

4.) Decision Trees für Regression & Klassifikation

Entscheidungsbäume (Decision Trees) können sowohl für Regressions- als auch für Klassifikationsprobleme verwendet werden.

Es wird dabei ganz allgemein versucht eine Zuordnung auf Basis von Entscheidungsregeln vorzunehmen.

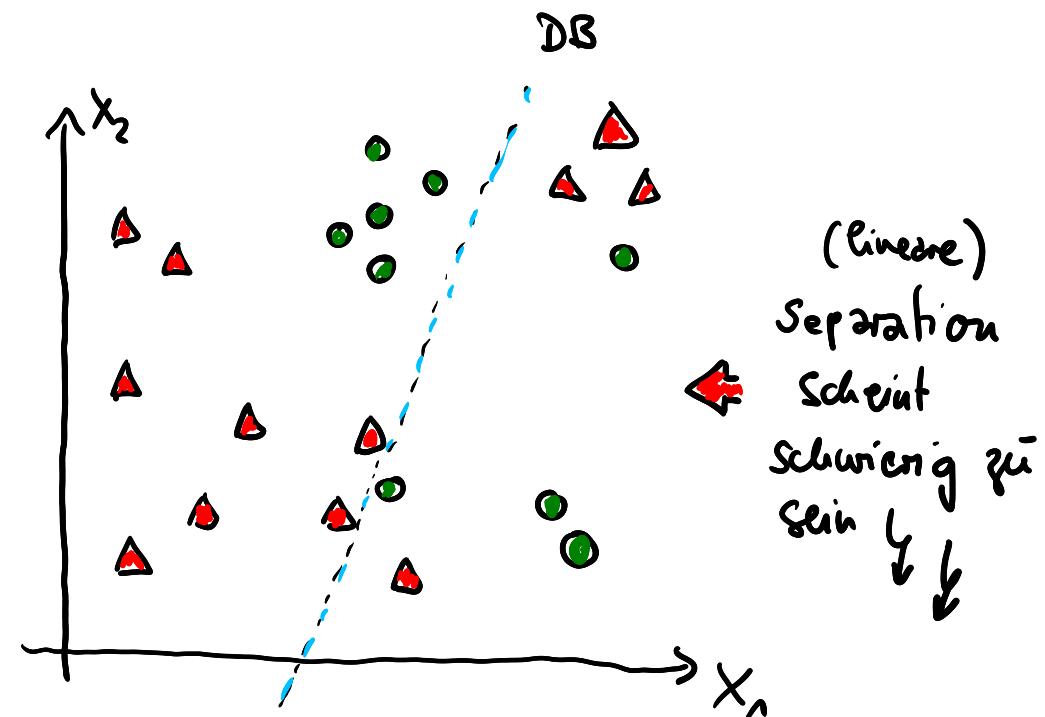
4.1) Vorüberlegungen

Geachte das folgende Beispiel

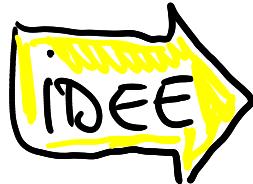
$X_1 \hat{=} \text{Alter}$

$X_2 \hat{=} \text{Einkommen}$

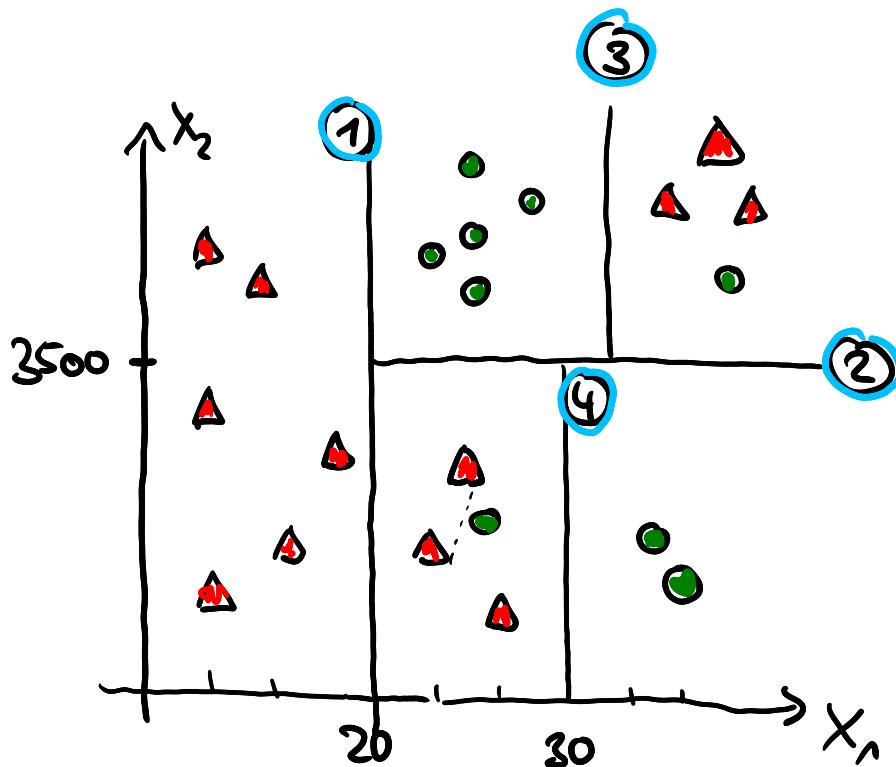
$Y \hat{=} \text{kreditwürdig ja } \circ \text{ nein } \Delta$



ein linearer Klassifikator würde versuchen eine optimale lineare Decision Boundary zu finden, die die 2 Gruppen trennt ...



Versuche Regionen zu finden, in denen "Mehrheitsentscheidungen" getroffen werden



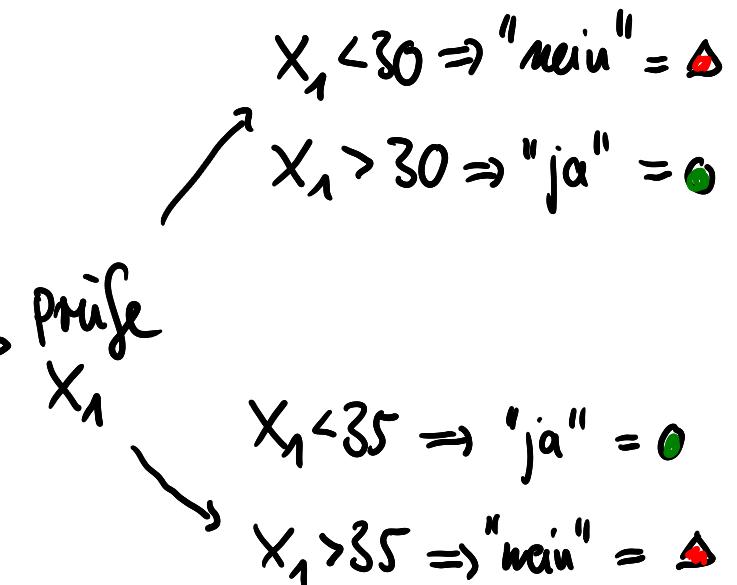
Separation durch ①, ②, ③ und ④ könnte sinnvoll sein, denn nur zwei Punkte werden "falsch" zugeordnet/eingeteilt

Jetzt muß man die Einteilung nur noch in ein Regelwerk übersetzen:

- wenn $X_1 < 20 \Rightarrow$ "nein" = Δ

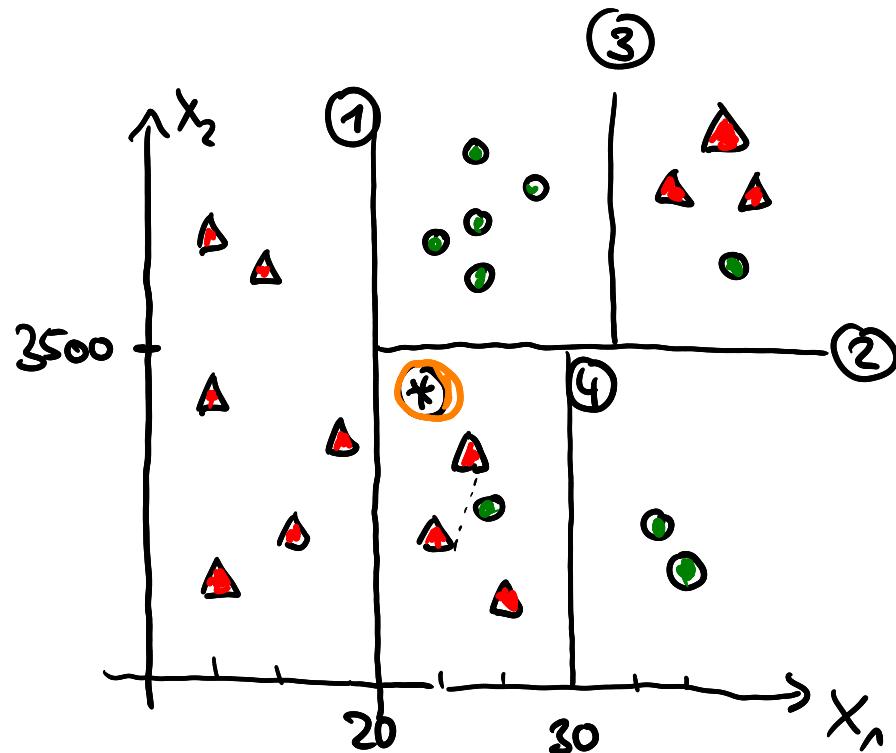
- wenn $X_1 > 20$ prüfe X_2 : $X_2 < 3500$

$X_2 > 3500$



Für einen neuen Punkt $* = (x_1^*, x_2^*)$ müssen jetzt nur noch die Regeln geprüft werden

d.h.: "in welchen Bereich fällt *?"



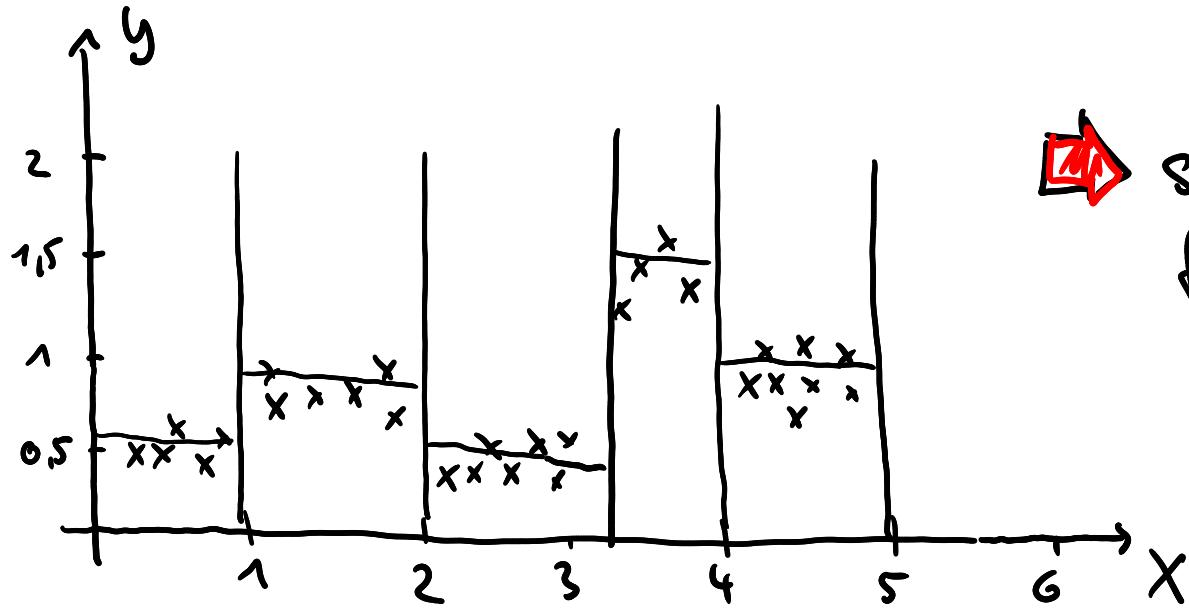
$x_1 \quad x_2$

→ neuer Punkt $\star = (22, 3000)$

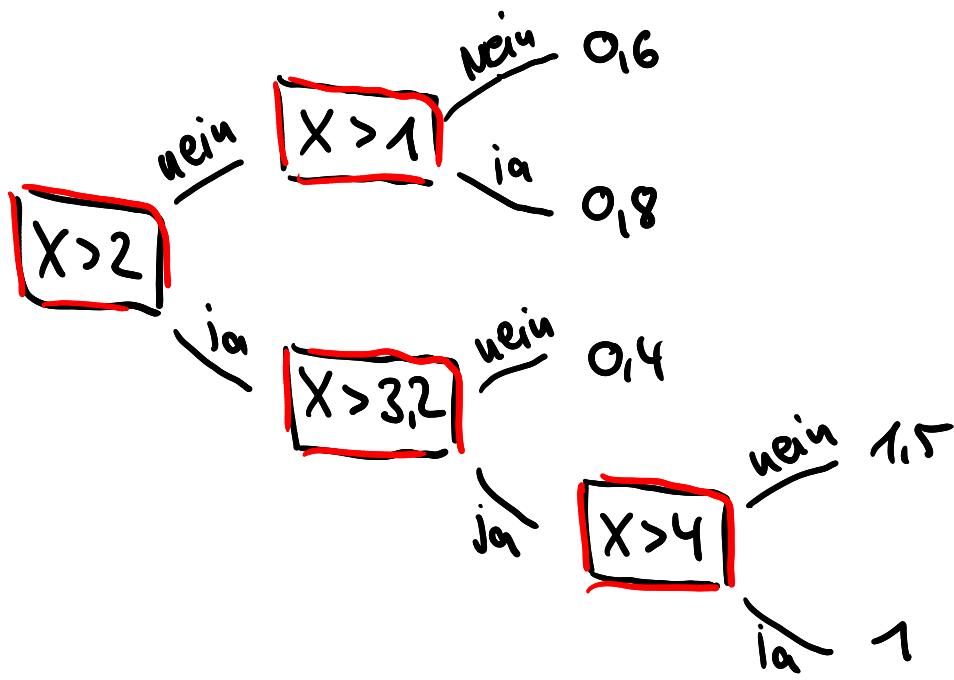
→ die Entscheidung heißt
3:1 für Δ !

→ der Punkt \star wird der Klasse
"nein" = Δ zugeordnet

Eine ähnliche Idee kann auch für ein Regressionsproblem
verwendet werden ...



→ starte bei einem x-Wert und finde Regeln, so dass eine Prognose = MW der y_i minimalen Fehler erreicht

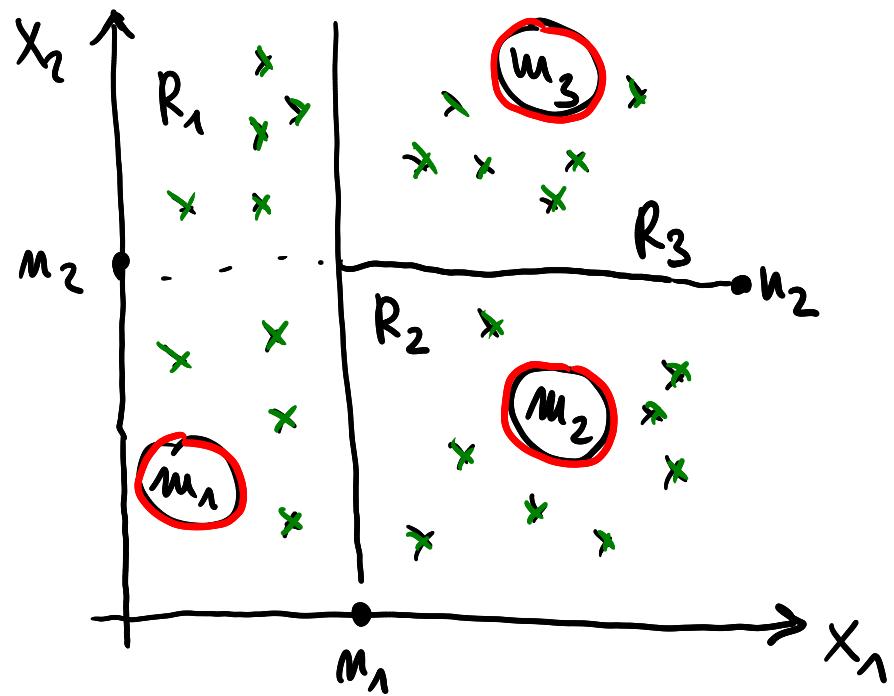


Die Prognose für einen neuen x-Wert entspricht dann einfach dem Mittelwert der Trainingsdaten im entsprechenden Bereich (Intervall)

4.1.1) Regression Trees

Betrachte ein (einfaches) Regressionsproblem mit 2 erklärenden Größen X_1, X_2 und Target Y .

Möglichen Ergebnis eines Splits des Feature-Raumes



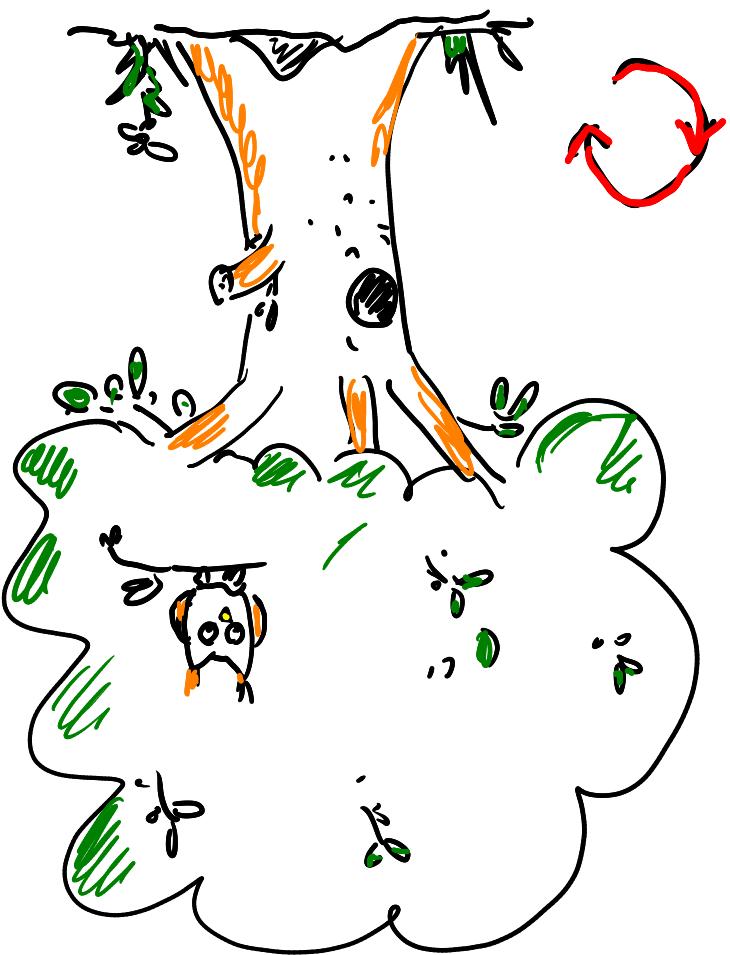
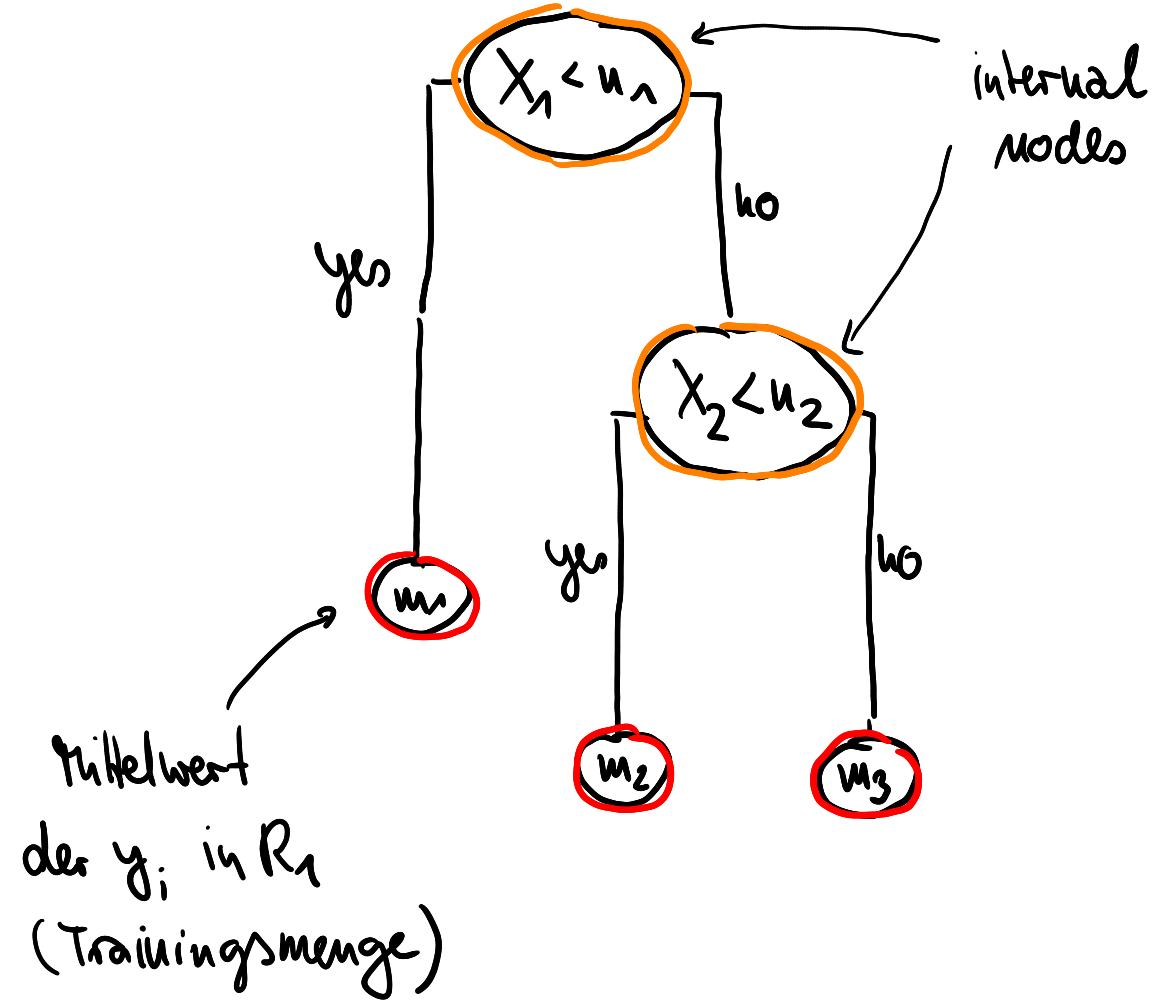
← Das Regelwerk unterteilt den Feature-Raum in 3 Regionen
 R_1, R_2 und R_3

Def.: Terminal Nodes

Die Regionen R_1, R_2, \dots im Feature-Raum, die einem Regression Tree zugrundeliegen heißen terminal nodes oder leaves (Blätter) des Baumes.

Die Punkte entlang des Baumes, an denen der Feature-Raum unterteilt wird heißen internal nodes.

Der eigentliche "Baum" wird üblicherweise von oben nach unten dargestellt ...



Wie geht man jetzt konkret vor um
den Baum zu finden?

Algorithmus:

STRATIFIKATION (Feature Space)

- ① Teile den Feature-Space (Menge möglicher Werte für X_1, \dots, X_p) in J nicht überlappende Regionen R_1, R_2, \dots, R_J (Boxes)
- ② Alle Beobachtungen in einem R_j ($j=1, \dots, J$) beherrschen als Prognosewert den Mittelwert derjenigen y_j für die $(x_{1j}, x_{2j}, \dots, x_{pj}) \in R_j$ gilt.

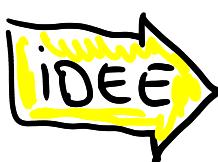
Die Einteilung wird optimalerweise so gewählt, dass auf der Trainingsmenge ein Minimum des Prognosefehlers erzielt wird, also

$$\sum_{j=1}^J \sum_{y_i \in R_j} (y_i - \hat{g}_{R_j})^2 \rightarrow \min$$

$$\hat{g}_{R_j} = \text{MW } y_i \in R_j \quad (\text{Training})$$



Es ist natürlich unmöglich alle denkbaren Einteilungen im Bezug auf minimalen Fehler zu prüfen! $\downarrow \downarrow$



TOP-DOWN GREEDY

\Rightarrow rekursives, binäres Splitten!

\leadsto wir gehen rekursiv vor und starten erstmal mit einem X_i ($i=1, \dots, p$)

dann schauen wir weiter...

Schritt

①

Wähle einen Prädiktor X_j und setze den Schwellenwert so, dass der Raum in die Regionen $\{X \mid X_j < s\}$ und $\{X \mid X_j \geq s\}$ zerfällt.

Bedingung: s ist so gewählt, dass der Fehler \otimes maximal reduziert wird (wir checken dabei alle X_j ($j=1, \dots, p$))

$\Rightarrow \forall j \forall s$

$$R_1(j, s) = \{X \mid X_j < s\}$$

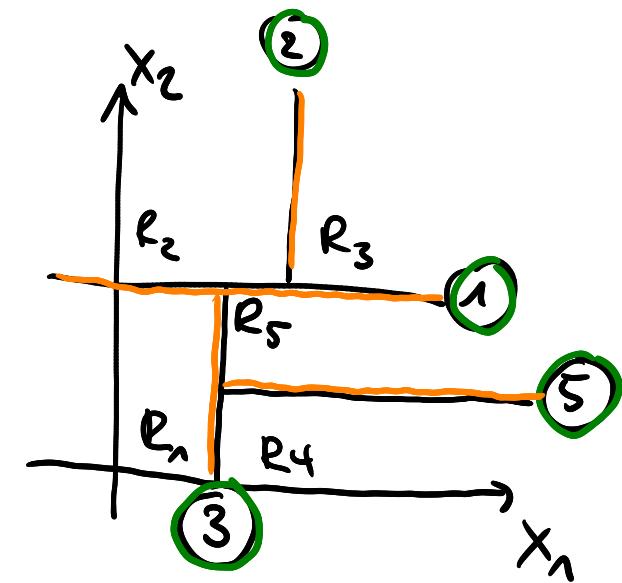
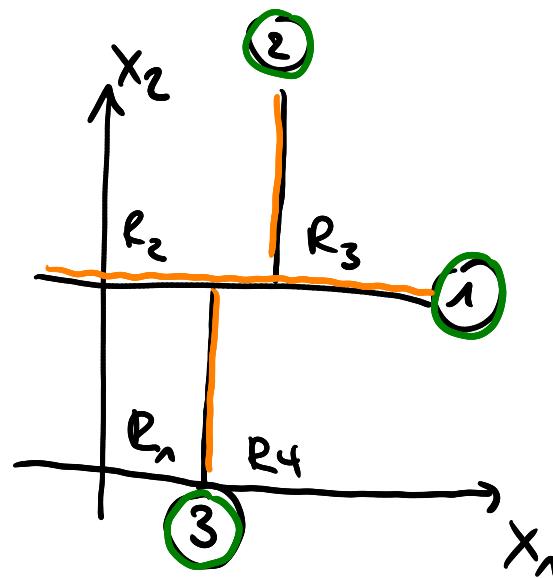
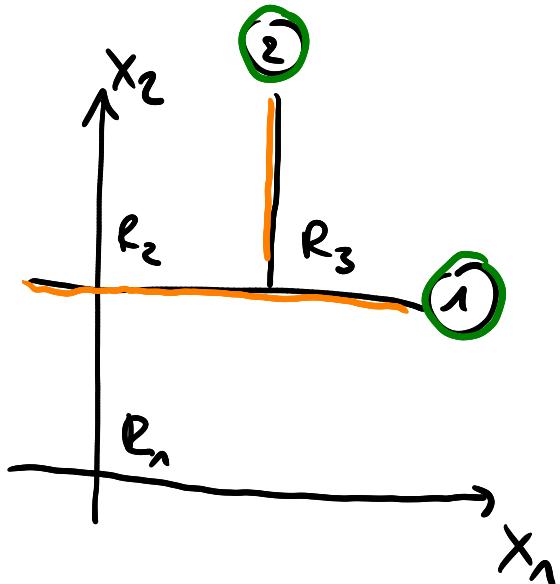
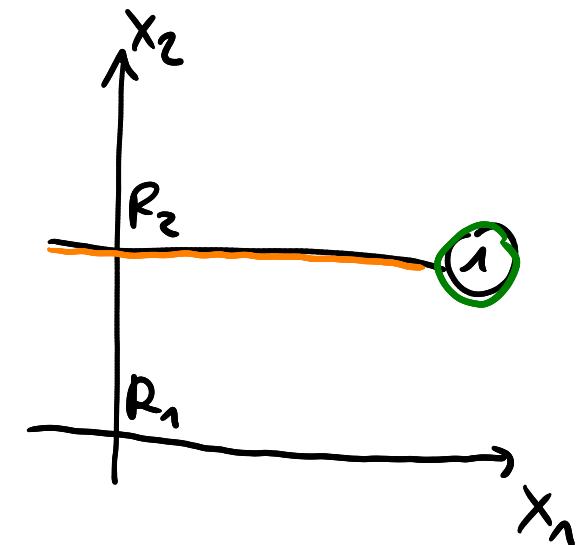
$$R_2(j, s) = \{X \mid X_j \geq s\}$$

suche (j, s) mit

$$\sum_{i: x_i \in R_1(j, s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i: x_i \in R_2(j, s)} (y_i - \hat{y}_{R_2})^2 \xrightarrow{j, s} \min$$

Im nächsten Schritt wiederholt man den Prozess für die Regionen R_1 und R_2 aus Schritt ①

Der Prozess läuft iterativ so lange weiter, bis ein Stop-Kriterium erreicht ist



... und so weiter ...

mögliche Stop-kriterien:

- nicht mehr als J Regionen
- nicht mehr als N Punkte pro Region
- Trainingsfehler für kein R_j größer als ...

Vorteil:

- der Algorithmus ist einfach zu implementieren
- das Fehlerkriterium kann angepasst werden
- das Ergebnis ist interpretierbar, Entscheidungsregeln liegen offen

⇒ Sensitivitätsanalyse ist einfach

Nachteil:

- die Bäume sind oft zu komplex
- ⇒ Super Ergebnis auf der Trainingsmenge, aber schlechte Generalisierung (hoher Testfehler)
- ⇒ OVERTFITTING



Starte mit einem sehr großen (komplexen) Baum T_0 und vereinfache ihn Schrittweise zu einem Teilbaum mit weniger nodes.

"Last in Kombinatorik" ...

→ Es wäre grundsätzlich denkbar, für jeden möglichen Teilbaum von T_0 den Testfehler zu bestimmen

→ Cross-Validation ist überläufig, weil die Bäume sich so stark unterscheiden für unterschiedliche Trainingsmengen (schon ein einzelner Punkt hat einen riesigen Einfluss)

→ wähle dann den besten Teilbaum aus

ABER...

riesiger Rechenaufwand, weil die Anzahl möglicher Teilbäume
zuviel groß sein kann



Man muss sich also auf eine kleinere Auswahl von
Teilbäumen beim Testen beschränken...

FRAGE Wie wähle ich sinnvoll ??

IDEA führe beim
Sichern des
Teilbaumes mit
dem geringsten Loss

$$\sum_{j=1}^J \sum_{y \in R_j} (y_i - \hat{y}_{R_j})^2 \rightarrow \text{min}$$

einen Penalty
für Komplexi-
tät ein ...

→ Betachte eine Folge von Bäumen, die mit einem nicht-negativen Tuning-parameter $\alpha \geq 0$ indiziert sind

Für jedes $\alpha \geq 0$ gibt es also einen Teilbaum $T \subset T_0$ (\leftarrow maximal groß),

so dass

$$\sum_{m=1}^{|T|} \sum_{i: x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha |T| \rightarrow \min$$

← soll so klein wie möglich werden...

dabei ist $|T| =$ Anzahl der terminal nodes von T , R_m Region zum m -ten terminal node

$$\sum_{m=1}^{|T|} \sum_{i: x_i \in R_m} (y_i - \hat{y}_{R_m})^2$$

$$+ \alpha \cdot |T|$$

Trainingsloss für den Teilbaum T

Penalty-Term für die Komplexität von T

Interpretation

$\alpha = 0$

: \Rightarrow der Teilbaum ist einfach T_0 , weil T_0 (wegen seiner Komplexität) auf jeden Fall den kleinsten Trainingsfehler hat

$\alpha \rightarrow \infty$

: \Rightarrow es wird "teuer" einen sehr komplexen Teilbaum $T \subseteq T_0$ zu wählen

(*) versucht einen möglichst kleinen Baum zu finden \rightsquigarrow Naives Modell $\hat{y} = \bar{y}$

Wie packt man das in einen Algorithmus?

α muss ja auch noch gefunden werden ...

Def.: Cost Complexity Pruning

- ① Bilde T_0 (stop prir bestimte Anzahl von Samples in den terminal nodes)

→ im "schlimmsten" Fall jeweils nur 1 sample pro $R_j \Rightarrow$ perfektes Modell (Training)

- ② Finde eine Folge bester Teilbäume mit Hilfe von

$$\sum_{m=1}^{|T|} \sum_{i: x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha |T| \rightarrow \min$$

in Abhängigkeit von α

- ③ Wähle α mit Hilfe von Cross-Validation

das geht so:

K-fold CV (Regression Tree)

für $k=1, \dots, K$:

- wiederhole ① + ② für alle außer dem k-ten Fold der Trainingsdaten
- Evaluiere den MSE auf dem k-ten Fold als Funktion von α
- Bilde den Mittelwert der K Fehler auf den K Folds (abhängig von α) und wähle α so, dass der gemittelte Fehler minimal wird.

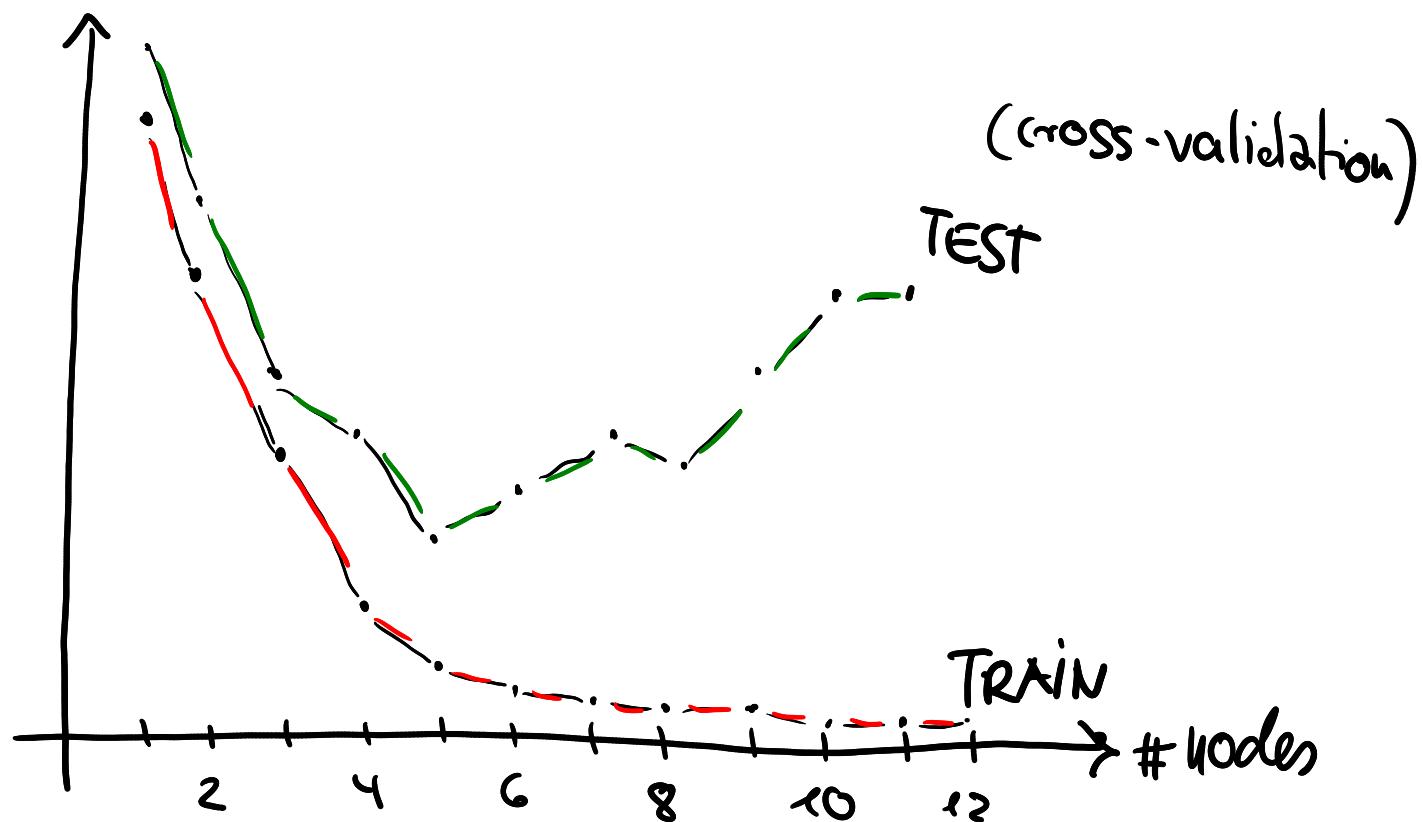
④ Ergebnis: Teilbaum aus ②, der zum berechneten α gehört (\rightsquigarrow "optimal" - zumindest bestmöglich)

Bemerkung : Das Verfahren kann als Regularisierung verstanden werden.

Üblicherweise sieht das Ergebnis beim Node Pruning so (oder so ähnlich) aus :

- Trainingsfehler sinkt
- Testfehler steigt ab einer bestimmten Komplexität

⇒ OVERTFITTING



wähle den Baum mit dem kleinsten Testfehler ...

4.1.2) Klassifikation

Die Zielgröße Y ist jetzt kategorial und nicht mehr metrisch !

Für Regression Trees halten wir als Prognose im Gebiet R_j den Mittelwert der dortigen Trainingswerte

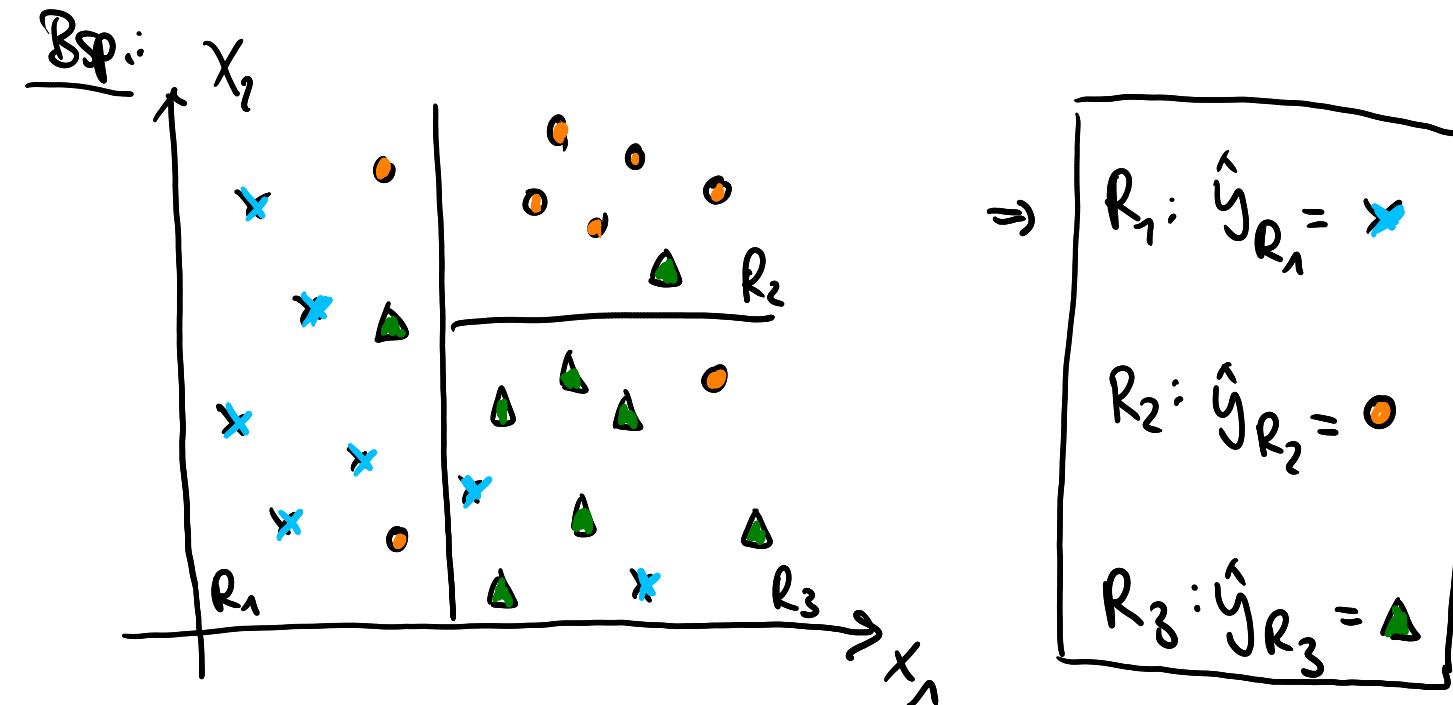
$$\hat{y}_{R_j} = \frac{1}{\#x_i \in R_j} \sum_{i: x_i \in R_j} y_i$$

gewählt.

Jetzt geht das leider so nicht mehr .

Mögliche Lösung : wähle als Prognose \hat{y}_{R_j} diejenige Klasse von Y für die die Anzahl der Beispiele im R_j am größten ist .

⇒ MEHRHEITSENTSCHEIDUNG (geht auch bei vielen Klassen)



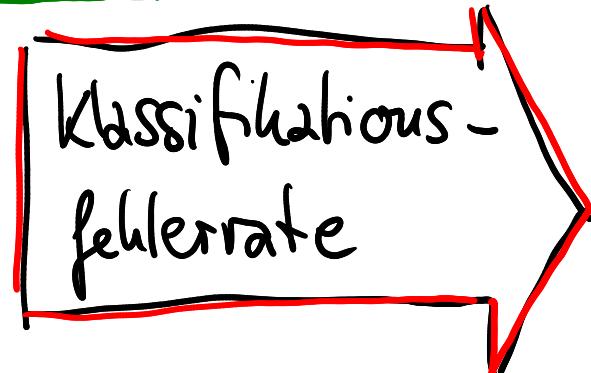
interessant ist dabei natürlich auch, wie eindeutig (sicher) die Entscheidung ausgefallen ist ...

\Rightarrow Anteil von \times, \circ, \triangle in R_1, R_2, R_3 ??

Zu Prinzip können wir die Aufteilung des Feature-Space in Regionen R_j ($j=1, \dots, J$) genau so machen wie bei den Regression Trees.

Wir brauchen nur eine andere Fehlerfunktion (Loss-function)

1 Möglichkeit:



Anteil der Trainingsamples in R_j ($j=1, \dots, J$), die nicht zur Klasse mit dem größten Anteil in R_j gehören

Bsp.: R_1 8 Samples: 5x \square 2o \triangle \Rightarrow

$$\hat{y}_{R_1} = x \quad \text{Fehlerrate für } R_1 = \frac{3}{8}$$

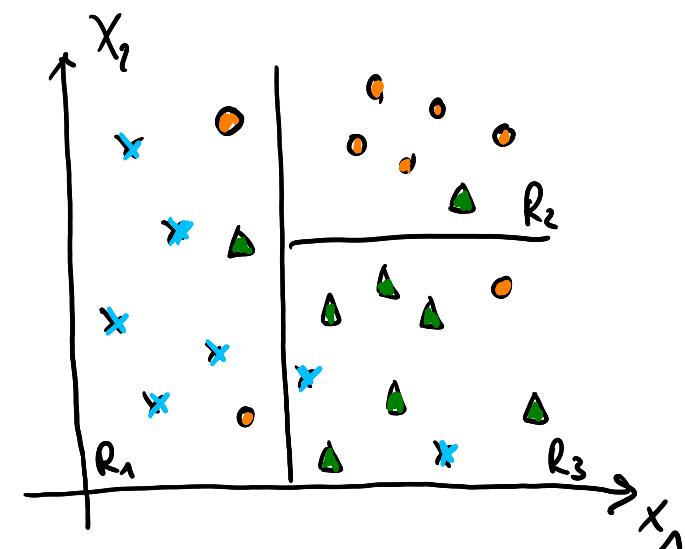
bezeichne mit \hat{P}_{mk} den Anteil der Trainingsamples im der m-ten Region R_m , die zur k-ten Klasse gehören, dann gilt:

$$\hat{P}_{1\Delta} = \frac{1}{8}$$

$$\hat{P}_{1\circ} = \frac{2}{8}$$

$$\hat{P}_{1x} = \frac{5}{8}$$

usw.



Def.: Klassifikationsfehlerrate

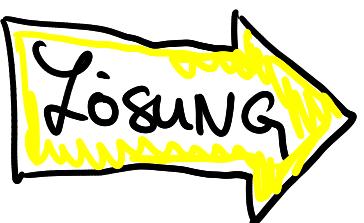
$$E = 1 - \max_k (\hat{P}_{m_k})$$

im der m-ten Region

Die Klassifikationsfehlerrate ist als Metrik einfach und intuitiv.

In der Praxis stellt sich aber heraus, dass sie zu wenig sensitiv ist um optimale Bäume zu strukturieren.

→ es wird nur darauf geschaut, was für "große" Fehler passieren ...



GINI - INDEX

ENTROPIE

Def.: Gini-Index

Der Gini-Index ist definiert als

$$G = \sum_{k=1}^K \hat{P}_{mk} (1 - \hat{P}_{mk})$$

G ist ein Maß für die totale Varianz der K Klassen innerhalb der Region R_m .

- Ein "optimaler" Gini-Index wird erreicht, wenn nur eine einzige Klasse in R_m vertreten ist (alle $\hat{P}_{mk} = 0$ bis auf einen $\hat{P}_{mk} = 1$)
⇒ "einstimmige Entscheidung"

G ist ein Reinheitsmaß für den m -ten Node!

→ kleiner G ⇒ hauptsächlich Samples einer Klasse ⇒ guter Split !!

Es liegt nahe, ein ähnliches Maß zu definieren, das sich von der Idee her an der Entropie (=Unordnung) in der Physik orientiert - ..

Def.: Entropie

Die Entropie ist definiert als

$$J = - \sum_{k=1}^K \hat{p}_{mk} \cdot \log(\hat{p}_{mk})$$

Überlegung: es gilt offensichtlich: $0 \leq \hat{p}_{mk} \leq 1$

$\Rightarrow 0 \leq -\hat{p}_{mk} \log(\hat{p}_{mk}) \Rightarrow$ für \hat{p}_{mk} nahe 1 ist $\log(\hat{p}_{mk})$ nahe Null!

$\Rightarrow J$ ist nahe Null, wenn nur Vertreter einer Klasse die Region R_m deutlich dominieren.

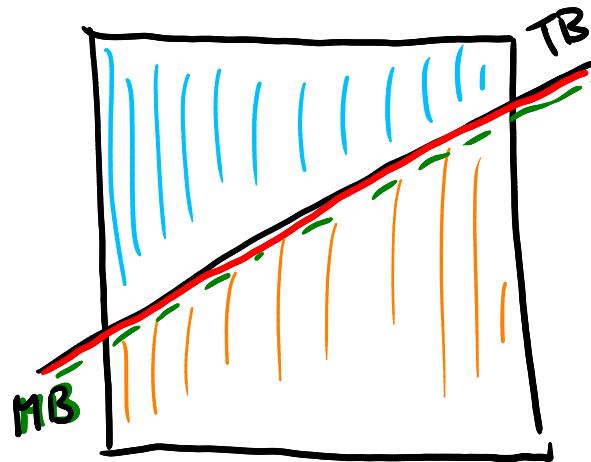
- Insgesamt gilt:
- Sowohl Gini-Index als auch Entropie D können zum Aufbau eines Klassifikationsbaumes und für das Pruning verwendet werden.
 - um final die Qualität der Prognosen auszuwerten eignet sich die Fehlerrate besser
→ intuitive Interpretierbarkeit!

Decision Trees sind eine gute Alternative zu z.B. linearen Klassifikationsmodellen

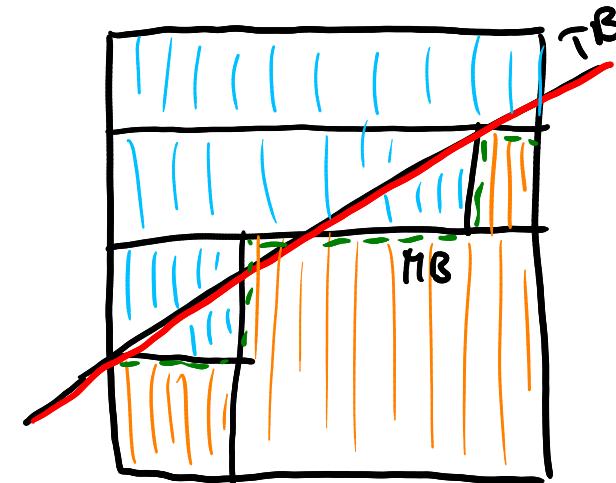
Sie sind in der Lage auch nicht-lineare Entscheidungsgrenzen zu berücksichtigen

Bsp.:

Fall
①



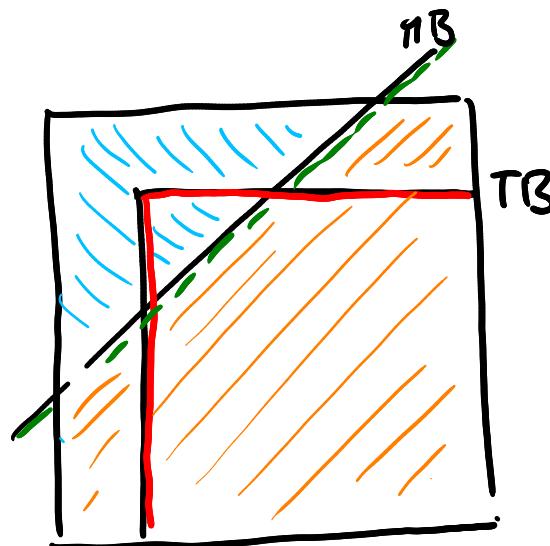
linearer
Klassifikator



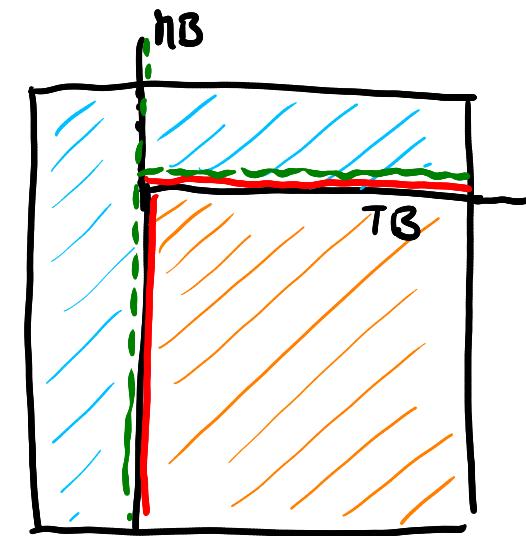
Tree

← nicht so toll, weil
die Decision
Boundary in Wahr-
heit linear ist!

Fall
②



linearer
Klassifikator



Tree

← hier klappts
perfekt 😊

ZUSAMMENFASSUNG



- leicht zu erklären, weil regelbasiert
- lehnen sich (algorithmisch) an den menschlichen Entscheidungsprozess an
- Päline können graphisch dargestellt werden → übersichtlich
- interpretierbar (besonders, wenn klein ☺)
- können auch direkt mit kategorischen Prädiktoren umgehen \Rightarrow keine Dummy-Variablen (OneHotEnc) nötig

NACHTEILE

- haben oft schlechte Prognosegüte
- neigen[↓] zu OVERTFITTING
 - hängen stark an den Daten =>
kleine Änderungen in den Trainingsdaten
können zu riesigen Änderungen in der
Baumstruktur (= Regelwerk) führen
=> geringe Robustheit

→ Gegen ein paar der Nachteile können wir vielleicht was tun -
mehr dazu beim nächsten Mal! ☺