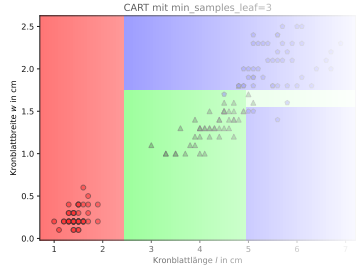
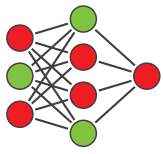
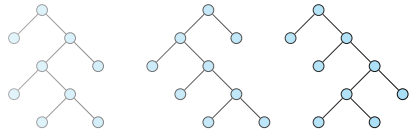
Entscheidungsbäume und Random Forest

Prof. Jörg Frochte **Christof Kaufmann**





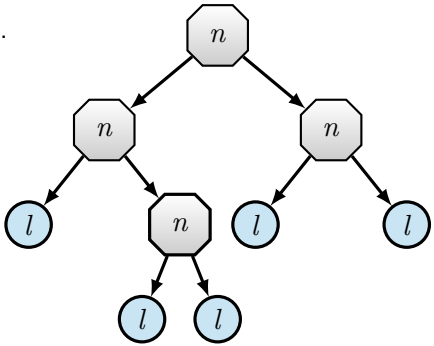
Weiterbildung AI
we-ai.de



CART-Algorithmus

CART (Classification and Regression Trees) ist ein einfacher Algorithmus um einen binären Entscheidungsbaum zu lernen.

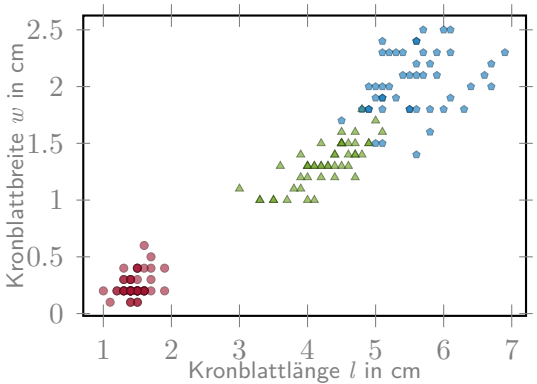
- CART wurde 1984 von Leo Breiman et al. publiziert.
- Wie der Name des Algorithmus schon erahnen lässt, kann er sowohl zur Klassifizierung als auch zur Regression eingesetzt werden.
- Wir beginnen mit der Klassifikation.



Grob läuft der Algorithmus wie folgt ab:

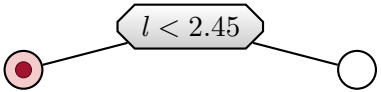
1. Es wird bestimmt wo der nächste Schnitt sein soll.
2. Die (Teil-)Datenmenge wird in zwei Teile aufgeteilt.
3. Auf jedem dieser Teildatenmengen fängt der Alorithmus bei Schritt 1 erneut an, falls noch weiter unterteilt werden soll.

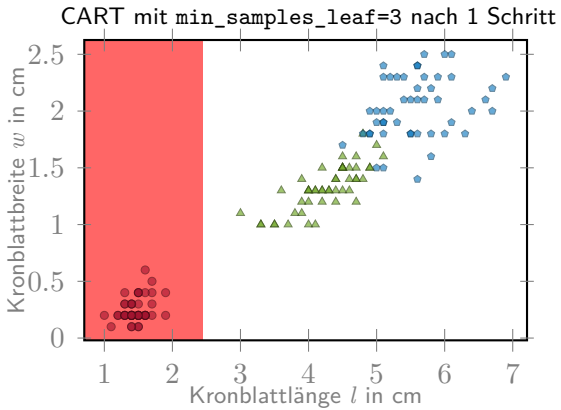
Beispiel-Baum mit CART-Algorithmus



- Der CART versucht so zu schneiden, dass die Teilmengen möglicht homogen sind.
- Dabei werden für jeden Schnitt alle möglichen Schnitte auf allen Achsen betrachtet.

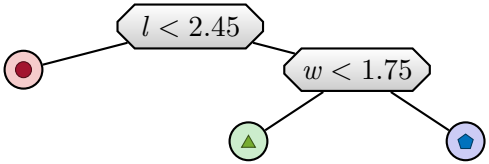
Beispiel-Baum mit CART-Algorithmus

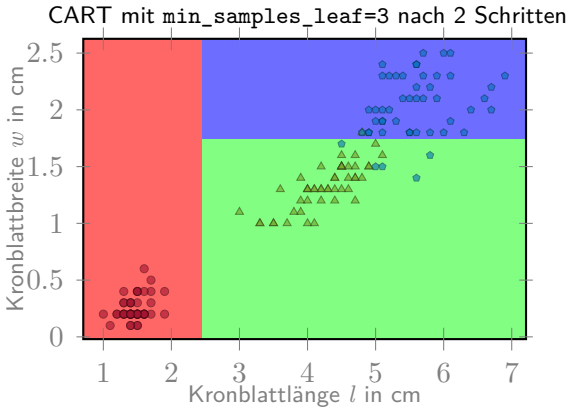




- Der CART versucht so zu schneiden, dass die Teilmengen möglicht homogen sind.
- Dabei werden für jeden Schnitt alle möglichen Schnitte auf allen Achsen betrachtet.

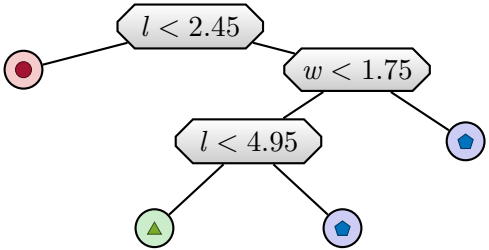
Beispiel-Baum mit CART-Algorithmus

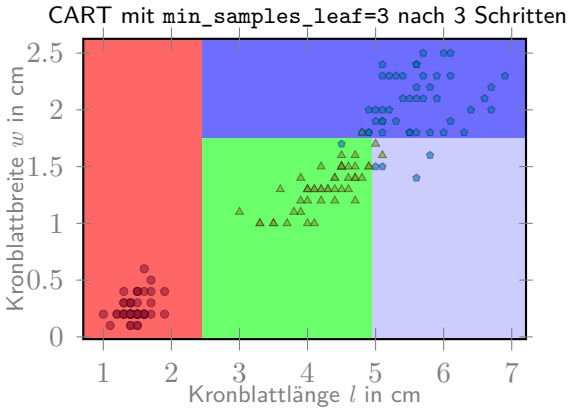




- Der CART versucht so zu schneiden, dass die Teilmengen möglicht homogen sind.
- Dabei werden für jeden Schnitt alle möglichen Schnitte auf allen Achsen betrachtet.

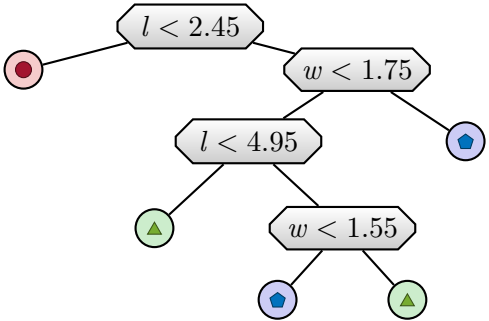
Beispiel-Baum mit CART-Algorithmus

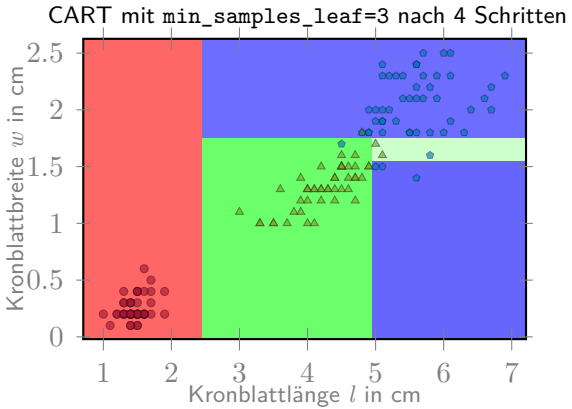




- Der CART versucht so zu schneiden, dass die Teilmengen möglicht homogen sind.
- Dabei werden für jeden Schnitt alle möglichen Schnitte auf allen Achsen betrachtet.

Beispiel-Baum mit CART-Algorithmus



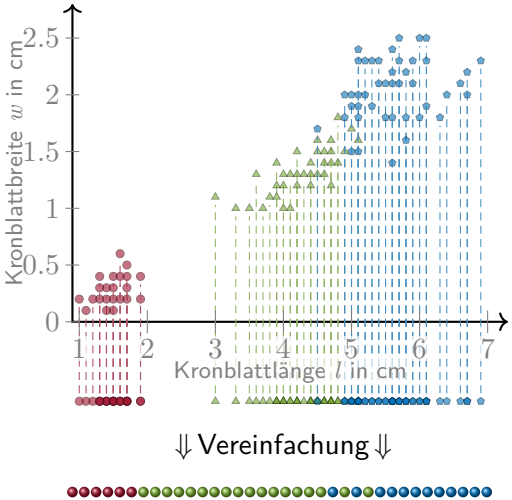


- Nun ist der Algorithmus fertig, denn als Bedingung für einen Schnitt wurde hier `min_samples_leaf=3` eingestellt.
- Der CART versucht so zu schneiden, dass die Teilmengen möglicht homogen sind.
- Dabei werden für jeden Schnitt alle möglichen Schnitte auf allen Achsen betrachtet.

Gini Impurity

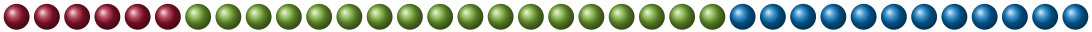
Wie wird entschieden, wo geschnitten werden soll?

- Das Ziel ist die *Verunreinigung* (Impurity) zu reduzieren.
- Als Maß für die Verunreinigung wird die **Gini Impurity** verwendet.
- Man will versuchen, die Datenmenge bei jeder Entscheidung homogener zu bekommen.
- Wir gehen im Folgenden davon aus, dass es $c \in \mathbb{N}$ (hier: $c = 3$) verschiedene Klassen gibt.
- Wir betrachten nur ein Feature um die Gini Impurity zu erklären. Der Algorithmus prüft jedoch für einen Schnitt immer alle Features.



Herleitung Gini Impurity

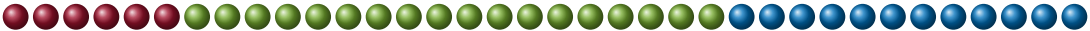
Beispiel: $6\times$●, $18\times$●, $12\times$●, also 36 Kugeln, hier sortiert:



Sei p_i der Anteil von Klasse i an der Menge. Hier $p_1 = \frac{1}{6}, p_2 = \frac{1}{2}, p_3 = \frac{1}{3}$

Herleitung Gini Impurity

Beispiel: $6\times$  , $18\times$  , $12\times$  , also 36 Kugeln, hier sortiert:

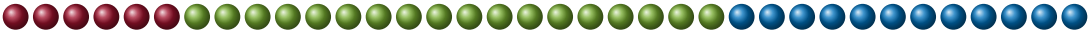


Sei p_i der Anteil von Klasse i an der Menge. Hier $p_1 = \frac{1}{6}, p_2 = \frac{1}{2}, p_3 = \frac{1}{3}$

Wenn man nun die Klassifizierung zufällig mit den Wahrscheinlichkeiten p_i vornimmt, wie hoch ist jeweils der Anteil der Fehlklassifikationen f_i?

Herleitung Gini Impurity

Beispiel: $6\times$  , $18\times$  , $12\times$  , also 36 Kugeln, hier sortiert:

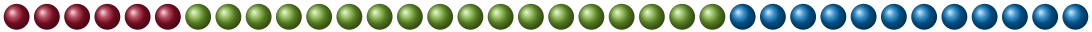


Sei p_i der Anteil von Klasse i an der Menge. Hier $p_1 = \frac{1}{6}, p_2 = \frac{1}{2}, p_3 = \frac{1}{3}$

Wenn man nun die Klassifizierung zufällig mit den Wahrscheinlichkeiten p_i vornimmt, wie hoch ist jeweils der Anteil der Fehlklassifikationen f_i?

Herleitung Gini Impurity

Beispiel: $6\times$●, $18\times$●, $12\times$●, also 36 Kugeln, hier sortiert:



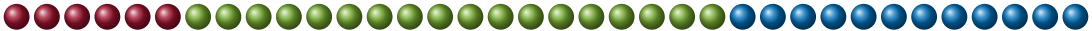
Sei p_i der Anteil von Klasse i an der Menge. Hier $p_1 = \frac{1}{6}, p_2 = \frac{1}{2}, p_3 = \frac{1}{3}$

Wenn man nun die Klassifizierung zufällig mit den Wahrscheinlichkeiten p_i vornimmt, wie hoch ist jeweils der Anteil der Fehlklassifikationen f_i?

1. Für Klassifikation 1 gibt es $\frac{1}{6} \cdot 18 = 3$ Fehlklassifikationen in Klasse 2 und $\frac{1}{6} \cdot 12 = 2$ in Klasse 3. Anteilig also $f_1 = \frac{1}{6} \cdot \left(\frac{1}{2} + \frac{1}{3}\right) = \frac{1}{6} \cdot \frac{5}{6}$, d. h. $\frac{1}{6}$ falsch in $\frac{5}{6}$ der Menge.

Herleitung Gini Impurity

Beispiel: $6\times$●, $18\times$●, $12\times$●, also 36 Kugeln, hier sortiert:



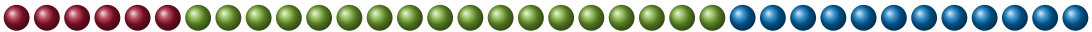
Sei p_i der Anteil von Klasse i an der Menge. Hier $p_1 = \frac{1}{6}, p_2 = \frac{1}{2}, p_3 = \frac{1}{3}$

Wenn man nun die Klassifizierung zufällig mit den Wahrscheinlichkeiten p_i vornimmt, wie hoch ist jeweils der Anteil der Fehlklassifikationen f_i?

1. Für Klassifikation 1 gibt es $\frac{1}{6} \cdot 18 = 3$ Fehlklassifikationen in Klasse 2 und $\frac{1}{6} \cdot 12 = 2$ in Klasse 3. Anteilig also $f_1 = \frac{1}{6} \cdot \left(\frac{1}{2} + \frac{1}{3}\right) = \frac{1}{6} \cdot \frac{5}{6}$, d. h. $\frac{1}{6}$ falsch in $\frac{5}{6}$ der Menge.
2. Für Klassifikation 2 ist $f_2 = \frac{1}{2} \cdot \left(\frac{1}{6} + \frac{1}{3}\right) = \frac{1}{2} \cdot \frac{1}{2}$.

Herleitung Gini Impurity

Beispiel: $6\times$●, $18\times$●, $12\times$●, also 36 Kugeln, hier sortiert:



Sei p_i der Anteil von Klasse i an der Menge. Hier $p_1 = \frac{1}{6}, p_2 = \frac{1}{2}, p_3 = \frac{1}{3}$

Wenn man nun die Klassifizierung zufällig mit den Wahrscheinlichkeiten p_i vornimmt, wie hoch ist jeweils der Anteil der Fehlklassifikationen f_i?

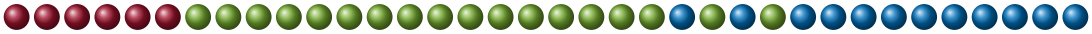
1. Für Klassifikation 1 gibt es $\frac{1}{6} \cdot 18 = 3$ Fehlklassifikationen in Klasse 2 und $\frac{1}{6} \cdot 12 = 2$ in Klasse 3. Anteilig also $f_1 = \frac{1}{6} \cdot \left(\frac{1}{2} + \frac{1}{3}\right) = \frac{1}{6} \cdot \frac{5}{6}$, d. h. $\frac{1}{6}$ falsch in $\frac{5}{6}$ der Menge.
2. Für Klassifikation 2 ist $f_2 = \frac{1}{2} \cdot \left(\frac{1}{6} + \frac{1}{3}\right) = \frac{1}{2} \cdot \frac{1}{2}$.
3. Für Klassifikation 3 ist $f_3 = \frac{1}{3} \cdot \left(\frac{1}{6} + \frac{1}{2}\right) = \frac{1}{3} \cdot \frac{2}{3}$. Allgemein gilt: $f_i = p_i \cdot (1 - p_i)$.

Herleitung Gini Impurity

Die Gini Impurity ist nun die Summe der anteiligen Fehlklassifikationen:

$$G = \sum_{i=1}^{c} f_i = \sum_{i=1}^{c} p_i \cdot (1 - p_i) = \underbrace{\sum_{i=1}^{c} p_i}_{=1} - \sum_{i=1}^{c} p_i^2 = 1 - \sum_{i=1}^{c} p_i^2$$

Für unser Beispiel ist also $G = 1 - \left(\frac{1}{6}\right)^2 - \left(\frac{1}{2}\right)^2 - \left(\frac{1}{3}\right)^2 = \frac{22}{36} = 0.6\bar{1}$



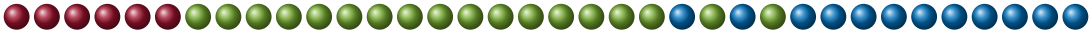
Bei einer reinen Menge, wäre die Gini Impurity übrigens 0.
Nun ist das Ziel jedoch die Menge in zwei Teilmengen X_1, X_2 aufzuteilen, denn wir wollen letztendlich einen binären Baum bekommen.

Gini Impurity für zwei Teilmengen

Um die Gini Impurity von zwei Teilmengen X_1, X_2 zu ermitteln, bilden wir das gewichtete Mittel der entsprechenden Gini Impurities G_1, G_2.

$$\bar{G} = \frac{G_1 \cdot |X_1| + G_2 \cdot |X_2|}{|X_1| + |X_2|}$$

Beispiel:

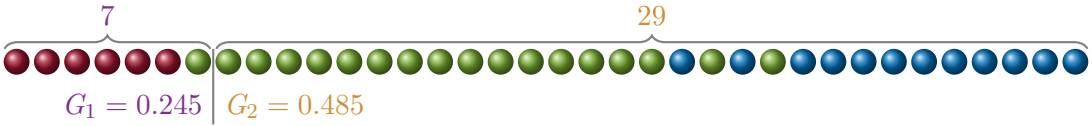


Gini Impurity für zwei Teilmengen

Um die Gini Impurity von zwei Teilmengen X_1, X_2 zu ermitteln, bilden wir das gewichtete Mittel der entsprechenden Gini Impurities G_1, G_2.

$$\bar{G} = \frac{G_1 \cdot |X_1| + G_2 \cdot |X_2|}{|X_1| + |X_2|}$$

Beispiel:



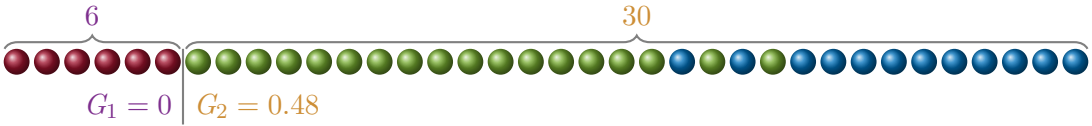
$$\bar{G} = \frac{0.245 \cdot 7 + 0.485 \cdot 29}{36} = 0.438$$

Gini Impurity für zwei Teilmengen

Um die Gini Impurity von zwei Teilmengen X_1, X_2 zu ermitteln, bilden wir das gewichtete Mittel der entsprechenden Gini Impurities G_1, G_2.

$$\bar{G} = \frac{G_1 \cdot |X_1| + G_2 \cdot |X_2|}{|X_1| + |X_2|}$$

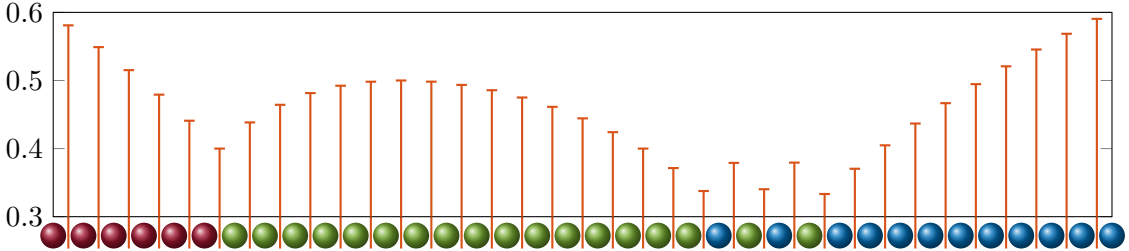
Beispiel:



$$\bar{G} = \frac{0 \cdot 6 + 0.48 \cdot 30}{36} = 0.4$$

Gini Impurity als Optimierungskriterium

Nun wählen wir den besten Schnitt für dieses Feature, d. h. den Schnitt bei dem die kleinste, mittlere Gini Impurity $\bar{G}$ entsteht. Dazu müssen alle möglichen Schnitte probiert werden.

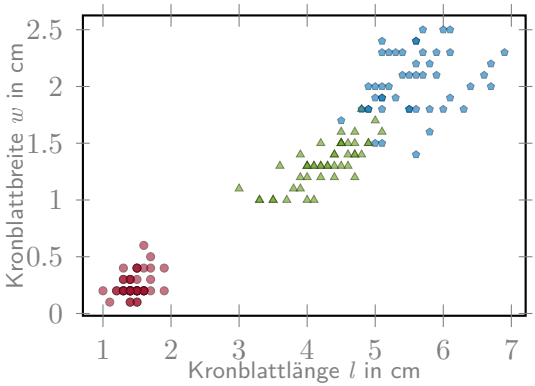


Allgemein...

- sind die Abstände nicht gleich. Es wird jeweils genau in der **Mitte zwischen zwei benachbarten Samples** geschnitten.
- gibt es mehrere Features. Es werden **alle Features** nacheinander probiert. Der beste Schnitt gewinnt.

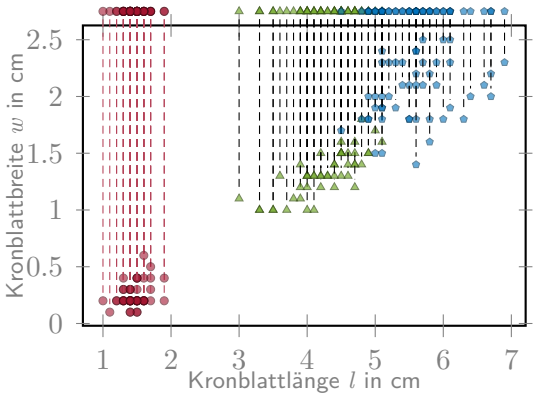
CART mit Gini Impurity

- Die Daten werden jeweils entlang einer Achse (ein Feature) betrachtet.



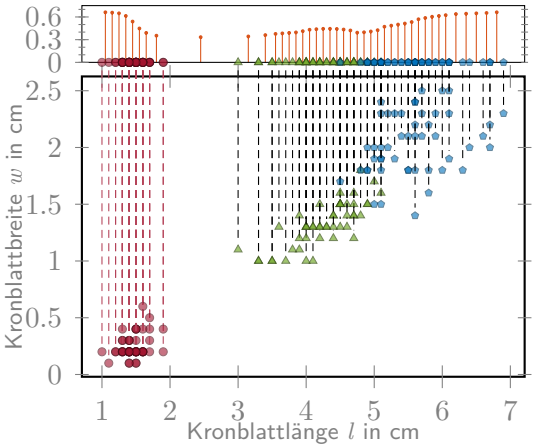
CART mit Gini Impurity

- Die Daten werden jeweils entlang einer Achse (ein Feature) betrachtet.
- Schnittmöglichkeiten bestehen jeweils mittig zwischen zwei Samples.



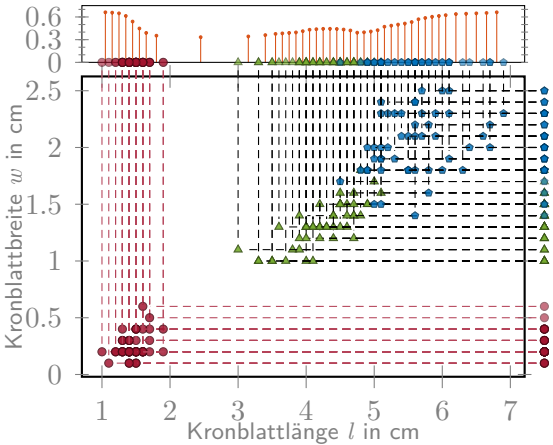
CART mit Gini Impurity

- Die Daten werden jeweils entlang einer Achse (ein Feature) betrachtet.
- Schnittmöglichkeiten bestehen jeweils mittig zwischen zwei Samples.
- Die Gini Impurity wird für jede Schnittmöglichkeit berechnet.



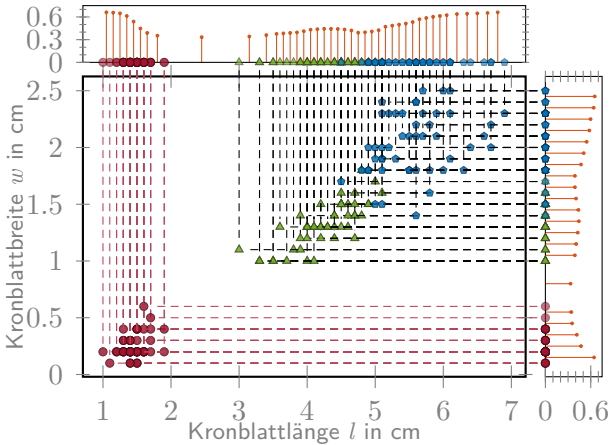
CART mit Gini Impurity

- Die Daten werden jeweils entlang einer Achse (ein Feature) betrachtet.
- Schnittmöglichkeiten bestehen jeweils mittig zwischen zwei Samples.
- Die Gini Impurity wird für jede Schnittmöglichkeit berechnet.
- Dies wird für alle Features gemacht.



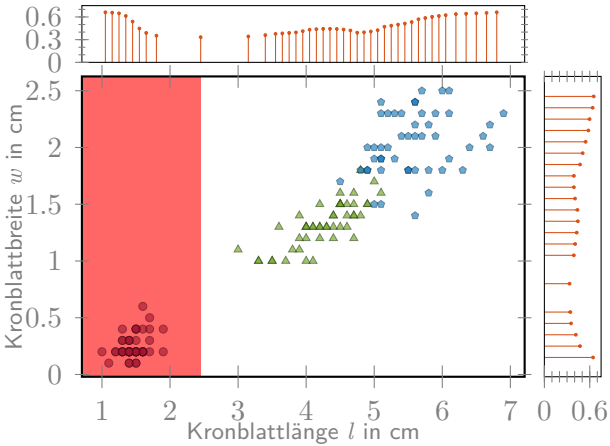
CART mit Gini Impurity

- Die Daten werden jeweils entlang einer Achse (ein Feature) betrachtet.
- Schnittmöglichkeiten bestehen jeweils mittig zwischen zwei Samples.
- Die Gini Impurity wird für jede Schnittmöglichkeit berechnet.
- Dies wird für alle Features gemacht.



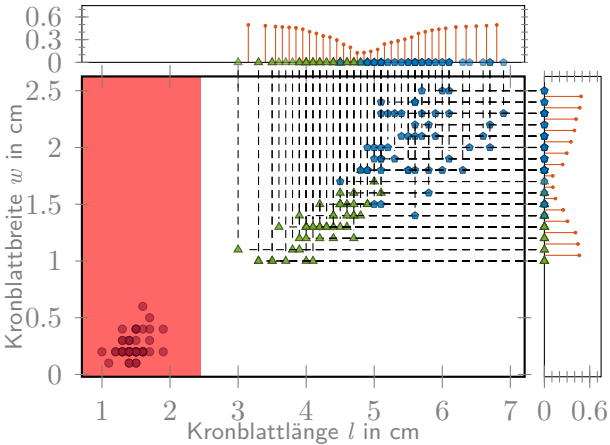
CART mit Gini Impurity

- Die Daten werden jeweils entlang einer Achse (ein Feature) betrachtet.
- Schnittmöglichkeiten bestehen jeweils mittig zwischen zwei Samples.
- Die Gini Impurity wird für jede Schnittmöglichkeit berechnet.
- Dies wird für alle Features gemacht.
- Der Schnitt wird bei der kleinsten Gini Impurity gemacht.



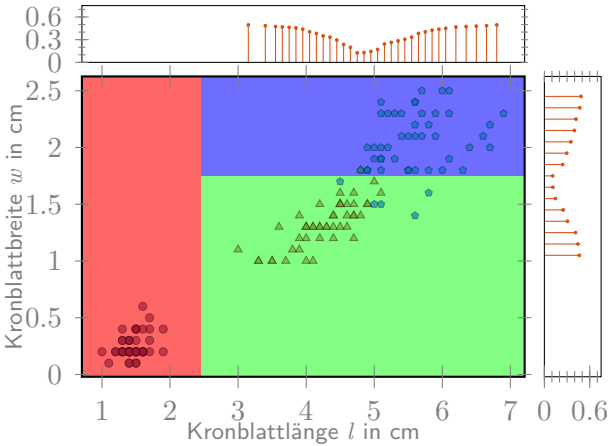
CART mit Gini Impurity

- Die Daten werden jeweils entlang einer Achse (ein Feature) betrachtet.
- Schnittmöglichkeiten bestehen jeweils mittig zwischen zwei Samples.
- Die Gini Impurity wird für jede Schnittmöglichkeit berechnet.
- Dies wird für alle Features gemacht.
- Der Schnitt wird bei der kleinsten Gini Impurity gemacht.



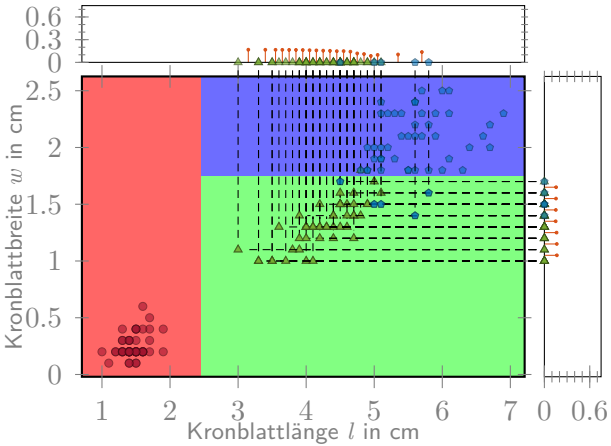
CART mit Gini Impurity

- Die Daten werden jeweils entlang einer Achse (ein Feature) betrachtet.
- Schnittmöglichkeiten bestehen jeweils mittig zwischen zwei Samples.
- Die Gini Impurity wird für jede Schnittmöglichkeit berechnet.
- Dies wird für alle Features gemacht.
- Der Schnitt wird bei der kleinsten Gini Impurity gemacht.



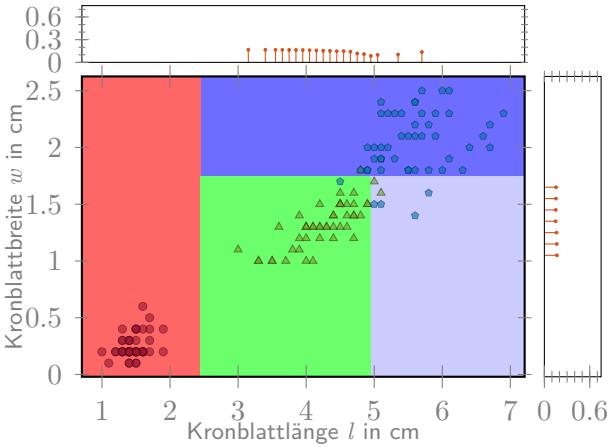
CART mit Gini Impurity

- Die Daten werden jeweils entlang einer Achse (ein Feature) betrachtet.
- Schnittmöglichkeiten bestehen jeweils mittig zwischen zwei Samples.
- Die Gini Impurity wird für jede Schnittmöglichkeit berechnet.
- Dies wird für alle Features gemacht.
- Der Schnitt wird bei der kleinsten Gini Impurity gemacht.



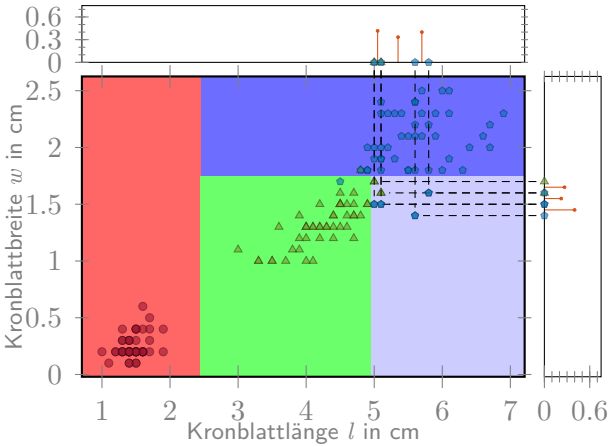
CART mit Gini Impurity

- Die Daten werden jeweils entlang einer Achse (ein Feature) betrachtet.
- Schnittmöglichkeiten bestehen jeweils mittig zwischen zwei Samples.
- Die Gini Impurity wird für jede Schnittmöglichkeit berechnet.
- Dies wird für alle Features gemacht.
- Der Schnitt wird bei der kleinsten Gini Impurity gemacht.



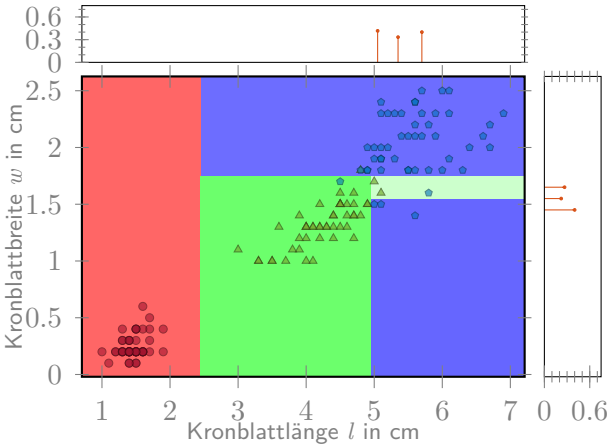
CART mit Gini Impurity

- Die Daten werden jeweils entlang einer Achse (ein Feature) betrachtet.
- Schnittmöglichkeiten bestehen jeweils mittig zwischen zwei Samples.
- Die Gini Impurity wird für jede Schnittmöglichkeit berechnet.
- Dies wird für alle Features gemacht.
- Der Schnitt wird bei der kleinsten Gini Impurity gemacht.



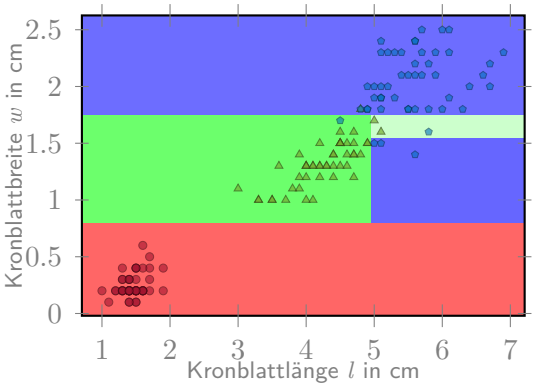
CART mit Gini Impurity

- Die Daten werden jeweils entlang einer Achse (ein Feature) betrachtet.
- Schnittmöglichkeiten bestehen jeweils mittig zwischen zwei Samples.
- Die Gini Impurity wird für jede Schnittmöglichkeit berechnet.
- Dies wird für alle Features gemacht.
- Der Schnitt wird bei der kleinsten Gini Impurity gemacht.



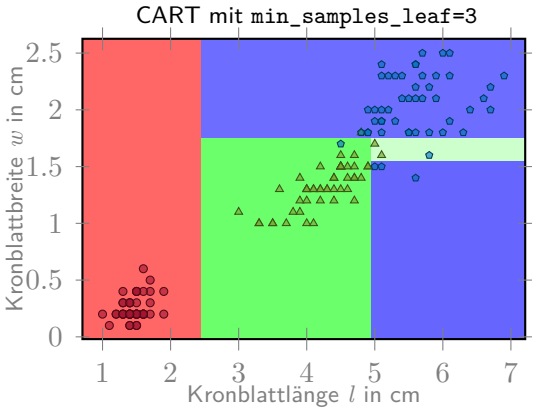
CART mit Gini Impurity

- Die Daten werden jeweils entlang einer Achse (ein Feature) betrachtet.
- Schnittmöglichkeiten bestehen jeweils mittig zwischen zwei Samples.
- Die Gini Impurity wird für jede Schnittmöglichkeit berechnet.
- Dies wird für alle Features gemacht.
- Der Schnitt wird bei der kleinsten Gini Impurity gemacht.
- Manche Schnitte sind nicht eindeutig und können zufällig zu einem anderen Ergebnis führen.



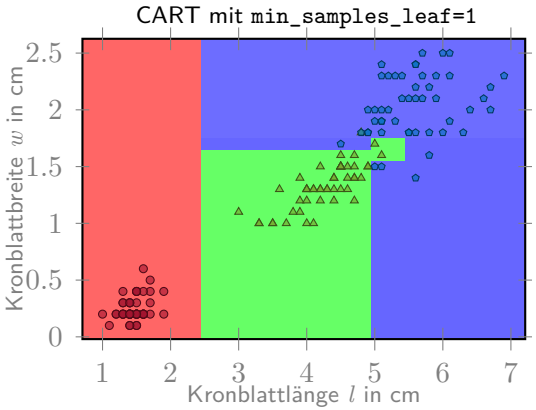
Overfitting

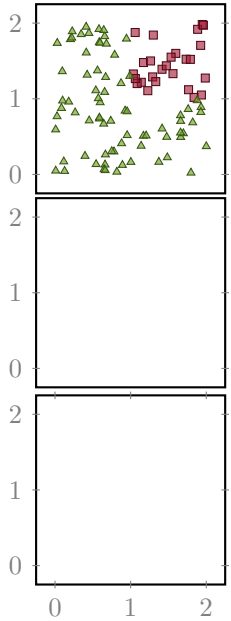
- Ohne Einschränkung, wann CART aufhören soll, bricht der Algorithmus erst ab, wenn keine Verbesserung mehr erreicht werden kann.



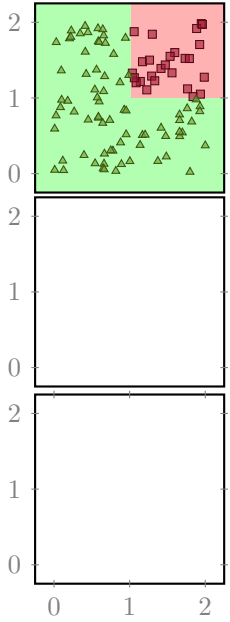
Overfitting

- Ohne Einschränkung, wann CART aufhören soll, bricht der Algorithmus erst ab, wenn keine Verbesserung mehr erreicht werden kann.
- Der Baum passt sich den Trainingsdaten bestmöglichst an.
- Bei verrauschten oder schwer zu trennenden Daten ist das nicht gewollt und nennt sich *overfitting*.
- Betrachten wir dazu ein extremes Beispiel, das zeigen soll wie empfindlich CART gegenüber Rauschen ist und was man dagegen machen kann.

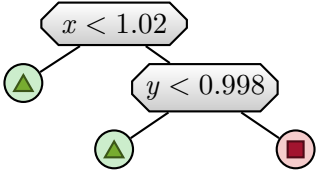


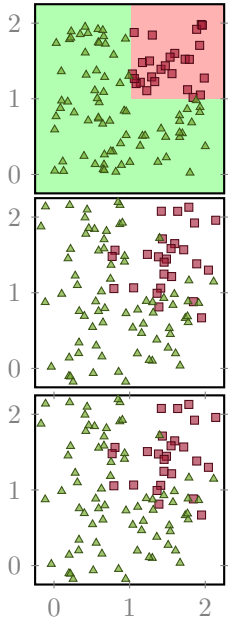


- 100 Samples 2D-Zufallsdaten gleichverteilt zwischen 0 und 2.
- Samples mit $x > 1$ und $y > 1$ gehören zu ■ der Rest zu ▲.

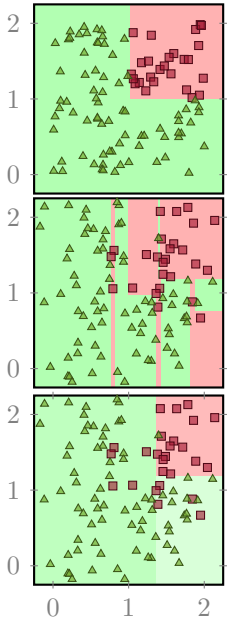


- 100 Samples 2D-Zufallsdaten gleichverteilt zwischen 0 und 2.
- Samples mit $x > 1$ und $y > 1$ gehören zu ■ der Rest zu ▲.
- Auf diesen Daten lernt CART einen kleinen Baum mit fast optimalen Entscheidungen

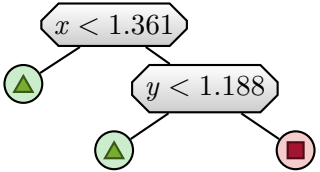


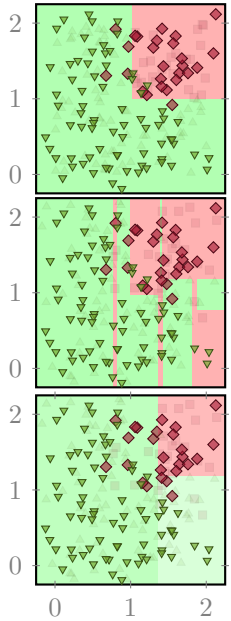


- 100 Samples 2D-Zufallsdaten gleichverteilt zwischen 0 und 2.
- Samples mit $x > 1$ und $y > 1$ gehören zu ■ der Rest zu ▲.
- Auf diesen Daten lernt CART einen kleinen Baum mit fast optimalen Entscheidungen
- Nun verschieben wir die Samples zufällig um jeweils $[-0.5, 0.5]$.

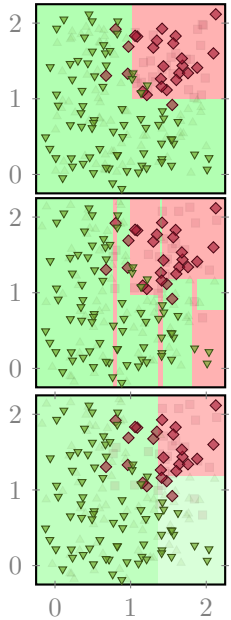


- 100 Samples 2D-Zufallsdaten gleichverteilt zwischen 0 und 2.
- Samples mit $x > 1$ und $y > 1$ gehören zu ■ der Rest zu ▲.
- Auf diesen Daten lernt CART einen kleinen Baum mit fast optimalen Entscheidungen
- Nun verschieben wir die Samples zufällig um jeweils $[-0.5, 0.5]$.
- Auf diesen Daten trainieren wir einen Baum ohne Einschränkung (Mitte) – ergibt Baum mit 25 Knoten – und einen mit der Einschränkung `max_depth=2` (unten), welcher 5 Knoten ergibt.





- 100 Samples 2D-Zufallsdaten gleichverteilt zwischen 0 und 2.
- Samples mit $x > 1$ und $y > 1$ gehören zu ■ der Rest zu ▲.
- Auf diesen Daten lernt CART einen kleinen Baum mit fast optimalen Entscheidungen
- Nun verschieben wir die Samples zufällig um jeweils $[-0.5, 0.5]$.
- Auf diesen Daten trainieren wir einen Baum ohne Einschränkung (Mitte) – ergibt Baum mit 25 Knoten – und einen mit der Einschränkung `max_depth=2` (unten), welcher 5 Knoten ergibt.
- Um zu testen wie gut die Bäume sind, verschieben wir die ursprünglichen Daten nochmals zufällig. Testdaten: ◆ und ▼



- 100 Samples 2D-Zufallsdaten gleichverteilt zwischen 0 und 2.
- Samples mit $x > 1$ und $y > 1$ gehören zu ■ der Rest zu ▲.
- Auf diesen Daten lernt CART einen kleinen Baum mit fast optimalen Entscheidungen
- Nun verschieben wir die Samples zufällig um jeweils $[-0.5, 0.5]$.
- Auf diesen Daten trainieren wir einen Baum ohne Einschränkung (Mitte) – ergibt Baum mit 25 Knoten – und einen mit der Einschränkung `max_depth=2` (unten), welcher 5 Knoten ergibt.
- Um zu testen wie gut die Bäume sind, verschieben wir die ursprünglichen Daten nochmals zufällig. Testdaten: ◆ und ▼
- Die Genauigkeiten auf diesen Testdaten sind: 91% (oben), 77% (Mitte), 87% (unten).
- Fazit: Mit mehr Knoten steigt die Genauigkeit auf einer ähnlichen Menge wie die Trainingsmenge nicht, sondern fällt hier sogar.

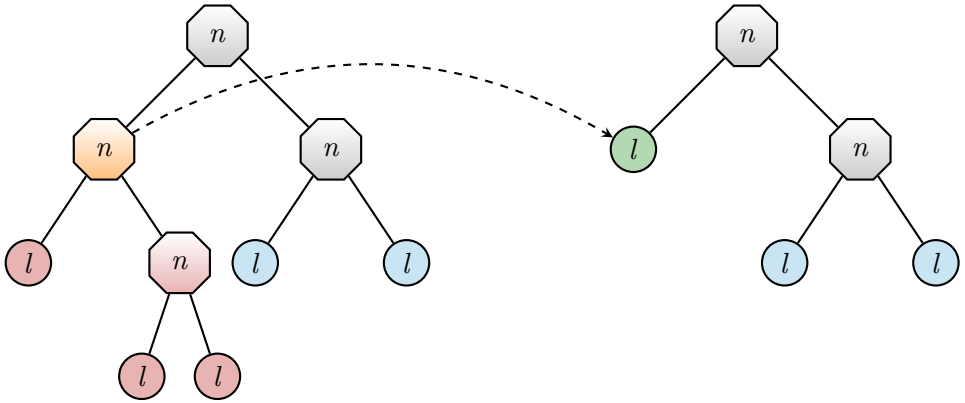
Pre-Pruning

- Overfitting wirkt sich im Allgemeinen negativ auf die Verallgemeinerung von Daten aus.
- Unser Ziel ist es also, diese unsinnige Verästelung zu reduzieren und den Baum auf die notwendige Komplexität zurechtzustutzen.
- Dies nennt man **Pruning**. Man unterscheidet zwischen **Pre-Pruning** und **Post-Pruning**.
- Beim **Pre-Pruning** wird eigentlich nichts gestutzt, sondern beim Aufstellen des Baumes zu starke Verästelung vermieden.
- Dabei werden Stopp-Kriterien (Einschränkungen) verwendet, wie beispielsweise:
  - `max_depth` (int or None); maximale Tiefe
  - `min_samples_split` (default=2); minimale Anzahl von Beispielen an einem Knoten für einen Split
  - `min_samples_leaf` (default=1); minimale Anzahl von Beispielen an einem Leaf-Node
  - `max_leaf_nodes` (int or default=None); maximale Anzahl an Leaf-Nodes im Baum
  - `min_impurity_decrease` (default=0.0) Split nur wenn die Reduktion des Fehlermaßes größer oder gleich diesem Parameter ist

Post-Pruning

- Beim **Post-Pruning** werden an einem fertigen Baum Knoten durch Blätter ersetzt, um die Komplexität zu verbessern.
- Die Beurteilung, was zurückgeschnitten wird, geschieht auch hierbei über eine Validierungsmenge.
- Ein einfacher und trotzdem sehr effektiven Ansatz ist der **Reduced-Error Ansatz**.
- Hierbei testet man einen Knoten innerhalb des Baumes darauf, wie sich der Fehler auf der Validierungsmenge verändert, wenn dieser Knoten durch ein Blatt ersetzt wird.
- Wenn der Fehler kleiner wird oder ggf. nur leicht größer, so wird der darunter liegende Teilbaum abgeschnitten und durch ein Blatt ersetzt.
- In scikit-learn ist dies nicht implementiert, aber dafür ein Cost-Complexity-Pruning (Parameter `ccp_alpha`), was im Prinzip ohne Validierungsmenge auskommt, aber um den Parameter zu wählen sollte man eine Validierungsmenge verwenden.

Post-Pruning illustriert



Komplexität

Nun bleibt die Frage wie der Algorithmus für größere Datenmengen in der Theorie skaliert? Für diese Frage muss man das Lernen und die Auswertung des Baumes unterscheiden.

- Sei n_S die **Anzahl Samples in der Trainingsmenge** und n_F die **Anzahl Merkmale**.
- Wir nehmen an, dass ein perfekt ausbalancierter Baum entsteht.
- Die Komplexität für eine Auswertung liegt bei $\mathcal{O}(\log(n_S))$, weil die Tiefe des Baums mit n_S logarithmisch wächst.
- Die Komplexität für das Training beträgt:

$$\mathcal{O}(n_F \cdot n_S^2 \cdot \log(n_S))$$ (scikit-learn Doku) mit Annahmen jedoch $\mathcal{O}(n_F \cdot n_S \cdot \log(n_S))$

Mit dem Parameter `splitter='random'` wird für jedes Feature nur ein zufällig gewählter Split getestet, was die Kosten stark senkt und gegen Overfitting helfen kann, aber auch das Ergebnis verschlechtern und den Baum größer werden lassen kann.

Was ist bei Regressionsproblemen anders?

- Im Fall einer Regression besteht ein Datensatz aus dem Featurevektor $\mathbf{x}$ und dem Zielwert $y = f(\mathbf{x}) \in \mathbb{R}$.
- Zwei Änderungen sind nötig für den Wechsel von der Klassifikation zur Regression:
  - Einmal brauchen wir einen Ersatz für die Gini Impurity und
  - zum anderen eine andere Art, den Wert an einem Blattknoten zu berechnen.
- Zunächst ersetzen wir die Gini Impurity durch die Varianz:

$$\sigma_y^2 = \frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y})^2 \text{ mit } \bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$$

- Der Wert am Blattknoten ist einfach der Mittelwert der Zielwerte $\bar{y}$. Über jeweils den Bereich, den ein Blattknoten abdeckt, ist die so enstandene Funktion also konstant.

Beispiel

- Als Beispiel verwenden wir das **Bike Sharing Data Set** (`https://archive.ics.uci.edu/ml/datasets/bike+sharing+dataset`).
- Es besteht aus 17389 protokollierten Daten über das Mietverhalten von Fahrrädern in einer Großstadt.
- Der Datenbestand enthält Daten bzgl. des Wetters und des Mietverhalten für Fahrräder protokolliert nach Uhrzeit, Feiertag usw.
- Die Originaldatei enthält die Informationen zum Datum bzw. der Uhrzeit teilweise redundant in einem Zeitstempel und einzelnen Spalteneinträgen.
- Nach einer Bereinigung haben wir 13 Merkmale übrig, die in der Tabelle auf der nächsten Seite notiert sind.

Merkmale des Bike Sharing Data Set (2013)

Nr.	Merkmal	Bedeutung	Wertebereich
0	season	Frühling(1), Sommer(2), Herbst(3), Winter(4)	$\{1, 2, 3, 4\}$
1	yr	Jahr 2011 (0) oder 2012 (1)	$\{0, 1\}$
2	mnth	Monat des Jahres	1 bis 12
3	day	Tag des Monats	1 bis 31
4	hr	Stunde des Tages	0 bis 23
5	holiday	Ist es ein Feiertag?	0 (False) oder 1 (True)
6	weekday	Welcher Wochentag (0: Sonntag)	$\{0, 1, 2, 3, 4, 5, 6\}$
7	workingday	Kein Wochenende und kein Feiertag?	0 (False) oder 1 (True)
8	weathersit	Qualität des Wetters in Abstufungen	$\{1, 2, 3, 4\}$
9	temp	Normierte Temperatur	$[0, 1]$
10	atemp	Normierte gefühlte Temperatur	$[0, 1]$
11	hum	Normierte relative Luftfeuchtigkeit	$[0, 1]$
12	windspeed	Normierte Windgeschwindigkeit	$[0, 1]$
13	casual	Anzahl Fahrräder von Gelegenheitsradlern	$\in \mathbb{N}$
14	registered	Anzahl Fahrräder von registrierten Nutzern	$\in \mathbb{N}$
15	cnt	Gesamtanzahl vermieteter Fahrräder	$\in \mathbb{N}$

CART am Beispiel des Bike-Sharing Data Set

Test- und Trainingset erzeugen und Modell trainieren

```
1 import pandas as pd
2 from sklearn.tree import DecisionTreeRegressor
3 from sklearn.model_selection import train_test_split
4
5 df = pd.read_csv('bikes.csv')
6 X = df.drop(columns=['instant', 'dteday', 'casual', 'registered', 'cnt'])
7 y = df.loc[:, 'cnt']
8 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
9
10 my_tree = DecisionTreeRegressor(min_samples_leaf=5) # CART für Regression erzeugen
11 my_tree.fit(X_train, y_train)                       # Baum mit Trainingsdaten lernen
```

Der Wert `min_samples_leaf` gibt an wie viele Elemente mindestens in einem Leaf-Knoten zur Bestimmung des Ausgabewertes verbleiben sollen.

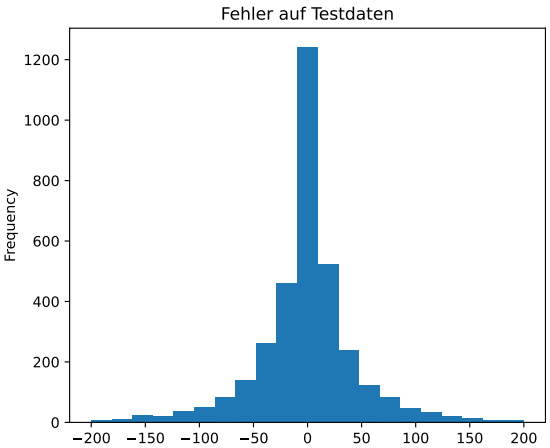
CART am Beispiel des Bike-Sharing Data Set

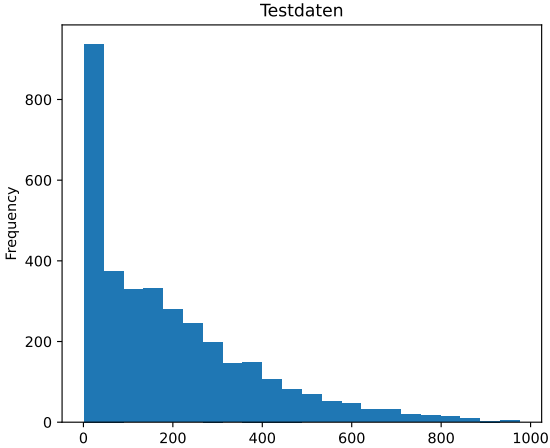
Modell auswerten und visualisieren

```
13 y_predict = my_tree.predict(X_test)                    # Baum für die Testmenge auswerten
14 y_diff = y_predict - y_test                            # Fehler für Testdaten berechnen
15 print('Mittlerer y-Wert: {:.2f}'.format(y_test.mean()))          # z. B. 186.41
16 print('Mittlere Abweichung: {:.2f}'.format(y_diff.abs().mean())) # z. B. 30.86
17
18 import matplotlib.pyplot as plt
19 fig1 = plt.figure()                                    # Werte-Histogramm
20 y_test.plot.hist(bins=22, title='Testdaten')
21
22 fig2 = plt.figure()                                    # Fehler-Histogramm
23 y_diff.plot.hist(bins=21, range=(-200, 200), title='Fehler auf Testdaten')
```

Mittlerer y-Wert z. B. 186.41
Mittlere Abweichung z. B. 30.86
D. h. ca. 16.6% relativer Fehler bzgl. der Mittelwerte

Ergebnisse für den CART-Baum

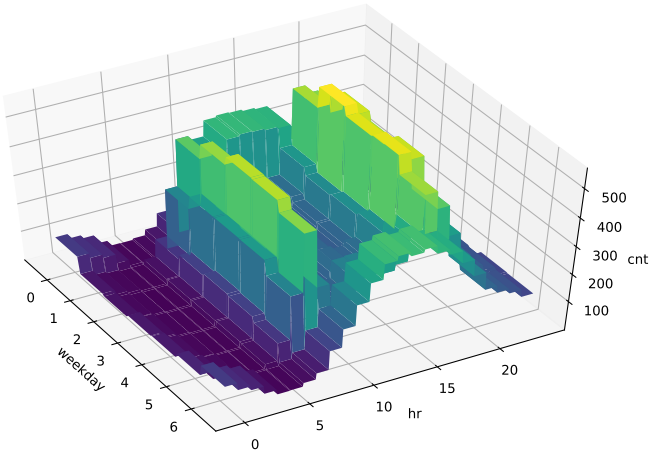




Wir erhalten hier einen mittleren Fehler von ca. 31 Fahrrädern, um die sich die Vorhersage auf der Testmenge verschätzt.

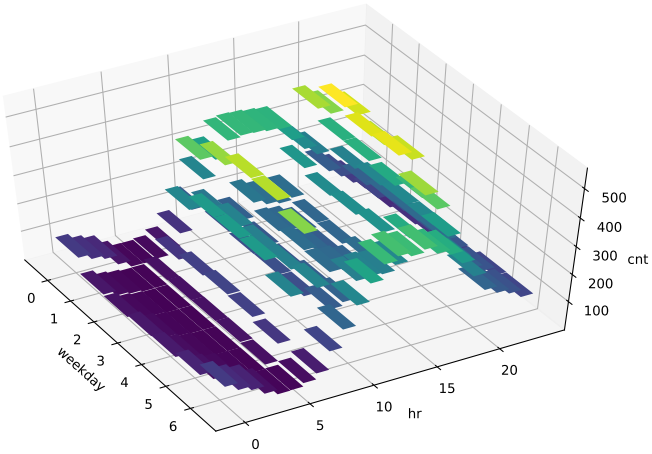
Stückweise konstante Funktion – Illustration

Hier wurden nur Stunde und Wochentag verwendet.
Man sieht, dass die entstandene Funktion stückweise konstant ist.
Der vorhergesagte Wert für *cnt*, ist jeweils der Mittelwert der *cnt*-Werte in jedem Bereich.

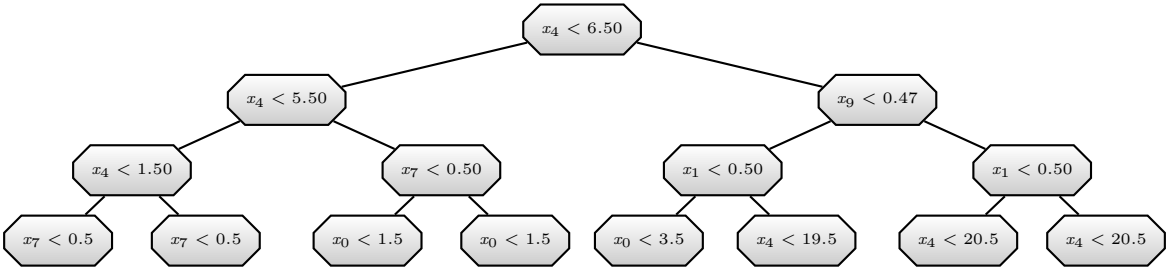


Stückweise konstante Funktion – Illustration

Hier wurden nur Stunde und Wochentag verwendet.
Man sieht, dass die entstandene Funktion stückweise konstant ist.
Der vorhergesagte Wert für *cnt*, ist jeweils der Mittelwert der *cnt*-Werte in jedem Bereich.



Erste Ebenen des CART Entscheidungsbaumes



- Wie man am oberen Ende des Baumes sieht, ist die *Uhrzeit* – Merkmal 4 – ein sehr wichtiger Aspekt.
- Ansonsten spielen die *Jahreszeit* (Nr. 0), das *Jahr* (Nr. 1), *Feiertag* (ja oder nein) (Nr. 7) und die *Temperatur* (Nr. 9) eine große Rolle.
- Andere Merkmale kommen erst weiter unten als Feinabstimmung im Baum vor.

Einordnung der Ergebnisse

- Wie man sieht, schwankt die Anzahl der vermieteten Fahrräder beträchtlich.
- Im Mittel ist unser Ergebnis eigentlich sehr gut, was jedoch daran liegt, wie wir unser Testset gebildet haben.
- Unsere Testmenge wurde zufällig aus der Gesamtmenge der Daten gezogen.
- Das bedeutet, dass zur Trainingsmenge zum Beispiel die vermieteten Fahrräder an einem bestimmten Tag um 09:00, 10:00 und 12:00 gehören und die Testmenge den Wert um 11:00 enthält.
- Eine solche Interpolation auf zeitlichen Daten ist wesentlich leichter als eine Extrapolation.

Hinweis auf die spätere Aufgabenstellung

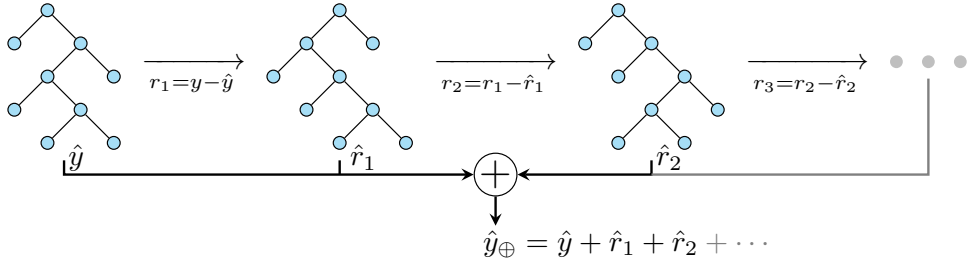
In der selbstständigen Übung zum Random Forest Algorithmus, welchen wir gleich besprechen, wird die Aufgabe u. a. darin bestehen eine realistischere Vorhersage umzusetzen.

Ensemble Learning – Boosting

Beim **Ensemble Learning** werden mehrere Lerner zusammengeschaltet. Dazu gibt es zwei Ansätze: Boosting und Mitteln.

Boosting-Verfahren:

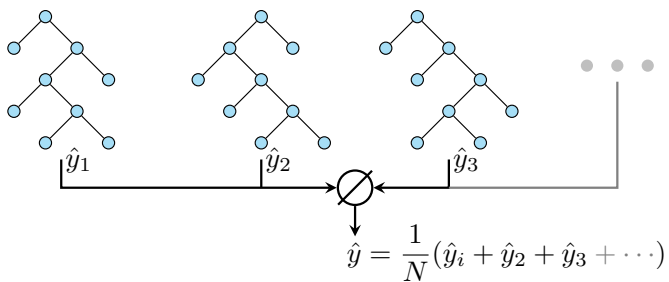
- Beim **Boosting** werden mehrere Lerner hintereinander geschaltet.
- Jeder neue Lerner versucht den Restfehler der vorangegangen Lerner zu reduzieren.
- Eine Iteration entspricht einem neuen Lerner. Am Ende werden die Ergebnisse addiert.
- Die einzelnen Lerner selbst sind recht schwach (geringe Maximale Tiefe).



Ensemble Learning – Mitteln

Mittelnde-Verfahren:

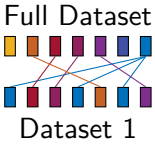
- Beim **Mitteln** werden mehrere Lerner parallel geschaltet.
- Die Lerner werden unabhängig voneinander trainiert und ausgewertet.
- Die mittlere Vorhersage aller Lerner ist normalerweise wesentlich besser und robuster (kaum Overfitting), als die Vorhersagen der einzelnen Lerner.
- Die einzelnen Lerner sind recht stark (z. B. volle Tiefe).
- Gut parallelisierbar.
- Bei einer Klassifikation erfolgt die Mittelung über die Wahrscheinlichkeiten.



Ensemble Learning – Bagging

Bagging-Verfahren:

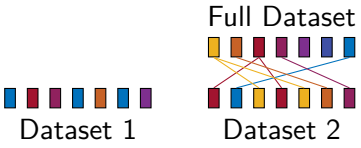
- **Bagging** (von **B**oostrap **Agg**regat**ing**) ist eine Unterart der mittelnden Verfahren.
- Ziel ist, dass die Lerner sich zufällig unterscheiden, denn ein Mittelwert aus N mal demselben Lerner brächte überhaupt keinen Vorteil!
- Die Lerner werden mit zufällig gezogenen Teilmengen der Gesamtdatenmenge trainiert.
- Dabei können Samples mehrfach gezogen werden und wenn die Teilmenge genauso groß wie die Gesamtmenge ist, sind ca. 36.8% der Daten nicht enthalten.
- Der **Random Forest** Algorithmus ist ein Bagging-Verfahren.



Ensemble Learning – Bagging

Bagging-Verfahren:

- **Bagging** (von **B**oostrap **Agg**regat**ing**) ist eine Unterart der mittelnden Verfahren.
- Ziel ist, dass die Lerner sich zufällig unterscheiden, denn ein Mittelwert aus N mal demselben Lerner brächte überhaupt keinen Vorteil!
- Die Lerner werden mit zufällig gezogenen Teilmengen der Gesamtdatenmenge trainiert.
- Dabei können Samples mehrfach gezogen werden und wenn die Teilmenge genauso groß wie die Gesamtmenge ist, sind ca. 36.8% der Daten nicht enthalten.
- Der **Random Forest** Algorithmus ist ein Bagging-Verfahren.



Ensemble Learning – Bagging

Bagging-Verfahren:

- **Bagging** (von **B**oostrap **Agg**regat**ing**) ist eine Unterart der mittelnden Verfahren.
- Ziel ist, dass die Lerner sich zufällig unterscheiden, denn ein Mittelwert aus N mal demselben Lerner brächte überhaupt keinen Vorteil!
- Die Lerner werden mit zufällig gezogenen Teilmengen der Gesamtdatenmenge trainiert.
- Dabei können Samples mehrfach gezogen werden und wenn die Teilmenge genauso groß wie die Gesamtmenge ist, sind ca. 36.8% der Daten nicht enthalten.
- Der **Random Forest** Algorithmus ist ein Bagging-Verfahren.

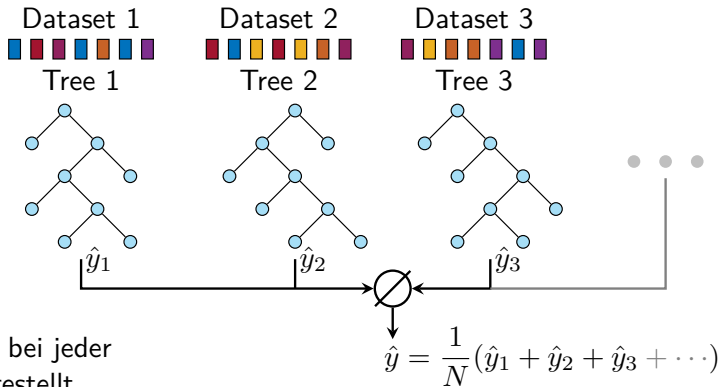
Ensemble Learning – Bagging

Bagging-Verfahren:

- **Bagging** (von **B**oostrap **Agg**regat**ing**) ist eine Unterart der mittelnden Verfahren.
- Ziel ist, dass die Lerner sich zufällig unterscheiden, denn ein Mittelwert aus N mal demselben Lerner brächte überhaupt keinen Vorteil!
- Die Lerner werden mit zufällig gezogenen Teilmengen der Gesamtdatenmenge trainiert.
- Dabei können Samples mehrfach gezogen werden und wenn die Teilmenge genauso groß wie die Gesamtmenge ist, sind ca. 36.8% der Daten nicht enthalten.
- Der **Random Forest** Algorithmus ist ein Bagging-Verfahren.

Random Forest

- Der **Random Forest** Algorithmus wurde von Breiman im Jahr 2001 publiziert.
- Wie unterscheidet sich Random Forest vom reinen Bagging mit CART-Bäumen?
- Für noch mehr Diversität kann beim Random Forest jeweils beim Lernen einer Entscheidung auf einer zufälligen Teilmenge der Features nach dem besten Schnitt gesucht werden. Diese Teilmenge wird bei jeder Entscheidung neu zusammengestellt.



Pseudocode für den Random Forest

Sei m die Anzahl an Merkmalen.
Wähle die Anzahl N von Bäumen, welche der Random Forest umfassen soll.

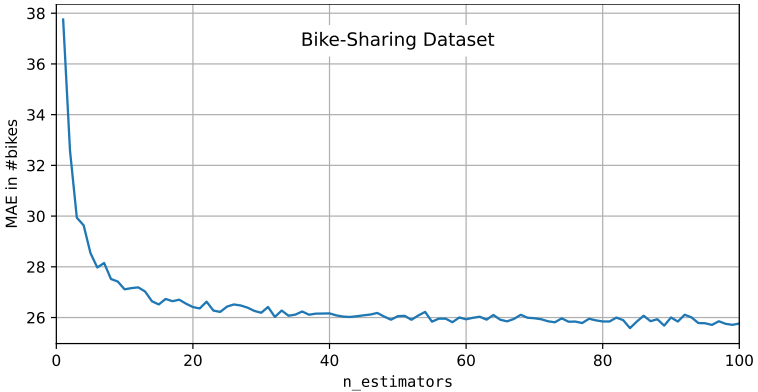
1: **for** $i = 1$ **to** N **do**
2: Erzeuge neue Bootstrap-Trainingsmenge $\tilde{D}_i$.
3: **repeat** ▷ (CART mit zufälligen Merkmalen)
4: Wähle $\tilde{m} \leq m$ Merkmale aus.
5: Lerne eine Entscheidung ▷ wie bei CART.
6: **until** Stop-Bedingung erreicht.
7: **end for**

Wahl von $\tilde{m}$ bzw. `max_features`

In der Literatur wird als recht robuster Ansatz oft die Wahl $\tilde{m} = \text{int}(\sqrt{m})$ (default für Klassifikation) oder $\tilde{m} = \text{int}(\log_2(m))$ angegeben. Bei der Regressionen wird hingegen oft $\tilde{m} = m$ genutzt (default für Regression), was aber häufig nicht optimal ist.

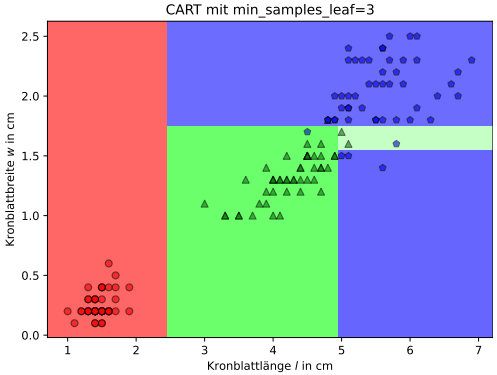
Anzahl Bäume N bzw. `n_estimators`

- Generell kann man sagen, dass mehr Bäume besser für die Vorhersage sind, aber bei genug Bäumen (Default-Wert: 100) ist die Verbesserung nicht mehr messbar.
- Die Trainings- und Auswertungszeit hängt linear von der Anzahl der Bäume ab.



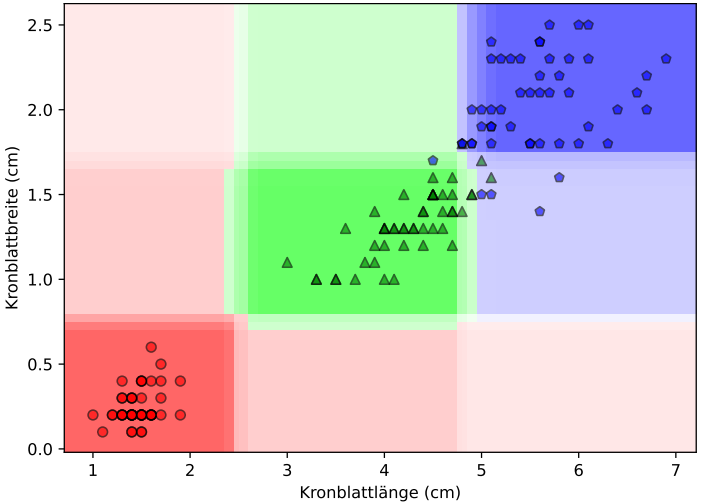
- Die Notwendigkeit zum Pruning ist geringer als bei CART bzw. ein Overfitting kaum möglich.

Realistischere Wahrscheinlichkeiten



- Der erste Schnitt könnte entlang der Kronblattlänge (x-Achse) oder der Kronblattbreite (y-Achse) erfolgen und würde in beiden Fällen die rote Gruppe separieren.
- In einem Random Forest werden beide Schnitte vorkommen, da die Reihenfolge der Features immer zufällig ist bzw. auch nicht immer alle zur Verfügung stehen.
- Das bedeutet, die Menge unterschiedlicher Bäume hilft uns das Problem zu adressieren, das Merkmale implementierungsabhängig als erstes ausgewählt werden.
- Bei einem einzelnen Baum würde der gesamte rote Bereich immer die rote Klasse mit einer 100%igen Wahrscheinlichkeit wählen. Das ist bei einem Random Forest nicht so.

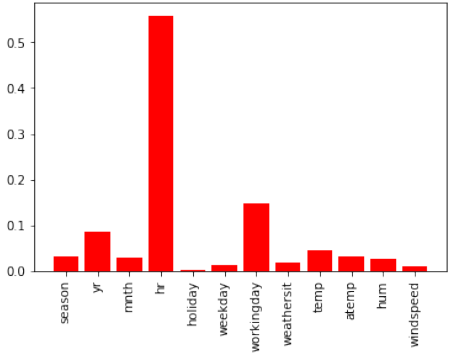
Ergebnis aller Bäume



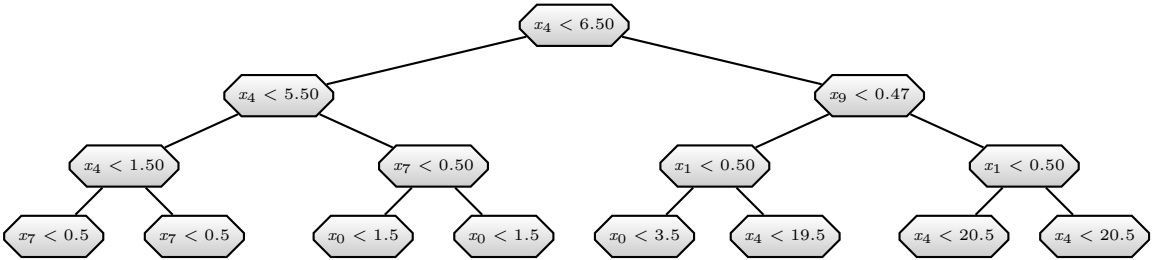
- Wenn alle Bäume verwendet werden lassen sich Unsicherheiten besser greifen.
- Links dargestellt sind die Gini-Impurities der Ergebnisse eines Punktes. Bei N Bäumen gibt es N Ergebnisse welcher Klasse der Punkt angehört.
- Alternativ lassen sich die Klassenwahrscheinlichkeiten der Bäume mitteln um eine ähnliche Aussage zu erhalten.

Random Forest zur Feature Analyse

- Bei einer geeigneten Implementierten wie z. B. in *scikit-learn* kann man den Random Forest auch zur Feature Analyse nutzen
- Wenn man auswertet, wie oft und an welcher Stelle ein Merkmal in den unterschiedlichen Bäumen ausgewählt wurde, kann man deren Bedeutung bewerten
- Welchen Vorteil gibt es bzgl. der Bedeutung eines Merkmals, wenn man den Random Forest einsetzt statt dem einzelnen CART-Baum?
- Egal ob es um einen einzelnen Baum geht oder einen ganzen Wald. Wenn wir uns nur auf "oben im Baum" als Kriterium verlassen würden, wäre dies zu kurz gegriffen.
- Können Sie sich vorstellen warum?



Ausschnitt eines Baums



- Nehmen wir einmal die zweite Ebene des bekannten Baumes.
- Es ist keinesfalls klar, dass der Baum als Ganzes balanciert ist oder dass genauso viele Trainingsbeispiele rechts wie links angefertigt werden.
- Das bedeutet, dass die Knoten unabhängig von ihrer Lage im Baum unterschiedlich viel dazu beitragen das Fehlermaß zu verringern bzw. die Impurity zu senken.

Feature Importance

- Die Standardmethode zur Berechnung der *feature importance* ist die *mean decrease in impurity*, auch Gini-Bedeutung genannt.
- Die Grundidee ist, bei jeder Teilung im Baum die Verbesserung im Verunreinigungsmaß dem Merkmal zuzuschreiben, welches verwendet wurde.
- Abschließend werden die Ergebnisse über alle Bäume im Wald gemittelt.
- Ein Vorteil ist, dass die Arbeit für die *Feature Importance* quasi eh beim Erstellen der Bäume des Random Forest anfällt.
- Die Feature Importance muss man allerdings kritisch hinterfragen[1].
- Ein bekanntes Problem ist, dass durch den CART-Algorithmus Merkmale, die viele Schnittmöglichkeiten haben (z. B. kontinuierliche Merkmale), bevorzugt werden.

[1] u. a. Altmann, A. et al 2010 *Permutation importance: a corrected feature importance measure*

- Durch diesen Effekt wird die Feature Importance von Merkmalen, die viele mögliche Schnittpunkte beinhalten aus statistischen Gründen tendenziell überbewertet gegenüber solchen mit nur wenigen Kategorien oder im Extremfall binären Merkmalen.
- Ein weiteres Problem ist, dass die Feature Importances verschiedener Merkmale nichts darüber aussagen, ob diese untereinander korrelieren bzw. abhängig sind.
- Beispielsweise werden ggf. die Variabel Gewicht, Länge und Breite eines Schiffes für eine Aussage als besonders wichtig eingeschätzt.

- Diese Werte sind jedoch eher gemeinsam ein Merkmal für *Schiffsgröße*. Wir werden später darauf eingehen.
- Die Merkmale sind allein betrachtet wichtig, würden aber zusammen nicht unbedingt die optimale Basis für unseren Merkmalsraum darstellen.

California Housing Price Dataset (Data: 1990, Paper: 1997)

Wir nutzen nun den bekannten Iris Dataset und den im folgenden beschriebenen California Housing Dataset um uns die Feature Importance genauer anzuschauen.

	Name	Beschreibung
X	MedInc	median income in block group
	HouseAge	median house age in block group
	AveRooms	average number of rooms per household
	AveBedrms	average number of bedrooms per household
	Population	block group population
	AveOccup	average number of household members
	Latitude	block group latitude
	Longitude	block group longitude
y	MedHouseVal	median house value in \$100 000

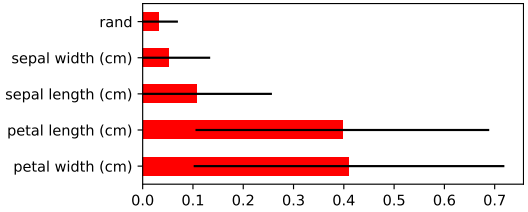
Random Forest in scikit-learn

```
1 import numpy as np
2 import pandas as pd
3 from sklearn.datasets import fetch_california_housing, load_iris
4 from sklearn.ensemble import RandomForestRegressor, RandomForestClassifier
5
6 if True: # True: iris, False: california
7     X, y = load_iris(as_frame=True, return_X_y=True)                    # Klassifikation
8     rf = RandomForestClassifier(n_estimators=100)
9 else:
10    X, y = fetch_california_housing(as_frame=True, return_X_y=True)   # Regression
11    rf = RandomForestRegressor(n_estimators=100)
12
13 # Nun erzeugen wir Zufallszahlen als Testmerkmal für die Feature Importance
14 X.loc[:, 'rand'] = np.random.randint(100, size=y.size) / 10
15 rf.fit(X, y)
```

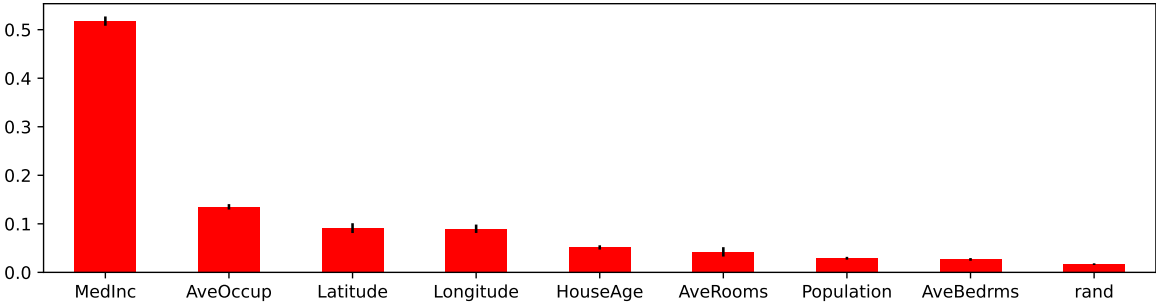
Das Bsp. dient nur der Illustration der Feature Importance. Daher: keine Aufteilung der Daten.

```
17 #%% Feature Importance sortiert ausgeben
18 importances = pd.Series(rf.feature_importances_, index=X.columns)
19 importances.sort_values(ascending=False, inplace=True)
20 print(importances)
21
22 # Wir wollen die Standardabweichung der Importance über alle Bäume darstellen
23 all_importances = pd.DataFrame(columns=X.columns)
24 for i, tree in enumerate(rf.estimators_):
25     all_importances.loc[i] = tree.feature_importances_
26
27 ax = importances.plot.barh(xerr=all_importances.std(), color='r', xlim=(0, None))
28 ax.get_figure().tight_layout()
```

Feature	Importance
petal width (cm)	0.410351
petal length (cm)	0.397090
sepal length (cm)	0.107896
sepal width (cm)	0.052988
rand	0.031676



Die große Standardabweichung resultiert aus der Einschränkung der Merkmale auf 2.

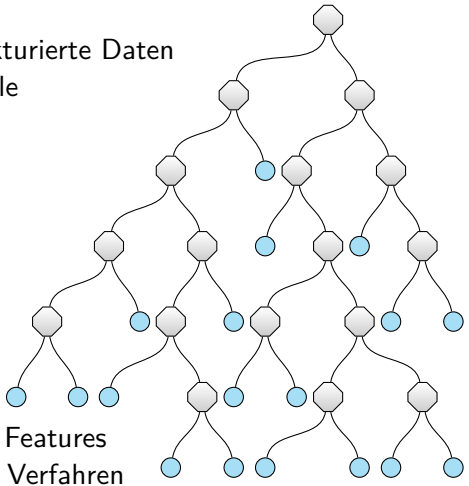


- Auch beim *California Housing Dataset* ist der Zufall das unwichtigste Merkmal.
- Aber: Er hat eine positive Feature Importance. Das liegt vielleicht an den unteren Verästelungen der Bäume, bei denen alle Features gleichwertig trennen.
- Es gibt auch Fälle, bei denen echte Features noch unwichtiger sind als der Zufall. Das sind häufig binäre Features bei denen in den unteren Verästelungen keine Trennmöglichkeit besteht.

Zusammenfassung zu CART und Random Forest

Gemeinsame Eigenschaften:

- gut für strukturierte Daten, eher nicht für unstrukturierte Daten
- Größenskalen der Eingangsdaten spielen keine Rolle
- nur achsparallele Schnitte
- stückweise konstantes Modell
- Pre-Pruning kann Overfitting vermeiden



CART bzw. Entscheidungsbäume:

- bei geringer Tiefe sehr einfach interpretierbar

Random Forest:

- durch Ensemble Learning robust und zuverlässig
- Feature Importances geben Hinweise auf wichtige Features
- sehr einfach zu benutzendes, aber trotzdem gutes Verfahren
  $\Longrightarrow$ sollte immer ausprobiert werden!