

Zusammenfassung

*Maschinelles Lernen*

*WS 19/20*

December 2, 2019

# Grundlagen

## 1.1 Lineare Algebra

### 1.1.1 Skalarprodukt

- Vektoren  $x, y \in \mathbb{R}^n$ :  $x \circ y = \sum_{i=1}^n x_i \cdot y_i = x^T y$
- $\begin{bmatrix} 1 \\ 2 \end{bmatrix} \circ \begin{bmatrix} 3 \\ 4 \end{bmatrix} = 1 \cdot 3 + 2 \cdot 4 = 11$

### 1.1.2 Vektornorm

$f: \mathbb{R}^n \rightarrow \mathbb{R}$  mit

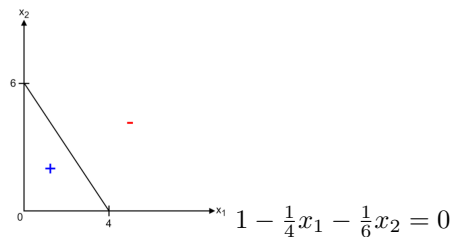
- $f(x) = 0 \Rightarrow x = 0$
  - $f(x + y) \leq f(x) + f(y)$  (Dreiecksungleichung)
  - $f(\alpha x) = |\alpha| f(x)$
- $L_1$ -Norm:  $\|x\|_1 = \sum_i |x_i|$
  - $L_2$ -Norm:  $\|x\|_2 = \sqrt{\sum_i x_i^2}$  (euklidische Norm)

### 1.1.3 Matrizen

- $m$  Zeilen und  $n$  Spalten  $A = \begin{bmatrix} A_{11} & \dots & A_{1n} \\ A_{m1} & \dots & A_{mn} \end{bmatrix}$ ,  $\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}^T = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$
- $\begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix} \cdot \begin{bmatrix} g & h \\ i & j \\ k & l \end{bmatrix} = \begin{bmatrix} ag + bi + ck & ah + bj + cl \\ dg + ei + fk & dh + ej + fl \end{bmatrix}$ ,  $I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$
- $A^{-1}A = I$  (Matrizen mit linear abhängigen Zeilen oder Spalten (niedriger Rang) sind nicht invertierbar)

### 1.1.4 Hyperebene

- $x \in \mathbb{R}^d$  erfüllen Gleichung  $w_0 + w_1 x_1 + w_2 x_2 + \dots + w_d x_d = 0$  ( $w_0 + w^T x = 0$ )
- $d = 1$ : Skalar ( $w_0 + w_1 x_1$ ),  $d = 2$ : Gerade ( $w_0 + w_1 x_1 + w_2 x_2$ ),  $d = 3$ : Ebene
- Für einen Punkt  $x$  entscheidet das Vorzeichen  $\text{sgn}(w_0 + w^T x) \in \{-1, 0, 1\}$  auf welcher Seite der Hyperebene er liegt (bzw. ob er auf ihr liegt)



## 1.2 Statistik

- Durchschnittswert: (Summe über alle Zeilen) / (Anzahl an Zeilen)
- Standardabweichung: Wurzel von Varianz
- 25%-Quantile: 25% aller Werte sind kleiner als dieser Wert
- 50%-Quantile: 50% aller Werte sind kleiner als dieser Wert (= *Median*)
- 75%-Quantile: 75% aller Werte sind kleiner als dieser Wert

## 1.3 Analysis

### 1.3.1 Kettenregel

- Wenn  $z$  von  $y$  und  $y$  von  $x$  abhängt, dann gilt:  $\frac{dz}{dx} = \frac{dz}{dy} \frac{dy}{dx}$
- $f(x) = g(h(x)) = \frac{1}{2} \cdot (x_1 - x_2)^2 \rightarrow g(x) = \frac{1}{2}x^2$  und  $h(x) = x_1 - x_2$
- $\frac{df}{dx_2} = \frac{dg}{dh} \frac{dh}{dx_2} = h(x)(-1) = -(x_1 - x_2) = x_2 - x_1$

### 1.3.2 Partielle Ableitung

$$f(x) = 2x_1^3 - 5x_2^2 + 3, \frac{df}{dx_1} = 6x_1^2, \frac{df}{dx_2} = -10x_2$$

### 1.3.3 Gradient

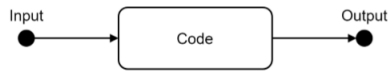
$$\nabla f = \begin{bmatrix} \frac{df}{dx_1} \\ \vdots \\ \frac{df}{dx_n} \end{bmatrix}, f(x) = 2x_1^3 - 5x_2^2 + 3, \nabla f = \begin{bmatrix} 6x_1^2 \\ -10x_2 \end{bmatrix}$$

## 1.4 Was ist maschinelles Lernen

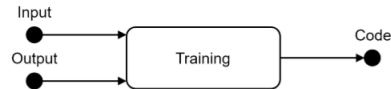
### 1.4.1 Paradigmenwechsel

Es ist schwierig, den entsprechenden Programmcode manuell zu schreiben, daher wird ein anderes Paradigma verwendet:

Traditionelle Programmierung:



Maschinelles Lernen:



Drei verschiedene Lernmethoden

- Überwachtes Lernen (*Supervised Learning*)
- Unüberwachtes Lernen (*Unsupervised Learning*)
- Bestärkendes Lernen (*Reinforcement Learning*)

## 1.5 Überwachtes Lernen

- Ziel: finden einer Funktion  $f : X \rightarrow Y$  wobei  $X$  auch *Features* / *Prädiktoren* und  $Y$  auch *Responses* genannt werden
- $X = \mathbb{R}^d$  ( $d$ -dimensionaler Vektorraum) mit  $d \in \mathbb{N}$
- Eine perfekte Abbildung ist nicht möglich, es treten *reduzierbare* Fehler (z.B. durch eine bessere Funktion  $f$ ) und *nicht reduzierbare* Fehler (z.B. Messfehler in Eingabedaten) auf

- *Vorhersage*:  $y = f(x)$  optimieren wobei  $f$  auch *Blackbox* sein kann
- *Inferenz*: Interpretierbarkeit von  $f$  steht im Vordergrund (Welche Prädiktoren sind für welche Response verantwortlich)
- *Parametrische* Methoden: Annahme einer parametrisierten Struktur von  $f$  dessen Parameter mit Hilfe von Daten bestimmt werden
- *Nicht-parametrische* Methoden: Keine Annahme einer Struktur von  $f$  sondern möglichst direkte Definition mit Hilfe von Daten

- Menge  $X$  und  $Y$  bekannt, genaue Abbildung  $f$  kann aber nur anhand von Beispielen  $D = \{(x^i, y^i) | x^i \in X, y^i \in Y, 1 \leq i \leq n\}$  (*Trainingsdatensatz* bzw. *gelabelte* Daten) erahnt werden

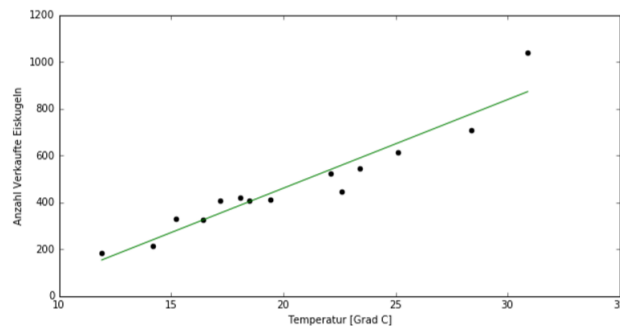
### 1.5.1 Beispiel Klassifikation

- Wenn  $Y$  diskrete Menge  $\{C_1, \dots, C_k\}$  für  $k \in \mathbb{N}$  dann handelt es sich um ein *Klassifikationsproblem*,  $C_1, \dots, C_k$  sind dann *Klassen* / *Kategorien*

- $|Y| = 2$  (*Binäre* Klassifikation) mit  $f : \mathbb{R} \rightarrow \{\text{angenehm, unangenehm}\}$  (Temperaturklassifikation)
- $|Y| = 5$  (*Mehrklassen*-Klassifikation) mit  $f : \mathbb{R} \rightarrow \{\text{frostig, kalt, angenehm, warm, heiß}\}$

### 1.5.2 Beispiel Regression

- Wenn  $Y$  kontinuierliche Menge, d.h.  $Y \subseteq \mathbb{R}$ , dann handelt es sich um ein *Regressionsproblem*
- Interesse an *quantitativen* Aussagen



- Ausgabemenge  $Y$  kann auch mehrdimensional sein (z.B.  $\{\text{gut, schlecht}\} \times \{\text{günstig, normal, teuer}\}$ )

## 1.6 Unüberwachtes Lernen

- Mehrwert erhalten ohne Zuhilfenahme von gelabelten Daten
- Man geht von Menge an Daten  $D = \{x^i | x^i \in X, 1 \leq i \leq n\}$  aus und versucht mehr über Beschaffenheit von  $X$  herauszufinden
- z.B. *Verteilung* von  $X$  bei Sprachmodellen, *Dimensionsreduktion* zur Verbesserung von überwachten Lernverfahren

## 1.7 Datenvisualisierung

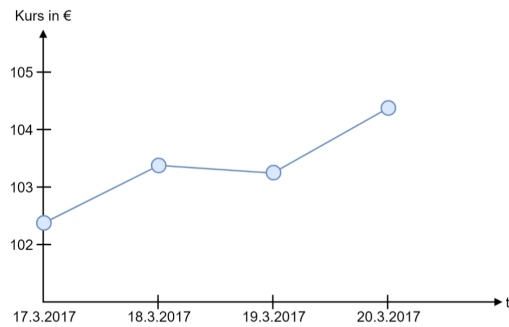


Abbildung 6: Beispiel eines Liniendiagramms.

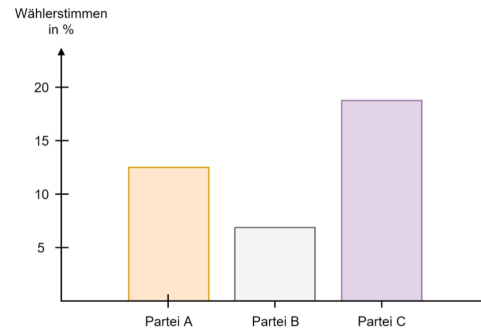


Abbildung 7: Beispiel eines Balkendiagramms.

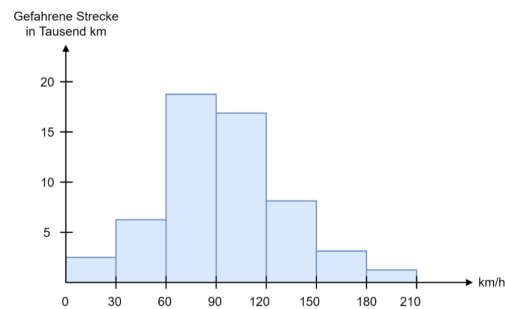


Abbildung 8: Beispiel eines Histogramms – eines speziellen Balkendiagramms.

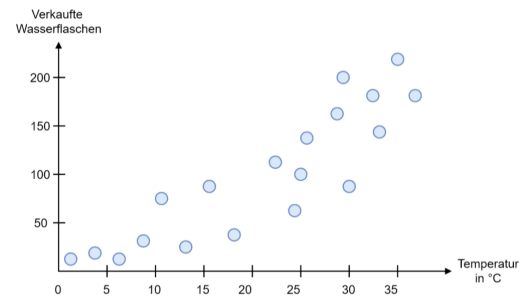
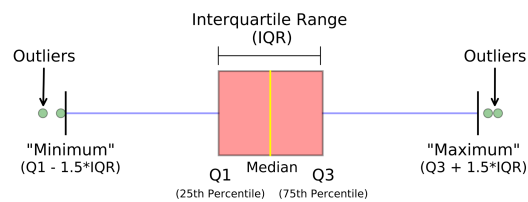


Abbildung 9: Beispiel eines Streudiagramms.

### 1.7.1 Boxplot



- Zwischen dem linken waagerechten Strich (Minimum) und dem rechten waagerechten Strich (Maximum) liegen 99.3% aller Daten
- Die *Outliers* an den beiden Enden sind die letzten 0.7%
- Der Abstandsfaktor (hier 1.5) ist frei wählbar
- Sollte der Punkt  $Q1 - 1.5 \cdot IQR$  bzw.  $Q3 + 1.5 \cdot IQR$  nicht existieren wird der Strich auf den nächst-näheren Punkt gesetzt

## 1.8 Datenvorverarbeitung

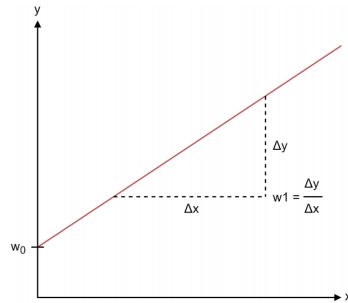
Bevor ein Modell erstellt und trainiert werden kann, müssen Daten durch

- *Auswahl*: Nur für den Anwendungsfall relevante Daten verwenden
- *Aufbereitung*
  - Dateiformat (Tabellen, BigData)
  - Bereinigung von unvollständigen oder ungültigen Daten
  - Repräsentative Auswahl bei langer Laufzeit / großem Speicheraufwand
- *Transformation*
  - Features in geeigneten Wertebereich bringen ( $[0, 1]$ )
  - Zerlegen in sinnvolle Features
  - Aggregation mehrerer Features

# Lineare Regression

## 2.1 Lineare Regression im Eindimensionalen

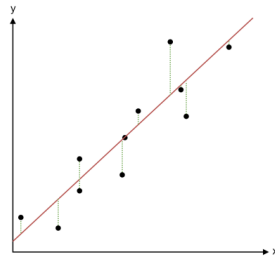
- $f: \mathbb{R} \rightarrow \mathbb{R}$  mit  $f_w(x) = w_1x + w_0$
- $w = (w_0, w_1)^T \in \mathbb{R}^2$  sind die *Parameter* des Modells



- Wie mit Daten  $D = \{(x^i, y^i) \in \mathbb{R}^2 | 1 \leq i \leq n\}$  die *besten* Parameter von  $f$  bestimmen?

### 2.1.1 Lösungsverfahren

- Quadratischen Fehler (*Residual Sum of Squares*) mit  $RSS(w) = \sum_{i=1}^n (y^i - f_w(x^i))^2$  bestimmen



- Zur besseren Vergleichbarkeit verwendet man oft die normalisierte Variante *Mean Squared Error*:  $MSE(w) = \frac{1}{n} \cdot RSS(w)$  ( $n$  = Anzahl Trainingsdaten)
- Die beste Funktion durch Minimierung des Fehlers finden  $\Rightarrow w^* = \arg \min E(w) = \arg \min \frac{1}{2} \cdot \sum_{i=1}^n (y^i - f_w(x^i))^2$

- Ableitung von  $E(w)$  gleich Null setzen und Gleichungssystem lösen

$$\bullet \nabla E(w) = \begin{bmatrix} \frac{dE(w)}{dw_0} \\ \frac{dE(w)}{dw_1} \end{bmatrix} = 0$$

$$\frac{dE(w)}{dw_0} = - \sum_{i=1}^n y^i + w_1 \cdot \sum_{i=1}^n x^i + n \cdot w_0$$

$$\frac{dE(w)}{dw_1} = - \sum_{i=1}^n x^i y^i + w_1 \cdot \sum_{i=1}^n x^i x^i + w_0 \cdot \sum_{i=1}^n x^i$$

- Gleichungssystem mit zwei Gleichungen und zwei Unbekannten lösbar, aber numerisch ungenau bei großen Matrizen



### 2.1.2 Gradientenabstiegsverfahren

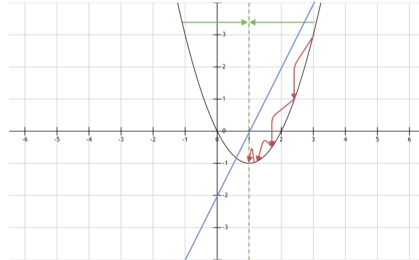


Abbildung 5: Gradientenabstiegsverfahren auf  $f(x) = x(x - 2)$

- Iterativ einem Bruchteil der negativen Ableitung:  $-\eta f'(x) = \eta \cdot (2 - 2x)$  folgen
- *Lernrate*  $\eta$  hat direkten Einfluss auf Konvergenz (zu klein  $\Rightarrow$  viele Schritte, zu groß  $\Rightarrow$  Oszillation)

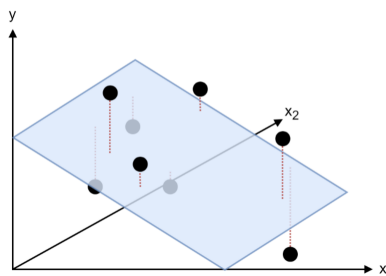
```

w0 = 0, w1 = 0
for (x, y) in D
    dw0 += -y + w1*x + w0
    dw1 += -xy + w1*x*x + w0*x
end for
w0 += -eta*dw0
w1 += -eta*dw1

```

## 2.2 Mehrdimensionale Lineare Regression

- $X = \mathbb{R}^d$  und  $f : \mathbb{R}^d \rightarrow \mathbb{R}$  sowie  $f_w(x) = \sum_{i=1}^d w_i x_i + w_0$
- mit Parametern  $w = (w_0, w_1, \dots, w_d)^T \in \mathbb{R}^{d+1}$  - Kompaktere Schreibweise mit  $x_0 = 1$ :  $f_w(x) = \sum_{i=1}^d w_i x_i + w_0 = w^T x$



- Im Mehrdimensionalen wird eine Hyperebene, im dreidimensionalen eine Ebene, im Raum so positioniert, dass der Abstand zu den Datenpunkten minimiert wird

- Angepasste Fehlermetrik  $E(w) = \frac{1}{2} \cdot \sum_{i=1}^n (y^i - f(x^i))^2$  mit  $\nabla E(w) = \begin{bmatrix} \frac{dE(w)}{dw_0} \\ \frac{dE(w)}{dw_1} \\ \dots \\ \frac{dE(w)}{dw_d} \end{bmatrix}$

```
dw = 0
for (x, y) in D
    dw += -(y - f(x) * gradF(x))
end for
w += -eta * dw
```

wobei  $\text{gradF}(x) = \nabla f(x) = \begin{bmatrix} 1 \\ x_1 \\ \dots \\ x_d \end{bmatrix}$

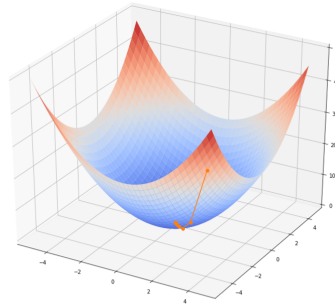


Abbildung 9: Gradientenabstiegsverfahren im mehrdimensionalen Raum bei der Funktion  $f(\mathbf{x}) = \mathbf{x}_1^2 + \mathbf{x}_2^2$ .

## 2.3 Genauigkeit

- Wie gut ist das durch das Gradientenabstiegsverfahren gefundene Modell?  
 $\Rightarrow$  Quadratischer Fehler  $RSS$  oder mittlerer quadratischer Fehler  $MSE$
- Letzterer ist unabhängig von der Anzahl an Trainingsdaten allerdings gibt es keine allgemein gültige Skala da diese vom Wertebereich der y-Werte abhängt

### 2.3.1 $R^2$ Statistik

- Definiert über den quadratischen Gesamtfehler  $TSS = \sum_{i=1}^n (y^i - \bar{y})^2$
- $\bar{y} = \frac{1}{n} \cdot \sum_{i=1}^n y^i \Rightarrow R^2(w) = \frac{TSS - RSS(w)}{TSS} = 1 - \frac{RSS(w)}{TSS}$
- $TSS$  misst die komplette Varianz in den Ausgabedaten  $y^i$
- $TSS - RSS(w)$  misst die durch das Modell mit Parametern  $w$  erklärte Varianz
- $R^2$  misst die komplette Varianz des Modells und ist  $\in [0, 1]$ 
  - $R^2$  nahe 1 zeugt von einem passenden Model das die Daten gut erklärt (viele Datenpunkte liegen auf der Geraden bzw. Hyperebene)

- $R^2$  nahe 0 bedeutet, dass das Modell die Daten schlecht erklärt (umso weiter entfernt die Datenpunkte von der Hyperebene sind umso näher ist  $R^2$  bei 0)
- $R^2$  ist unabhängig von Anzahl an Trainingsdaten *UND* dem Wertebereich
- Allgemeine Aussage ab welchem  $R^2$ -Wert das Modell *gut* ist, ist nicht möglich. Hängt vom Anwendungsfall (Medizin / Physik) ab

## 2.4 Interpretierbarkeit

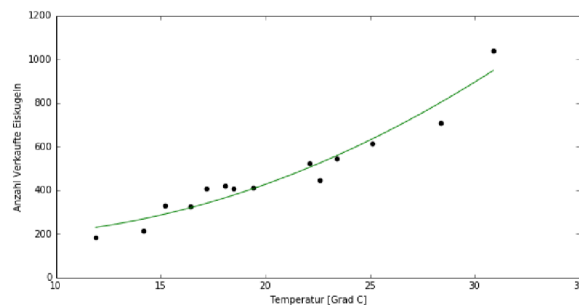
- Die Parameter  $w$  von Linearen Regressionsmodellen sind interpretierbar:
  - $w_i > 0$ : positiver Zusammenhang, steigt  $x_i$  um  $m$  so steigt  $y$  um  $m \cdot |w_i|$
  - $w_i \text{ nahe } 0$ : kein linearer Zusammenhang zwischen  $x_i$  und  $y$
  - $w_i < 0$  negativer Zusammenhang, steigt  $x_i$  um  $m$  so sinkt  $y$  um  $m \cdot |w_i|$

## 2.5 Nichtlineare Zusammenhänge

- Mit der mehrdimensionalen linearen Regressions lassen sich auch *nichtlineare* Zusammenhänge lernen
- Mit Funktion  $\Phi : \mathbb{R} \rightarrow \mathbb{R}^d$  wird ein *Basiswechsel* vollzogen
  - Die Konkatenation von  $\Phi : \mathbb{R} \rightarrow \mathbb{R}^d$ ,  $\Phi(x) = (x, x^2)^T$  und  $f : \mathbb{R}^2 \rightarrow \mathbb{R}$ ,  $f(x) = w_2 x_w + w_1 x_1 + w_0$  durch  $f \circ \Phi$  erlaubt Darstellung der quadratischen Funktion  $(f \circ \Phi)(x) = f(\Phi(x)) = w_2 x^2 + w_1 x + w_0$
  - $\Phi : \mathbb{R}^2 \rightarrow \mathbb{R}^5$ ,  $\Phi(x) = (x_2, x_1, x_1 x_2, x_2^2, x_1^2)^T$  und  $f : \mathbb{R}^5 \rightarrow \mathbb{R}$ ,  $f(x) = \sum_{i=1}^5 w_i x_i + w_0$  ergibt  $(f \circ \Phi)(x) = f(\Phi(x)) = w_5 x_1^2 + w_4 x_2^2 + w_3 x_1 x_2 + w_2 x_1 + w_1 x_2 + w_0$

### 2.5.1 Beispiel

- Annahme eines quadratischen Zusammenhangs  $f(x) = w_2 \cdot x^2 + w_1 \cdot x + w_0$



## 2.5.2 Richtiger Grad

- Mit der mehrdimensionalen Regression, dem Basiswechsel und Gradientenabstiegsverfahren ist es möglich, ein Polynom  $n$ -ten Grades an  $n$  Datenpunkte zu fitten

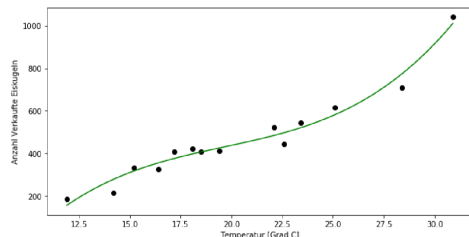


Abbildung 12: Lineare Regression eines Polynoms 3-ten Grades an die Eisverkaufdaten durch Basiserweiterung. Gewichte  $\mathbf{w} \approx (-1853, 307, -14.6, 0.247)^T$

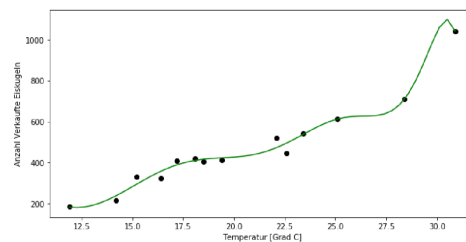


Abbildung 13: Lineare Regression eines Polynoms 12-ten Grades an die Eisverkaufdaten durch Basiserweiterung. Gewichte  $\mathbf{w} = (0, -0.0000457, -0.00000496, -0.0000570, -0.000489, -0.00297, -0.00977, 0.00256, -0.000271, 0.0000152, -0.000000471, 0.00000000777, -0.000000000530)^T$

- Mit höherer Modellkomplexität (Grad und Koeffizienten des Polynoms) kommt es zu

- Numerischen Problemen
- *Overfitting*: Das Modell passt sich zu sehr an die Daten an und ist nicht mehr in der Lage zu generalisieren → Schlechte Leistung in der Praxis

## 2.6 Trainings- und Testdaten

- Datensatz  $D$  wird in zwei disjunkte Teile  $T$  und  $V$  aufgeteilt
- Trainingsdatensatz  $T$  wird für das Lernen verwendet
- Testdatensatz  $V$  enthält ungesehene Daten zur Validierung der Praxistauglichkeit
  - Ein hoher Fehler auf  $T$  lässt auf Unteranpassung schließen (zu geringe Modellkomplexität, zu wenig Daten)
  - Ein geringer Fehler auf  $T$  aber hoher Fehler auf  $V$  bedeutet Überanpassung → Komplexität verringern

## 2.7 Optimierung von Hyperparametern

- Lineare Regression auf Polynomen mit Gradientenabstiegsverfahren besitzt
    - Lernrate  $\eta$ : Einfluss auf Modellkomplexität
    - Anzahl Lernschritte: Je geringer desto unwahrscheinlicher ist Überanpassung, allerdings Unteranpassung wiederum möglich
    - Polynomgrad Zu Hoch → Überanpassung, zu niedrig → Unteranpassung
- als Hyperparameter

### 2.7.1 Rastersuche

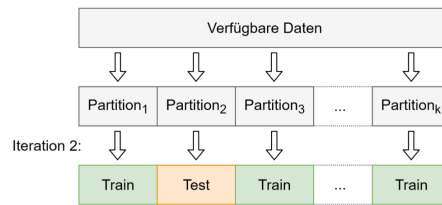
- Durchsuchen des Hyperparameterraums entweder
  - Entlang eines gleichmäßigen Rasters mit linearer oder logarithmischer Skala
  - Entlang eines zufälligen Rasters mit uniformer oder logarithmischer Skala
- Verfeinern der Suche durch Rekursion

### 2.7.2 Validierungsdaten

- Sollen die Hyperparameter des Modells optimiert werden, werden die verfügbaren Daten  $D$  in *Trainingsdaten*, *Validierungsdaten* und *Testdaten* aufgeteilt.
- Die Hyperparameter werden mit dem Validierungsdatensatz optimiert - Endgültige Performance des Modells wird auf den Testdaten bestimmt

### 2.7.3 Kreuzvalidierung

- Zerteilen des Datensatzes in  $k$  Partitionen, wo wird nun  $k$ -mal trainiert
- Mit jeder Iteration  $i$  wird eine andere Partition  $i$  getestet
- Die Restlichen Partitionen dienen als Trainingsdaten
- Final wird die ausgewählte Leistungsmetrik über  $k$  Iterationen gemittelt



### 2.7.4 Ridge Regression

- Verhindern von Überanpassung durch Bestrafung von  $w$  für exzessive Werte mit angepasster Fehlerfunktion  $E(w) = \frac{1}{2} \cdot \sum_{i=1}^n (y^i - f_w(x^i))^2 + \alpha ||w||^2$
- Hyperparameter  $\alpha \in \mathbb{R} \geq 0$  ist ein weiterer Freiheitsgrad mit dem sich der Polynomgrad stufenlos einstellen lässt

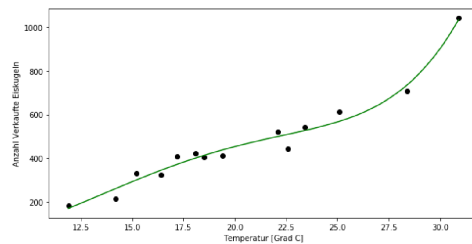


Abbildung 16: Ridge Regression eines Polynoms 5-ten Grades an die Eisverkaufsdaten durch Basiserweiterung,  $\alpha = 10$ .

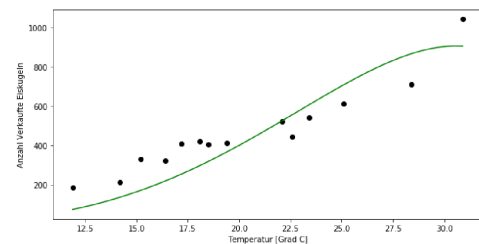


Abbildung 17: Ridge Regression eines Polynoms 5-ten Grades an die Eisverkaufsdaten durch Basiserweiterung,  $\alpha = 10^{10}$ .

- $\alpha = 0$ : klassische Regression
- $\alpha > 0$ : Normaler Wirkungsbereich, mit wachsendem  $\alpha$  werden  $w$  immer weiter eingeschränkt und der effektive Polynomgrad sinkt
- $\lim \alpha \rightarrow \infty$ :  $f(x) = 0$  da Parameter  $\lim w \rightarrow 0$

# Logistische Regression

## 3.1 Klassifikation

### 3.1.1 Lineare Regression

- Lineare Regression als Klassifikator zu verwenden:  $f : \mathbb{R} \rightarrow \mathbb{R}$  mit  $f(x) = w_1x + w_0$

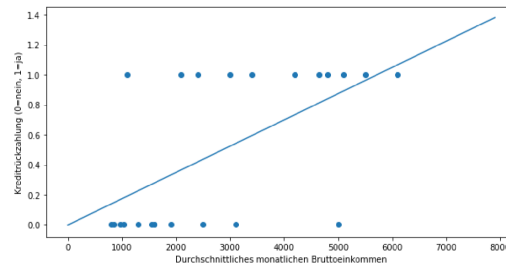


Abbildung 2: Lineares Regressionsmodell zur Bestimmung der Kreditwürdigkeit.

- Im Beispiel:  $x$  = Monatseinkommen,  $f(x)$  = Kunde kreditwürdig ja / nein, ABER:

- Diskrete Ausgabewerte (Klasse 0 / 1) wird nicht eingehalten, Ausgabe nimmt alle Werte in  $[-0.0014, 1.4]$  an
- Interpretation von  $f(x)$  als Wahrscheinlichkeit auch nicht möglich da  $f(0) = -0.0014 < 0$  und  $f(8000) = 1.4 > 1$

### 3.1.2 Logistische Regression

- Idee: Wahrscheinlichkeit der Klassenzugehörigkeit aufgreifen aber Wertebereich von  $f$  mit Hilfe der logistischen Funktion

$$\text{logistic}(x) = \frac{e^x}{1 + e^x}$$

unter Kontrolle bekommen

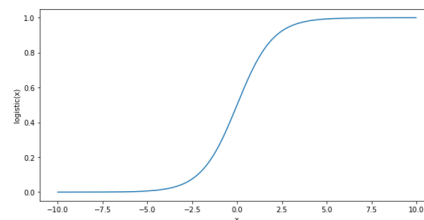


Abbildung 3: Logistische Funktion.

- Kombination der Linearen Regression  $f(x) = w_1x + w_0$  mit der logistischen Funktion

$$p(x) = \text{logistic}(f(x)) = \frac{e^{w_1x + w_0}}{1 + e^{w_1x + w_0}}$$

$$p(x) \in (0, 1) \forall x \in \mathbb{R}$$

-  $p(x)$  ist die Wahrscheinlichkeit, dass  $x$  zur Klasse 1 gehört:

$$p(x) = \Pr(y = 1 | X = x)$$

-  $x$  gehört zur Klasse 0 mit Wahrscheinlichkeit  $1 - p(x)$ :

$$\Pr(y = 0 | X = x) = 1 - \Pr(y = 1 | X = x) = 1 - p(x)$$

## 3.2 Maximum Likelihood

Parameter der Modells werden so bestimmt, dass die Wahrscheinlichkeit, dass das Modell die Daten generiert, maximiert wird

### 3.2.1 Beispiel Münzwurf

- Eine Münze zeigt mit unbekannter Wahrscheinlichkeit  $w \in [0, 1]$  Kopf und mit Wahrscheinlichkeit  $(1 - w)$  Zahl
- Münze wird  $n$ -mal geworfen, die Wahrscheinlichkeit,  $k$ -mal Kopf zu erhalten ist

$$L(w) = w^k (1 - w)^{n-k}$$

-  $L(w)$  wird *Likelihood* genannt, man sucht:  $\arg \max L(w)$   
Maximum finden durch Ableiten

$$\begin{aligned} \bullet \frac{dL(w)}{dw} &= kw^{k-1} \cdot (1 - w)^{n-k} + w^k (n - k)(1 - w)^{n-k-1} \cdot (-1) \\ &= w^{k-1} \cdot (1 - w)^{n-k-1} \cdot [k(1 - w) - (n - k) \cdot w] \end{aligned}$$

anschließend gleich Null setzen

$$\begin{aligned} \bullet \frac{dL(w)}{dw} = 0 &\Leftrightarrow w = 0 \vee w = 1 \vee w = \frac{k}{n} \\ \bullet k(1 - w) - w \cdot (n - k) = 0 &\Leftrightarrow k - wk - wn + wk = 0 \Leftrightarrow k - wn = 0 \Leftrightarrow w = \frac{k}{n} \end{aligned}$$

und Überprüfen von

$$\begin{aligned} \bullet w = 0 : L(0) &= 0^k (1 - 0)^{n-k} = 0 \\ \bullet w = 1 : L(1) &= 1^k (1 - 1)^{n-k} = 0 \\ \bullet w = \frac{k}{n} : L\left(\frac{k}{n}\right) &= \left(\frac{k}{n}\right)^k \cdot \left(1 - \frac{k}{n}\right)^{n-k} > 0 \text{ für } k > 0 \text{ und } k \neq n \end{aligned}$$

Die *Maximum Likelihood* Schätzung ist demnach  $w = \frac{k}{n}$



### 3.2.2 Beispiel Logistische Regression

Die Likelihood ist wie folgt definiert

$$L(w) = \prod_{i=1}^n \begin{cases} p(x^i) & \text{falls } y^i = 1 \\ 1 - p(x^i) & \text{falls } y^i = 0 \end{cases} = \prod_{i|y^i=1} p(x^i) \cdot \prod_{i|y^i=0} (1 - p(x^i))$$

Anstatt das Maximum mit Hilfe der Ableitung zu finden, kann auch das Minimum des negativen Logarithmus gesucht werden

$$-\log(L(w)) = - \sum_{i|y^i=1} \log(p(x^i)) - \sum_{i|y^i=0} \log(1 - p(x^i))$$

und Ableiten

$$\frac{d(-\log(L(w)))}{dw_j} = - \sum_{i|y^i=1} \frac{\frac{dp(x^i)}{dw_j}}{p(x^i)} + \sum_{i|y^i=0} \frac{\frac{dp(x^i)}{dw_j}}{1 - p(x^i)}$$

$$\frac{dp(x)}{dw_j} = \frac{d}{dw_j} \cdot \frac{e^{w_1 x + w_0}}{1 + e^{w_1 x + w_0}} = \frac{e^{w_1 x + w_0}}{(1 + e^{w_1 x + w_0})^2} \begin{cases} 1 & \text{falls } j = 0 \\ x & \text{falls } j = 1 \end{cases}$$

$$1 - p(x) = \frac{1}{1 + e^{w_1 x + w_0}}$$

$$\frac{\frac{dp(x)}{dw_j}}{p(x)} = \frac{1}{p(x)} \cdot \frac{dp(x)}{dw_j} = \frac{1 + e^{w_1 x + w_0}}{e^{w_1 x + w_0}} \cdot \frac{e^{w_1 x + w_0}}{(1 + e^{w_1 x + w_0})^2} = (1 - p(x)) \begin{cases} 1 & \text{falls } j = 0 \\ x & \text{falls } j = 1 \end{cases}$$

$$\frac{\frac{dp(x)}{dw_j}}{1 - p(x)} = \frac{1}{1 - p(x)} \cdot \frac{dp(x)}{dw_j} = (1 + e^{w_1 x + w_0}) \cdot \frac{e^{w_1 x + w_0}}{(1 + e^{w_1 x + w_0})^2} = p(x) \begin{cases} 1 & \text{falls } j = 0 \\ x & \text{falls } j = 1 \end{cases}$$

Zusammenfassend

$$\frac{d(-\log(L(w)))}{dw_0} = - \sum_{i|y^i=1} (1 - p(x^i)) + \sum_{i|y^i=0} p(x^i)$$

$$\frac{d(-\log(L(w)))}{dw_1} = - \sum_{i|y^i=1} (1 - p(x^i)) \cdot x^i + \sum_{i|y^i=0} p(x^i) \cdot x^i$$

Mit dem Gradientenabstiegsverfahren erhält man für das Beispiel der Kreditvergabe

$$w_0 = -1.25238942, w_1 = 0.000542.$$

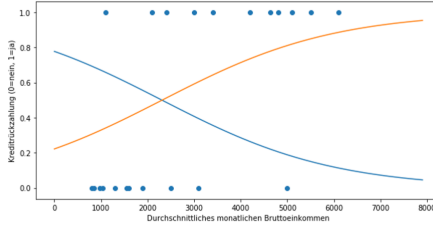


Abbildung 4: Kreditbeispiel: Wahrscheinlichkeit für die Rückzahlung (orange)  $p(x)$  bzw. Nicht-Rückzahlung (blau)  $1 - p(x)$ .

### 3.3 Bayes Klassifikator

- Der Kredit wird also genau dann ausgegeben, wenn die Wahrscheinlichkeit, dass er zurück gezahlt wird größer ist, als dass er es nicht wird
- Der *Bayes Klassifikator* weißt jeder Beobachtung  $x \in X$  die wahrscheinlichste Klasse zu:  $f(x) = \arg \max Pr(y = y * | X = x)$
- Im Beispiel liegt die *Entscheidungsgrenze* bei 2300EUR

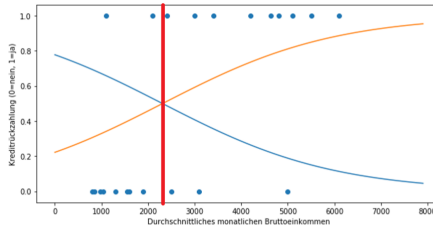


Abbildung 5: Kreditbeispiel: In Rot ist nun die *Entscheidungsgrenze* zwischen den beiden Klassen eingezeichnet.

### 3.4 Mehrdimensionale Logistische Regression

- Für  $x \in \mathbb{R}^d$  wird das Modell beschrieben durch  $p(x) = \frac{e^{w^T x}}{1 + e^{w^T x}}$
- Der Gradient aus der negativen, logarithmierten Likelihood ergibt sich durch

$$\frac{d(-\log(L(w)))}{dw_j} = - \sum_{i|y^i=1} (1 - p(x_j^i)) \cdot x_j^i + \sum_{i|y^i=0} p(x_j^i) \cdot x_j^i$$

### 3.5 Nichtlineare Logistische Regression

- Mit der Basiserweiterung  $\Phi : \mathbb{R}^d \rightarrow \mathbb{R}'$  und der mehrdimensionalen Logistischen Regression können auch nichtlineare Klassifikatoren gelernt werden

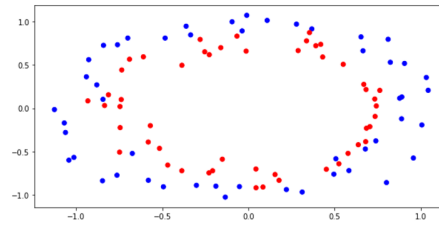


Abbildung 6: Zweidimensionaler Datensatz mit zwei Klassen, die nicht linear trennbar sind.

- Mit der Basiserweiterung  $\Phi(x) = (x_1^2, x_2^2)$  kann ein Logistisches Regressionsmodell gelernt werden, das die beiden Klassen trennt

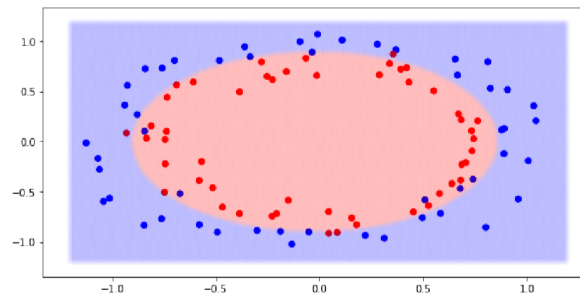


Abbildung 7: Zweidimensionaler Datensatz mit zwei Klassen, die nichtlinear (Kreis) trennbar sind.

## 3.6 Leistungsmetriken

Für die Binäre Klassifikation ist kein  $R^2$ -Wert möglich  $\Rightarrow$  Wahrheitsmatrix:

		prediction	
		+	-
truth	+	true ✓ positive tp	false ✗ negative fn
	-	false ✗ positive fp	true ✓ negative tn

Abbildung 8: Wahrheitsmatrix der binären Klassifikation.

- Genauigkeit (*accuracy*)  $\frac{tp+tn}{tp+tn+fp+fn}$ : Ist der Anteil der korrekt klassifizierten Daten am Gesamtdatensatz. Es wird versucht, die Genauigkeit zu maximieren

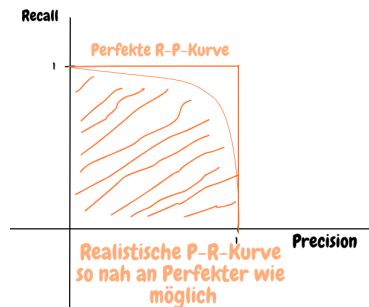
- Fehlerrate  $\frac{fp+fn}{tp+tn+fp+fn} = 1 - \text{Genauigkeit}$ : Gegenteil der Genauigkeit. Wird minimiert.
- Präzision (*precision*)  $\frac{tp}{tp+fp}$ : Anteil der korrekt positiv vorhergesagten Datensätze an der Gesamtheit der als positiv vorhergesagten Datensätze
- Trefferquote (*recall*)  $\frac{tp}{tp+fn}$ : Anteil der korrekt positiv vorhergesagten Datensätze an der Gesamtheit der echt positiven Datensätze

Präzision und Trefferquote werden maximiert, allerdings muss üblicherweise ein Kompromiss getroffen werden.

Es kommt außerdem auf den Anwendungsfall an, welche Leistungsmetrik verwendet werden sollte:

- Medizinische Tests: Positiv bedeutet *krank* (z.B. HIV positiv), dann sollte ein Test einen hohen Recall haben
- Spam-Erkennung: Positiv bedeutet *gewollte* E-Mail, dann sollte ein Spam-Erkennen eine hohe Precision haben

### 3.6.1 Recall-Precision-Kurve



### 3.6.2 Beispiel steigende Kurse an der Börse

Man setzt auf steigende Kurse immer dann wenn das Modell *up* vorhersagt. Es ist wichtig zu wissen, wie oft das Modell relativ gesehen richtig liegt.

- Die Präzision bezieht sich auf den Anteil der korrekt vorhergesagten positiven Renditen. Auf diesen Wert sollte man achten.
- Die Genauigkeit besagt in diesem Fall lediglich, welcher Anteil der Vorhersagen richtig war, unabhängig ob *up* oder *down*
- Die Trefferquote besagt, welcher Anteil der positiven Renditen tatsächlich auch korrekt vorhergesagt wurde

# Perzeptron und Adaline

## 4.1 Perzeptron

### 4.1.1 Biologischer Hintergrund

In einem biologischen neuronalen Netz finden Berechnungen statt, indem elektrische Ladungen zwischen Nervenzellen ausgetauscht werden

- Ein Neuron kann viele Synapsen haben und somit mit hunderten weiteren Neuronen verknüpft sein
- Ein Neuron kann auch viele Dendriten besitzen und somit Input von vielen Neuronen empfangen
- Verbindungen können *verstärkend* oder *hemmend* wirken, je nach der Chemie innerhalb des synaptischen Spalts

	Computer	Biologische neuronale Netze
Einheiten	Prozessoren	Neuronen
Geschwindigkeit	GHz	100 Hz
Signal/Rauschen	$\gg 1$	$\sim 1$
Signalgeschw.	$\sim 10^8 m/s$	$\sim 1 m/s$
Berechnung	sequenziell	parallel
Konfiguration	Programm und Daten	Verbindungen und Chemie (Synapsen)
Programmierung	statisch	adaptiv
Robustheit	gering	hoch
Anwendbarkeit	nur bekannte Daten	chaotische, unvorhergesehene, inkonsistente Daten

Tabelle 1: Vergleich der Berechnungsmodelle adaptiert von [Theory of Neural Information Processing Systems, A. Coolean et al., 2005]

### 4.1.2 Einführung

Ein *Perzeptron* ist ein binärer Klassifikator  $f : \mathbb{R}^d \rightarrow \{0, 1\}$  definiert als

$$f(x) = \alpha \cdot (w \circ x + w_0)$$

Alternativ wird das Perzeptron in der Literatur auch definiert als

$$f(x) = \alpha \cdot (w \circ x)$$

Hier ist  $x = (1, x_1, \dots, x_n)^T$  und  $w = (w_0, w_1, \dots, w_n)^T$ , d.h.  $x_0 = 1$  und  $w_0$  sind in  $x$  und  $w$  enthalten

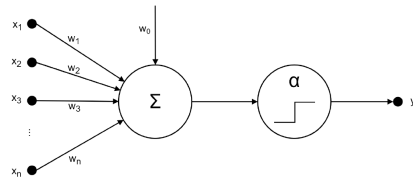


Abbildung 4: Grafische Darstellung eines Perzeptrons.

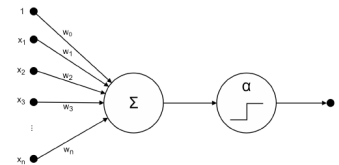


Abbildung 7: Alternative grafische Darstellung eines Perzeptron

Die *Heaviside* Aktivierungsfunktion ist definiert als

$$\alpha(x) = \begin{cases} 1 & \text{falls } x > 0 \\ 0 & \text{andernfalls} \end{cases}$$

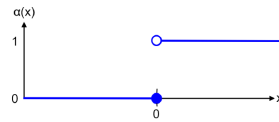


Abbildung 5: Die Heaviside Aktivierungsfunktion, wie sie im Perzeptron verwendet wird.

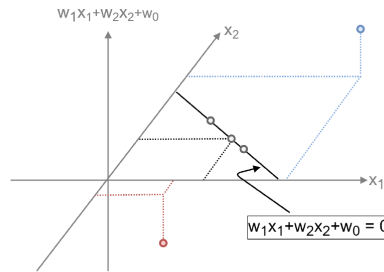
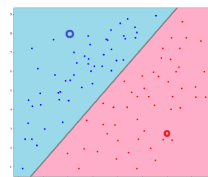


Abbildung 6: Grafische Darstellung der Hyperebene  $\mathbf{w} \circ \mathbf{x} + w_0 = 0$



**Beispiel**

$\mathbf{w} = [17, -37, 30]^T$

►  $\mathbf{x} = [7 \ 3]^T$ :  
 $-37 \cdot 7 + 30 \cdot 3 + 17 = -152 < 0 \Rightarrow$   
**class 0**

►  $\mathbf{x} = [3 \ 8]^T$ :  
 $-37 \cdot 3 + 30 \cdot 8 + 17 = 146 > 0 \Rightarrow$   
**class 1**

Die Klassifizierung lässt sich umdrehen, indem man alle Gewichte mit  $\cdot(-1)$  multipliziert

### 4.1.3 Lernalgorithmus

- Beginne mit einer zufälligen oder festen Wahl für  $w$ , z.B.  $w = 0$
- Bestimme die falsch klassifizierten Datenpunkte
- Versuche iterativ die einzelnen Parameter so zu verändern, dass die Anzahl der falsch klassifizierten Daten sinkt
- Höre auf sobald keine Verbesserung mehr eintritt

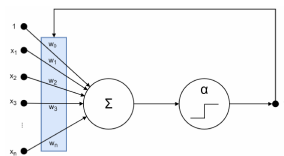


Abbildung 8: Schema des Perzeptron Lernalgorithmus

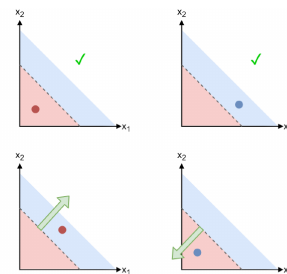


Abbildung 9: Die vier möglichen Fälle, die bei binärer Klassifikation auftreten können.

- $x$  wurde als Klasse 1 eingestuft, ist aber Klasse 0

$$f_w(x) = \alpha \cdot (w \circ x) = 1 \Rightarrow w \circ x > 0$$

$$\begin{aligned}
w' &= w - x \\
w' \circ x &= (w - x) \circ x = w \circ x - x \circ x \\
f_{w'}(x) &= \alpha \cdot (w \circ x - x \circ x) = 0
\end{aligned}$$

- $x$  wurde als Klasse 0 eingestuft, ist aber Klasse 1

$$\begin{aligned}
f_w(x) &= \alpha \cdot (w \circ x) = 0 \Rightarrow w \circ x \leq 0 \\
w' &= w + x \\
w' \circ x &= (w + x) \circ x = w \circ x + x \circ x \\
f_{w'}(x) &= \alpha(w \circ x + x \circ x) = 1
\end{aligned}$$

```

w = 0
while (1.0 / n) * sum(|yi - alpha(w.dot(xi))|) > gamma do
  w' = w
  for i = 1 until n
    oi = alpha(w.dot(xi))
    w' += (yi - oi) * xi
  end for
  w = w'
end while

```

Die Gewichte eines Perzeptrons können auch einfacher berechnet werden, wenn z.B. bekannt ist, dass

- die Punkte  $(3, 0)$  und  $(0, 3)$  auf der Entscheidungsgrenze (bzw. Entscheidungsoberfläche) liegen
- der Ursprung  $(0, 0)$  negativ klassifiziert wird

$$\begin{aligned}
f(x_1, x_2) &= w_0 + w_1 x_1 + w_2 x_2 \\
f(3, 0) = w_0 + 3x_1 &= 0 \Rightarrow w_0 = -3x_1 \\
f(0, 3) = w_0 + 3x_2 &= 0 \Rightarrow w_0 = -3x_2 \\
f(0, 0) < 0 &\Rightarrow x_1 = x_2 = 1 \Rightarrow w_0 = -3 \\
\text{eine Mögliche Lösung ist dann: } f(x_1, x_2) &= -3 + x_1 + x_2
\end{aligned}$$

### Beispiel Lernalgorithmus

- $(x^1, y^1) = ([1, 1], 0)$  bzw.  $([1, 1, 1], 0)$
- $(x^2, y^2) = ([1, 2], 1)$  bzw.  $([1, 1, 2], 0)$
- $(x^3, y^3) = ([2, 2], 0)$  bzw.  $([1, 2, 2], 0)$

$w_0$	$w_1$	$w_2$	$([1, 1], 0)$	$([1, 2], 1)$	$([2, 2], 0)$
0	0	0	$0^\checkmark$	$0^x$	$0^\checkmark$
+1	+1	+2			
1	1	2	$1^x$	$1^\checkmark$	$1^x$
-1	-1	-1			
-1	-2	-2			
-1	-2	-1	$0^\checkmark$	$0^x$	$0^\checkmark$
+1	+1	+2			
0	-1	1	$0^\checkmark$	$1^\checkmark$	$0^\checkmark$

#### 4.1.4 Grenzen des Perzeptron

Probleme wie das Exklusiv-Oder (XOR), welche nicht *linear trennbar* sind, können von einem Perzeptron nicht gelernt werden.

Auch wenn ein Problem linear trennbar ist, erhält man mit dem Perzeptron Lernalgorithmus kein eindeutiges Modell

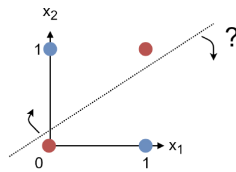


Abbildung 10: Ein Perzeptron kann das Exklusiv-Oder nicht lernen, da es keine Gerade gibt, die die beiden Klassen trennt.

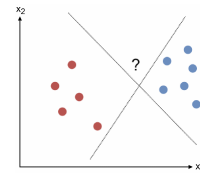


Abbildung 11: In diesem Beispiel gibt es unendlich viele Geraden, welche die Klassen trennen und der Perzeptron Lernalgorithmus gibt nur eine der Lösungen zurück.

## 4.2 Adaline

Das *Adaline* ähnelt im Aufbau dem Perzeptron besitzt jedoch eine andere Aktivierungsfunktion und einen unterschiedlichen Lernalgorithmus genannt *Deltaregel*

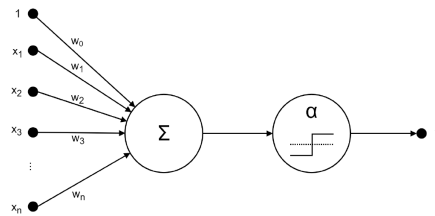


Abbildung 12: Grafische Darstellung eines Adalines.

Das *Adaline* ist ein *Binärklassifikator*  $f : \mathbb{R}^d \rightarrow -1, 0, 1$  definiert als

$$f(x) = \alpha \cdot (w \circ x + w_0)$$

wobei  $\alpha$  die *Signum* Aktivierungsfunktion ist.

Auch hier nehmen wir implizit an, dass  $x_0 = 1$  und  $w$  den Parameter  $w_0$  beinhaltet



### 4.2.1 Einführung

Die *Signum* Aktivierungsfunktion ist definiert als

$$\alpha(x) = \begin{cases} 1 & \text{falls } x > 0 \\ 0 & \text{falls } x = 0 \\ -1 & \text{andernfalls} \end{cases}$$

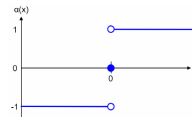


Abbildung 13: Plot der Signum Aktivierungsfunktion

### 4.2.2 Lernalgorithmus

Wie bei der Linearen Regression verwenden wir beim Adaline ein bekanntes Fehlermaß inspiriert durch die RSS/MSE in Verbindung mit dem *Gradientenabstiegsverfahren*, um den negativen Gradienten des Fehlermaßes zum Minimum zu folgen:

$$E(w)^i = \frac{1}{2} \cdot (y^i - f(x^i))^2 = \frac{1}{2} \cdot (y^i - w \circ x^i)^2$$

$$\frac{dE(w)^i}{dw_j} = (y^i - w \circ x^i) \cdot \frac{d}{dw_j}(y^i - w \circ x^i) = -(y^i - w \circ x^i)$$

$$\nabla_w E(w)^i = \begin{bmatrix} \frac{dE(w)^i}{dw_0} \\ \dots \\ \frac{dE(w)^i}{dw_n} \end{bmatrix} = -(y^i - w \circ x^i) \cdot x^i$$

```
w = 0
while (1 / n) * sum(|yi - alpha(w.dot(xi))|) > gamma
  for i = 1 until n
    w += eta * (yi - w.dot(xi)) * xi /* Deltaregel */
  end for
end while
```

- Bei hoher Lernrate  $\eta$  beginnt der Algorithmus zu oszillieren und konvergiert nicht
- Aufgrund seiner linearen Natur kann auch das Adaline die XOR-Funktion nicht direkt lernen
- Je nach Datensatz und Initialisierung ist der Klassifikator eindeutig

# K-Nearest Neighbors

## 5.1 Einführung

Ausgehend vom optimalen Bayes-Klassifikator mit Klassen  $C_1, \dots, C_m$ ,  $x \in X$

$$f(x) = \arg \max Pr(y = C_j | X = x)$$

Bei der KNN-Klassifikation gibt es keine direkte Formel für  $Pr(y = C_j | X = x)$ . KNN kommt ohne Parameter aus und ist somit eine *nicht-parametrische* Methode.

Man merkt sich direkt alle Punkte  $x^i$  zusammen mit ihrer Klasse  $y^i$ , welche im Datensatz

$$T = \{(x^i, y^i) | 1 \leq i \leq n\}$$

enthalten sind. Es gibt somit keine Trainingsphase. Man nimmt stattdessen an, dass ein zu klassifizierender Datenpunkt  $x$  die gleiche Klasse hat, wie die Trainingsdaten in seiner Nähe

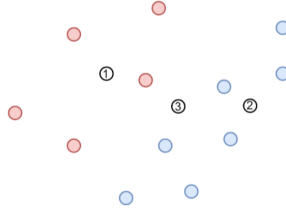


Abbildung 1: KNN-Klassifikation: Punkt 1 gehört wohl wahrscheinlich zur roten Klasse, Punkt 2 zur blauen Klasse und bei Punkt 3 ist die Klasse nicht deutlich erkennbar.

Wobei mit *Nähe* der euklidische Abstand

$$dist(x, x') = ||x - x'||_2 = \sqrt{\sum_{i=1}^d (x_i - x'_i)^2}$$

für  $x, x' \in \mathbb{R}^d$  gemeint ist.

Für eine Menge an Trainingsdaten  $T = \{(x^i, y^i) | 1 \leq i \leq n\}$  und einen Punkt  $x$  definiert man mit  $N^k(x) \subseteq T$  die Menge der  $k$  Trainingspunkte mit der geringsten Distanz  $dist(x, x')$  zum Punkt  $x$ , die sogenannte *k-Nachbarschaft*. Die Anzahl der Elemente in  $N$  ist definiert als  $|N^k(x)| = k$ .

Die Wahrscheinlichkeit der Klassenzugehörigkeit zur Klasse  $C_j$  ist definiert als

$$Pr(y = C_j | X = x) = \frac{|\{(x^i, y^i) \in N^k(x) | y^i = C_j\}|}{k}$$

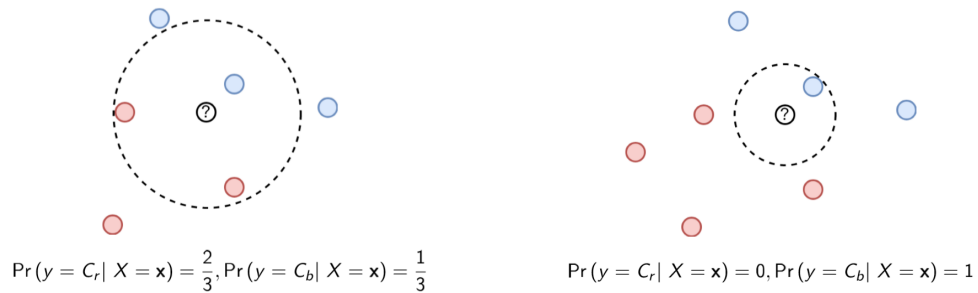


Abbildung 2: Beispiel KNN-Klassifikation mit zwei Klassen und  $k = 3$ . **Abbildung 3:** Beispiel KNN-Klassifikation mit zwei Klassen und  $k = 1$ .

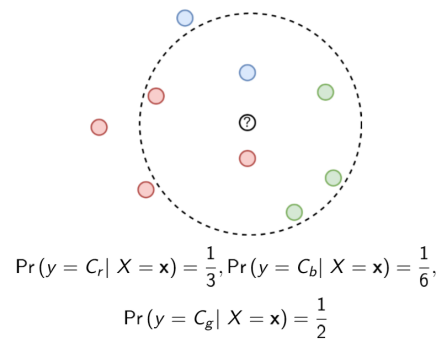


Abbildung 4: Beispiel KNN-Klassifikation mit drei Klassen und  $k = 6$ .

### 5.1.1 Auswirkungen von $K$

KNN-Methoden besitzen einen einzigen *Hyperparameter*  $k \in \mathbb{N}$  welcher maßgeblich die Leistung beeinflusst. Ziel ist daher,  $k$  zu optimieren um

- die *Fehlerrate* auf dem Testdatensatz zu minimieren
- die Genauigkeit zu maximieren

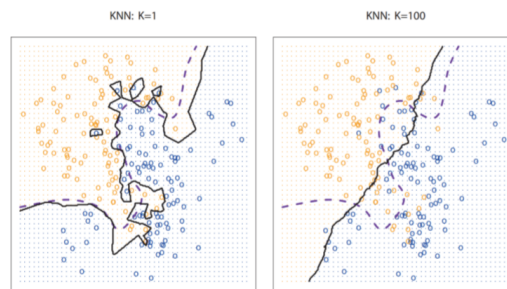


Abbildung 6: Auswirkungen der Wahl des Hyperparameters  $k$  in KNN. Ist  $k$  zu gering, werden die Trainingsdaten zu stark *auswendig gelernt* und es kommt zur **Überanpassung**. Ist  $k$  zu hoch, ist das Model nicht flexibel genug und es kommt zur Unteranpassung. Abbildung entnommen aus [JWHT14].

Allgemein kann man sagen, dass ein kleines  $k$  zu einer rauen Trennkurve zwischen den Klassen führt und größeres  $k$  zu einer glatten Trennkurve. Die Entscheidungsgrenze nähert sich mit steigendem  $k$  immer mehr einer Geraden an.

Wenn die Entscheidungsgrenze des optimalen Bayes-Klassifikators sehr stark nichtlineare wäre führt demnach ein kleines  $k$  zu besseren Ergebnissen. Zu kleines  $k$  birgt jedoch die Gefahr der Überanpassung, zu großes  $k$  wiederum kann Unteranpassung zur Folge haben.

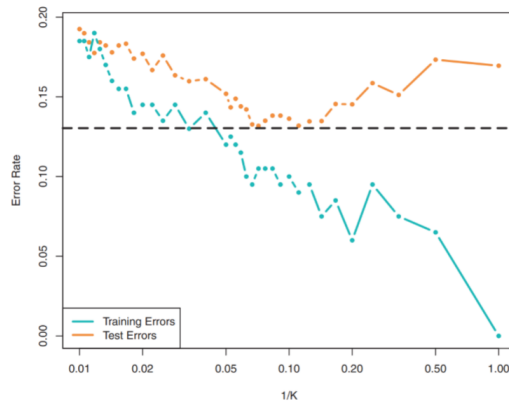


Abbildung 7: Die Fehlerrate auf den Test- und Trainingsdaten in Abhängigkeit von  $k$ . Abbildung entnommen aus [JWHT14].

### 5.1.2 Laufzeit und Beschleunigungsstrukturen

Ein naiver Ansatz ist bei jedem zu klassifizierenden Datenpunkt die Abstände zu allen  $n$  Trainingsdaten zu berechnen, aufsteigend zu sortieren und die  $k$ -ersten Punkte auszuwählen. Ein solcher Algorithmus hat die Laufzeitkomplexität von

$$\mathcal{O}(n^2 \cdot \log(n))$$

Reduzieren der Laufzeit gelingt z.B. mit speziellen Datenstrukturen wie dem *KD-Baum*:

