Data Science 1

Vorlesung 5 - Entscheidungsbäume

Prof. Dr. Christian Bockermann

Hochschule Bochum

Wintersemester 2025/2026

Fokus: Binäre Klassifikation

- Foliensatz 5 fokussiert sich auf binäre Klassifikation
- Gegeben ist also ein Datensatz $\mathbf{X} \times \mathbf{y}$, bei dem jedem Beispiel eine von zwei möglichen Klassen zugeordnet ist.

Wir nutzen die Iris Daten, allerdings **ohne** die Klasse **`setosa`**:

```
iris = pd.read_csv('data/iris.csv')

# Klasse 'setosa' herausfiltern:
iris = iris[ iris['species'] != 'setosa' ]
```

Alle Beispiele in diesem Foliensatz beziehen sich auf diese gefilterten Daten!

Einfacher Classifier: Zufall

- Zufällig Klasse aus Trainingsdaten $\mathbf{X}_{train} \times \mathbf{y}_{train}$ wählen
- Wahrscheinlichkeit für Vorhersage der Klasse C

$$P(\hat{y} = C) = \frac{\text{Häufigkeit von } C \text{ in } \mathbf{y}_{train}}{|\mathbf{y}_{train}|}$$

Einfacher Classifier: Zufall

- Zufällig Klasse aus Trainingsdaten $\mathbf{X}_{train} \times \mathbf{y}_{train}$ wählen
- Wahrscheinlichkeit für Vorhersage der Klasse C

$$P(\hat{y} = C) = \frac{\text{Häufigkeit von } C \text{ in } \mathbf{y}_{train}}{|\mathbf{y}_{train}|}$$

- Binäre Klassifikation, bei Gleichverteilung der Klassen führt zu durchschnittlichem Fehler von ~ 0.5

Einfacher Classifier: Zufall

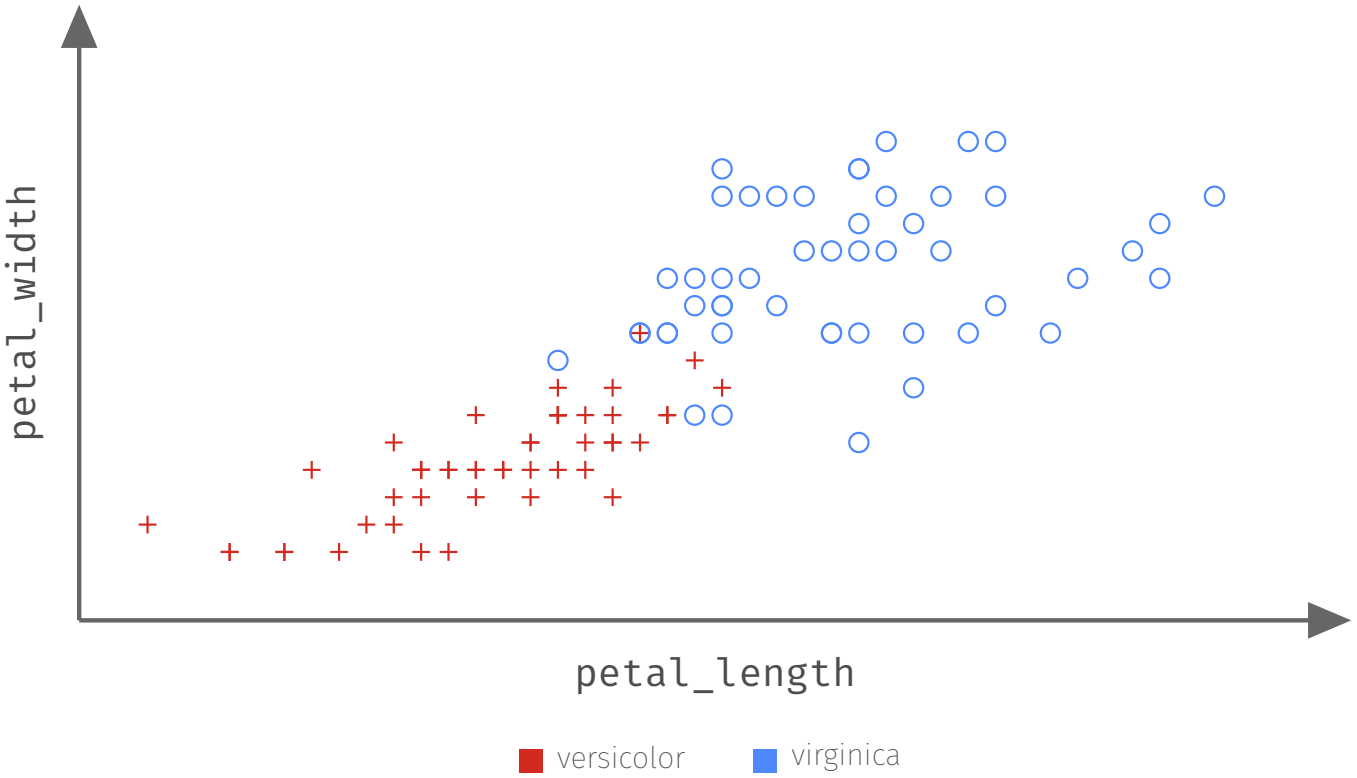
- Zufällig Klasse aus Trainingsdaten $\mathbf{X}_{train} \times \mathbf{y}_{train}$ wählen
- Wahrscheinlichkeit für Vorhersage der Klasse C

$$P(\hat{y} = C) = \frac{\text{Häufigkeit von } C \text{ in } \mathbf{y}_{train}}{|\mathbf{y}_{train}|}$$

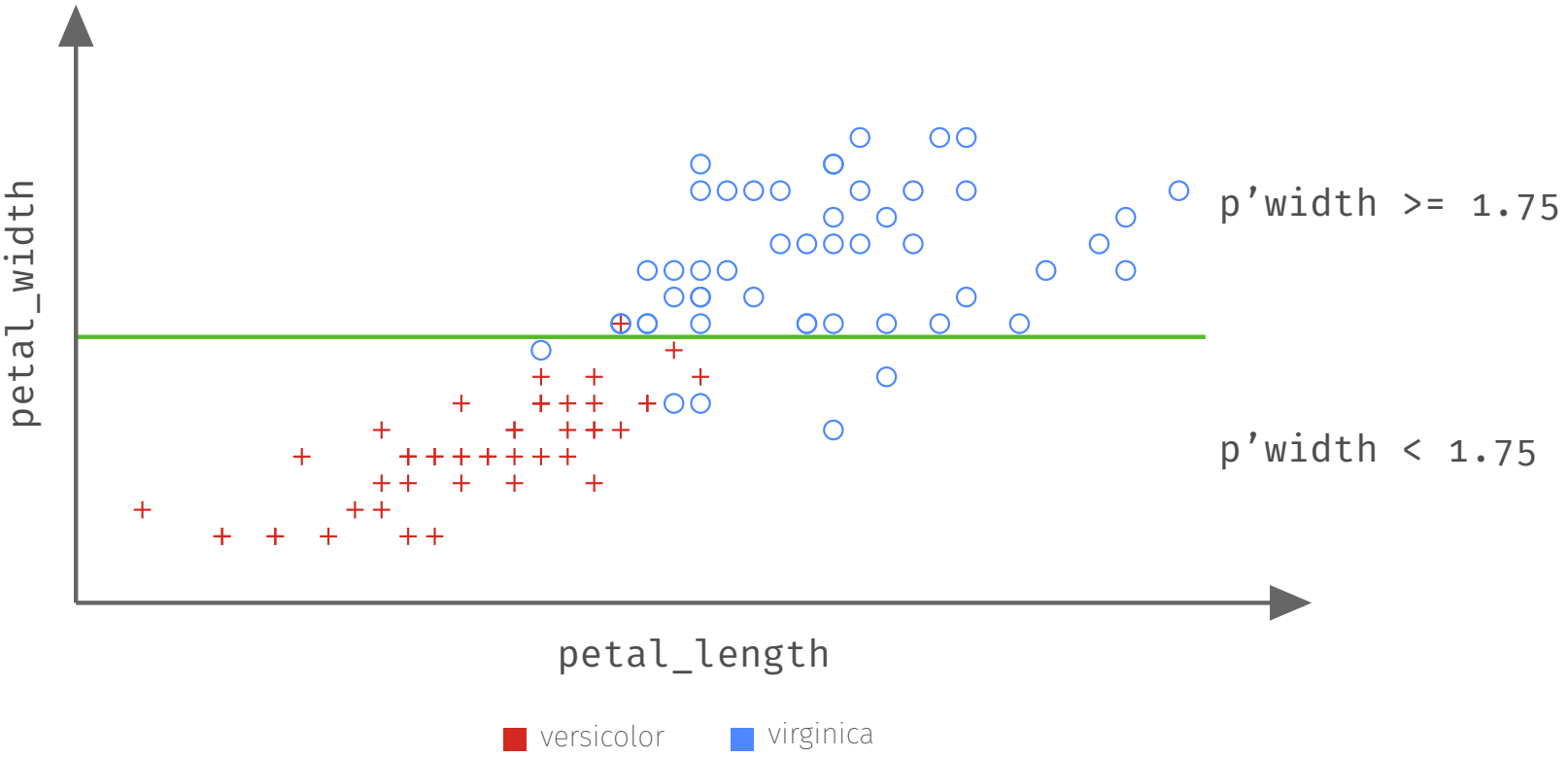
- Binäre Klassifikation, bei Gleichverteilung der Klassen führt zu durchschnittlichem Fehler von ~ 0.5

Wie können wir die Vorhersage verbessern?

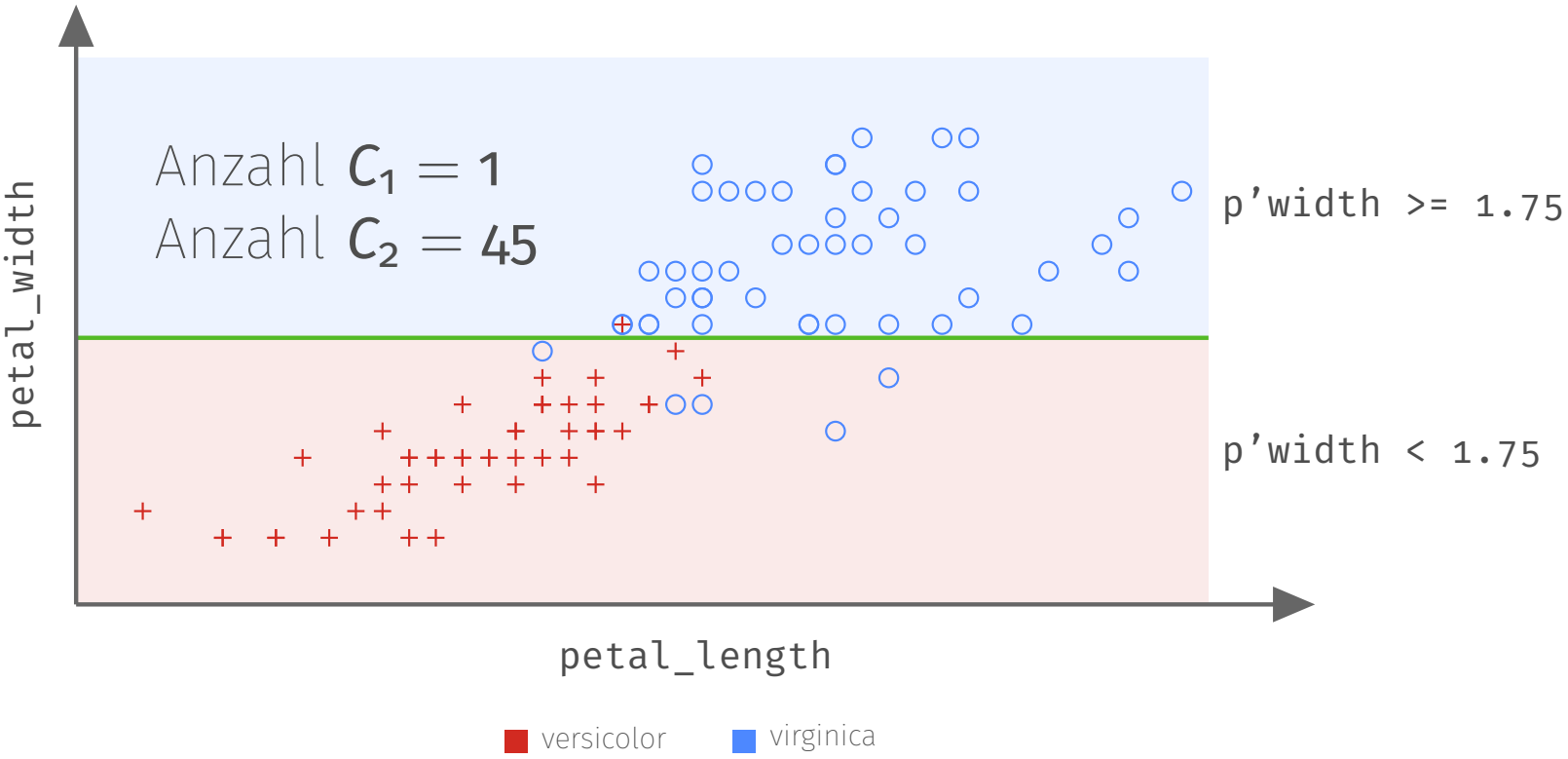
Idee: Daten teilen und W'keit für korrekte Vorhersage erhöhen



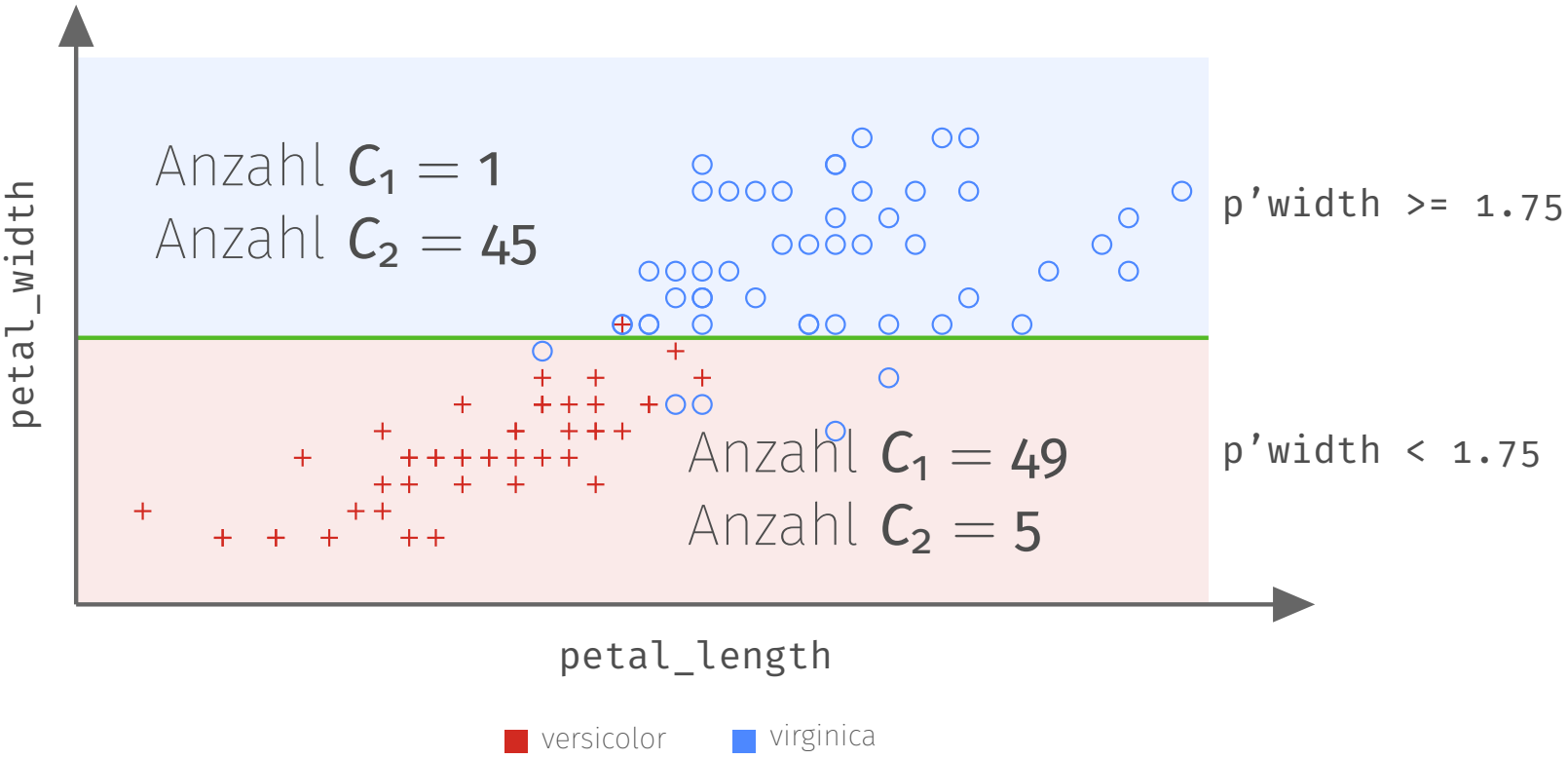
Idee: Daten teilen und W'keit für korrekte Vorhersage erhöhen



Idee: Daten teilen und W'keit für korrekte Vorhersage erhöhen



Idee: Daten teilen und W'keit für korrekte Vorhersage erhöhen



Einfache Vorhersage-Funktion

Die folgende Funktion berechnet die Vorhersage mit der Entscheidungsfunktion

$$f(x) = \begin{cases} \text{versicolor} & \text{, falls } x[petal\_width] >= 1.75 \\ \text{virginica} & \text{, sonst.} \end{cases}$$

```
def _predict(row):
  if row["petal_width"] >= 1.75:
    return "versicolor"
  else:
    return "virginica"

def simple_predict(X):
  return [_predict(x) for i,x in X.iterrows()]
```

Die Funktion `simple_predict` …

- ist ein statisch definiertes Modell (kein Training)
- dient lediglich dem Verständnis
- lädt ein zum Experimentieren `;-)`

Fragen:

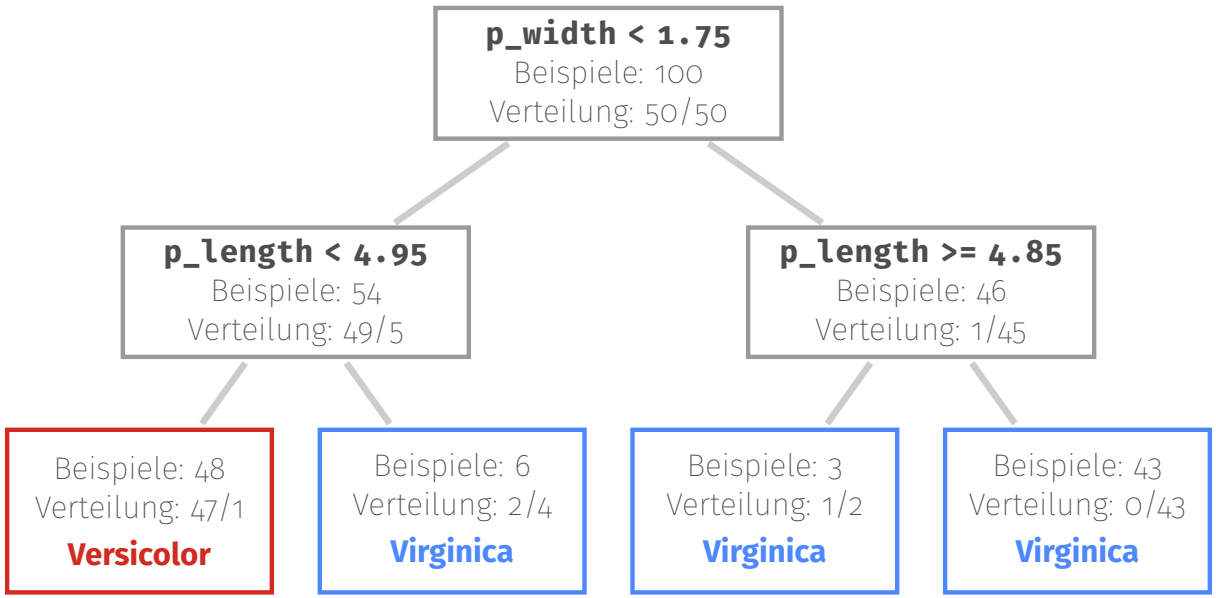
- Wie groß ist der Fehler der Funktion auf dem Iris Datensatz?
- Wie verändert sich der Fehler für andere Werte als 1.75?

◀ Probieren Sie es im Notebook aus!

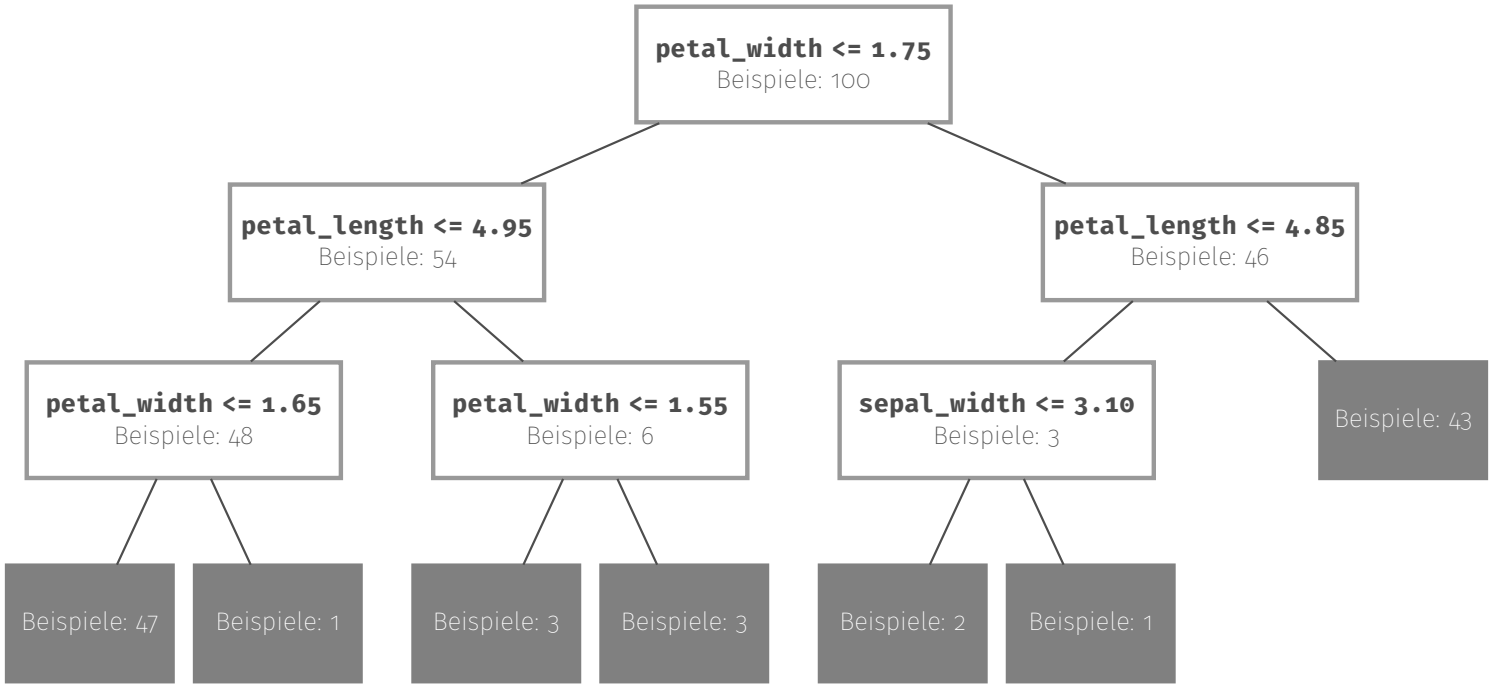
Notebook: `Vorlesung/V5-Simple-Classifier`

Entscheidungsbäume sind einfaches Modell

- Innere Knoten sind Entscheidungsknoten
- Blätter stellen Vorhersage dar

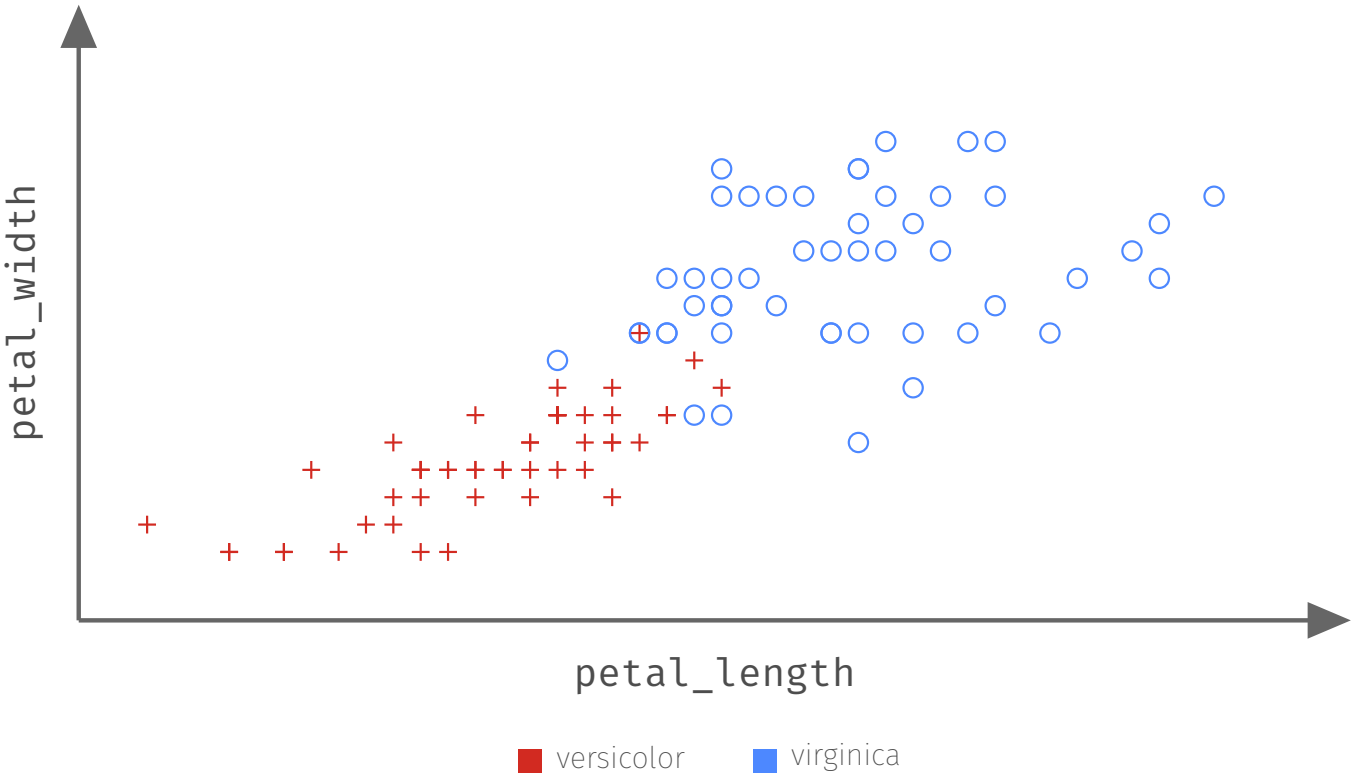


Ein weiterer Baum und die Aufteilung der Daten



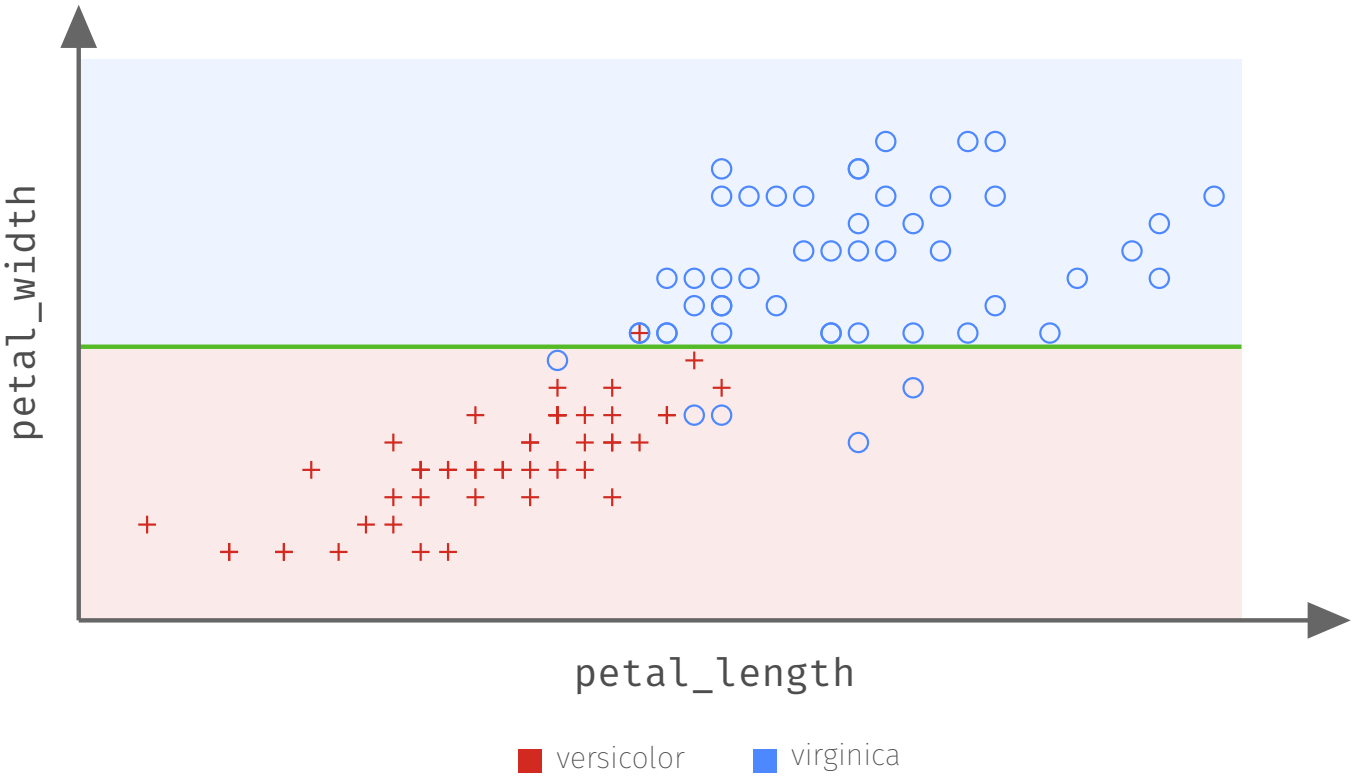
In den Blättern ist zu sehen, dass dort teilweise nur sehr wenige Beispiele landen!

Beispiel: Aufteilung der Daten durch einen Baum



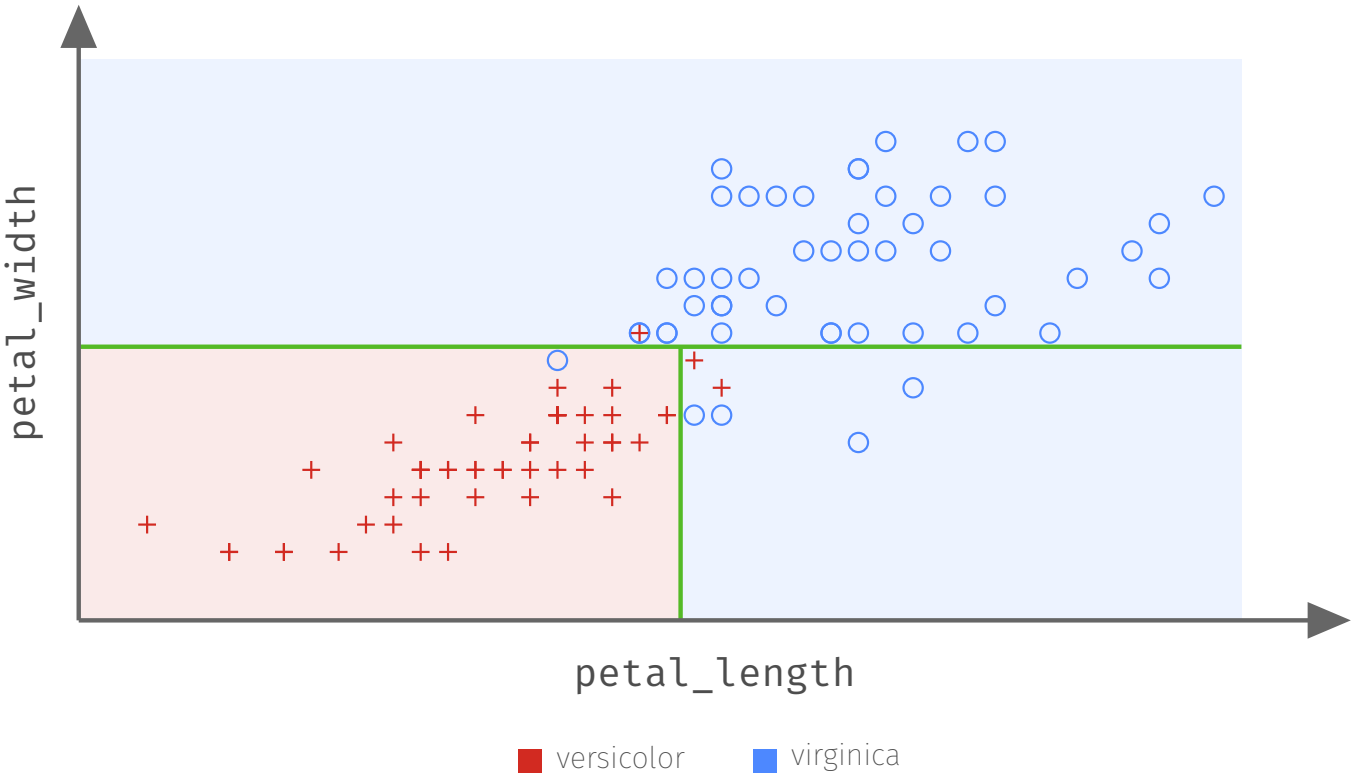
Das Beispiel zeigt schrittweise die Aufteilung des Iris-Datensatzes über die Attribute `petal_width` und `petal_length`.

Beispiel: Aufteilung der Daten durch einen Baum



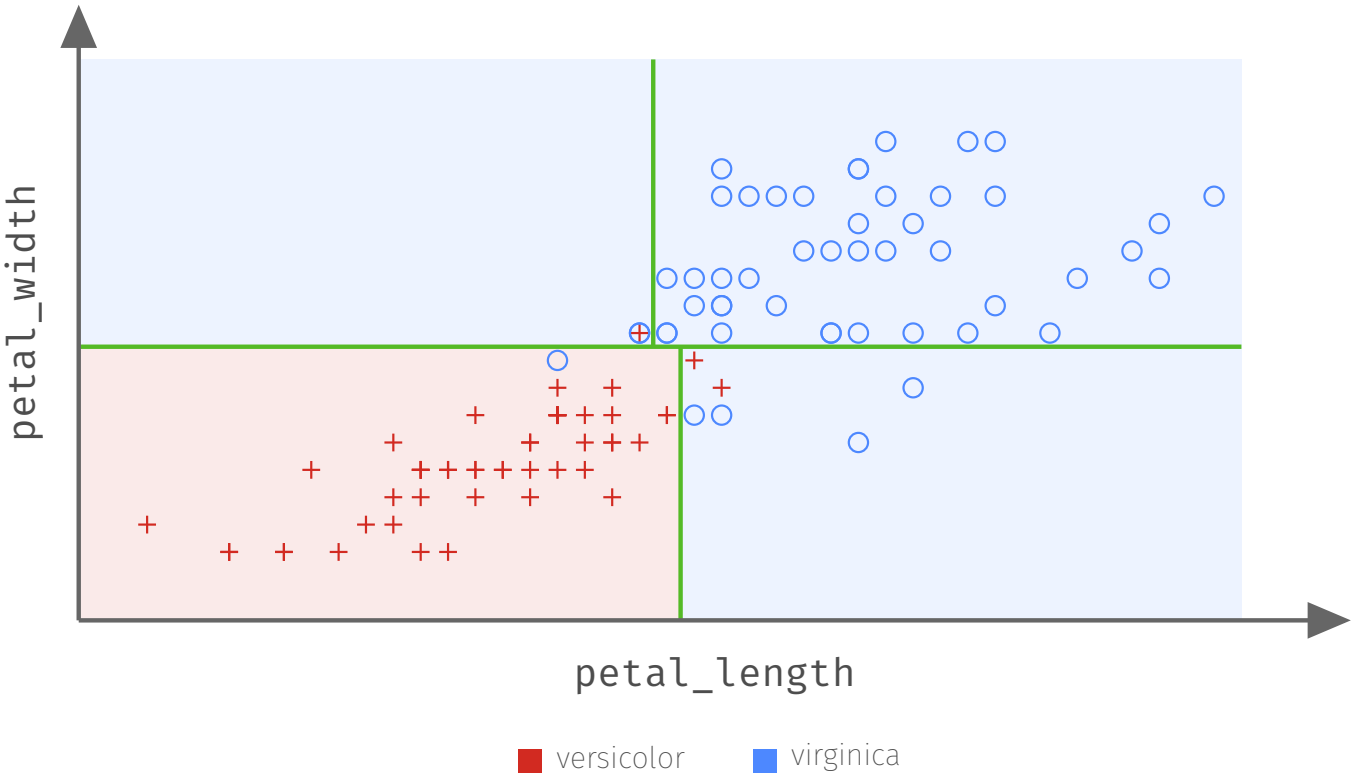
Das Beispiel zeigt schrittweise die Aufteilung des Iris-Datensatzes über die Attribute `petal_width` und `petal_length`.

Beispiel: Aufteilung der Daten durch einen Baum



Das Beispiel zeigt schrittweise die Aufteilung des Iris-Datensatzes über die Attribute `petal_width` und `petal_length`.

Beispiel: Aufteilung der Daten durch einen Baum



Das Beispiel zeigt schrittweise die Aufteilung des Iris-Datensatzes über die Attribute `petal_width` und `petal_length`.

Entscheidungsbäume

Split-Kriterien

Frage: Wie finden wir gute Aufteilung der Daten?

- Gegeben sind die Trainingsdaten $\mathbf{X} = \mathbf{X}_{train} \times \mathbf{y}_{train}$
- Wir wollen einen Baum erstellen, der den Trainingsfehler minimiert
- Schrittweises Vorgehen (rekursiv):

Training eines Baumes:

1. Finde bestes Attribut a^* und Wert t zum Aufteilen von $\mathbf{X}$
2. Teile $\mathbf{X}$ in $\mathbf{X}_{a \leq t}$ und $\mathbf{X}_{a > t}$

Frage: Wie finden wir das beste Attribut a und Wert t ?

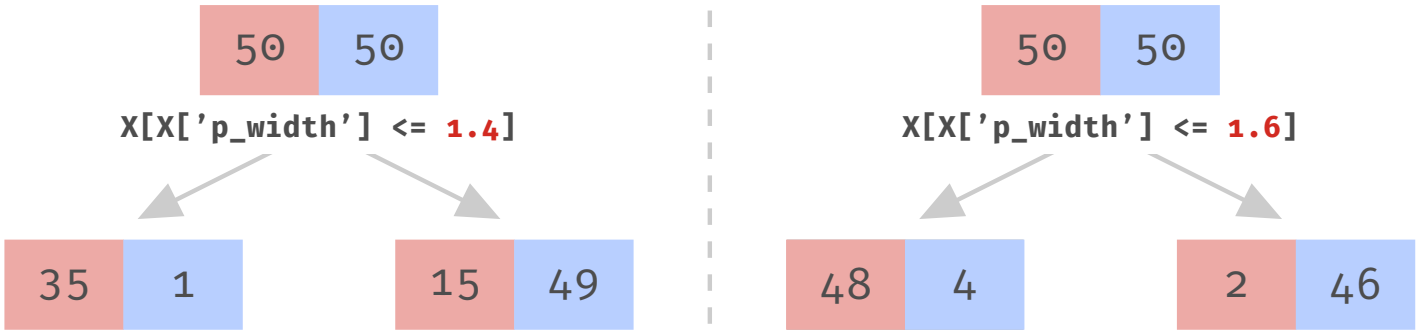
- Wir probieren alle Attribute und deren Werte aus!

Frage: Wie finden wir das beste Attribut a und Wert t ?

- Wir probieren alle Attribute und deren Werte aus!

Beispiel: Iris Daten mit Klassen *virginica* und *versicolor*

- Datensatz hat Klassenverhältnis `50:50`
- Aufteilung nach Attribute `p_width` mit `t=1.4` und `t=1.6`



Aber welche Aufteilung ist besser?

Gini Index definiert Maßzahl für Unreinheit

Sei $\mathbf{X}$ die Datenmenge mit den Klassen $C_1, \ldots, C_k$, dann

$$G(\mathbf{X}) = 1 - \sum_{i=1}^{k} [P(C_i)]^2 \quad \text{, mit } P(C_i) = \frac{\text{Anzahl } C_i \text{ in } \mathbf{X}}{|\mathbf{X}|}$$

Idee:

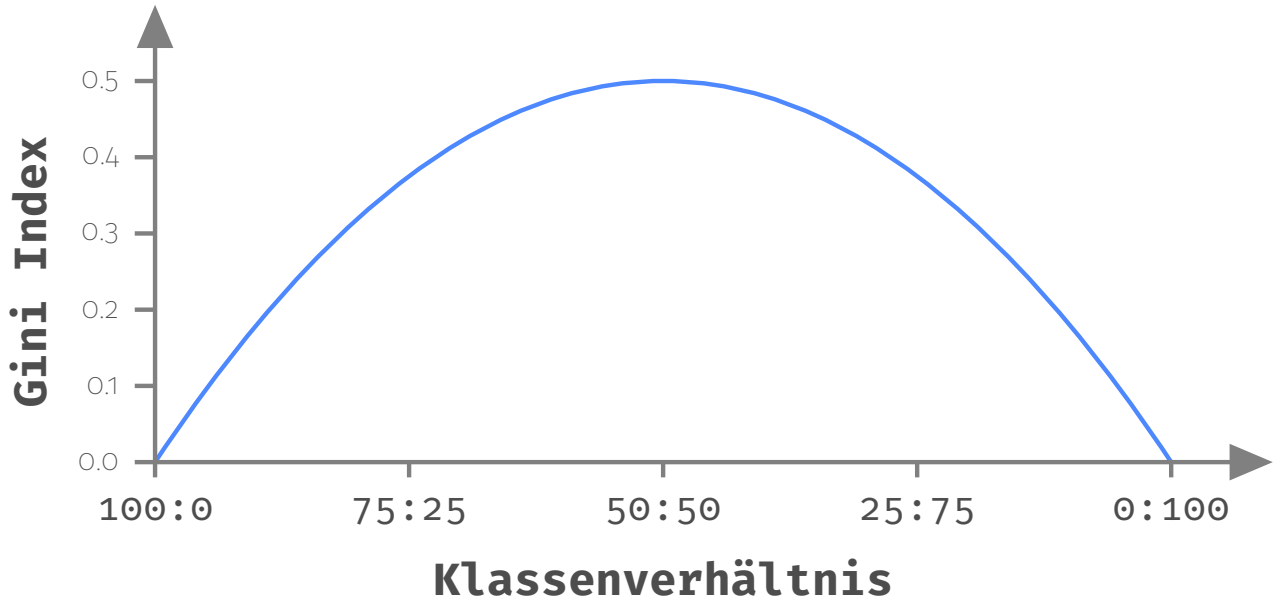
- Funktion, die bei Gleichverteilung (z.B. **50:50**) maximal ist und bei reinen Verhältnissen (z.B. 100:0) minimal.

Gini Index für Verhältnis von zwei Klassen in Python

Die folgende Funktion definiert den Gini Index für das Verhältnis von zwei Klassen in Python:

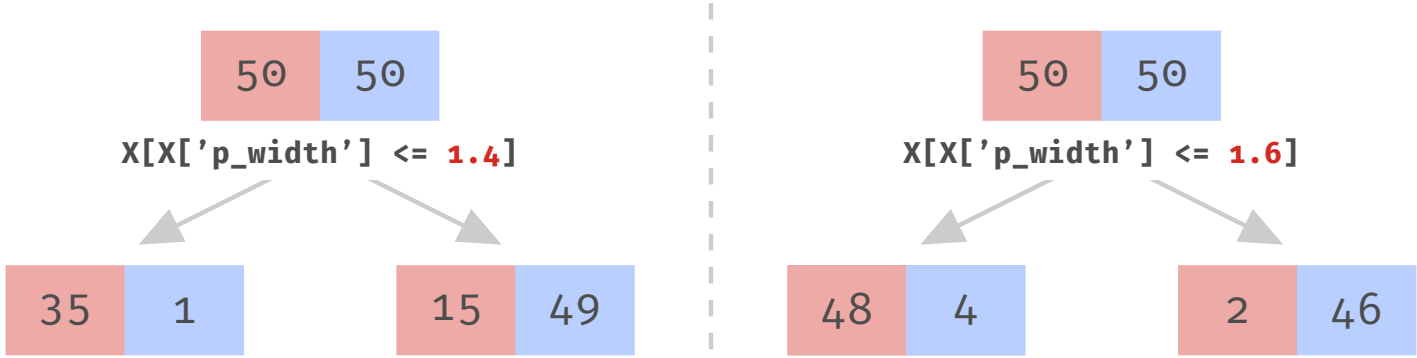
```
def gini(c1,c2):
  """ Berechnet den Gini Index fuer
    das Klassenverhaeltnis c1:c2"""
  total = c1 + c2
  sum = (c1/total)**2 + (c2/total)**2
  return 1 - sum

# Beispiel:
halb_halb = gini(50,50)  # ergibt 0.5
```

Verlauf von Gini Index für unterschiedliche Klassenverhältnisse

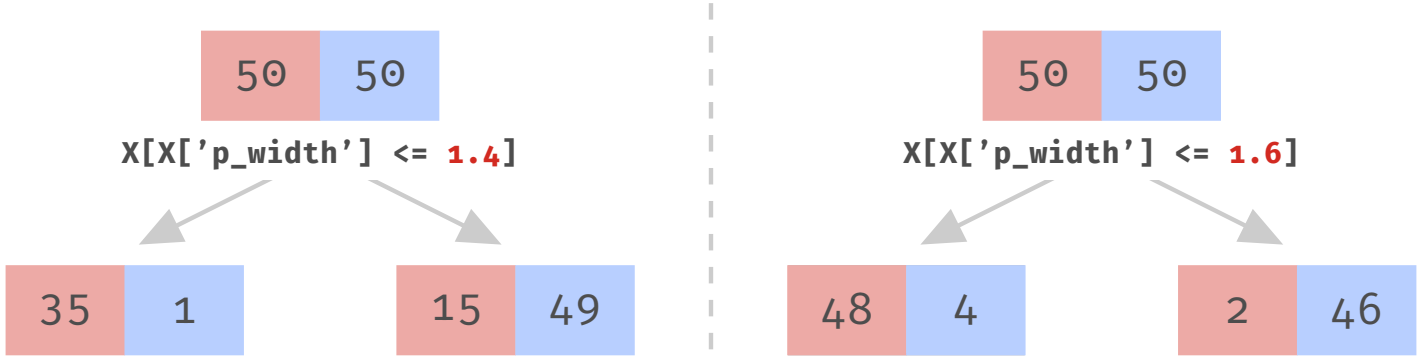


Die Grafik zeigt den Verlauf des Gini Index (blau) für unterschiedliche Klassenverhältnisse. Eine reinere Aufteilung führt zu einem kleineren Gini Index Wert.
Das Maximum hat der Gini Index beim Verhältnis `50:50`.

Beispiel: Gini Index für unsere Split-Auswahl

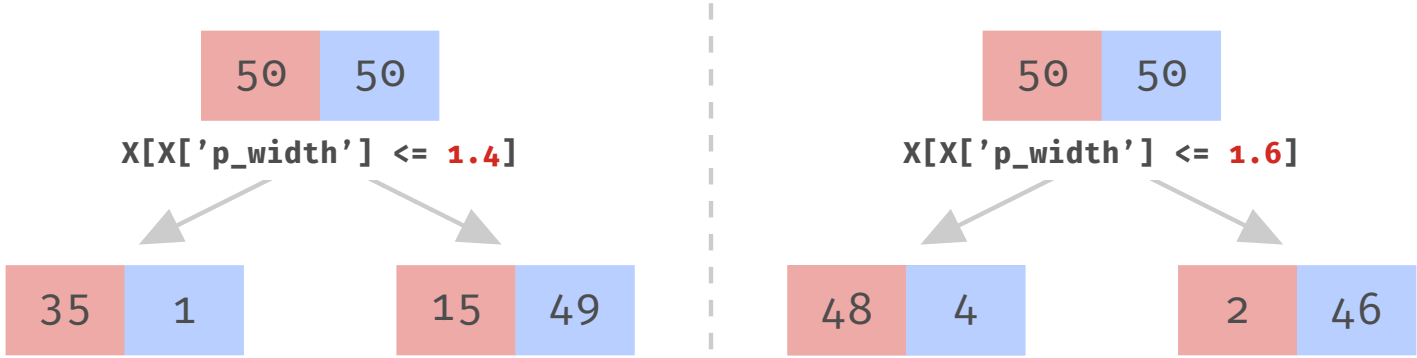


Beispiel: Gini Index für unsere Split-Auswahl



$$\text{gini}(35,1) = 1 - \left[\left(\frac{35}{36}\right)^2 + \left(\frac{1}{36}\right)^2\right] = \mathbf{0.054}$$

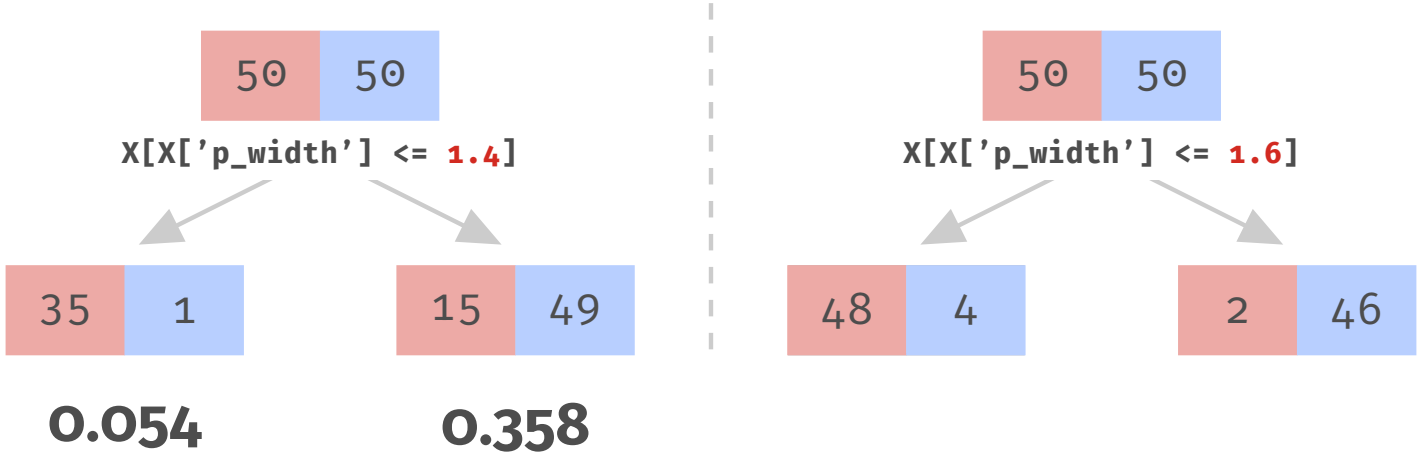
Beispiel: Gini Index für unsere Split-Auswahl



0.054

$$\text{gini}(35,1) = 1 - \left[\left(\frac{35}{36}\right)^2 + \left(\frac{1}{36}\right)^2\right] = \mathbf{0.054}$$

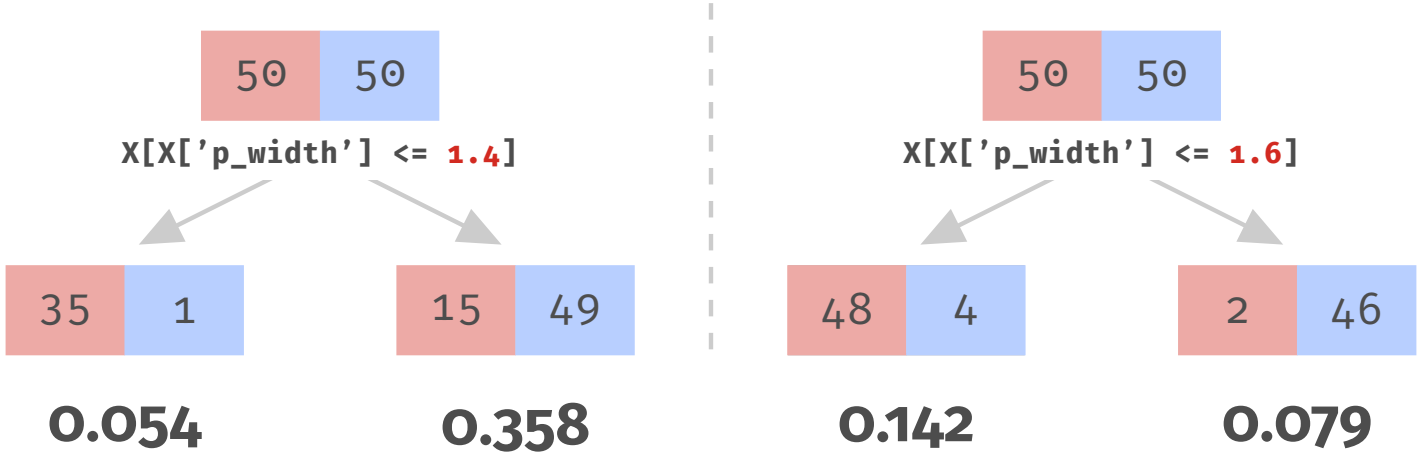
Beispiel: Gini Index für unsere Split-Auswahl



$$\text{gini}(35,1) = 1 - \left[\left(\frac{35}{36}\right)^2 + \left(\frac{1}{36}\right)^2\right] = \mathbf{0.054}$$

$$\text{gini}(15,49) = 1 - \left[\left(\frac{15}{64}\right)^2 + \left(\frac{49}{64}\right)^2\right] = \mathbf{0.358}$$

Beispiel: Gini Index für unsere Split-Auswahl



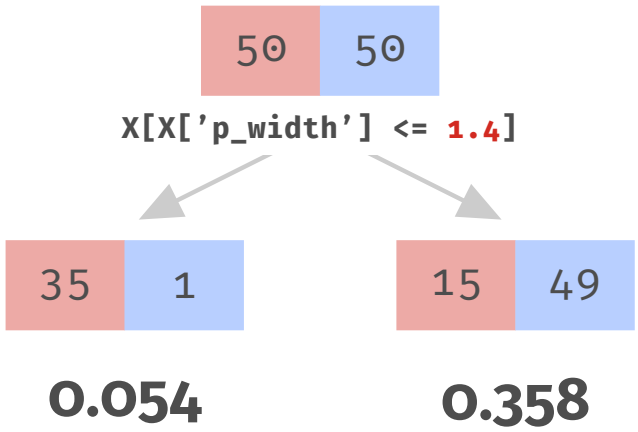
$$\text{gini}(35,1) = 1 - \left[\left(\frac{35}{36}\right)^2 + \left(\frac{1}{36}\right)^2\right] = \mathbf{0.054}$$

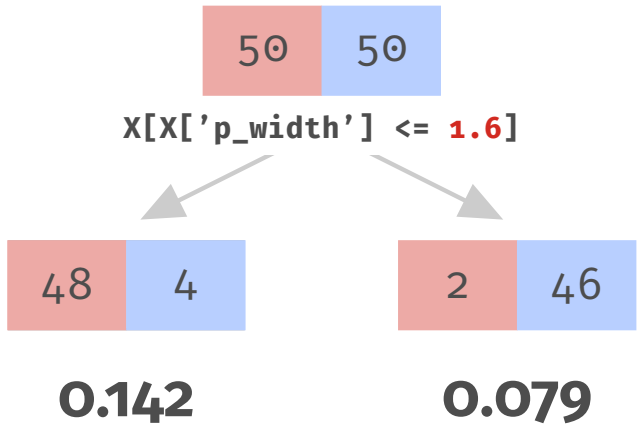
$$\text{gini}(15,49) = 1 - \left[\left(\frac{15}{64}\right)^2 + \left(\frac{49}{64}\right)^2\right] = \mathbf{0.358}$$

$$\text{gini}(48,4) = \mathbf{0.142}$$

$$\text{gini}(2,46) = \mathbf{0.079}$$

Beispiel: Gini Index für unsere Split-Auswahl





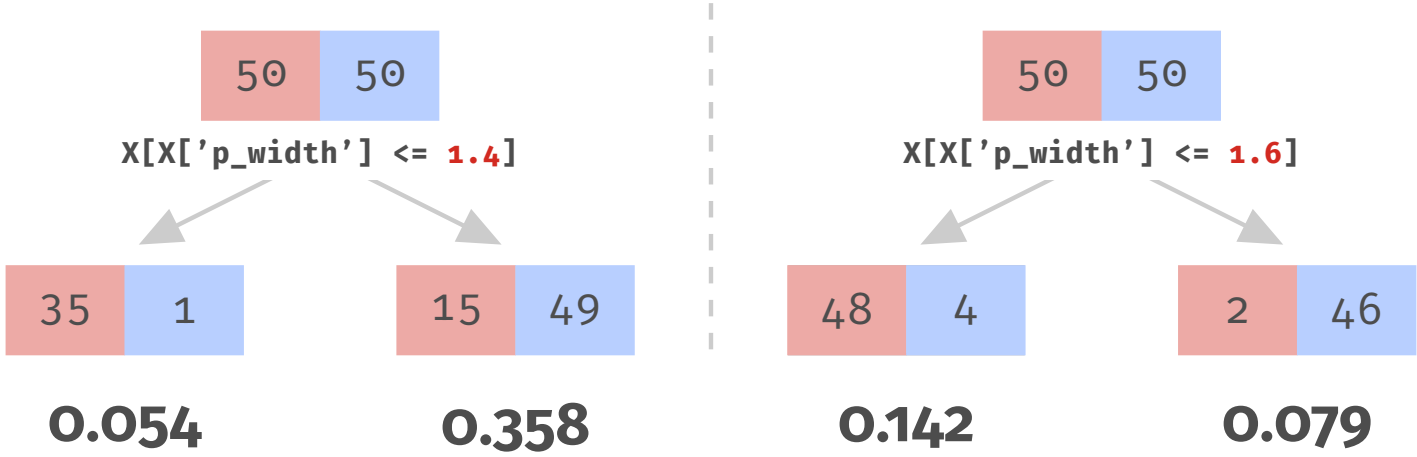
$$\frac{36}{100} \cdot 0.054 + \frac{64}{100} \cdot 0.358$$

$$= \mathbf{0.248}$$

$$\frac{52}{100} \cdot 0.142 + \frac{48}{100} \cdot 0.079$$

$$= \mathbf{0.112}$$

Beispiel: Gini Index für unsere Split-Auswahl



$$\frac{36}{100} \cdot 0.054 + \frac{64}{100} \cdot 0.358 = \mathbf{0.248}$$

$$\frac{52}{100} \cdot 0.142 + \frac{48}{100} \cdot 0.079 = \mathbf{0.112}$$

Aufteilung mit `petal_width <= 1.6` liefert besseres Ergebnis als mit 1.4

Beispiel

- Betrachten wir weiter das Attribute `petal_width`
- Der DataFrame hat in der Spalte `petal_width` 16 verschiedene Werte

```
values = list(set(iris["petal_length"].values))
values.sort()

print(values)
```

Gini Index: Beispiel

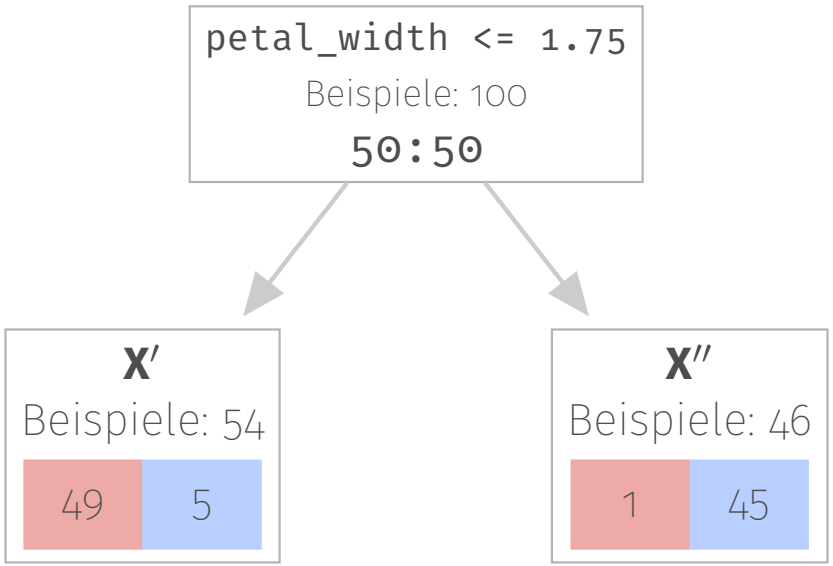
	Bedingung: $\leq t$			Bedingung: $> t$			$\sum$
	#Versi	#Virgi	gini	#Versi	#Virgi	gini	
petal_width <= 1.0	7	0	0.000	43	50	0.497	0.462
petal_width <= 1.1	10	0	0.000	40	50	0.494	0.444
petal_width <= 1.2	15	0	0.000	35	50	0.484	0.412
petal_width <= 1.3	28	0	0.000	22	50	0.424	0.306
petal_width <= 1.4	35	1	0.054	15	49	0.359	0.249
petal_width <= 1.5	45	3	0.117	5	47	0.174	0.147
petal_width <= 1.6	48	4	0.142	2	46	0.080	0.112
petal_width <= 1.7	49	5	0.168	1	45	0.043	0.110
petal_width <= 1.8	50	16	0.367	0	34	0.000	0.242
petal_width <= 1.9	50	21	0.417	0	29	0.000	0.296
petal_width <= 2.0	50	27	0.455	0	23	0.000	0.351
petal_width <= 2.1	50	33	0.479	0	17	0.000	0.398
petal_width <= 2.2	50	36	0.487	0	14	0.000	0.419
petal_width <= 2.3	50	44	0.498	0	6	0.000	0.468
petal_width <= 2.4	50	47	0.500	0	3	0.000	0.485
petal_width <= 2.5	50	50	0.500	0	0	0.000	0.500

Gini Index: Beispiel

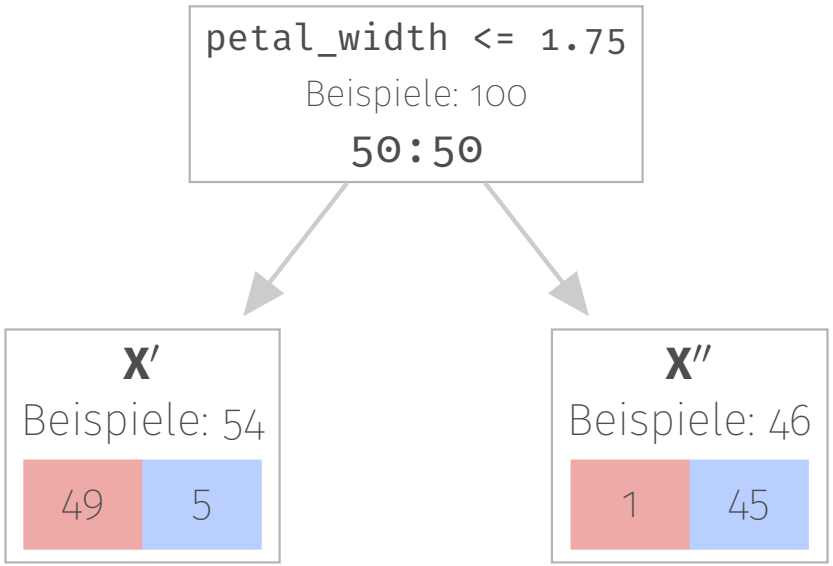
	Bedingung: $\leq t$			Bedingung: $> t$			$\sum$
	#Versi	#Virgi	gini	#Versi	#Virgi	gini	
petal_width <= 1.0	7	0	0.000	43	50	0.497	0.462
petal_width <= 1.1	10	0	0.000	40	50	0.494	0.444
petal_width <= 1.2	15	0	0.000	35	50	0.484	0.412
petal_width <= 1.3	28	0	0.000	22	50	0.424	0.306
petal_width <= 1.4	35	1	0.054	15	49	0.359	0.249
petal_width <= 1.5	45	3	0.117	5	47	0.174	0.147
petal_width <= 1.6	48	4	0.142	2	46	0.080	0.112
petal_width <= 1.7	49	5	0.168	1	45	0.043	0.110
petal_width <= 1.8	50	16	0.267	0	34	0.000	0.242
petal_width <= 1.9	50	23	0.377	0	27	0.000	0.296
petal_width <= 2.0	50	27	0.455	0	23	0.000	0.351
petal_width <= 2.1	50	33	0.479	0	17	0.000	0.398
petal_width <= 2.2	50	36	0.487	0	14	0.000	0.419
petal_width <= 2.3	50	44	0.498	0	6	0.000	0.468
petal_width <= 2.4	50	47	0.500	0	3	0.000	0.485
petal_width <= 2.5	50	50	0.500	0	0	0.000	0.500

Minimaler Wert für gewichteten Gini Index

Wir wählen `petal_width <= 1.75` als ersten Split-Punkt



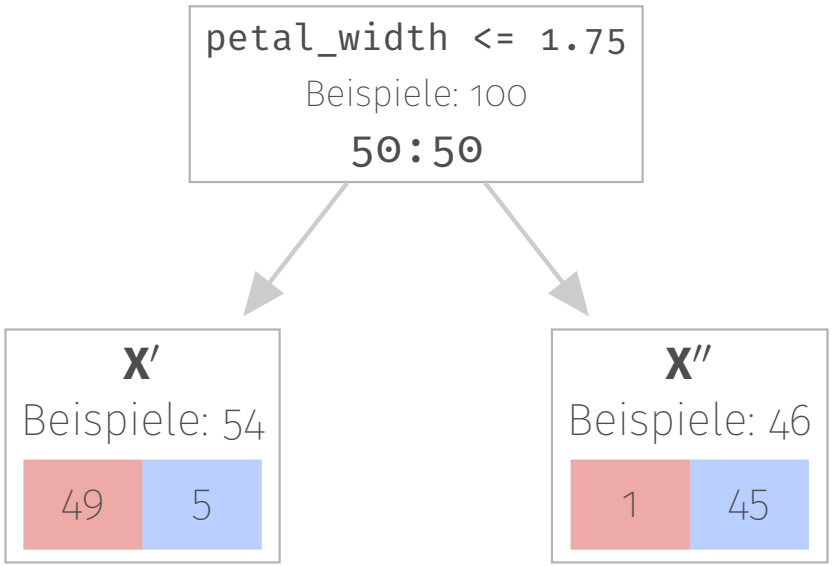
Wir wählen `petal_width <= 1.75` als ersten Split-Punkt



Damit haben wir die Daten **X** in **X′** und **X″** aufgeteilt.

Der Trainingsfehler hat sich auf 6% verbessert.

Wir wählen `petal_width <= 1.75` als ersten Split-Punkt



Damit haben wir die Daten **X** in **X′** und **X″** aufgeteilt.
Der Trainingsfehler hat sich auf 6% verbessert.

Mit **X′** (linke Seite) und **X″** (rechte Seite)
machen wir weiter und teilen weiter auf...

Gini Index wird häufig verwendet (standard bei vielen Tools)

Das zweite wichtige Kriterium ist die Entropie.

Die zeige ich nur auf den nächsten beiden Folien, damit Sie die Begriffe zumindest schonmal gehört haben.

Die Idee/das Konzept dahinter ist das Gleiche.

Weiteres Split-Kriterium: Entropie

Entropie misst den mittleren Informationsgehalt (in *bit*):

$$\text{Ent}(\mathbf{X}) = \sum_{i=1}^{k} -P(C_i) \log_2 P(C_i)$$

In Kombination mit dem Information Gain (kurz: *Info Gain*)

$$\text{InfoGain}(\mathbf{X}, a) = \text{Ent}(\mathbf{X}) - \sum_{v \in Values(a)} \frac{|\mathbf{X}_{\leq v}|}{|\mathbf{X}|} \text{Ent}(\mathbf{X}_{\leq v})$$

Weiteres Split-Kriterium: Entropie

Entropie misst den mittleren Informationsgehalt (in *bit*):

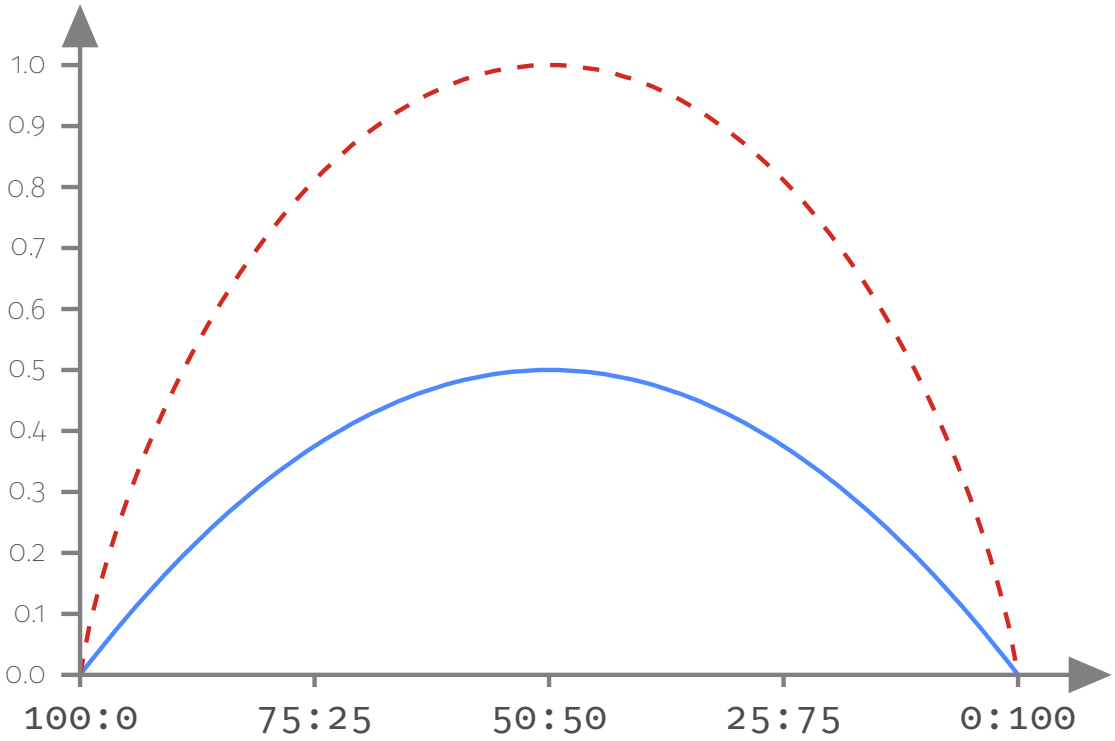
$$\text{Ent}(\mathbf{X}) = \sum_{i=1}^{k} -P(C_i) \log_2 P(C_i)$$

In Kombination mit dem Information Gain (kurz: *Info Gain*)

$$\text{InfoGain}(\mathbf{X}, a) = \text{Ent}(\mathbf{X}) - \sum_{v \in Values(a)} \frac{|\mathbf{X}_{\leq v}|}{|\mathbf{X}|} \text{Ent}(\mathbf{X}_{\leq v})$$

Info Gain beschreibt den Informationsgewinn, den wir erhalten, wenn wir das Attribut a mit Wert v zum Teilen nutzen.

Verlauf von Gini und Entropie



Die Grafik zeigt den Verlauf des Gini Index (blau) und der Entropy (rot) für verschiedene Klassenverhältnisse. Je *reiner* die Klassenaufteilung, desto kleiner sind die Werte für Gini, bzw. die Entropie.

Entscheidungsbäume

SciKit-Learn DecisionTreeClassifier

Das `scikit-learn` Modul enthält...

- viele verschiedene Lernverfahren (u.a. Entscheidungsbäume)
- Funktionen für train/test Splitting, Kreuzvalidierung,...
- arbeitet auch mit Pandas zusammen!

scikit-learn ist in mehrere Pakete geglieder, z.B.

```
sklearn.clusters
sklearn.datasets
sklearn.linear_model
sklearn.model_selection
sklearn.tree
sklearn.svm
...
```

Das `scikit-learn` Modul enthält...

- viele verschiedene Lernverfahren (u.a. Entscheidungsbäume)
- Funktionen für train/test Splitting, Kreuzvalidierung,...
- arbeitet auch mit Pandas zusammen!

scikit-learn ist in mehrere Pakete geglieder, z.B.

```
sklearn.clusters
sklearn.datasets
sklearn.linear_model
sklearn.model_selection
sklearn.tree  <-- Paket für Entscheidungsbäume
sklearn.svm
...
```

Grundlegende Funktionen von Modellen

Die Klasse `Zufall` der letzten Vorlesung folgte genau den Konventionen für Modelle Python, die auch SciKit-Learn verfolgt:

- `m = Model()` - neues Modells anlegen (`init(..)`)
- `m.fit(X, y)` - zum Lernen eines Modells
- `m.predict(X)` - Model auf Daten anwenden

Wichtiger Hinweis zu SciKit-Learn

- Zur Effizienz basiert scikit-learn auf den Arrays des **numpy** Moduls
- Pandas DataFrame, Series nutzen ebenfalls **numpy** Arrays
- An einigen Stellen müssen wir konvertieren

Entscheidungsbaum mit `sklearn` erstellen

```
from sklearn.tree import DecisionTreeClassifier

# Daten X und y vorbereiten
X = iris[['sepal_width','petal_width',...]]
y = iris['species']

# neues Model 'DecisionTreeClassifer' erstellen:
m = DecisionTreeClassifier()

m.fit(X, y)              # trainieren des Modells
y_hat = m.predict(X)     # Anwenden auf Trainingsdaten
```

◀ Probieren Sie es im Notebook aus!

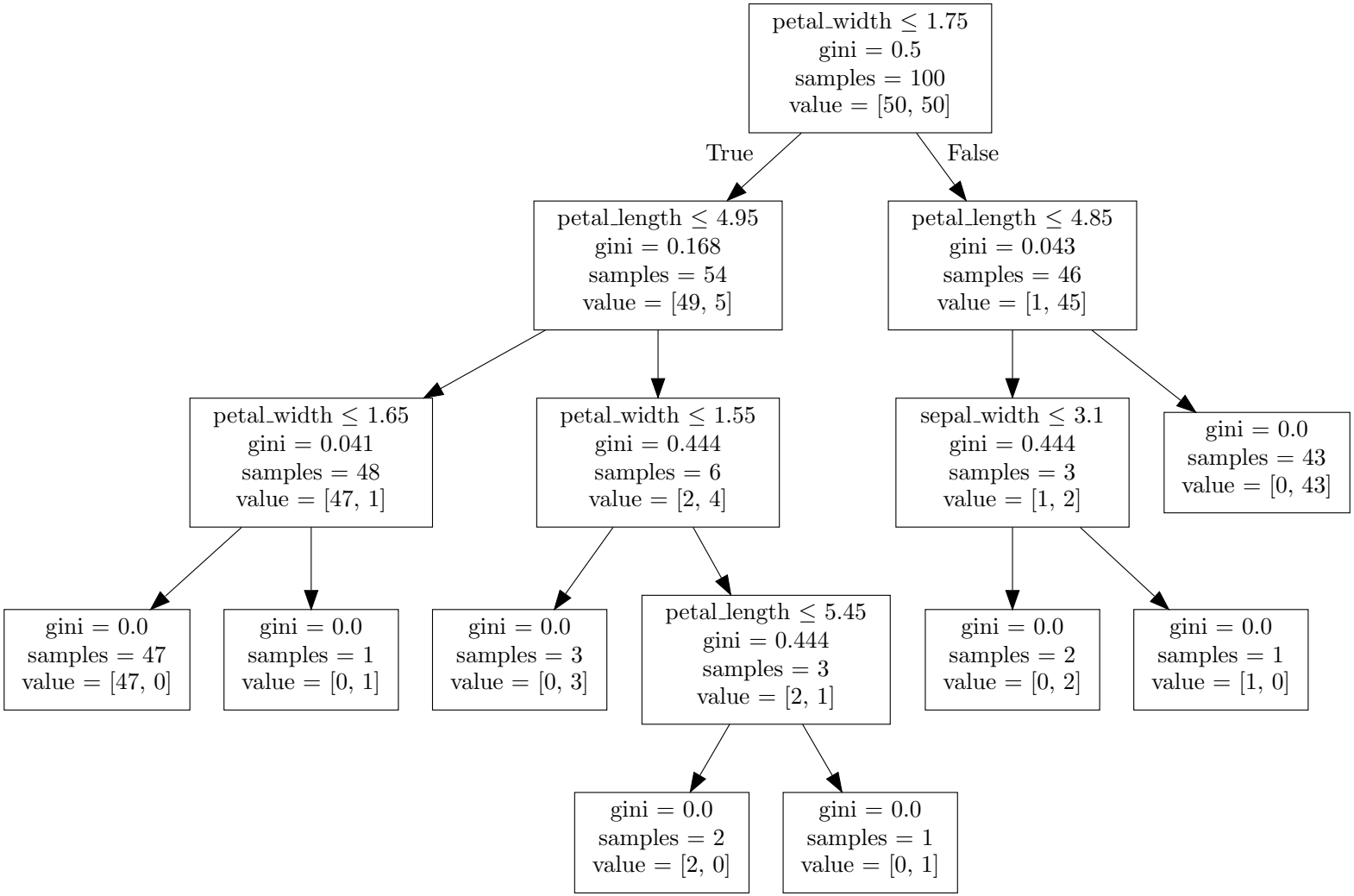
Notebook: `Vorlesung/V5-First-DecisionTree`

Baum anzeigen

Mit der Funktion plot_tree läßt sich der Baum anzeigen

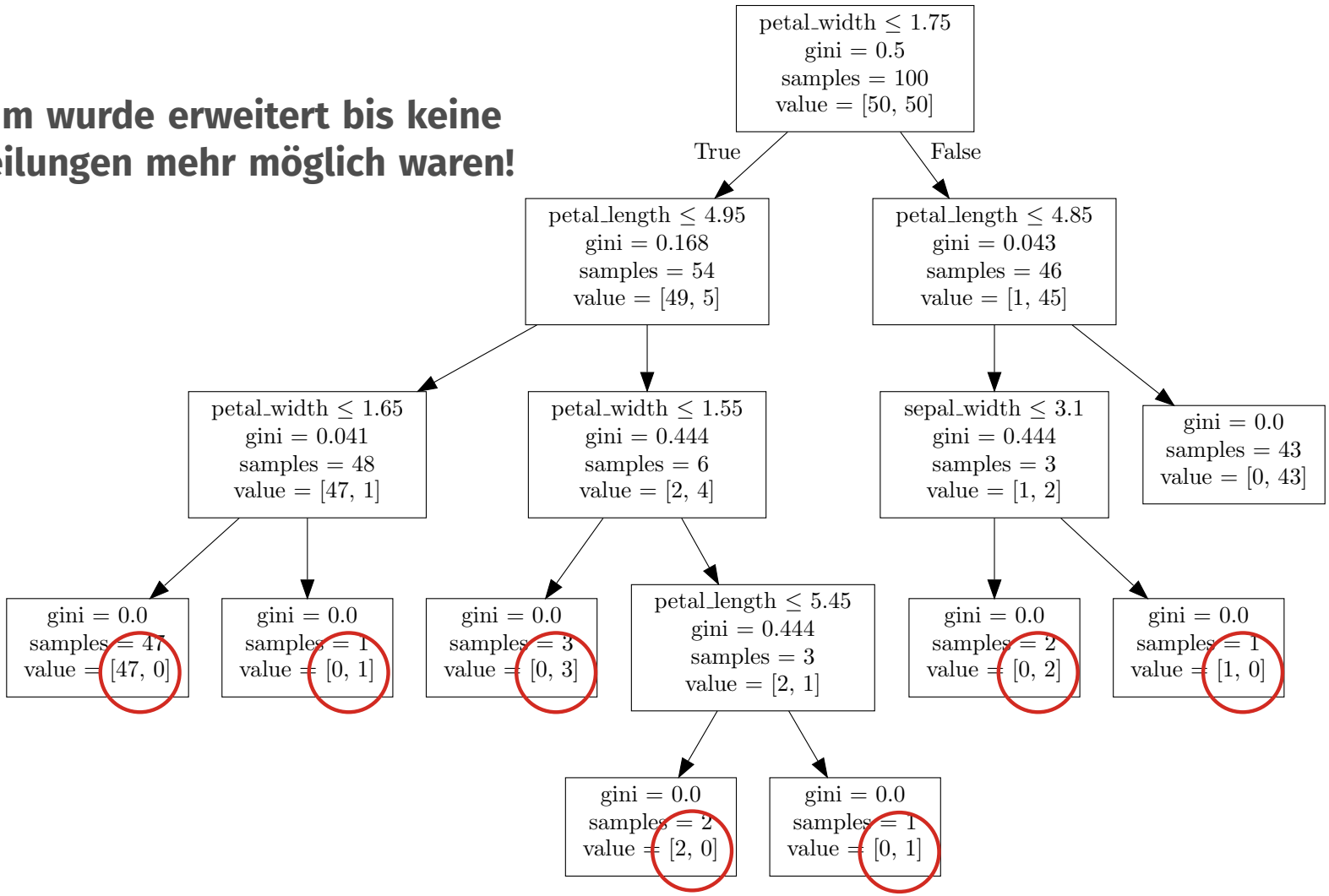
```
from sklearn.tree import plot_tree

# Erzeugen, trainieren...
m = DecisionTreeClassifier()
m.fit(..)

# Den Baum im Notebook anzeigen
plot_tree(m)
```

Graphische Darstellung des Baumes



Graphische Darstellung des Baumes

Baum wurde erweitert bis keine Aufteilungen mehr möglich waren!



Wie sieht der Trainingsfehler unseres Baumes aus?

```
# DataFrame X, Series y von zuvor
m.fit(X, y)

# Achtung!  y_hat ist ein numpy-array!!
y_hat = m.predict(X)
type(y_hat)                 # => numpy.ndarray
```

Wie sieht der Trainingsfehler unseres Baumes aus?

```
# DataFrame X, Series y von zuvor
m.fit(X, y)

# Achtung!  y_hat ist ein numpy-array!!
y_hat = m.predict(X)
type(y_hat)                 # => numpy.ndarray
```

Wir wollen ja die Anzahl der Fehler berechnen:

$$AbsoluteError(\mathbf{X}, f) = \sum_{(x,y)\in \mathbf{X}\times\mathbf{y}} err(y, f(x))$$

Vorhersage von sklearn-Modellen sind numpy-Arrays

```
y_hat = m.predict(X)
print(y_hat)
```

ergibt:

```
array(['versicolor', 'versicolor', ...
       ... , 'virginica', 'virginica'])
```

Wie vergleichen wir das Array in y_hat mit der Series y?

- Unterschiedliche Typen - Series.values ist Sequenz der Werte!
- Erinnern Sie sich an `zip`?

Wie vergleichen wir das Array in `y_hat` mit der Series `y`?

Wir können mit `zip` über zwei Sequenzen gleichzeitig laufen:

```
errors = 0

for (true_y,predicted_y) in zip(y.values,y_hat):
  if predicted_y == true_y:
    print( "Vorhersage richtig!" )
  else:
    print( "Vorhersage falsch!" )
    errors = errors + 1

absError = errors            # absoluter Fehler
relError = errors / len(y)   # relativer Fehler
```

◀ Probieren Sie es im Notebook aus!

Berechnung des absoluten Trainingsfehlers

```python
# 0-1 Loss Function
def err_01(y,y_hat):
  if y == y_hat:
    return 0
  else:
    return 1

# Absoluter Fehler mit Fehlerfunktion  loss
def absoluteError(ys, ys_hat, loss=err_01):
  sum = 0.0
  for (y,y_hat) in zip(ys, ys_hat):
    sum = sum + loss(y,y_hat)

  return sum
```

Beispiel für Python Funktion für den absoluten Fehler.

Zeit für eine Pause + Kaffee/Tee!?

- Das sind natürlich wieder viele Grundlagen
- Der nun folgende, letzte Teil der Folien geht um die Auswertung mit Python/Jupyter und SciKit-Learn

Modell-Evaluation

Frage: Wie bewerten wir das gelernte Modell f ?

Wir kennen bereits:

- den durchschnittlichen Fehler auf Trainingsdaten
- die Idee, separate Test-Daten zu nutzen

ABER: Wie aussagekräftig ist der durchschnittliche Fehler?

- Bisher: Gleichverteilung der Klassen
- Was passiert bei ungleich verteilten Klassen?
- Welche Klasse wird schlechter vorhergesagt?

Vorhersage nach Mehrheitsklasse

Modell: Mehrheit

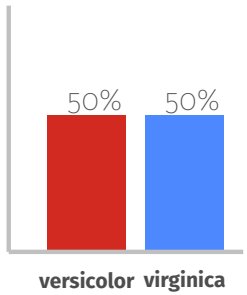
Das Modell Mehrheit sagt immer die gleiche Klasse vorher - und zwar die, die im Trainingsdatensatz häufiger auftrat.

Vorhersage nach Mehrheitsklasse

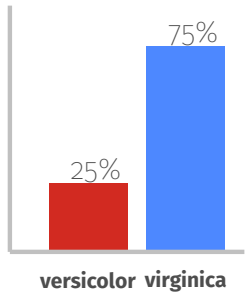
Modell: Mehrheit

Das Modell Mehrheit sagt immer die gleiche Klasse vorher - und zwar die, die im Trainingsdatensatz häufiger auftrat.

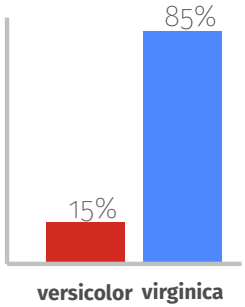
Überlegen Sie sich, welchen Fehler das Modell Mehrheit auf den folgenden Datensätzen erzeugt:



Modell Zufall
⌀ Fehler: **50%**



Modell Mehrheit
⌀ Fehler: **____ %**



Modell Mehrheit
⌀ Fehler: **____ %**

Durchschnittlicher relativer Fehler reicht nicht!

Wir benötigen genauere Kennzahlen für die Modellbewertung:

- Vorhersage-Güte für jede Klasse
- *Gewichteter* Durchschnittsfehler
- Fehlermodell für mehr als zwei Klassen!?

Durchschnittlicher relativer Fehler reicht nicht!

Wir benötigen genauere Kennzahlen für die Modellbewertung:

- Vorhersage-Güte für jede Klasse
- *Gewichteter* Durchschnittsfehler
- Fehlermodell für mehr als zwei Klassen!?

Bisherige Fehler könnten wir pro Klasse bestimmen:

- absoluter Fehler für jede Klasse
- relativer Fehler für jede Klasse

Durchschnittlicher relativer Fehler reicht nicht!

Wir benötigen genauere Kennzahlen für die Modellbewertung:

- Vorhersage-Güte für jede Klasse
- *Gewichteter* Durchschnittsfehler
- Fehlermodell für mehr als zwei Klassen!?

Bisherige Fehler könnten wir pro Klasse bestimmen:

- absoluter Fehler für jede Klasse
- relativer Fehler für jede Klasse

Die Fehler pro Klasse werden in der Confusion Matrix zusammengefasst!

Die *Confusion Matrix* (Konfusionsmatrix)

		Vorhersage ŷ	
“Wahrheit” y	**Klasse Pos**	**Klasse Neg**	
Klasse Pos	True Pos (TP)	False Neg (FN)	TP / (TP + FN)
Klasse Neg	False Pos (FP)	True Neg (TN)	TN / (FN + TN)
	TP / (TP + FP)	TN / (TN + FP)	

Die *Confusion Matrix* (Konfusionsmatrix)

	Vorhersage ŷ: Klasse Pos	Vorhersage ŷ: Klasse Neg	
"Wahrheit" y: Klasse Pos	True Pos (TP)	False Neg (FN)	TP / (TP + FN)
Klasse Neg	False Pos (FP)	True Neg (TN)	TN / (FN + TN)
	TP / (TP + FP)	TN / (TN + FP)	

TP = Anzahl der richtigen Vorhersagen für Klasse Pos

Die *Confusion Matrix* (Konfusionsmatrix)

		Vorhersage ŷ	
	“Wahrheit” y	**Klasse Pos**	**Klasse Neg**
	Klasse Pos	True Pos (TP)	False Neg (FN)
	Klasse Neg	False Pos (FP)	True Neg (TN)

TP / (TP + FN)	
TN / (FN + TN)	

TP / (TP + FP)	TN / (TN + FP)

TP = Anzahl der richtigen Vorhersagen für Klasse Pos
FP = Anzahl der falschen Vorhergesagten für Klasse Pos

Die *Confusion Matrix* (Konfusionsmatrix)

		Vorhersage ŷ	
“Wahrheit” y	**Klasse Pos**	**Klasse Neg**	
Klasse Pos	True Pos (TP)	False Neg (FN)	TP / (TP + FN)
Klasse Neg	False Pos (FP)	True Neg (TN)	TN / (FN + TN)
	TP / (TP + FP)	TN / (TN + FP)	

TP = Anzahl der richtigen Vorhersagen für Klasse Pos
FP = Anzahl der falschen Vorhergesagten für Klasse Pos
FN = Anzahl der falschen Vorhergesagten für Klasse Neg

Die *Confusion Matrix* (Konfusionsmatrix)

		Vorhersage ŷ	
“Wahrheit” y		**Klasse Pos**	**Klasse Neg**
	Klasse Pos	True Pos (TP)	False Neg (FN)
	Klasse Neg	False Pos (FP)	True Neg (TN)

TP / (TP + FN)	
TN / (FN + TN)	
TP / (TP + FP)	TN / (TN + FP)

TP = Anzahl der richtigen Vorhersagen für Klasse Pos
FP = Anzahl der falschen Vorhergesagten für Klasse Pos
FN = Anzahl der falschen Vorhergesagten für Klasse Neg
TN = Anzahl der richtig Vorhergesagten für Klasse Neg

Die *Confusion Matrix* (Konfusionsmatrix)

	Vorhersage ŷ: Klasse Pos	Vorhersage ŷ: Klasse Neg	
"Wahrheit" y: Klasse Pos	True Pos (TP)	False Neg (FN)	TP / (TP + FN)
"Wahrheit" y: Klasse Neg	False Pos (FP)	True Neg (TN)	TN / (FN + TN)
	TP / (TP + FP)	TN / (TN + FP)	

Precision

Wie oft lag das Modell bei Klasse Pos richtig?

Die *Confusion Matrix* (Konfusionsmatrix)

		Vorhersage ŷ		Recall
	“Wahrheit” y	**Klasse Pos**	**Klasse Neg**	Wie viele von Klasse Pos hat des Modell gefunden?
	Klasse Pos	True Pos (TP)	False Neg (FN)	TP / (TP + FN)
	Klasse Neg	False Pos (FP)	True Neg (TN)	TN / (FN + TN)
		TP / (TP + FP)	TN / (TN + FP)	

Kennzahlen für die Evaluation von Klassifikatoren

Aus der Confusion Matrix lassen sich weitere Kennzahlen ableiten:

- accuracy
- precision
- recall
- prevalence
- false pos rate
- f-measure

Definition: accuracy

Die accuracy ist definiert als

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Hinweis:

$$accuracy = 1 - relativeError$$

Definition: precision

Die Kennzahl precision ist definiert als

$$precision = \frac{TP}{TP + FP}$$

Definition: recall

Die Kennzahl recall definiert, wie viele Elemente einer Klasse C tatsächlich vom Model als solche entdeckt wurden:

$$recall = \frac{TP}{TP + FN}$$

Confusion Matrix für mehr als zwei Klassen

Wahrheit **y** \ Vorhersage **ŷ**	setosa	versicolor	virginica	
setosa	23	0	0	100% 0%
versicolor	0	15	1	94% 6%
virginica	0	3	18	86% 14%
	100%	83%	95%	
	0%	17%	5%	

Confusion Matrix für mehr als zwei Klassen

Vorhersage $\hat{y}$

Wahrheit y	setosa	versicolor	virginica		
setosa	23	0	0	100%	0%
versicolor	0	15	1	94%	6%
virginica	0	3	18	86%	14%
	100%	83%	95%		
	0%	17%	5%		

TP Werte finden sich jeweils auf der Hauptdiagonalen

Confusion Matrix für mehr als zwei Klassen

Wahrheit **y** \ Vorhersage **ŷ**	setosa	versicolor	virginica	
setosa	23	0	0	100% 0%
versicolor	0	15	1	94% 6%
virginica	0	3	18	86% 14%
	100%	83%	95%	
	0%	17%	5%	

FN für eine Klasse ergeben sich als Summe der Zeile ohne den TP Wert

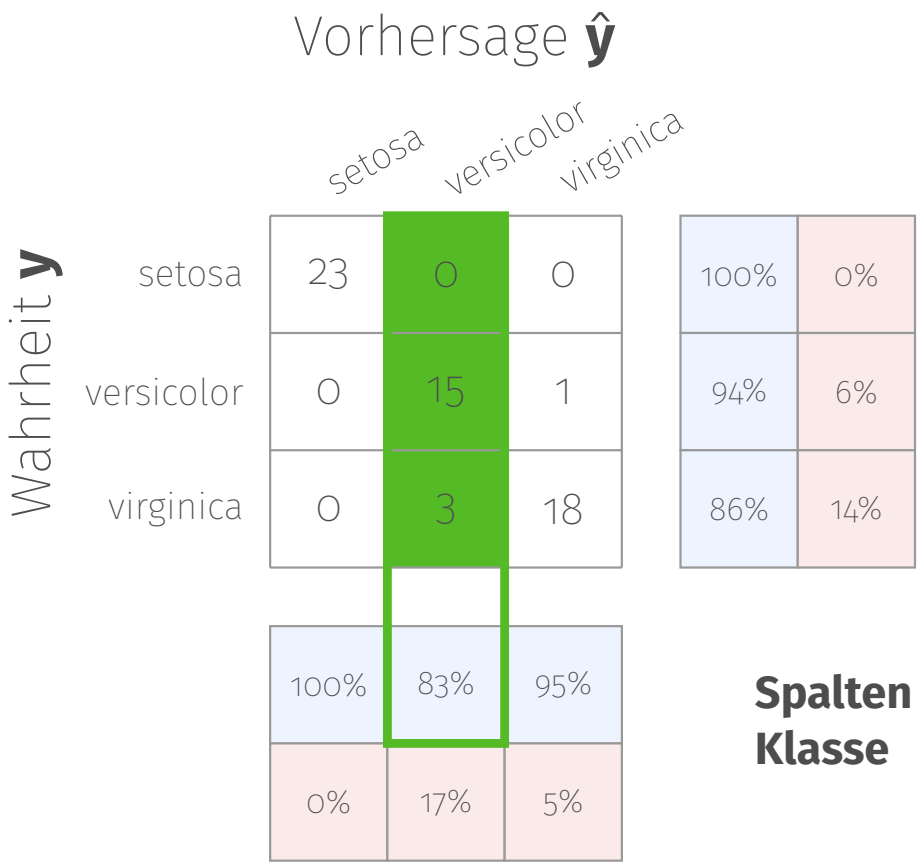
Confusion Matrix für mehr als zwei Klassen

Vorhersage $\hat{y}$

Wahrheit y	setosa	versicolor	virginica		
setosa	23	0	0	100%	0%
versicolor	0	15	1	94%	6%
virginica	0	3	18	86%	14%
	100%	83%	95%		
	0%	17%	5%		

Zeilen liefern *Recall* für eine Klasse

Confusion Matrix für mehr als zwei Klassen



Spalten liefern *Precision* für eine Klasse

Auch SciKit-Learn unterstützt die Modell-Evaluation

- Das Paket `sklearn.metrics` enthält u.a. Funktionen `precision_score`, `accuracy_score`,...
- Und unterstützt die Erstellung der *confusion matrix*:

```
from sklearn.metrics import confusion_matrix

y = ['setosa', 'setosa', 'virginica',...]
y_hat = ['setosa', 'versicolor', 'virginica',....]

matrix = confusion_matrix(y, y_hat)
```

Beispiel mit SciKit-Learn

Zum Abschluss: Beispiel mit SciKit-Learn

Wir betrachten im Folgenden diese Schritte:

1. Daten lesen
2. Trainings- und Test-Daten aufteilen
3. Modell trainieren
4. Mit Test-Daten das Modell evaluieren

Zum Abschluss: Beispiel mit SciKit-Learn

Wir betrachten im Folgenden diese Schritte:

1. Daten lesen
2. Trainings- und Test-Daten aufteilen
3. Modell trainieren
4. Mit Test-Daten das Modell evaluieren

Die confusion matrix haben wir auch für mehr als zwei Klassen kennengelernt.

Beispiel auf Iris Daten mit allen drei Klassen!

Schritt 1: Daten lesen

Das haben wir ja bereits häufiger gemacht (Pandas):

```
import pandas as pd

df = pd.read_csv('data/iris.csv')

# Daraus nehmen wir die Spalten mit Features
# und separat (fuer y) die Spalte mit den Labeln

features = [c for c in df.columns if c != 'species']

X = iris[features]
y = iris['species']
```

Schritt 2: Trainings- und Test-Daten aufteilen

Eigene Split-Methode war ja Bestandteil von Übungsblatt 4.

Wir schauen uns das jetzt mal mit SciKit-Learn an:

```
from sklearn.model_selection import train_test_split

# Wir teilen den Datensatz in 80% Trainingsdaten
# und 20% Testdaten auf:
X_tr, X_tst, y_tr, y_tst = train_test_split(X, y,
                                            test_size=0.2)
```

Hinweis: Ich habe hier lediglich aus Platzgründen auf der Folie `X_train` als `X_tr` und `X_test` als `X_tst` abgekürzt.

Schritt 3: Modell trainieren

Wir erzeugen einen leeren Entscheidungsbaum und trainineren ihn mit den Trainingsdaten:

```
from sklearn.tree import DecisionTreeClassifier

# 'gini' ist der Default Fall als Split-Kriterium,
                                wir
# koennten auch 'entropy' nehmen:
model = DecisionTreeClassifier(criterion="gini")

# Den Baum trainieren:
model.fit(X_tr, y_tr)
```

Schritt 4: Modell auf Test-Daten ausprobieren

Wir wenden das gelernte Modell mit `predict` auf die Test-Daten an und schauen uns die confusion matrix an:

```
from sklearn.metrics import confusion_matrix

# y_hat enthaelt dann die vorhergesagten Werte:
y_predicted = model.predict(X_tst)

# Wir berechnen die Confusion Matrix
# aus (wahrheit,vorhersage):
matrix = confusion_matrix(y_tst, y_predicted)
print(matrix)
```

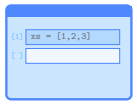
Auch interessant - der classification report

SciKit-Learn bietet noch einen kleinen Bericht zusätzlich zur confusion matrix, nämlich den **classification report**:

```
from sklearn.metrics import classification_report

# Aus den Test y und den vorhergesagten y_predicted
# kann man den classification report erstellen:
#
report = classification_report(y_tst, y_predicted,
                               digits=3)
print(report)
```

Beispiel: Zusammenfassung

- Die 4 Schritte zeigen Modell Training + Evaluation mit SciKit-Learn
- Schritt 1 sollte Ihnen aus den vorherigen Folien bekannt sein
- Schritte 2-4 benutzen sklearn Bausteine aus dieser Vorlesung



◀ Probieren Sie es im Notebook aus!

Notebook:
Kurse/DataScience1/V5-Entscheidungsbaum-sklearn