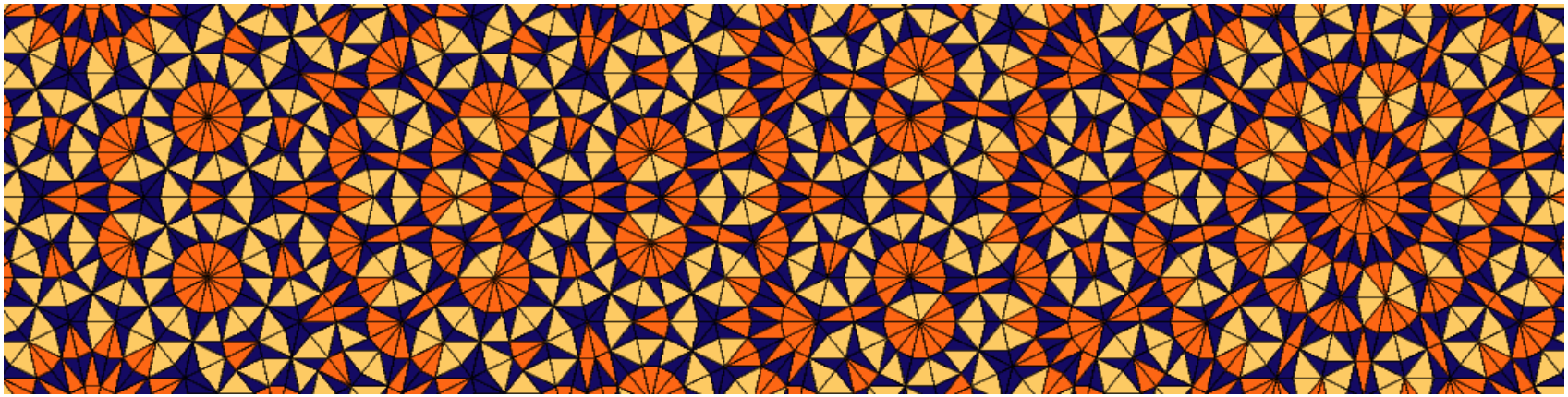
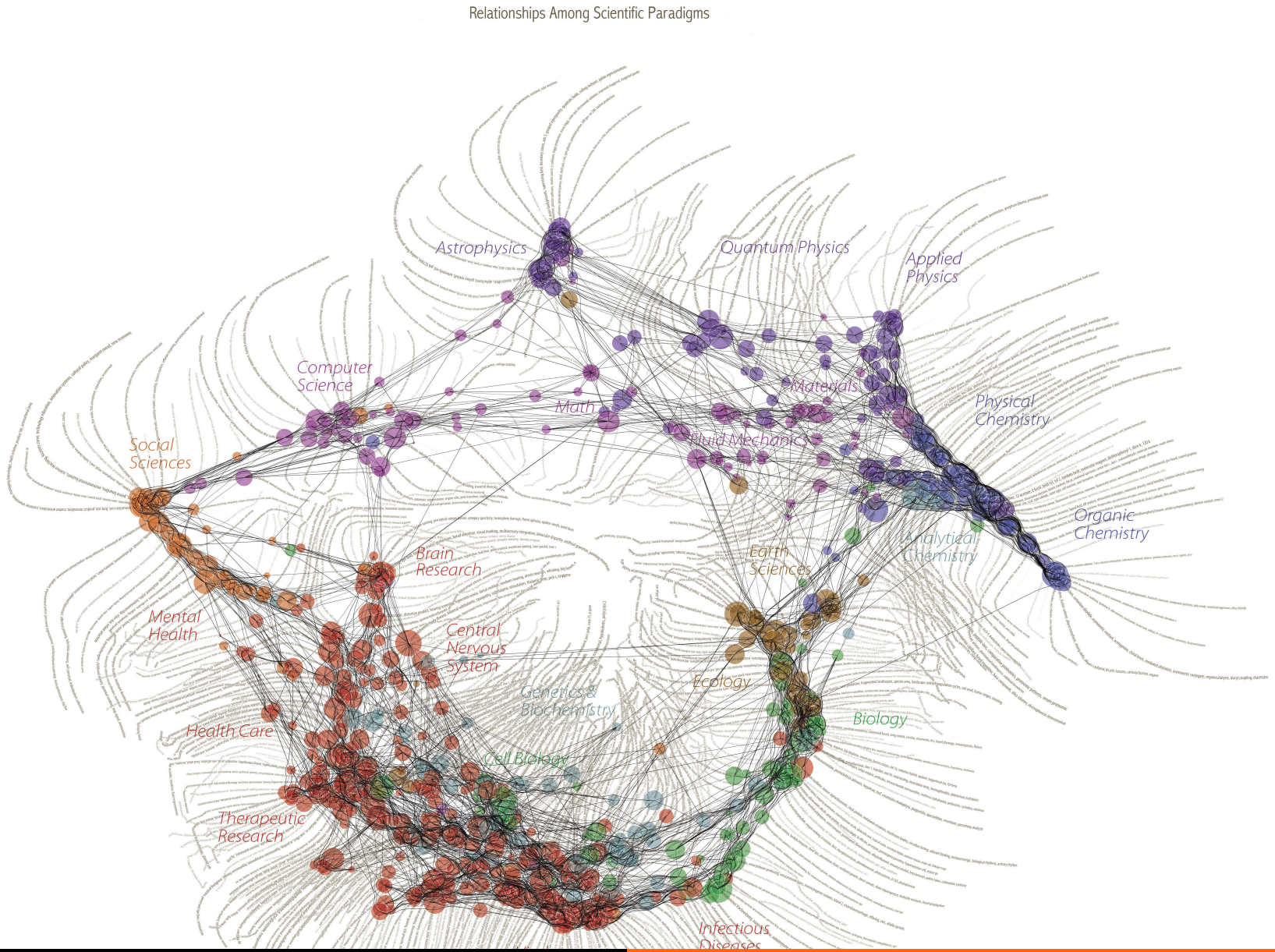
Panorama der Mathematik und Informatik

27/28: Forschung in BI: Maschinelles Lernen

Dirk Frettlöh
Technische Fakultät / Richtig Einsteigen

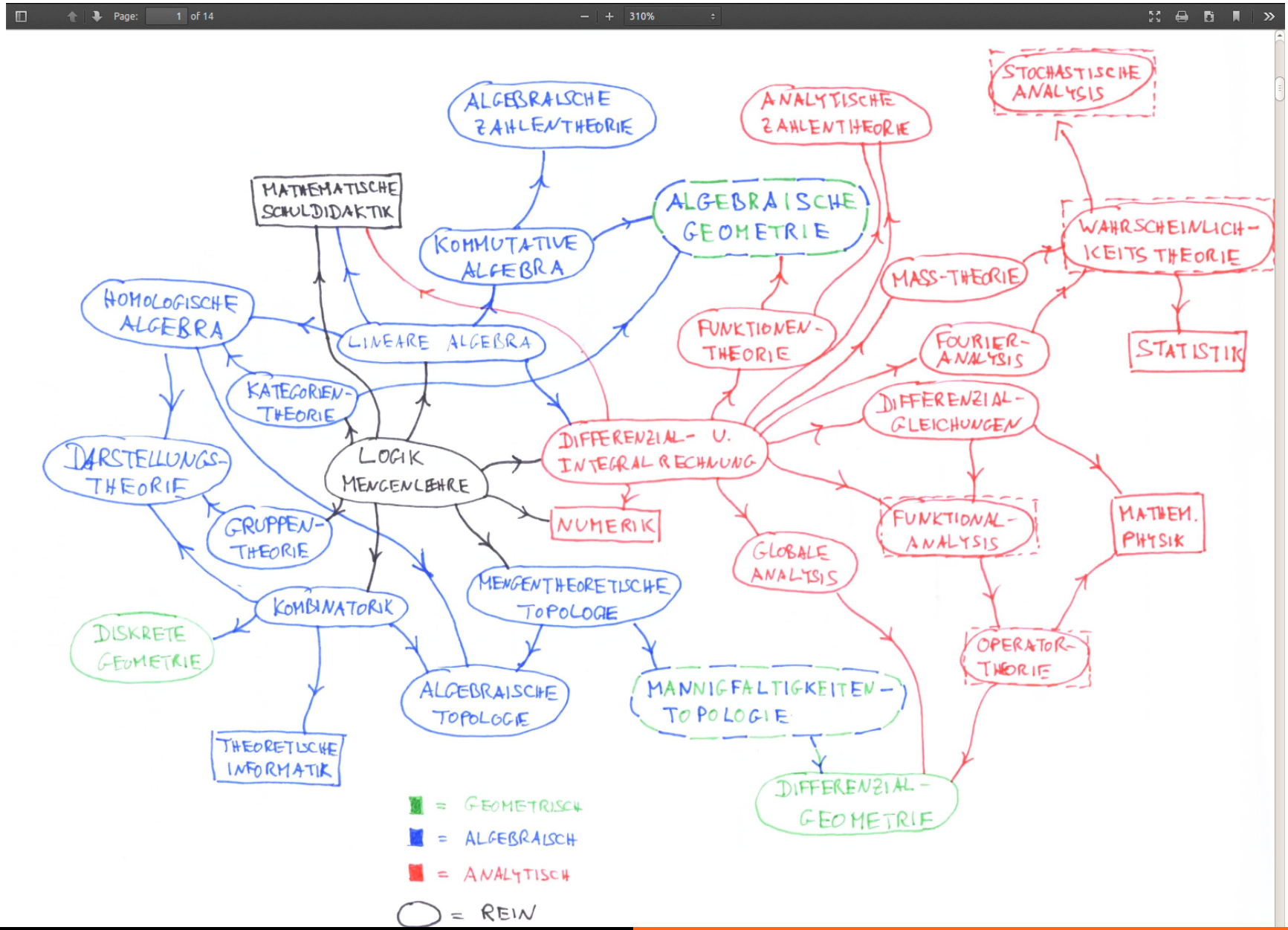


Landkarte der (MINT-)Wissenschaften

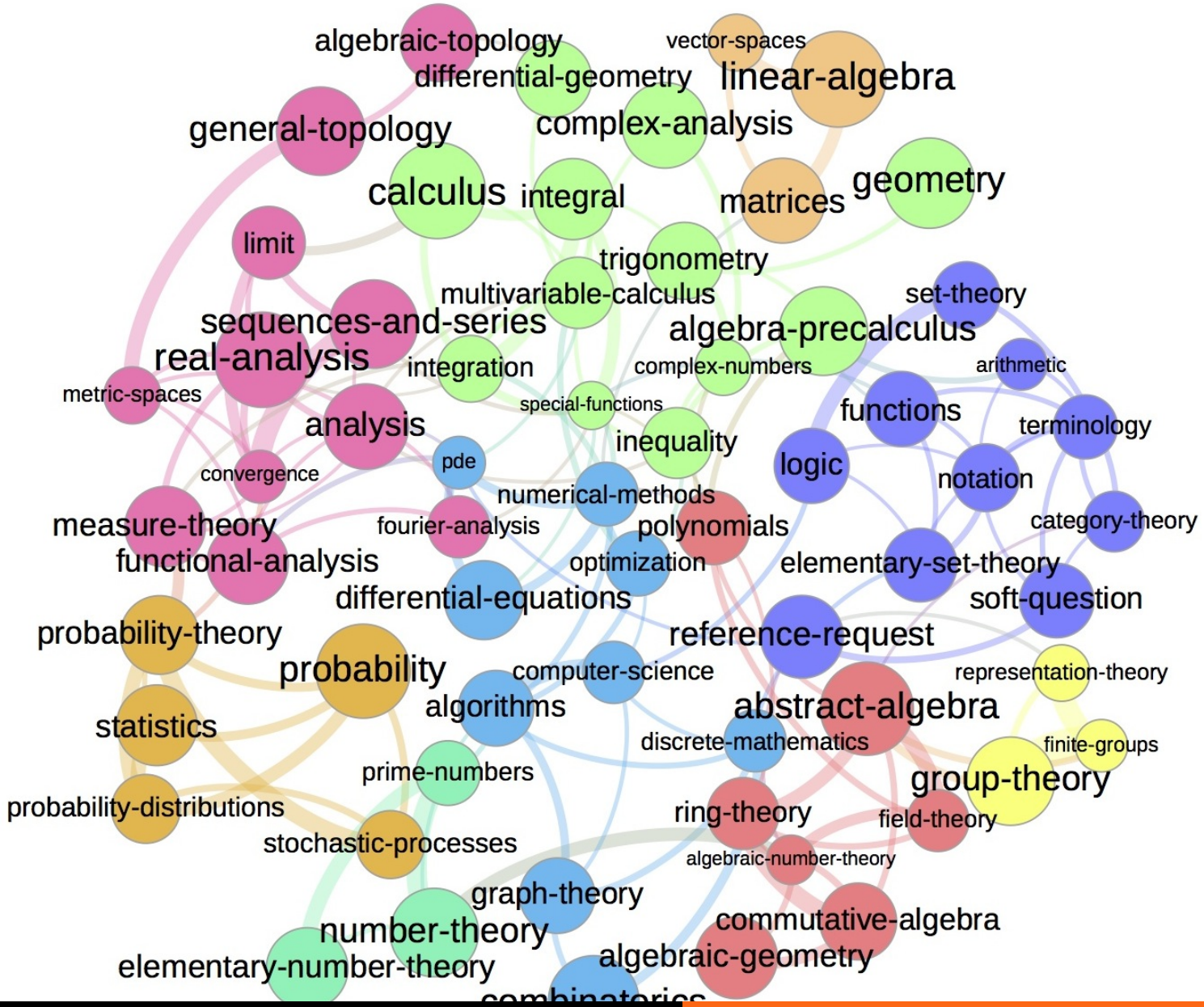


Landkarte der Mathematik

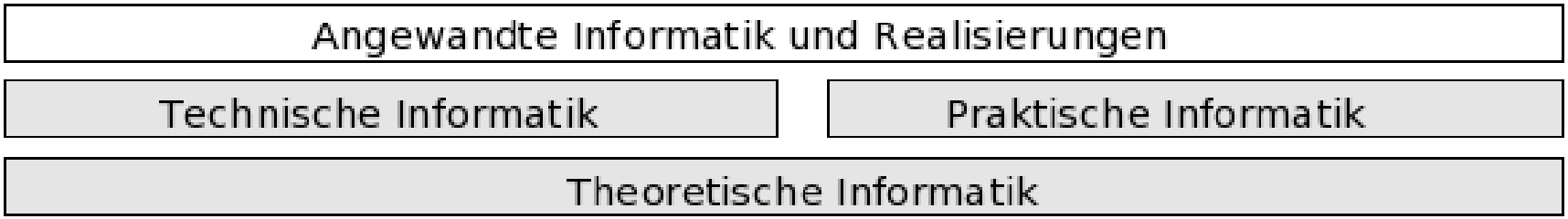
Landkarte der Mathematik



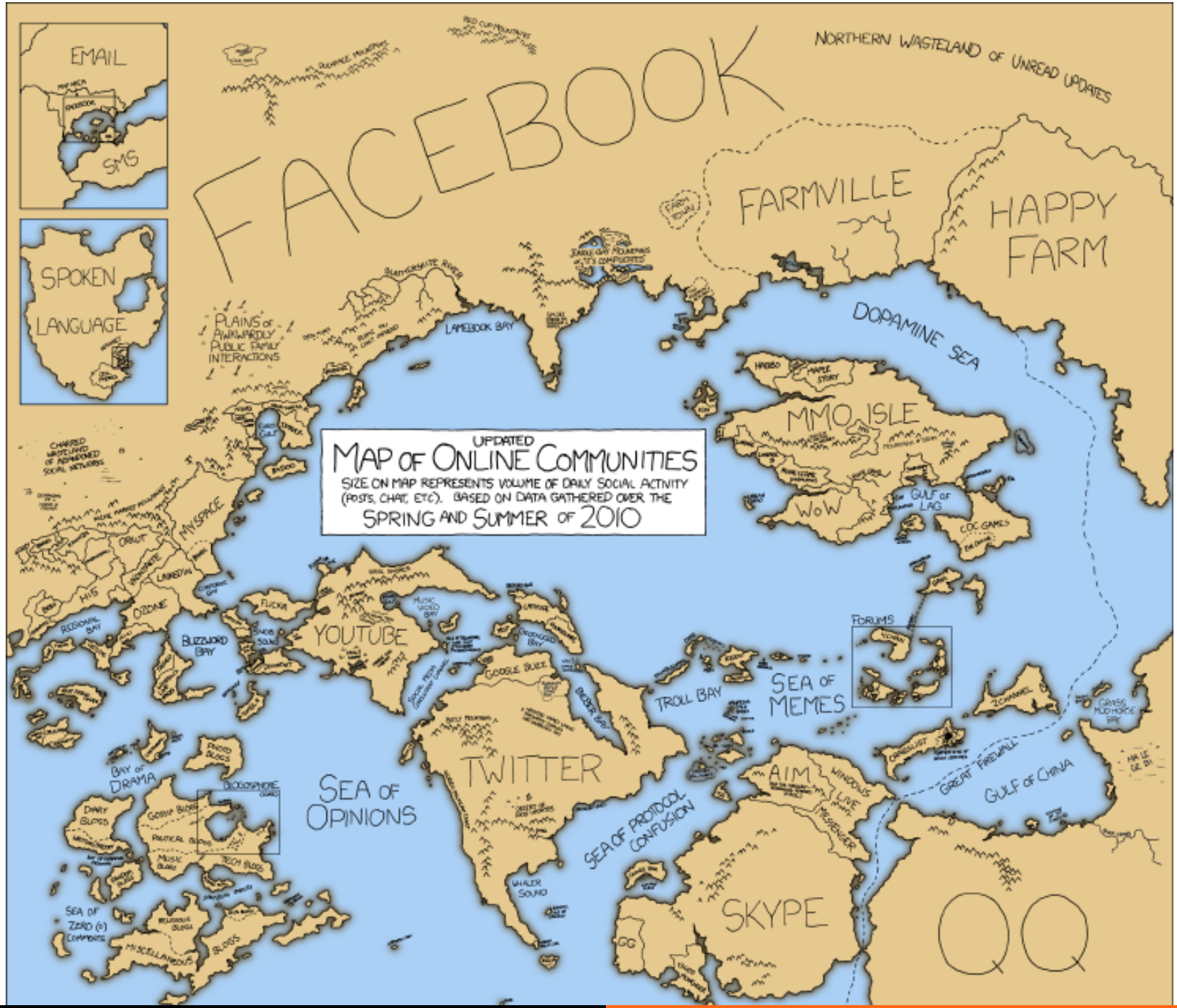
Landkarte der Mathematik



Landkarte der Informatik



Naja, in der Informatik ändert sich die Landschaft in wenigen Jahren radikal. Eine Landkarte ist das falsche Bild. Würde man sich die Mühe machen, z.B. für Onlinecommunities:


EMAIL
SMS
SPOKEN LANGUAGE
FACEBOOK
NORTHERN WASTELAND OF UNREAD UPDATES
FARMVILLE
HAPPY FARM
PLAINS OF AWKWARDLY PUBLIC FAMILY INTERACTIONS
DOPAMINE SEA
MMO ISLE
WoW
UPDATED
MAP OF ONLINE COMMUNITIES
SIZE ON MAP REPRESENTS VOLUME OF DAILY SOCIAL ACTIVITY (POSTS, CHAT, ETC). BASED ON DATA GATHERED OVER THE SPRING AND SUMMER OF 2010
MYSPACE
YOUTUBE
FORUMS
SEA OF MEMES
TROLL BAY
TWITTER
SEA OF OPINIONS
BAY OF DRAMA
AIM
GULF OF CHINA
SEA OF PROTOCOL CONFUSION
SKYPE
QQ

...so wäre die Karte nach 10 Jahren überholt.

Hier sehe ich z.B. nicht tumblr oder instagram (zu neu); oder AOL oder geocities (zu alt).

Dafür google buzz, das gibt's nicht mehr. Farmville ist lange nicht mehr so ein Hit.

Dennoch: wo auf der Karte der Informatik liegt Bielefeld?

Siehe techfak.de: zwei große Forschungsinstitute

- Biotechnologie und Bioinformatik (CeBiTec): Produkte aus Bakterien, Algen... sowie Verarbeitung großer Datenmengen aus DNA-Sequenzierung
- Kognitive Informatik (CITEC): Maschinen sehen, greifen, kommunizieren, lernen

Also auch ein Schwerpunkt: Maschinelles Lernen

Maschinelles Lernen

Computer sollen etwas lernen (Schach, Bilderkennung,...) ohne spezifisch dafür programmiert zu sein.

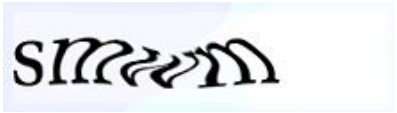
Typisches Problem: m Bilder, entweder Katze oder Hund, Computer soll daran den Unterschied lernen. **(Trainingsdaten)**

Danach soll er neue Bilder, entweder Katze oder Hund, möglichst gut unterscheiden können. **(Testdaten)**

Knifflig....

- Deep Blue beat Kasparov at chess in 1997.
- Watson beat the brightest trivia minds at Jeopardy in 2011.
- AlphaGo beats Go master Lee Se-dol 2016.
- 2013: bester Hund-oder-Katze Algorithmus: Fehlerrate 1,1 %

(Mensch: ??? aber wohl besser; Menschen waren verboten)



Maschinelles Lernen

Allgemein: Modellierung als Vektoren $x_1, \ldots, x_m \in \mathbb{R}^d$.

Typischerweise ist d groß: Z.B. Zahl der Pixel des Fotos, oder Parameter: (Fell?, Bellt?, Farbe, Länge,...)

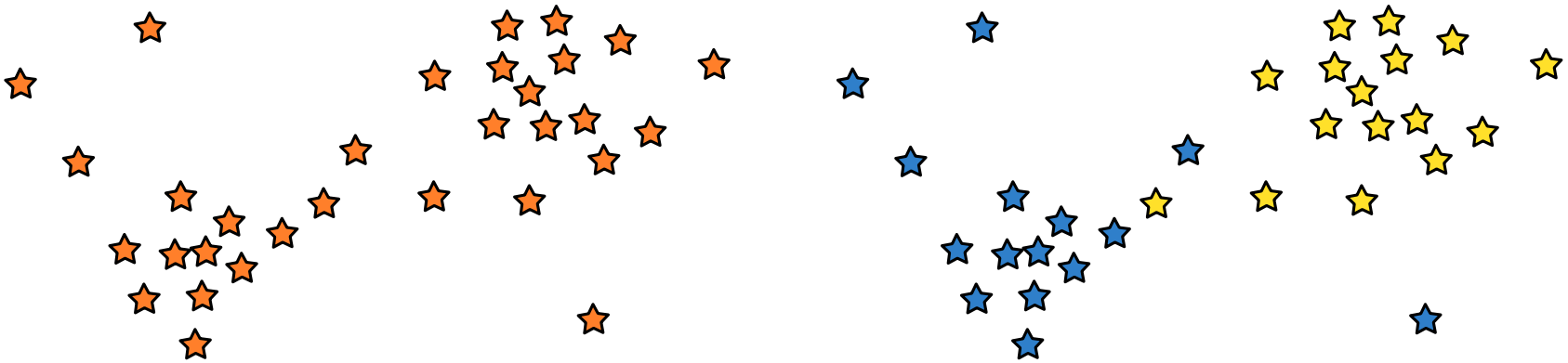
- Hund z.B.: (1, 1, bc9a81, 110 cm...)
- Katze z.B. (1, 0, ccb4a1, 30 cm...)

...kann man auch in Vektoren in $\mathbb{R}^d$ übersetzen.

Maschinelles Lernen

Grobe Kategorien:

- Unüberwachtes Lernen (oft: *Clustering*) findet selbst eine Struktur in den Trainingsdaten
- Überwachtes Lernen: Trainingsdaten tragen "labels" (z.B. "Hund", "Katze"...)
- Semi-überwachtes Lernen (oder "reinforced learning"): in etwa der ganze Rest. Z.B. nicht alle label der Trainingsdaten werden verraten, oder nur "gewonnen/verloren" (ohne Fehler explizit aufzuzeigen), oder...



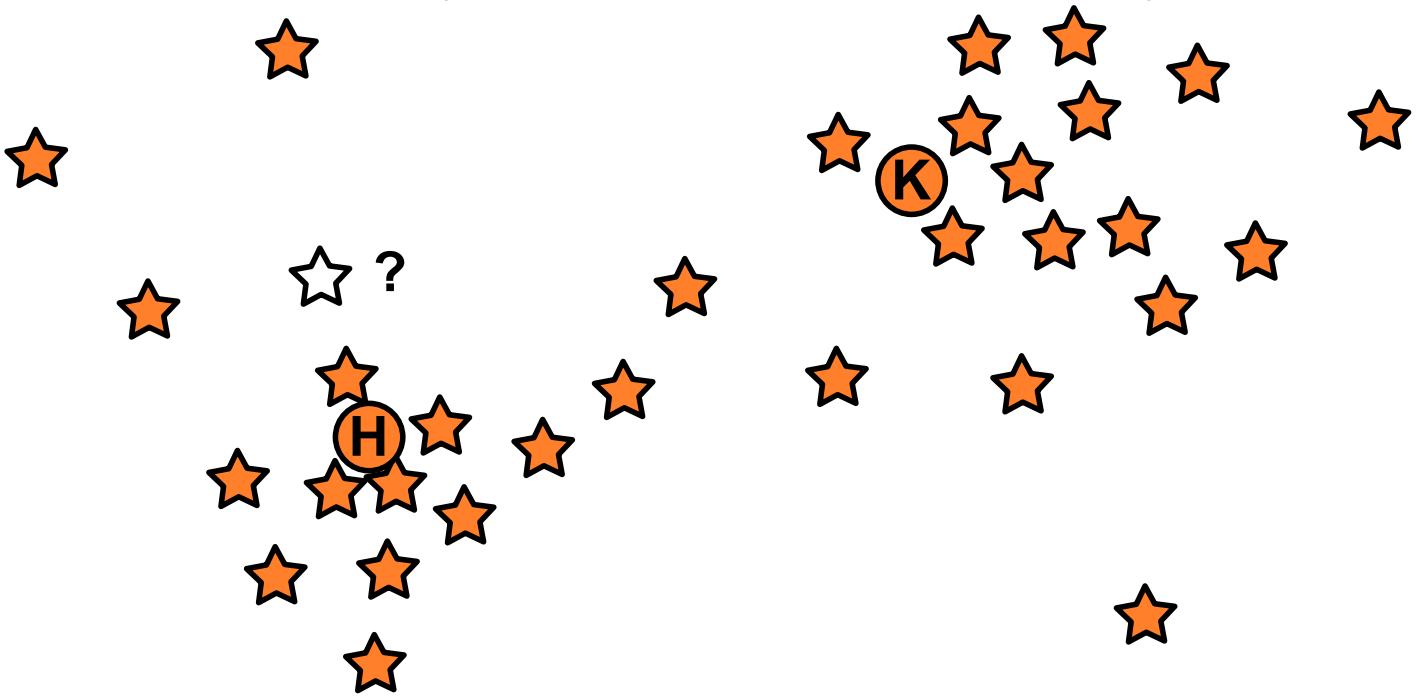
unlabelled labelled

Unüberwachtes Lernen

Läuft oft auf Identifizierungen von Häufungsgebieten (*Clustern*) in den Daten hinaus.

Generelle Idee: Darstellung der Daten durch *Prototypen.*

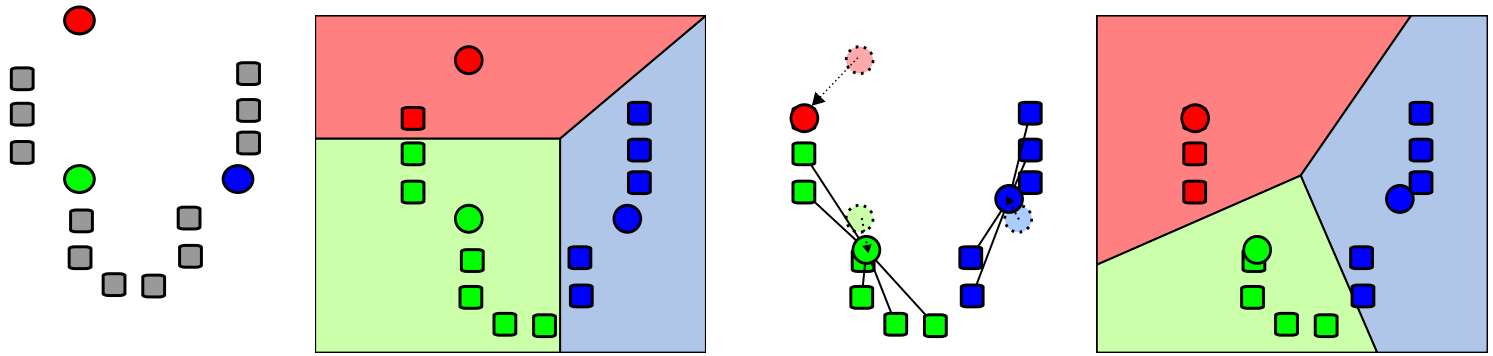
Hier: zwei Prototypen (auch: "Neuronen"). Für einen neuen Punkt: einteilen, je nachdem welchem Prototyp er näher ist.



Wie finden wir die Prototypen?

Eine Möglichkeit: **k-means clustering** (1957). Algorithmus:

1. Setze zufällig k Prototypen $w_1, \ldots, w_k$
2. Betrachte deren *Voronoizellen* V_i (Menge aller Punkte näher an w_i als an w_j, $i \neq j$.
3. Verschiebe w_i in den *Schwerpunkt* der $x \in V_i$
4. Falls sich nix mehr ändert: STOP, sonst gehe zu Schritt 2



Für jedes Verfahren stellen sich die Fragen nach

- Laufzeit
- Heuristische Güte (also experimentell)
- Beweisbare Güte (exakt, bzw stochastisch: mit Wahrsch. 0 (oder ε) wird Fehler gemacht)

Letzteres ist wünschenswert, wenn man bedenkt, dass solche Algorithmen auch für medizinische Diagnosen benutzt werden können, oder selbstfahrende Autos, oder...

Für die mathematische Analyse wichtig: **Kostenfunktion**

Die misst, bezüglich welchen Maßes das Verfahren optimal / nicht optimal ist.

Hier ist bekannt: Fragen wir, für welche k-Clusterung dieser Ausdruck minimal wird:

$$\sum_{i=1}^{k} \sum_{\mathbf{x} \in V_i} \|\mathbf{x} - \boldsymbol{\mu}_i\|^2$$

(w_i Prototypen der Punkte in Cluster V_i. Also $\sum \|x - \mu_i\|^2$ Summe der quadrierten Abstände; vgl Methode der kleinsten Quadrate)

...dann ist die Antwort: für den Output des k-means-clustering.

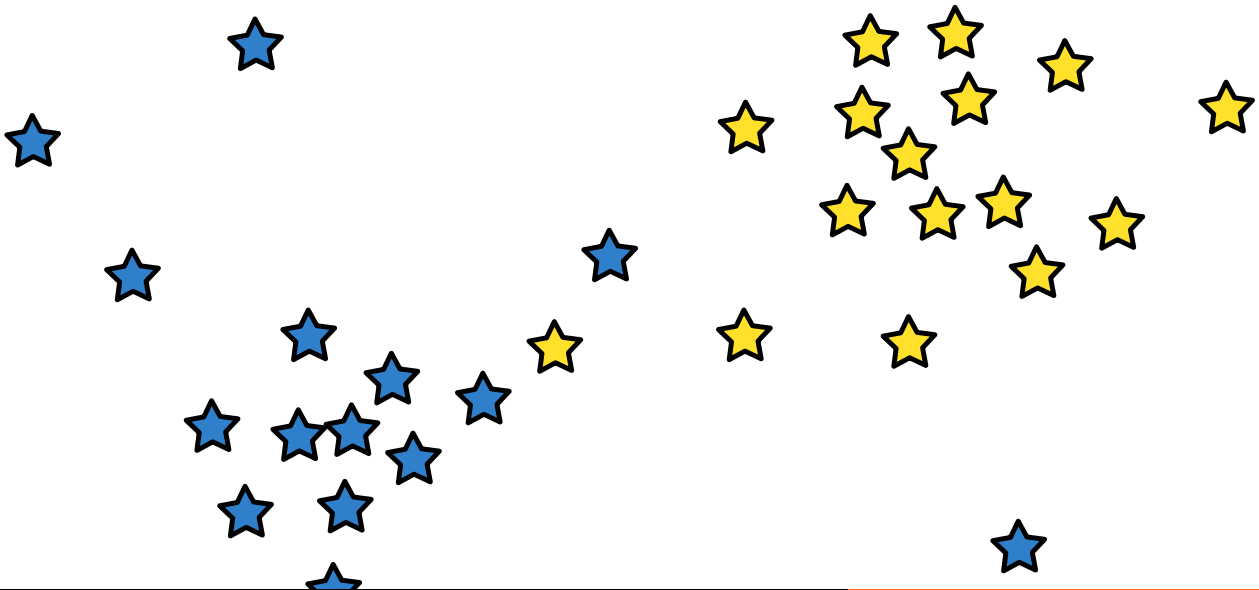
Das ist eine statistische Aussage, da wird über alle möglichen Eingabedatensätze $\{x_1, \ldots, x_m\}$ gemittelt.

Die Kostenfunktion liefert daher auch einen umgekehrten Ansatz:

Wir geben ein Maß für Optimalität vor. Dies minimieren wir. Wenn wir das schaffen, dann liefert das einen Algorithmus, der beweisbar gut ist.

Dies unüberwachte Lernen, also oft: Clustering, ist auch ein Schwerpunkt hier in Bielefeld (Helge Ritter: *Neural Gas*, Barbara Hammer, ...)

Mehr nun am Beispiel von überwachtem Lernen. Dazu Trainingsdatensatz mit Labeln.



Modell: Nun Trainingsdaten
$\{(x_1, y_1), \ldots, (x_m, y_m) \,|\, x_i \in \mathbb{R}^d, y_i \in \{1, \ldots, C\}\}$.

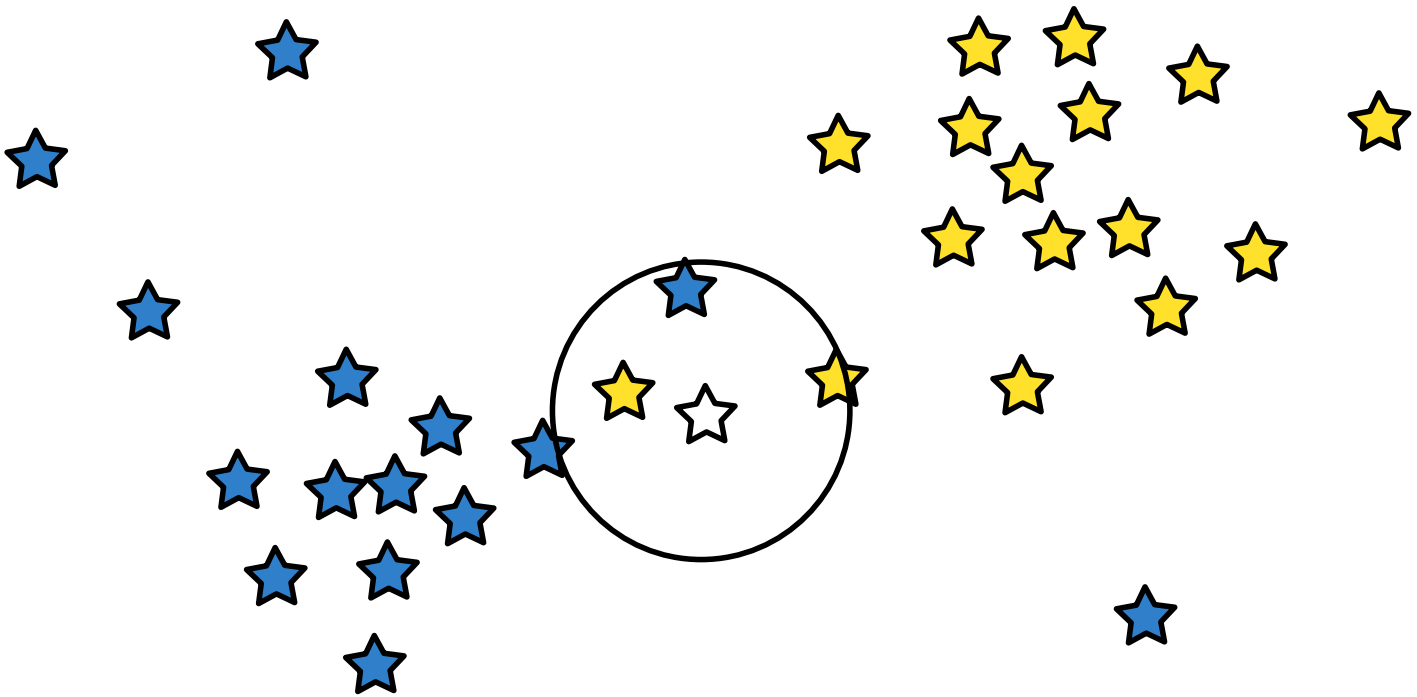
Dabei sind die x_i die Kandidaten (also z.B. Bilder von Katzen und Hunden) und die y_i die Label, also hier: "Katze" oder "Hund".

"Lernen" heißt nun: suche f in einer gewissen Klasse von Funktionen, so dass dies

- optimal auf den Trainingsdaten arbeitet, also $f(x_i) = y_i$ für $i = 1, \ldots, m$.
- Die Fehler-Wahrscheinlichkeit auf *allen* möglichen Eingaben $\{\bar{x}_1, \ldots, \bar{x}_m\}$ minimal wird

k-nearest neighbour

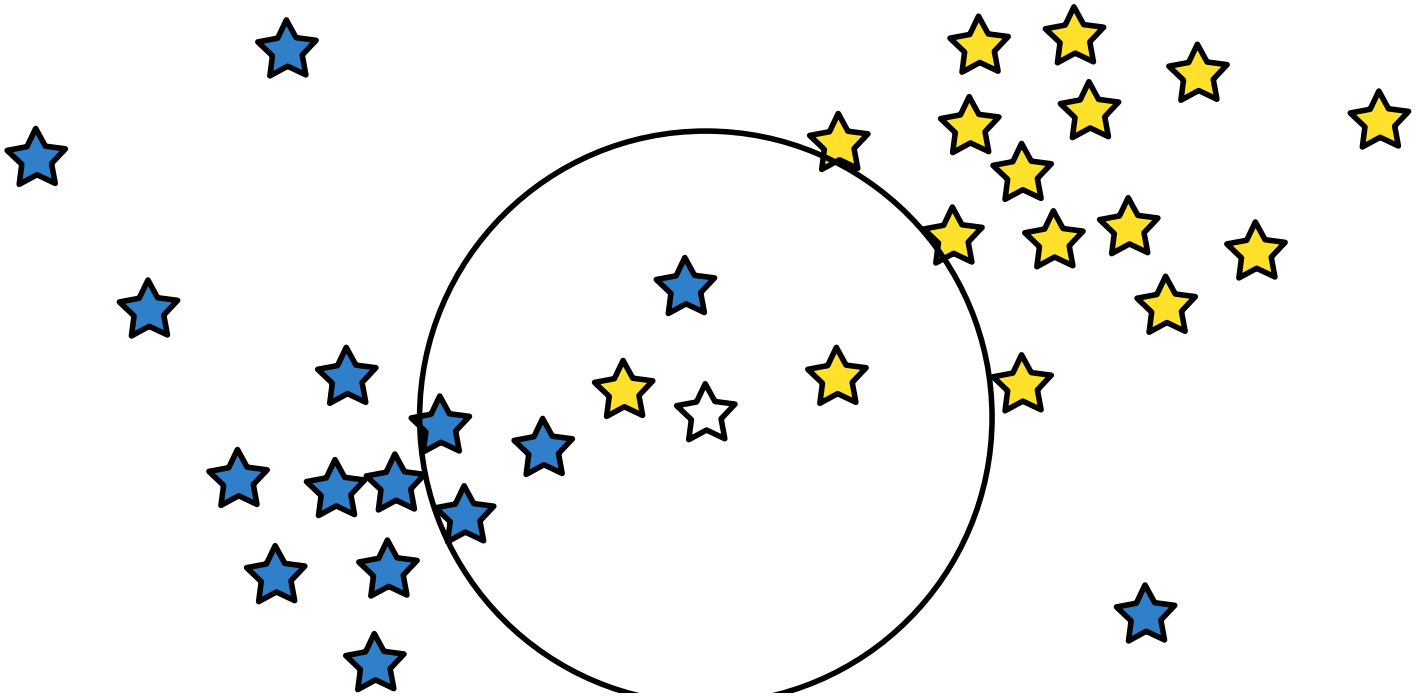
"Überwachtes Lernen" muss man nicht wörtlich nehmen. Kann auch heißen: gar nicht lernen.



k=3: die drei nächsten Nachbarn bestimmen den Typ. Hier: 2:1 für Gelb, also neuer Punkt = gelb.

k-nearest neighbour

Die "gelernten" Daten sind hier keine Prototypen o.Ä., sondern einfach die Trainigsdaten selbst.



k=6: die sechs nächsten Nachbarn bestimmen den Typ. Hier: 4:2 für Blau, also neuer Punkt = blau.

Man kann zeigen: k-nearest neigbour hat keine Kostenfunktion. Dennoch hat man ein Ergebnis (eines der frühesten exakten):

Theorem (Stone 1977)
$\lim_{k_m \to \infty} \frac{1}{m} |\{f_{knn}(x_i) \neq y_i \mid i = 1, \ldots, m\}| = E_{opt}$ *fast sicher*; wobei $\lim_{m \to \infty} \frac{k_m}{m} = 0$. (Also $k_m \in o(m)$)

E_{opt} der optimale Erwartungswert. "Fast sicher" heißt: mit Wahrscheinlichkeit 1. (Interpretation E_{opt} ??)

Generelles Problem: Over-/underfitting, bias-variance dilemma:

Recall Prototypenansatz. Natürlich können wir für m Trainingsdaten $k = m$ Prototypen nehmen. Dann arbeitet das Verfahren auf den Trainingsdaten 100% perfekt. Ist aber bei Testdaten dann wohl aufgeschmissen. (overfitting)

Umgekehrt: zu wenig Prototypen, Feinstruktur in Trainigsdaten kann übersehen werden (underfitting)

Learning Vector Quantisation (LVQ)

Historisch: erst Algorithmus, dann Kostenfunktion. Wir machen's hier anders rum: Prototypen $w_1, \ldots, w_k$. Kostenfunktion

$$\sum_{i=1}^{m} \operatorname{sgn}(d^+ - d^-) \|x_i - w\|^2$$

wobei w der nächste Prototyp, d^+ Abstand zum nächsten korrekten Prototypen, d^- Abstand zum nächsten unkorrekten Prototypen, sgn die Signumfunktion:

$$\operatorname{sgn}(x) = \begin{cases} 1 & \text{falls } x > 0 \\ 0 & \text{falls } x = 0 \\ -1 & \text{falls } x < 0 \end{cases}$$

Algorithmus dazu:

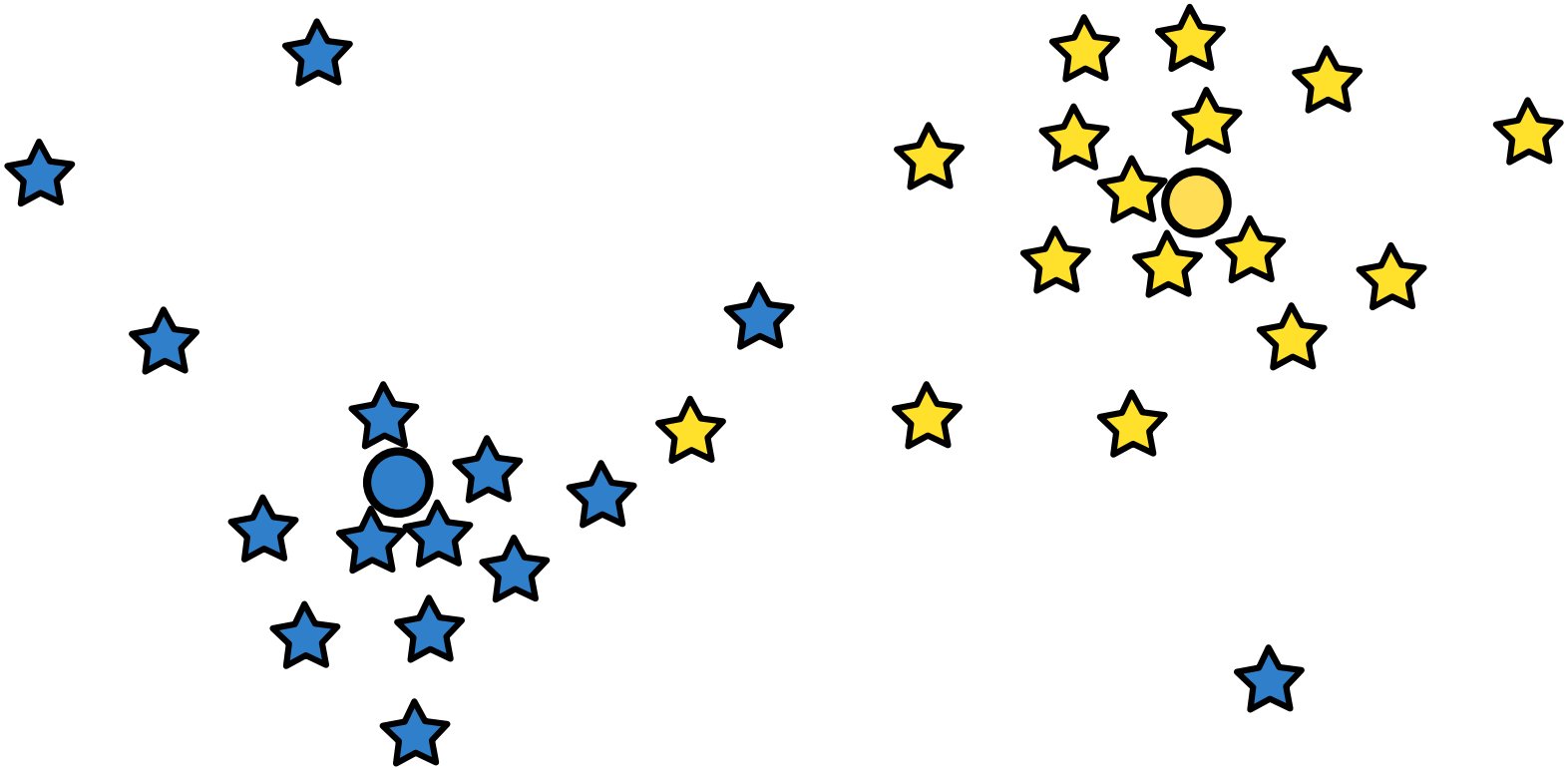
Learning Vector Quantisation (LVQ)

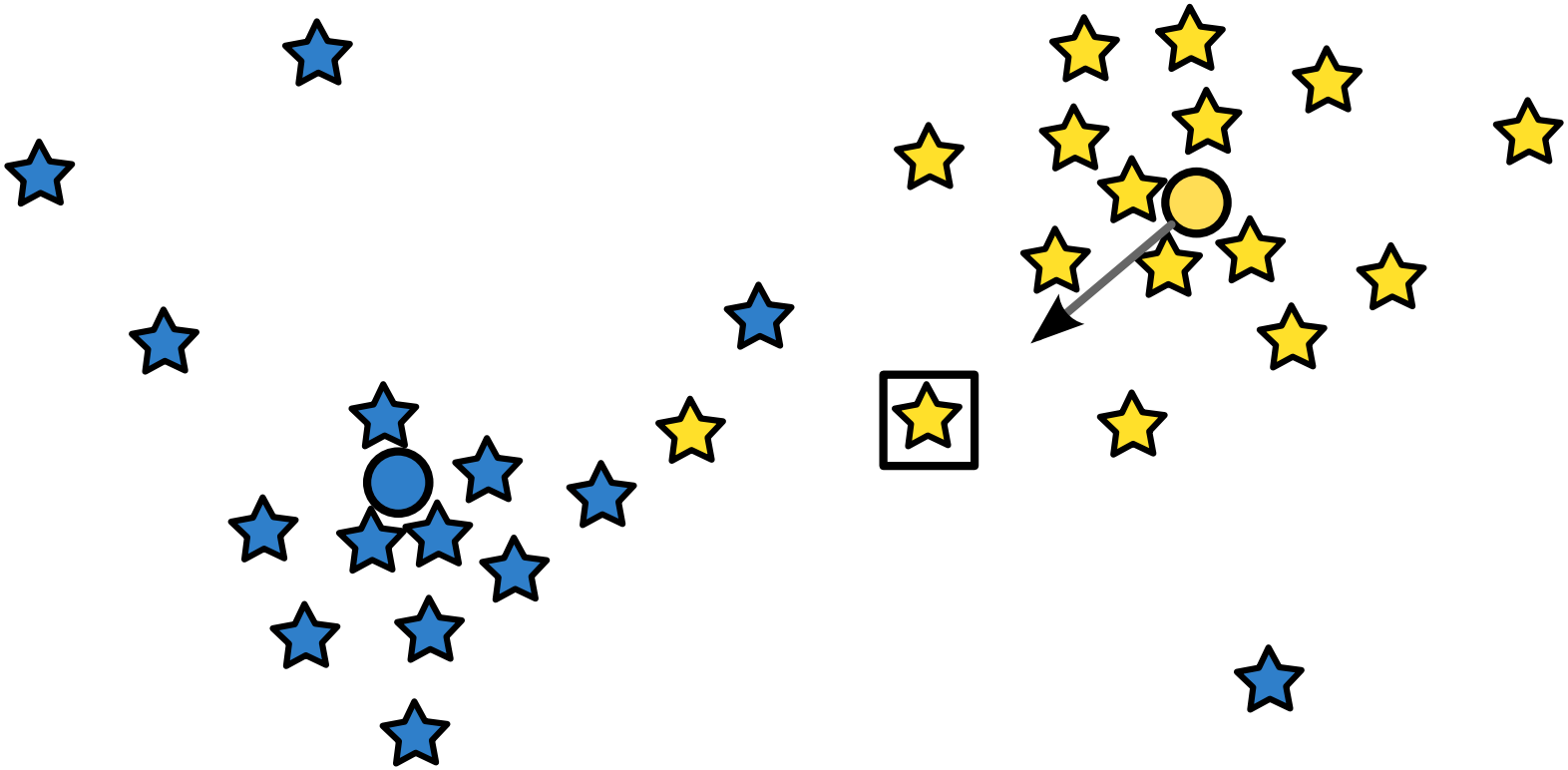
Vorab: Wähle Prototypen $w_1, \ldots, w_k$ (z.B. Schwerpunkte der verschiedenen Klassen)

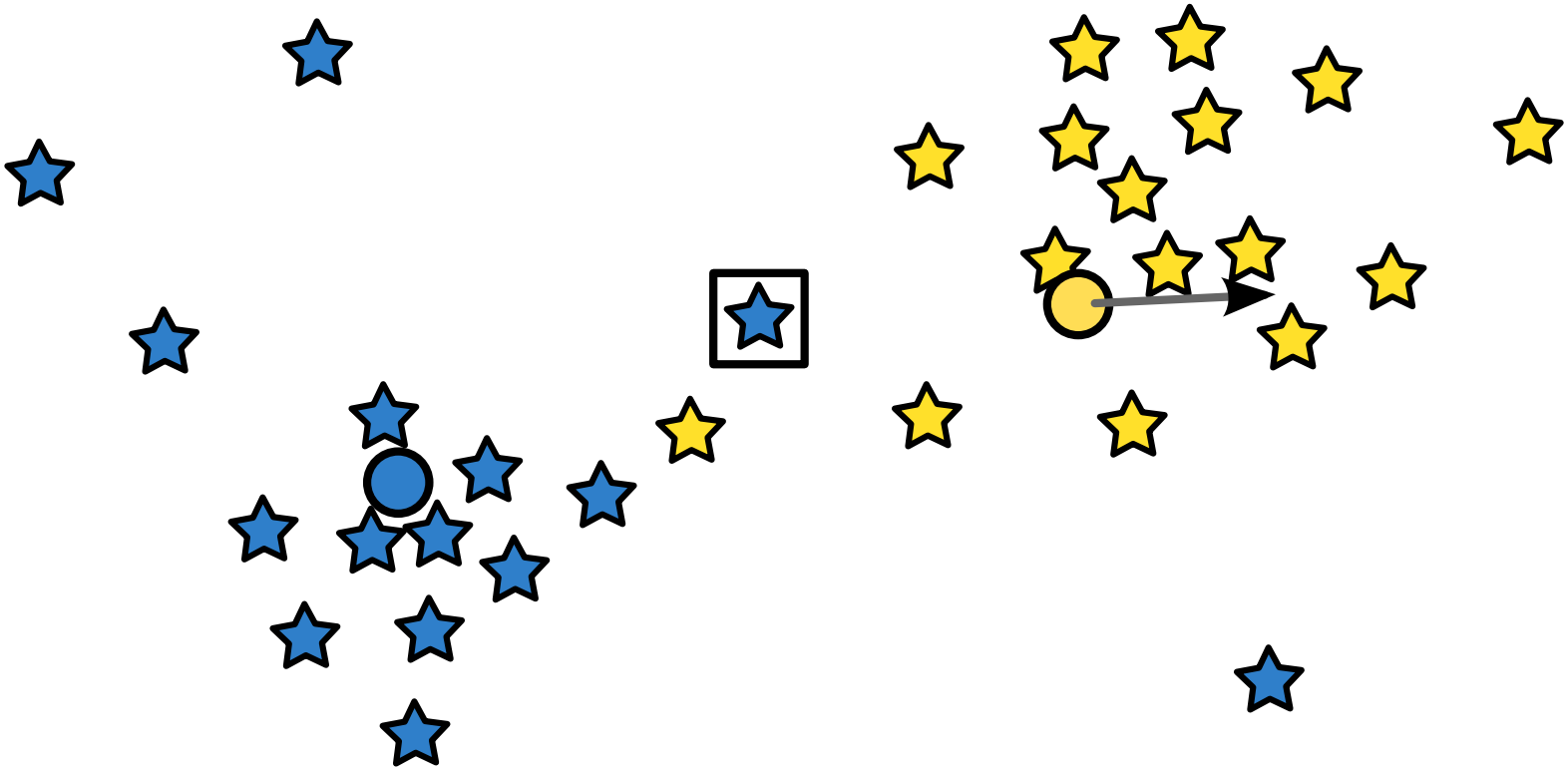
Lernen:

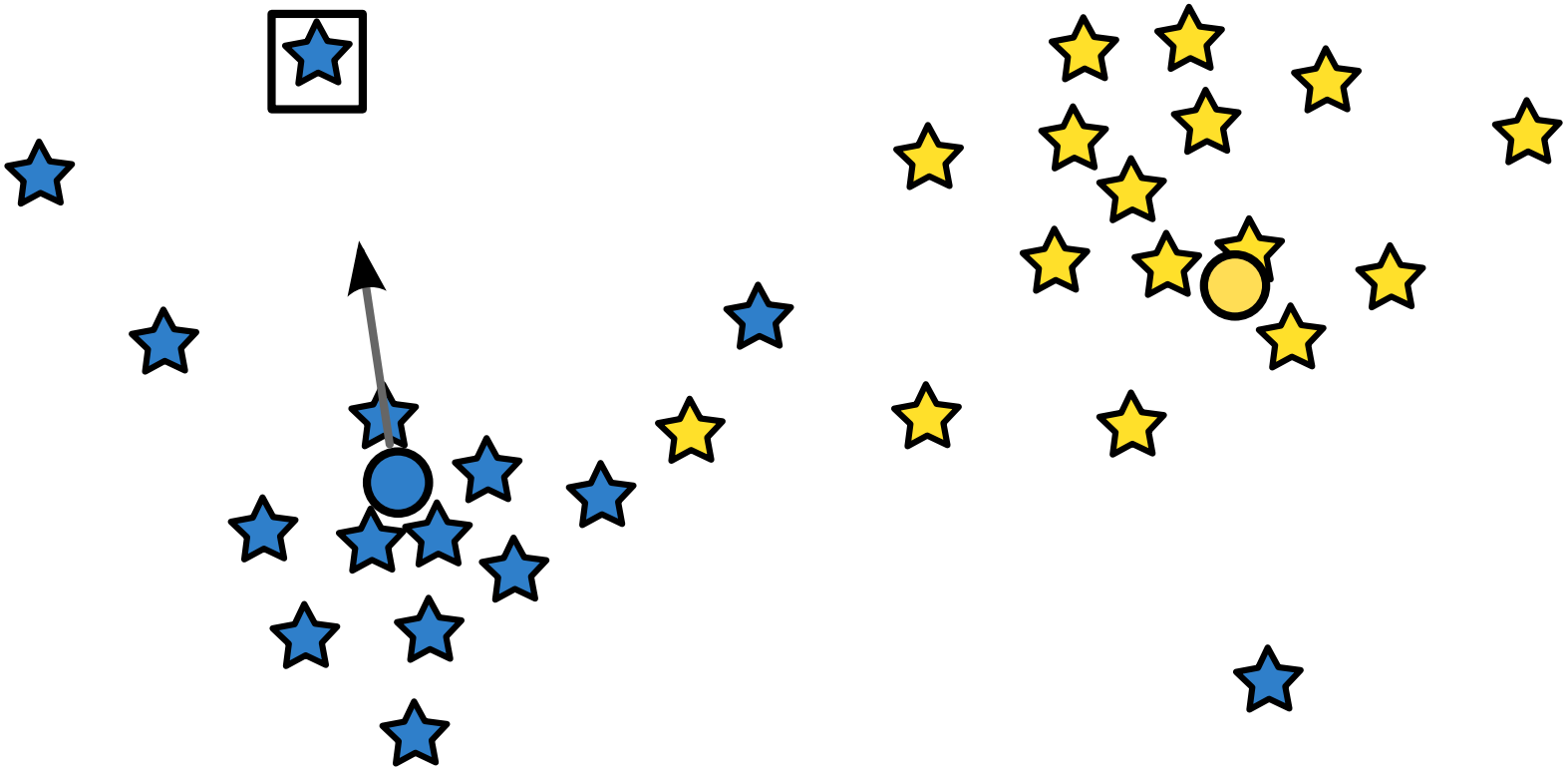
1. Wähle x_i
2. Finde Prototyp w zu x_i mit minimalem Abstand
3. Falls $f(x_i) = f(w)$, schiebe w zu x_i hin:
   $w_{\text{neu}} = w + \eta(x_i - w)$
4. Falls $f(x_i) \neq (w)$, schiebe w von x_i weg.
   $w_{\text{neu}} = w - \eta(x_i - w)$
5. Wiederhole.

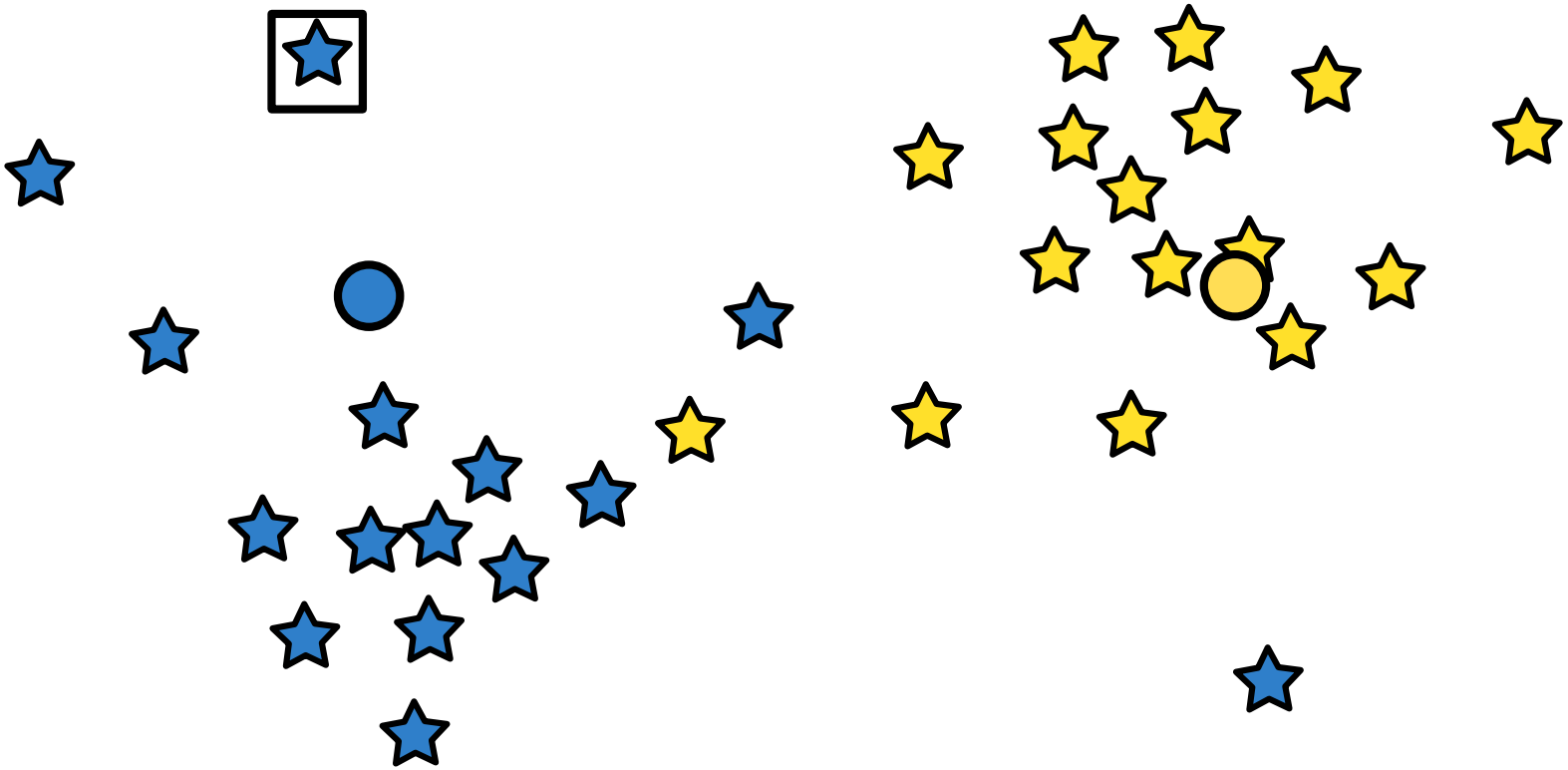
Typische Werte für η etwa $\frac{1}{2}$.
Falls $0 < \eta < 0,1$: w wird wenig verschoben.
Falls $0,9 < \eta < 1$: w wird viel verschoben, fast zu x_i.
Falls $\eta > 1$: w wird über x_i hinaus verschoben.









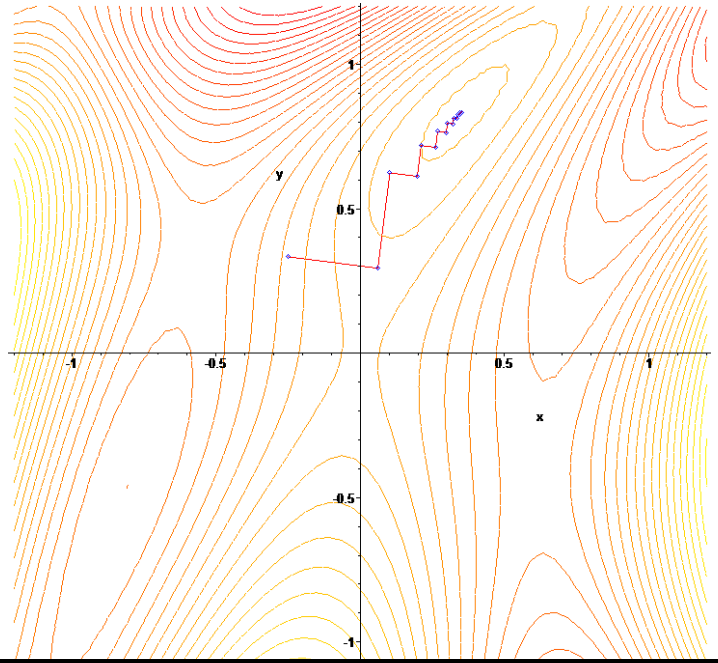


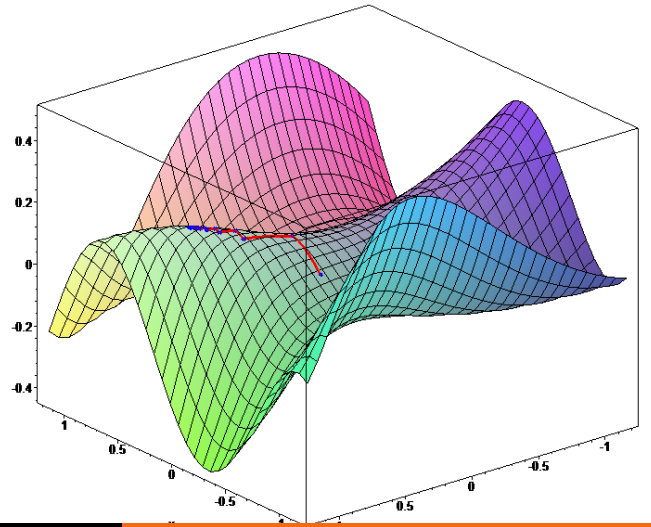
Usw. Man kann zeigen, dass LVQ die obige Kostenfunktion hat.

Ein mächtiges allgemeines Verfahren, um aus der Kostenfunktion einen Lern-Algorithmus zu machen: **stochastischer Gradientenabstieg**.

Kostenfunktion: $K : \mathbb{R}^{kd} \to \mathbb{R}$, $K(w_1, \ldots, w_k)$ ableiten.

Was heißt das? Mathe II: Gradient. Der Gradient zeigt in die Richtung des steilsten Anstiegs. Also gehe in die Gegenrichtung = steilster Abstieg (und zwar solange, bis es *in dieser Richtung* nicht mehr tiefer geht). Finde so ein Minimum.





Eigentlich muss man die Kostenfunktion über alle möglichen Daten mitteln und minimieren. (Daher "Stochastischer" Gradientenabstieg) Praktisch mittelt man eben über die Trainingsdaten.

Die Kostenfunktion oben sieht etwas seltsam aus.
Eigentlich sinnvolle Kostenfunktion könnte sein:

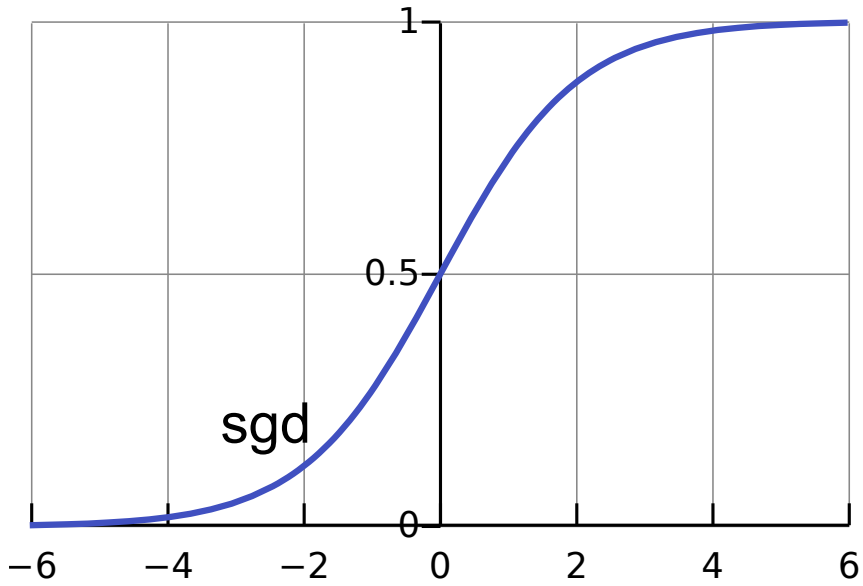
$$\sum_{i=1}^{m} H(d^+ - d^-),\ H(x) = \begin{cases} 1 & \text{falls } x > 0 \\ \frac{1}{2} & \text{falls } x = 0 \\ 0 & \text{falls } x < 0 \end{cases}$$

(also ist H die Heavisidefunktion, vgl. Vorlesung 11)

Problem bei beiden: Signumfunktion und Heavisidefunktion haben

1. viele Plateaus (Gebiete mit Steigung 0, Gradient zeigt nix)
2. Sprungstellen, dort lassen sie sich nicht ableiten (außer... vgl. Vorlesung 11: verallgemeinerte Funktionen).

Trick: Funktion glätten. Statt H nimm *Sigmoide* sgd:



Stochastischer Gradientenabstieg angewandt auf Kostenfunktion

$$\sum_{i=1}^{m} \text{sgd}\left(\frac{d^+ - d^-}{d^+ + d^-}\right)$$

liefert **Generalised LVQ (GLVQ)** (Sato, Yamada, *NIPS* 1995; Hammer,Villmann, *Neural Networks* 2002)

Um Güte exakt abzuschätzen, modifiziere den empirischen Erwartungswert

$$\hat{E}_m(f) = \frac{1}{m} \sum_{i=1}^{m} 1_{y_i \neq f(x_i)}$$

(der Einfachheit halber: $y_1 = -1, y_2 = 1$; $d^- = d(x_i, w_{-1})$, $d^+ = d(x_i, w_1)$) zu

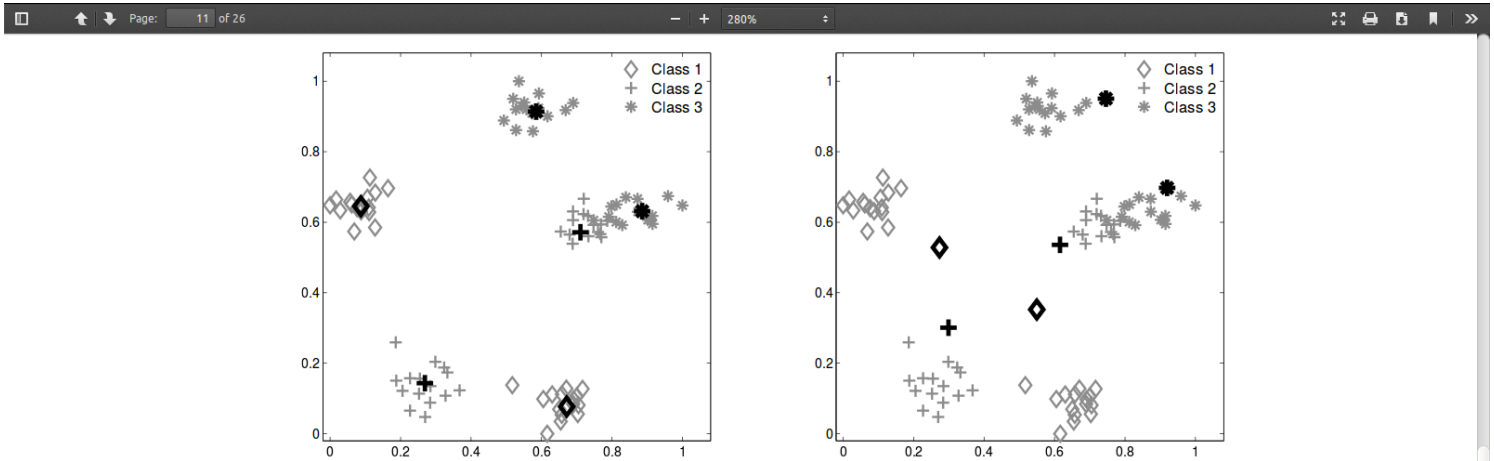
$$\hat{E}_m^\rho(f) = \frac{1}{m} \sum_{i=1}^{m} L(y_i \cdot (d^- - d^+)), \quad L(x) = \begin{cases} 0 & \text{falls } x > \rho \\ 1 - \frac{x}{\rho} & \text{falls } 0 \leq x \leq \rho \\ 1 & \text{falls } x < 0 \end{cases}$$

Damit kann man zeigen: Mit Wahrscheinlichkeit ε ist

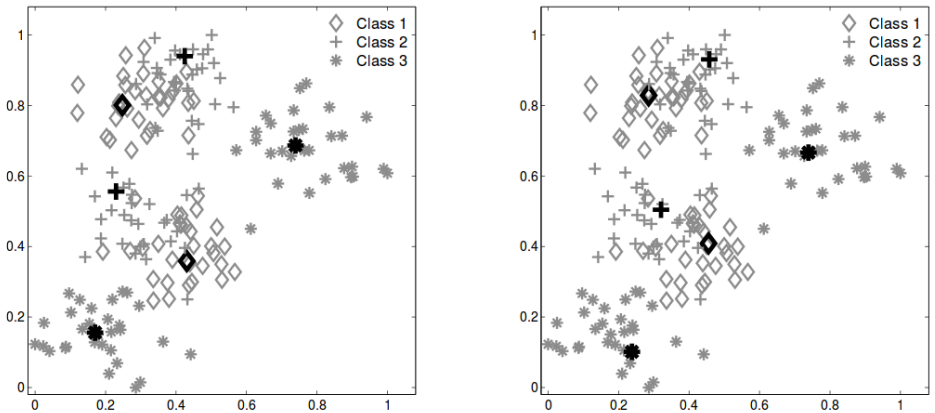
$$E(f) \leq \hat{E}_m^\rho(f) + \frac{1}{\sqrt{m}} O\left(\frac{k^{\frac{3}{2}} B^3}{\rho} + \frac{\sqrt{\ln \frac{1}{\varepsilon}}}{\min\{1, \rho\}} \right)$$

(Schneider, Biehl, Hammer, *Neural Computation* 2009)

Neben dem theoretischen (aber durchaus praktischen) Ergebnis kann man nun auch experimentieren: Algorithmus auf Testdatensätze loslassen. (Es gibt oft standardisierte Testdatensätze, der Vergleichbarkeit wegen)



(a) Data set 1. Left: GLVQ prototypes. Right: RSLVQ protoypes



(b) Data set 2. Left: GLVQ prototypes. Right: RSLVQ prototypes

Nun können weitere Ideen eingebaut werden. Und in Theorie und Experiment geprüft werden.

Hier nur noch eine (die gut ist): Bisher arbeiteten wir mit dem euklidischen (also normalen) Abstand. Das ist manchmal naiv:

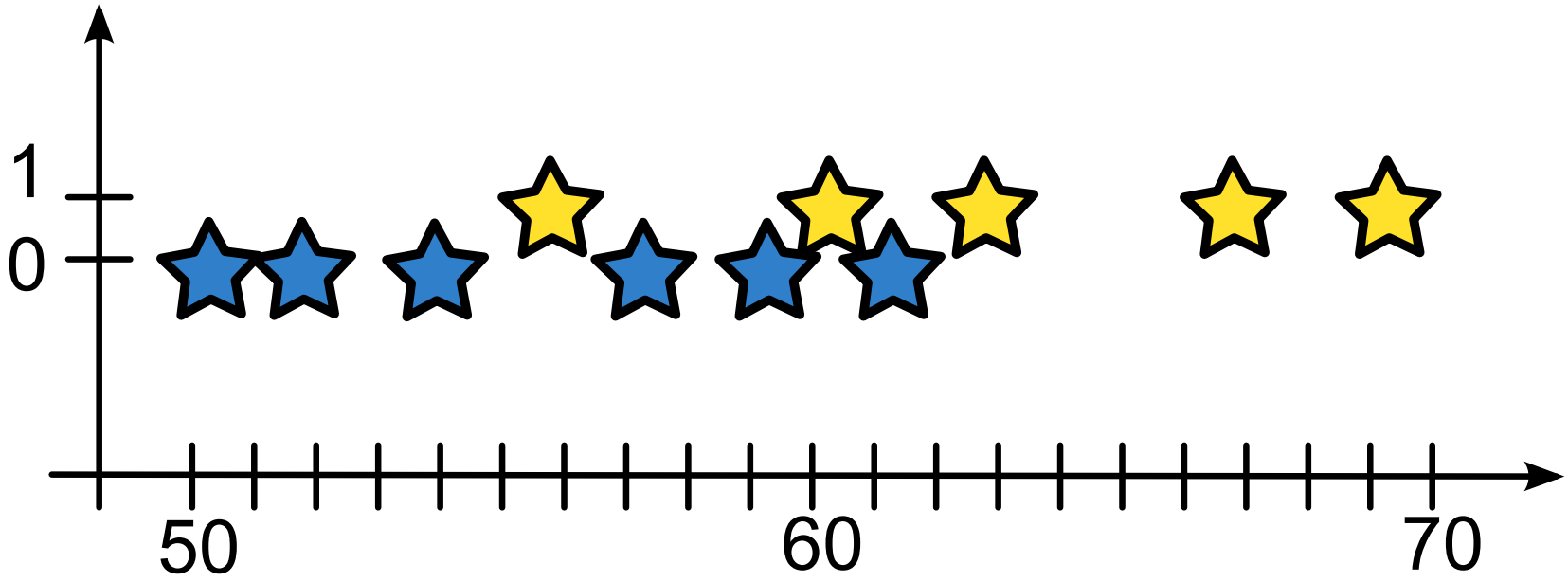
Katze vs Hund als Paramater: (Fell?, Bellt?, Farbe, Länge,...)

- Hund z.B.: (1, 1, bc9a81, 110 cm...)
- Katze z.B. (1, 0, ccb4a1, 30 cm...)

"Fell" ist offenbar ein für die Unterscheidung irrelevanter Parameter. "Bellt" ein sehr guter. Aber:

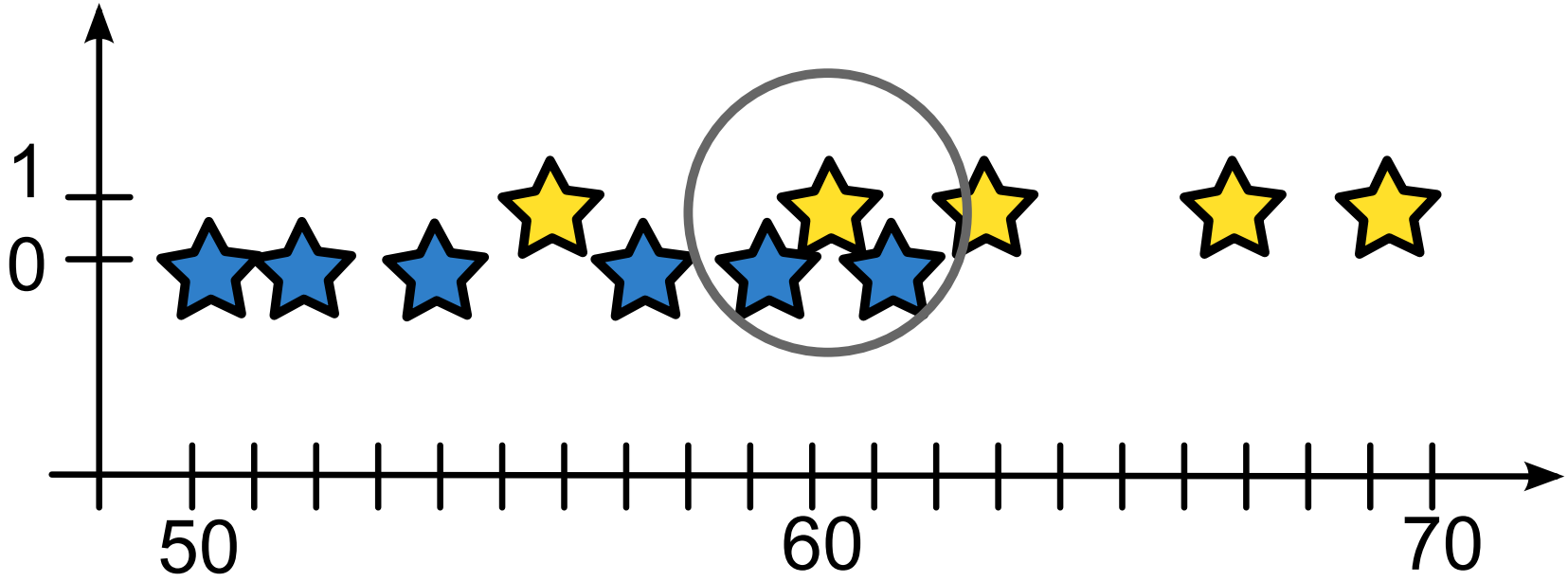
- Der Unterschied "Bellt" - "Bellt nicht" liefert Abstand 1.
- Der Unterschied 60cm - 70 cm liefert Abstand 10.

Generalised Metric LVQ (GMLVQ)



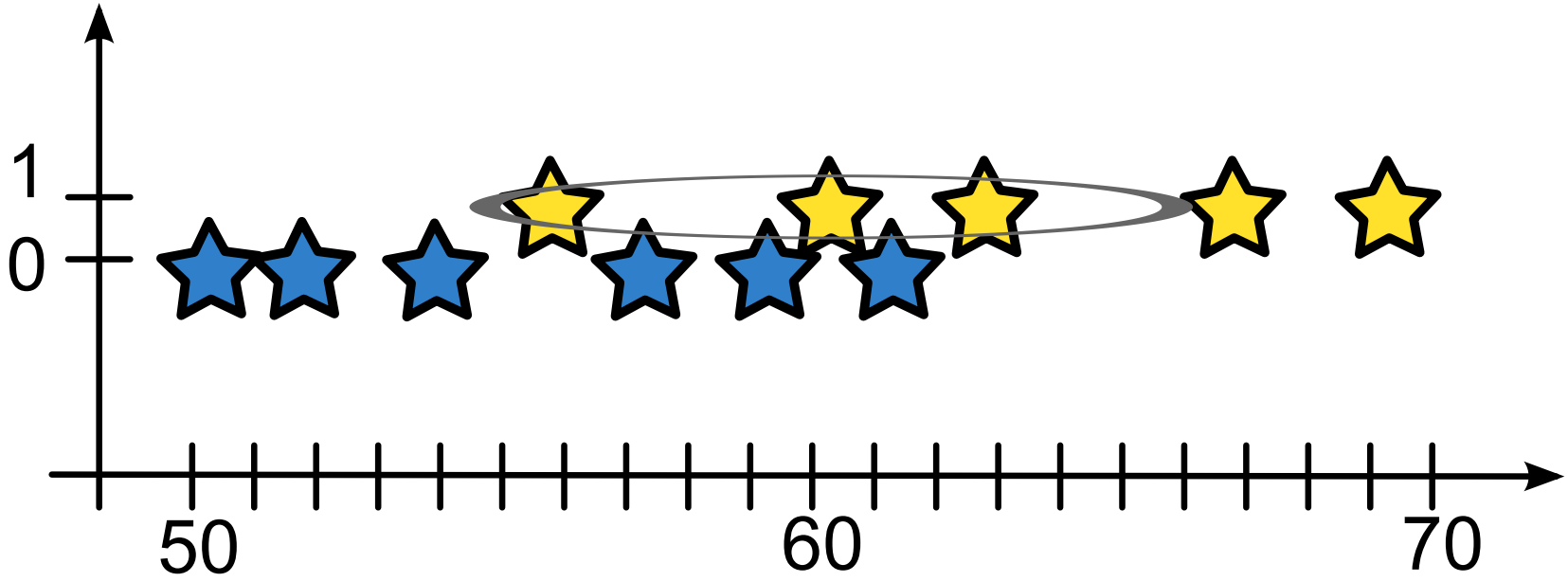
Waagerechte Achse: Länge in cm.
Senkrechte Achse: Bellt = 1, bellt nicht = 0.

Generalised Metric LVQ (GMLVQ)



Normaler (euklidischer) Abstand hier ungut.

Generalised Metric LVQ (GMLVQ)



Abstand senkrecht teurer machen, Abstand waagerecht billiger.

Recall: euklidischer Abstand:

$x, y \in \mathbb{R}^d$: $\quad d(x,y) = \|x-y\|$, wobei $\|x\|^2 = x_1^2+x_2^2+\cdots+x_d^2 = x^T \cdot x$

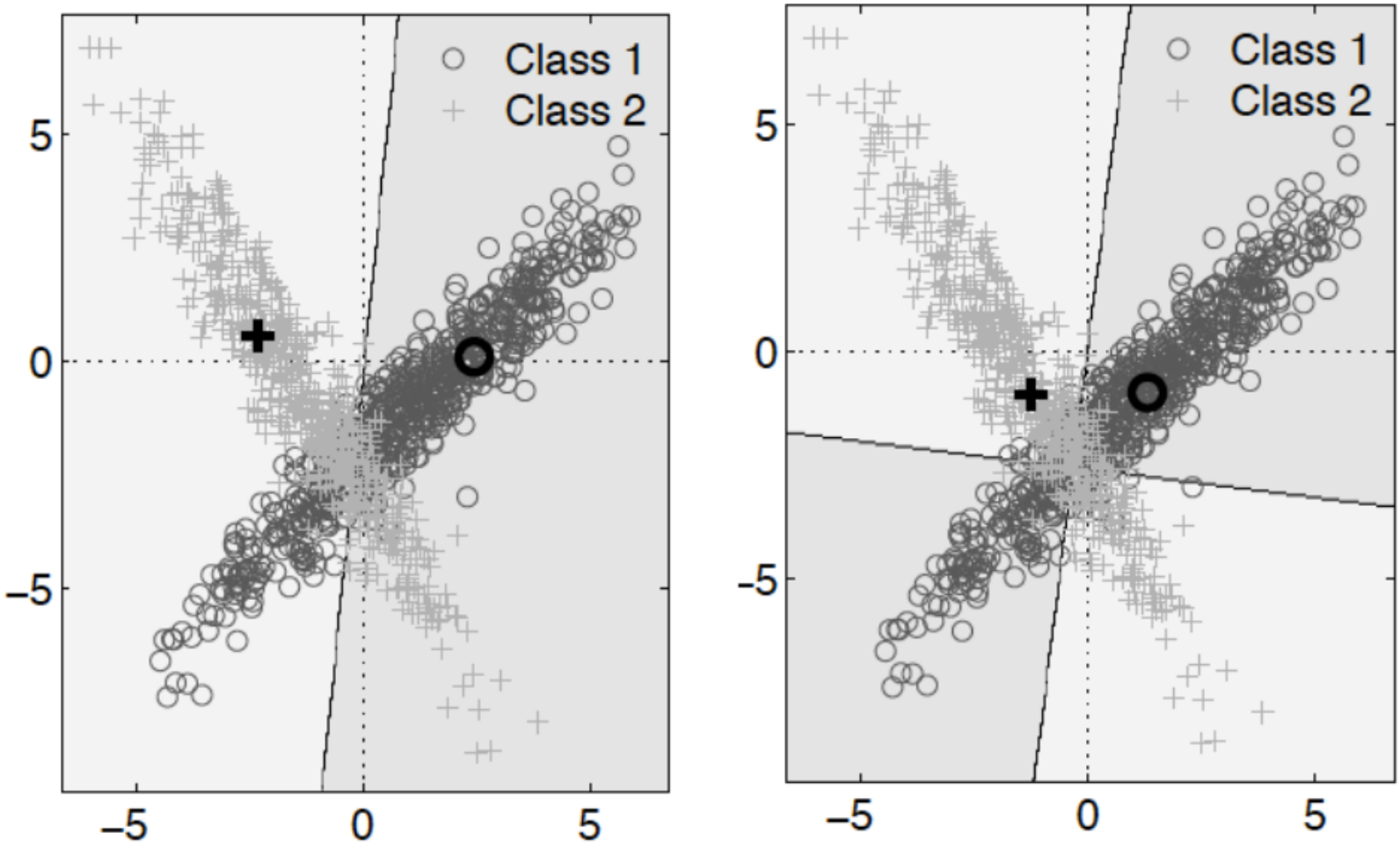
Nimm jetzt als Abstand nicht $x^T \cdot x$, sondern $x^T A x$.

(A muss positiv definit sein, sonst negative Abstände möglich)

für Bsp. oben z.B. $A = \begin{pmatrix} 1 & 0 \\ 0 & 50 \end{pmatrix}$.

Allgemeine Idee: Lerne (auch) A. Liefert **GMLVQ**. Varianten:

- Lerne einmal Protoypen, lerne bei jeder neuen Anwendung A. (Interessante Anwendung: Unterarmprothesen)
- Lerne für jede Klasse S_i ein eigenes A_i.
- Reduziere um unwesentliche Dimensionen
- ...



Links: Für jede Klasse eine eigene Metrik (ein eigenes A_i).
Rechts: Für alle Klassen dieselbe Metrik.
Mehr in Vorlesungen etc von Barbara Hammer und Helge Ritter und Co

Ende.

Viele Erfolg bei den Prüfungen!

Schöne Ferien!