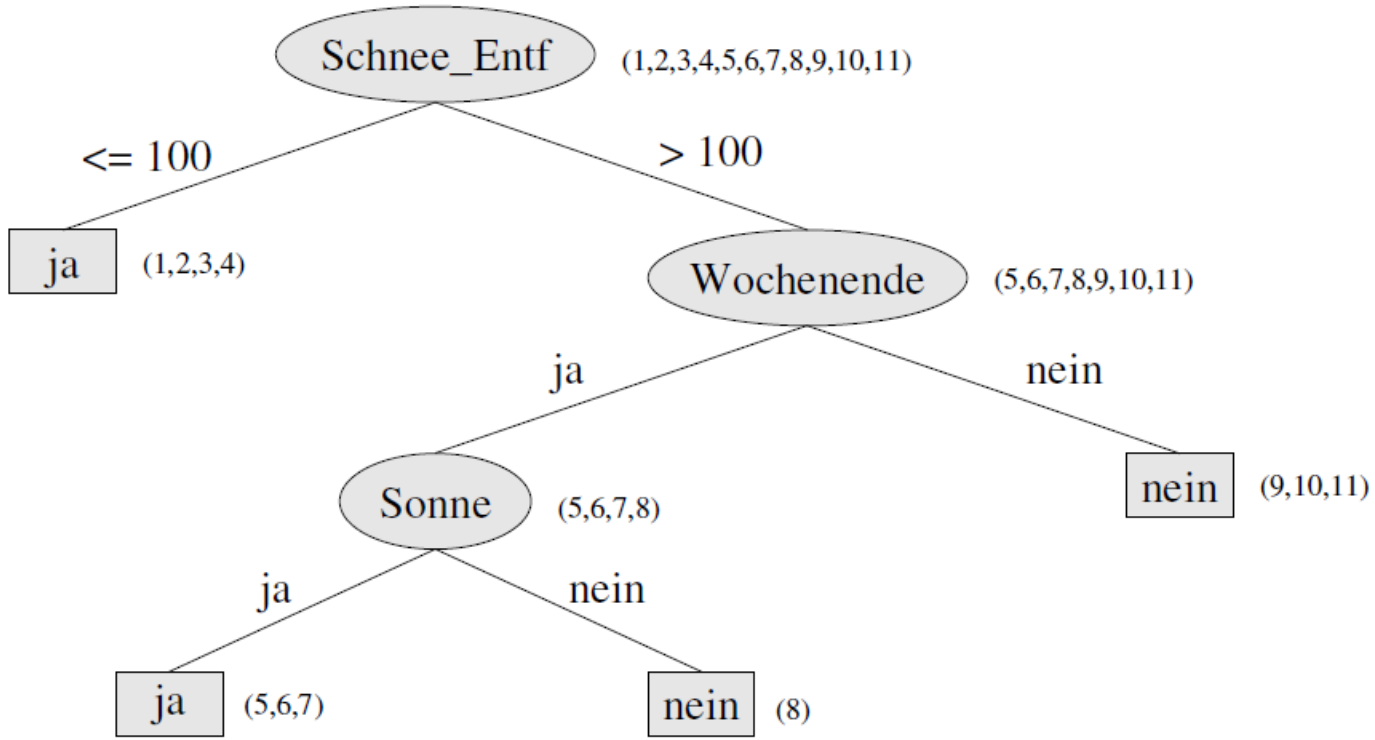
Ähnlichkeit passendes Ähnlichkeitsmass für symbolische Merkmale.

Transformation Wie findet man die Abbildung und deren Umkehrung?

Lösungen:

- Interessante Alternative zu CBR: Bayes-Netze.
- Symbolische Problemrepräsentation ist oft abbildbar auf numerische Merkmale.
- Dann sind andere Verfahren einsetzbar.

Lernen von Entscheidungsbäumen



Entscheidungsbaum für das Klassifikationsproblem Skifahren.

Bäume und Merkmale

Ein **Entscheidungsbaum** ist ein Baum, dessen innere Knoten Merkmale (Attribute) repräsentieren. Jede Kante steht für einen Attributwert. An jedem Blattknoten ist ein Klassenwert angegeben.

Variable	Werte	Beschreibung
Skifahren (Zielvariable)	ja, nein	Fahre ich los in das nächstgelegene Skigebiet mit ausreichend Schnee?
Sonne	ja, nein	Sonne scheint heute?
Schnee_Entf	≤ 100, > 100	Entfernung des nächstes Skigebiets mit guten Schneeverhältnissen (über/unter 100 km)
Wochenende	ja, nein	Ist heute Wochenende?

Variablen für das Klassifikationsproblem Skifahren.

Trainingsdaten

Tag Nr.	Schnee_Entf	Wochenende	Sonne	Skifahren
1	≤ 100	ja	ja	ja
2	≤ 100	ja	ja	ja
3	≤ 100	ja	nein	ja
4	≤ 100	nein	ja	ja
5	> 100	ja	ja	ja
6	> 100	ja	ja	ja
7	> 100	ja	ja	nein
8	> 100	ja	nein	nein
9	> 100	nein	ja	nein
10	> 100	nein	ja	nein
11	> 100	nein	nein	nein

Zeilen 6 und 7 widersprechen sich! Baum aus Abbildung 449 ist also optimal!

Wie entsteht der Baum aus den Daten?

optimaler Algorithmus:
alle Bäume erzeugen, für jeden Baum die Zahl der Fehlklassifikationen auf den Daten berechnen und schließlich einen Baum mit minimaler Fehlerzahl auswählen.

Nachteil: inakzeptabel hohe Rechenzeit!

also: heuristischer Algorithmus mit Greedy-Strategie

Die Entropie als Maß für den Informationsgehalt

- Attribut mit höchstem Informationsgewinn
- Trainingsdaten $S = (\text{ja,ja,ja,ja,ja,ja,nein,nein,nein,nein,nein})$ mit

$$p_1 = p(\text{ja}) = 6/11 \quad \text{und} \quad p_2 = p(\text{nein}) = 5/11.$$

- Wahrscheinlichkeitsverteilung

$$\mathbf{p} = (p_1, p_2) = (6/11, 5/11).$$

- allgemein

$$\mathbf{p} = (p_1, \ldots, p_n)$$

mit

$$\sum_{i=1}^{n} p_i = 1.$$

zwei Extremfälle

sicheres Ereignis:

$$\mathbf{p} = (1, 0, 0, \ldots, 0). \tag{30}$$

Gleichverteilung:

$$\mathbf{p} = (\frac{1}{n}, \frac{1}{n}, \ldots, \frac{1}{n}) \tag{31}$$

Ungewissheit ist maximal!

Claude Shannon:
wieviele Bits benötigt man mindestens, um solch ein Ereignis zu kodieren?

1. (Gleichung 30): Null Bits
2. (Gleichung 31): $\log_2 n$ Bits

allgemeiner Fall

$$\mathbf{p} = (p_1, \ldots, p_n) = (\frac{1}{m_1}, \frac{1}{m_2}, \ldots, \frac{1}{m_n}) \tag{32}$$

- $\log_2 m_i$ Bits für den i-ten Fall benötigt.
- Erwartungswert H für die Zahl der Bits:

$$H = \sum_{i=1}^{n} p_i \log_2 m_i = \sum_{i=1}^{n} p_i \log_2 1/p_i = -\sum_{i=1}^{n} p_i \log_2 p_i.$$

- Je mehr Bits man benötigt, um ein Ereignis zu kodieren, desto höher ist die Unsicherheit über den Ausgang.
- Daher: **Entropie** H als Maß für die Unsicherheit einer Wahrscheinlichkeitsverteilung:

$$H(\mathbf{p}) = H(p_1, \ldots, p_n) := -\sum_{i=1}^{n} p_i \log_2 p_i.$$

Problem: $0 \log_2 0$ undefiniert!
Definition $0 \log_2 0 := 0$ (siehe).
damit:

$$H(1, 0, \ldots, 0) = 0$$

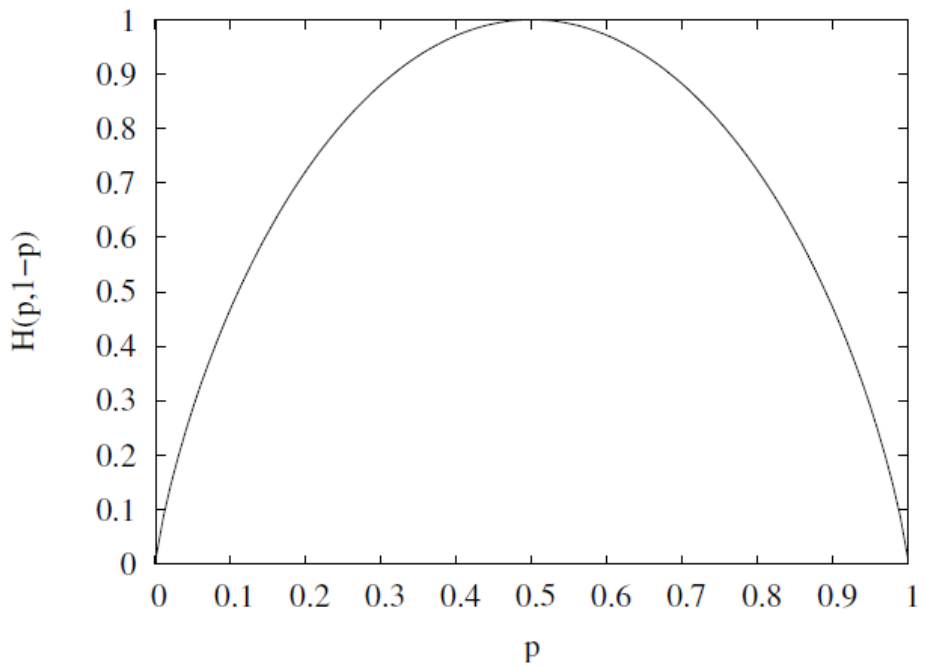
und

$$H(\frac{1}{n}, \ldots, \frac{1}{n}) = \log_2 n$$

eindeutiges globales Maximum der Entropie!

Spezialfall: zwei Klassen

$$H(\mathbf{p}) = H(p_1, p_2) = H(p_1, 1-p_1) = -(p_1 \log_2 p_1 + (1-p_1) \log_2 (1-p_1))$$



Die Entropiefunktion für den Zweiklassenfall.

Datenmenge D mit Wahrscheinlichkeitsverteilung $\mathbf{p}$:

$$H(D) = H(\mathbf{p}).$$

Informationsgehalt $I(D)$ der Datenmenge D ist das Gegenteil von Unsicherheit, also

$$I(D) := 1 - H(D). \tag{33}$$

Der Informationsgewinn

Wenden wir nun die Entropieformel an auf das Beispiel, so ergibt sich

$$H(6/11, 5/11) = 0.994$$

Informationsgewinn

$$\mathrm{G(D,A)} = \sum_{i=1}^{n} \frac{|D_i|}{|D|} I(D_i) - I(D)$$

Mit Gleichung 33 erhalten wir

$$\begin{aligned} \text{G(D,A)} &= \sum_{i=1}^{n} \frac{|D_i|}{|D|} I(D_i) - I(D) = \sum_{i=1}^{n} \frac{|D_i|}{|D|} (1 - H(D_i)) - (1 - H(D \\ &= 1 - \sum_{i=1}^{n} \frac{|D_i|}{|D|} H(D_i) - 1 + H(D) \\ &= H(D) - \sum_{i=1}^{n} \frac{|D_i|}{|D|} H(D_i) \end{aligned}$$

Angewendet auf Schnee_Entf:

$$\begin{aligned} \text{G(D,Schnee\_Entf)} &= H(D) - \left(\frac{4}{11} H(D_{\leq 100}) + \frac{7}{11} H(D_{>100}) \right) \\ &= 0.994 - \left(\frac{4}{11} \cdot 0 + \frac{7}{11} \cdot 0.863 \right) = 0.445 \end{aligned}$$

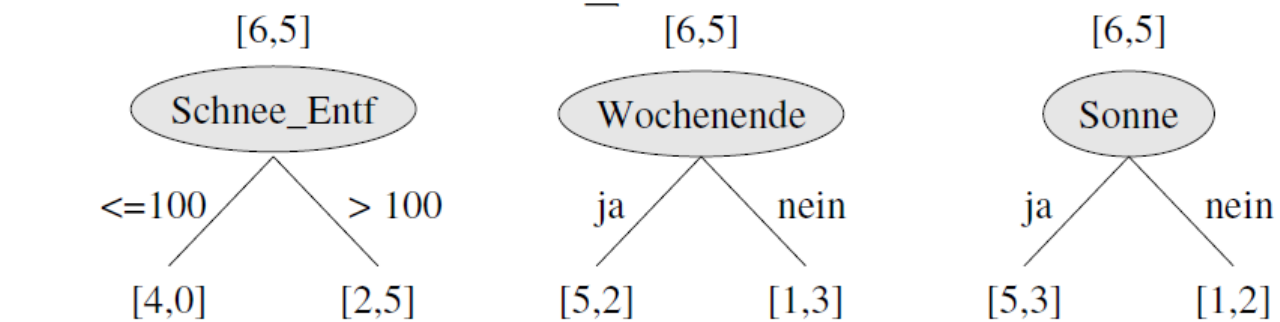
analog erhält man

$$G(D,\text{Wochenende}) = 0.150$$

und

$$G(D,\text{Sonne}) = 0.049$$

also: Wurzelknoten wird Schnee_Entf



Gewinn: 0.445 0.150 0.049

Vergleich

Tag Nr.	Schnee_Entf	Wochenende	Sonne	Skifahren
1	≤ 100	ja	ja	ja
2	≤ 100	ja	ja	ja
3	≤ 100	ja	nein	ja
4	≤ 100	nein	ja	ja
5	> 100	ja	ja	ja
6	> 100	ja	ja	ja
7	> 100	ja	ja	nein
8	> 100	ja	nein	nein
9	> 100	nein	ja	nein
10	> 100	nein	ja	nein
11	> 100	nein	nein	nein

Datenmenge für das Klassifikationsproblem Skifahren.

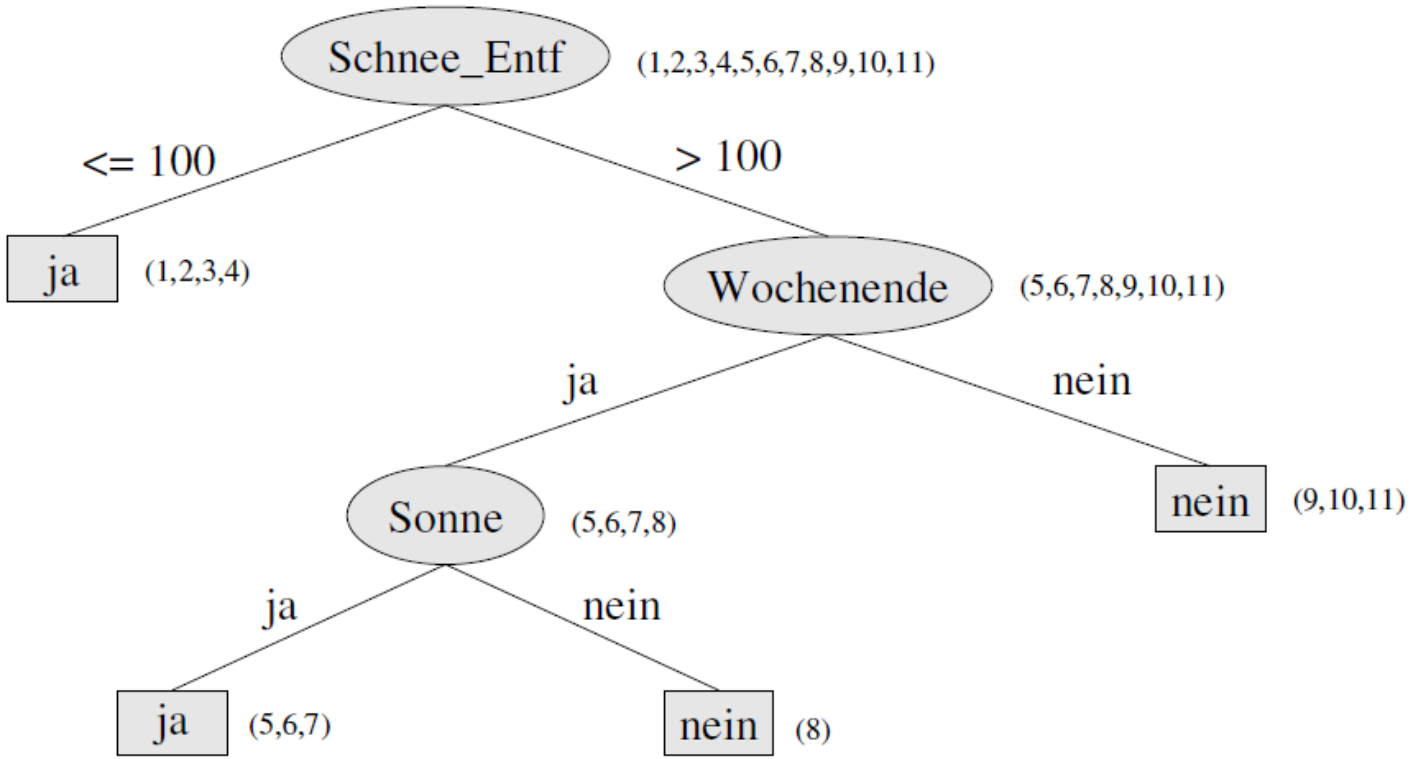
Für $D_{\leq 100}$ ist die Klassifikation eindeutig **ja**.
Also terminiert der Baum hier.

Im Zweig $D_{>100}$ herrscht keine Eindeutigkeit. Also

$$\text{InfGewinn}(D_{>100}, \text{Wochenende}) = 0.292$$

$$\text{InfGewinn}(D_{>100}, \text{Sonne}) = 0.170$$

Der Knoten erhält also das Attribut **Wochenende**.
Für **Wochenende = nein** terminiert der Baum mit der Entscheidung **nein**.
Für **Wochenende = ja** bringt **Sonne** einen Gewinn von 0.171.
Aufbau des Baumes beendet, weil keine weiteren Attribute mehr verfügbar sind.



Entscheidungsbaum für das Klassifikationsproblem Skifahren.

Systeme

C4.5 1993, Ross Quinlan, Weiterentwicklung von **ID3** (Iterative Dichotomiser 3, 1986). Nachfolger (kostenpflichtig): C5.0

CART (Classification and Regression Trees), 1984, Leo Breiman, ähnlich zu C4.5, komfortable graphische Benutzeroberfläche, teuer.

CHAID (Chi-square Automatic Interaction Detectors), 1964, J. Sonquist und J. Morgan, stoppt das Wachsen des Baumes bevor er zu groß wird. Heute bedeutungslos

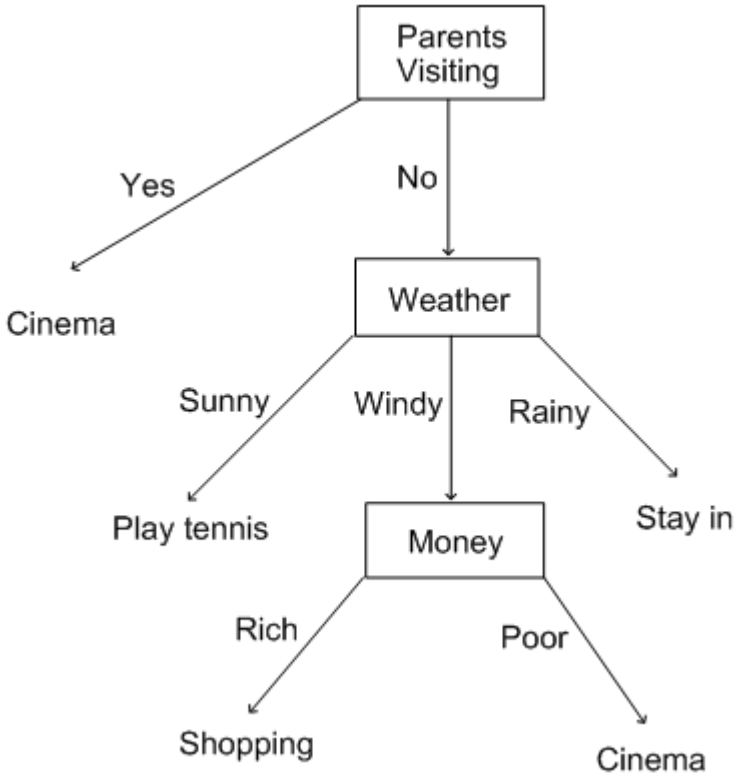
Knime Data Mining Werkzeug (Konstanz Information Miner), sehr komfortable Benutzeroberfläche, Verwendet die **Weka** Java-Bibliothek.

C4.5 Algorithmus

- C4.5 ist eine Suite von Algorithmen für Klassifizierungsprobleme in Machine Learning und Data Mining, auch Entscheidungsbaumalgorithmen genannt.
- Es konzentriert sich auf supervised learning:
  - Gegeben ein Datensatz, in dem *Instanzen* durch Auflistungen von *Attributen* beschrieben werden und zu einer Gruppe sich gegenseitig ausschließender *Klassen* gehören, beschreibt C4.5 eine Funktion von den Attributwerten in den Klassen, die zum Klassifizieren neuer Instanzen verwendet werden kann.

C4.5 Algorithmus

- Der C4.5-Algorithmus wird in Data Mining als *Decision Tree Classifier* verwendet, mit dem eine Entscheidung basierend auf einer bestimmten Datenprobe (univariate oder multivariate Prädiktoren) generiert werden kann.
- Decision trees:



Beispiel

Day	Outlook	Temperature	Humidity	Windy	Play Golf?
1	Sunny	85	85	False	No
2	Sunny	80	90	True	No
3	Overcast	83	78	False	Yes
4	Rainy	70	96	False	Yes
5	Rainy	68	80	False	Yes
6	Rainy	65	70	True	No
7	Overcast	64	65	True	Yes
8	Sunny	72	95	False	No
9	Sunny	69	70	False	Yes
10	Rainy	75	80	False	Yes
11	Sunny	75	70	True	Yes
12	Overcast	72	90	True	Yes
13	Overcast	81	75	False	Yes
14	Rainy	71	80	True	No

Beispiel

- Jede Zeile bezeichnet eine Instanz, die durch Werte für Attribute beschrieben wird, wie z. B. *Outlook* (Zufallsvariable mit ternärem Wert), *Temperatur* (kontinuierlich), *Feuchte* (auch kontinuierlich) und *Windig* (binär). Die Klassenvariable PlayGolf? ist boolesch.
- Gesucht wird eine Funktion, welche aus diesem Datensatz abgeleitet wird. Diese Funktion wird dann auf andere Instanzen angewandt, in denen Werte nur für die Attribute vorhanden sind, um den Wert für die zufällige Klassenvariable vorherzusagen.

Einführung in C4.5

- C4.5, entworfen von J. Ross Quinlan, ist so benannt, weil es ein Nachfolger des ID3-Ansatzes ist, um Entscheidungsbäume zu erzeugen, die wiederum die dritte Inkarnation einer Reihe von "iterativen Dichotomizern" sind.
- Ein *Entscheidungsbaum* ist eine Reihe von Fragen, die systematisch so angeordnet sind, dass jede Frage ein Attribut (z. B. *Outlook*) und Verzweigungen basierend auf dem Wert des Attributs abfragt. Die Blättern des Baumes liefern Vorhersagen der Klassenvariablen (hier *PlayGolf?*).
- C4.5 kann nicht nur Bäume erzeugen, sondern auch Bäume in verständlicher Regelform darstellen. Darüber hinaus führen die von C4.5 unterstützten Regelvorbereitungsvorgänge (postpruning operations) der Regel zu Klassifizierern, die nicht als Entscheidungsbaum dargestellt werden können.

Einführung in C4.5

- Pruning bezieht sich auf das Entfernen der Zweige im Entscheidungsbaum, von denen wir glauben, dass sie nicht wesentlich zum Entscheidungsprozess beitragen.
- Der Begriff pruning hilft ein Überanpassen (overfitting) des Regressions- oder Klassifikationsmodells zu vermeiden, sodass bei einem kleinen Datensatz keine Messfehler bei der Generierung des Modells berücksichtigt werden.

C4.5 vereinfachte Version

1. Check **for base** cases.
2. For each attribute *a*, find the normalised information gain ratio **from** splitting on *a*.
3. Let *a_best* be the attribute **with** the highest normalized information gain.
4. Create a decision node that splits on *a_best*.
5. Recur on the sublists obtained **by** splitting on *a_best*, **and** add those nodes **as** children of node.

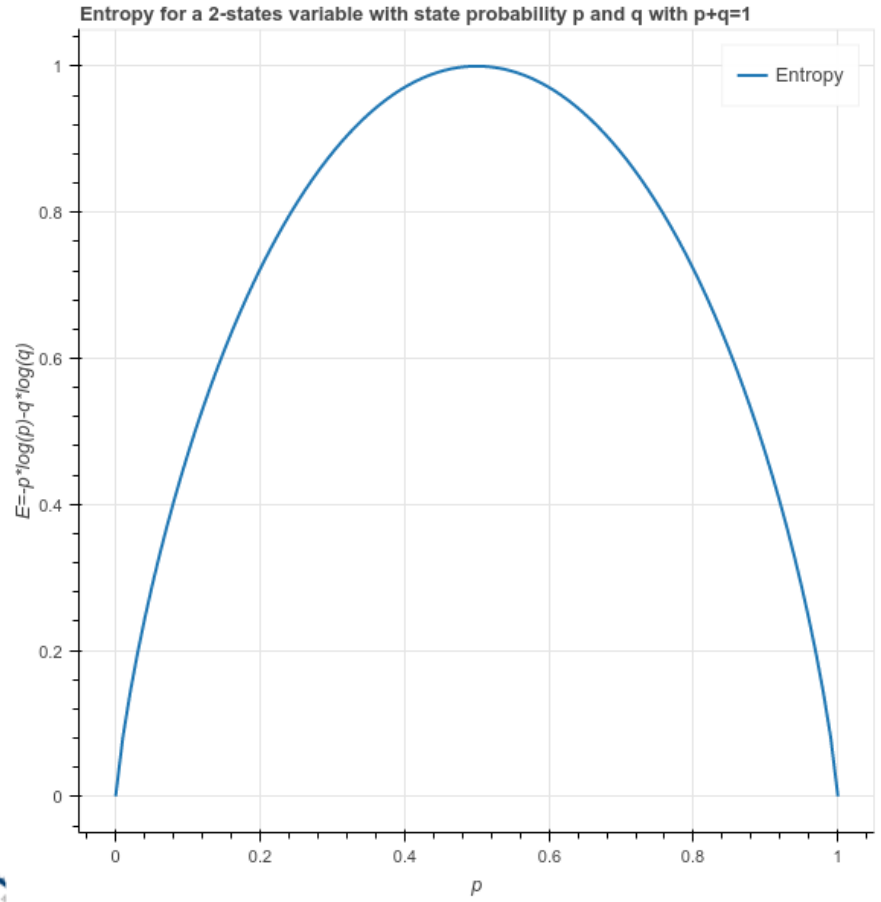
Information Gain

- Wenn man im Laufe der Zeit Informationen erhalten hat, mit denen man genau vorhersagen kann, ob etwas passieren wird, handelt es sich zu dem vorhergesagten Ereignis nicht um neue Informationen. Wenn die Situation jedoch in falscher Richtung verläuft und ein unerwartetes Ergebnis auftritt, zählt dies als nützliche und notwendige Information.
- *Je mehr man über ein Thema weißt, desto weniger neue Informationen erhält man darüber. Um es kurz zu fassen: Wenn Sie wissen, dass ein Ereignis sehr wahrscheinlich ist, ist es nicht überraschend, wenn es passiert, d.h. es gibt nur wenige Zusatzinformationen*.

Information Gain

- Die Menge der gewonnenen Informationen ist umgekehrt proportional zur Wahrscheinlichkeit eines Ereignisses.
- D.h., mit zunehmender Entropie sinkt der Informationsgewinn.
- Dies ist darauf zurückzuführen, dass Entropie sich auf die Wahrscheinlichkeit eines Ereignisses bezieht.

Entropie



- Die Entropie ist maximal, wenn die Wahrscheinlichkeiten beider Ereignisse gleich sind (hier 0,5).
- Im Fall von Decision Trees müssen die Knoten so ausgerichtet sein, dass die Entropie mit dem *splitting* abnimmt.
- Dies bedeutet: umso mehr *splitting*, umso einfacher ist es, eine bestimmte Entscheidung zu treffen.

Information Gain

- Wir überprüfen jeden Knoten auf splitting Möglichkeit.
- **Information Gain ist das Verhältnis der Beobachtungen zur Gesamtzahl der Beobachtungen**

(m/N = p) und (n/N = q) wobei m+n=N und p+q=1.

Nach dem splitting, falls die Entropie des nächsten Knotens vor dem Aufteilen (splitting) kleiner ist und wenn dieser Wert der kleinste Wert ist, wird der Knoten in seine reinsten Bestandteile aufgeteilt.

Anwendungen

- clinical decision making,
- manufacturing,
- document analysis,
- bioinformatics,
- spatial data modeling (geographic information systems),
- und praktisch jede Domäne, in der die Grenzen zwischen Klassen anhand von Baumzerlegungen oder durch Regeln identifizierten Regionen erfasst werden können.

Algorithmus

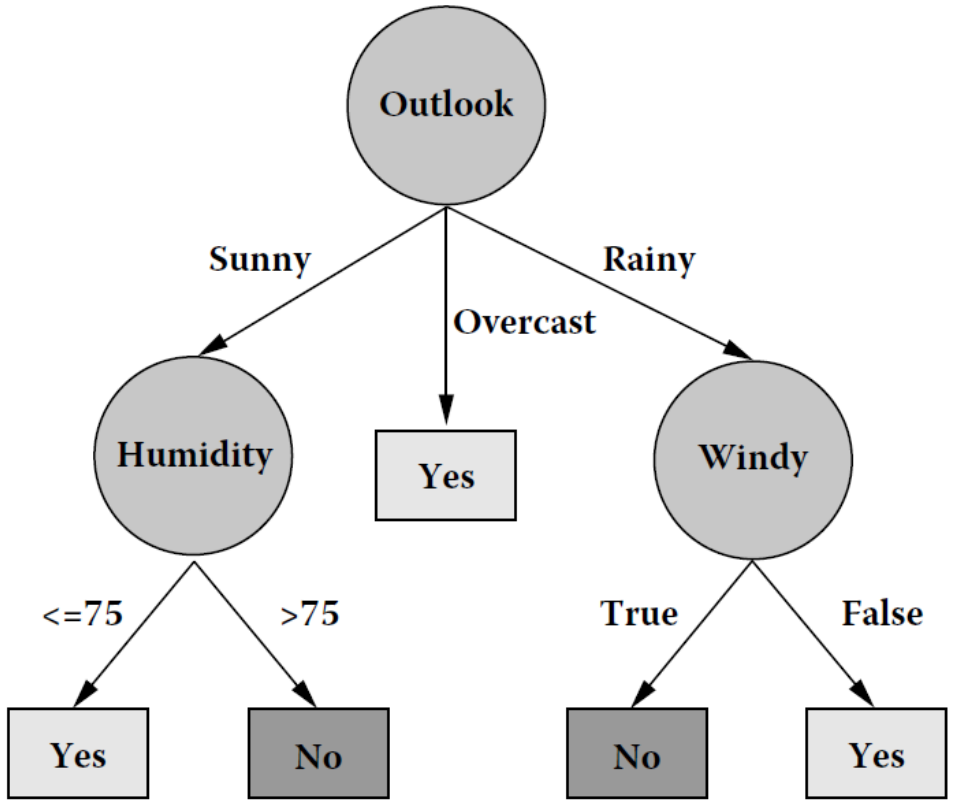
Algorithm 1.1 C4.5(D)

Input: an attribute-valued dataset D

```
1:  Tree = {}
2:  if D is "pure" OR other stopping criteria met then
3:      terminate
4:  end if
5:  for all attribute a ∈ D do
6:      Compute information-theoretic criteria if we split on a
7:  end for
8:  a_best = Best attribute according to above computed criteria
9:  Tree = Create a decision node that tests a_best in the root
10: D_v = Induced sub-datasets from D based on a_best
11: for all D_v do
12:     Tree_v = C4.5(D_v)
13:     Attach Tree_v to the corresponding branch of Tree
14: end for
15: return Tree
```

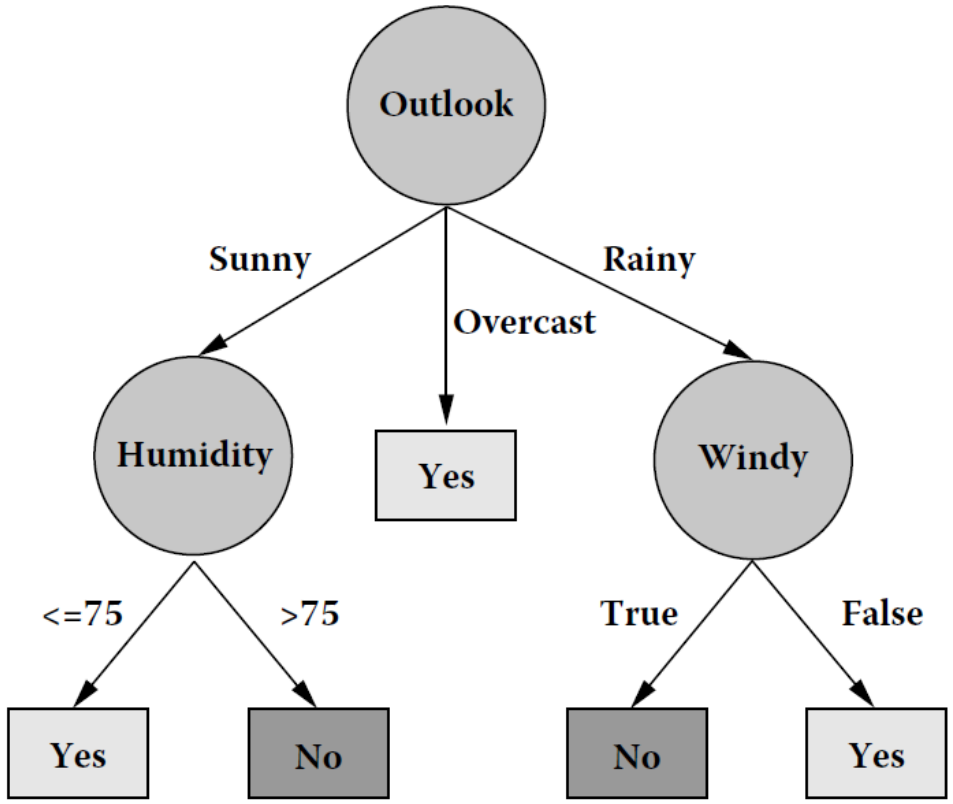
- Alle Bauminduktionsmethoden beginnen mit einem Wurzelknoten, der das gesamte, gegebene Dataset darstellt, und rekursiv die Daten in kleinere Teilmengen aufspaltet, indem an jedem Knoten ein bestimmtes Attribut getestet wird.
- Die Unterbäume geben die Partitionen des ursprünglichen Datensatzes an, die den angegebenen Attributwerttests entsprechen. Dieser Prozess wird normalerweise fortgesetzt, bis die Teilmengen "rein" sind, d. h. alle Instanzen in der Teilmenge fallen in dieselbe Klasse. Zu diesem Zeitpunkt ist das Baumwachstum beendet.

Induced decision tree



- Ziel ist es, vorherzusagen, ob die Wetterbedingungen an einem bestimmten Tag zum Golfspielen günstig sind.

Testarten



- C4.5 ist nicht auf binäre Teste beschränkt und ermöglicht Teste mit zwei oder mehrere Ergebnisse. Wenn das Attribut boolesch ist, generiert der Test zwei Zweige.
- If the attribute is categorical, the test is multivalued, but different values can be grouped into a smaller set of options with one class predicted for each option.
- Wenn das Attribut kategorial ist, so ist der Test mehrwertig, aber verschiedene Werte können in einer kleineren Menge von Optionen gruppiert werden, wobei für jede Option eine Klasse vorhergesagt wird.
- Wenn das Attribut numerisch ist, sind die Tests erneut binär und haben die Form {≤ x?,> x?}, wobei x der geeignete Schwellenwert für dieses Attribut ist.

Teste & Schwellwerte

C4.5 uses information-theoretic criteria such as:

- gain (reduction in entropy of the class distribution due to applying a test) and
- gain ratio (a way to correct for the tendency of gain to favor tests with many outcomes).
- The default criterion is gain ratio. At each point in the tree-growing, the test with the best criteria is greedily chosen.

- For Boolean and categorical attributes, the test values are simply the different possible instantiations of that attribute.
- For numerical attributes, the threshold is obtained by sorting on that attribute and choosing the split between successive values that maximize the criteria above.

Exercise C4.5

- Observe how the first attribute chosen for a decision test is the *Outlook* attribute.
- To see why, let us first estimate the entropy of the class random variable (*PlayGolf?*).
- This variable takes two values with probability 9/14 (for "Yes") and 5/14 (for "No").
- The entropy of a class random variable that takes on c values with probabilities $p_1, p_2, \ldots, p_c$ is given by:

$$\sum_{i=1}^{c} -p_i \log_2 p_i$$

The entropy of *PlayGolf?* is thus

$$-(9/14)\log_2(9/14) - (5/14)\log_2(5/14)$$

or 0.940. This means that on average 0.940 bits must be transmitted to communicate information about the *PlayGolf?* random variable. The goal of C4.5 tree induction is to ask the right questions so that this entropy is reduced. We consider each attribute in turn to assess the improvement in entropy that it affords. For a given random variable, say *Outlook*, the improvement in entropy, represented as *Gain(Outlook)*, is calculated as:

$$\text{Entropy}(\textit{PlayGolf?} \text{ in } D) - \sum_{v} \frac{|D_v|}{|D|} \text{Entropy}(\textit{PlayGolf?} \text{ in } D_v)$$

where v is the set of possible values (in this case, three values for *Outlook*), D denotes the entire dataset, D_v is the subset of the dataset for which attribute *Outlook* has that value, and the notation $|\cdot|$ denotes the size of a dataset (in the number of instances).

Exercise C4.5

- *Gain(Outlook)* is 0.940−0.694 = 0.246
- *Gain(Windy)* is 0.940−0.892 = 0.048
- Working out the above calculations for the other attributes systematically will reveal that *Outlook* is indeed the best attribute to branch on.
- This is a greedy choice and does not take into account the effect of future decisions.
- The tree-growing continues till termination criteria such as purity of subdatasets are met.
- In the above example, branching on the value "Overcast" for *Outlook* results in a pure dataset, that is, all instances having this value for *Outlook* have the value "Yes" for the class variable *PlayGolf?*; hence, the tree is not grown further in that direction.
- However, the other two values for *Outlook* still induce impure datasets.

Exercise C4.5

- Therefore the algorithm recurses, but observe that *Outlook* cannot be chosen again (why?).
- For different branches, different test criteria and splits are chosen, although, in general, duplication of subtrees can possibly occur for other datasets.
- The default splitting criterion is actually the gain ratio, not the gain.
- To understand the difference, assume we treated the *Day* column as a nominal valued attribute.
- Of course, each day is unique, so *Day* is really not a useful attribute to branch on.
- Nevertheless, because there are 14 distinct values for *Day* and each of them induces a “pure” dataset (a trivial dataset involving only one instance), *Day* would be unfairly selected as the best attribute to branch on.

More details on C4.5

- The gain ratio for an attribute *a* is defined as:

$$\text{GainRatio}(a) = \frac{\text{Gain}(a)}{\text{Entropy}(a)}$$

- Observe that entropy(*a*) does not depend on the class information and simply takes into account the distribution of possible values for attribute *a*, whereas gain(*a*) does take into account the class information. (Also, recall that all calculations here are dependent on the dataset used, although we haven't made this explicit in the notation.)

- For instance, GainRatio(*Outlook*) = 0.246/1.577 = 0.156.
- Similarly, the gain ratio for the other attributes can be calculated.

- C4.5 Features:
  - Tree Pruning
  - Improved use of continuous attributes
  - Handling Missing Values
  - Inducing Rulesets

Available Software Implementations

- J. Ross Quinlan's original implementation of C4.5 is available at his personal site:

http://www.rulequest.com/Personal/.

- Many public domain implementations of C4.5 are available, for example, Ronny Kohavi's MLC++library, which is now part of SGI's Mineset data mining suite, and the Weka data mining suite from the University of Waikato, New Zealand (http://www.cs.waikato.ac.nz/ml/weka/).
- The (Java) implementation of C4.5 in Weka is referred to as J48.
- Commercial implementations of C4.5 include ODBCMINE from Intelligent Systems Research,
- LLC, which interfaces with ODBC databases and
- Rulequest's See5/C5.0, which improves upon C4.5 in many ways and which also comes with support for ODBC connectivity.

Anwendung von C4.5

Trainingsdaten in Datei `ski.data`:

```
<=100, ja  , ja  , ja
<=100, ja  , ja  , ja
<=100, ja  , nein, ja
<=100, nein, ja  , ja
>100, ja  , ja  , ja
>100, ja  , ja  , ja
>100, ja  , ja  , nein
>100, ja  , nein, nein
>100, nein, ja  , nein
>100, nein, ja  , nein
>100, nein, nein, nein
```

Informationen über Attribute und Klassen in Datei `ski.names`:

```
|Klassen:     nein: nicht Skifahren, ja: Skifahren
|
nein,ja.
|
|Attribute
|
Schnee_Entf:                <=100,>100.
Wochenende:                 nein,ja.
Sonne:                      nein,ja.
```

```
unixprompt> c4.5 -f ski -m 1
C4.5 [release 8] decision tree generator        Wed Aug 2
----------------------------------------
|
    Options:
        File stem <ski>
        Sensible test requires 2 branches with >=1 cases
|
Read 11 cases (3 attributes) from ski.data
|
Decision Tree:
|
Schnee_Entf = <=100: ja (4.0)
Schnee_Entf = >100:
|   Wochenende = nein: nein (3.0)
|   Wochenende = ja:
|   |   Sonne = nein: nein (1.0)
|   |   Sonne = ja: ja (3.0/1.0)
```

```
Simplified Decision Tree:
|
Schnee_Entf = <=100: ja (4.0/1.2)
Schnee_Entf = >100: nein (7.0/3.4)
|
Evaluation on training data (11 items):
|
         Before Pruning           After Pruning
        ----------------   ---------------------------
        Size      Errors   Size      Errors   Estimate
|
          7    1( 9.1%)       3    2(18.2%)    (41.7%)
```

Der Lernalgorithmus

ErzeugeEntscheidungsbaum(*Daten*,*Knoten*)

A_{max} = Attribut mit maximalem Informationsgewinn
If InfGewinn(A_{max}) = 0
  Then *Knoten* wird Blattknoten mit häufigster Klasse in *Daten*
  Else ordne *Knoten* das Attribut A_{max} zu
    Erzeuge für jeden Wert $a_1, \ldots, a_n$ von A_{max}
      einen Nachfolgeknoten: $K_1, \ldots, K_n$
    Teile *Daten* auf in $D_1, \ldots, D_n$ mit $D_i = \{x \in Daten | A_{max}(x) = a_i\}$
    For all $i \in \{1, \ldots, n\}$
      If alle $x \in D_i$ gehören zur gleichen Klasse C_i
        Then Erzeuge Blattknoten K_i mit Klasse C_i
        Else ErzeugeEntscheidungsbaum(D_i, K_i)

Lernen von Appendizitisdiagnose (LEXMED)

app.names:

```
|Definition der Klassen und Attribute
|
|Klassen    0=Appendizitis negativ
|           1=Appendizitis positiv
0,1.
|
|Attribute
|
Alter:                                              continuous.
Geschlecht_(1=m___2=w):                             1,2.
Schmerz_Quadrant1_(0=nein__1=ja):                   0,1.
Schmerz_Quadrant2_(0=nein__1=ja):                   0,1.
Schmerz_Quadrant3_(0=nein__1=ja):                   0,1.
Schmerz_Quadrant4_(0=nein__1=ja):                   0,1.
Lokale_Abwehrspannung_(0=nein__1=ja):               0,1.
Generalisierte_Abwehrspannung_(0=nein__1=ja):       0,1.
Schmerz_bei_Loslassmanoever_(0=nein__1=ja):         0,1.
Erschuetterung_(0=nein__1=ja):                      0,1.
Schmerz_bei_rektaler_Untersuchung_(0=nein__1=ja):   0,1.
Temp_ax:                                            continuous.
Temp_re:                                            continuous.
Leukozyten:                                         continuous.
Diabetes_mellitus_(0=nein__1=ja):                   0,1
```

app.data:

```
19,1,0,0,1,0,1,0,1,1,0,362,378,13400,0,1
13,1,0,0,1,0,1,0,1,1,1,383,385,18100,0,1
32,2,0,0,1,0,1,0,1,1,0,364,374,11800,0,1
18,2,0,0,1,1,0,0,0,0,0,362,370,09300,0,0
73,2,1,0,1,1,1,0,1,1,1,376,380,13600,1,1
30,1,1,1,1,1,0,1,1,1,1,377,387,21100,0,1
56,1,1,1,1,1,0,1,1,1,0,390,?,14100,0,1
36,1,0,0,1,0,1,0,1,1,0,372,382,11300,0,1
36,2,0,0,1,0,0,0,1,1,1,370,379,15300,0,1
33,1,0,0,1,0,1,0,1,1,0,367,376,17400,0,1
19,1,0,0,1,0,0,0,1,1,0,361,375,17600,0,1
12,1,0,0,1,0,1,0,1,1,0,364,370,12900,0,0
...
```

Die Datei enthält 9764 Fälle.

```
unixprompt> c4.5 -f app -u -m 100
|
C4.5 [release 8] decision tree generator        Wed Aug 23 13:13:15
----------------------------------------
|
    Options:
        File stem <app>
        Trees evaluated on unseen cases
        Sensible test requires 2 branches with >=100 cases
|
Read 9764 cases (15 attributes) from app.data
|
Decision Tree:
|
Leukozyten <= 11030 :
|   Schmerz_bei_Loslassmanoever = 0:
|   |   Temp_re > 381 : 1 (135.9/54.2)
|   |   Temp_re <= 381 :
|   |   |   Lokale_Abwehrspannung = 0: 0 (1453.3/358.9)
|   |   |   Lokale_Abwehrspannung = 1:
|   |   |   |   Geschlecht_(1=m___2=w) = 1: 1 (160.1/74.9)
|   |   |   |   Geschlecht_(1=m___2=w) = 2: 0 (286.3/97.6)
|   Schmerz_bei_Loslassmanoever = 1:
|   |   Leukozyten <= 8600 :
|   |   |   Temp_re > 378 : 1 (176.0/59.4)
|   |   |   Temp_re <= 378 :
|   |   |   |   Geschlecht_(1=m___2=w) = 1:
|   |   |   |   |   Lokale_Abwehrspannung = 0: 0 (110.7/51.7)
```

Beschnittener Baum

```
Leukozyten > 11030 : 1 (5767.0/964.1)
Leukozyten <= 11030 :
|   Schmerz_bei_Loslassmanoever = 0:
|   |   Temp_re > 381 : 1 (135.9/58.7)
|   |   Temp_re <= 381 :
|   |   |   Lokale_Abwehrspannung = 0: 0 (1453.3/370.9)
|   |   |   Lokale_Abwehrspannung = 1:
|   |   |   |   Geschlecht_(1=m___2=w) = 1: 1 (160.1/79.7)
|   |   |   |   Geschlecht_(1=m___2=w) = 2: 0 (286.3/103.7)
|   Schmerz_bei_Loslassmanoever = 1:
|   |   Leukozyten > 8600 : 1 (984.7/322.6)
|   |   Leukozyten <= 8600 :
|   |   |   Temp_re > 378 : 1 (176.0/64.3)
|   |   |   Temp_re <= 378 :
|   |   |   |   Geschlecht_(1=m___2=w) = 1:
|   |   |   |   |   Lokale_Abwehrspannung = 0: 0 (110.7/55.8)
|   |   |   |   |   Lokale_Abwehrspannung = 1: 1 (160.6/73.4)
|   |   |   |   Geschlecht_(1=m___2=w) = 2:
|   |   |   |   |   Alter <= 14 : 1 (131.1/67.6)
|   |   |   |   |   Alter > 14 : 0 (398.3/144.7)
```

```
Evaluation on training data (9764 items):
|
         Before Pruning           After Pruning
        ----------------   --------------------------
        Size      Errors   Size      Errors   Estimate
          37  2197(22.5%)     21  2223(22.8%)    (23.6%)   <<
|
|
|
         Before Pruning           After Pruning
        ----------------   --------------------------
        Size      Errors   Size      Errors   Estimate
          37  1148(23.5%)     21  1153(23.6%)    (23.6%)   <<
|
          (a)  (b)      <-classified as
         ----  ----
          758   885      (a): class 0
          268  2971      (b): class 1
```

Stetige Attribute

Knoten: `Leukozyten > 11030`
Die Schwelle $\Theta_{D,A}$ für ein Attribut A wird bestimmt durch

$$\Theta_{D,A} = \max_{v}\{\text{InfGewinn}(D, A > v)\}.$$

für alle in den Trainingsdaten D vorkommenden Werte v von A.
Jedes stetige Attribut wird an jedem neu erzeugten Knoten wieder getestet!

Pruning – Der Baumschnitt

Aristoteles:

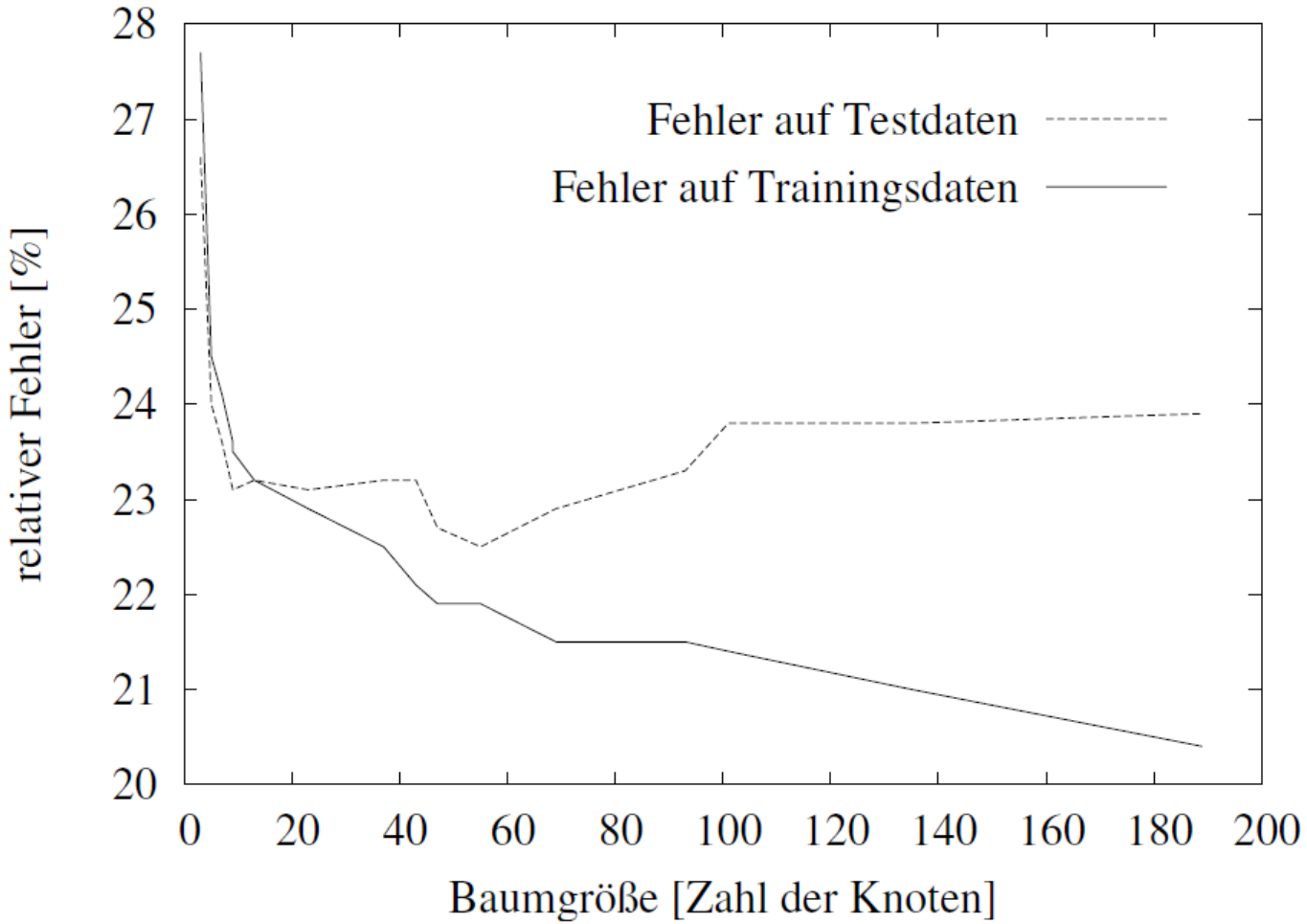
von zwei wissenschaftlichen Theorien, die den gleichen Sachverhalt gleich gut erklären, ist die einfachere zu bevorzugen.

Occam's Rasiermesser (engl. Occam's razor)
Unter allen Bäumen mit einer festen Fehlerrate ist also immer ein kleinster Baum auszuwählen.

Generalisieren

- Je komplexer ein Modell, desto mehr Details werden repräsentiert, aber umso schlechter läßt sich das Modell auf neue Daten übertragen. Daher:
- Trainingsdaten (z.B. 2/3 aller Daten)
- Testdaten (z.B. 1/3 aller Daten)

Überanpassung (engl. overfitting)



Überanpassung

Definition

> Sei ein bestimmtes Lernverfahren, das heißt eine Klasse lernender Agenten, gegeben. Man nennt einen Agenten A überangepasst an die Trainingsdaten, wenn es einen anderen Agenten A' gibt, dessen Fehler auf den Trainingsdaten größer ist als der von von A, aber auf der gesamten Verteilung von Daten ist der Fehler von A' kleiner als der von A.

Kreuzvalidierung (engl. cross validation):
Fehler auf den Testdaten messen bis er signifikant ansteigt!

Pruning bei C4.5

Reduced Error Pruning schneidet solange Knoten des Baumes ab, bis ein aus dem Fehler auf den Trainingsdaten geschätzter Fehler auf den Testdaten wieder zunimmt.[18]

Greedy-Verfahren:

- nicht optimal,
- Bias hin zu kleinen Bäumen mit Attributen mit hohem Gain ganz oben.
- Gain ratio bei vielen Attributwerten.
- Stärken: sehr schnell, bei den Lexmed-Daten (ca. 10000 Trainingsdaten mit 15 Attributen) ca. 0.18 sec

[18] Besser wäre es allerdings, beim Pruning den Fehler auf den Testdaten zu verwenden.

Fehlende Werte

$$56, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 390, ?, 14100, 0, 1,$$

Möglichkeiten beim Aufbau des Baumes: verwende

- häufigsten Wert unter allen anderen Trainingsdaten
- häufigsten Wert unter allen anderen Trainingsdaten in der gleichen Klasse
- Wahrscheinlichkeitsverteilung aller Attributwerte einsetzen und das fehlerhafte Trainingsbeispiel entsprechend dieser Verteilung aufteilen auf die verschiedenen Äste im Baum [19]

Bei der Klassifikation werden fehlende Werte gleich behandelt wie beim Lernen.

[19] Grund für das Vorkommen von nicht ganzzahligen Werten in den Klammerausdrücken hinter den Blattknoten der C4.5-Bäume.

Zusammenfassung

- Für Klassifikationsaufgaben sehr beliebtes Verfahren
- Einfache Anwendung
- hohe Geschwindigkeit: Auf 10 000 Daten mit 15 Attributen benötigt C4.5 etwa 0.3 Sekunden für das Lernen
- Irrelevante Attribute werden automatisch gelöscht
- Anwender kann den Entscheidungsbaum verstehen
- Anwender kann den Entscheidungsbaum verändern
- Entscheidungsbaum kann leicht in bestehende Programme eingebaut werden

Kreuzvalidierung und Überanpassung

- Problem der Überanpassung.
- Komplexität des gelernten Modells passt sich Komplexität der Trainingsdaten!
- Resultat: Überanpassung

Modellkomplexität:

Entscheidungsbaumlernen: Baumgröße

k-Nearest-Neighbour-Methode: Zahl k der nächsten Nachbarn

Neuronale Netze: Zahl der verdeckten Schichten / Neuronen

k-fache Kreuzvalidierung

Kreuzvalidierung($\mathbf{X}, k$)

Daten zufällig aufteilen in k gleich große Blöcke $\mathbf{X} = \mathbf{X}_1 \cup \ldots \cup \mathbf{X}_k$
For all $\gamma \in \{\gamma_{min}, \ldots, \gamma_{max}\}$
    For all $i \in \{1, \ldots, k\}$
        Trainiere ein Modell der Komplexität γ auf $\mathbf{X} \backslash \mathbf{X}_i$
        Berechne den Fehler $E(\gamma, \mathbf{X}_i)$ auf der Testdatenmenge $\mathbf{X}_i$
    Berechne den mittleren Fehler $E(\gamma) = \frac{1}{k} \sum_{i=1}^{k} E(\gamma, \mathbf{X}_i)$
Wähle den Wert $\gamma_{opt} = \text{argmin}_\gamma E(\gamma)$ mit kleinstem mittlerem Fehler
Trainiere finales Modell mit γ_{opt} auf gesamter Datenmenge $\mathbf{X}$

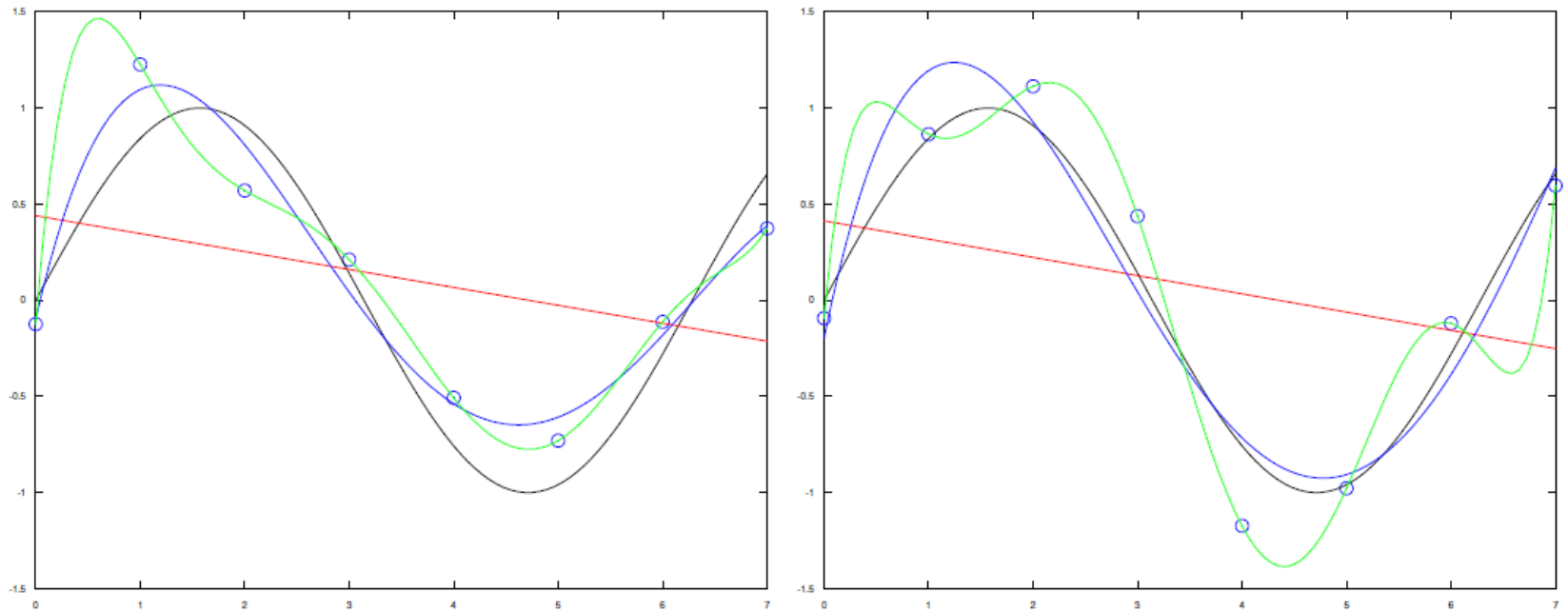
Bias-Variance-Tradeoff

- Zu einfaches Modell: nicht optimale Approximation (Bias)
- Zu komplexes Modell: Das Modell variiert sehr bei veränderten Daten (Variance).

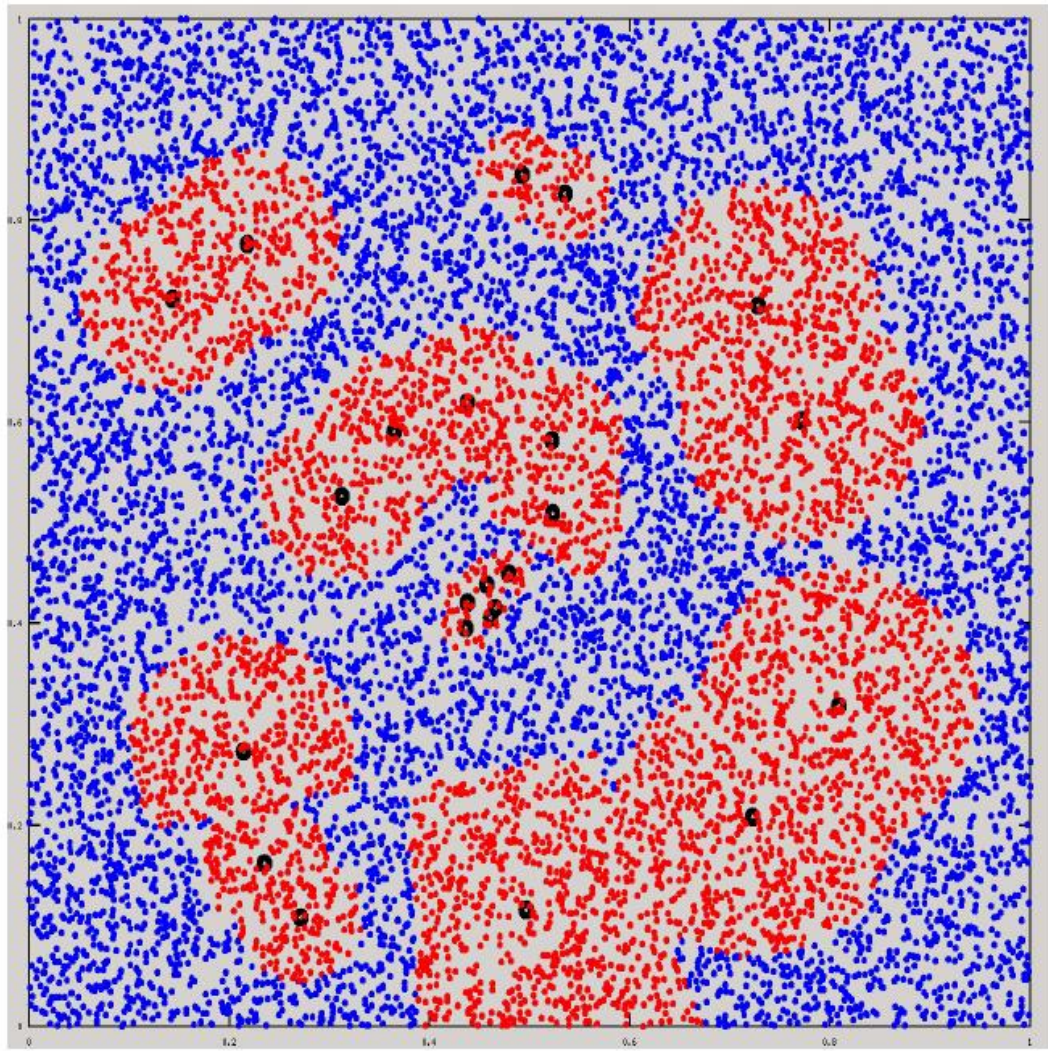
Beispiel

Funktionsapproximation mit Polynomen

acht Datenpunkte nach $y(x) = \sin(x) + r(0, 0.2)$ *mit normalverteiltem Rauschen erzeugt:*



Nearest Neighbour Data Description



$\gamma = 1$

Nearest Neighbour Data Description

Gegeben:

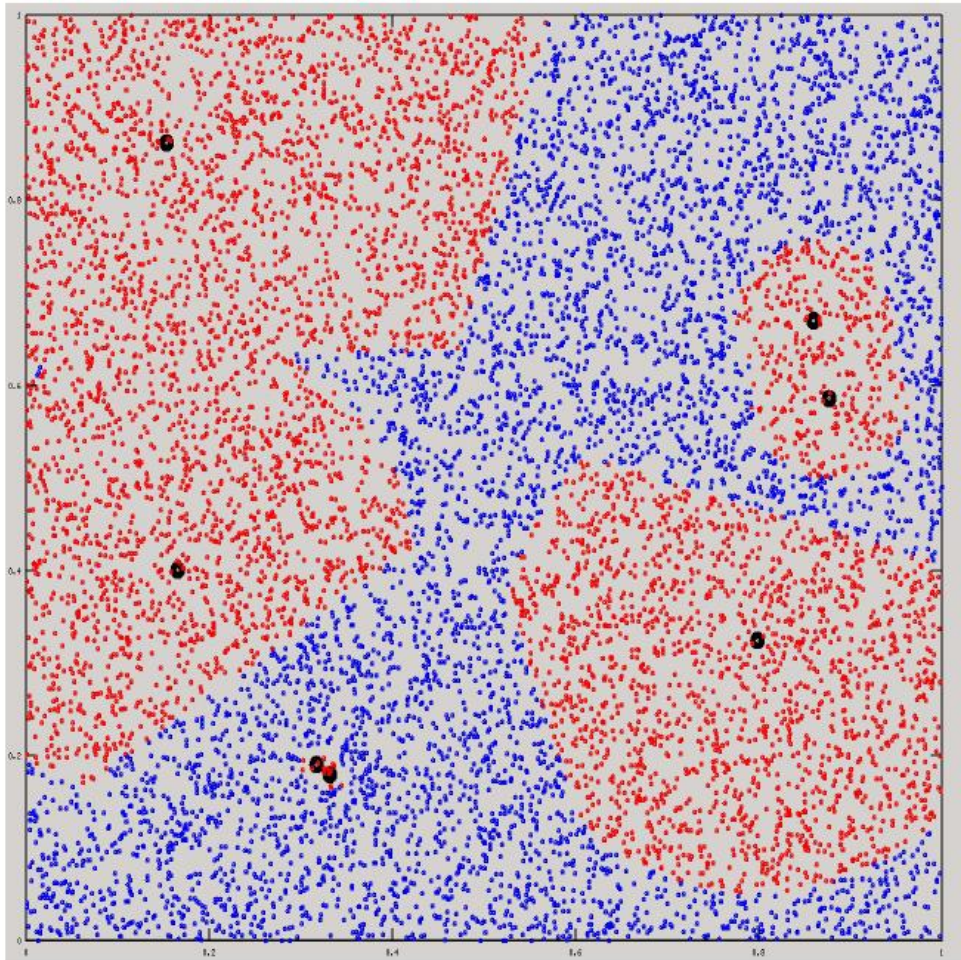
Datenmenge $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$

NNDD-Formel:

Ein neuer Punkt $\mathbf{q}$ wird akzeptiert, wenn

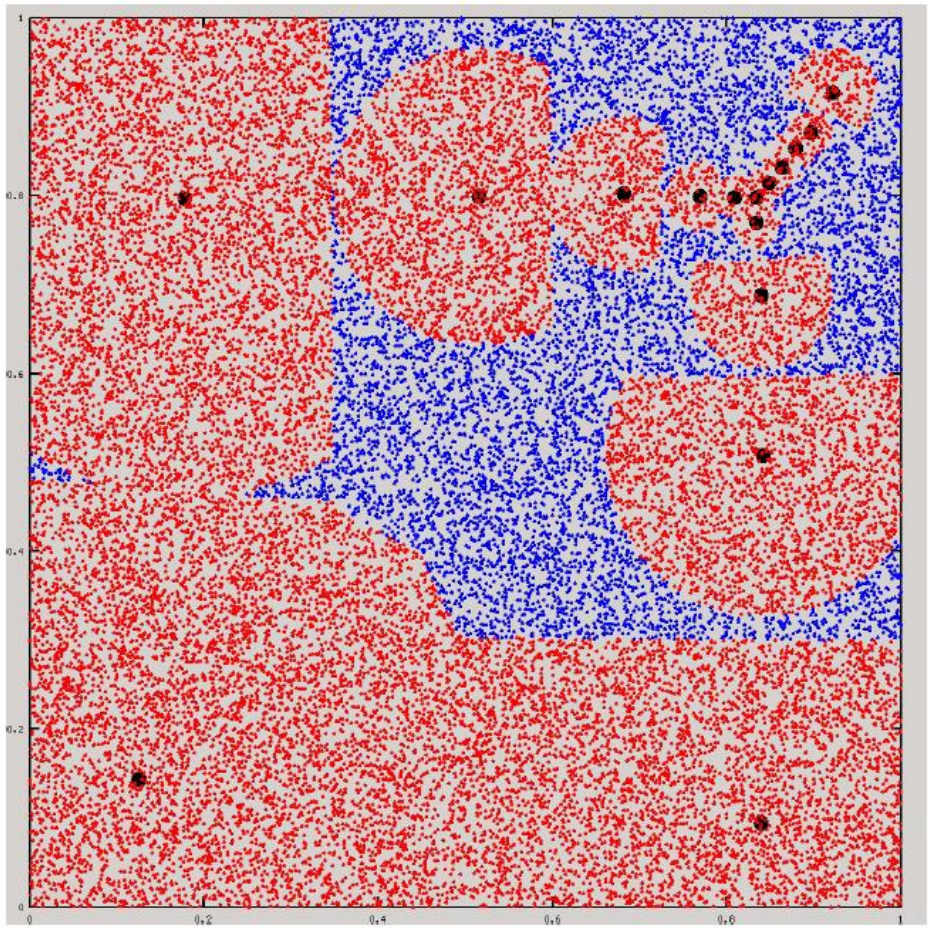
$$D(\mathbf{q}, NN(\mathbf{q})) \leq \gamma D(NN(\mathbf{q}), NN(NN(\mathbf{q})))$$

Nearest Neighbour Data Description



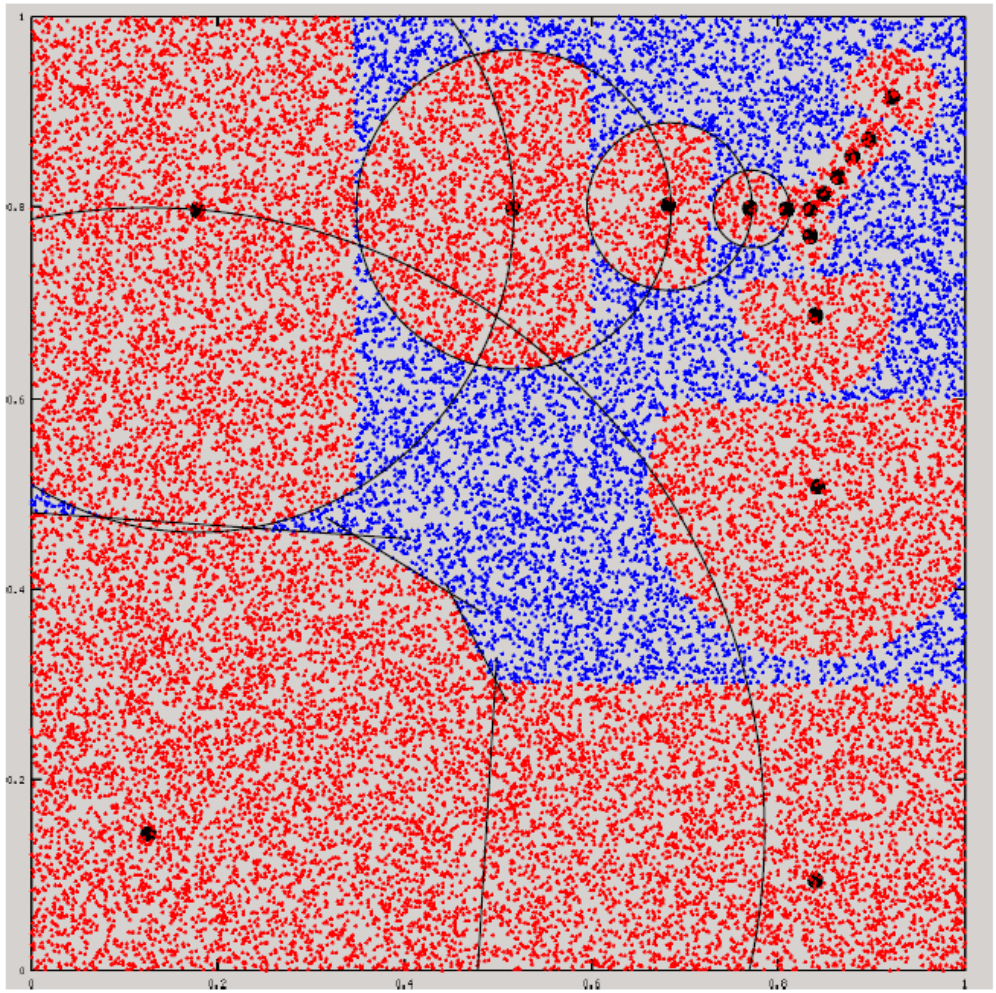
$\gamma = 1$

Nearest Neighbour Data Description



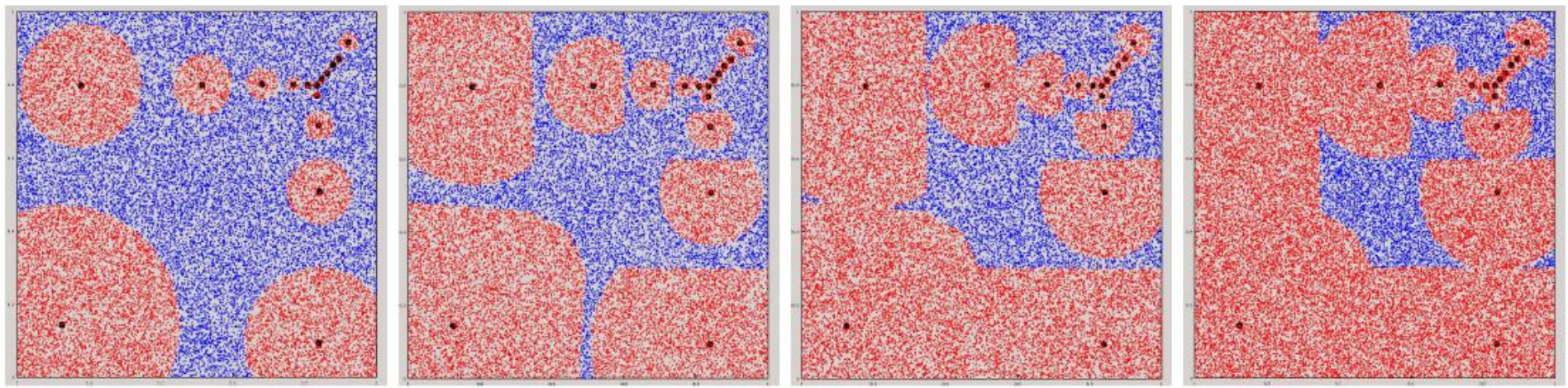
$\gamma = 1$

Nearest Neighbour Data Description



$\gamma = 1$

Nearest Neighbour Data Description



$\gamma = 0.5$ $\gamma = 0.8$ $\gamma = 1.0$ $\gamma = 1.2$

k – Nearest Neighbour Data Description

Gegeben:

Datenmenge $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$

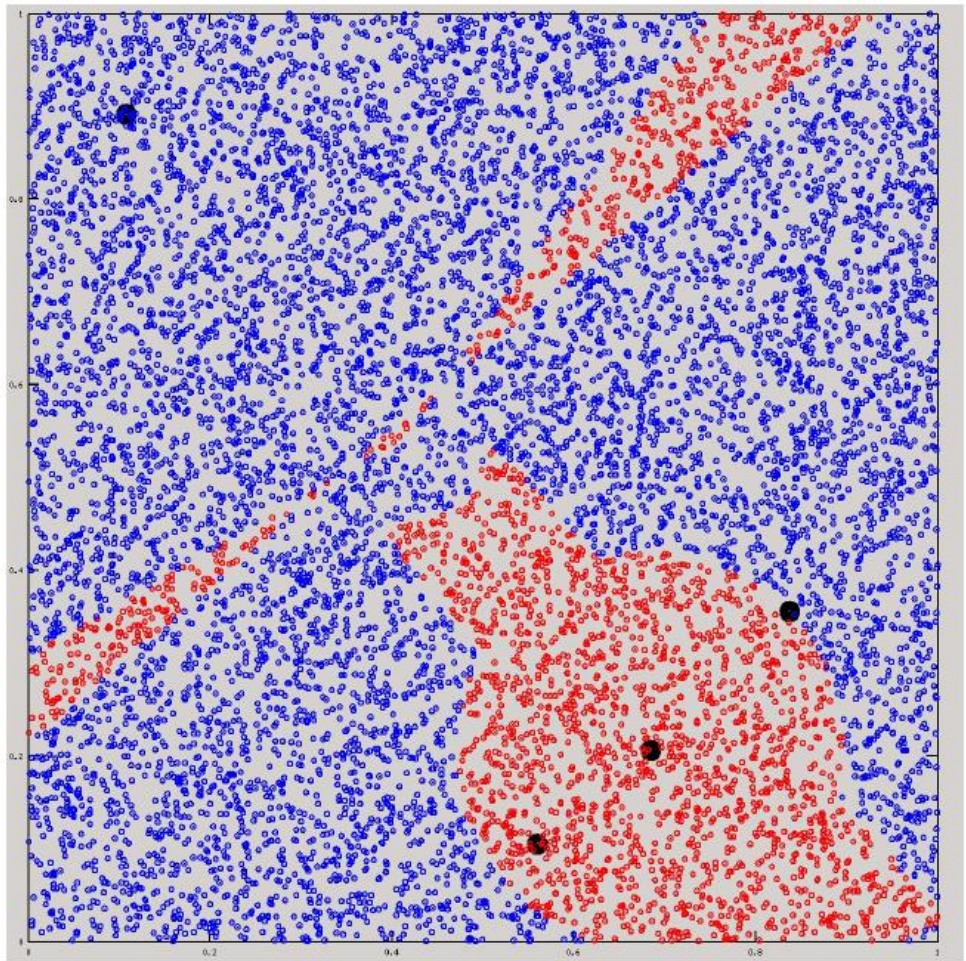
neue Formel:

Ein neuer Punkt $\mathbf{q}$ wird akzeptiert, wenn

$$D(\mathbf{q}, kNN(\mathbf{q})) \leq \gamma D(kNN(\mathbf{q}), kNN(kNN(\mathbf{q})))$$

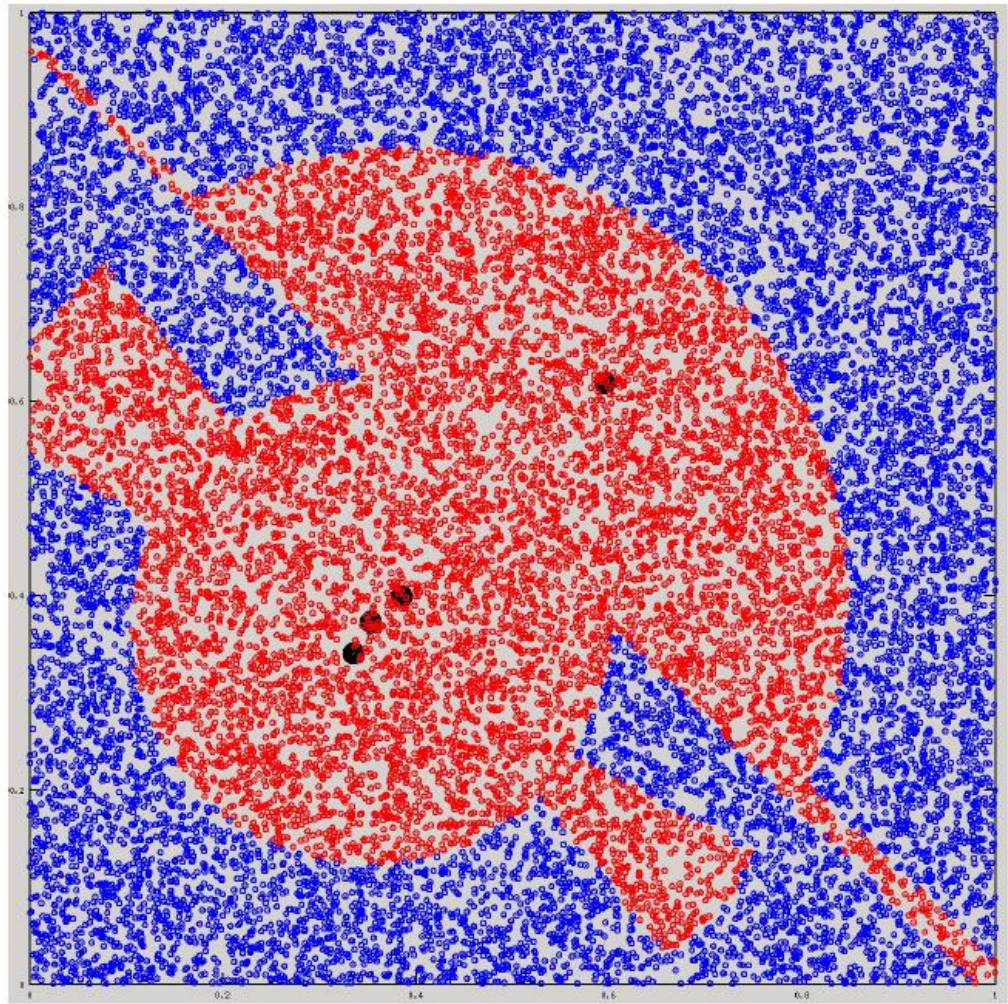
$kNN(\mathbf{q})$ ist der k-te nächste Nachbar zu $\mathbf{q}$.

k – Nearest Neighbour Data Description



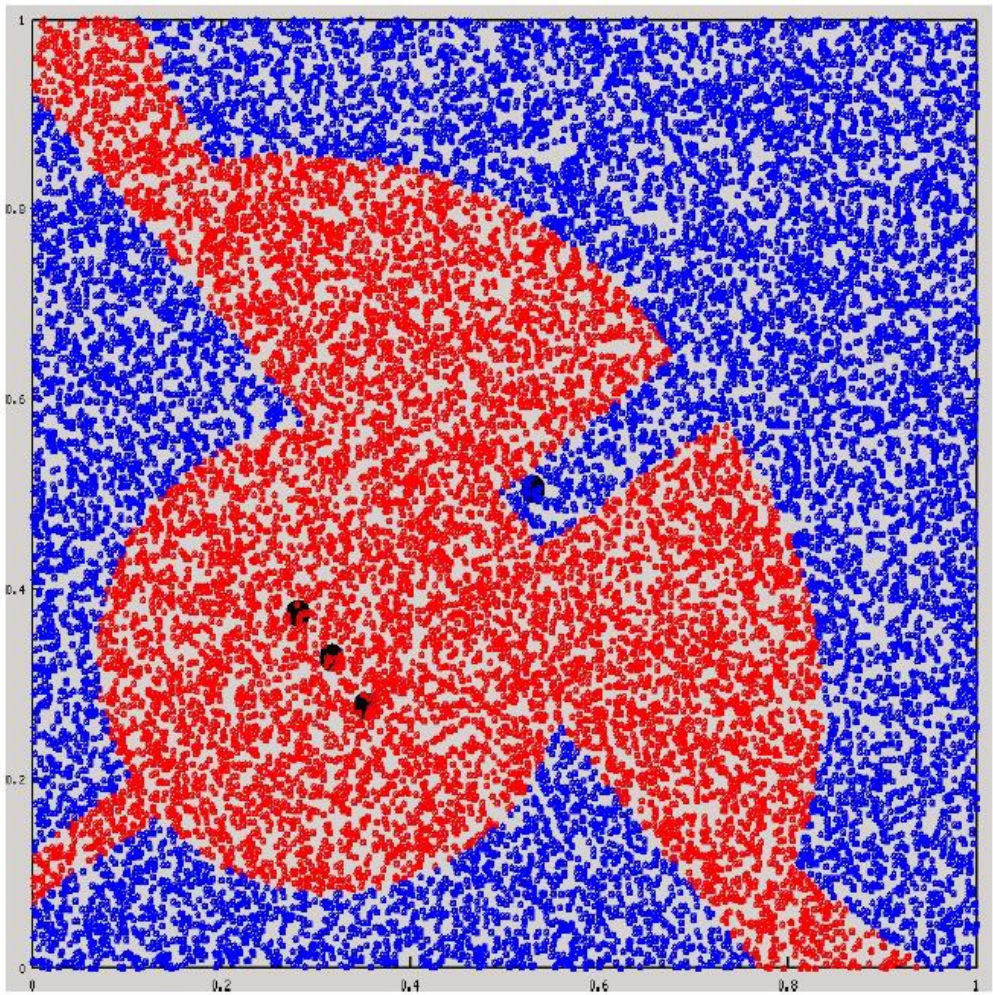
$k = 2,\ \gamma = 1$

k – Nearest Neighbour Data Description



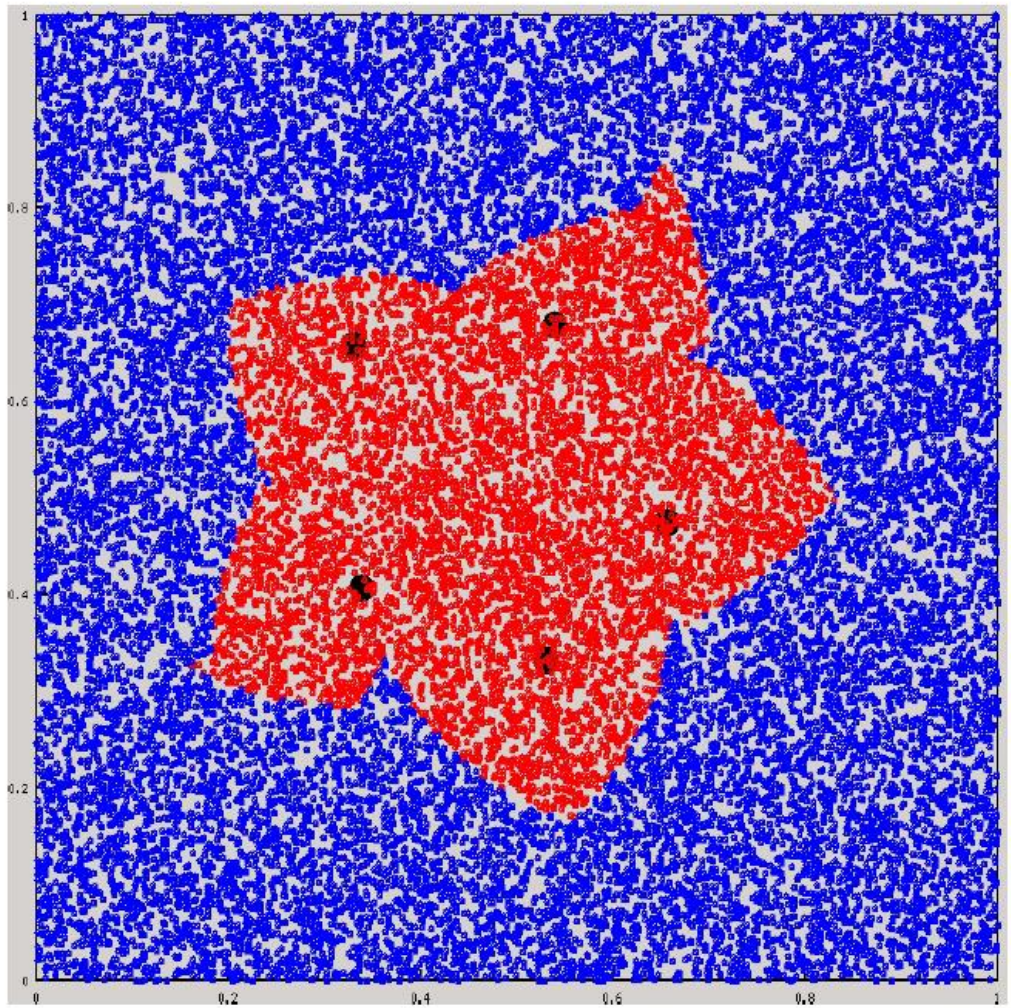
$k = 2,\ \gamma = 6$

k – Nearest Neighbour Data Description



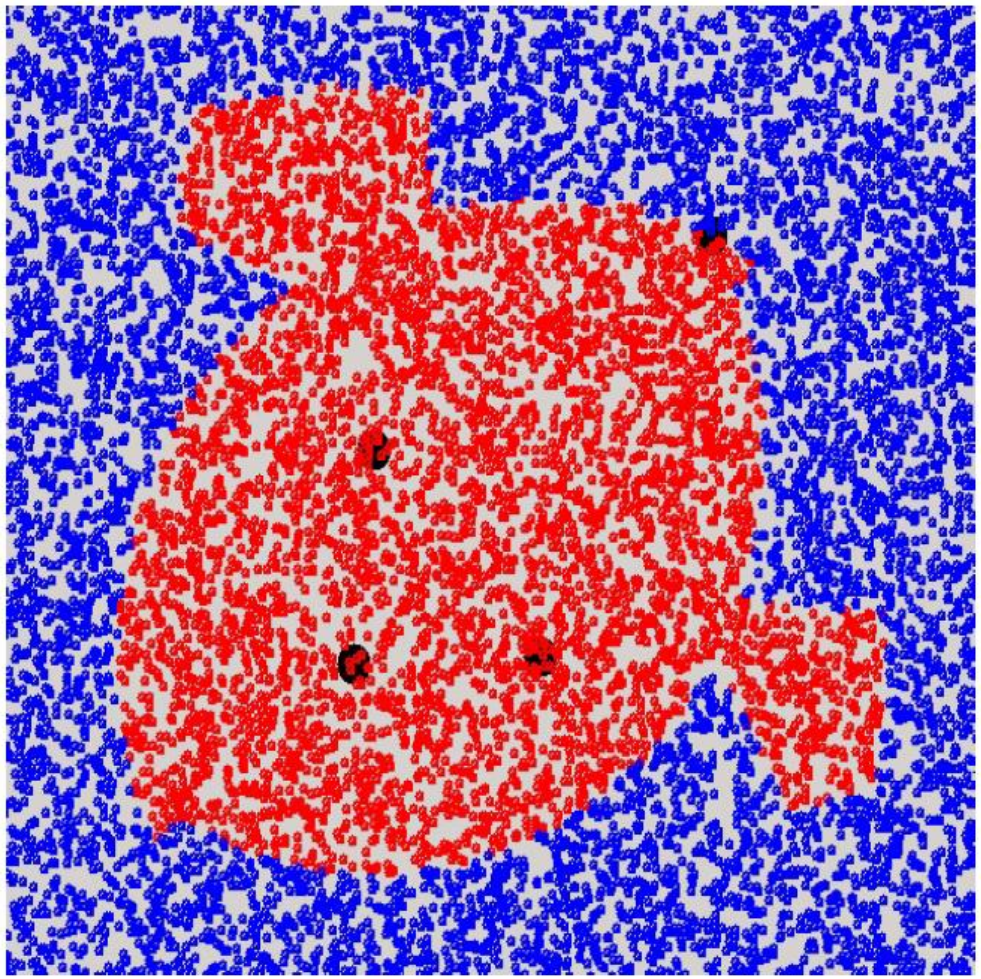
$k = 2,\ \gamma = 4$

k – Nearest Neighbour Data Description



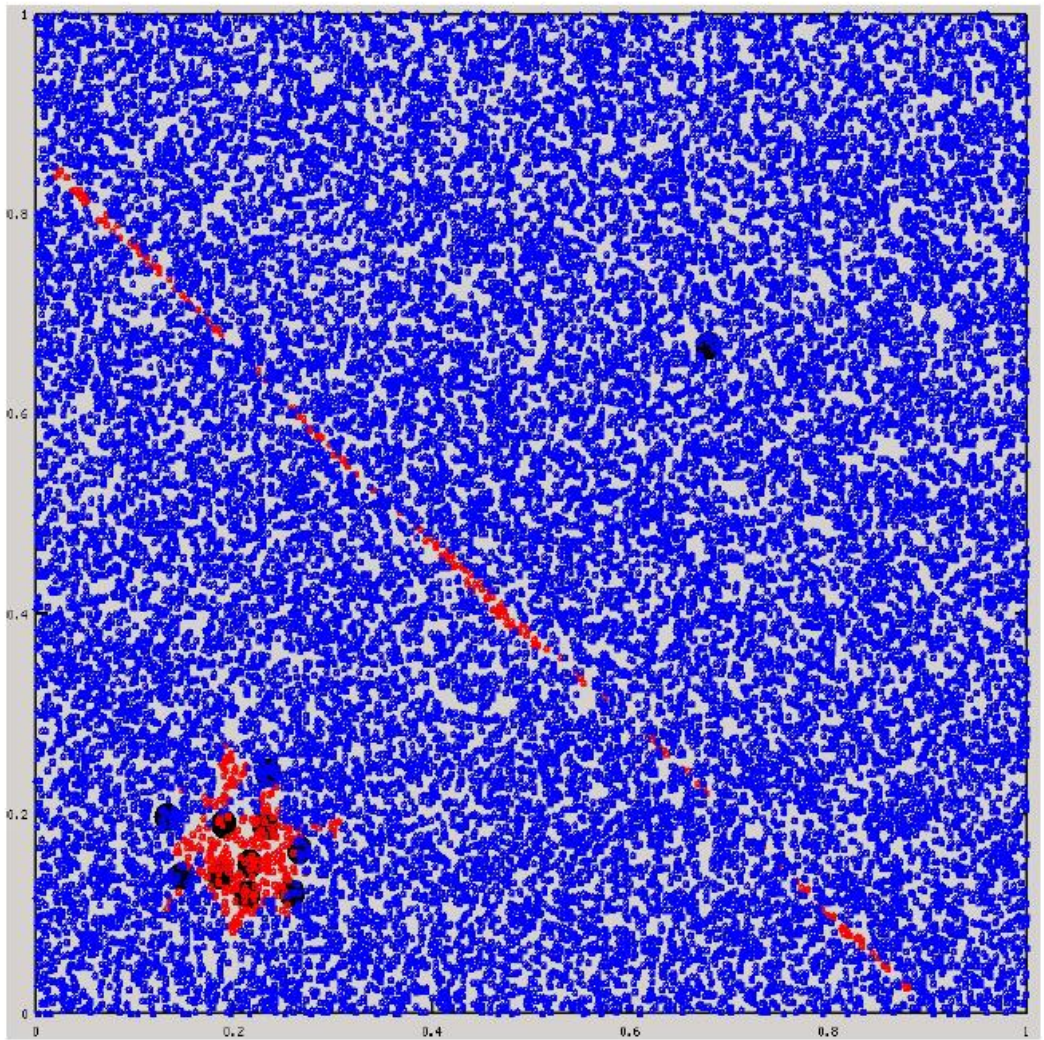
$k = 3,\ \gamma = 1$

k – Nearest Neighbour Data Description



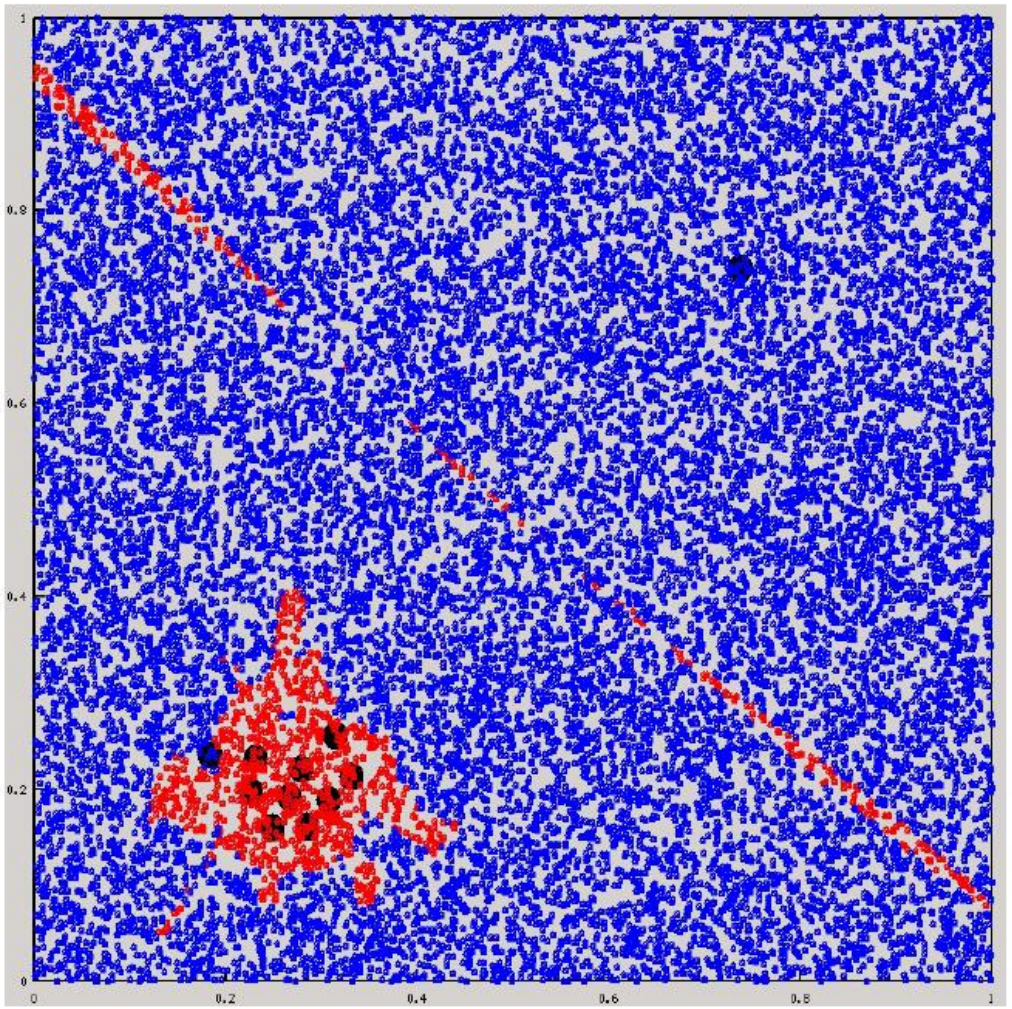
$k = 3,\ \gamma = 1$

k – Nearest Neighbour Data Description



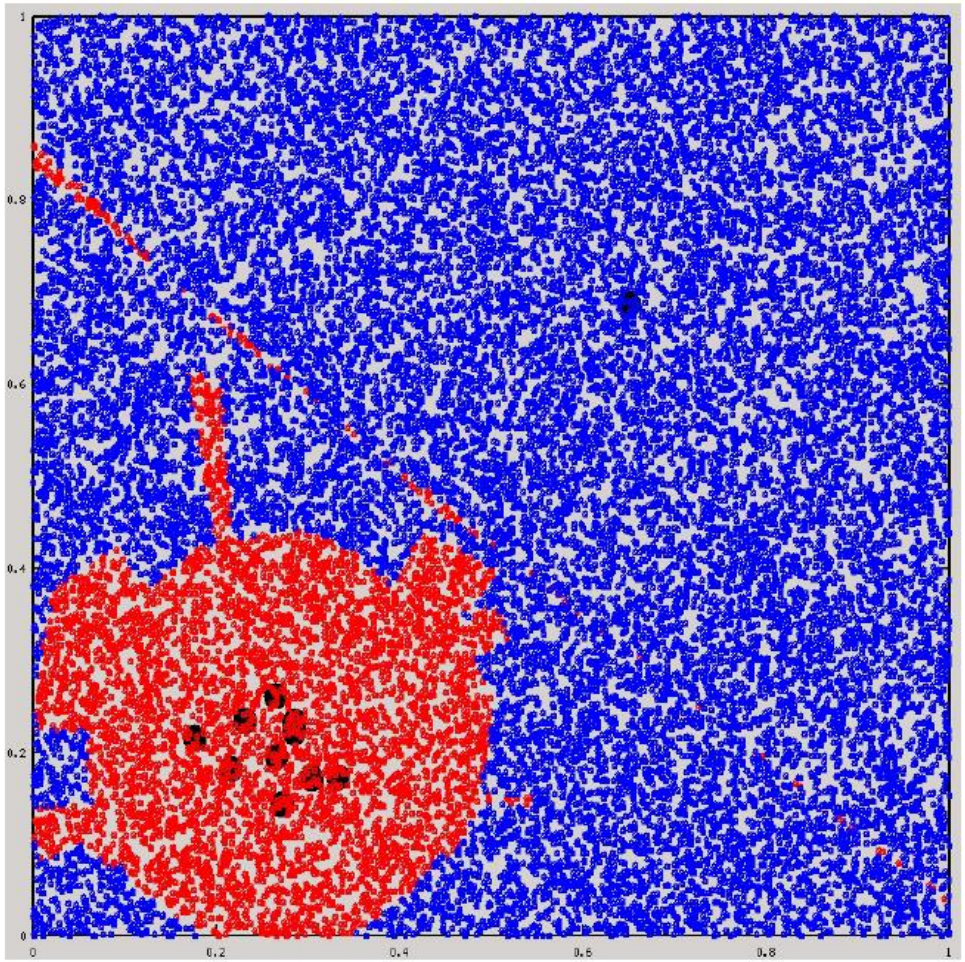
$k = 4,\ \gamma = 1$

k – Nearest Neighbour Data Description



$k = 4,\ \gamma = 2$

k – Nearest Neighbour Data Description



$k = 4,\ \gamma = 4$

Nearest Neighbour Data Description (Var. Ertel)

Gegeben:

Datenmenge $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$

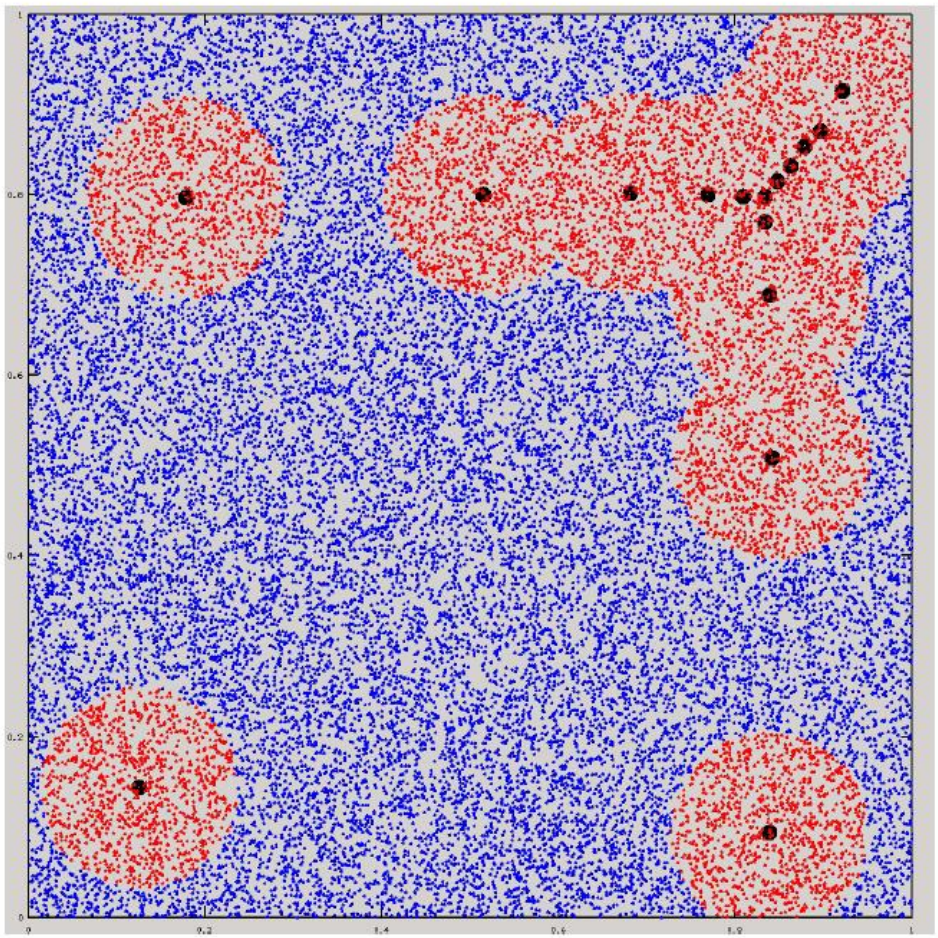
neue Formel:

Ein neuer Punkt $\mathbf{q}$ wird akzeptiert, wenn

$$D(\mathbf{q}, NN(\mathbf{q})) \leq \gamma \bar{D}$$

$\bar{D}$ = mittlerer Abstand der nächsten Nachbarn zu allen Datenpunkten in $\mathbf{X}$.

Nearest Neighbour Data Description (Var. Ertel)



$\gamma = 0.5, \quad \bar{D} = 0.223$

Naive Bayes Classification

- **Naive Bayes** is a simple, yet effective and commonly-used, machine learning classifier.
- It is a probabilistic classifier that makes classifications using the Maximum A Posteriori decision rule in a Bayesian setting.
- It can also be represented using a very simple Bayesian network.
- Naive Bayes classifiers have been especially popular for text classification, and are a traditional solution for problems such as spam detection.

The Model:

- The goal of any probabilistic classifier with features x_0 through x_n and classes c_0 through c_k is to determine the probability of the features occurring in each class, and to return the most likely class.
- Therefore, for each class, we want to be able to calculate $P(c_i \mid x_0, \ldots, x_n)$.
- In order to do this, we use **Bayes rule**:

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

Naive Bayes Classification

- In the context of classification, you can replace A with a class, c_i, and B with the set of features, x_0 through x_n.
- Since P(B) serves as normalization, and we are usually unable to calculate $P(x_0, \ldots, x_n)$, we can simply ignore that term, and instead just state that $P(c_i \mid x_0, \ldots, x_n) \propto P(x_0, \ldots, x_n \mid c_i) * P(c_i)$, where $\propto$ means "is proportional to".
- $P(c_i)$ is simple to calculate; it is just the proportion of the data-set that falls in class i.
- $P(x_0, \ldots, x_n \mid c_i)$ is more difficult to compute. In order to simplify its computation, we make the assumption that x_0 through x_n are **conditionally independent** given c_i, which allows us to say that $P(x_0, \ldots, x_n \mid c_i) = P(x_0 \mid c_i) * P(x_1 \mid c_i) * \ldots * P(x_n \mid c_i)$.
- This assumption is most likely not true — hence the name *naive* Bayes classifier, but the classifier nonetheless performs well in most situations.

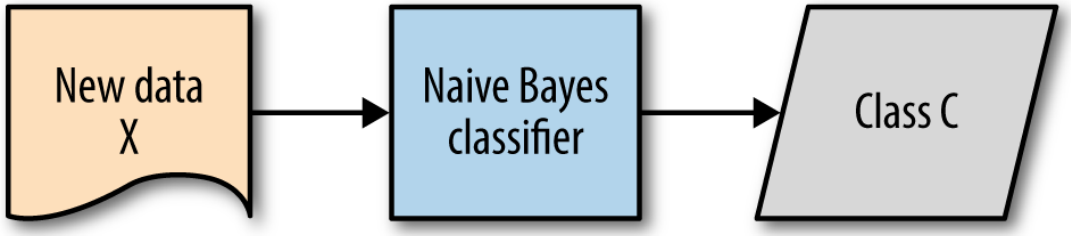
- Therefore, the final representation of class probability is the following:

$$P(c_i|x_0,\ldots,x_n) \propto P(x_0,\ldots,x_n|c_i)P(c_i)$$
$$\propto P(c_i)\prod_{j=1}^{n} P(x_j|c_i)$$

- Calculating the individual $P(x_j \mid c_i)$ terms will depend on what distribution your features follow. In the context of text classification, where features may be word counts, features may follow a **multinomial distribution**.
- In other cases, where features are continuous, they may follow a **Gaussian distribution**.

Naive Bayes Classification

- Note that there is very little explicit training in Naive Bayes compared to other common classification methods.
- The only work that must be done before prediction is finding the parameters for the features' individual probability distributions, which can typically be done quickly and deterministically.
- This means that Naive Bayes classifiers can perform well even with high-dimensional data points and/or a large number of data points.



Naive Bayes Classification

Classification:

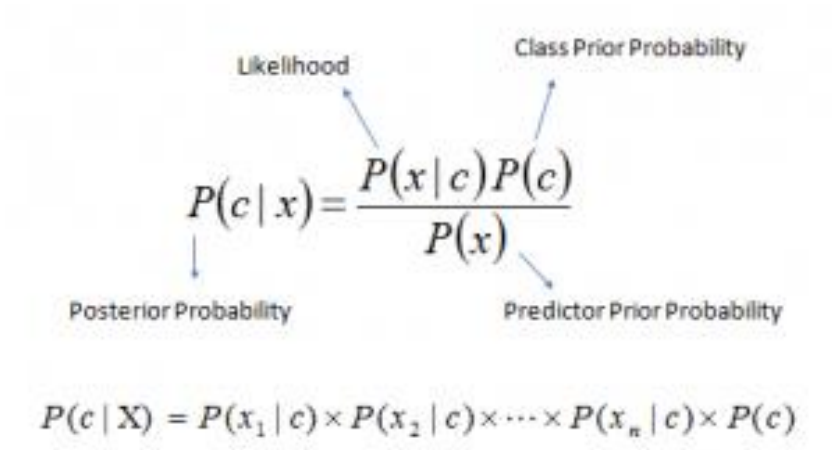
- Now that we have a way to estimate the probability of a given data point falling in a certain class, we need to be able to use this to produce classifications.
- Naive Bayes handles this in a very simple manner; simply pick the c_i that has the largest probability given the data point's features.

$$y = \underset{c_i}{argmax}\ P(c_i) \prod_{j=1}^{n} P(x_j|c_i)$$

- This is referred to as the **Maximum A Posteriori** decision rule.
- This is because, referring back to our formulation of Bayes rule, we only use the P(B|A) and P(A) terms, which are the likelihood and prior terms, respectively.
- If we only used P(B|A), the likelihood, we would be using a **Maximum Likelihood** decision rule.

Naive Bayes Classification: Example

- Keep in mind that: *Learning a Naive Bayes classifier is just a matter of counting how many times each attribute co-occurs with each class*



- $P(c|x)$ is the posterior probability of *class* c given *predictor* (*features*).
- $P(c)$ is the probability of *class*.
- $P(x|c)$ is the likelihood which is the probability of *predictor* given *class*.
- $P(x)$ is the prior probability of *predictor*.

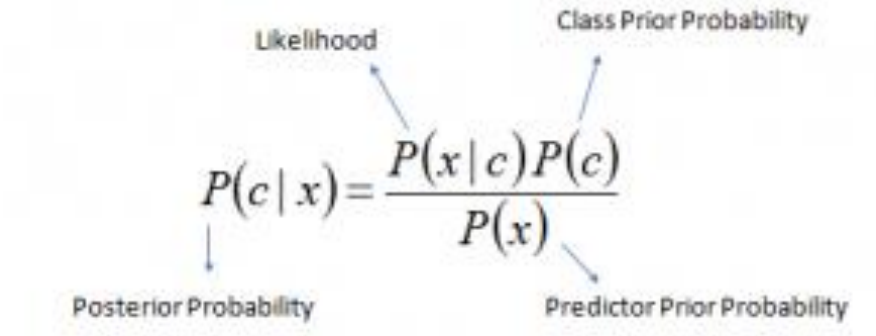
Naive Bayes Classification: Example

Fruit	Long	Sweet	Yellow	Total
Banana	400	350	450	500
Orange	0	150	300	300
Other	100	150	50	200
Total	**500**	**650**	**800**	**1000**

- 50% of the fruits are bananas
- 30% are oranges
- 20% are other fruits
- Based on our training set we can also say the following:
  - From 500 bananas 400 (0.8) are Long, 350 (0.7) are Sweet and 450 (0.9) are Yellow
  - Out of 300 oranges, 0 are Long, 150 (0.5) are Sweet and 300 (1) are Yellow
  - From the remaining 200 fruits, 100 (0.5) are Long, 150 (0.75) are Sweet and 50 (0.25) are Yellow

Naive Bayes Classification: Example

- Given the features of a piece of fruit and we need to predict the class.
- If we're told that the additional fruit is Long, Sweet and Yellow, we can classify it using the following formula and subbing in the values for each outcome, whether it's a Banana, an Orange or Other Fruit.
- The one with the highest probability (score) being the winner.



$$P(c|\mathbf{X}) = P(x_1|c) \times P(x_2|c) \times \cdots \times P(x_n|c) \times P(c)$$

Naive Bayes Classification: Example

Banana:

$$P\left(\frac{Banana}{Long, Sweet, Yellow}\right) = \frac{P\left(\frac{Long}{Banana}\right) \times P\left(\frac{Sweet}{Banana}\right) \times P\left(\frac{Yellow}{Banana}\right) \times P(Banana)}{P(Long)\, P(Sweet)\, P(Yellow)}$$

$$P\left(\frac{Banana}{Long, Sweet, Yellow}\right) = \frac{(0.8) \times (0.7) \times (0.9) \times (0.5)}{0.25 \times 0.33 \times 0.41}$$

$$P\left(\frac{Banana}{Long, Sweet, Yellow}\right) = 0.252$$

Naive Bayes Classification: Example

Orange:

$$P(\frac{Orange}{Long, Sweet, Yellow}) = 0$$

Naive Bayes Classification: Example

Other fruit:

$$P\left(\frac{Other}{Long, Sweet, Yellow}\right) = \frac{P\left(\frac{Long}{Other}\right) \times P\left(\frac{Sweet}{Other}\right) \times P\left(\frac{Yellow}{Other}\right) \times P(Other)}{P(Long)\, P(Sweet)\, P(Yellow)}$$

$$P\left(\frac{Other}{Long, Sweet, Yellow}\right) = \frac{(0.5) \times (0.75) \times (0.25) \times (0.2)}{0.25 \times 0.33 \times 0.41}$$

$$P\left(\frac{Other}{Long, Sweet, Yellow}\right) = 0.01875$$

Naive Bayes Classification: Example

- In this case, based on the higher score (0.252 for banana) we can assume this Long, Sweet and Yellow fruit is in fact, a Banana.