



Tippverhaltensbiometrie mit KI-Methoden

Grundlagen

- Zwei Phasen: Enrolment und Authentifizierung

- **Enrolment:** Das System „erlernt“ das Tippverhalten einer Person

 - Dazu muss die Person einen bestimmten Satz mit ca. 50 Zeichen **9-mal (!)** tippen

 - Auf dieser Basis berechnet das System ein **Template** (oder Profil) des Benutzers

- **Authentifizierung:** Das System entscheidet, ob eine abgegebene Tippprobe zu einem vorhandenen Template passt.

 - Die Person tippt den vorgegebenen Satz einmal









 - Das System vergleicht den getippten Satz mit dem Template der Person, errechnet dazu einen **Matchscore** und vergleicht den erreichten Score mit einem **Threshold**

↳ Daten falsch, da evtl. Latenzen auftreten durch parallel processing oder durch Verwendung untersch. Browser

Aufzeichnung der Tippproben

↳ einer kann Shift-Tasten unterscheiden, der andere nicht

- Auf Client-Seite werden die **KeyPressed-** und **KeyReleased-Ereignisse** mit zugehörigen Timestamps aufgezeichnet:

								
KeyPressed	1	0	1	0	1	0	1	0
KeyCode	65	65	8	8	66	66	32	32
Rel. Timestamp	0	16	32	80	16	15	32	80

- Es entsteht ein **RawSample**
- Beschränkungen durch Browser und Betriebssystem, z. B. Auflösung
- Im Browser liefert ein [Adobe Flash-Movie] die zuverlässigsten Ergebnisse:

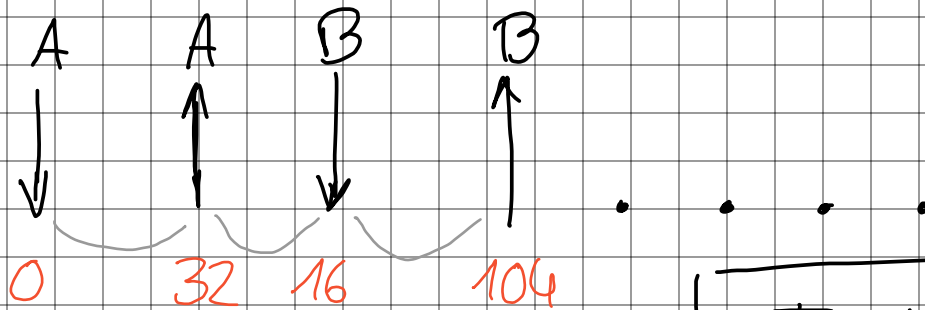


Bitte tippen Sie den folgenden Satz:

Ich bin der Meinung, die richtige Antwort lautet:

Besser: JavaScript

Rohdaten



Rohdatenstring: 65 d 000 | 65 u 032 | ..

↳ muss daraus Originaltext
erstellbar sein?

Features 1. Ordnung // Ebene
↳ direkt aus Eingabe ablesbar
"ohne Magie"

Feature Extraktion: Überholungen
von Tasten

Features 2. Ebene: Ergeben sich aus
Features 1. Ebene

↳ Geschwindigkeit zw. Tasten je nach Distanz

a	b	x	y	z
10	16	110	98	10
11	15	105	99	null
10	16	2000	98	12
9	14	103	97	9

Bsp Tabelle: Ausreißer mit Grubbs finden
↳ Hypothesentests mit α -Signi-
fikanzniveau

→ Ausreißer durch Imputationsverfahren ersetzen

Imputationsverfahren: Mittelwert, Median, gemischter Mittelwert, Einsetzen durch besten Nachbarn

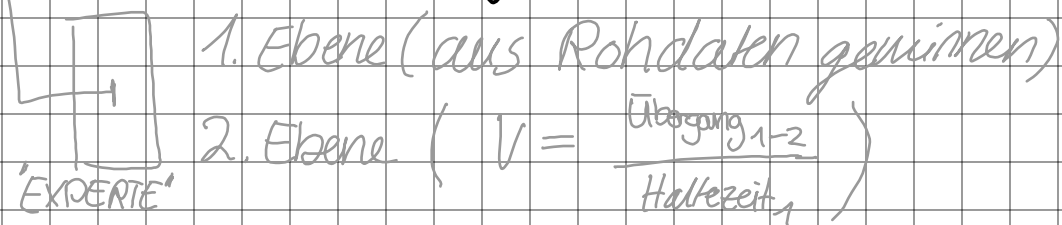
Ablauf:

1) Daten erfassen / hinterfragen ↯

↳ Daten normieren / "runden"
(auf den schlechtesten Windows)

2) Datenqualität (automatisch)
prüfen

3) Features festlegen



4) Ausreißer entfernen, z.B. GRUBBS Test

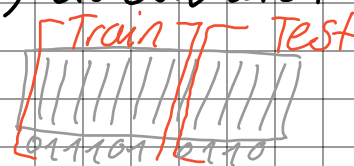
5) Fehlende Werte ersetzen (z.B. Imputation)

6) Features normieren $[0, 1]$ / $[-1, 1]$

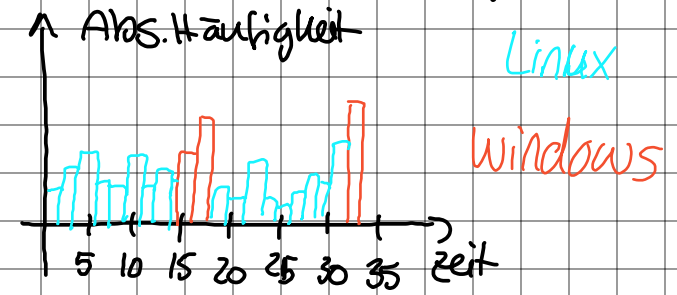
7) Quelldaten inkl. Label & teile auf: - Testset

- Trainingsset

untersch.
Aufteilung
v. Test & Train
Daten
"Kreuz-
Validation"



Bsp Abtippen Patientennamen
oder Tippverhalten



=> Taktung des PCs beeinflusst
Ergebnisse ↯

03.02

NaN als "null"

kann schlecht sein, da
linear, aber Notlösung

8) Trainieren mit Trainingsdaten

1 = krank

Test Daten 0 = gesund

9) Testen mit Testdaten

Overfitting → erkennt nur Trainingsdaten

Schritt 8 und 9 mehrmals wiederholen

Gewichtung von Fehler 1. Art: 66% // 70%

8 Fehler 2. Art: 34% // 30%

Soll	1 1 1 0 0 1	
Erkannt	1 1 0 0 1 1	2/6

Fehler 1. Art: krank erkannt, obwohl gesund 1/2

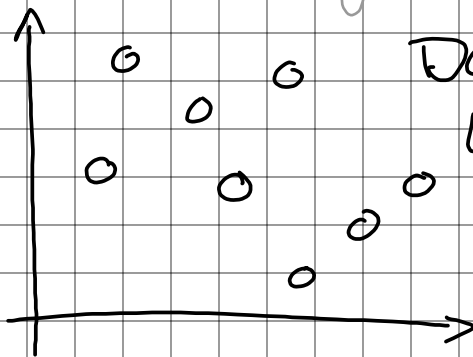
Fehler 2. Art: gesund, obwohl krank 1/2

FRR ↑
Fehler-Rückweisung-Rate

FAR ↑
Fehler-Akzeptanz-Rate

⇒ man müsste alle Konstellationen von C & y probieren!

Tipp: nicht in ganzen Zahlen denken, sondern in e-Potenzen: e^1, e^2, \dots



DOE-Pattern → funktioniert sehr gut

↳ Idee von HP

Zahlen normieren,
damit SVM gut
damit arbeiten kann

Übersicht über den Enrolment-Prozess (1/2)

9 RawSamples *von dem Nutzer den man erkennen will*

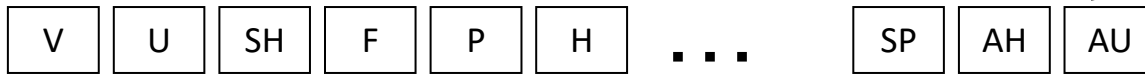


Vorverarbeitung

Alignment & Tippfehlerprüfung

RawSample Qualitätsprüfung

Merkmale (Features) berechnen



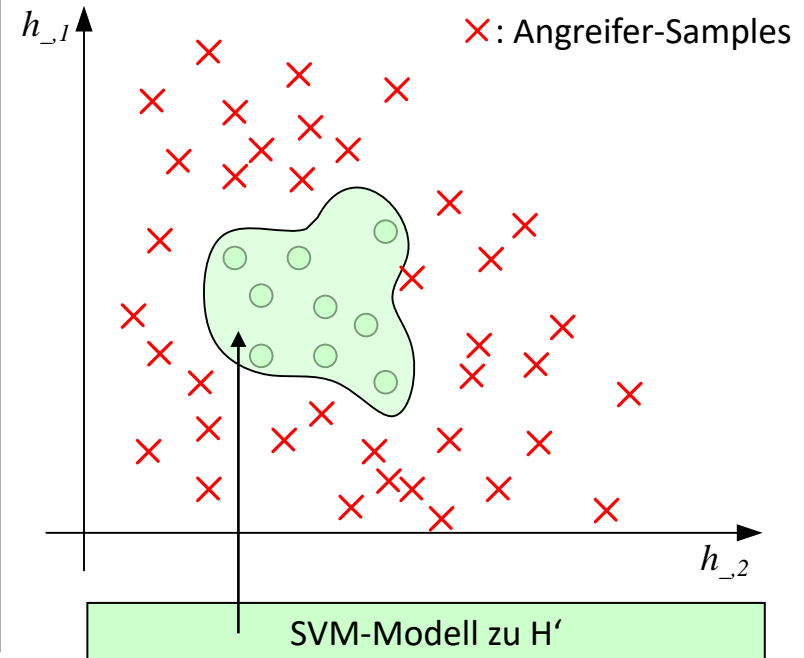
Ausreißer entfernen

Fehlende Werte ersetzen

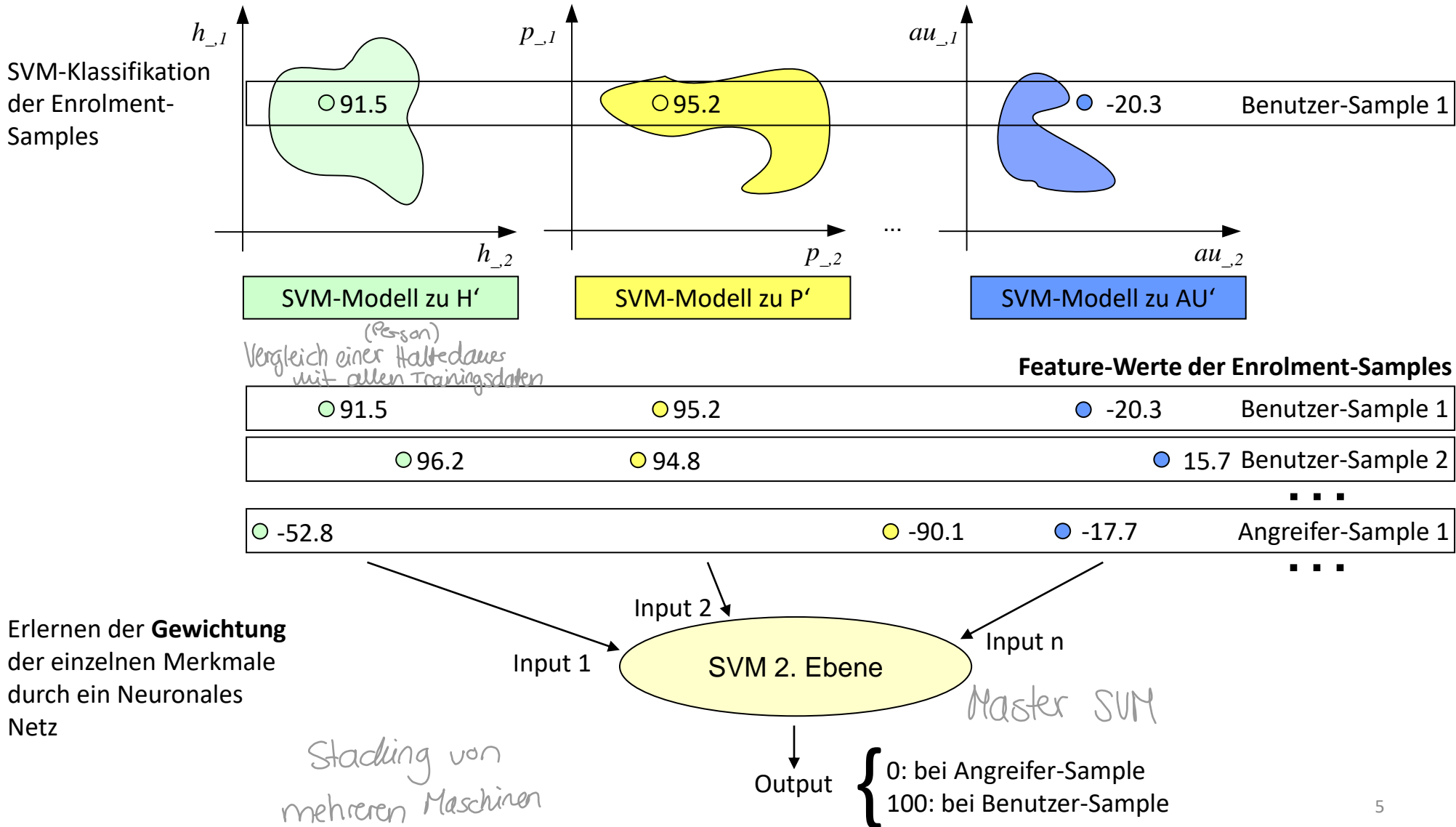


Benutzer-Samples

$h_{1,1}$	$h_{1,2}$	$h_{1,3}$...	$h_{1,50}$	Sample 1	○
$h_{2,1}$	$h_{2,2}$	$h_{2,3}$...	$h_{1,50}$	Sample 2	○
...
$h_{9,1}$	$h_{9,2}$	$h_{9,3}$...	$h_{9,50}$	Sample 9	○



Übersicht über den Enrolment-Prozess (2/2)



SVM-Demo

options:

-s svm_type : set type of SVM (default 0)

0 -- C-SVC

1 -- nu-SVC

2 -- one-class SVM

3 -- epsilon-SVR

4 -- nu-SVR

-t kernel_type : set type of kernel function (default 2)

0 -- linear: $u \cdot v$

1 -- polynomial: $(\gamma u \cdot v + \text{coef0})^{\text{degree}}$

2 -- radial basis function: $\exp(-\gamma |u - v|^2)$

3 -- sigmoid: $\tanh(\gamma u \cdot v + \text{coef0})$

-d degree : set degree in kernel function (default 3)

-g gamma : set gamma in kernel function (default $1/k$)

-r coef0 : set coef0 in kernel function (default 0)

-c cost : set the parameter C of C-SVC, epsilon-SVR, and nu-SVR (default 1)

-n nu : set the parameter nu of nu-SVC, one-class SVM, and nu-SVR (default 0.5)

-p epsilon : set the epsilon in loss function of epsilon-SVR (default 0.1)

-m cachesize : set cache memory size in MB (default 100)

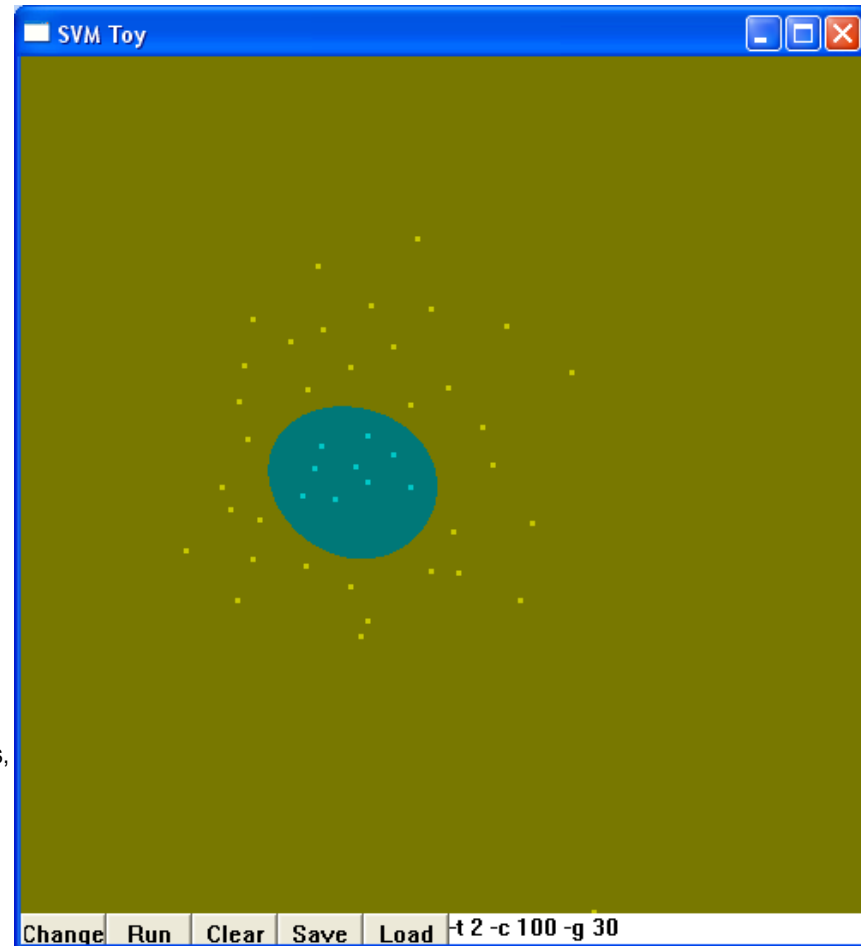
-e epsilon : set tolerance of termination criterion (default 0.001)

-h shrinking: whether to use the shrinking heuristics, 0 or 1 (default 1)

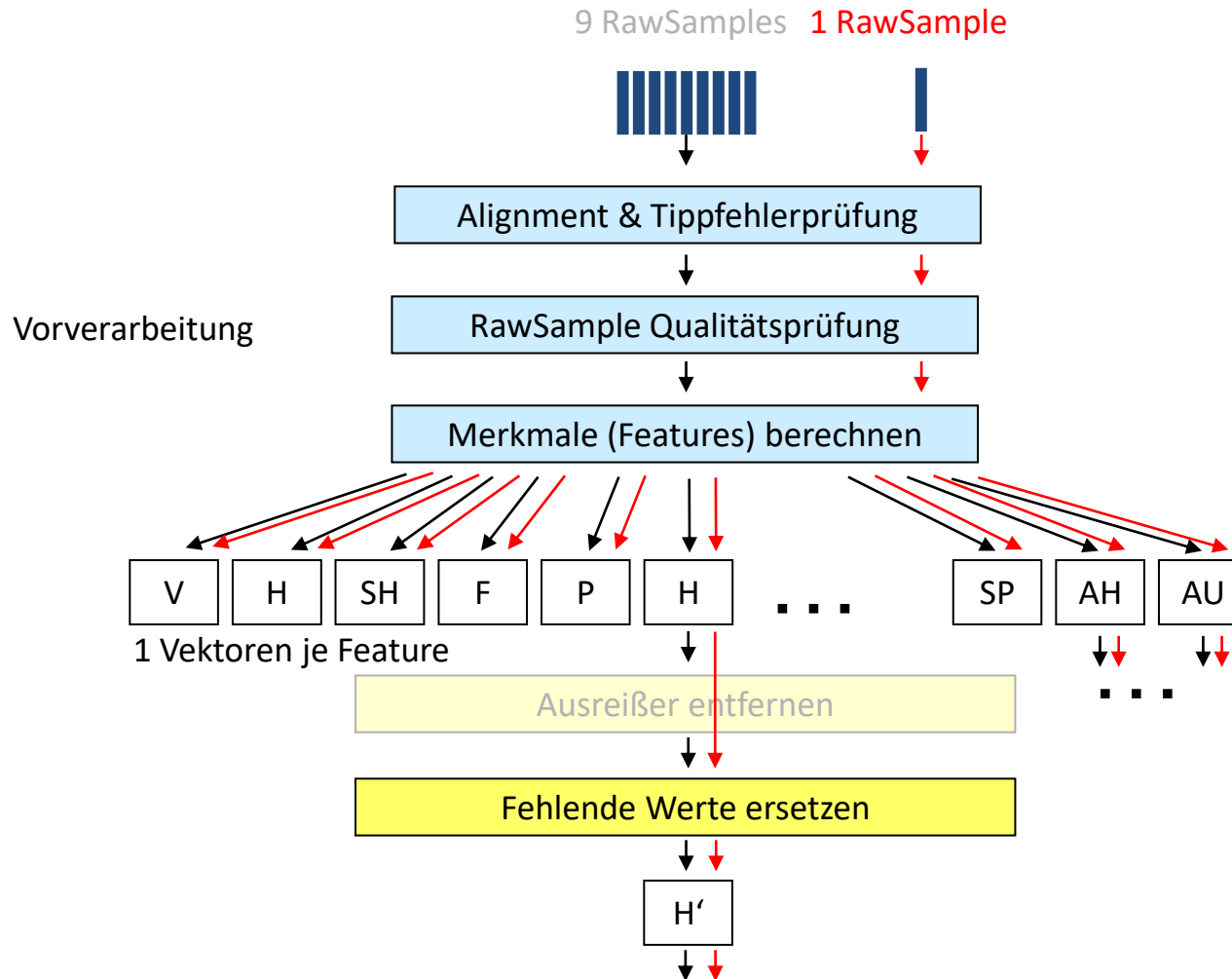
-b probability_estimates: whether to train a SVC or SVR model for probability estimates,

-wi weight: set the parameter C of class i to $\text{weight} \cdot C$, for C-SVC (default 1)

-t 2 -c 100 -g 30

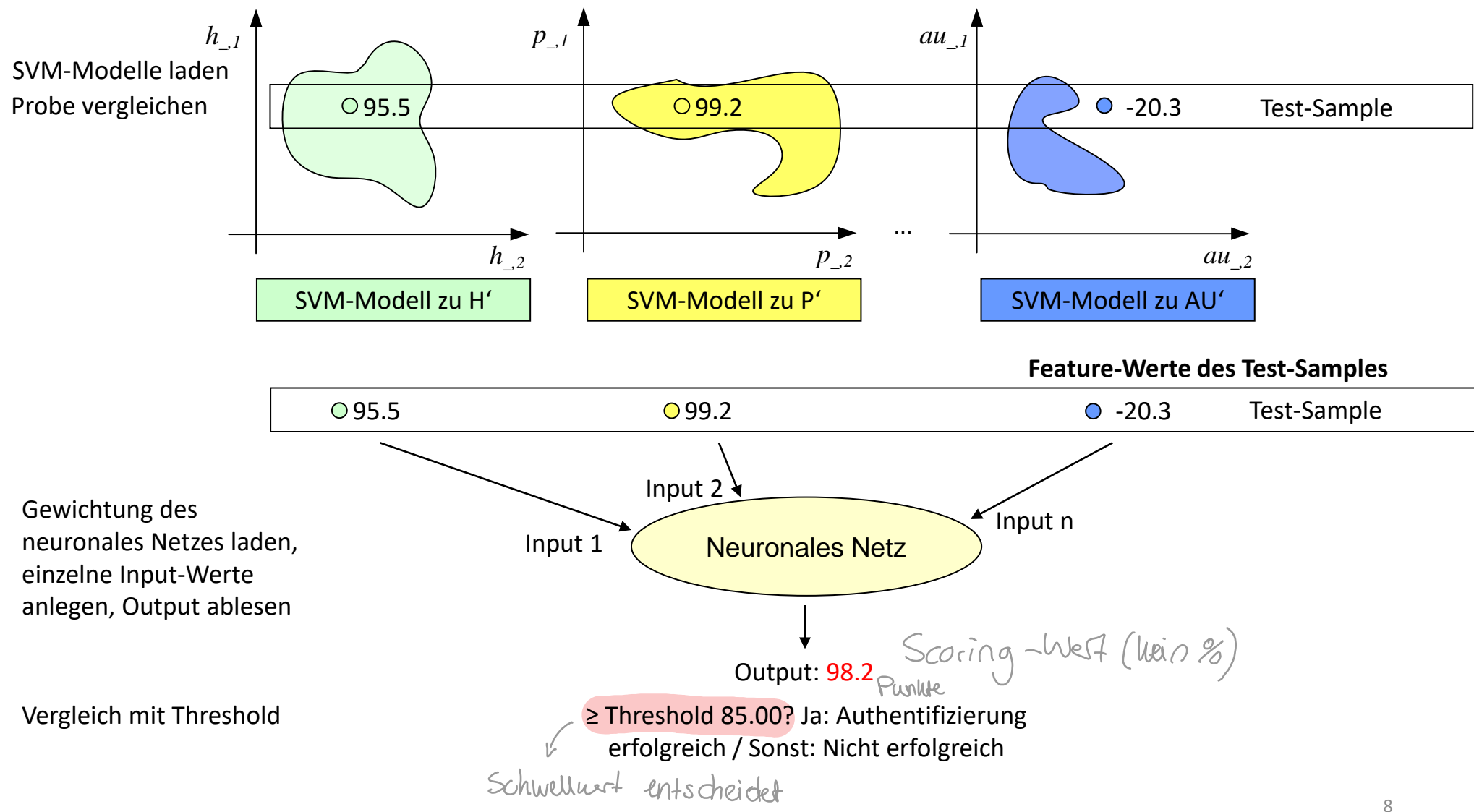


Übersicht über den Authentisierungs-Prozess (1/2)



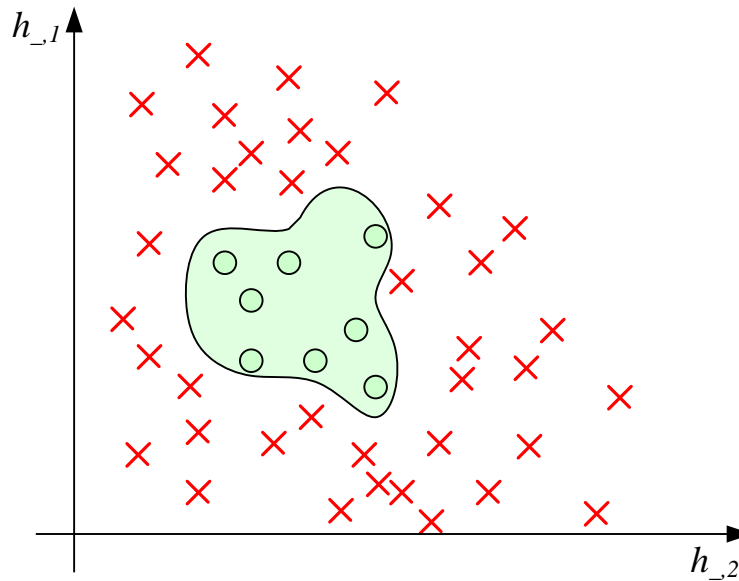
predict-propabilities -> LIBSVM -> Bedienungsanleitung -> proposed procedures with
↳ Scoring berechnen

Übersicht über den Authentisierungs-Prozess (2/2)



Ausblick

- Verfahren ohne Angreiferdaten (z. B. One-Class-SVM)



- Ersatz des Neuronalen Netzes zur Gewichtung der einzelnen Features
- Freitext-Variante
- Lebenderkennung