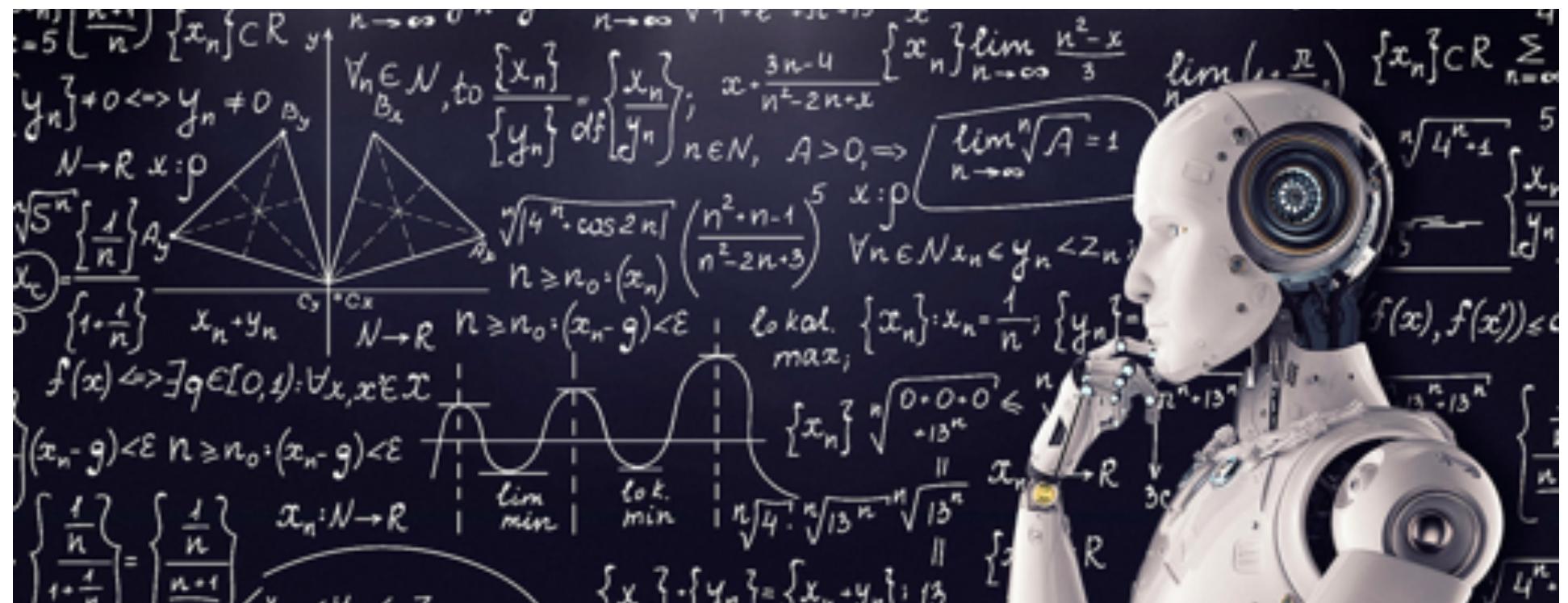


# Data Mining und Maschinelles Lernen



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

## K-nächste-Nachbarn, Überanpassung, Kreuzvalidierung und Datenanalyse-Zyklus



Basierende auf Folien von Katharina Morik, Uwe Ligges, Claus Weihs, Lutz Plümer und vielen anderen.  
Danke fürs Offenlegen ihrer Folien

# Ein erster Lernalgorithmus: k-nächste-Nachbarn



- Sie werden in der Vorlesung einige Verfahren kennenlernen. Diese können in globale und lokale Ansätze unterteilt werden
- Lineare Modelle und die Stützvektormethode (später in der Vorlesung) finden eine trennende Hyperebene. Die Ebene trennt alle Beispiele auf einmal. Lineare Modelle sind daher global.
- Klassifiziert man ein Beispiel nur anhand der Beispiele seiner Umgebung, spricht man von einem lokalen Modell. Nächste Nachbarn sind ein lokales Modell.



# K Nächste Nachbarn

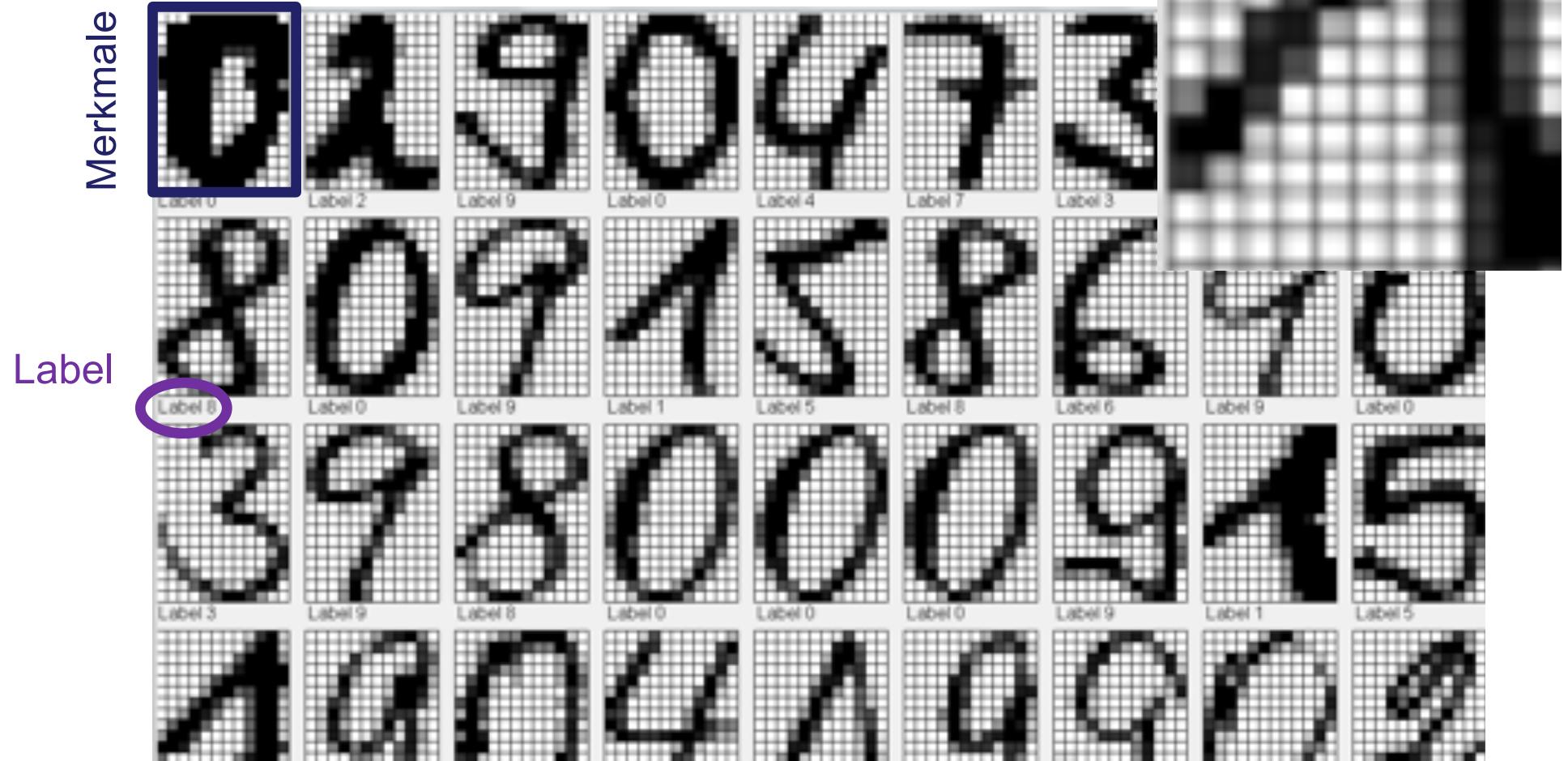
- Das  $k$ NN-Modell betrachtet nur die  $k$  nächsten Nachbarn eines Beispiel  $\vec{x}$ :

$$\hat{f}(\vec{x}) = \frac{1}{k} \sum_{\vec{x}_i \in N_k(\vec{x})} y_i \quad (1)$$

- Die Nachbarschaft  $N_k(\vec{x})$  wird durch ein Abstandsmaß, z.B. den Euklidschen Abstand bestimmt.
- Wenn sich die Nachbarschaften nicht überlappen, gibt es maximal  $\frac{N}{k}$  Nachbarschaften und in jeder bestimmen wir den Durchschnitt (1).

# K Nächste Nachbarn

Trainingsbeispiele





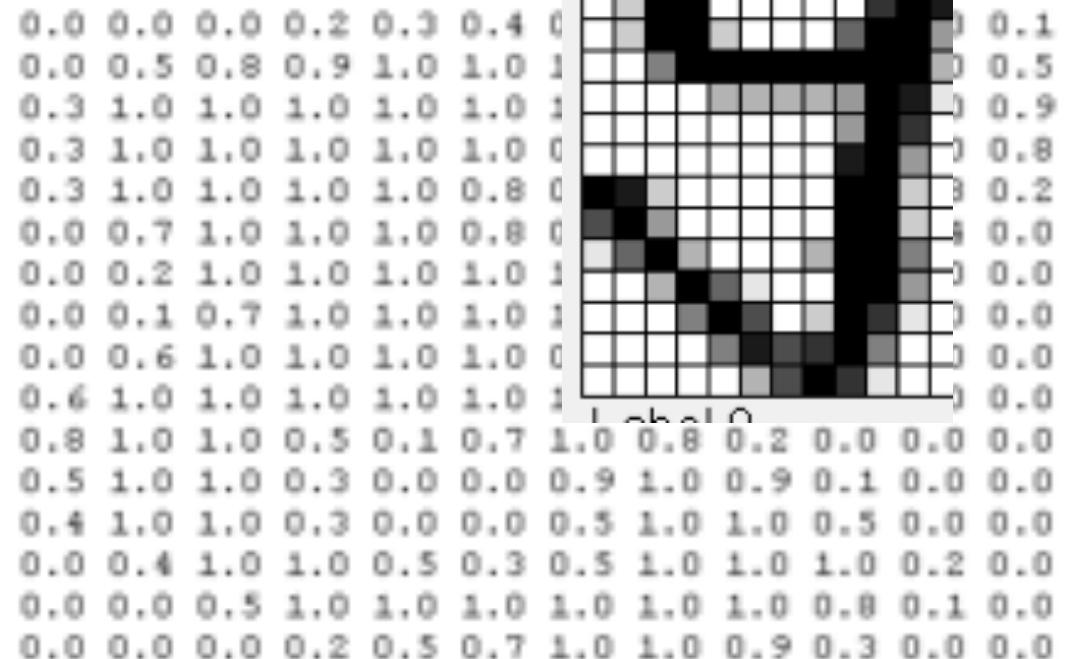
# Grundidee

Zwei Bilder repräsentieren die gleiche Ziffer, wenn die Bilder ähnlich sind  
**Ähnlich = ähnliche Grauwertverteilung**

Wir stellen Bilder als eine Matrix von Grauwerten dar

Ziffer =  $12 \times 16$  Matrix von  
Grauwerten in  $[0,1]$

Vektor von Grauwerten der  
Länge 192

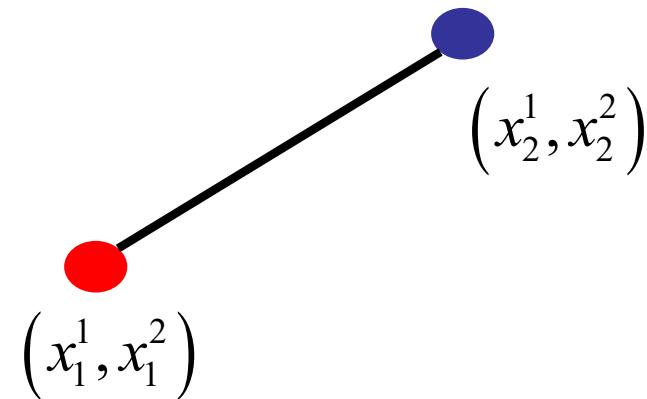


# Ähnlichkeit, Abstand

Es stellt sich also die Frage, wie ähnlich zwei Bilder/Datenpunkte sind  
Die Ähnlichkeit wird durch den Abstand beschrieben

- Je ähnlicher, desto kleiner der Abstand
- Sie sind identisch, wenn ihr Abstand 0 ist

Das wichtigste Abstandsmaß für reelle Merkmale/Vektoren ist der  
“euklidische Abstand”



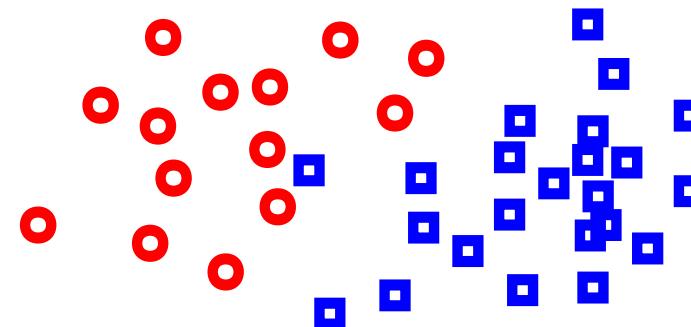
$$dist(\vec{x}, \vec{y}) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$$

$$\cos(\vec{x}, \vec{y}) = \frac{\vec{x} \cdot \vec{y}}{\|\vec{x}\| \|\vec{y}\|} = \frac{\sum_{i=1}^n x_i y_i}{\sqrt{\sum_{i=1}^n x_i^2} \sqrt{\sum_{i=1}^n y_i^2}}$$

# Handschriftenerkennung

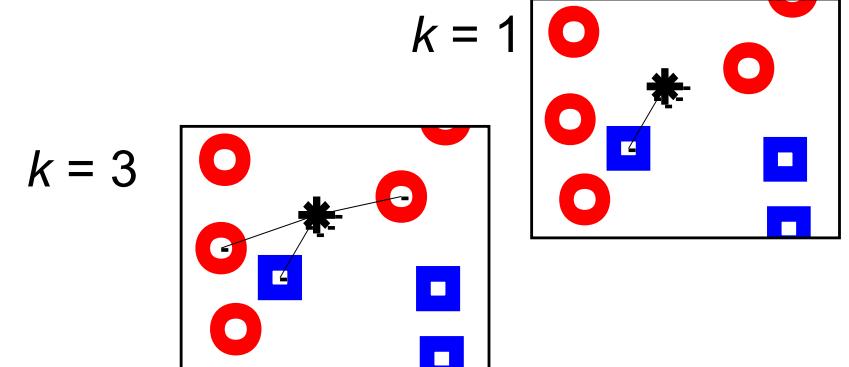
- (1) Um die Bedeutung des Bildes  $p$  zu finden, finde das Trainingsbild  $x$  mit  $\text{dist}(p,x)$  minimal (durch lineare Suche über alle Trainingsdaten)

- (2) Gib das Label von  $x$  aus



Erkennungsrate für  $k=1$ : 0.934

Mehrheitslabel für  $k=3$ : 0.945





# Allgemein: K nächste Nachbarn

Idee:

- Finde in den Trainingsdaten den/die zur aktuellen Instanz ähnliche(n) Instanz(en)
- Die aktuelle Instanz bekommt denselben Wert wie diese(r) Nachbar(n)

Schwierigkeiten:

- Welches Ähnlichkeitsmaß?
- Wieviele Nachbarn?
- Was, wenn die Werte der Nachbarn nicht übereinstimmen?
- Wie können wir die Suche effizient gestalten?



# Allgemein: K nächste Nachbarn

Idee:

- Finde in den Trainingsdaten den/die zur aktuellen Instanz ähnliche(n) Instanz(en)
- Die aktuelle Instanz bekommt denselben Wert wie diese(r) Nachbar(n)

Jede Instanz hat die Form  $\langle \mathbf{x}_i, f(\mathbf{x}_i) \rangle$

1. Berechne Abstand zwischen Testinstanz  $\mathbf{x}_i$  und jeder Trainingsinstanz
2. Wähle die k-nächsten Nachbarn  $\mathbf{n}_1, \dots, \mathbf{n}_k$  aus
3. Der Wert/das Label für  $\mathbf{x}$  ergibt sich durch:
  - $f(\mathbf{x}) = A(f(\mathbf{n}_1), \dots, f(\mathbf{n}_k))$
  - A ist eine Auswahlfunktion



## Ähnlichkeitsmaße für 0/1 Features

X seien die positiven Features von Instanz A und Y die positiven Features von Instanz B, die Ähnlichkeit von A und B ist dann:

Matching Koeffizient:  $|X \cap Y|$

Dice Koeffizient :  $2|X \cap Y|/(|X|+|Y|)$

Jaccard Koeffizient :  $|X \cap Y|/|X \cup Y|$

Overlap Koeffizient :  $|X \cap Y|/\min(|X|,|Y|)$

Kosinus:  $|X \cap Y|/(|X|\times|Y|)^{1/2}$



# Was ist das richtige Ähnlichkeitsmaß?

Im Allgemeinen ist das schwer zu sagen:

„We know it when we see it“

Die Bedeutung von „Ähnlichkeit“ scheint eher ein philosophisches Problem zu sein. Maschinelles Lernen benutzt den Begriff und seine Bedeutung eher pragmatisch





## Auswahlfunktion

Klassifikation: Mehrheitsentscheidung

Regression:

$$f(x) = \frac{\sum_{i=1}^k f(n_i)}{k}$$

Oder auch gewichtete Varianten:

$$f(x) = \sum_{i=1}^k sim(n_i, x) f(n_i)$$

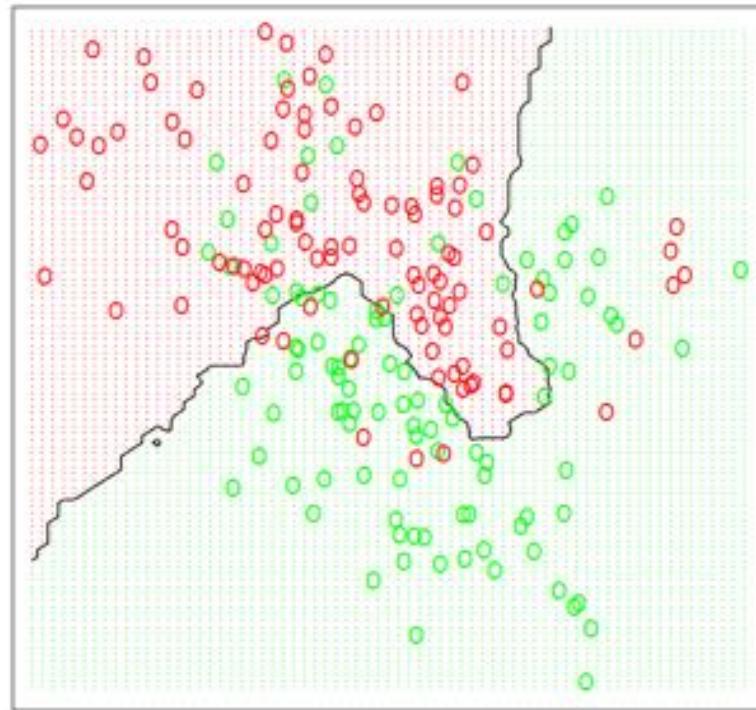
oder

$$f(x) = \frac{\sum_{i=1}^k w_i \times f(n_i)}{\sum_{i=1}^k w_i} \quad \text{mit} \quad w_i = \frac{1}{d(n_i, x)^2}$$

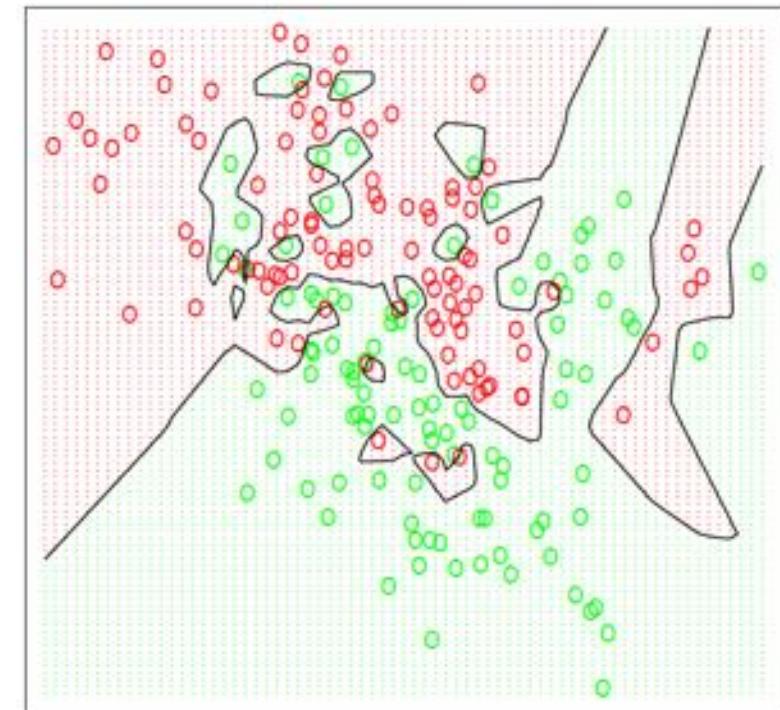
# Overfitting / Überanpassung

Die Wahl eines guten „k“ ist schwierig aber auch wichtig

15-Nearest Neighbor Classifier



1-Nearest Neighbor Classifier

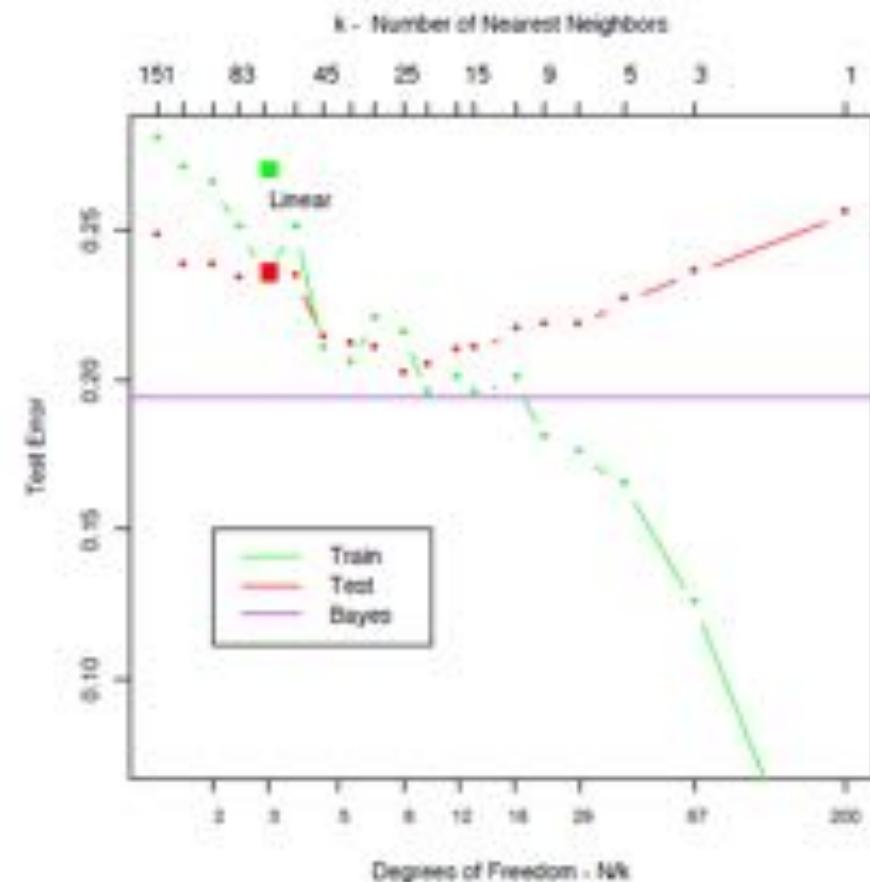


# Overfitting

Falls aus  $\mathbf{x} = \mathbf{x}^*$  gerade  $y = y^*$  folgt,  
gibt es bei  $k = 1$  keinen  
Trainingsfehler

Wenn allein der Trainingsfehler das  
Optimierungskriterium ist, würden wir  
stets  $k = 1$  nehmen und nur  
auswendig lernen

Wie wir gesehen haben, ergibt das  
vermutlich auf den Testdaten einen  
großen Fehler!





## Asymptotisches Ergebnis zu kNN

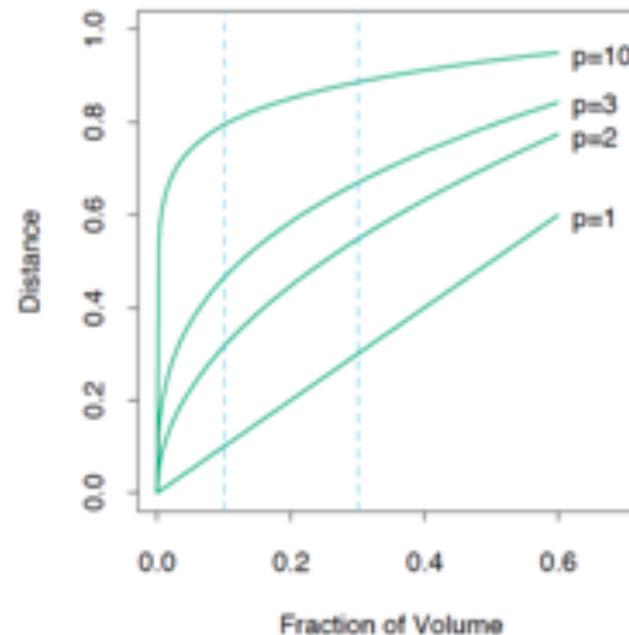
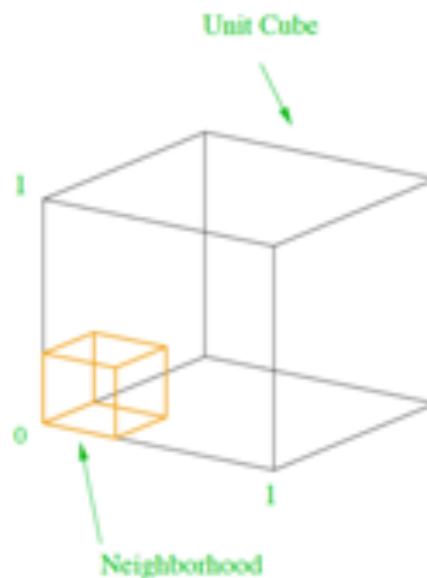
Wenn  $k/N$  gegen 0 und  $N, k$  gegen unendlich konvergieren, konvergiert die Vorhersage gegen die zu erwartende Vorhersage [Hastie et al. 2001]. Haben wir also den perfekten Lernen gefunden?

**Leider nein! Fluch der hohen Dimensionen**

- Die Dichte der Beispiele ist proportional zu  $N^{1/d}$  wobei  $d$  die Dimension ist.
- Schon bei  $D = 10$  brauchen wir 80% der möglichen Werte jedes Attributs  $X_i$ , um wenigstens 10% der Daten in einer Nachbarschaft gesehen zu haben!

**Die Dichte der Datenpunkte in der Nachbarschaft ist bei hoher Dimension furchtbar spärlich.**

# Fluch der hohen Dimensionen bei kNN



- Seitenlänge des Subquaders um  $r$ -Prozent des Volumens abzudecken für verschiedene Dimensionen  $p$ .
- Wenn 100 Beispiele bei  $p = 1$  einen dichten Raum ergeben, muss man für die selbe Dichte  $N^{1/p}$  bei  $p = 10$  schon  $100^{10}$  Beispiele sammeln:  $100^{1/1} = 100, 100^{10 \cdot \frac{1}{10}} = 100$



# Fluch der hohen Dimensionen bei kNN

## Fausregel (unabhängig von kNN)

Mit steigender Zahl der Merkmale (Dimensionen) wächst die Zahl der Beispiel (Trainingsbeispiele) exponentiell, die wir brauchen, um akkurate Vorhersagen lernen zu können!



# Problem

- Wir haben nur eine endliche Menge von Beispielen. Alle Funktionen, deren Werte durch die Beispiele verlaufen, haben einen kleinen Fehler.
- Wir wollen aber für alle Beobachtungen das richtige  $y$  voraussagen. Dann sind nicht mehr alle Funktionen, die auf die Beispiele gepasst haben, gut.
- Wir kennen nicht die wahre Verteilung der Beispiele.
- Wie beurteilen wir da die Qualität unseres Lernergebnisses?



# Reicht Auswendiglernen?

- Wenn aus  $x=x'$  folgt, dass  $y=y'$ , gilt es bei  $k=1$  keinen Trainingsfehler!
- Wenn allein der Trainingsfehler das Optimierungskriterium ist, würde wir stets  $k=1$  nehmen und nur auswendig lernen.
- Vermutlich ergibt das auf noch nicht gesehenen Daten einen großen Fehler!

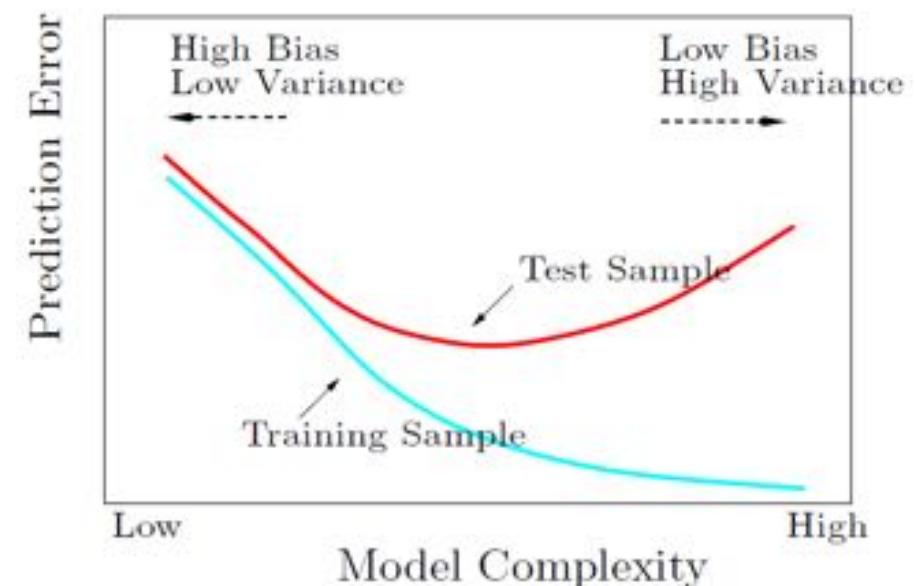
„Du kannst ein Kochbuch von vorne bis hinten auswendig lernen. Ob du wirklich kochen kannst, weißt Du erst, wenn du dich selbst hinter den Herd stellst!“

# Bias-Varianz Tradeoff kNN

Wenn man die richtige, dicht besetzte Nachbarschaft hat, verzerrt kNN die Vorhersage nicht (**kleiner Bias**).

Wenn — wie bei hohen Dimensionen — die Nachbarschaft wild variiert, schwankt auch die Güte der Vorhersage (**große Varianz**)

**“Kleiner Fehler auf den Trainingsbeispielen, aber großer Fehler auf Testbeispielen”**





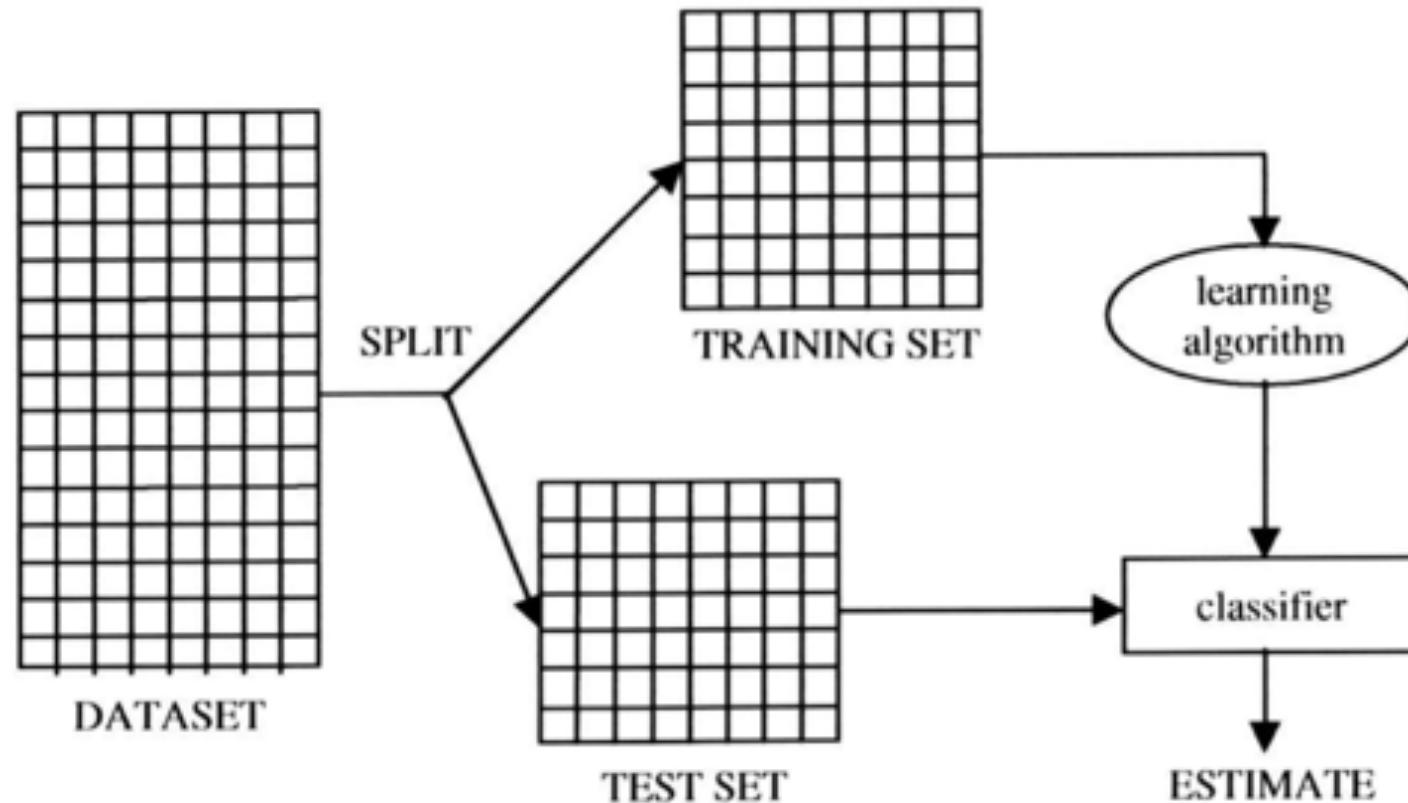
## Besser: Lern- und Testmenge

Wir teilen die Daten, die wir haben, auf:

**Lern-/Trainingsmenge:** Ein Teil der Daten übergeben wir unserem Lernalgorithmus zum Lernen.

**Testmenge:** Bei den restlichen Daten vergleichen wir die Vorhersagen mit den Daten

# Besser: Lern- und Testmenge





# Aufteilung Trainings- und Testmenge

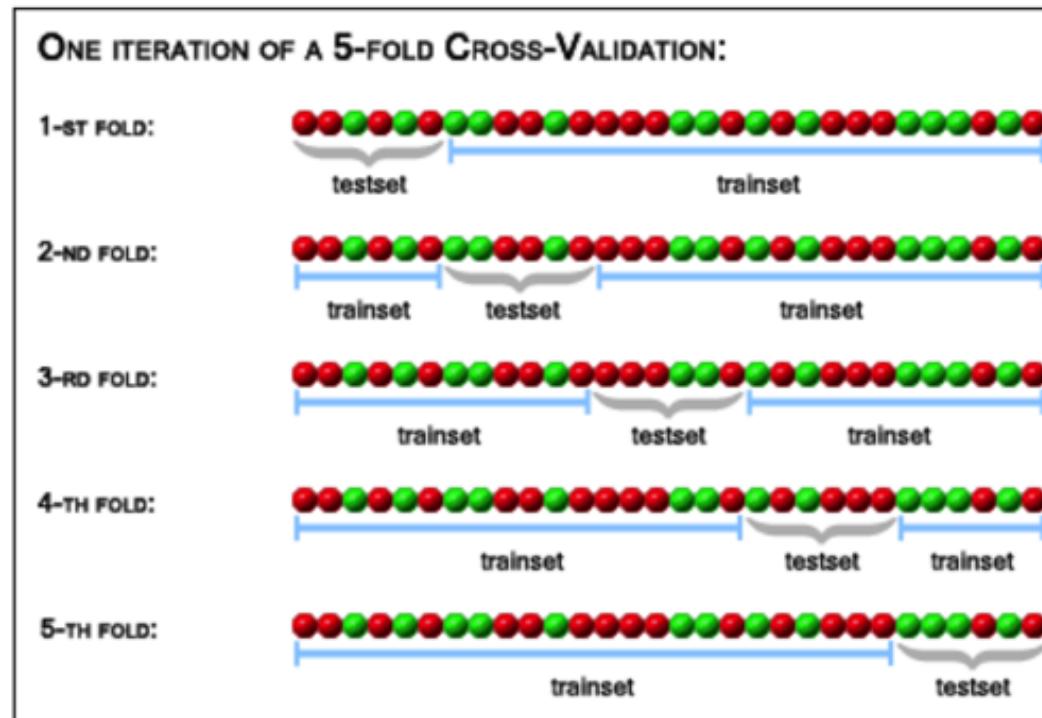
Vielleicht haben wir zufällig aus lauter Ausnahmen gelernt und testen dann an den normalen Fällen. Um das zu vermeiden, verändern wir die Aufteilung mehrfach.

- **leave-one-out:** Der Algorithmus lernt aus  $N-1$  Beispielen und testet auf dem ausgelassenen Beispiel. Dies wird  $N$  mal gemacht, die Fehler addiert.

Aus Zeitgründen wollen wir den Algorithmus nicht zu oft anwenden.

- **Kreuzvalidierung:** Die Lernmenge wird zufällig in  $n$  Mengen aufgeteilt. Der Algorithmus lernt aus  $n-1$  Mengen und testet auf der ausgelassenen Menge. Dies wird  $n$  mal gemacht.

# Kreuzvalidierung



# Verlaufsmodell der Wissensentdeckung

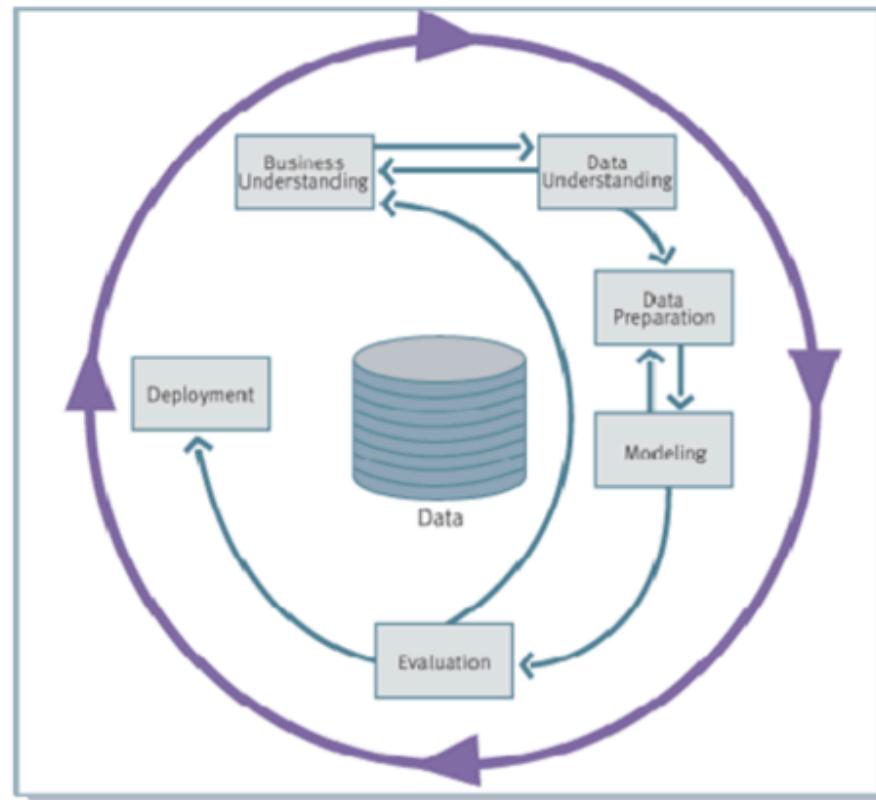


Figure : Übersicht über CRISP-DM



# CRISP: Schritte

**Problem verstehen:** Analyseziele, Situationsbewertung, Datenanalyseziele, Projektplan

**Daten verstehen:** Sammeln, beschreiben, untersuchen, Qualität von Rohdaten

**Daten aufbereiten:** Ein- und Ausschluss, Bereinigung, Transformation von Variablen

**Modellierung:** Methoden- und Testdesignwahl, Schätzung, Modellqualität

**Evaluierung:** Modell akzeptieren, Prozess überprüfen, nächste Schritte

**Nachbereitung:** Anwendungs- und Wartungsplan, Präsentation, Bericht



## Was wissen wir jetzt?

kNN ist ein einfacher Lernalgorithmus

Seine Performanz hängt von k und der gewählten Metrik ab

Fluch der hohen Dimensionen!

Sie haben das CRISP kennengelernt, das den gesamten Ablauf der Wissensentdeckung beschreibt.

Sie wissen, was Kreuzvalidierung ist

# Data Mining und Maschinelles Lernen

## Klassifikation, Regression und Lineare Modelle

Lineare Modelle sind statistisches Modelle, bei denen der Erwartungswert einer Variable  $Y$  in einer bestimmten ("linearen") Weise von Eingabeveriablen  $\mathbf{X}$  abhängt. Sie versuchen also den Zusammenhang zwischen einer abhängigen Variablen (oder Responsevariablen)  $Y$  und einer oder mehreren erklärenden Variablen  $X_1, \dots, X_k$  zu modellieren. In R kann z.B. `lm()` benutzt werden.

Was wollen wir hier kennenlernen?

- ▶ Was versteht man unter Klassifikation?
- ▶ Was versteht man unter Regression?
- ▶ Was sind lineare Modelle?
- ▶ Wie bestimmt ich lineare Modelle aus Daten?
- ▶ Wie evaluiere ich Modelle auf Daten?

Sei  $X = \{X_1, \dots, X_p\}$  eine Menge von Zufallsvariablen und  $Y \neq \emptyset$  eine Menge.

Ein **Beispiel** (oder *Beobachtung*)  $\vec{x}$  ist ein konkreter  $p$ -dimensionaler Vektor über diese Zufallsvariablen.

Eine **Menge von  $n$  Beispielen**  $\mathbf{X} = \{\vec{x}_1, \dots, \vec{x}_N\}$  können wir dann als  $(N \times p)$ -Matrix auffassen:

$$\mathbf{X} = \begin{pmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,p} \\ x_{2,1} & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ x_{N,1} & x_{N,2} & \cdots & x_{N,p} \end{pmatrix}$$

Dabei entspricht jede Zeile  $\vec{x}_i$  der Matrix  $\mathbf{X}$  einem Beispiel.

# Klassifikation und Regression

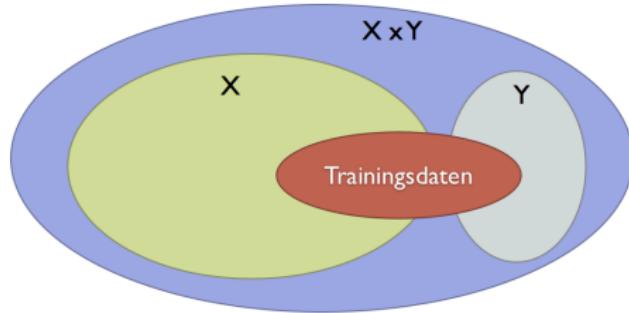
Beim *überwachten Lernen* (darum geht es hier), ist zusätzlich zu jeder Beobachtung  $\vec{x}$  ein *Label (Klasse)*  $y$  gegeben, d.h. wir haben Beobachtungen  $(\vec{x}, y) \in X \times Y$ .

$Y$  kann sowohl eine **qualitative**, als auch eine **quantitative** Beschreibung von  $\vec{x}$  sein.

Für den quantitativen Fall ist z.B.  $Y = \mathbb{R}$  und wir versuchen für unbekanntes  $\vec{x}$  den Wert  $y$  vorherzusagen **Regression**.

Im Falle qualitativer Beschreibungen ist  $Y$  eine diskrete Menge und wir nutzen  $f$  zur **Klassifikation**.

Wovon gehen wir also aus? Was ist unser Ziel?



- ▶ Wir suchen *die wahre Funktion*  $f : X \rightarrow Y$  mit
$$f(\vec{x}) = y \quad \forall (\vec{x}, y) \in X \times Y$$
- ▶ Wir haben jedoch nur eine Teilmenge der Beobachtungen gegeben (Trainingsdaten)

# Klassifikation und Regression

Auf Grundlage der Trainingsdaten suchen wir eine möglichst gute Annäherung  $\hat{f}$  an die *wahre Funktion f*.

Die Funktion  $\hat{f}$  bezeichnen wir auch als das gelernte **Modell**.

Haben wir ein Modell  $\hat{f}$  gelernt, so liefert uns dieses Modell mit

$$\hat{y} = \hat{f}(\vec{x})$$

für *neue Daten*  $\vec{x} \in X$  eine Vorhersage  $\hat{y} \in Y$ .

# Klassifikation und Regression

Im Falle der *Regression* lässt sich so für zuvor unbekannte  $\vec{x} \in X$  der Wert

$$\hat{y} = \hat{f}(\vec{x})$$

mit  $\hat{y} \in \mathbb{R}$  vorhersagen.

Dieses Modell  $\hat{f}$  lässt sich auch für die Klassifikation nutzen, bei der z.B.  $\hat{y} \in \{-1, +1\}$  vorhergesagt werden sollen:

$$\hat{y} = \begin{cases} +1, & \text{falls } \hat{f}(\vec{x}) \geq \theta \\ -1, & \text{sonst} \end{cases}$$

Hier ist  $\theta$  ein vorgegebener Schwellwert.

# Beispiel

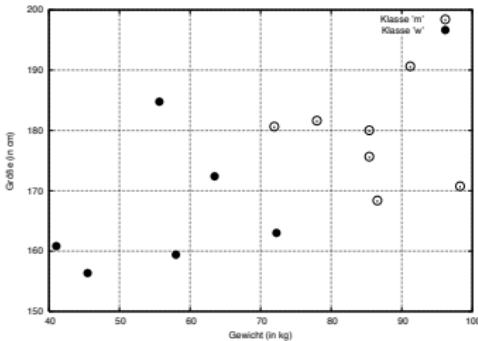
Gegeben seien Gewicht ( $X_1$ ) und Größe ( $X_2$ ) einiger Personen und ein Label  $y \in \{m, w\}$ :

	$X_1$	$X_2$	$Y$
$x_1$	91	190	m
$x_2$	60	170	w
$x_3$	41	160	w
:	:	:	:

# Beispiel

Es wird nun eine Funktion  $\hat{f}$  gesucht, die für neue Daten  $\vec{x}$  das Attribut  $Y$  (Geschlecht) voraussagt, also

$$\hat{y} = \begin{cases} m, & \text{falls } \hat{f}(x) > \theta \\ w, & \text{sonst} \end{cases}$$



Welche Art von Funktionen sind denkbar?

Lineare Funktionen als einfachste Funktionenklasse:

$$y = f(x) = mx + b \quad \text{Gerade im } \mathbb{R}^2$$

Allerdings betrachten wir als Beispielraum den  $\mathbb{R}^p$ , d.h. wir brauchen eine verallgemeinerte Form:

$$y = f(\vec{x}) = \sum_{i=1}^p \beta_i x_i + \beta_0 \quad \text{mit } \beta_0 \in \mathbb{R}, \vec{x}, \vec{\beta} \in \mathbb{R}^p \quad (1)$$

Die Funktion  $f$  wird also durch  $\vec{\beta}$  und  $\beta_0$  festgelegt und sagt uns für ein gegebenes  $\vec{x}$  das entsprechende  $y$  voraus

# Notation, Vereinbarungen



Bei genauerer Betrachtung von Formel (1) lässt sich  $\sum_{i=1}^p \beta_i x_i$  als Matrizenmultiplikation oder Skalarprodukt schreiben, also

$$y = \sum_{i=1}^p \beta_i x_i + \beta_0 = \vec{x}^T \vec{\beta} + \beta_0 = \langle \vec{x}, \vec{\beta} \rangle + \beta_0$$

Zur einfacheren Darstellung von  $f$ , wird  $\beta_0$  in den Vektor  $\vec{\beta}$  codiert, indem jedes Beispiel  $x = (x_1, \dots, x_p)$  aufgefasst wird als  $(p+1)$ -dimensionaler Vektor

$$(x_1, \dots, x_p) \mapsto (1, x_1, \dots, x_p)$$

Dies ermöglicht die Darstellung von  $f$  als:

$$y = f(\vec{x}) = \sum_{i=0}^p \beta_i x_i = \vec{x}^T \vec{\beta} = \langle \vec{x}, \vec{\beta} \rangle$$

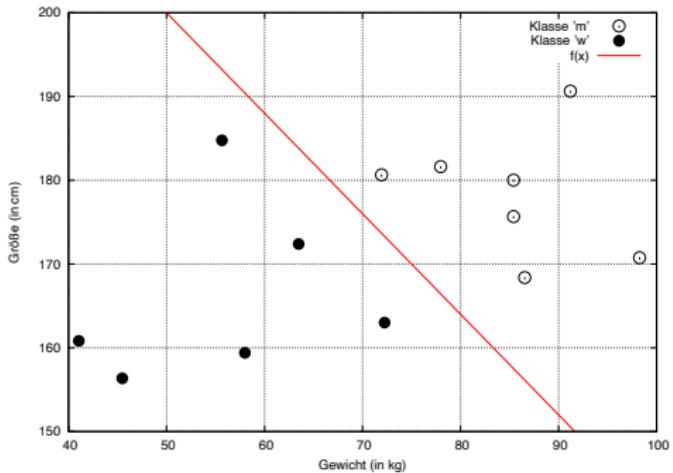
# Was haben wir nun gemacht?

Wir haben (bei der Beschränkung auf lineare Modelle) nun eine Darstellung für das, was wir *lernen* wollen:

$$y = \hat{f}(\vec{x}) = \vec{x}^T \vec{\beta}$$

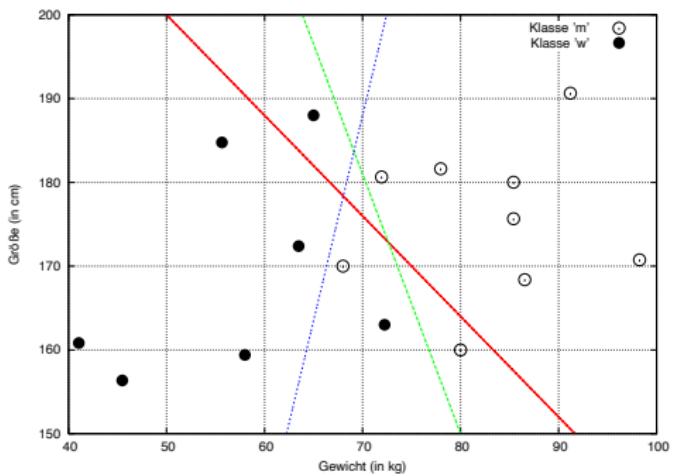
Wir haben die Zielfunktion  $\hat{f}$  in Abhängigkeit von  $\vec{\beta}$  geschrieben und müssen *nur noch* das passende  $\vec{\beta}$  finden.

# Beispiel: Ein mögliches $\vec{\beta}$



$$f(\vec{x}) = \vec{x}^T \hat{\vec{\beta}} \quad \text{mit} \quad \hat{\vec{\beta}} = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \end{pmatrix} = \begin{pmatrix} 260 \\ 1 \\ 1.2 \end{pmatrix} \theta = 550$$

# Es ist nicht garantiert, dass $\vec{\beta}$ immer passt!



Unsere linearen Modelle sind durch  $\vec{\beta}$  parametrisiert, das Lernen eines Modells haben wir also auf die Wahl eines  $\vec{\beta}$  abgewälzt.

Das wirft eine Reihe von Fragen auf:

- ▶ Was ist ein gutes  $\vec{\beta}$ ?
- ▶ Gibt es ein optimales  $\vec{\beta}$ ?
- ▶ Welche Möglichkeiten haben wir, unser Modell zu beurteilen?

Eine Möglichkeit: Berechne den *Trainingsfehler*

$$Err(\vec{\beta}) = \sum_{i=1}^N |y_i - \hat{f}(\vec{x}_i)| = \sum_{i=1}^N |y_i - x_i^T \vec{\beta}|$$

Häufig wird als Fehlerfunktion die *quadratische Fehlersumme* (RSS) verwendet:

$$\begin{aligned} RSS(\vec{\beta}) &= \sum_{i=1}^N (y_i - \vec{x}_i^T \vec{\beta})^2 \\ &= (\vec{y} - \mathbf{X}\vec{\beta})^T (\vec{y} - \mathbf{X}\vec{\beta}) \end{aligned}$$

Wir wählen jetzt  $\vec{\beta}$  derart, dass der Fehler minimiert wird:

$$\min_{\vec{\beta} \in \mathbb{R}^p} RSS(\vec{\beta}) \tag{2}$$

⇒ Konvexes (=“einfaches”) Minimierungsproblem!

# Minimierung von $RSS(\vec{\beta})$

Um  $RSS(\vec{\beta})$  zu minimieren, bilden wir die partielle Ableitung nach  $\vec{\beta}$ :

$$\frac{\partial RSS(\vec{\beta})}{\partial \beta} = \mathbf{X}^T (\mathbf{y} - \mathbf{X}\vec{\beta})$$

Notwendige Bedingung für die Existenz eines (lokalen) Minimums von  $RSS$  ist

$$\frac{\partial RSS(\vec{\beta})}{\partial \beta} = \mathbf{X}^T (\mathbf{y} - \mathbf{X}\vec{\beta}) = 0$$

Ist  $\mathbf{X}^T \mathbf{X}$  regulär, so erhalten wir

$$\hat{\vec{\beta}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y} \tag{3}$$

# Reguläre Matrix

Wenn es zu einer quadratischen Matrix  $\mathbf{X}$  eine Matrix  $\mathbf{X}^{-1}$  gibt mit

$$\mathbf{X}\mathbf{X}^{-1} = \mathbf{X}^{-1}\mathbf{X} = \mathbf{I}$$

Einheitsmatrix

$$I = \begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ . & . & \dots & 0 \\ . & . & \dots & 0 \\ 0 & 0 & \dots & 1 \end{pmatrix}$$

dann ist die Matrix  $\mathbf{X}$  invertierbar oder *regulär*, sonst *singulär*.

# Optimales $\hat{\beta}$ ?

Mit Hilfe der Minimierung der (quadratischen) Fehlerfunktion  $RSS$  auf unseren Trainingsdaten haben wir ein (bzgl.  $RSS$ ) optimales  $\hat{\beta}$  gefunden.

Bei einem konvexen Problem ist das lokale auch das globale Minimum.  
Damit liefert unser Modell Voraussagen  $\hat{y}$  für  $\vec{x} \in X$ :

$$\hat{y} = \hat{f}(\vec{x}) = \vec{x}^T \hat{\beta}$$

# Sind wir schon fertig?

- ▶ Schön wär's!
- ▶ Aber drei Gründe sprechen für weitere Arbeit:
  1. Es ist nicht immer so einfach, z.B. dann nicht, wenn wir viele Dimensionen haben (Fluch der hohen Dimension).
  2. Vielleicht lassen sich die Beispiele nicht linear trennen!
  3. Nur den Fehler zu minimieren reicht nicht aus, wir suchen noch nach weiteren Beschränkungen, die zu besseren Lösungen führen.
- ▶ Also schauen wir uns den Fehler noch einmal genauer an, stoßen auf Bias und Varianz und merken, dass wir noch keine perfekte Lösung haben.

- ▶ Bisher haben wir mit RSS die Fehler einfach summiert.
- ▶ Wir wollen aber einbeziehen, wie wahrscheinlich der Fehler ist – vielleicht ist er ja ganz unwahrscheinlich! Das machen wir über den Erwartungswert.
- ▶ Wir können sehr unterschiedliche Stichproben als Beispielmenge haben. Der Fehler soll sich auf alle möglichen Trainingsmengen beziehen – nicht nur eine, zufällig günstige!

# Zur Erinnerung: Erwartungswert

## Erwartungswert

Sei  $X$  eine **diskrete Zufallsvariable**, mit Werten  $x_1, \dots, x_n$  und  $p_i$  die Wahrscheinlichkeit für  $x_i$ . Der Erwartungswert von  $X$  ist

$$E(X) = \sum_i x_i p_i = \sum_i x_i P(X = x_i)$$

Ist  $X$  eine **stetige Zufallsvariable** und  $f$  die zugehörige Wahrscheinlichkeitsdichtefunktion, so ist der Erwartungswert von  $X$

$$E(X) = \int_{-\infty}^{\infty} x f(x) dx$$

# Erwartungswert (Eigenschaften)



Seien  $X, Y$  und  $X_1, \dots, X_n$  Zufallsvariablen, dann gilt:

- Der Erwartungswert ist additiv, d.h. es gilt

$$E\left(\sum_{i=1}^n X_i\right) = \sum_{i=1}^n E(X_i) \quad (4)$$

- Ist  $Y = kX + d$ , so gilt für den Erwartungswert

$$E(Y) = E(kX + d) = kE(X) + d \quad (5)$$

- Sind die Zufallsvariablen  $X_i$  stochastisch unabhängig, gilt

$$E\left(\prod_{i=1}^n X_i\right) = \prod_{i=1}^n E(X_i)$$

Über den Erwartungswert einer Zufallsvariablen  $X$  sind mehrere Eigenschaften von  $X$  definiert, die helfen,  $X$  zu charakterisieren:

## Varianz

Sei  $X$  eine Zufallsvariable mit  $\mu = E(X)$ . Die **Varianz**  $Var(X)$  ist definiert als

$$Var(X) := E((X - \mu)^2).$$

Die Varianz wird häufig auch mit  $\sigma^2$  bezeichnet.

## Standardabweichung

Die **Standardabweichung**  $\sigma$  einer Zufallsvariable  $X$  ist definiert als

$$\sigma := \sqrt{Var(X)}$$

Varianz = durchschnittliche Abweichung der Werte vom Zentrum der Verteilung.

Diese durchaus naheliegende und fast schon intuitive Vorgehensweise für die Bestimmung der Varianz (Streuung) überrascht nur in einem Detail: Die zu addierenden Differenzen werden vorab quadriert um zu verhindern, dass sich positive und negative Abweichungen vom arithmetischen Mittel gegenseitig neutralisieren.

## Verschiebungssatz

Sei  $X$  eine Zufallsvariable, für die Varianz gilt

$$\text{Var}(X) = E(X - E(X))^2 = E(X^2) - (E(X))^2$$

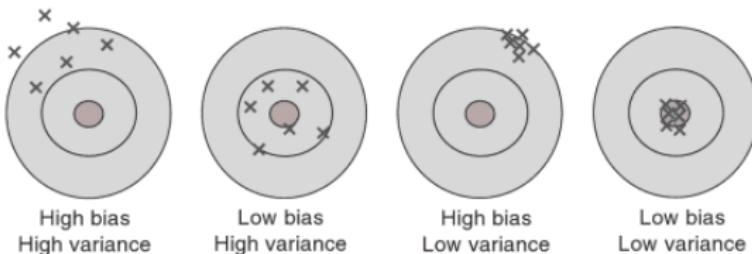
Eine weitere Charakteristik, die häufig zur Beschreibung von erwarteten Fehlern verwendet wird, ist die Verzerrung:

## Verzerrung (Bias)

Sei  $Y$  eine Zufallsvariable, dann ist die Verzerrung definiert als der erwartete Schätzfehler für  $Y$  also wie im Durchschnitt die Schätzungen vom wahren Mittelwert abweichen

$$Bias(\hat{y}) = E(Y - \hat{y}) = E(Y) - \hat{y}$$

# Bias-Varianz Trade-Off



**Bias Variance Decomposition.** Figure 1. The bias-variance decomposition is like trying to hit the bullseye on a dartboard. Each dart is thrown after training our “dart-throwing” model in a slightly different manner. If the darts vary wildly, the learner is *high variance*. If they are far from the bullseye, the learner is *high bias*. The ideal is clearly to have both low bias and low variance; however this is often difficult, giving an alternative terminology as the bias-variance “dilemma” (*Dartboard analogy*, Moore & McCabe (2002))

- ▶ Fehlerfunktion  $L(y, \hat{y})$  für gelernte Modelle  $\hat{f}$ 
  - ▶ absolut  $\sum(y_i - \hat{y}_i)$
  - ▶ quadratisch  $\sum(y_i - \hat{y}_i)^2$
  - ▶ 0,1-Fehler  $\sum \delta_i$ ,  $\delta = 1$ , falls  $y = \hat{y}$ , sonst 0.
- ▶ Es geht um  $Y$ . Wir unterscheiden
  - ▶ das wahre  $y$ ,
  - ▶ das in der Beispieldmenge genannte  $y$ ,
  - ▶ das vom Modell vorhergesagte  $\hat{y}$
- ▶ Wir wollen den Erwartungswert des Fehlers minimieren.
- ▶ Wir mitteln über alle möglichen Beispieldmengen  $\mathcal{T}$ .

# Erwartungswert des Fehlers einer Regression minimieren!

**Erwarteter quadratischer Vorhersagefehler:** Gelernte Funktion  $\hat{f} : X \rightarrow Y$ , der Erwartungswert ihres Fehlers ist:

$$EPE(f) = E(Y - \hat{f}(X))^2 \quad (6)$$

**Optimierungsproblem:** Wähle  $\hat{f}$  so, dass der erwartete Fehler minimiert wird!

$$\hat{f}(x) = \operatorname{argmin}_c E_{Y|X}((Y - c)^2 | X = x) \quad (7)$$

**Lösung (Regressionsfunktion):**  $\hat{f}(x) = E(Y | X = x)$

Zwei Aspekte machen den erwarteten Fehler aus, die Verzerrung (Bias) und die Varianz. Wir wollen den Fehler an einem Testpunkt  $x_0 = 0$  angeben und mitteln über allen Trainingsmengen  $\mathcal{T}$ .

- ▶ Wir gehen davon aus, dass die Angaben in den Daten nicht immer ganz stimmen, so dass es einen Messfehler  $\epsilon$  gibt, dessen Erwartungswert aber 0 ist.
- ▶ Der Bias ist unabhängig vom Beispielsatz und 0 bei einem perfekten Lerner.
- ▶ Die Varianz ist unabhängig vom wahren Wert  $y$  und 0 bei einem Lerner, der bei allen Beispielsätzen dasselbe ausgibt.

# Dekomposition in Bias und Varianz



Wir nehmen für unser Modell an, dass  $Y = f(x) + \epsilon$  und  $E(\epsilon) = 0$ .

$$\begin{aligned} EPE(x_0) &= E_{Y,\mathcal{T}}((Y - \hat{y}_0)^2 | x_0) \\ &= E_Y((Y - f(x_0))^2 | x_0) + \sigma^2 \text{Rauschen} \\ &\quad E_{\mathcal{T}}((f(x_0) - E_{\mathcal{T}}(\hat{y}_0))^2 | x_0) + \text{Bias}^2 \\ &\quad E_{\mathcal{T}}((E_{\mathcal{T}}(\hat{y}_0) - \hat{y}_0)^2 | x_0) \quad \text{Varianz} \end{aligned}$$

Wie das?!

Haupttrick: kreatives Einfügen von Termen,  $+a - a$ , die nichts ändern, aber Umformungen erlauben. Wir leiten das hier aber nicht her.

# Bias und Varianz bei linearen Modellen

Das lineare Modell wird an die Daten angepasst durch

$$\hat{f}_p(\vec{x}) = \hat{\beta}^T \vec{x}$$

Der Fehler ist dann für ein beliebiges  $\vec{x}$ :

$$Err(\vec{x}_0) = E[(Y - \hat{f}_p(\vec{x}_0))^2 | X = \vec{x}_0] \quad (8)$$

$$= \sigma_\epsilon^2 + Var(\hat{f}_p(\vec{x}_0)) + [f(\vec{x}_0) - E\hat{f}_p(\vec{x}_0)]^2 \quad (9)$$

Die Anpassung des linearen Modells geht über alle  $N$  Beispiele und gewichtet alle  $p$  Merkmale (s. (3)).

Diese Varianz ist von  $x_i$  zu  $x_i$  verschieden. Im Mittel über allen  $\vec{x}_i$  ist  $Var(\hat{f}_p) = (p/N)\sigma_\epsilon^2$ .

# Zusammenhang zwischen Anzahl der Beispiele, der Attribute und erwartetem Fehler

Modellkomplexität ( $p, N$ ) und Varianz der Schätzungen bei unterschiedlichen Trainingsmengen hängen bei linearen Modellen direkt zusammen.

Gemittelt über alle  $x_i$  ist der Trainingsfehler linearer Modelle:

$$\frac{1}{N} \sum_{i=1}^N Err(x_i) = \sigma_\epsilon^2 + \frac{p}{N} \sigma_\epsilon^2 + \frac{1}{N} \sum_{i=1}^N [f(\vec{x}_i) - E\hat{f}(\vec{x}_i)]^2 \quad (10)$$

Wir haben also wieder das Rauschen, die Varianz, die die Schwankungen der Schätzungen angibt, und den Bias, der sich auf die Differenz von Schätzung und Wahrheit bezieht (in-sample error).

# Fluch der hohen Dimension bei linearen Modellen

- ▶ Leider mussten wir annehmen, dass das Modell genau passt, um den erwarteten Fehler klein zu halten.
- ▶ Wir wissen aber nicht, welche Art von Funktion gut zu unseren Daten passt! **Modellselektion** ist schwierig!
- ▶ Das Modell muss immer komplizierter werden, je mehr Dimensionen es gibt.
- ▶ Bei linearen Modellen entspricht die Komplexität des Modells direkt  $p$ , denn  $\beta$  hat so viele Komponenten wie  $p$  bzw.  $p + 1$ .

Die grünen und roten Datenpunkte werden durch eine Ebene getrennt.

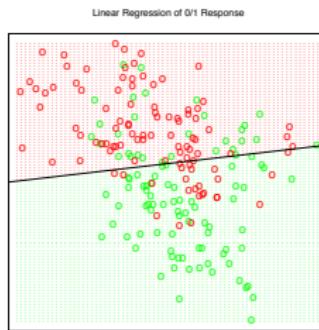


Figure 2.1: *A classification example in two dimensions. The classes are coded as a binary variable—**GREEN** = 0, **RED** = 1—and then fit by linear regression. The line is the decision boundary defined by  $x^T \hat{\beta} = 0.5$ . The red shaded region denotes that part of input space classified as **RED**, while the green region is classified as **GREEN**.*

# Was wissen Sie jetzt?

- ▶ Sie haben theoretisch lineare Modelle für Klassifikation und Regression kennengelernt.
- ▶ Sie kennen das **Optimierungsproblem** der kleinsten Quadrate RSS (Gleichung 2) für lineare Modelle (Gleichung 3).
- ▶ Sie kennen den erwarteten Fehler EPE bei linearen Modellen (Gleichung 6).
- ▶ Sie kennen den **Fluch der hohen Dimension** bei linearen Modellen: Komplexität und Varianz hängen an der Dimension! Der Bias kann sehr hoch sein, wenn die Beispiele tatsächlich nicht linear separierbar sind.