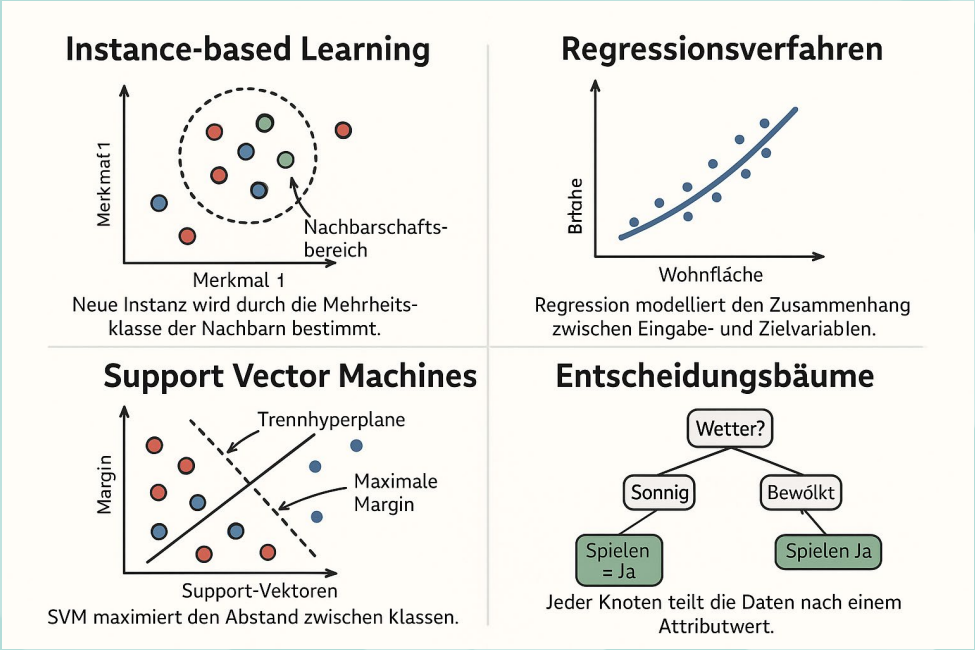


Big Data & Data Science

Machine Learning – Klassische Verfahren

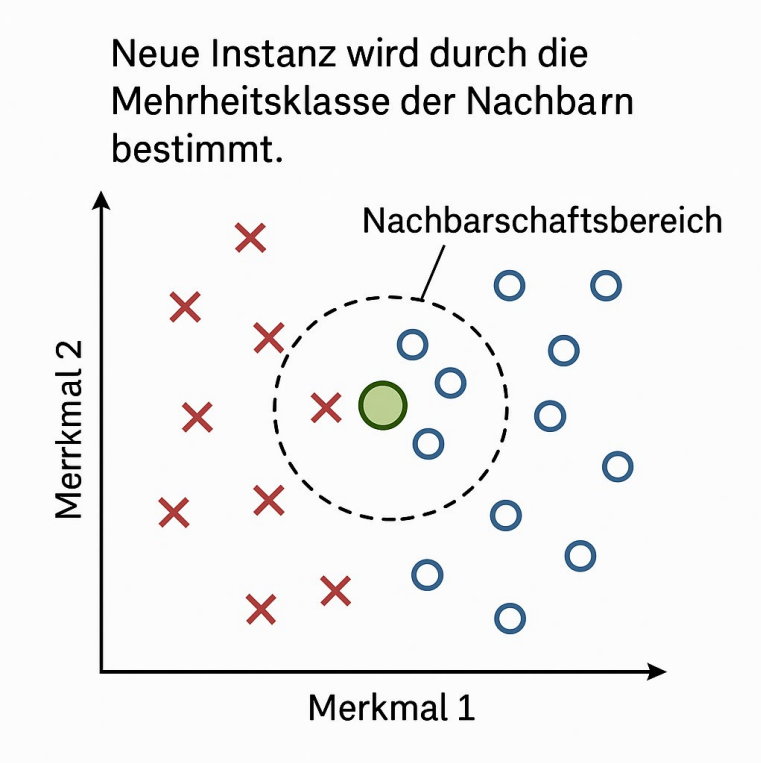
WS 2025/26

Prof. Dr. Klemens Waldhör

Copyright

Inhalt

- Instance-based Learning
- Regressionsverfahren
- Support Vector Machines (SVM)
- Entscheidungsbäume



Instance-based Learning

Instance-based Learning

- Ein instanzbasiertes Lernverfahren trifft Entscheidungen indem es neue Beispiele mit gespeicherten Trainingsinstanzen vergleicht deren Klassen (Labels) bekannt sind und als Grundlage für die Zuordnung dienen.
- Es lernt aus beschrifteten Trainingsdaten (Eingaben mit bekannten Zielwerten oder Klassen Labels) ⇨ Supervised learning-Verfahren
- Es baut in der Regel auf einer Datenbank mit Beispieldaten auf und vergleicht neue Daten mit der Datenbank anhand eines Ähnlichkeitsmaßes um die beste Übereinstimmung zu finden und eine Vorhersage zu treffen.
- Aus diesem Grund werden instanzbasierte Methoden auch als winner-take-all-Methoden und speicherbasiertes Lernen bezeichnet (memory based learning lazy learning)
- Der Schwerpunkt liegt dabei auf der Darstellung der gespeicherten Instanzen und den zwischen den Instanzen verwendeten Ähnlichkeitsmaßen
- Die beliebtesten instanzbasierten Algorithmen sind:
  - k-Nearest Neighbor (kNN)
  - Learning Vector Quantization (LVQ)
  - Self-Organizing Map (SOM)
  - Locally Weighted Learning (LWL)

Vorteile

- Leicht anpassbar an neue Daten
- Sehr einfaches Verfahren
- Je mehr Beispiele desto besser (?)

Nachteile

- Klassifikationsaufwand hoch da die Abstände für alle Beispiele bestimmt werden müssen (bestimmt die Komplexität)
- Große Speicherplatzbedarf
- Voraussetzung Existenz eines Abstandsmaß

Abstandsmaße

- Euklidischer Abstand $d(xx') = \sum \sqrt{(\mathrm{x_i} - x_i')^2}$
- Manhattan Distanz $d(xx') = \sum |x_i - xi'|$
- Maximum-Distanz $d(xx') = max_i(|x_i - xi'|)$
  - größter Abstand in einer der i Dimensionen
- Hammingdistanz $d(xx') = counti(x_i \neq x_i')$
  - An wieviel Positionen unterscheidet sich der Datensatz?
  - Abstandmaß für nominale *x*

- Beliebtestes Verfahren für eine Beispielmenge x und einen Nachbarn x' ist der Euklidische Abstand

https://importq.files.wordpress.com/2017/11/knn_neigh.gif?w=656

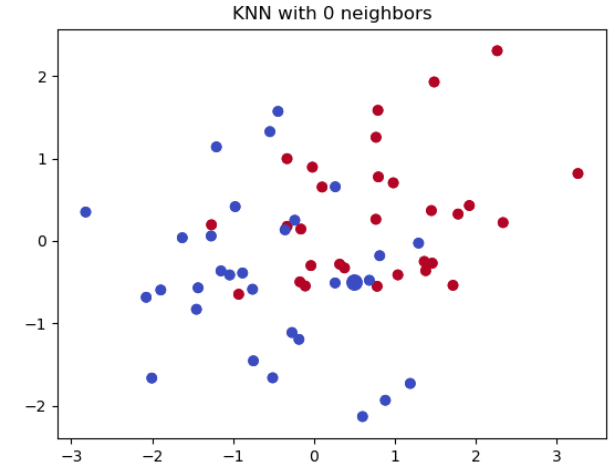
Instance-based Learning - K-Nearest-Neighbour

Algorithmus

(1) Für alle x_i aus den Trainingsbeispie
Ähnlichkeit zu y

(2) Wähle daraus diejenigen k Beispiele
ähnlichsten sind

(3) Die Klasse K(y) ist die Klasse die i
Trainingsbeispielen am häufigsten vo



https://importq.files.wordpress.com/2017/11/knn_neigh.gif?w=656

- Brute Force Algorithmus
- Modifikation
  - Es werden in (3) alle k Datenpunkte gleichwertig behandelt – ohne Berücksichtigung des Abstandes
  - Shepards Methode
    Gewichtung der Abstände der 1..k Datenpunkte i mit dem $1/d_i^2$
  - Wenn $d_i = 0$ dann i korrekte Klasse

Instance-based Learning / k-Nearest Neighbor (kNN)

Verfahren

- Messung der Distanz (z.B. euklidische Distanz) zwischen dem neuen zu klassifizierenden Datensatz und allen anderen Datensätzen.

$$d(xx') = \sqrt{(x_1 - x'_1)^2 + \cdots + (x_n - x'_n)^2}$$

- Wählen Sie die k kleinsten Distanzen (z.B. k=3 für 3 nächstgelegene Nachbarn).
- Wahl der Klasse die am häufigsten unter den **k** nächsten Nachbarn vorkommt. Diese wird als Vorhersage für den neuen Datensatz gewählt.

https://importq.files.wordpress.com/2017/11/knn_neigh.gif?w=656

Punkt	**x**	**y**	**Klasse**
P1	1	1	A
P2	2	2	A
P3	4	4	B
P4	6	5	B

Neuer Punkt mit den Koordinaten (33)	3	3	**Zu A oder B?**
Punkt			**k = 3**
P1	*2,82842712*		***A***
P2	*1,41421356*		***A***
P3	*1,41421356*		*B*
P4	3,60555128		B
	Wahl A da 2:1		

Instance-based Learning - K-Nearest-Neighbour

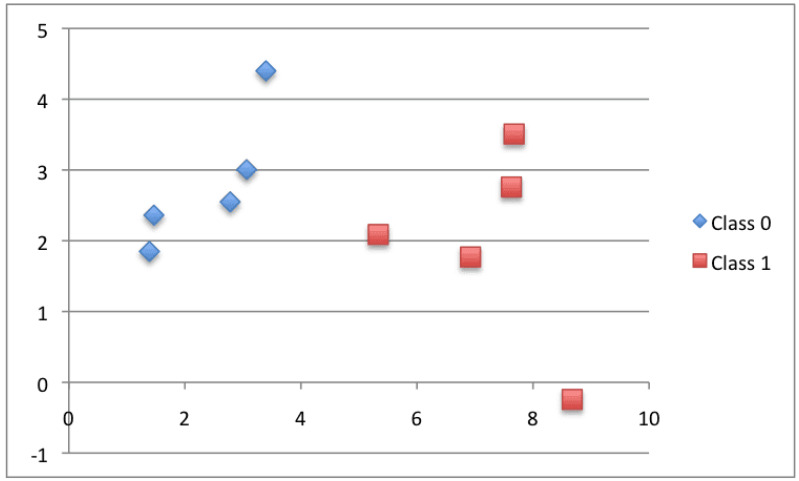
```
[1]: # Example of getting neighbors for an instance
from math import sqrt

# calculate the Euclidean distance between two vectors
def euclidean_distance(row1, row2):
    distance = 0.0
    for i in range(len(row1)-1):
        distance += (row1[i] - row2[i])**2
    return sqrt(distance)

# Locate the most similar neighbors
def get_neighbors(train, test_row, num_neighbors):
    distances = list()
    for train_row in train:
        dist = euclidean_distance(test_row, train_row)
        distances.append((train_row, dist))
    distances.sort(key=lambda tup: tup[1])
    neighbors = list()
    for i in range(num_neighbors):
        neighbors.append(distances[i][0])
    return neighbors

# Test distance function
dataset = [[2.7810836,2.550537003,0],
    [1.465489372,2.362125076,0],
    [3.396561688,4.400293529,0],
    [1.38807019,1.850220317,0],
    [3.06407232,3.005305973,0],
    [7.627531214,2.759262235,1],
    [5.332441248,2.088626775,1],
    [6.922596716,1.77106367,1],
    [8.675418651,-0.242068655,1],
    [7.673756466,3.508563011,1]]
neighbors = get_neighbors(dataset, dataset[0], 3)
for neighbor in neighbors:
    print(neighbor)
```

```
[2.7810836, 2.550537003, 0]
[3.06407232, 3.005305973, 0]
[1.465489372, 2.362125076, 0]
```



https://machinelearningmastery.com/tutorial-to-implement-k-nearest-neighbors-in-python-from-scratch/

Instance Based Learning

Die Tabelle stellt die Klassenzuordnung zu zwei Produkttypen A und B basierend auf ihrer Qualität dar. Q1 und Q2 stellen Messungen zu Qualitätseigenschaften der Produkte P_{1-5} dar.

Produkt	Q1	Q2	Produkttyp
P1	2	3	A
P2	5	4	A
P3	3	6	B
P4	7	2	B
P5	4	5	A

Für ein neues Produkt P_{neu} mit den Messungen (4,4) soll der Produkttyp klassifiziert werden.

- Berechnen Sie sowohl die euklidische Distanz und Manhattan Distanz zwischen P_{neu} und jedem Produkt im Datensatz.
- Wie lautet die jeweilige Klassenzuordnung mit k=3?
- Wie ändert sich die Zuordnung mit k=1 und k=5?
- Was sind die Vor- und Nachteile des k-NN Algorithmus? Nennen Sie mindestens zwei Vor- und zwei Nachteile.
- Wie beeinflusst die Wahl von k die Klassifizierung im k-NN Algorithmus? Erklären Sie warum es wichtig ist einen geeigneten Wert für k zu wählen.

Golf Spielen ja oder nein?

Golfen ja / nein

- Beispieldaten

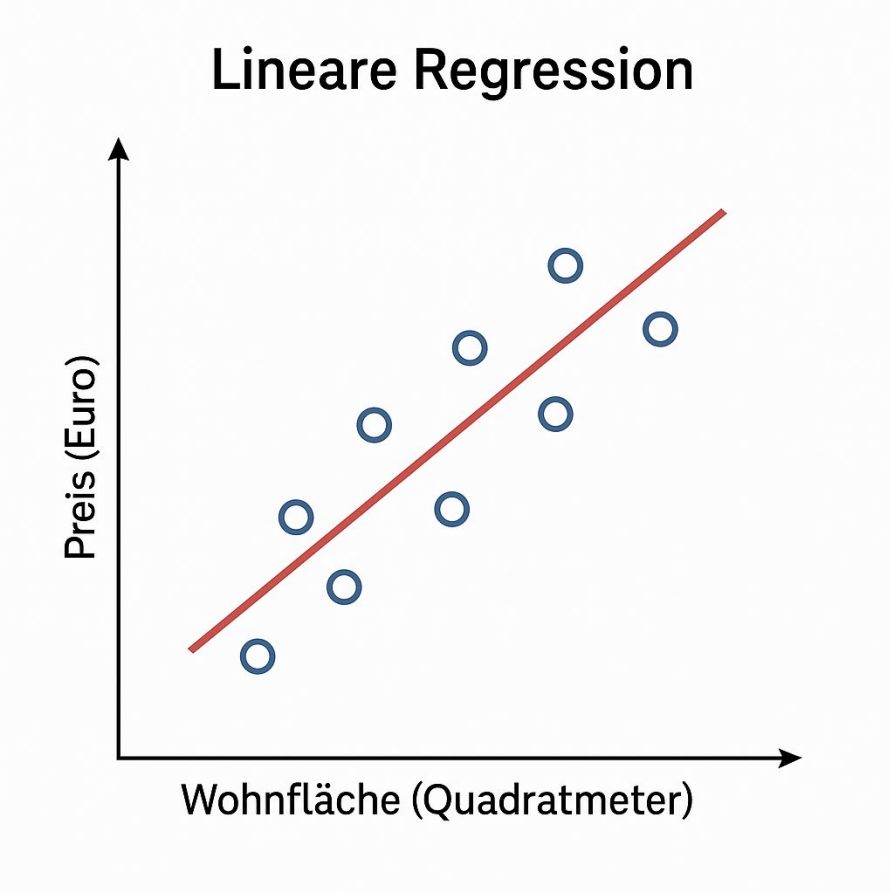
Datensatz	Outlook	Temperature	Humidity	Windy	Play
1	sunny	hot	high	FALSCH	no
2	sunny	hot	high	WAHR	no
3	overcast	hot	high	FALSCH	yes
4	rain	mild	high	FALSCH	yes
5	rain	cool	normal	FALSCH	yes
6	rain	cool	normal	WAHR	no
7	overcast	cool	normal	WAHR	yes
8	sunny	mild	high	FALSCH	no
9	sunny	cool	normal	FALSCH	yes
10	rain	mild	normal	FALSCH	yes
11	sunny	mild	normal	WAHR	yes
12	overcast	mild	high	WAHR	yes
13	overcast	hot	normal	FALSCH	yes
14	rain	mild	high	WAHR	no

- Welches Abstandsmaß wählen Sie?
- Verwenden Sie unterschiedliche ks für knn.

- Sollen Sie unter folgenden Bedingungen Golf spielen oder nicht?

```
outlook = sunny, temperature = mild, humidity = normal, windy = false
outlook = sunny, temperature = mild, humidity = normal, windy = true
outlook = rainy, temperature = hot, humidity = normal, windy = false
```

Quelle: Cleve J Lämmel U (2014) Data Mining. Oldenbourg Wissenschaftsverl. München. S. 92

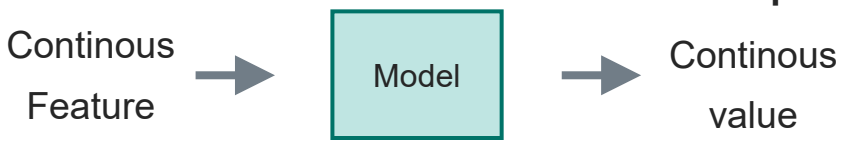


Machine Learning
Linear Regression

Linear Regression

- Beliebteste und am besten erforschte Methode zur Modellierung eines Datensatzes mit i=1 2...n numerischen Eingangswerten x_i zu numerischen Ausgangswerten y_i
- Abgeleitet aus dem Bereich der Statistik
- Annahme: Die Beziehung zwischen x_i und y_i ist linear
- Jeder Eingangswert x_i wird mit einem Gewicht m multipliziert und zu einer Bias b addiert
- Simple Linear Regression: Nur ein Eingabefeature x_i
- Multiple Linear Regression: Mehrer Inputfeatures $x_i^1 \dots x_i^n$

Simple Linear Regression Model



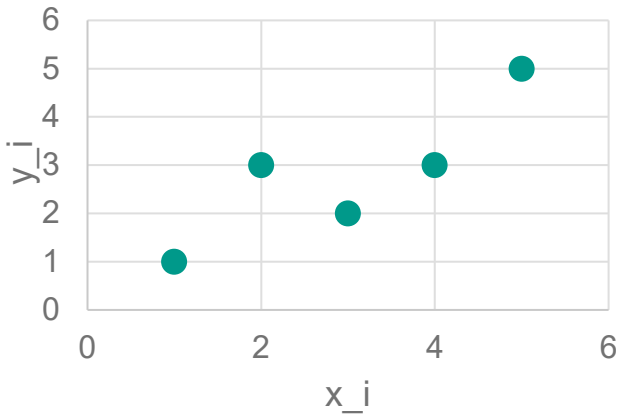
$$x_i \qquad \hat{y}_i = mx_i + b \qquad \hat{y}_i$$

(Geradengleichung)

Simple Linear Regression

Datenbasis

i	x_i	y_i
1	1	1
2	2	3
3	4	3
4	3	2
5	5	5



Ziel

- Vorhersage kontinuierlicher y_i-Werte auf der Grundlage von x_i-Werten unter Verwendung eines geeigneten Modells wobei y_i die vorhergesagten Werte des Modells sind

Lineares Regressionsmodell

Kontinuierliche Variable → Modell → Kontinuierliche Variable

x_i $\quad \hat{y}_i = mx_i + b$ $\quad \hat{y}_i$

(Geradengleichung)

- **Lösung:**
- Verwenden Sie ein lineares Regressionsmodell zur Annäherung an $\hat{y}_i$ an x_i

Parameters:

m - Steigung

b - Bias

Simple Linear Regression mit Gradientenabstieg

Das **Gradientenabstiegsverfahren (Gradient Descent)** ist ein Optimierungsverfahren das schrittweise die Parameter eines Modells anpasst um den Fehler (Kostenfunktion) zu minimieren. Die Parameter werden in Richtung des **negativen Gradienten** der Fehlerfunktion verändert sodass der Fehler am schnellsten abnimmt.

Modell initialisieren

Parameter zufällig

$$\hat{y}_i = mx_i + \mathrm{b}$$

$$m = 0.5$$

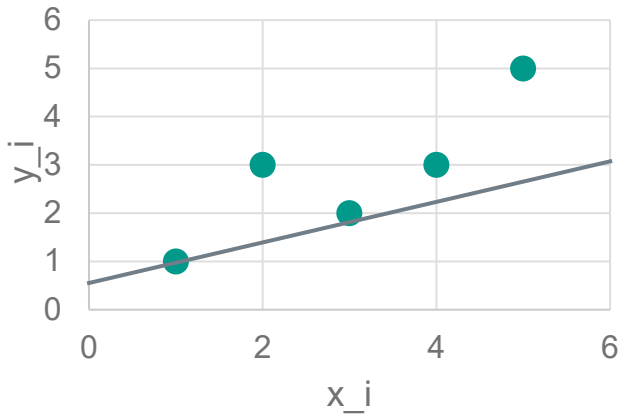
$$b = 0.5$$

Benütze Modell $\hat{y}_i = 0.5x_i + 0.5$

zur Vorhesage der $\hat{y}_i$ Werte

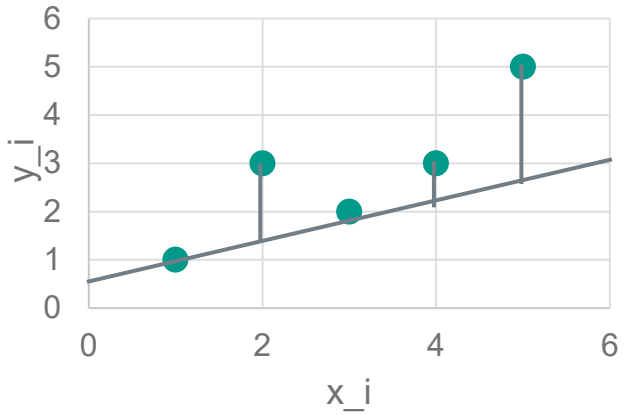
i	x_i	y_i	$\hat{y}_i$
1	1	1	**1**
2	2	3	**1.5**
3	4	3	**2.5**
4	3	2	**2**
5	5	3	**3**

$\hat{y}_i = 0.5x_i + 0.5$



Simple Linear Regression mit Gradientenabstieg

Berechnung der Fehler (Residuen) für einzelne Stichproben



i	x_i	y_i	$\hat{y}_i$	$\hat{y}_i - y_i$	$(\hat{y}_i - y_i)^2$
1	1	1	1	0	0
2	2	3	1.5	-1.5	2.25
3	4	3	2.5	-0.5	0.25
4	3	2	2	0	0
5	5	5	3	-2	4

Berechne Fehler über alle Datensätze

Mean Squared Error (MSE):

$$J = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$$

$$J = 13$$

- Die Fehler werden quadriert um nur positive Fehler zu erhalten.
- Regressionsgeraden mit einer guten Anpassung an die Daten haben einen geringen Fehler.
- **Ziel**: Suche nach der Regressionsgeraden mit dem geringsten Fehler (beste Anpassung an die Daten)

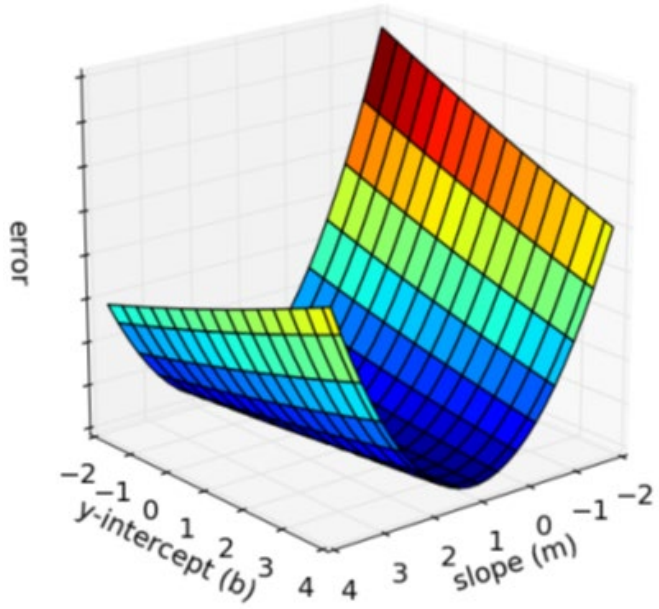
Simple Linear Regression mit Gradientenabstieg

- **Ziel:** Die Regressionsgerade finden die den geringsten Fehler aufweist (beste Anpassung an die Daten)
- Die Regressionsgerade wird durch m und b dargestellt.
- Wir wollen die Werte für m und b finden die den Fehler J minimieren (niedrigster Punkt in der Oberfläche)

$$J = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$$

$$= \frac{1}{n}\sum_{i=1}^{n}(mx_i + b - y_i)^2$$

Fehlerfunktion



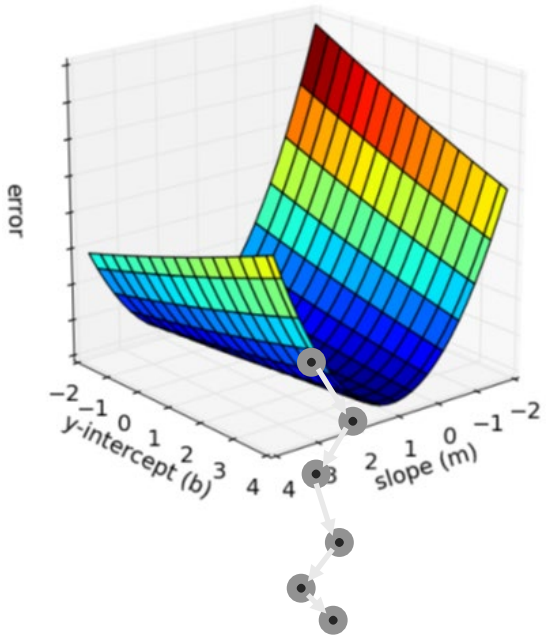
- Wie findet man die beste Lösung?
- Die meisten Methoden des maschinellen Lernens verwenden Gradientenabstieg um die besten Parameter zu finden

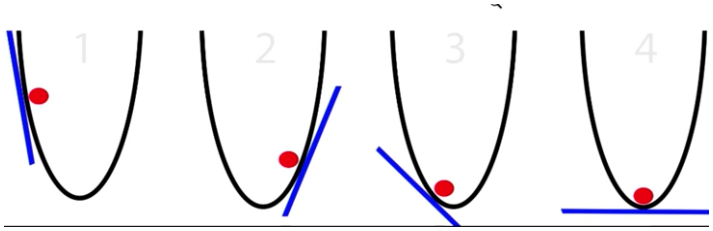
https://spin.atomicobject.com/wp-content/uploads/gradient_descent_error_surface.png

Simple Linear Regression mit Gradientenabstieg

Lösung der linearen Regression mit Gradientenabstieg

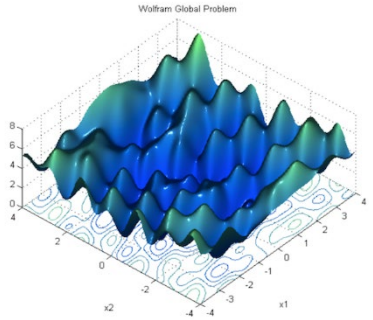
- **Ziel**: Finde die Werte für m und b die den Fehler J minimieren.
- **Idee**: Wir beginnen an einem zufälligen Punkt der Fehleroberfläche und gehen in kleinen Schritten bergab bis wir den niedrigsten Punkt erreichen.
- Jedes Mal wenn wir einen Schritt machen aktualisieren wir unsere Parameter m und b um die nächste Position auf der Oberfläche zu erreichen.
- **Frage**: Wie finden wir heraus in welche Richtung wir gehen müssen?
- **Lösung**: Berechne die Steigung (Anstieg) einer Tangente an die Fläche an unserem aktuellen Punkt und bewege dich in die entgegengesetzte Richtung





Tangente für die aktuelle Position auf der Fehlerfläche

https://ml4a.github.io/ml4a/how_neural_networks_are_trained/



https://spin.atomicobject.com/wp-content/uploads/gradient_descent_error_surface.png

https://cdn-images-1.medium.com/max/800/1*xihrqM-cOGFWILBHv_xOLA.png

Simple Linear Regression mit Gradientenabstieg

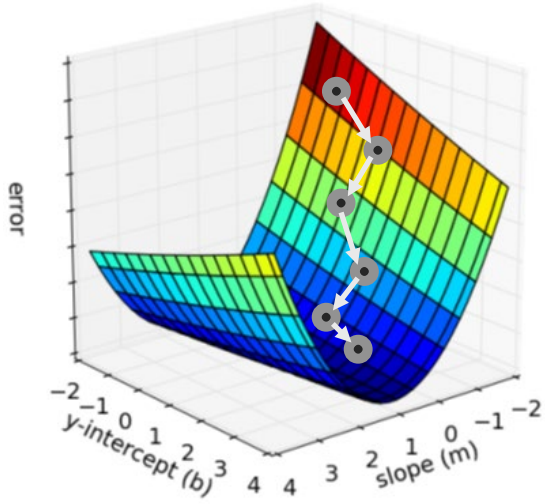
Lösung der linearen Regression mit Gradientenabstieg

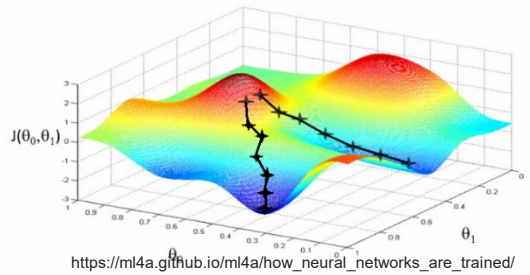
- Berechnung der Steigung für jede Parameterachse:
- Die Steigung kann durch Berechnung der partiellen Ableitungen von J $= \frac{1}{n}\sum_{i=1}^{n}(mx_i + b - y_i)^2$ für die beiden Parameter m und b ermittelt werden
  - **m-Achse:**

$$\delta_m = \frac{\partial J}{\partial m} = \frac{2}{n}\sum_{i=1}^{n} -x_i(y_i - (mx_i + b))$$

  - **b-Achse:**

$$\delta_b = \frac{\partial J}{\partial b} = \frac{2}{n}\sum_{i=1}^{n} -(y_i - (mx_i + b))$$





https://ml4a.github.io/ml4a/how_neural_networks_are_trained/

https://spin.atomicobject.com/wp-content/uploads/gradient_descent_error_surface.png

https://cdn-images-1.medium.com/max/800/1*xihrqM-cOGFWILBHv_xOLA.png

Übung: Ableitung der J Funktion

Zeigen Sie dass die dargestellten Ableitungen für korrekt sind!

$$J = \frac{1}{n}\sum_{i=1}^{n}(mx_i + b - y_i)^2$$

$$\delta_m = \frac{\partial J}{\partial m} = \frac{2}{n}\sum_{i=1}^{n} -x_i(y_i - (mx_i + b))$$

$$\delta_b = \frac{\partial J}{\partial b} = \frac{2}{n}\sum_{i=1}^{n} -(y_i - (mx_i + b))$$

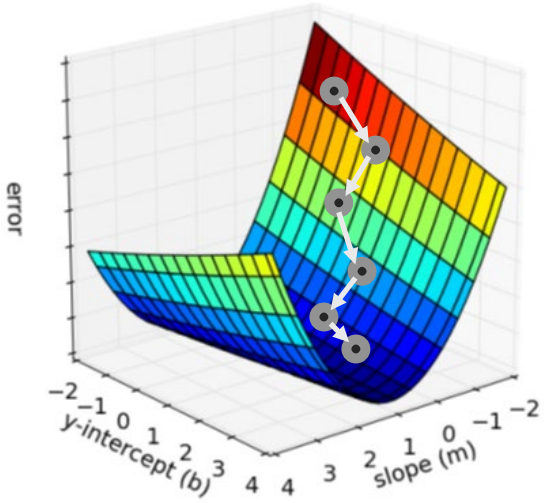
Simple Linear Regression mit Gradientenabstieg

Lösung der linearen Regression mit Gradientenabstieg

- **Berechnung der Steigung (partielle Ableitungen) für jeden Parameter**

$$\delta_m = \frac{\partial J}{\partial m} = \frac{2}{n}\sum_{i=1}^{n} -x_i(y_i - (mx_i + b)) \qquad \delta_b = \frac{\partial J}{\partial b} = \frac{2}{n}\sum_{i=1}^{n} -(y_i - (mx_i + b))$$

- **Aktualisierung der Parameter in Richtung der partiellen Ableitungen (Steigung)**



$$m = m - \alpha\, \delta_m$$

$$b = b - \alpha\, \delta_b$$

mit

$$\alpha = 0.01$$

Lernrate α

- Ist ein Hyperparameter für den Gradientenabstieg
- Legt fest wie groß die Schritte sind die wir in Richtung des Gradienten machen
- Schwierige Einstellung für viele Lernprobleme
- Wenn α zu niedrig ist wird das Training sehr langsam
- Wenn α zu groß ist könnte das Training das Minimum verfehlen (z.B. in diesem Szenario schlägt das Training fehl wenn wir α=0.1 setzen)

Linear Regression mit Gradientenabstieg

Simple Linear Regr. (eine unabhängie Variable)

- Datensatz mit einer unabhängigen Variable x und einer abhängigen Variable y
- Ziel: Vorhersage von y auf der Grundlage von x unter Verwendung eines Modells mit Gewichtung m und Verzerrung b

$$\hat{y}_i = mx_i + b$$

Gradientenabstieg mit einer unabhängigen Variable

```
Define error measure J
set hyperparameter α
initialize model parameters mb
repeat until convergence
    m = m − α ∂J/∂m
    b = b − α ∂J/∂b
```

Multiple Linear Regr. (n unabhängige Variablen)

- Datensatz mit n unabhängigen Variablen $x^1 \ldots x^n$ und einer abhängigen Variable y
- Ziel: Vorhersage von y auf der Grundlage von x unter Verwendung eines Modells mit Gewichten $\theta_1 \ldots \theta_n$ und Vorspannung b

$$\hat{y}_i = b + \theta_1 x_i^1 + \cdots + \theta_n x_i^n = \theta X^T + b$$

Gradientenabstieg mit n unabhängigen Variablen

```
Define error measure J
set hyperparameter α
initialize model parameters θ
repeat until convergence
        for (j=1;j<n;j++)
                θ_j = θ_j − α ∂J/∂θ_j

        b = b − α ∂J/∂b
```

Linear Regression mit Gradientenabstieg

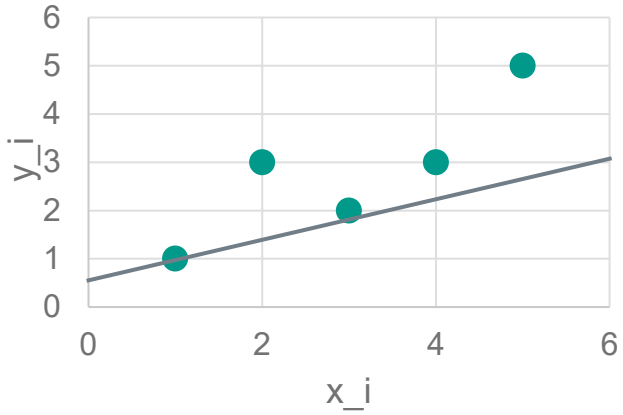
Lösung der linearen Regression mit Gradientenabstieg

Trainingsresultate ($\alpha = 0.01$)

Iteration	0	1	2	3	4	5	6	7	8	9	10
m	0,50	0,56	0,61	0,64	0,67	0,69	0,70	0,72	0,72	0,73	0,74
b	0,50	0,52	0,53	0,54	0,54	0,55	0,55	0,56	0,56	0,56	0,56
J	1,30	0,96	0,76	0,65	0,58	0,54	0,52	0,50	0,50	0,49	0,49

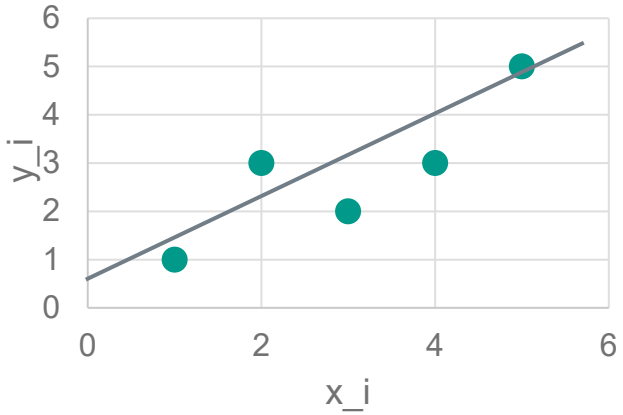
Regressionsgerade vor dem Training

$$\hat{y}_i = 0.5x_i + 0.5$$



Regressionsgerade nach dem Training

$$\hat{y}_i = 0.74x_i + 0.56$$



Linear Regression

Alternative Methode zur Lösung der linearen Regression: Berechnen Sie die exakte analytische Lösung

Transformieren Sie Daten und Gewichte in Matrizen/Vektoren und lösen Sie das Problem mit Matrixoperationen (lineare Algebra)

Funktioniert nur für lineare Regression (und nur für eine kleine Anzahl von Stichproben/Merkmalen)

$$X = \begin{bmatrix} 1 & x_1^1 & \cdots & x_1^n \\ 1 & \vdots & \ddots & \vdots \\ 1 & x_m^1 & \cdots & x_m^n \end{bmatrix}$$ – Merkmalsmatrix (m-Zeilen=Proben n-Spalten=Merkmale) (Spalte 0 enthält nur 1en da wir b zu unserem Gewichtsvektor hinzugefügt haben)

$$y = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}$$ – Ausgabevektor

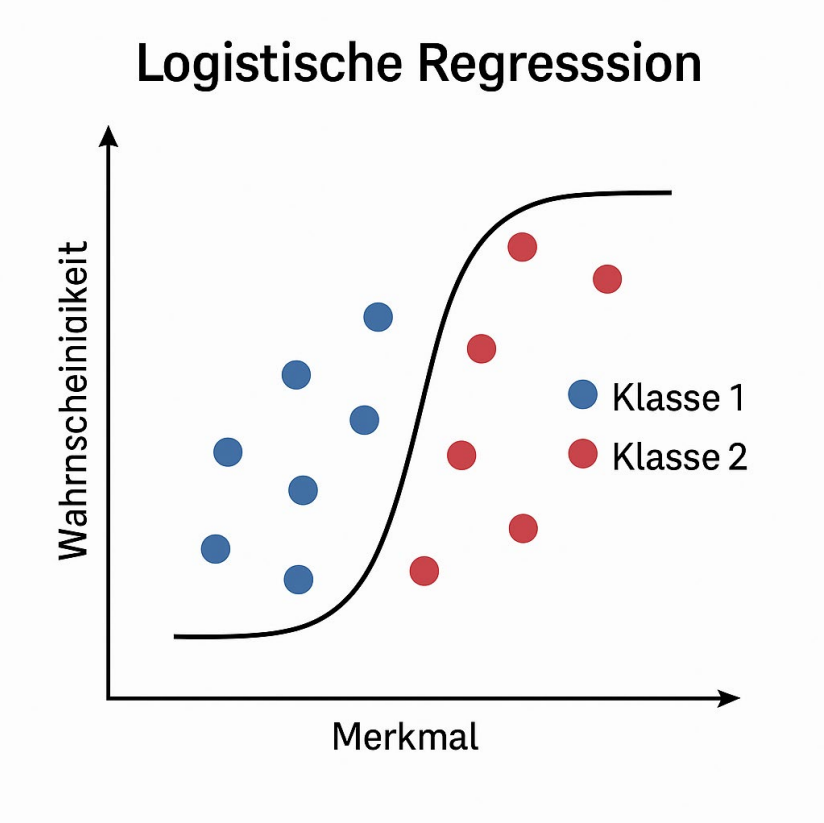
$$\theta = \begin{pmatrix} \theta_0 = b \\ \theta_1 \\ \vdots \\ \theta_n \end{pmatrix}$$ – Gewichtsvektor

Multiple lineare Regression (matrix-form): $y = X\theta$

Lösung: $\theta = (X^T X)^{-1} X^T y$

Linear Regression Übung 2

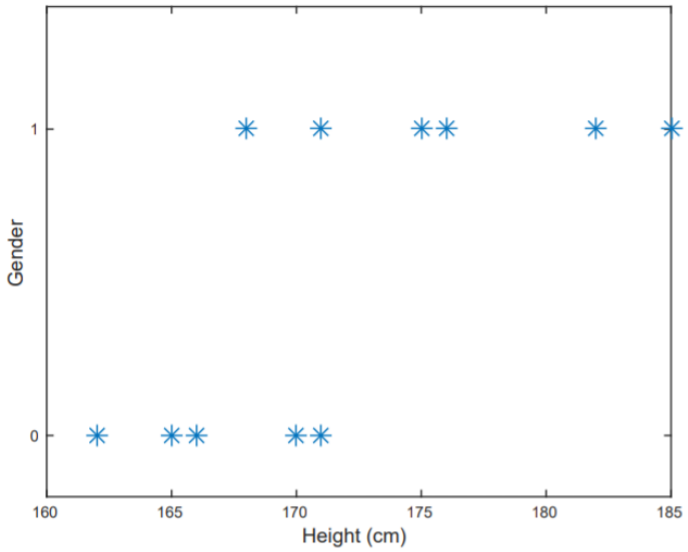
Berechnen Sie eine lineare Regression für den Titanic Datensatz

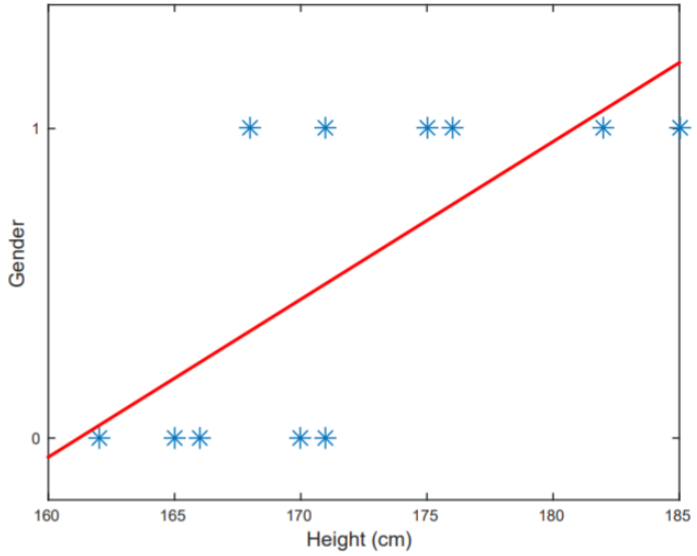


Logistic Regression

Klassifikation mit logistischer Regression

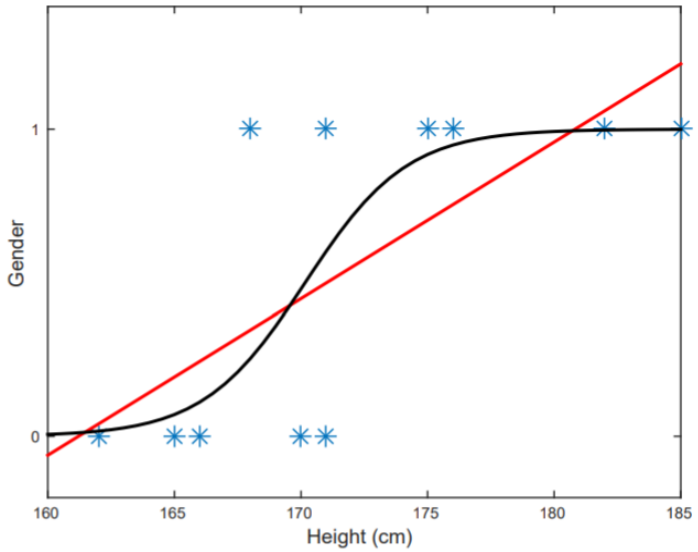
- Klassifikation kann als Regressionsproblem mit diskreten Ausgaben (Klassen) betrachtet werden.
- Beispiel: das Geschlecht einer Person basierend auf ihrer Körpergröße vorhersagen.
- Größen-Parameter: kontinuierlich numerisch unabhängig.
  Geschlecht: kategorial abhängig.
- Lineare Regression funktioniert hier nicht (siehe Grafiken).





Klassifikation mit logistischer Regression

- Idee: Stattdessen eine gekrümmte Funktion verwenden.
- Sigmoid-Funktion: $\hat{y}_i = g(z_i) = \frac{1}{1 + e^{-z_i}}$



- Alle Ausgabewerte liegen in einem festen Bereich: $0 < \hat{y}_i < 1$.
- Wenn $\hat{y}_i > 05$:Klasse 1 andernfalls Klasse 0.
- Wir verwenden die lineare Regressionsgleichung $z_i = \theta x_i + b$ weiter wenden anschließend jedoch die Sigmoid-Funktion $g(z_i)$an um die kurvenförmige Gestalt zu erhalten.
- Die Parameter θund $b$bestimmen die Lage und die Steilheit der Kurve.

Klassifikation mit logistischer Regression

Frage: Wie trainieren wir unser logistisches Modell auf die Daten?

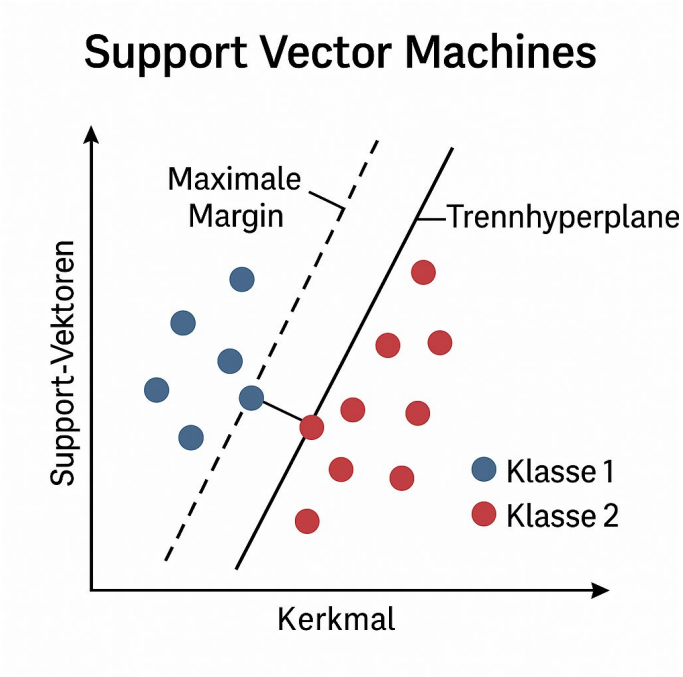
$$\hat{y}_i = g(z_i) = \frac{1}{1 + e^{-z_i}} \qquad z_i = \theta x_i + b$$

Antwort: Ähnlich wie bei der linearen Regression.

Wir verwenden Gradientenabstieg um die Parameter θund $b$zu finden die für unser Modell auf den Trainingsdaten den geringsten Fehler Jerzeugen.

Allerdings nutzen wir für Klassifikation eine andere Fehlerfunktion: Kreuzentropie-Verlust (Cross-Entropy Loss).

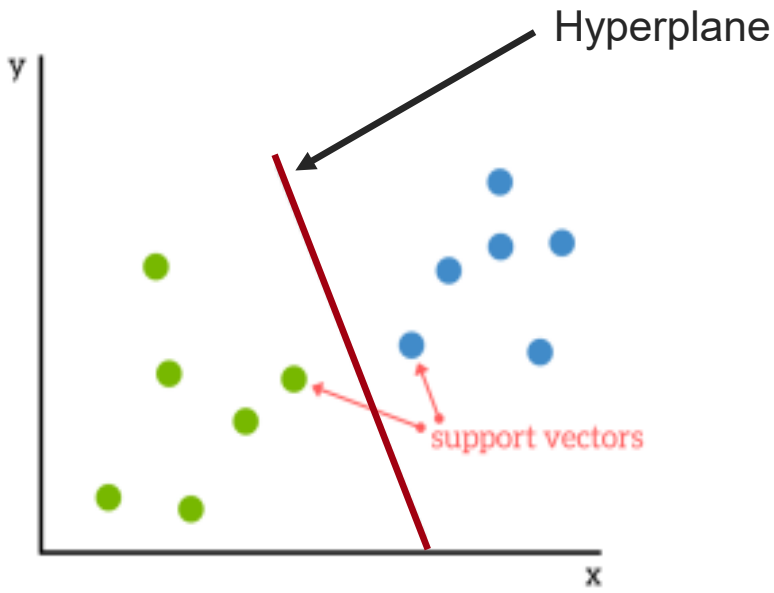
$$J = \frac{-1}{n} \sum_{i=1}^{n} y_i \cdot \log \hat{y}_i + (1 - y_i) \cdot \log(1 - \hat{y}_i)$$



Support Vector Machines (SVM)

Support Vector Machines

- Klassifikationsalgorithmus der Vorhersagen auf Basis einer Hyperebene trifft, die die Klassen in den Daten linear trennt.
- Standardmäßig nur für binäre - zwei-Klassen - Klassifikationsprobleme verwendbar.
- Es gibt Erweiterungen für den Multiklassen-Fall.
- Hyperbene: n-dimensionale Ebene die die beiden Klassen trennt.
- Ziel: Die Ebene soll möglichst weit von den nächstgelegenen Datenpunkten (Stützvektoren) entfernt liegen.
- Daher maximieren wir die Abstände (Margins) zwischen der Hyperbene und den Stützvektoren.

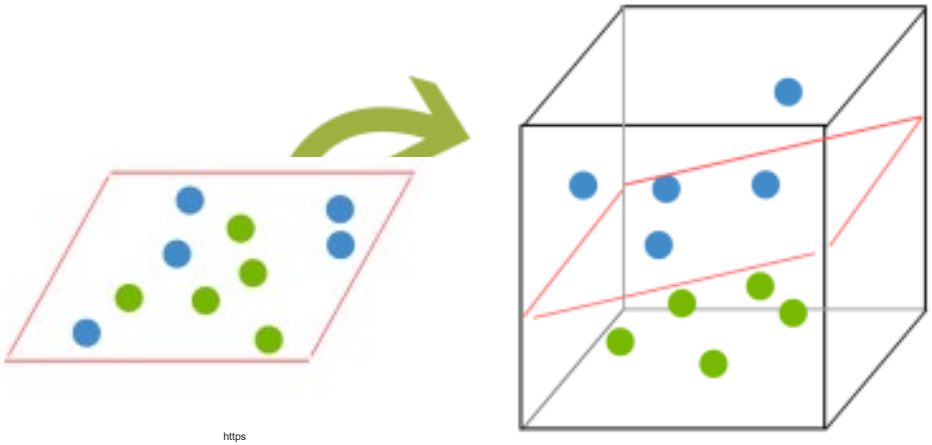


https://www.kdnuggets.com/2016/07/support-vector-machines-simple-explanation.html

Support Vector Machines

Was tun wenn der Datensatz nicht linear trennbar ist?

- Wir wenden eine Kernelfunktion an um die Daten in einen höherdimensionalen Raum zu transformieren in dem sie trennbar sind („Kernel-Trick“).
- Problem: Die explizite Transformation ist rechnerisch sehr aufwendig.



https

Beispiel Iris-Datensatz

- 150 Datensätze 3 Klassen (Setosa Versicolor Virginica)
- 4 Merkmale: Kelchblattlänge, Kelchblattbreite, Blütenblattlänge, Blütenblattbreite
- Modell: Support Vector Machine (SVM) mit RBF-Kernel
- Vorverarbeitung: Standardisierung der Merkmale
- Ziel: Trennung der drei Iris-Klassen (Setosa, Versicolor, Virginica)
- Optimierung über Kreuzvalidierung (3-fach)
- Gewählte Parameter: C = 10, gamma = 'scale'

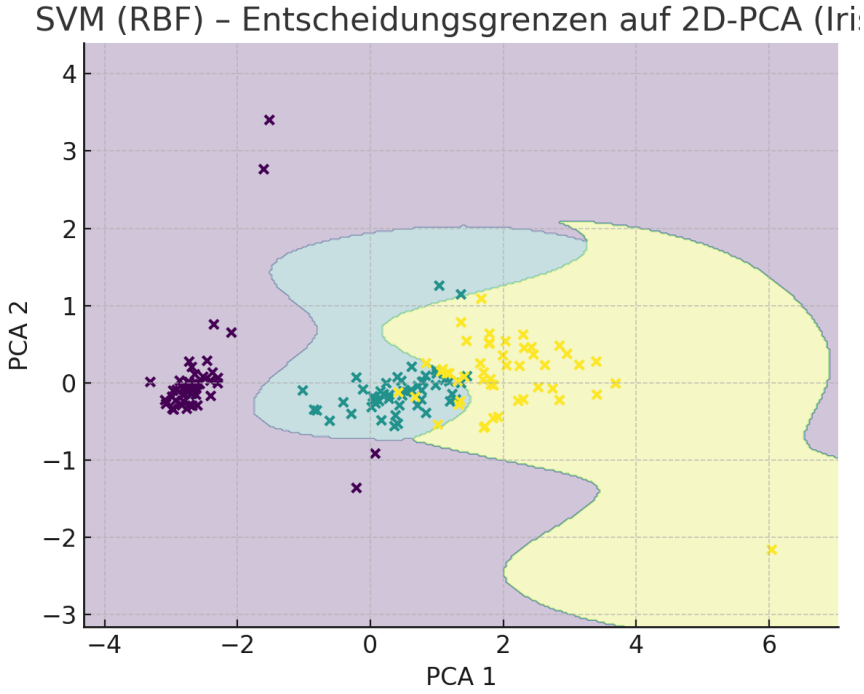
Ergebnis

- Klassifikationsgenauigkeit (CV): ca. 97–99 %
- Sehr gute Trennung zwischen allen drei Klassen
- Nur wenige Verwechslungen zwischen Versicolor und Virginica

SVM – Entscheidungsgrenzen (PCA 2D)

- Die SVM trennt die Iris-Arten nahezu perfekt.
- Petal-Attribute (Länge, Breite) sind die wichtigsten Merkmale.
- Das Modell veranschaulicht klar die lineare Trennbarkeit von Setosa und die nichtlineare Grenze zwischen Versicolor und Virginica.
- PCA 1 und PCA 2
  - Die ersten zwei Hauptkomponenten der Merkmale (sepal length, sepal width, petal length, petal width).
  - Sie fassen die vier ursprünglichen Merkmale zu zwei Dimensionen zusammen, die den größten Teil der Varianz im Datensatz erklären.

- Jede Beobachtung (eine Iris-Blüte). Drei Farben → drei Klassen:
  - Lila = *Iris setosa*
  - Türkis = *Iris versicolor*
  - Gelb = *Iris virginica*



Support Vector Machines

Vorteile

Im Allgemeinen: sehr gute Klassifikationsleistung.

SVMs eignen sich auch für kleine Datensätze.

Für das Training benötigen wir nur eine Teilmenge der Punkte (die Stützvektoren).

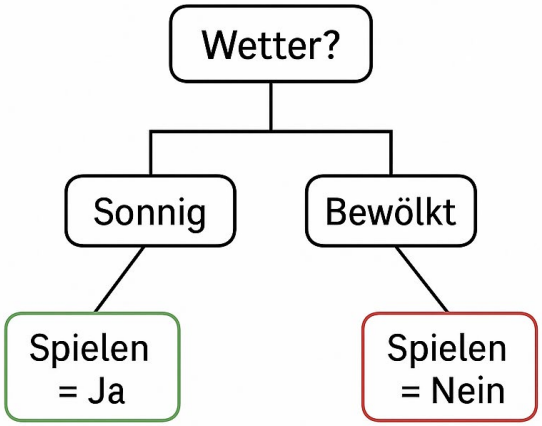
Nachteile

Erfordert einen sehr sauberen Datensatz um gute Ergebnisse zu erzielen (Datenvorbereitung ist entscheidend).

Sehr empfindlich gegenüber rauschbehafteten/überlappenden Klassenverteilungen.

Für sehr große Datensätze oft ungeeignet da das Training sehr langsam ist.

Entscheidungsbäume



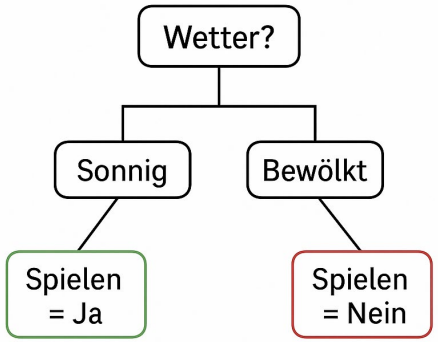
Jeder Knoten teilt die Daten nach einem Attributwert.

Decision Tree Models / Entscheidungsbäume

Decision Trees

- Leistungsfähiger Algorithmus für Klassifikations- und Regressionsprobleme.
- Regressionsbäume liefern kontinuierliche Ausgabewerte Klassifikationsbäume kategoriale Ausgaben.
- Kann lineare und nichtlineare Zusammenhänge in Daten lernen.

Entscheidungsbäume



Jeder Knoten teilt die Daten nach einem Attributwert.

Das Training von Entscheidungsbäumen erzeugt eine Top-down-Struktur binärer Entscheidungen:

- Jede Entscheidung teilt den Eingabedatensatz in zwei Teilmengen die in den entstehenden Unterästen weiterverarbeitet werden.
- Der Aufspaltungsprozess läuft weiter bis ein vordefiniertes Abbruchkriterium erreicht ist.

Die resultierende Baumstruktur überfittet das Trainingsset in vielen Fällen und muss durch Pruning (Beschneiden) reduziert werden.

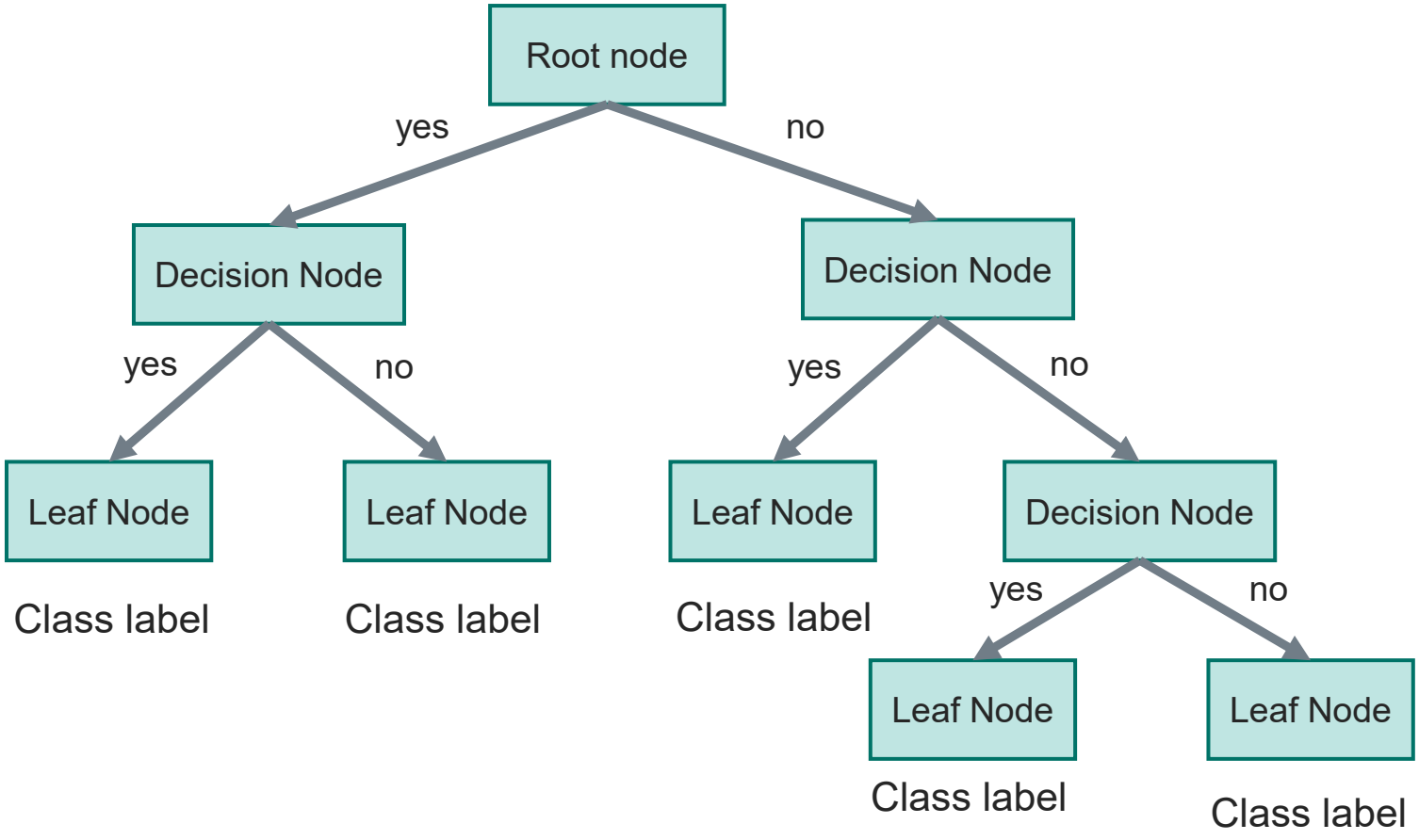
Decision Trees

- Entscheidungsbaum-Methoden konstruieren ein Modell von Entscheidungen, die auf den tatsächlichen Attributwerten in den Daten beruhen.
- Die Entscheidungen verzweigen sich in Baumstrukturen bis für einen gegebenen Datensatz eine Vorhersage getroffen wird.
- Entscheidungsbäume werden für Klassifikations- und Regressionsprobleme auf Daten trainiert.
- Aufgrund ihrer Einfachheit sind Entscheidungsbäume oft schnell und genau und daher ein großer Favorit im Machine Learning.

Die beliebtesten Entscheidungsbaum-Algorithmen sind:

- CART (Classification and Regression Tree)
- ID3 (Iterative Dichotomiser 3)
- C4.5 und C5.0 (verschiedene Versionen eines leistungsfähigen Ansatzes)
- CHAID (Chi-squared Automatic Interaction Detection)
- Decision Stump
- M5
- Konditionale Entscheidungsbäume (Conditional Decision Trees)

Beispiel eines Entscheidungsbaums

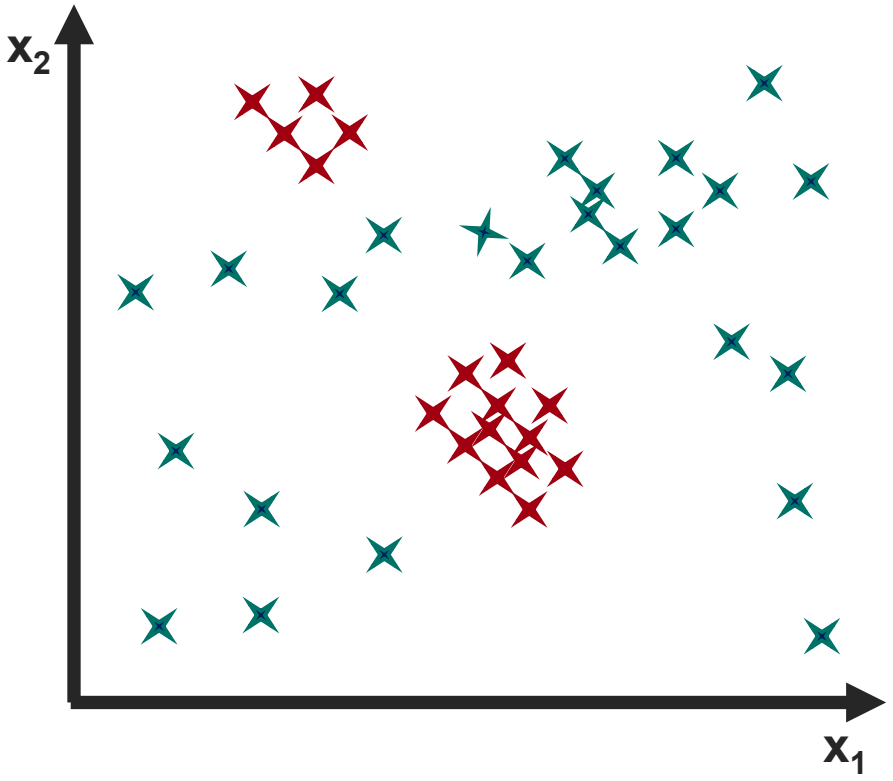


Decision Trees - Training

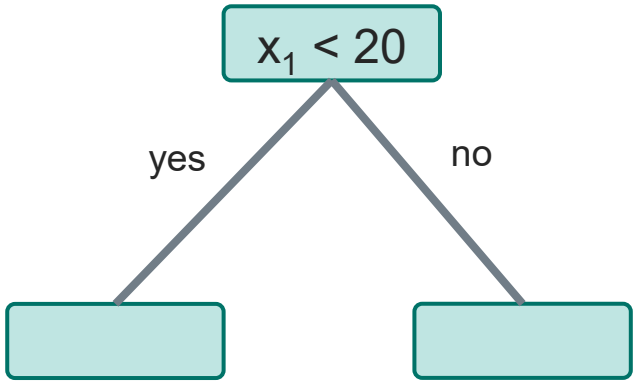
Beispiel:
Einfaches binäres Klassifikationsproblem.

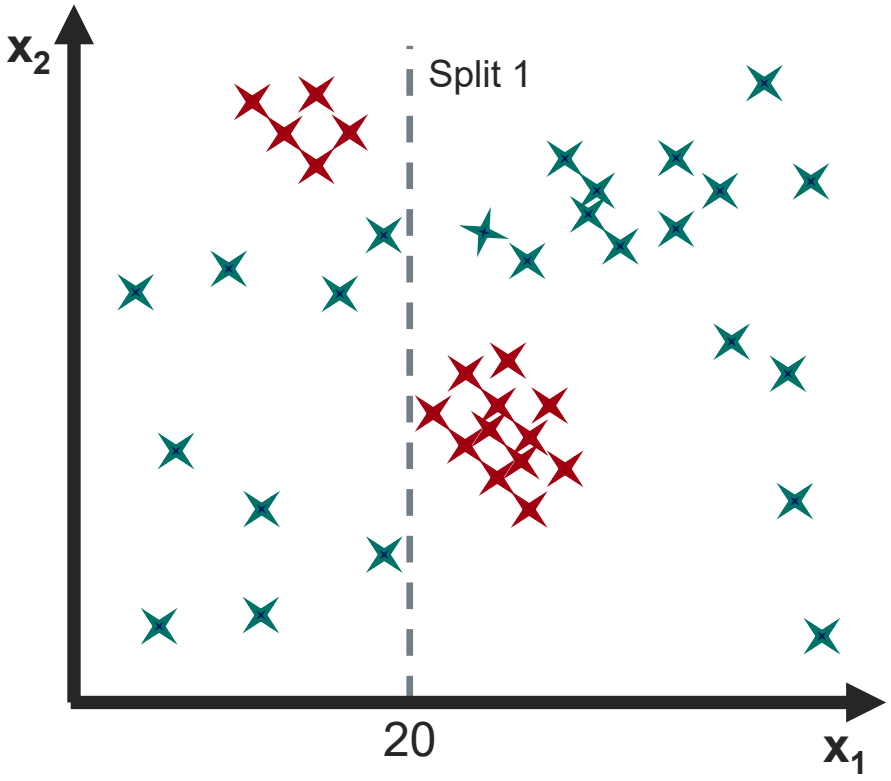
Der Plot zeigt dass die Daten nicht linear trennbar sind.

Ein Entscheidungsbaum kann die nichtlineare Beziehung in den Daten lernen.

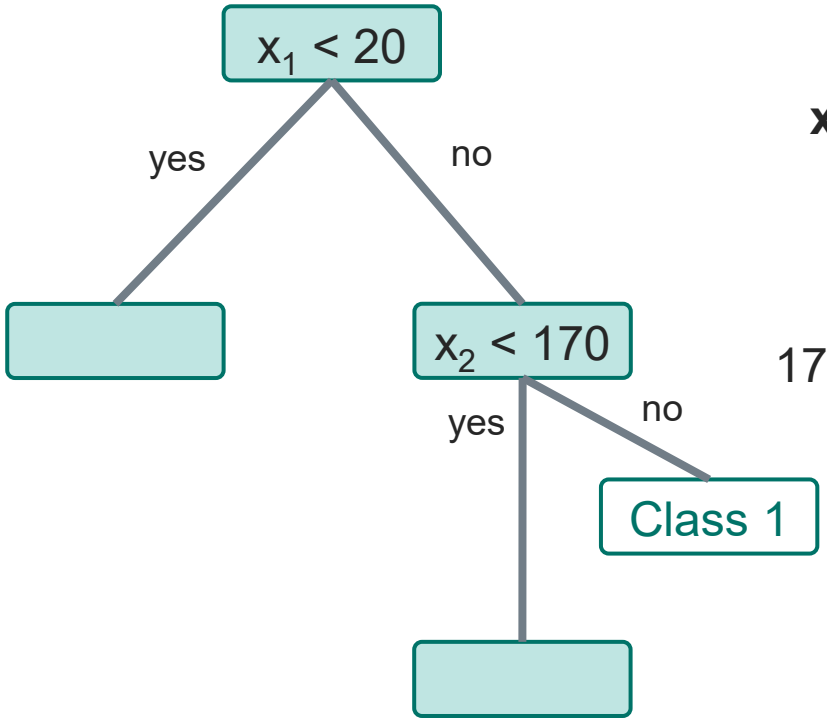


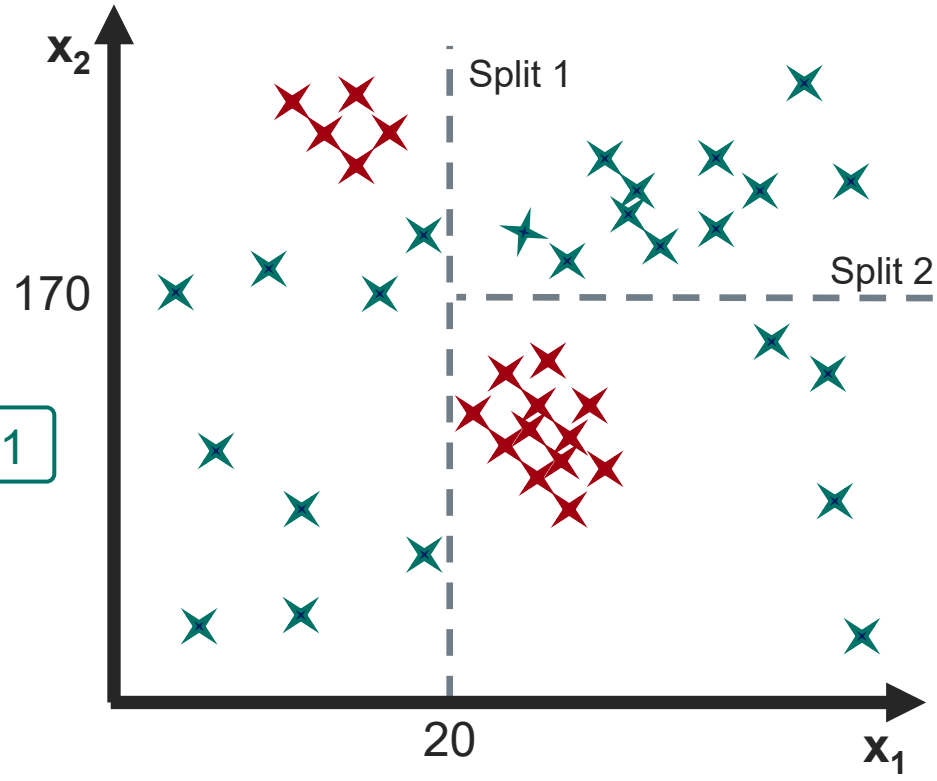
Decision Trees - Training



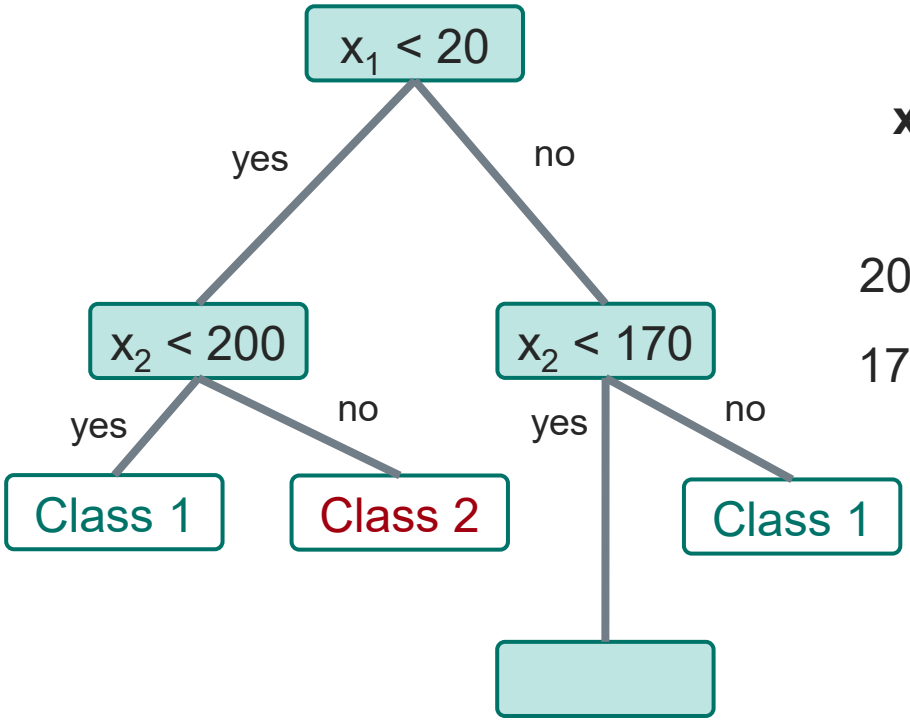


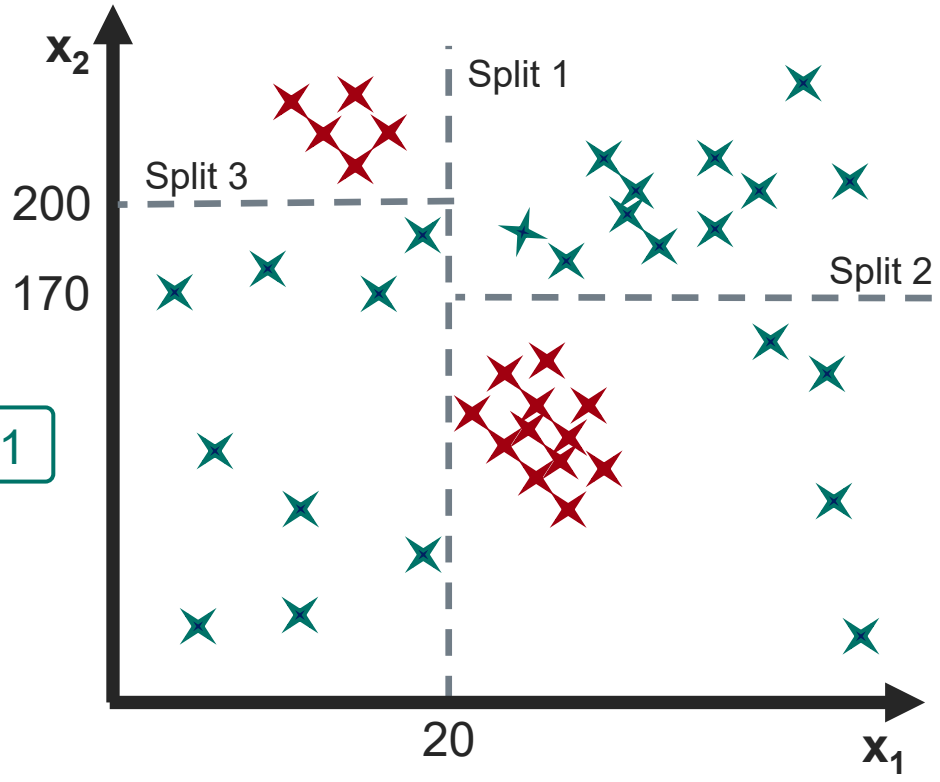
Decision Trees - Training



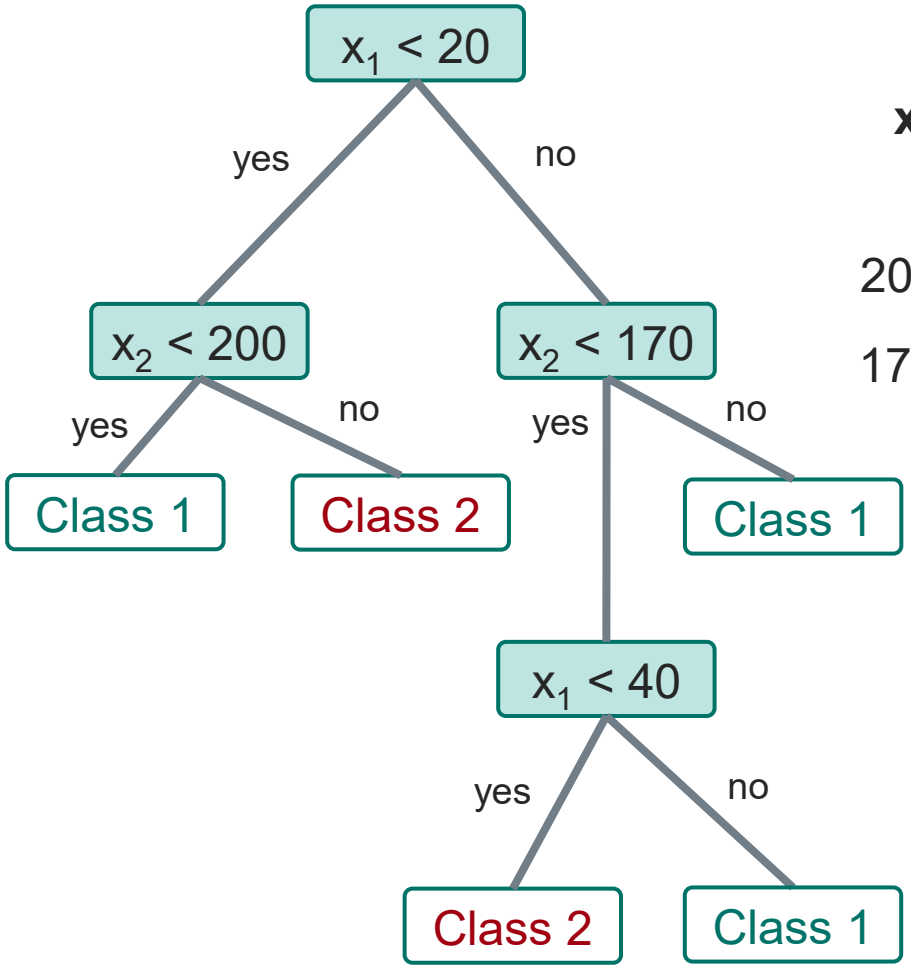


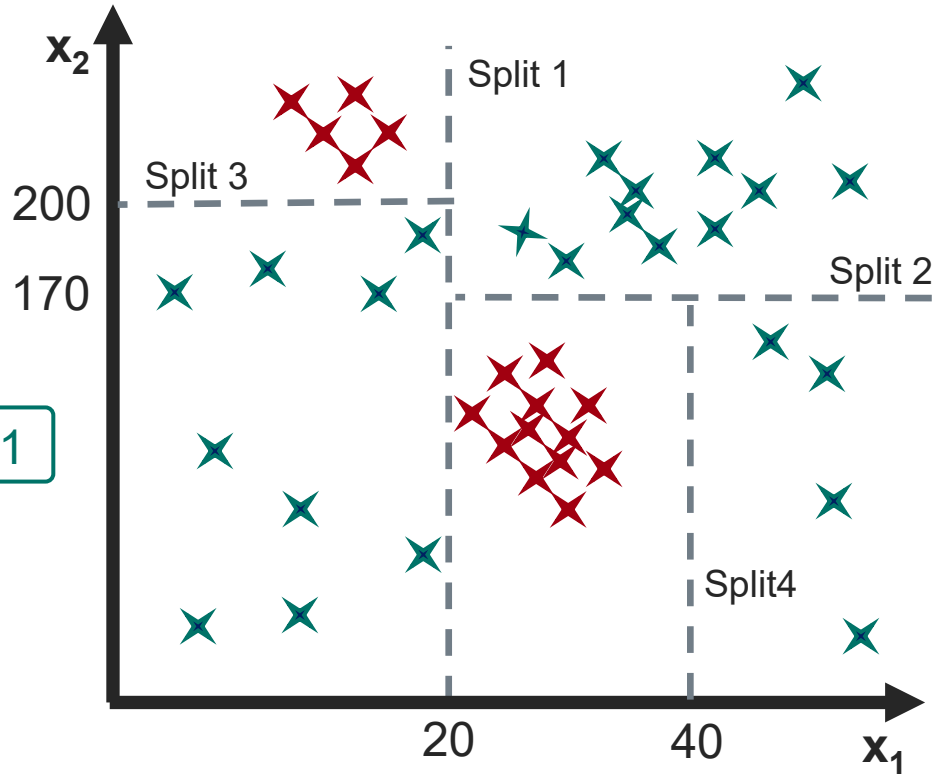
Decision Trees - Training



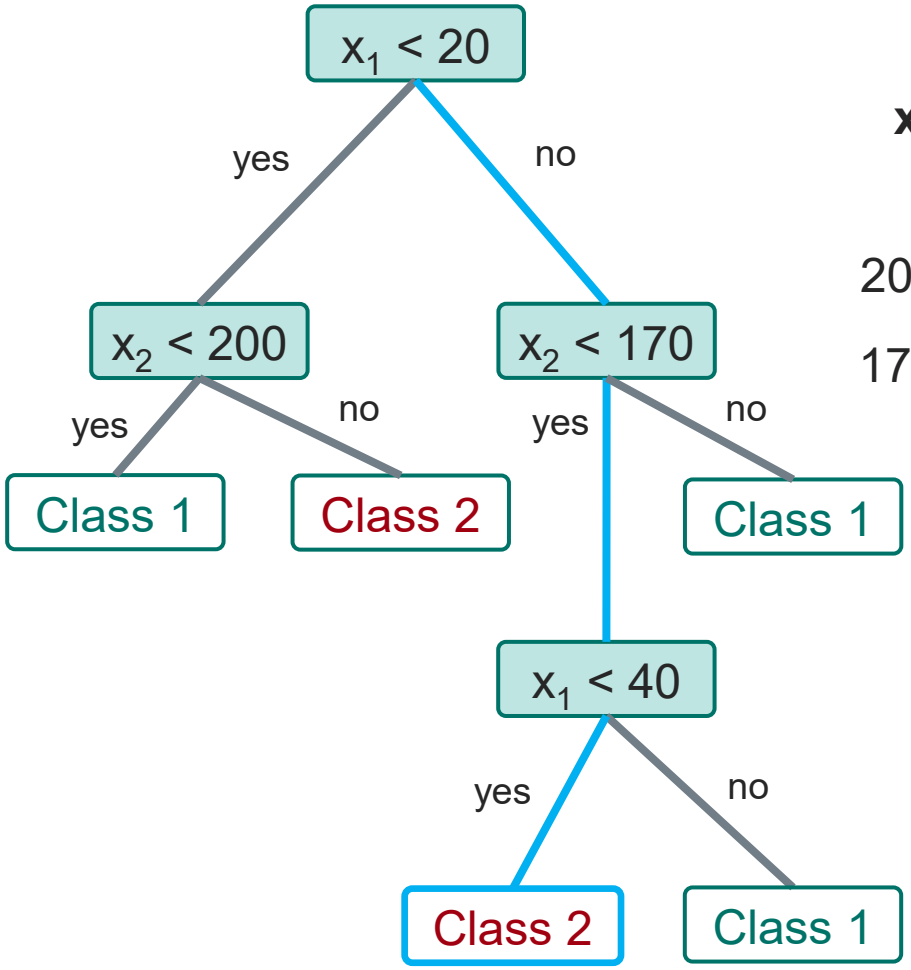


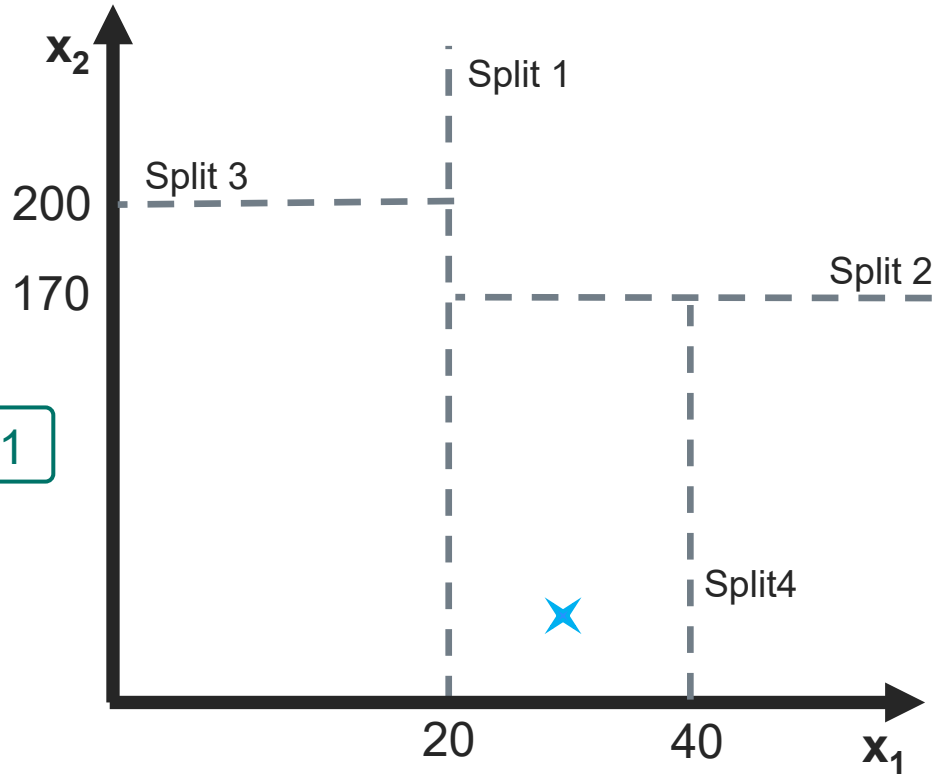
Decision Trees - Training





Decision Trees - Test





Beispiel Durchführung einer Klassifizierung eines Objektes mit (30, 30)

Attributauswahl

- Wir müssen entscheiden welches Attribut wir an jedem Knoten zum Aufteilen des Datensatzes verwenden.
- In der Praxis testen man Splits für jedes Attribut und bewertet anschließend die entstehenden Teilmengen - wie „rein“ ein Datensatz nach einer Aufteilung ist.
- Reinheit: Grad zu dem die Daten in einem Teil (Knoten) einer einzigen Klasse angehören.
  - Hohe Reinheit: Alle Elemente im Knoten gehören derselben Klasse an.
  - Geringe Reinheit: Die Daten im Knoten sind gemischt also enthalten mehrere Klassen.
- Der Algorithmus versucht durch weitere Aufteilungen die Reinheit zu erhöhen also die Knoten homogener zu machen.
- Beliebte Bewertungsmetriken (Beispiele)
  - Informationsgewinn (Entropy/Information Gain)
  - Gini-Impurity
  - χ^2-Test (CHAID)
  - (für Regression) MSE/MAE

Gini Index

$$G = 1 - \sum_{c} p(c)^2$$

- p(c) = Anteil der Klasse c in einem Teil des Datensatzes.
- Misst wie stark die Klassen gemischt sind.
- Erzwingt die Erstellung größerer Datenteilungen
- P(c) bester Wert = 0 → vollständige Trennung erreicht → perfekte Reinheit.
- Ziel: Minimierung von Unreinheit

Entropy

$$E = \sum_{c} -p(c)\log(p(c))$$

- Misst die Unordnung oder Unsicherheit in den Daten.
- Erzwingt die Erstellung kleinerer Partitionen mit vielen unterschiedlichen Werten kleinere Werte sind besser
- Ist 0 wenn nur eine Klasse vorkommt (keine Unsicherheit).
- Kleinere Werte durch die Aufteilung erreichen → mehr Ordnung weniger Unsicherheit.
- Ziel: Maximierung des Informationsgewinns

Ausgangssituation (Iris-Datensatz)

- 150 Datensätze 3 Klassen (Setosa Versicolor Virginica)
- 4 Merkmale: Kelchblattlänge, Kelchblattbreite, Blütenblattlänge, Blütenblattbreite
- Ziel: Attribut finden, das die Daten am besten trennt
- Der Algorithmus sortiert die Werte, etwa von Blütenblattlänge, aufsteigend und berechnet die Mittelpunkte zwischen benachbarten Werten, wo sich die Klasse ändert.
- Der Algorithmus testet also „petal length ≤ 3.1“, weil hier eine Trennlinie zwischen Klassen im Datenraum liegt.
  - (1.5+4.7)/2=3.1
- Ausgangs-Gini: 0.666

Index	Blütenblattlänge (cm)	Klasse
1	1.3	setosa
2	1.5	setosa
3	4.7	versicolor
4	5.0	virginica

Vergleich der besten Splits je Merkmal

Merkmal	Bester Wert	Gini nach Split	Gini Gewinn	N l	N r	Gini l	Gini l	Verteilung l	Verteilung r
Blütenblattlänge (cm)	2,4500	0,3700	0,2963	46	101	0,0425	0,5192	iris-setosa: 45 iris-versicolor: 1 iris-virginica: 0	iris-setosa: 2 iris-versicolor: 50 iris-virginica: 49
Blütenblattbreite (cm)	0,8568	0,3892	0,2771	44	103	0,0444	0,5365	iris-setosa: 43 iris-versicolor: 1 iris-virginica: 0	iris-setosa: 4 iris-versicolor: 50 iris-virginica: 49
Kelchblattlänge (cm)	5,4500	0,4629	0,2034	49	98	0,2791	0,5548	iris-setosa: 41 iris-versicolor: 7 iris-virginica: 1	iris-setosa: 6 iris-versicolor: 44 iris-virginica: 48
Kelchblattbreite (cm)	3,3500	0,5644	0,1018	113	34	0,6296	0,3478	iris-setosa: 20 iris-versicolor: 48 iris-virginica: 45	iris-setosa: 27 iris-versicolor: 3 iris-virginica: 4

l … links

r … rechts

$$G_{\text{split}} = \frac{46}{147} \cdot 0.0425 + \frac{101}{147} \cdot 0.5192 \approx 0.3700$$

Bester erster Split nach Gini

Merkmal: petal_length

- Schwelle: ≤ 2.45
- Gewichteter Gini nach Split: 0.370

$$G_{\text{split}} = \frac{46}{147} \cdot 0.0425 + \frac{101}{147} \cdot 0.5192 \approx 0.3700$$

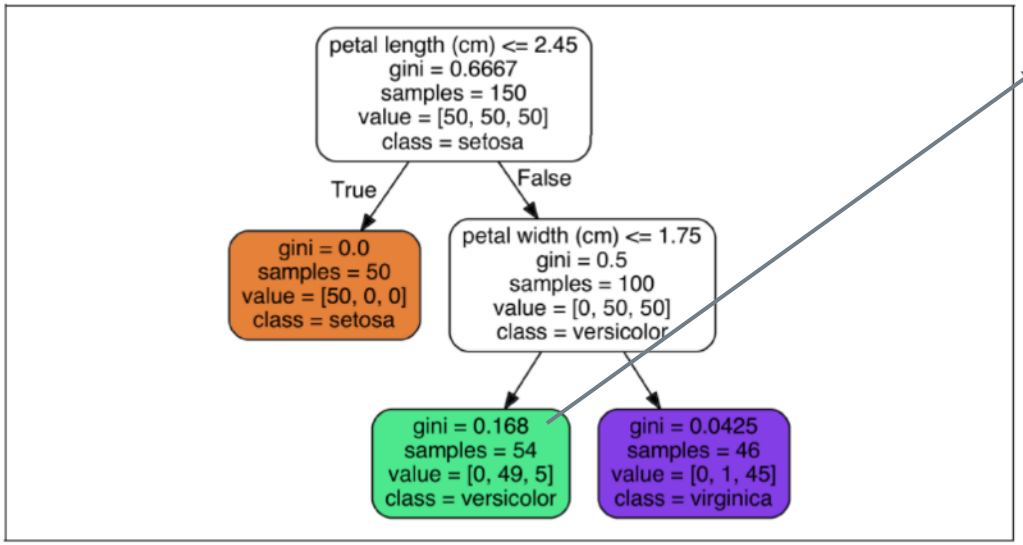
- Gini-Gewinn: 0.296
- Verteilung links (≤ 2.45): 46 Samples
  - → Setosa: 45 Versicolor: 1 Virginica: 0
  - Gini(left) = 0.0425
- Verteilung rechts (> 2.45): 101 Samples
  - → Setosa: 2 Versicolor: 50 Virginica: 49
  - Gini(right) = 0.5192

Vergleich der besten Splits je Merkmal

1. petal_length ≤ 2.45
→ gewichteter Gini = 0.370
2. petal_width ≤ 0.857
→ gewichteter Gini = 0.389
3. sepal_length ≤ 5.45
→ gewichteter Gini = 0.463
4. sepal_width ≤ 3.35
→ gewichteter Gini = 0.564

⇒ Beste Trennung durch petal_length ≤ 2.45

Entscheidungsbäume - Beispiel Iris



$$1 - (0/54)^2 - (49/54)^2 - (5/54)^2 \approx 0.168.$$

Formel 6-1: Gini-Unreinheit

$$G_i = 1 - \sum_{k=1}^{n} p_{i,k}^{\ 2}$$

$$\left(-\frac{49}{54}\log\left(\frac{49}{54}\right) - \frac{5}{54}\log\left(\frac{5}{54}\right)\right) \approx 0.31.$$

Formel 6-3: Entropie

$$H_i = -\sum_{\substack{k=1 \\ p_{i,k} \neq 0}}^{n} p_{i,k}\log\left(p_{i,k}\right)$$

Abbildung 6-1: Iris-Entscheidungsbaum

[GE S. 167ff]

samples – Anzahl der gültigen Trainingsdaten im Knoten.

value – Anzahl der Trainingsbeispiele im Knoten **je Klasse**.

Beispiel-Reihenfolge: $setosaversicolorvirginica$ (häufige Schreibweise: virginica).

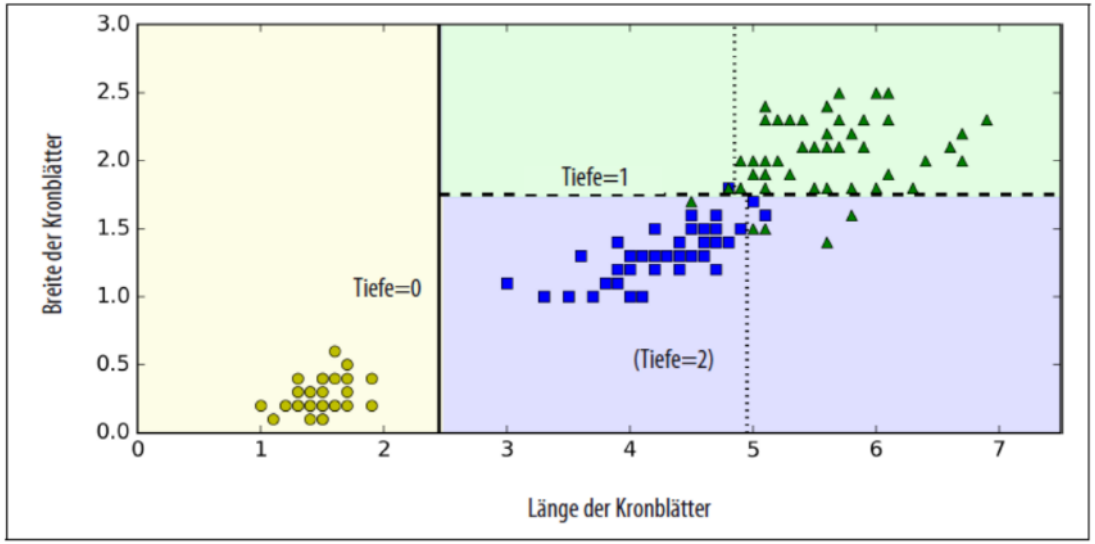
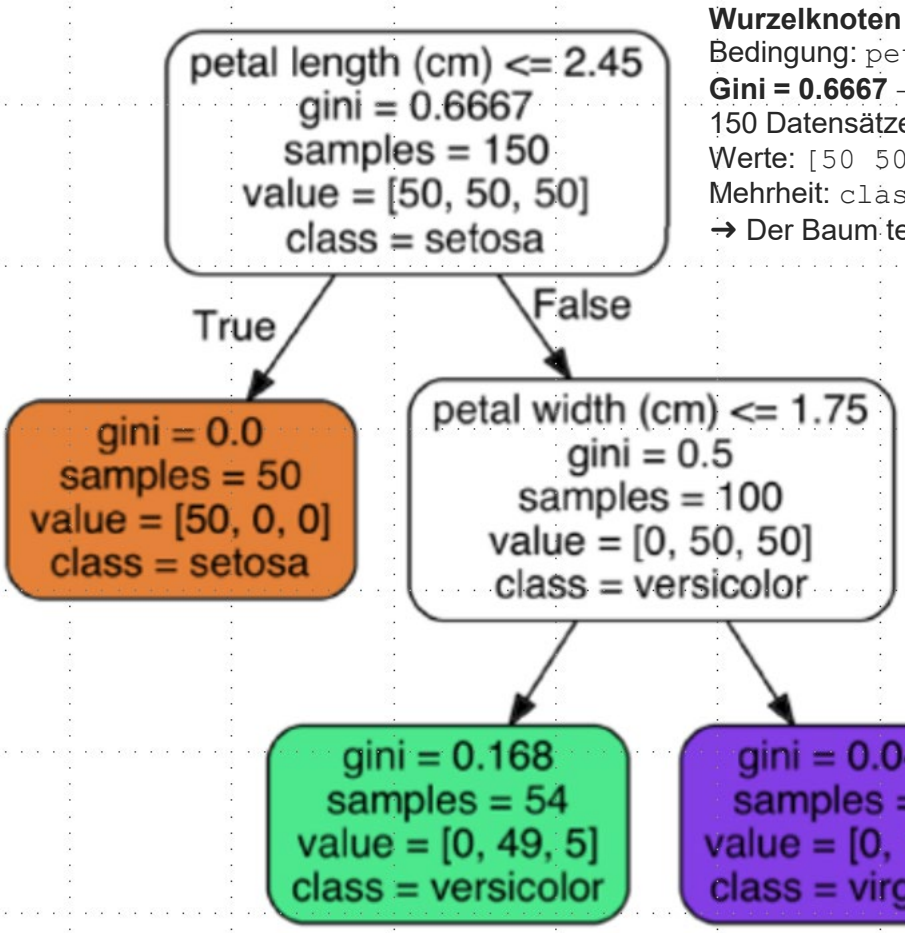


Abbildung 6-2: Entscheidungsgrenzen in einem Entscheidungsbaum

Entscheidungsbäume - Beispiel Iris



Wurzelknoten (oben)
Bedingung: `petal length (cm) <= 2.45`
Gini = 0.6667 → hohe Unreinheit weil alle drei Klassen gemischt vorkommen.
150 Datensätze insgesamt (`samples = 150`)
Werte: `[50 50 50]` → je 50 pro Klasse
Mehrheit: `class = setosa`
➜ Der Baum teilt also zunächst nach der Blütenblattlänge.

Linker Ast (True-Zweig)
Bedingung erfüllt: `petal length <= 2.45`
Gini = 0.0 → perfekte Reinheit
50 Datensätze alle Klasse *setosa* (`value = [50 0 0]`)
➜ **Alle Blüten mit kurzen Blütenblättern sind Setosa.**

Rechter Ast (False-Zweig)
Bedingung nicht erfüllt: `petal length > 2.45`
100 Datensätze verbleiben (`samples = 100`)
Werte: `[0 50 50]` → Mischung aus *versicolor* und *virginica*
Gini = 0.5 → noch unrein
➜ Der Baum teilt weiter nach **petal width (cm)**.

Rechter Teil – weitere Aufteilung
Bedingung: `petal width <= 1.75`

a) True-Zweig:
54 Datensätze fast alle *versicolor*
`value = [0 49 5]` → **Gini = 0.168**
Noch etwas Unreinheit da 5 *virginica* enthalten sind.

b) False-Zweig:
46 Datensätze fast alle *virginica*
`value = [0 1 45]` → **Gini = 0.0425**
Fast perfekte Reinheit.

Der Entscheidungsbaum hat gelernt dass
kurze Blütenblätter → *Setosa*
mittlere Blütenblätter mit schmaler Breite → *Versicolor*
lange und breite Blütenblätter → *Virginica* sind.

Overfitting

Entscheidungsbäume überanpassen die Daten in fast allen Fällen.
Daher nimmt man beim Training Anpassungen vor um die Größe (Komplexität) des Baums zu begrenzen.

Grenzen/Constraints setzen

- Minimale Stichprobenzahl für einen Knotensplit
- Minimale Stichprobenzahl pro Blattknoten
- Maximale Baumtiefe
- Maximale Anzahl an Baumknoten

Beschneiden (Pruning) von Teilbäumen anwenden

- Einen vollständigen Baum ohne Einschränkungen aufbauen.
- Die am wenigsten nützlichen End-Teilbäume anhand der Attributauswahl-Kriterien identifizieren und entfernen.