

# Machine Learning 1

Dr. Harald Bögeholz

Oktober 2023

# Was ist Machine Learning?

Computer lernen aus Daten, Aufgaben zu lösen, ohne explizit dafür programmiert zu sein.

- Spam-Filter
- Bilder klassifizieren
- Tumore auf Röntgenbildern erkennen
- Spracherkennung
- Umsatzprognosen treffen
- Kreditkartenbetrug erkennen
- Empfehlungssysteme
- Künstliche Intelligenz für Spiele (Schach, Go, ...)
- Autonomes Fahren
- Chatbots

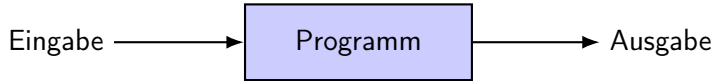
**Überwachtes Lernen:** Eingaben und die zugehörigen Ausgaben sind bekannt. Daraus lernt das System, zu unbekannten Eingaben korrekte Ausgaben zu generieren.

**Unüberwachtes Lernen:** Die Daten sind nicht mit zugehörigen Ausgaben versehen. Das System erkennt Zusammenhänge und Muster in den Daten.

**Verstärkendes Lernen:** Ein Agent lernt durch Beobachten seiner Umgebung die Wirkung seines Handelns und optimiert eine Belohnungsfunktion.

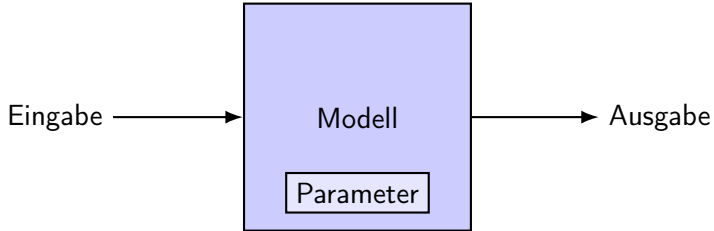
- Klassifikation** **Diskrete Zielgröße:** Die Ausgabe ist eine von endlich vielen Klassen (Hund, Katze, Maus, ...)
- Regression** **Kontinuierliche Zielgröße:** Die Ausgabe ist ein Zahlenwert (Preis, Umsatz, Temperatur, ...)

# Herkömmliche Programmierung vs. Machine Learning



**Herkömmliche Programmierung:** Wir schreiben ein Programm.

# Herkömmliche Programmierung vs. Machine Learning



## Machine Learning:

- Wir *wählen* ein Modell.
- Wir *trainieren* das Modell anhand von Trainingsdaten, d.h. wir bestimmen die Parameter des Modells.
- Wir *evaluieren* das Modell anhand von Testdaten.

Sprachmodell GPT-4, die Basis von ChatGPT von OpenAI:

- Eingabe: Text
- Ausgabe: Text
- Parameter: 8 Modelle mit jeweils 220 Milliarden Parametern

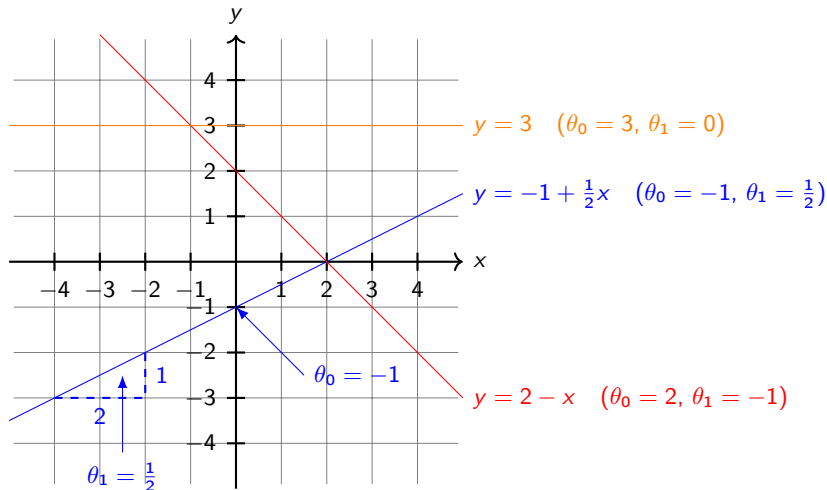
Quelle: <https://medium.com/@mlubbad/the-ultimate-guide-to-gpt-4-parameters-everything-you-need-to-know-about-nlps-game-changer-109b8767855a>



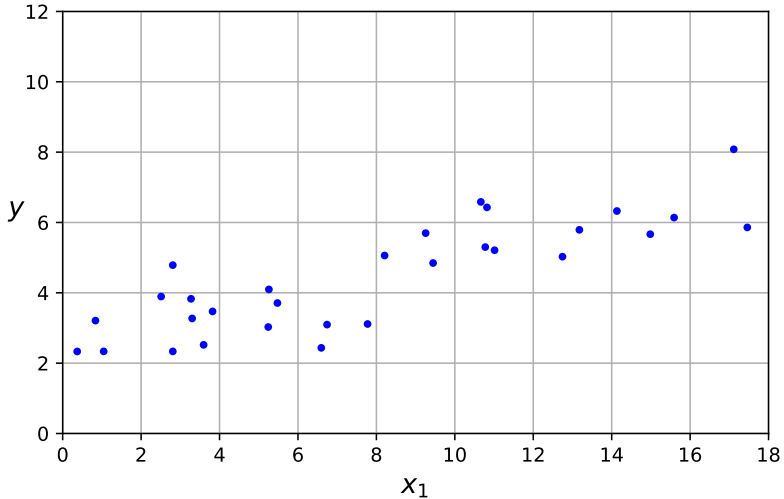
Einfaches Beispiel: Lineares Modell  $y = \theta_0 + \theta_1 x$ .

- Eingabe: Eine reelle Zahl  $x$
- Ausgabe: Eine reelle Zahl  $y$
- Parameter: Zwei reelle Zahlen  $\theta_0, \theta_1$ , die  $y$ -Achsenabschnitt und Steigung einer Geraden bestimmen.

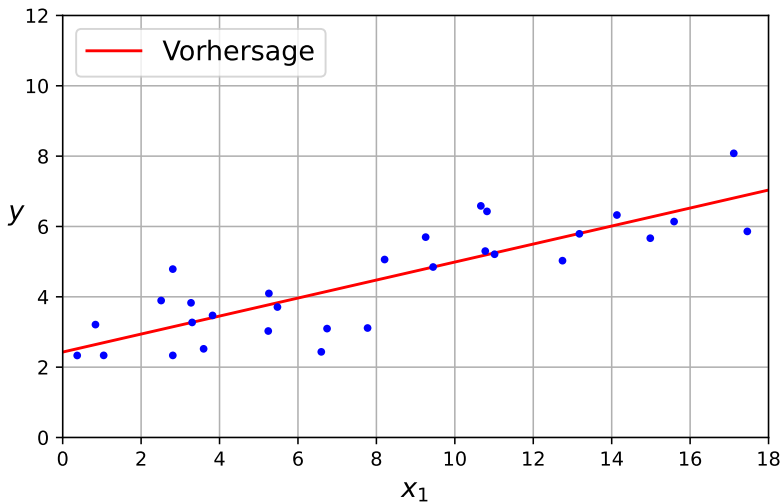
# Geradengleichung



# Beispiel: Lineare Regression



# Beispiel: Lineare Regression



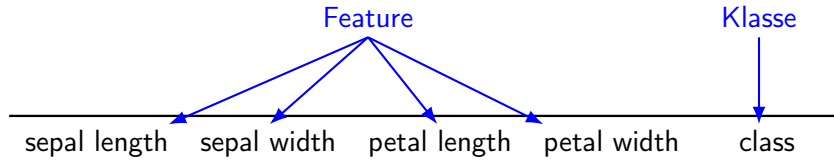
**Datensatz:** Eine Menge an Datenpunkten (Samples)

**Datenpunkt (Sample):** Beim überwachten Lernen eine Eingabe mit zugehöriger Ausgabe

**Merkmal (Feature):** Eine Eingabe kann aus mehreren Werten bestehen. Jeder einzelne ist ein Merkmal (Feature)

**Klasse:** Ein möglicher Wert für die Zielgröße (bei Klassifizierung)

# Beispiel: Der Iris-Datensatz



The diagram illustrates the structure of the Iris dataset. A horizontal line separates the header from the data. Above the line, the word "Feature" is centered, with four arrows pointing down to the first four columns: "sepal length", "sepal width", "petal length", and "petal width". To the right, the word "Klasse" is centered, with one arrow pointing down to the "class" column. On the left side, the word "Datenpunkt" is followed by an arrow pointing to the first data row.

	sepal length	sepal width	petal length	petal width	class
	5.8	2.7	5.1	1.9	virginica
	5.4	3.9	1.7	0.4	setosa
Datenpunkt →	6.3	2.9	5.6	1.8	virginica
	5.8	4.0	1.2	0.2	setosa
	5.5	2.4	3.7	1.0	versicolor
	7.6	3.0	6.6	2.1	virginica
	6.9	3.1	4.9	1.5	versicolor
	⋮	⋮	⋮	⋮	⋮

- $n$  ist die Anzahl der Features
- $m$  ist die Anzahl der Datenpunkte
- $x^{(i)}$  ist ein Vektor mit allen Features des  $i$ -ten Datenpunktes (Input-Variablen).  
Beispiel:

$$x^{(1)} = \begin{pmatrix} 5.8 \\ 2.7 \\ 5.1 \\ 1.9 \end{pmatrix}$$

- $x^T$  steht für den transponierten Vektor  $x$ , d.h. aus einer Spalte wird eine Zeile:

$$(x^{(1)})^T = (5.8 \quad 2.7 \quad 5.1 \quad 1.9)$$

- $X$  ist eine Matrix mit den Feature-Werten aller Datenpunkte. Jeder Datenpunkt ist eine Zeile in der Matrix.

$$X = \begin{pmatrix} (x^{(1)})^T \\ (x^{(2)})^T \\ (x^{(3)})^T \\ \vdots \\ (x^{(m)})^T \end{pmatrix} = \begin{pmatrix} 5.8 & 2.7 & 5.1 & 1.9 \\ 5.4 & 3.9 & 1.7 & 0.4 \\ 6.3 & 2.9 & 5.6 & 1.8 \\ 5.8 & 4.0 & 1.2 & 0.2 \\ \vdots & \vdots & \vdots & \vdots \end{pmatrix}$$

- $y^{(i)}$  ist der gewünschte Output (Label) des  $i$ -ten Datenpunktes. Beispiel:

$$y^{(1)} = 2 \quad (\text{der Label virginica ist als die Zahl 2 codiert})$$

- $y$  ist der Vektor aller Outputs  $y^{(i)}$ .



- Lineare Regression:  $\hat{y} = h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n$ .
- Die Schreibweise  $\hat{y}$  bezeichnet den vom Modell vorhergesagten Wert, während  $y$  der tatsächliche Wert eines Datenpunktes ist.
- Die Funktion  $h_{\theta}(x)$  heißt auch Hypothesen-Funktion. Sie ist parameterisiert durch den Vektor

$$\theta = \begin{pmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{pmatrix}$$

Die Hypothesen-Funktion der linearen Regression lässt sich auch als Skalarprodukt von Vektoren bzw. als Matrixmultiplikation formulieren. Wir ergänzen dazu den Feature-Vektor  $x$  um ein Element  $x_0 = 1$ . Dann haben wir

$$\hat{y} = h_{\theta}(x) = \theta^T x = (\theta_0 \quad \theta_1 \quad \theta_2 \quad \cdots \quad \theta_n) \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$$

Ein Maß für den Fehler eines Regressionsmodells ist die *mittlere quadratische Abweichung* (Mean Square Error, MSE):

$$\text{MSE}(X, h_{\theta}) = \frac{1}{m} \sum_{i=1}^m \left( h_{\theta}(x^{(i)}) - y^{(i)} \right)^2$$

Besser interpretierbar ist die Wurzel der mittleren quadratischen Abweichung (Root Mean Square Error, RMSE):

$$\text{RMSE}(X, h_{\theta}) = \sqrt{\text{MSE}(X, h_{\theta})} = \sqrt{\frac{1}{m} \sum_{i=1}^m \left( h_{\theta}(x^{(i)}) - y^{(i)} \right)^2}$$

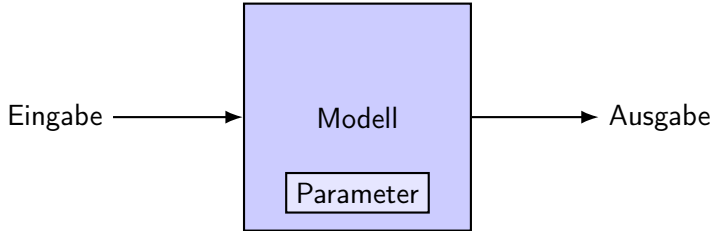
Ein anderes Maß ist die *mittlere absolute Abweichung* (Mean Absolute Error, MAE):

$$\text{MAE}(X, h_\theta) = \frac{1}{m} \sum_{i=1}^m \left| h_\theta(x^{(i)}) - y^{(i)} \right|$$

Es gilt immer

$$\text{MAE}(X, h_\theta) \leq \text{RMSE}(X, h_\theta)$$

Der mittlere quadratische Fehler gewichtet größere Abweichungen stärker als kleinere. Er ist das bevorzugte Gütemaß bei Regression; im Falle der linearen Regression führt er auch zu einer eleganten mathematischen Lösung.



## Machine Learning:

- Wir *wählen* ein Modell.
- Wir *trainieren* das Modell anhand von Trainingsdaten, d.h. wir bestimmen die Parameter des Modells.
- Wir *evaluieren* das Modell anhand von Testdaten.

- Lineares Modell:  $h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n$
- Trainieren heißt: Bestimme den Parameter-Vektor  $\theta$  so, dass der mittlere quadratische Fehler  $\text{MSE}(X, h_{\theta})$  minimiert wird.
- Hierfür fügen wir der Feature-Matrix  $X$  ein Dummy-Feature hinzu, das überall den Wert 1 hat:

$$X = \begin{pmatrix} 1 & x_1^{(1)} & x_2^{(1)} & x_3^{(1)} & \dots & x_n^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & x_3^{(2)} & \dots & x_n^{(2)} \\ 1 & x_1^{(3)} & x_2^{(3)} & x_3^{(3)} & \dots & x_n^{(3)} \\ 1 & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(m)} & x_2^{(m)} & x_3^{(m)} & \dots & x_n^{(m)} \end{pmatrix}$$

Für die Minimierung des MSE gibt es eine geschlossene Formel. Das Optimum  $\hat{\theta}$  errechnet sich als

$$\hat{\theta} = (X^T X)^{-1} X^T y.$$

- Die einzelnen Faktoren sind Matrizen, die per Matrix-Multiplikation multipliziert werden. **In Numpy:** Der Operator @ multipliziert Matrizen.
- $X^T$  steht für die zu  $X$  transponierte Matrix, d.h. die Matrix wird an der Diagonalen gespiegelt, Zeilen und Spalten tauschen ihre Rollen. **In Numpy:** `X.T`.
- $X^{-1}$  ist die Inverse der Matrix  $X$ . **In Numpy:** `np.linalg.inv(X)`.

- Zum Beurteilen der **Qualität** des Modells kann man das *Bestimmtheitsmaß* (Determinationskoeffizient, Coefficient of determination,  $R^2$ ) heranziehen.
- **Idee:** Vergleiche die Fehlerquadratsumme des Modells mit der Varianz der Daten
- **Definition:**

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=1}^m (y^{(i)} - \hat{y}^{(i)})^2}{\sum_{i=1}^m (y^{(i)} - \bar{y})^2}$$

Dabei ist  $\bar{y}$  der Mittelwert des Vektors  $y$ .

- **Vorteil:** Es ist ein Prozent-Wert unabhängig von der Größenordnung der Daten, daher leichter interpretierbar



Interpretation des Bestimmungsmaßes  $R^2$ :

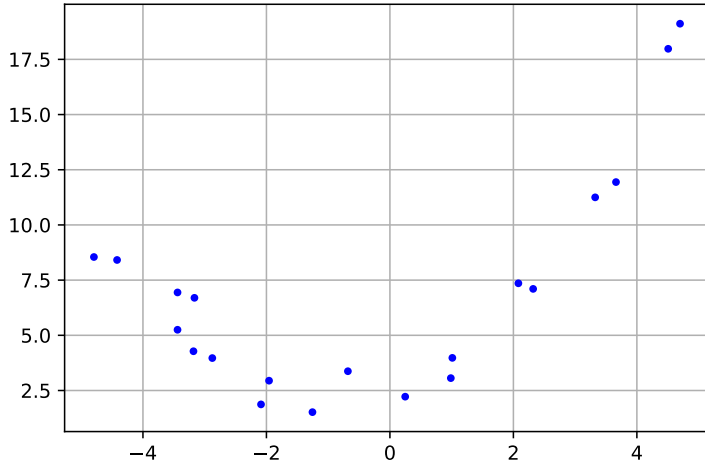
$R^2 = 0$ : Modell ist nur so gut wie der Mittelwert.

$R^2 > 0$ : Modell ist besser als der Mittelwert.

$R^2 = 1$ : Modell ist perfekt.

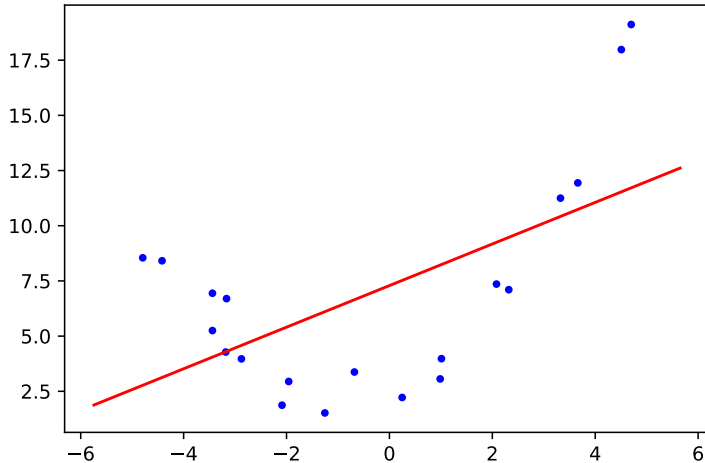
$R^2 < 0$ : Negative Werte sind möglich; Modell ist schlechter als Mittelwert.

# Polynomiale Regression



# Polynomiale Regression

Eine Gerade passt nicht auf jeden Datensatz:



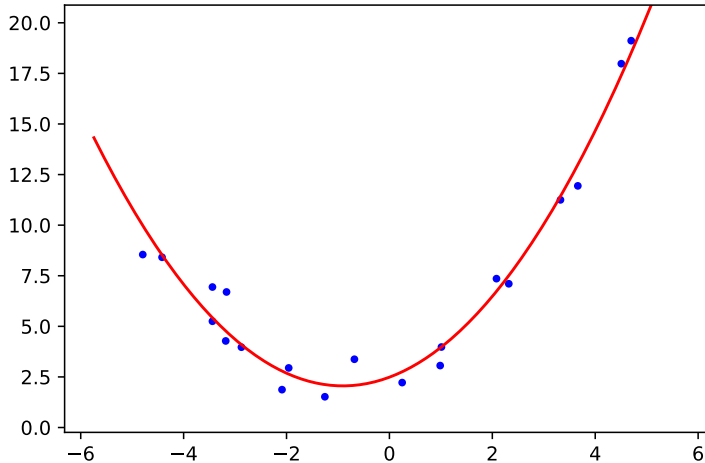
Wenn  $x$  ein einzelnes Feature ist (kein Vektor), können wir als Modell ein Polynom  $n$ -ten Grades wählen:

$$h_{\theta}(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \dots + \theta_n x^n$$

Wir bestimmen die Parameter  $\theta_i$ , indem wir lineare Regression auf die Features  $x, x^2, \dots, x^n$  anwenden.

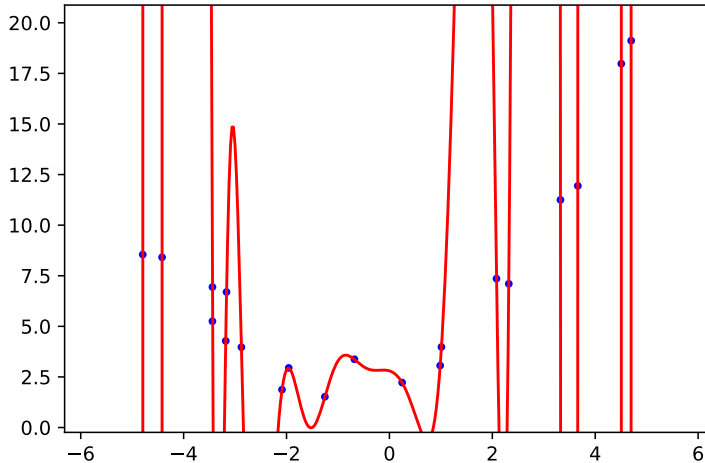
# Polynomiale Regression

Auf diesen Datensatz passt besser ein Polynom 2. Grades:



# Überanpassung

Was passiert, wenn wir auf 20 Punkte ein Polynom 20. Grades anpassen?



Überanpassung (overfit):

- Das Modell passt sehr gut zu den Trainingsdaten
- Das Modell ist schlecht darin, auf neue Daten zu verallgemeinern

Unteranpassung (underfit):

- Das Modell sagt schon die Trainingsdaten nicht gut voraus
- Daher wird es wahrscheinlich auch für neue Daten nicht gut funktionieren.

Wie gut verallgemeinert ein Modell auf neue Daten? Das kann man nur mit neuen Daten herausfinden. Daher:

- **Ganz zu Beginn**, vor jeglicher Analyse, Datensatz aufteilen in
  - Trainingsdaten (z.B. 80 %)
  - Testdaten (z.B. 20 %)
- Wichtig: Testdaten müssen zufällig ausgewählt sein, nicht einfach nur die letzten 20 % nehmen!
- Jegliche Analyse nur mit Trainingsdaten
- Training von Modellen nur mit Trainingsdaten
- **Ganz am Ende**: Validieren des Modells mit Testdaten



- Was tun bei Overfitting?
- Wir schränken das Modell ein, damit es einfacher wird (Regularisierung).
- Ridge Regression: Wir optimieren nicht den MSE allein, sondern fügen unserer Kostenfunktion einen Regularisierungsterm hinzu. Die neue Kostenfunktion lautet

$$J(\theta) = \text{MSE}(\theta) + \alpha \sum_{i=1}^n \theta_i^2$$

- $\alpha$  ist ein *Hyperparameter*.

## Parameter

- Interne Variablen eines Modells
- Während des Trainings gelernt
- Beispiel: Die Parameter  $\theta_0, \theta_1, \dots, \theta_n$  der linearen Regression
- Automatische Anpassung durch Algorithmen

## Hyperparameter

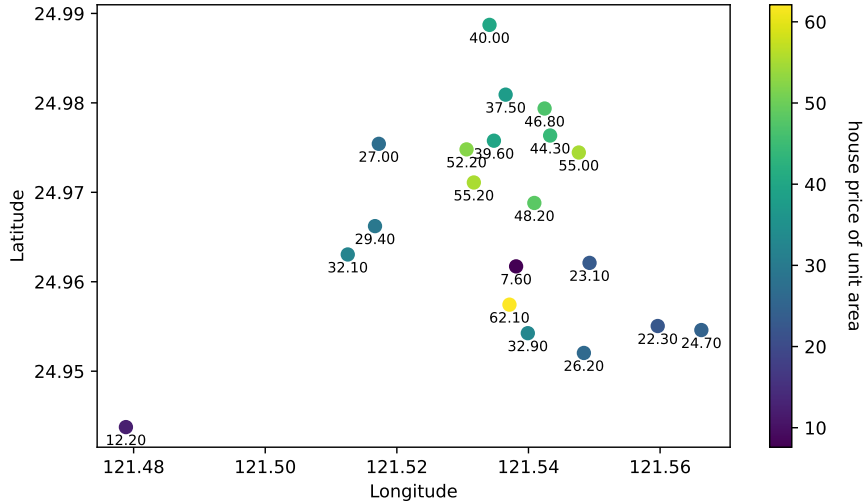
- Vor dem Training festgelegt
- Beeinflusst das Training
- Zur Vorhersage nicht mehr benutzt
- Beispiel: Der Parameter  $\alpha$  bei der Ridge Regression
- Einstellung durch Erfahrung oder automatisierte Suche

# Parameter vs. Hyperparameter

In scikit-learn werden Hyperparameter dem Modell beim Anlegen des Modell-Objekts mitgegeben. Parameter kann man nach dem Lernen abfragen:

```
>>> from sklearn import linear_model
>>> reg = linear_model.Ridge(alpha=.5)
>>> reg.fit([[0, 0], [0, 0], [1, 1]], [0, .1, 1])
Ridge(alpha=0.5)
>>> reg.coef_
array([0.34545455, 0.34545455])
>>> reg.intercept_
0.13636...
```

# Auszug aus Immobilien-Datensatz



# $k$ -Nächste Nachbarn ( $k$ -Nearest Neighbors)

$k$ -Nächste Nachbarn-Algorithmus:

- Vergleiche neuen Datenpunkt mit komplettem Datensatz
- Finde die  $k$  Datenpunkte mit der kleinsten Distanz
- Bilde den Mittelwert der Zielvariablen dieser  $k$  Datenpunkte. Dies ist der Ausgabewert.

Variationsmöglichkeiten:

- Verwendung verschiedener Distanz-Funktionen
- Mittelwert gewichten nach Distanz der gefundenen Nachbarn zum Eingabewert

Der Abstand zweier Punkte ist die Norm ihrer Differenz:

$$d(p, q) := \|p - q\|$$

Wenn nichts anderes gesagt wird, versteht man unter der Norm die *euklidische Norm*. Die euklidische Norm eines Vektors  $a = (a_1, a_2, \dots, a_n)$  ist wie folgt definiert:

$$\|a\| := \sqrt{\sum_{i=1}^n a_i^2}$$

- $\ell^1$ -Norm, auch *Manhattan-Distanz* genannt:

$$\|a\|_1 = \sum_{i=1}^n |a_i|$$

- $\ell^p$ -Norm:

$$\|a\|_p = \sqrt[p]{\sum_{i=1}^n |a_i|^p}$$

- *Supremumsnorm*  $\ell^\infty$ :

$$\|a\|_\infty = \max_i |a_i|$$

Je höher das  $p$ , desto stärker werden größere Komponenten eines Vektors in der  $\ell^p$ -Norm gewichtet.

- Manhattan-Distanz ( $\ell^1$ ): Alle Komponenten sind gleichwertig, die Norm ist die Summe der Absolutwerte
- Euklidische Norm ( $\ell^2$ ): Durch das Quadrieren erhalten größere Komponenten stärkeres Gewicht
- $\ell^p$ -Norm: Je größer  $p$ , desto mehr Gewicht auf größeren Komponenten
- $\ell^\infty$  ist der Extremfall. Hier kommt es nur noch auf die größte Komponente eines Vektors an.



$k$ -NN funktioniert am besten, wenn alle Features skaliert sind. Das funktioniert wie folgt:

- Berechne für jedes Feature  $x_i$  den Mittelwert  $\mu_i$  und die Standardabweichung  $\sigma_i$ .
- Ersetze jedes Feature  $x_i$  durch  $(x_i - \mu_i)/\sigma_i$ .
- Jetzt hat jedes Feature den Mittelwert 0 und die Standardabweichung 1.

scikit-learn stellt hierfür eine Klasse `StandardScaler` bereit.

- $k$ -NN beruht auf dem Abstand des Eingabewerts zu den gelernten Datenpunkten.
- Die Abstandsfunktion (Norm der Differenz) für Vektoren behandelt alle Komponenten gleich.
- Wenn eine Komponente eine größere Standardabweichung hat, erhält diese Komponente mehr Gewicht beim Abstand, die Abstandsfunktion wird verzerrt.
- Beispiel: In unserem Immobilien-Datensatz (siehe Übungen) hat das Feature  $x_3$  eine Standardabweichung von 1216, während die nächst kleinere Standardabweichung (Feature  $x_2$ ) mit dem Wert 11 erheblich kleiner ist. Dadurch wird die Abstandsfunktion praktisch nur das Feature  $x_3$  berücksichtigen.

- Erster Schritt ist immer die Aufteilung in Trainings- und Testdaten
- Für die Skalierung nur Mittelwert und Standardabweichung der Trainings-Daten verwenden
- Trainings- und Testdaten anschließend mit denselben Parametern skalieren

Mit scikit-learn:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Bei der Klassifikation ist die Zielgröße diskret, das heißt es gibt nur wenige mögliche Werte. Im einfachsten Fall gibt es nur Ja oder Nein, dann spricht man von *binärer Klassifikation*.

Beispiele:

- Alltag: Ist eine E-Mail Spam?
- Medizin: Handelt es sich bei einer Gewebeprobe um Brustkrebs?
- Industrie: Zeigt ein Bild ein Teil, das den Qualitätsanforderungen genügt?

Zur Beurteilung der Qualität eines binären Klassifikators können wir die Konfusionsmatrix verwenden. Hier ein Beispiel dafür, wie das Ergebnis einer Vorhersage aussehen könnte:

Tatsächlich	Vorhersage	
	Ja	Nein
Ja	15	5
Nein	9	21

Tatsächlich	Vorhersage	
	Ja	Nein
Ja	TP	FN
Nein	FP	TN

**TP** True Positive, richtig-positive Vorhersagen

**FP** False Positive, falsch-positive Vorhersagen

**FN** False Negative, falsch-negative Vorhersagen

**TN** True Negative, richtig-negative Vorhersagen

**ACHTUNG:** Die Konfusionsmatrix kann sehr verwirrend sein, weil verschiedene Autoren die Werte verschieden anordnen.

- Tatsächlich und Vorhersage können vertauscht sein. Dann ist die Matrix an der Diagonalen gespiegelt.
- Ja und Nein können vertauscht sein. Dann ist die Matrix an der anderen Diagonalen gespiegelt.
- Daher immer genau hinsehen: Wo ist ja/nein bzw. 1/0 und wo ist tatsächlich vs. Vorhersage.

Für die Beurteilung der Qualität eines Klassifikators werden verschiedene Metriken herangezogen. Um Missverständnisse zu vermeiden, nennen wir hier immer auch die englischen Begriffe.

- Genauigkeit (accuracy)
- Präzision (precision)
- Sensitivität (recall)
- $F_1$ -Score



$$\begin{aligned}\text{accuracy} &= \frac{\text{Zahl korrekter Vorhersagen}}{\text{Gesamtzahl Vorhersagen}} \\ &= \frac{TP + TN}{TP + TN + FP + FN}\end{aligned}$$

- Wie viel Prozent der Vorhersagen sind richtig?
- In skikit-learn: `from sklearn.metrics import accuracy_score`

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

- Unter den Vorhersagen, die Ja lauten, wie viel Prozent davon sind richtig? Oder: Wenn die Vorhersage Ja lautet, mit welcher Präzision tut sie das?
- In scikit-learn: `from sklearn.metrics import precision_score.`

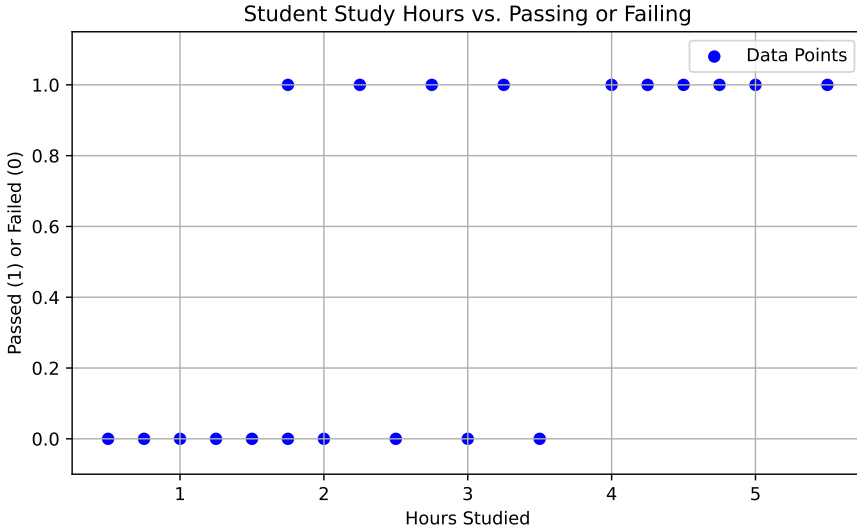
$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

- Wie viel Prozent der positiven Fälle werden korrekt erkannt?
- In scikit-learn: `from sklearn.metrics import recall_score.`

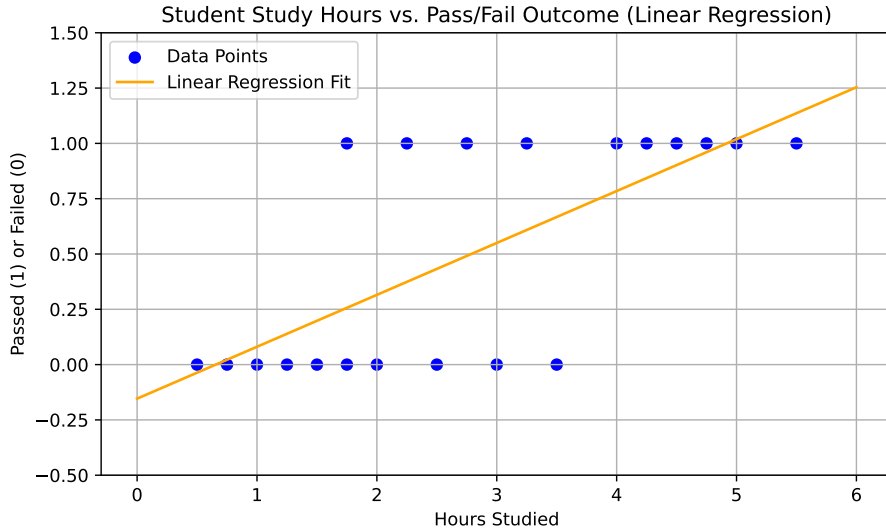
$$\begin{aligned} F_1 &= \frac{2}{\frac{1}{\text{precision}} + \frac{1}{\text{recall}}} \\ &= \frac{2 \cdot \text{TP}}{2 \cdot \text{TP} + \text{FN} + \text{FP}} \end{aligned}$$

- Harmonisches Mittel aus precision und recall
- Liegt zwischen 0 (precision *oder* recall sind 0) und 1 (sowohl precision als auch recall sind perfekt)
- In scikit-learn: `from sklearn.metrics import f1_score.`

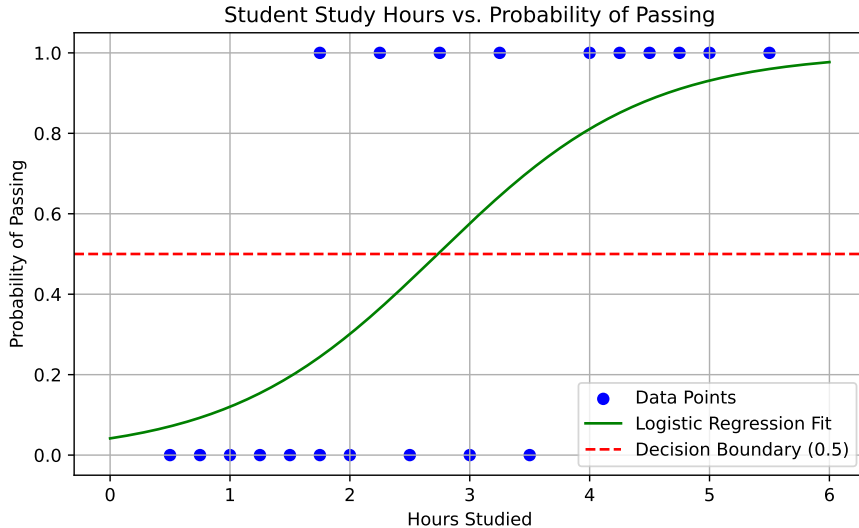
# Beispiel für binäre Klassifikation



# Linear Regression



# Logistische Regression



- Bei einer binären Klassifikationsaufgabe liefert die logistische Regression die *Wahrscheinlichkeit*, dass die Antwort Ja lautet.
- Eindimensionaler Fall: Nur ein Feature  $x$ . Parameter  $\theta_0, \theta_1$ . Modell:

$$\hat{p} = h_{\theta}(x) = \sigma(\theta_0 + \theta_1 x)$$

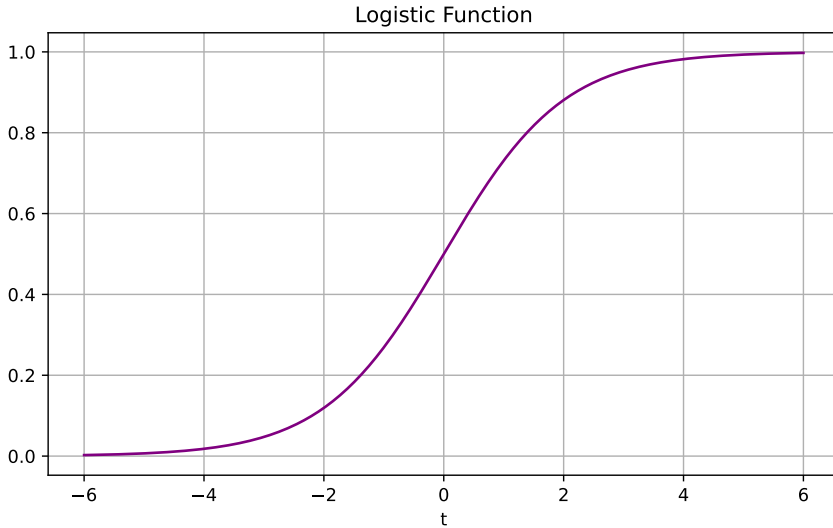
Dabei ist

$$\sigma(t) = \frac{1}{1 + e^{-t}}$$

die *logistische Funktion*.



# Logistische Funktion



- Mehrdimensionaler Fall: Features  $x_1, \dots, x_n$  plus Dummy-Feature  $x_0 = 1$
- Parameter  $\theta_0, \theta_1, \dots, \theta_n$
- Mit Feature-Vektor  $x$  und Parameter-Vektor  $\theta$  haben wir

$$\hat{p} = h_{\theta}(x) = \sigma(\theta^T x)$$

Dabei ist

$$\sigma(t) = \frac{1}{1 + e^{-t}}$$

wieder die logistische Funktion.

