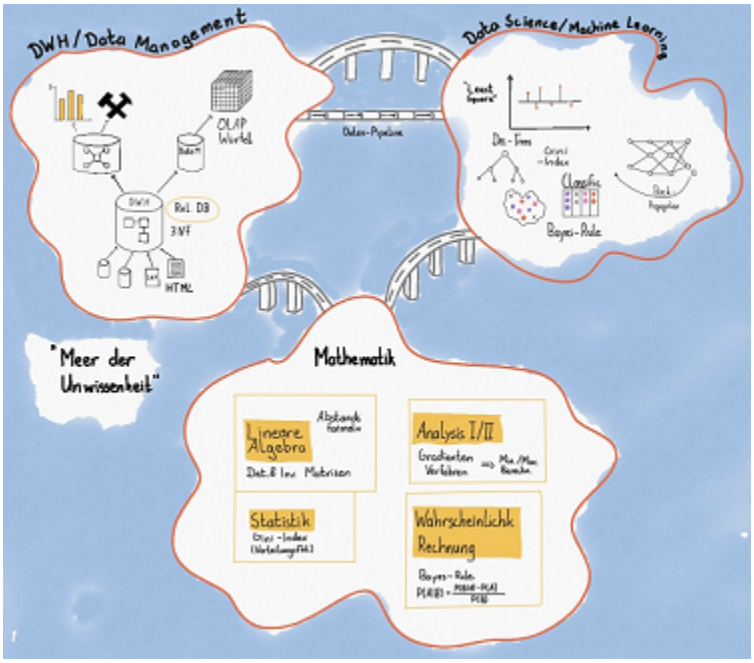
Mathematische Grundlagen des Maschinellen Lernens (ML)

—- Ausarbeitung zu einer Vorlesung ("Buch") —



"3-Insel-Bild"

Dr. Hermann Völlinger, Mathematik und IT Architektur
Stuttgart, Herbst 2023

Diese Version ist nicht final !!! (Beachte: ****** Kommentare ******). Aktuelle Versionen findet man in meinem GitHub:
https://github.com/HVoellinger/Mathematische-Grundlagen_von_ML/blob/main/Math%2BMaschLernen(ML)v0.6.pdf

Dieses Skript wurde generiert mit LaTeX

Inhaltsangabe

Contents

1 Management Zusammenfassung

Ziel der Ausarbeitung ist es, die mathematischen Konzepte des Maschinellen Lernens (ML) zu erkennen.
Leider wird in der gängigen Literatur dieses Thema nicht genug gewürdigt. Dies mag damit zu tun haben das heutzutage mathematische Ausbildung in der Schule und in der Universität etwas den Bezug zur Realität verloren hat und deshalb viele Schüler und Studenten den Nutzen und die Schönheit der Mathematik nicht erkennen können. Es gehört ja manchmal schon "zum guten Ton", wenn technische Hochschulabsolventen fast schon scherzhaft sagen, dass Sie mit Mathematik "nichts am Hut haben". Dies ist leider sehr schade und ist zudem beim Lösen von ML Fragestellungen ein große Hürde.
Ein Ziel der Ausarbeitung ist es deshalb zu zeigen wie nützlich mathematische Ideen bei der Lösung der Fragestellungen des Maschinellen Lernens (aka: https://en.wikipedia.org/wiki/Machine_learning) sind.Die mathematischen Grundlagen des Machine Learning umfassen mehrere Bereiche der Mathematik, die für das Verständnis und die Entwicklung von Machine Learning Algorithmen von Bedeutung sind. Hier sind einige wichtige **mathematische Konzepte und Grundlagen:**

1. Lineare Algebra: Lineare Algebra ist ein grundlegender Bestandteil von Machine Learning. Vektoren und Matrizen werden verwendet, um Daten darzustellen, und Operationen wie Skalierung, Addition, Multiplikation und Transformationen werden angewendet, um Berechnungen durchzuführen. Matrixoperationen wie Matrixmultiplikation und Matrixinversion sind wichtig für viele Algorithmen.
2. Differentialrechnung: Differentialrechnung wird verwendet, um Funktionen zu analysieren und Optimierungsalgorithmen abzuleiten. Berechnung von Gradienten, Ableitungen, partielle Ableitungen und Optimierungsmethoden wie das Verfahren zum Gradientenabstieg sind entscheidend für das Trainieren von Modellen und das Anpassen von Parametern.
3. Wahrscheinlichkeitstheorie und Statistik: Wahrscheinlichkeitstheorie und Statistik spielen eine wichtige Rolle im Machine Learning, insbesondere im Bereich des überwachten und nicht überwachten Lernens. Wahrscheinlichkeitsverteilungen, Schätzungen, Hypothesenbildung und statistische Modelle werden verwendet, um Muster in den Daten zu erkennen, Vorhersagen zu treffen und Unsicherheiten zu quantifizieren.
4. Optimierung: Optimierungsmethoden werden verwendet, um Modelle zu trainieren und die besten Parameterwerte zu finden. Die Optimierungstechniken umfassen lineare Programmierung, konvexe Optimierung und nichtlineare Optimierung.

5. Informationstheorie: Informationstheorie befasst sich mit der Quantifizierung und Übertragung von Informationen. Konzepte wie Entropie, Informationsgewinn und Kompressionsalgorithmen spielen eine Rolle in der Modellierung und Auswahl von Merkmalen sowie in der Reduzierung der Komplexität von Daten.
Diese mathematischen Grundlagen werden verwendet, um Modelle zu definieren, Daten zu analysieren, Funktionen anzupassen, Vorhersagen zu treffen und Modelle zu evaluieren. Sie dienen als Grundlage für das Verständnis der Algorithmen und Methoden des Machine Learning. Es ist wichtig, ein solides Verständnis dieser mathematischen Konzepte zu haben, um Machine Learning effektiv anzuwenden und weiterzuentwickeln.

*** Weitere Kapitel Texte ***

Mathematik ist eine Säule des maschinellen Lernens. Vor etwa 250 Jahren begann man, sie zu formalisieren, aber erst vor etwa 100 Jahren wurde die moderne Mathematik entwickelt. Es handelt sich um ein riesiges Fachgebiet, mit vielen Unterbereichen, wie Lineare Algebra, Statistik, etc. sowie mit Anwendungen in den Ingenieurwissenschaften und der Physik.
Zum Glück müssen wir nicht die ganze Breite und Tiefe der Mathematik kennen, um unser Verständnis und unsere Anwendung des maschinellen Lernens zu verbessern. Die wesentlichen Bereiche der Mathematik die näher betrachten werden, sind:

1. Lineare Algebra für Maschinelles Lernen (ML)

2. Wahrscheinlichkeitsrechnung und Statistische Methoden für ML

3. Optimierungsmethoden für ML

4. Analysis (i.e. "Gradienten-Verfahren") für ML

Die wichtigsten ML Anwendungen und Methoden, die mathematische Verfahren nutzen, werden vorgestellt und in ihren wesentlichen Merkmalen erläutert.
Es werden zahlreiche Hinweise auf vertiefende Anwendungen oder Informationen durch Internet-Links oder weiterführende Literatur gegeben. An vielen Stellen werden konkrete Umsetzungsbeispiele mit Werkzeugen wie *KNIME Analytics Platform* gezeigt.
Die Beziehungen ("Brücken") zwischen Mathematik, Maschinellen Lernen (ML) und

Data Science (DL) (insbesondere Data Mining werden konkret gebaut und anschließend beispielhaft "beschritten" (siehe auch "3-Insel-Bild" von der Titelseite).
Die Leser sollen befähigt werden, die Mathematik hinter den ML Anwendungen zu verstehen um mögliche weitere sinnvolle Ergänzungen und Erweiterungen die sich aus dem mathematischen Erkenntnisse ergeben zu nutzen.
Zu jedem Kapitel gibt es Übungen, die das im Kapitel Gelernte nochmals an konkreten Fragestellungen und Problemen erproben und vertiefen. Musterlösungen bzw. Hinweise zur Lösung der Übungen findet man am Ende des Buches.
Insgesamt beinhaltet die Arbeit zur Zeit 11 Kapitel.

*** Weitere Kapitel Texte ***

2 Wichtige Anwendungen der Mathematik in ML

In diesem Skript/Buch werden Sie unter anderen sechs konkrete Beispiele für Mathematik bei bekannten Verfahren des Maschinellen Lernen (ML) kennenlernen. Pro Verfahren gibt es ein separates Kapitel im Skript
Die einzelne Kapitel werden durch anschauliche Beispiele motiviert und ergänzt. Anhand dieser Beispiele kann mal auch schnell und anschaulich die allgemeinen mathematischen Grundlage der entsprechenden ML Verfahren nachvollziehen.
Zu den einzelnen Kapitel Themen gibt es jeweils mehrere Übungen bzw. Hinweise zu deren Lösung.
Diese sechs ML Verfahren sind:

1. Entscheidungsbäume und GINI Index (Statistik)

2. Lineare Regression und "Best Fit" Verfahren der Analysis (AN)

3. Text-Klassifikation mit Bayes-Verfahren (Bedingte Wahrscheinlichkeiten)

4. Verfahren der "Support Vector Machines" (SVM)

5. Neuronale Netzwerke und Backpropagation via "Absteigende Gradienten"

6. Mathematik und grosse Sprachmodelle "Large Language Models" (LLMs))

Am Ende kommt noch ein Kapitel "Anhang" in den fünf weitere ML-Verfahren der Vollständigkeit halber eher kurz erläutert werden. Diese sind:

1. Empfehlungssysteme "Recommender Systems" (LinAlgebra)

2. Regularisierungen "Regularization" und Lin. Algebra (LA)

3. Hauptkomponenten-Analyse "Principal Component Analysis" (PCA)

4. Singulärwertzerlegung "Singular Value Decomposition" (SVD)

5. Mathematische Verfahren im NLP, i.e. "Latent Semantic Analysis/Indexing (LSI)"?

3 Lernen von Entscheidungsbäumen / "Decision Tree" (DT) Learning

Entscheidungsbaum-Algorithmen sind weit verbreitete Methoden im maschinellen Lernen, die zur **Klassifikation und Regression** verwendet werden.

3.1 Mathematik bei Entscheidungsbäumen

Es gibt verschiedene mathematische Verfahren und Techniken, die bei der Konstruktion und Optimierung von Entscheidungsbäumen eingesetzt werden. Hier sind einige davon:

1. Entropie und Informationsgewinn:
Die Entropie ist ein Maß für die Unordnung oder Unsicherheit in einem Datensatz. Beim Konstruieren eines Entscheidungsbaums wird der Informationsgewinn verwendet, um festzustellen, wie gut eine Funktion (Attribut) den Datensatz in Bezug auf die Zielvariable aufteilt. Der Informationsgewinn wird oft in Form von Entropieänderungen zwischen den ursprünglichen und den aufgeteilten Datensätzen gemessen.

2. Gini-Index:
Der Gini-Index misst die Wahrscheinlichkeit, dass eine zufällig ausgewählte Instanz falsch klassifiziert wird, wenn sie zufällig nach den Klassenverteilungen in einem Teilbaum ausgewählt wird. Ein niedriger Gini-Index deutet auf eine homogene Verteilung der Klassen hin.

3. Reduktion des quadratischen Fehlers (Regression):
Bei Entscheidungsbäumen für Regression werden mathematische Kriterien wie die Reduktion des quadratischen Fehlers verwendet, um die besten Aufteilungen der Daten zu finden, die zu einer geringeren Varianz der Zielvariablen führen.

3. Cost-Complexity-Pruning:
Nach dem Konstruieren eines Entscheidungsbaums können zu viele Verzweigungen zu Overfitting führen. Das Cost-Complexity-Pruning verwendet eine Kostenfunktion, um die Qualität eines Baums in Bezug auf seine Komplexität zu bewerten. Durch Entfernen von Verzweigungen, die nur geringfügig zur Verbesserung der Modellgenauigkeit beitragen, kann die Generalisierungsfähigkeit des Baums verbessert werden.

4. CART (Classification and Regression Trees):
Der CART-Algorithmus verwendet eine rekursive Zweig- und Bindemethode, um einen Baum zu erstellen. Er sucht nach der besten Spaltung eines Datensatzes anhand des Gini-Index (für Klassifikation) oder der Reduzierung des quadratischen Fehlers (für Regression).

5. Random Forests und Boosting:
Diese Ansätze basieren auf Entscheidungsbäumen, werden jedoch durch die Kombination mehrerer Bäume oder das Hinzufügen von Gewichten zu den Instanzen verbessert. Sie nutzen mathematische Methoden, um die Vorhersagegenauigkeit und Robustheit zu erhöhen.

5. Hyperparameter-Optimierung:
Die Wahl der Hyperparameter, wie beispielsweise die maximale Tiefe des Baums oder die Mindestanzahl der Instanzen in einem Blatt, beeinflusst die Leistung des Entscheidungsbaums. Mathematische Verfahren wie Rastersuche oder Bayes'sche Optimierung können verwendet werden, um die besten Hyperparameter für das Modell zu finden.

Zusammenfassung: Diese Verfahren sind alle Teil des Prozesses der Konstruktion, Anpassung und Optimierung von Entscheidungsbäumen im maschinellen Lernen. Die Wahl des geeigneten Verfahrens hängt von der Art des Problems und den Daten ab, mit denen Sie arbeiten.

3.2 DT-Verfahren mit Gini-Index- oder Entropie-Verfahren

****

ab hier bis Ende der Section sind die Folien der Vorlesung ML zu nutzen und diese sind in Latex umzusetzen

****

3.3 Mathematische Grundlagen GINI- und Entropie-Verfahren

3.4 Beispiele für GINI-Verfahren

Obersten Knoten berechnen:

outlook:

	overcast	sunny	rainy	
Y	4	2	3	9
N	0	3	2	5
	4	5	5	

GINI(outlook) = ~0,343

humidity:

	high	normal	
Y	3	6	9
N	4	1	5
	7	7	

GINI(humidity) = 0,367

temperature:

	hot	mild	cool	
Y	2	4	3	9
N	2	2	1	5
	4	6	4	

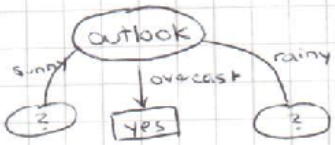
GINI(temperatur) = 0,44

windy:

	FALSE	TRUE	
Y	6	3	9
N	2	3	5
	8	6	

GINI(windy) = 0,429

⇒



Für sunny: temperature | sunny

	hot	mild	cool	
Y	0	1	1	2
N	2	1	0	3
				5

GINI(temp|outlook = sunny) = 0,2

Für windy: ~~temperature | windy~~ windy | sunny

	FALSE	TRUE	
Y	1	1	2

Für windy: windy | rain

	FALSE	TRUE	
Y	3	0	3
N	0	2	2
	3	2	5

$GINI(windy|rain) = \frac{3}{5}\left(1-\frac{3}{3}^2-\frac{0}{3}^2\right)+\frac{2}{5}\left(1-\frac{0}{2}^2-\frac{2}{2}^2\right)=0$

Für humidity | rain

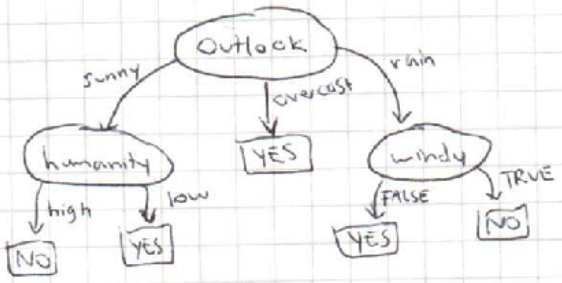
	high	normal	
Y	1	2	3
N	1	1	2
	2	3	5

$GINI(humidity|rain) = \frac{3}{5}\left(1-\frac{1}{2}^2-\frac{1}{2}^2\right)+\frac{3}{5}\left(1-\frac{2}{3}^2-\frac{1}{3}^2\right)>0$

Für temp | rain

	hot	mild	cool	
Y	0	2	1	3
N	0	1	1	2
		3	2	5

$GINI(temp|rain) = \frac{3}{5}\left(1-\frac{2}{3}^2-\frac{1}{3}^2\right)+\frac{2}{5}\left(1-\frac{1}{2}^2-\frac{1}{2}^2\right)>0$



3.4.1 Beispiele für Entropie-Verfahren

3.5 GINI-Index "Production Maintenance"

3.6 Mathematische Grundlagen

****

ab hier bis Ende der Section sind die Folien der Vorlesung ML zu nutzen und diese sind in Latex umzusetzen

****

3.7 Übungen zu Kapitel 3

****

ab hier bis Ende der ÜBUNGEN sind die Folien der Vorlesung ML zu nutzen und diese sind in Latex umzusetzen

****

3.7.1 Übung 3.1

Homework H4.1 - "Calculate ID3 and CART Measures"

Groupwork (2 Persons) - Calculate the measures of decision tree "Playing Tennis Game":

1. ID3 (Iterative Dichotomiser 3) method using ***Entropy Fct. & Information Gain.***
2. CART (Classification) → using ***Gini Index(Classification)*** as metric.

Homework H4.2 - "Define the Decision Tree for UseCase *Predictive Maintenance* (see slide p.77) by calculating the GINI Indexes"

Groupwork (3 Persons): Calculate the Decision Tree for UseCase "Predictive Maintenance" on slide p.77. Do the following steps (one person per step):

1. Calculate the **Frequency Matrices** for the features „Temp.", „Druck" and „Füllst."
2. Define the **Root-node** by calculating the GINI-Index for all values of the three features. Define the optimal **split-value for the root-node** (see slide p.88)
3. **Finalize the decision tree** by calculation the GINI-Index for the remaining values for the features "Temp." and "Füllst."

Figure 1: DT-Homework(1+2)

3.7.2 Übung 3.2

3.7.3 Übung 3.3

3.7.4 Übung 3.4

3.7.5 Übung 3.5

Homework H4.3 (advanced)*-"Create and describe the algorithm to automate the calculation of the Decision Tree for the Use Case "Predictive Maintenance"

Groupwork (2 Persons): Create and describe the **algorithm to automate the calculation** of steps 1. to 3. of homework H4.2. See more detailed description of the steps in the lecture:

1. Calculate the **Frequency Matrices** for the features „Temp.", „Druck" and „Füllst."
2. Def. **Root-node** by calculating GINI-Index of the three features & find the optimal **split-value** for the root-node.
3. **Finalize the decision tree** by calculation the GINI-Index for the remaining values for the features "Temp." and "Füllst."

Homework H4.4* - "Summary of the Article …*prozessintegriertes Qualitätsregelungssystem*..."

Groupwork (2 Persons) – read and create a short summary about a special part of article/dissertation from Hans W. Dörmann Osuna: "Ansatz für ein prozessintegriertes Qualitätsregelungssystem für nicht stabile Prozesse". Link to article: http://d-nb.info/992620961/34

For the two chapters (1 Person each Chapter, 15 Minutes):

- Chapter 7.1 „Aufbau des klassischen Qualitätsregelkreises"
- Chapter 7.2. "Prädiktive dynamische Prüfung"

Figure 2: DT-Homework(3+4)

Homework H4.5* - "Create and describe the algorithm to automate the calculation of the Decision Tree for the Use Case "Playing Tennis" using ID3 method"

Groupwork (2 Persons) - Calculate the measures of decision tree "Playing Tennis Game" by creating a Python Program (i.e. using Jupyter Notebook) with "ID3 (Iterative Dichotomiser 3)" method using Entropy Fct. & Information Gain

Figure 3: DT-Homework5

4 Lineare Regression (LR) in ML

4.1 Allgemeine Einführung in Regressionsmodelle

Ein **Regressionsmodell im Machine Learning** ist ein statistisches Modell, das dazu verwendet wird, die Beziehung zwischen einer abhängigen (oder Ziel-) Variable ("Target") und einer oder mehreren unabhängigen (oder erklärenden) Variablen (Features") zu analysieren und zu beschreiben. Das Hauptziel der Regression besteht darin, Vorhersagen für die abhängige Variable zu treffen, basierend auf den Werten der unabhängigen Variablen.
Die Grundidee hinter einem Regression ist es, eine Funktion zu finden, die die bestmögliche Anpassung an die gegebenen Daten bietet. Je nach Art der Daten und der Beziehung zwischen den Variablen gibt es verschiedene Arten von Regressionsmodellen, darunter:

1. **Lineare Regression**: Hierbei handelt es sich um das einfachste Regressionsmodell, bei dem versucht wird, eine lineare Beziehung zwischen den unabhängigen und abhängigen Variablen zu finden.

2. **Polynominale - oder auch Multidim. Regression**: Diese erweitert die lineare Regression, indem sie Polynome höheren Grades verwendet, um komplexere Beziehungen zwischen den Variablen zu modellieren.

3. **Logistische Regression**: Obwohl der Name "Regression" enthält, wird die logistische Regression hauptsächlich für Klassifikation Probleme verwendet, bei denen die abhängige Variable diskrete Werte annimmt. Sie wird verwendet, um die Wahrscheinlichkeit zu schätzen, dass eine bestimmte Klasse in einem binären oder mehrklassigen Klassifikationsproblem auftritt.

4. **Ridge Regression und Lasso Regression**: Diese Varianten der linearen Regression dienen dazu, mit möglicherweise hochdimensionalen Datensätzen umzugehen und Overfitting zu reduzieren, indem sie Regularisierungstechniken verwenden.

5. **Nichtlineare Regression**: Für komplexere Zusammenhänge zwischen den Variablen werden nichtlineare Regressionsmodelle verwendet, die nichtlineare Funktionen verwenden, um die Daten besser anzupassen.

6. **Zeitreihenregression**: Diese Art der Regression wird verwendet, um Zeitreihendaten zu modellieren, bei denen die abhängige Variable über einen Zeitverlauf hinweg beobachtet wird.

Die Auswahl des geeigneten Regressionsmodells hängt von der Natur der Daten, der Art der Beziehung zwischen den Variablen und den Zielen der Analyse ab.
Die Modellierung und Auswertung von Regressionsmodellen sind grundlegende Techniken im Bereich des maschinellen Lernens und werden in einer Vielzahl von Anwendungen eingesetzt, von der Vorhersage von Aktienkursen bis zur medizinischen Diagnose.

******* Beispiel von Latex Syntax für eine Tabelle *******

Spalte 1	Spalte 2
Inhalt 1	Inhalt 2

Table 1: Eine Tabelle mit Positionsangabe "h".

******* Ende Beispiel von Latex Syntax für eine Tabelle *******

****** Beispiel von Latex Syntax für eine Matrixe *******

Berechnen Sie die Determinante der Matrix

$$A := \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 1 \\ 3 & 4 & 1 & 2 \\ 4 & 1 & 2 & 3 \end{pmatrix}$$

****** Ende Beispiel von Latex Syntax für eine Matrix *******

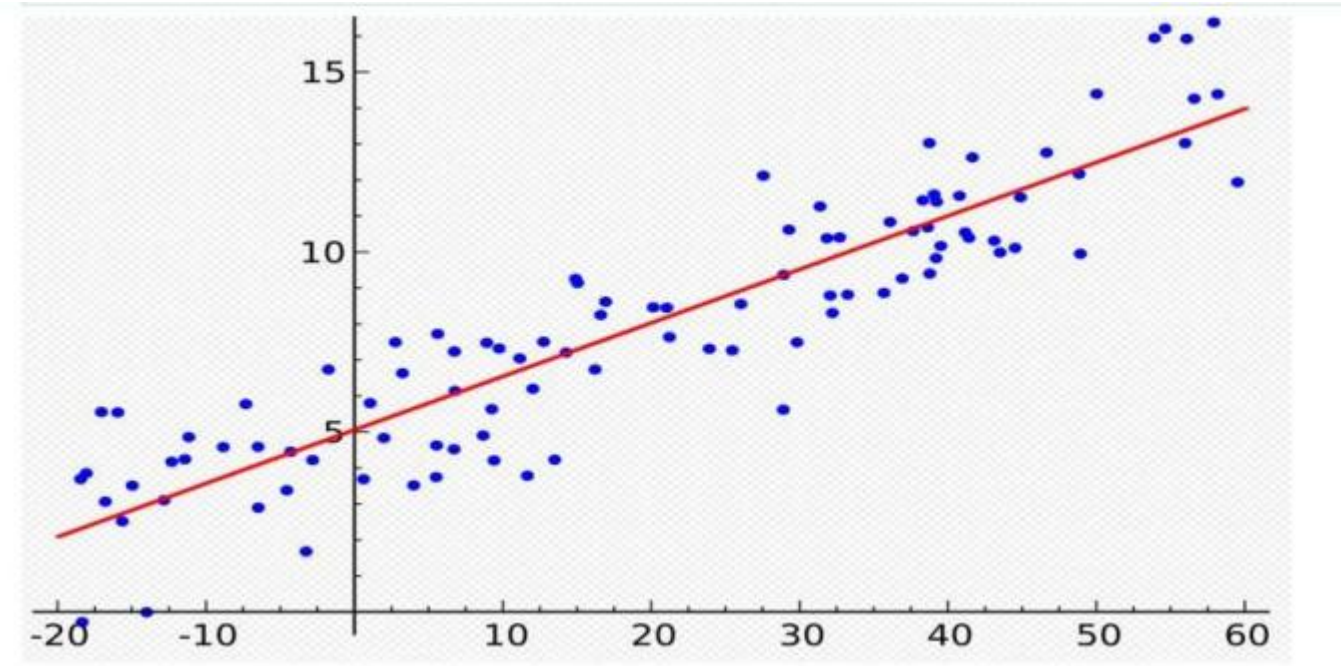
4.2 Motivation und Beispiele der Linearen Regression

In unserem Skript fokussieren wir uns auf die Linearen Regressionen (LR). Das LR Modell geht von einer linearen Beziehung zwischen den Variablen aus, was bedeutet, dass Änderungen in den unabhängigen Variablen ("Features") mit konstanten Veränderungen in der abhängigen Variable (Target") einhergehen.
Die lineare Regression nutzt eine Methode, die als **"Methode der kleinsten Quadrate"**

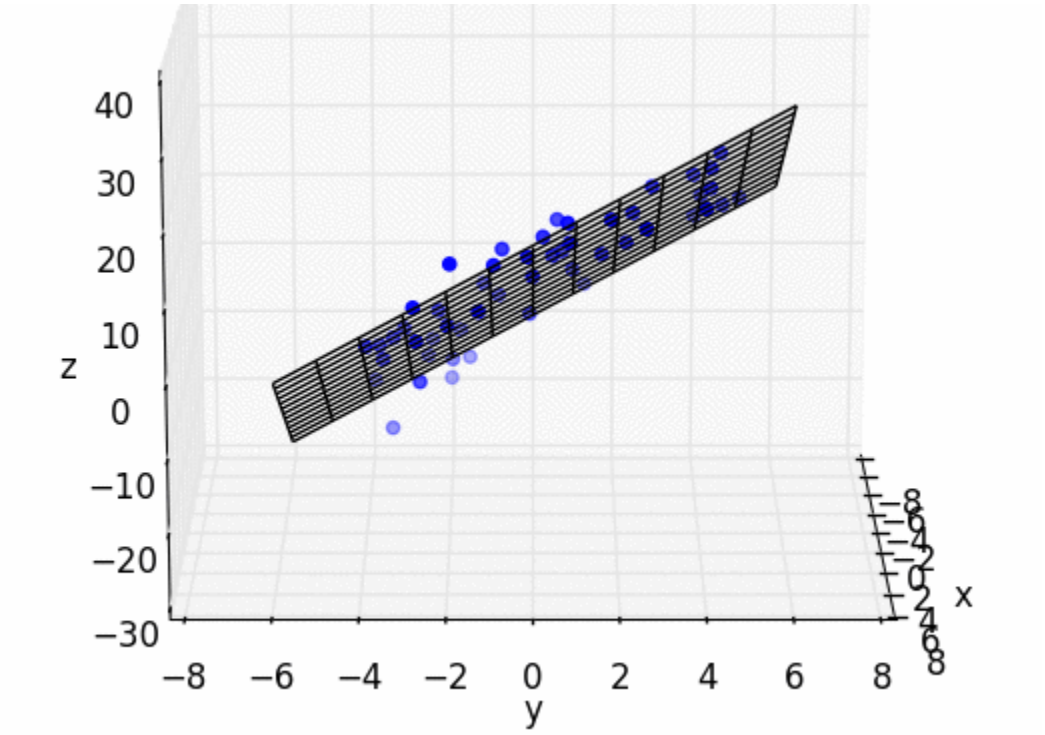
bezeichnet wird, um die besten Schätzwerte für die Koeffizienten zu finden. Diese Methode minimiert die quadratischen Abweichungen zwischen den beobachteten Werten und den vom Modell vorhergesagten Werten.
Sei k die Anzahl der unabhängigen Variablen dann sprechen wir bei k = 1 von einer "Einfachen (simple) Linearen Regression" (sLR) und bei $k \geqslant 2$. von einer "Multidimensionalen (multiplen) Linearen Regression"(mLR).
Eine Trainingsmenge von m Datenpunkten $\{P_j\}$ in $\mathbb{R}^{k+1}$ wird dargestellt als Vektor: $\{(x_{ij}, y_j)\}$, wobei $1 \leq i \leq k$ und $1 \leq j \leq m$.
Die ersten k Komponenten $\{x_{ij}\}$ pro Datenvektor sind die Komponenten der unabhängigen Variablen ("Features") und die letzte Komponente $\{(y_j)\}$ ist die Komponente der abhängigen Variable ("Target" oder auch "Label").
Mit anderen Worten sollen aus den k Eigenschaften $\{x_{ij}\}$ eine Aussage über die Zieleigenschaft $\{y_j\}$ gemacht werden.

4.2.1 Beispiele von LR Lösungen für k=1, k=2

Mathematisch ("Geometrie") erhalten wir für k=1 Gerade in $\mathbb{R}^2$.
Für k = 2 eine Ebene in $\mathbb{R}^3$ und für $k \geqslant 3$ eine Hyperebene in $\mathbb{R}^{k+1}$.
Die ersten zwei Fälle lassen sich visualisieren:
k=1: Das folgende Bild zeigt die "*Regressionsgerade*" für die n blauen Beobachtungspunkte $\{P_1 = (x_{11}, y_1)...P_n = (x_{1n}, y_n)\}$



k=2: Das nächste Bild zeigt die "*Regressionsebene*" für die n blauen Beobachtungspunkte $\{P_1 = (x_{11}, x_{21}, y_1)...P_n = (x_{1n}, x_{2n}, y_n)\}$



.....

********** test *********

Berechnen Sie die Determinante der Matrix

$$A := \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 1 \\ 3 & 4 & 1 & 2 \\ 4 & 1 & 2 & 3 \end{pmatrix}$$

********** end test *********

4.3 Kennzahlen R^2 und $adj.R^2$

Für die Definition der *simple Linear Regression (sLR)* und *multiple Linear Regression (mLR)* brauchen wir einige Kennzahlen für die Datenpunkte(i.e."Beobachtungspunkte")

aus der Trainingsmenge (*observation points*).
Diese Kennzahlen sind *Sum of Squares Total (SST), Sum of Squares Error (SSE)* und *Sum of Squares Regression (SSR).*

4.3.1 Definition der sLR Kennzahl R^2

Die Definition der Kennzahl R^2 benutzt SST und SSE. SSR wird für die eigentliche Definition von R^2 nicht genutzt. Wir brauchen diese Kennzahl aber später im Kapitel:

`Definiere` $f_i = f(x_i)$ `und` $\overline{y} = 1/n \sum_{i=1}^{n}\Big(y_i\Big)$ `(wobei n = Anzahl der Beobachtungen)`

SST, SSE und SSR sind gegeben durch die folgenden Definitionen (D-4.x). Ohne Verlust der Allgemeinheit (o.A.) können wir dabei annehmen das SST größer als Null ist:

$$\texttt{(D-4.1): Sum of Squares Total (SST) :=} \sum_{i=1}^{n}\Big(y_i - \overline{y})\Big)^2$$

$$\texttt{(D-4.2): Sum of Squares Error (SSE) :=} \sum_{i=1}^{n}\Big(y_i - f_i\Big)^2$$

$$\texttt{(D-4.3): Sum of Squares Regression (SSR) :=} \sum_{i=1}^{n}\Big(f_i - \overline{y})\Big)^2$$

Für die Definition von R^2 brauchen wir Sum of Squares Error (SSE) und Sum of Squares Total (SST). Die Definition von R^2 ist gegeben durch die Differenz von 1 und dem Quotienten SSE/SST- Wir nennen R^2 in Deutsch auch "Bestimmtheitsgrad":

$$\texttt{(D-4.4):} \quad R^2 := 1 - \frac{SSE}{SST}$$

****
hier Bild mit SSE; SSR und SST einfügen
****

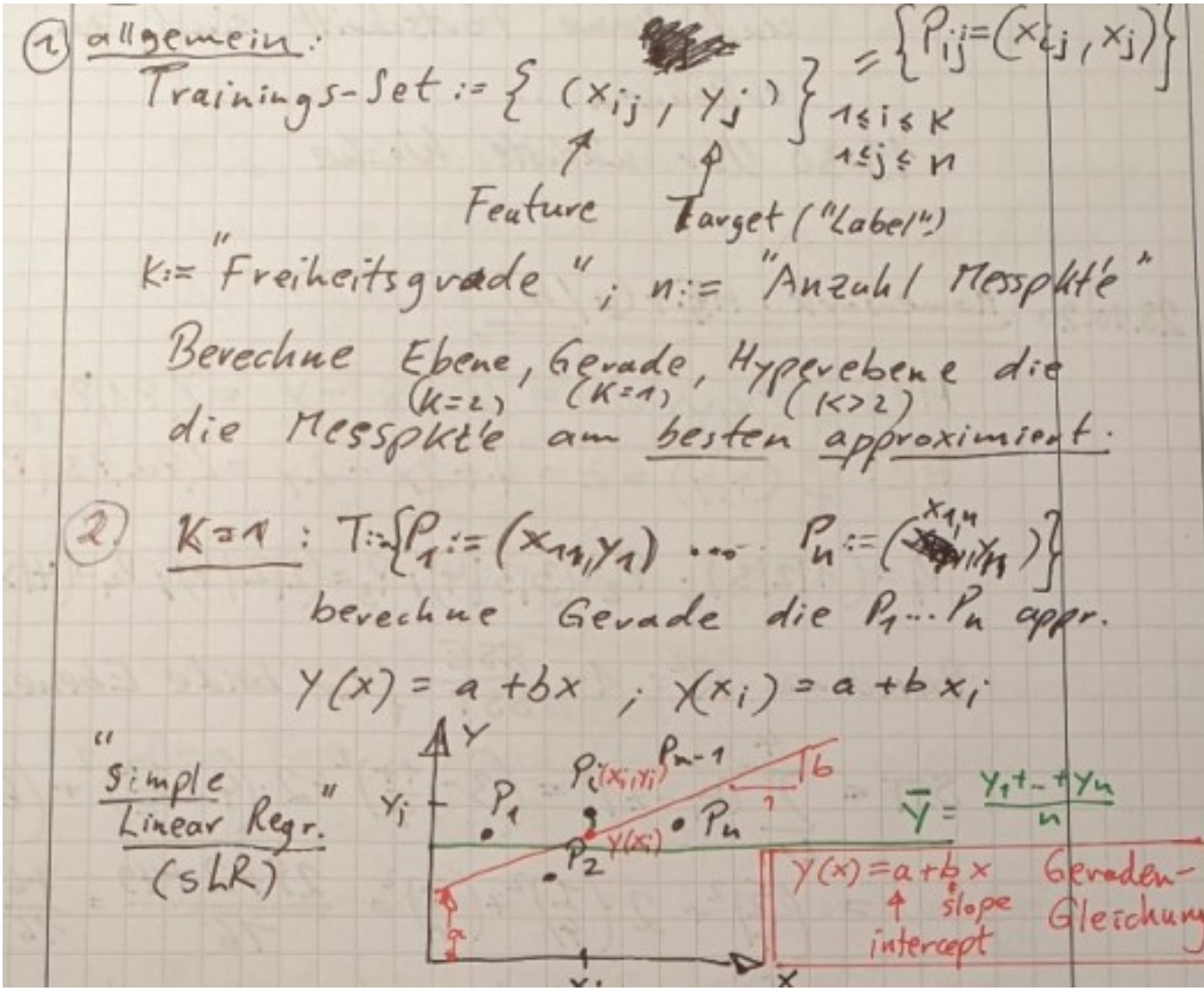
Sprechweise: Wir sagen eine Gerade ist eine **"optimale"** (sLR)-Gerade wenn R^2 maximal ist (dies ist analog zu SSE ist minimal). Oftmals wird in der Literatur auch der Zusatz "optimale" weggelassen und man sagt nur "sLR-Gerade".

Anmerkung: Wir sehen später das für optimale sLR-Geraden eine andere Formel für R^2 auch gültig ist. Diese Formel benutzt die Kennzahl SSR.

****

ab hier bis Ende der subsection sind die Folien der Vorlesung ML zu nutzen und diese sind in Latex umzusetzen

****



4.3.2 Eigenschaften und Theoreme zu R^2

Nun wollen wir die ersten offensichtlichen mathematische Aussagen zu $\mathbf{R^2}$ machen

We see the following theorems:

Define:
$SST = \sum_i (y_i - \text{Mean}(y_i))^2$
$SSE = \sum_i (y_i - f_i)^2$
$SSR = \sum_i (f_i - \text{Mean}(y_i))^2$

Without limitation of generality let $SST \neq 0$

Theorem (TS.1): "Prop. of R^2"

(i) $0 \leq R^2 \leq 1$ "Limitation"
(ii) $R^2 = 0 \Leftrightarrow SSE = SST$ "Minimum"
(iii) $R^2 = 1 \Leftrightarrow SSE = 0$ "Maximum"

Proof: is trivial. We will write it down anyway to practice and repeat mathematical proofs

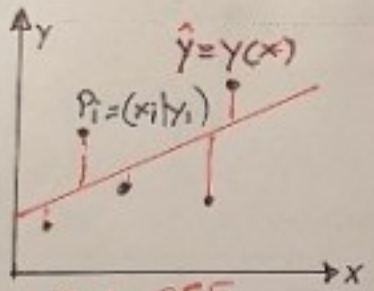
(i) $R^2 \geq 0 \Leftrightarrow 1 - \frac{SSE}{SST} \geq 0 \Leftrightarrow 1 \geq \frac{SSE}{SST} \Leftrightarrow SST \geq SSE$ (per def)

$R^2 \leq 1 \Leftrightarrow 1 \geq 1 - \frac{SSE}{SST} \Leftrightarrow 0 \geq -\frac{SSE}{SST} \Leftrightarrow \frac{SSE}{SST} \geq 0$ (p. def.)

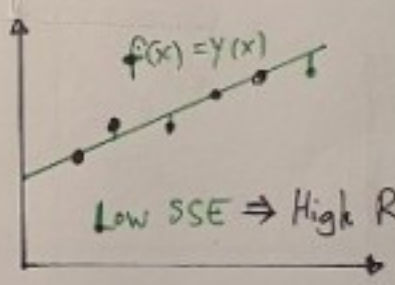
(ii) $0 = 1 - \frac{SSE}{SST} \Leftrightarrow \frac{SSE}{SST} = 1 \Leftrightarrow SSE = SST$

(iii) $1 = 1 - \frac{SSE}{SST} \Leftrightarrow \frac{SSE}{SST} + 1 = 1 \Leftrightarrow \frac{SSE}{SST} = 0 \Leftrightarrow SSE = 0$ q.ed.

Properties/Correlusion out of (TS.1):



High SSE
$\Rightarrow$ Low R^2



for this two examples the "green" f(x) is a better slin. Regr. than the red f(x)

Zur leichteren Schreibweise nutze ich die folgenden Notationen: sum(xi):= $\sum_{i=1}^{n}(x_i)$

und Mean(x)=$\overline{x}$:=1/n*sum(xi).
Manchmal ist es notwendig das die optimale sLR-Gerade durch den Ursprung (a=0) geht. In diesem Fall erhalten wir die folgende Aussage:

Korollar(K-4.1): "sLR ohne Achsenabschnitt(a =0)": Aus a=0 folgt $b = \frac{\overline{x \cdot y}}{\overline{x^2}}$

Beweis: y=b*x $\Rightarrow$ R^2=1-SSE/SST= 1-(1/SST)*sum(yi-b*xi)^2. Da die SLR-Gerade optimal ist, gilt die Ableitung von R nach b = 0.
$\Rightarrow$ 0 = dR^2/db = -2*sum[(yi -b*xi)*(xi)]=-2*sum[xi*yi-b*xi^2]
$\Rightarrow$ sum[xi*yi] =b*sum[xi^2] $\Rightarrow$ b= sum[xi*yi]/sum$(x_i)^2$ = n*Mean(x*y)/n*Mean(x^2)
$\Rightarrow$ b=Mean(x*y)/Mean(x^2).
q.e.d.
Wir brauchen für später auch noch weitere hilfreiche Formeln über Summen und Mittelwerte. Diese werden für die Berechnung der "optimalen" Koeffizienten a, b (siehe: "Least Square Fit" (LSF) Methode):

Proposition (P-4.1): Easily you can proof, that the following equations are valid (definiere M(x):=Mean(xi))
(i) Sum[(xi - M(x))^2] = sum(xi^2) - n*M(x)^2
(ii) Sum[(yi – M(y))^2] = sum(yi^2) - n*M(y)^2
(iii) Sum[(xi - M(x))*(yi - M(y))] = sum(xi*yi) – n*M(x)*M(y)
Beweis: "straightforward"
(i) sum[(xi - M(x))^2] = sum[xi^2- 2*M(x)*(xi) + M(x)^2] (binominal formula) = sum(xi^2)- 2*M(x)* sum(xi)+ sum(M(x)^2)= sum(xi^2)– 2*n*M(x)^2+ n*M(x)^2 weil sum(xi)=n*M(x))

(ii) analog: sum[(yi-M(y))]^2= sum(yi^2)-2*M(y)*sum(yi)+ sum[M(y)^2]=sum(yi^2) - 2n*M(y)^2+n*M(y)^2 = sum(yi^2)- n*M(y)

(iii) analog: sum[(xi-M(x))*(yi-M(y))]= sum[xi*yi- M(x)*yi– xi*M(y)+ M(x)*M(y)] (durch Ausmultiplizieren aller Faktoren)
= sum(xi*yi)– n*M(x)*M(y)- n*M(x)*M(y)+ n*M(x)*M(y) = sum(xi*yi*)-n*M(x)*M(y)
q.e.d.

*** Weitere Kapitel Texte ***

4.3.3 Definition der mLR-Kennzahl adj.R²

In diesem Abschnitt definieren wir die Theorie der "mutiple linear regression".
In einem multiplen Regression Problem arbeiten wir mit n Datenpaaren $\langle \mathbf{x}^{(i)}, y^{(i)} \rangle \in \mathbb{R}^{k+1}$ wobei $\mathbf{x}^{(i)} \in \mathbb{R}^k$ und $y^{(i)} \in \mathbb{R}$ für alle $i \in \{1, \cdots, n\}$.
Die Zahl k ist die Anzahl der unabhängigen Features ("Prädiktoren") $\mathbf{x}^{(i)}$, und $y^{(i)}$ ist die abhängige Zielvariable ("target column"). Die Datenpaare nennen wir auch Trainings-Menge. Unser Ziel ist es die folgende Funktion:

$$F : \mathbb{R}^k \to \mathbb{R} \quad \text{zu berechnen, so dass}$$

$F\big(\mathbf{x}^{(i)}\big)$ eine genaue Approximation von $y^{(i)}$ ist, für alle for all $i \in \{1, \cdots, n\}$, insbesondere wollen wir erreichen:

$$\forall i \in \{1, \cdots, n\} : F\big(\mathbf{x}^{(i)}\big) \approx y^{(i)}.$$

Die Kennzahl adj.R², wobei "adj." für "adjustiert" (angepasst) steht, soll nun genauer definiert werden. Die adj.R² ist eine modifizierte Version des gewöhnlichen R² ("Bestimmtheitsmaß").
Während das normale R² die Proportion der abhängigen Variabilität erklärt, kann das adj.R² bei Modellen mit mehreren unabhängigen Variablen ("Prädiktoren") nützlicher sein, da es für die Anzahl der verwendeten Prädiktoren oder Kovariaten in einem Modell nutzt.
Bezeichnen wir n = Anzahl der Beobachtungen (Datenpunke) und k = Anzahl der unabhängigen Variablen (Prädiktoren) und definieren wir df := n-k-1 ("Anzahl der Freiheitsgrade"). Dann ergibt sich, unter Nutzung von (D-4.4), folgende Formel:

$$\texttt{(D-4.5):} \quad adj.R^2 := 1 - (1 - R^2) \cdot \tfrac{n-1}{n-k-1} = 1 - (\tfrac{SSE}{SST}) \cdot \tfrac{n-1}{n-k-1}$$

Das adj.R² berücksichtigt die Anzahl der unabhängigen Variablen ("Prädiktoren") im Modell und passt das gewöhnliche R² an, um zu verhindern, dass es aufgrund von Überanpassung (Overfitting) zu optimistisch wird. Ein höheres adj.R² zeigt an, dass ein größerer Anteil der Variabilität im abhängigen Wert von den unabhängigen Variablen im Modell erklärt wird, wobei jedoch die Anzahl der Prädiktoren berücksichtigt wird. Zusammenfassend ergibt sich folgende Übersicht:

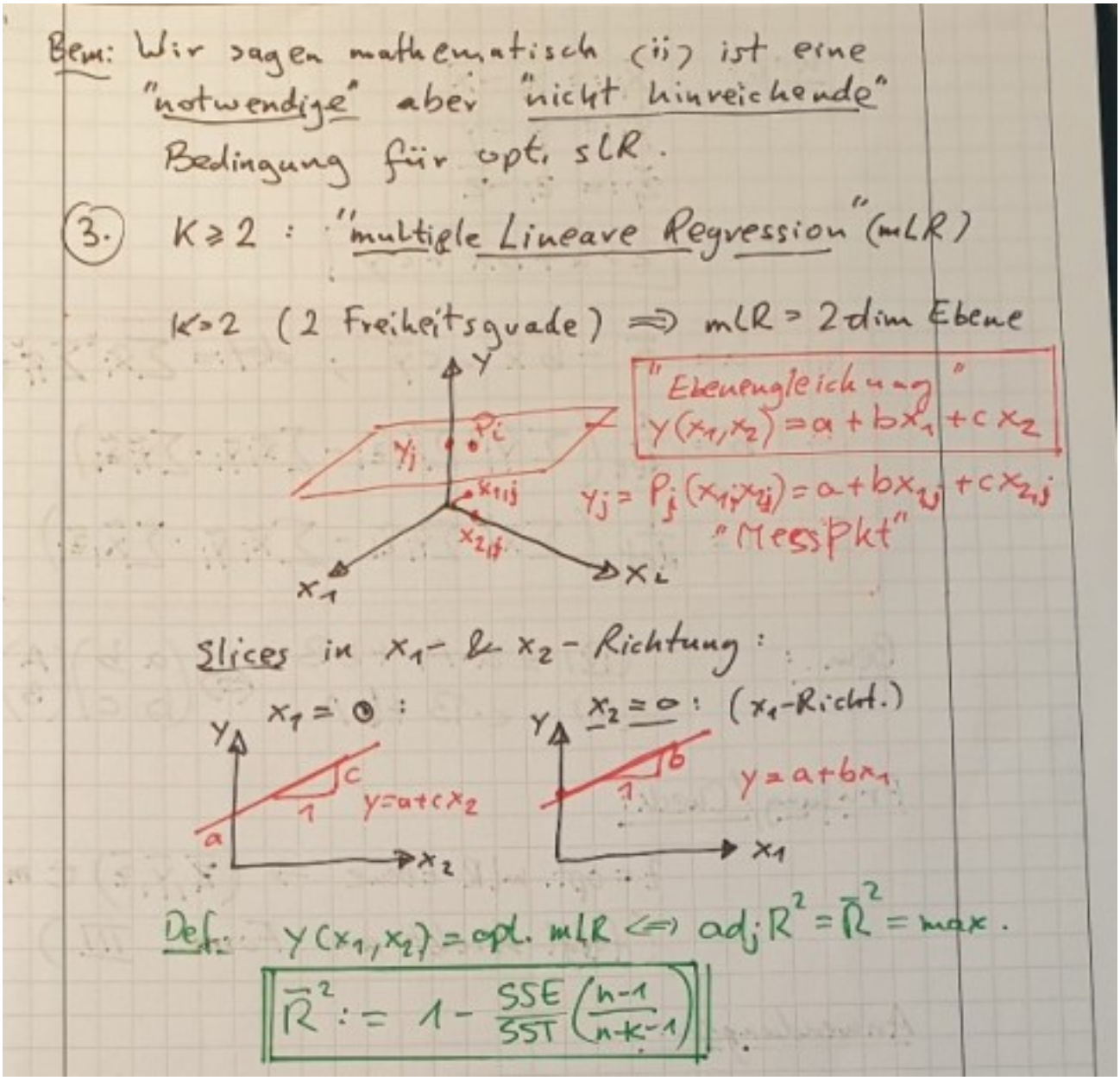
Value	Abbreviation	Formular	Meaning
Number of oberservations	n		measured points, number of training set points
Number of variables	k		several independent variables (k > 1) R^2 must be adjusted
Degrees of freedom	df	$df = n - k - 1$	e.g. df=1; n=4, k=2
Adjusted R-squared	Adj. R^2	$1 - (1 - R^2)\frac{n-1}{n-k-1}$ or $1 - \left(\frac{SSE}{SST}\right)\frac{n-1}{n-k-1}$	how well observed outcomes are replicated by the model

Sprechweise: Wir sagen eine Hyperebene ist eine **"optimale"** (mLR)- Hyperebene wenn adj.R^2 maximal ist (dies ist analog zu SSE ist minimal). Oftmals wird in der Literatur auch der Zusatz "optimale" weggelassen und man sagt nur "mLR-Hyperebene".

****

ab hier bis Ende der subsubsection sind die Folien der Vorlesung ML zu nutzen und diese sind in Latex umzusetzen

****



4.4 Mathematische Berechnungen zu adj.R^2

Die Notation ist analog zu obigen Kapitel. Um die Gleichung $F\left(\mathbf{x}^{(i)}\right) \approx y^{(i)}$ weiter zu präzisieren, definieren wir den gemittelten quadratischen Fehler "mean squared

error":

$$\texttt{MSE} := \frac{1}{n-1} \cdot \sum_{i=1}^{n} \Big(F\big(\mathbf{x}^{(i)}\big) - y^{(i)} \Big)^2. \tag{1}$$

Für eine Liste von Trainingsdaten $[\langle \mathbf{x}^{(1)}, y^{(1)} \rangle, \cdots, \langle \mathbf{x}^{n}, y^{(n)} \rangle]$, haben wir das Ziel der Minimierung von MSE.
Um dies zu erreichen brauchen wir eine Modell für die Funktion F. da seinfachste Modell ist das lineare Modell, i.e. wir nehmen an das F gegeben ist durch

$$F(\mathbf{x}) = \sum_{j=1}^{k} w_j \cdot x_j + b = \mathbf{x}^\top \cdot \mathbf{w} + b \quad \text{wobei } \mathbf{w} \in \mathbb{R}^k \text{ und } b \in \mathbb{R}.$$

hier bezeichnet der Ausdruck $\mathbf{x}^\top \cdot \mathbf{w}$ das Matrix Produkt des Vektors $\mathbf{x}^\top$, welcher eine 1-zu-n Matrix ist, mit dem Vektor $\mathbf{w}$.

********** ab hier weiter nach Deutsch übersetzen **********

Alternatively, this expression could be interpreted as the dot product of the vector $\mathbf{x}$ and the vector $\mathbf{w}$. At this point you might wonder why it is useful to introduce matrix notation here. The reason is that this notation shortens the formula and, furthermore, is more efficient to implement since most programming languages used in machine learning have special library support for matrix operations. Provided the computer is equipped with a graphics card, some programming languages are even able to delegate matrix operations to the graphics unit. This results in a considerable speed-up.

The definition of F given above is the model used in linear regression. Here, $\mathbf{w}$ is called the weight vector and b is called the bias. It turns out that the notation can be simplified if we extend the p-dimensional feature vector $\mathbf{x}$ to an $p+1$-dimensional vector $\mathbf{x}'$ such that

$$x'_j := x_j \quad \text{for all } j \in \{1, \cdots, p\} \quad \text{and} \quad x'_{m+1} := 1.$$

To put it in words, the vector $\mathbf{x}'$ results from the vector $\mathbf{x}$ by appending the number 1:

$$\mathbf{x}' = \langle x_1, \cdots, x_p, 1 \rangle^\top \quad \text{where } \langle x_1, \cdots, x_p \rangle = \mathbf{x}^\top.$$

Furthermore, we define

$$\mathbf{w}' := \langle w_1, \cdots, w_p, b \rangle^\top \quad \text{where } \langle w_1, \cdots, w_p \rangle = \mathbf{w}^\top.$$

Then we have

$$F(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b = \mathbf{w}' \cdot \mathbf{x}'.$$

Hence, the bias has been incorporated into the weight vector at the cost of appending the number 1 at the end of input vector. As we want to use this simplification, from now on we assume that the input vectors $\mathbf{x}^{(i)}$ have all been extended so that their last component is 1. Using this assumption, we define the function F as

$$F(\mathbf{x}) := \mathbf{x}^\top \cdot \mathbf{w}.$$

Now equation (1) can be rewritten as follows:

$$\texttt{MSE}(\mathbf{w}) = \frac{1}{m-1} \cdot \sum_{i=1}^{m} \Big((\mathbf{x}^{(i)})^\top \cdot \mathbf{w} - y^{(i)} \Big)^2. \tag{2}$$

Our aim is to rewrite the sum appearing in this equation as a scalar product of a vector with itself. To this end, we first define the vector $\mathbf{y}$ as follows:

$$\mathbf{y} := \langle y^{(1)}, \cdots, y^{(m)} \rangle^\top.$$

Note that $\mathbf{y} \in \mathbb{R}^m$ since it has a component for all of the m training examples. Next, we define the design matrix X as follows:

$$X := \begin{pmatrix} \big(\mathbf{x}^{(1)}\big)^\top \\ \vdots \\ \big(\mathbf{x}^{(m)}\big)^\top \end{pmatrix}$$

Defined this way, the row vectors of the matrix X are the vectors $\mathbf{x}^{(i)}$ transposed. Now we have the following:

$$X \cdot \mathbf{w} - \mathbf{y} = \begin{pmatrix} \big(\mathbf{x}^{(1)}\big)^\top \\ \vdots \\ \big(\mathbf{x}^{(m)}\big)^\top \end{pmatrix} \cdot \mathbf{w} - \mathbf{y} = \begin{pmatrix} \big(\mathbf{x}^{(1)}\big)^\top \cdot \mathbf{w} - y_1 \\ \vdots \\ \big(\mathbf{x}^{(m)}\big)^\top \cdot \mathbf{w} - y_m \end{pmatrix}$$

Taking the square of the vector $X \cdot \mathbf{w} - \mathbf{y}$ we discover that we can rewrite equation (2) as follows:

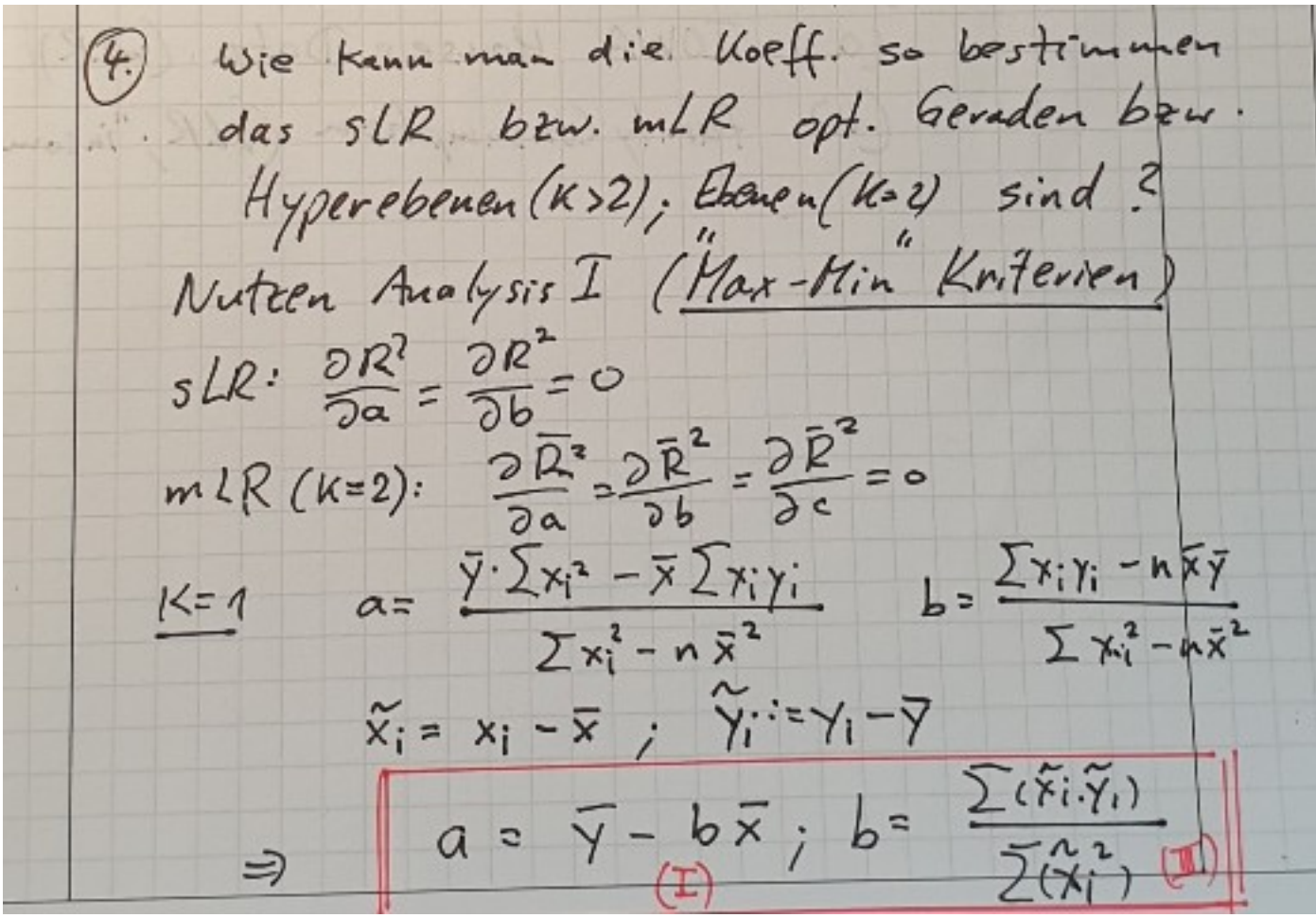
$$\texttt{MSE}(\mathbf{w}) = \frac{1}{m-1} \cdot \big(X \cdot \mathbf{w} - \mathbf{y}\big)^\top \cdot \big(X \cdot \mathbf{w} - \mathbf{y}\big). \tag{3}$$

****

Um das minimale MSE(w) zu bekommen, sagt uns die Mathematik (Analysis I), müssen wir den "Gradienten" der Funktion MSE(w)" null setzen. Die folgenden Berechnungen sind hier noch zu ergänzen. Nützlich dazu ist die Matrizenrechnung aus der Linearen Algebra.

****

4.5 "Least Square Fit"(LSF) Verfahren für LR



(4.) Wie kann man die Koeff. so bestimmen das sLR bzw. mLR opt. Geraden bzw. Hyperebenen ($K>2$); Ebenen ($K=2$) sind?

Nutzen Analysis I ("Max-Min" Kriterien)

sLR: $\frac{\partial R^2}{\partial a} = \frac{\partial R^2}{\partial b} = 0$

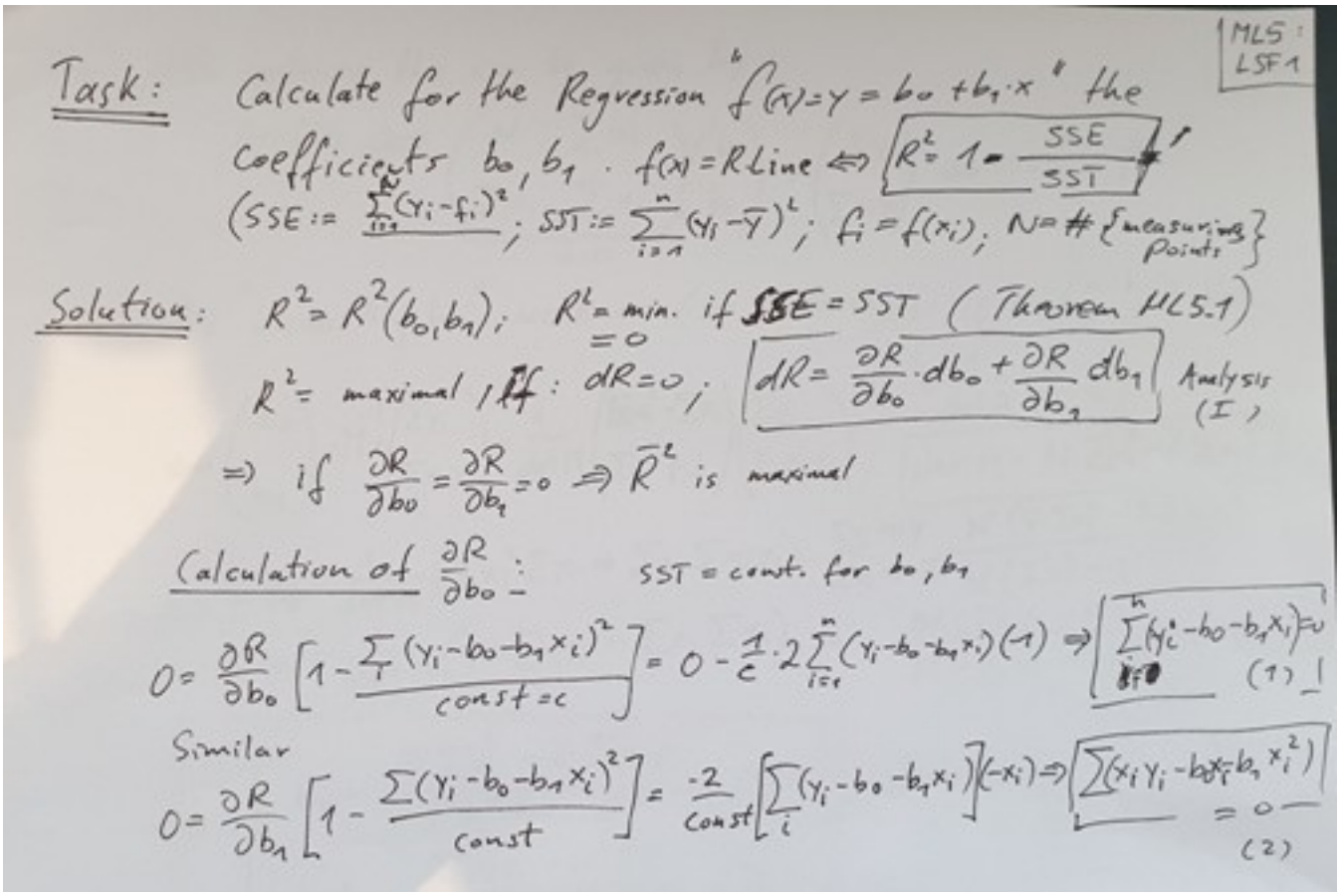
mLR ($K=2$): $\frac{\partial \bar{R}^2}{\partial a} = \frac{\partial \bar{R}^2}{\partial b} = \frac{\partial \bar{R}^2}{\partial c} = 0$

$K=1$ $\quad a = \frac{\bar{y} \cdot \sum x_i^2 - \bar{x} \sum x_i y_i}{\sum x_i^2 - n\bar{x}^2} \qquad b = \frac{\sum x_i y_i - n\bar{x}\bar{y}}{\sum x_i^2 - n\bar{x}^2}$

$\tilde{x}_i = x_i - \bar{x}$; $\tilde{y}_i := y_i - \bar{y}$

$\Rightarrow$ $a = \bar{y} - b\bar{x}$ (I) ; $b = \frac{\sum(\tilde{x}_i \cdot \tilde{y}_i)}{\sum(\tilde{x}_i^2)}$ (II)

4.5.1 Detailberechnung für sLR (k=1)



With matrices this can be given by:

$$(1),(2) \Leftrightarrow \underbrace{\begin{pmatrix} N & \sum x_i \\ \sum x_i & \sum x_i^2 \end{pmatrix}}_{=:M} \begin{pmatrix} b_0 \\ b_1 \end{pmatrix} = \begin{pmatrix} \sum y_i \\ \sum x_i y_i \end{pmatrix} \Leftrightarrow M \cdot \vec{b} = \vec{y}$$

By "Linear Algebra" we know (see Notes Page): if $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$ then $A^{-1} = \frac{1}{\det A} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix}$

$$\Leftrightarrow \begin{pmatrix} b_0 \\ b_1 \end{pmatrix} = M^{-1} \cdot \begin{pmatrix} \sum y_i \\ \sum x_i y_i \end{pmatrix} = \frac{1}{\det M} \begin{pmatrix} \sum x_i^2 & -\sum x_i \\ -\sum x_i & N \end{pmatrix} \begin{pmatrix} \sum y_i \\ \sum x_i y_i \end{pmatrix}; \quad \boxed{\det M = N \cdot \sum x_i^2 - (\sum x_i)^2}$$

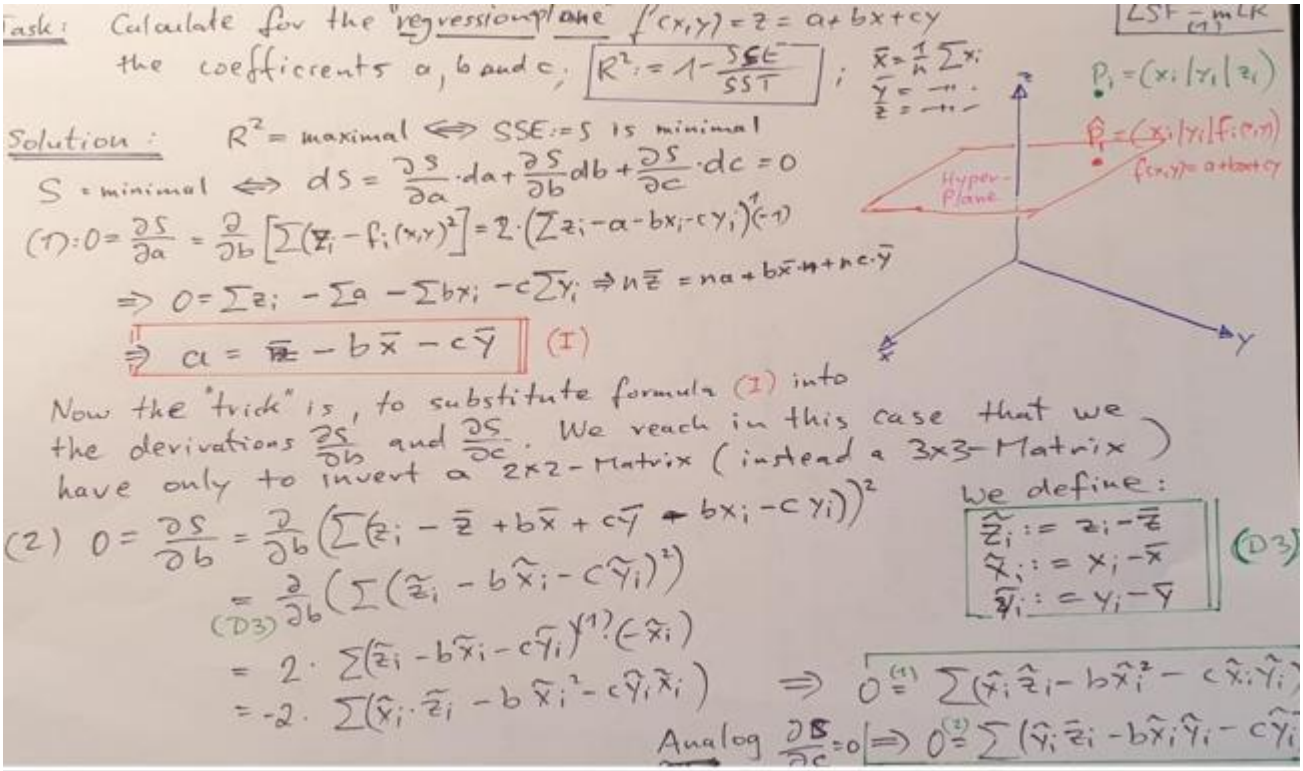
$$\Leftrightarrow \begin{cases} b_0 = \frac{1}{\det M} \left(\sum x_i^2 \sum y_i - \sum x_i \sum x_i y_i \right) \overset{\substack{\sum y_i = N\bar{y} \\ \sum x_i = N\bar{x}}}{=} \frac{N \cdot (\bar{y} \cdot \sum x_i^2 - \bar{x} \sum x_i y_i)}{N \left(\sum x_i^2 - N \bar{x}^2 \right)} \\ b_1 = \frac{1}{\det M} \left(N \cdot \sum x_i y_i - \sum x_i \sum y_i \right) = \frac{N \left(\sum x_i y_i - \bar{x} \cdot N \bar{y} \right)}{N \left(\sum x_i^2 - N \bar{x}^2 \right)} \end{cases}$$

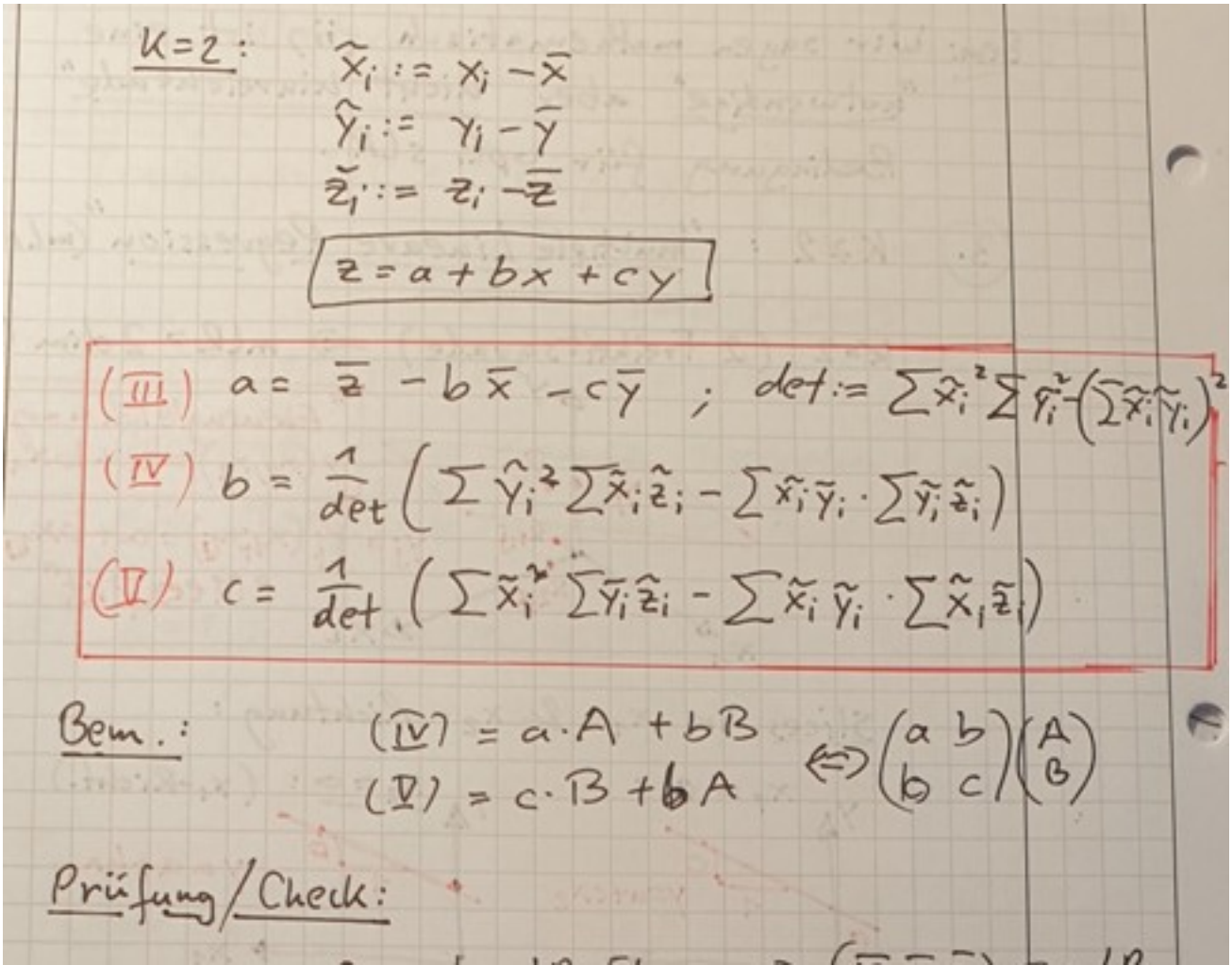
$$\Leftrightarrow \begin{cases} b_0 = \dfrac{\bar{y} \cdot \sum x_i^2 - \bar{x} \sum x_i y_i}{\sum x_i^2 - N \cdot \bar{x}^2} & (I) \\ b_1 = \dfrac{\sum x_i y_i - N \cdot \bar{x} \bar{y}}{\sum x_i^2 - N \cdot \bar{x}^2} & (II) \end{cases}$$

Formula for the coefficients of a Regression-Line (sLR)

$$\boxed{y = a + b \cdot x}$$

4.5.2 Detailberechnung für mLR(k=2)





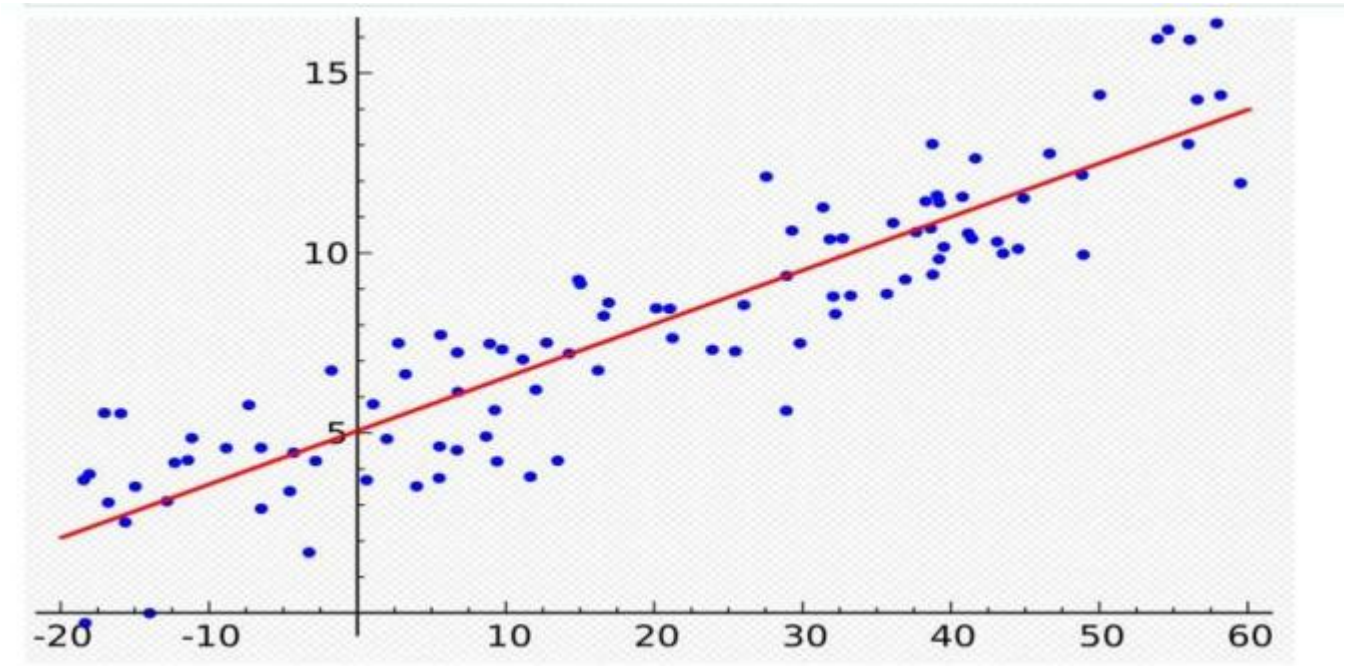
*** Weitere Kapitel Texte ***

4.6 simple Linear Regression (sLR) with Scikit-learn

In order to get a better feeling for linear regression, we want to test it to investigate the factors that determine the fuel consumption of cars. "cars.csv" which I have adapted from the file

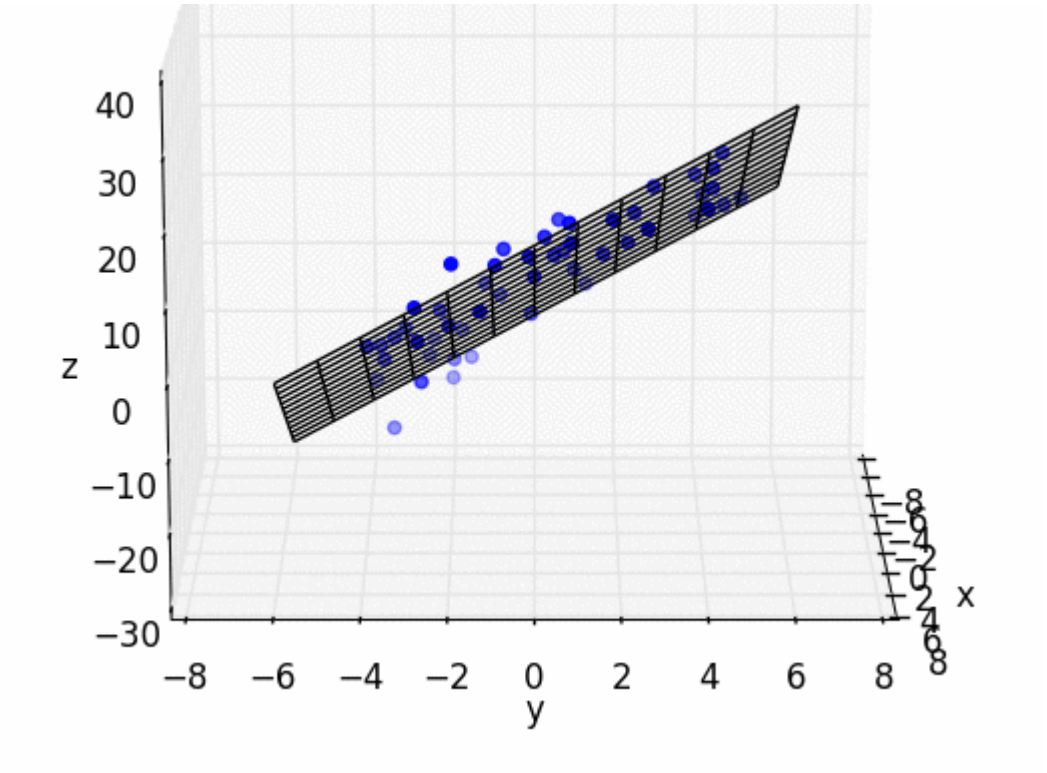
http://www-bcf.usc.edu/~gareth/ISL/Auto.csv.

Altogether, this file contains data of 392 different car models.



4.7 Anwendungen zu mLR

*** Weitere Kapitel Texte ***



Ein bewegtes Bild davon liegt in der Referenz [HVö-5]: https://github.com/HVoellinger/Lecture-Notes-to-ML-WS2020/blob/master/ML5-QYuIc.gif

4.8 Übungen zum Kapitel 5

4.8.1 Übung 5.1 - .....

dependent variable	explained variance
`displacement`	0.75
`cyl`	0.70
`hp`	0.73
`weight`	0.78
`acc`	0.21
`year`	0.31

Table 2: Explained variance for various dependent variables.

Spalte 1	Spalte 2
Inhalt 1	Inhalt 2

Table 3: Eine Tabelle mit Positionsangabe "t".

4.8.2 Übung 5.2 - .....

Spalte 1	Spalte 2
Inhalt 1	Inhalt 2

Table 4: Eine Tabelle mit Positionsangabe "h".

4.8.3 Übung 5.3 - .....

TEXT

4.8.4 Übung 5.4 - .....

4.8.5 Übung 5.5

:

Calculation of mLR(k=2)-model for "Students Examination Results"

Similar to Example (5.1): Find "least square fit" z = a + b*x + c*y for the z:="Achieved points(score) of exam[pt.]" depending on the two parameter: x:="Effort exam preparation[h]" and y:="Effort for homework [h]". Data from Training Set TS =(x, y; z) — (exam prep.[h], homework[h]; score[pt.])= (7,5;41), (3,4;27), (5,5;35), (3,3;26), (8,9;48), (7,8;45), (10,10;46), (3,5;27), (5,3;29), (3,3;19) Task: Build to the model mLR(x,y; z). Compare and check your result with the output of a Python-Program. Answer the following three Questions: Q1: How much points would a student achieve without any preparation and without doing any homework? Q2: How much points would a student achieve with (10 hours of preparation for the exam) and (10 hours homework) ? Q3: How much effort you will need to reach enough points (=25) to score enough points to pass the exam? Add. Question/Remark: Our calculation use both variables in the calculation. What is the difference to our sLR-model results? Compare Adj.R^2 (calculated here) to the two R^2 you got in Example (E5.1).

für eine Lösung vergleiche:
https://github.com/HVoellinger/Lecture-Notes-to-ML-WS2020/blob/master/LR-Calculation_of_Coeff.xlsx

*** Weitere Kapitel Texte ***

5 Text-Klassifikation mit dem Naive-Bayes-Klassifikator

Wir zeigen hier Text-Klassifikation mit dem **Naive Bayes- Klassifikator** ("Bayes Learning" via "naïve Bayes Classifier"). Grundlage ist das Bayes-Theorem ("Bayes Rule"), welches sich mit bedingten Wahrscheinlichkeiten befasst.
Die Naive-Bayes-Textklassifikation ist eine Methode des maschinellen Lernens, die auf dem Naive-Bayes-Theorem basiert und häufig zur Kategorisierung von Textdaten verwendet wird. Sie ist besonders nützlich für Anwendungen wie Spam-Erkennung, Sentiment-Analyse, Themenklassifikation und mehr.

Der Ansatz ist "naiv", weil er eine starke Unabhängigkeitsannahme zwischen den Features (Wörtern) macht, was in der Praxis oft nicht der Fall ist.

5.1 Eine Einführung in den naiven Bayes-Klassifikator

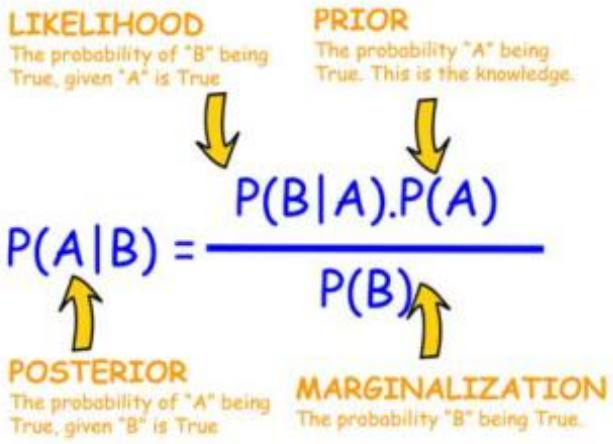
Hier ist eine grundlegende Erklärung, wie die Naive-Bayes-Textklassifikation funktioniert:

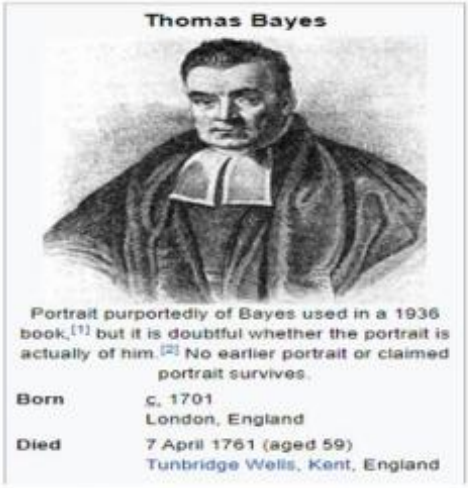
1. Naive-Bayes-Theorem:
Das Naive-Bayes-Theorem ist eine statistische Formel, die auf dem Satz von Wahrscheinlichkeitsregeln beruht, bekannt als **Bayes-Theorem**. Es gibt an, wie Wahrscheinlichkeiten nach der Anwendung neuer Informationen aktualisiert werden. Für die Textklassifikation bedeutet dies, dass wir die Wahrscheinlichkeit berechnen, mit der ein Dokument einer bestimmten Klasse (z. B. "Positiv" oder "Negativ") angehört, basierend auf den darin enthaltenen Wörtern.
2. Feature-Extraktion:
Für jeden Text werden Features (Wörter) extrahiert, die als Eingabe für den Naive-Bayes-Klassifikator dienen. In der Regel werden diese Wörter als "Bag of Words" betrachtet, d. h. die Reihenfolge der Wörter im Text wird ignoriert.
3. Berechnung der Wahrscheinlichkeiten:
Für jede Klasse wird die Wahrscheinlichkeit berechnet, dass ein gegebenes Dokument dieser Klasse angehört. Dies erfolgt, indem die Wahrscheinlichkeiten für jedes im Dokument vorkommende Wort (Feature) multipliziert werden. Die Wahrscheinlichkeiten werden aus Trainingsdaten abgeleitet.
4. Klassifikation:
Das Dokument wird der Klasse zugeordnet, für die die berechnete Wahrscheinlichkeit am höchsten ist. Dies wird oft durch den Satz von Klassenlabels erreicht, die im Trainingsprozess erlernt wurden.

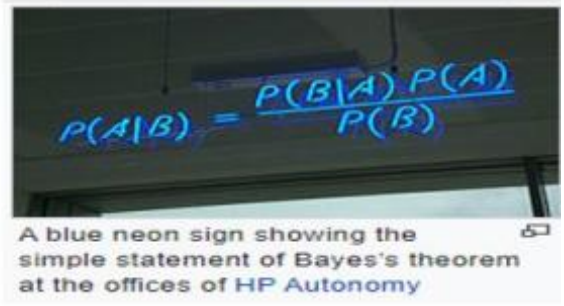
Bayes Learning (Conditional Probabilitiy):

In probability theory and statistics, **Bayes' theorem**) describes the probability of an event, based on prior knowledge of conditions that might be related to the event. Bayes theorem is named after Reverend Thomas Bayes (/beɪz/; 1701?–1761).

More details later in this chapter







A blue neon sign showing the simple statement of Bayes's theorem at the offices of HP Autonomy

Figure 4: Screenshots zum Naive-Bayes-Learning

5. Laplace-Glättung:
Ein Problem bei der Verwendung von Wahrscheinlichkeiten auf der Grundlage von Trainingsdaten ist, dass einige Wörter möglicherweise in bestimmten Klassen überhaupt nicht vorkommen. Dies könnte dazu führen, dass die berechnete Wahrscheinlichkeit null wird und das Modell nicht in der Lage ist, Vorhersagen für diese Klasse zu treffen. Um dieses Problem zu mildern, wird oft die Laplace-Glättung verwendet, um die Wahrscheinlichkeiten leicht zu verschieben.

Zusammenfassung:
Naive-Bayes-Klassifikatoren sind einfach zu implementieren, schnell und können auch bei begrenzten Trainingsdaten gut funktionieren. Allerdings kann die starke Unabhängigkeitsannahme zwischen den Features in einigen Fällen zu weniger genauen Vorhersagen führen, insbesondere wenn es komplexe Abhängigkeiten zwischen den Wörtern gibt. Dennoch ist die Naive-Bayes-Textklassifikation eine nützliche Methode, um eine erste Annäherung an die Klassifikation von Textdaten zu erhalten.
****

ab hier bis Ende der Section sind die Folien der Vorlesung ML zu nutzen und diese sind in Latex umzusetzen
****

5.2 Mathematische Grundlagen bei Textklassifikatoren

5.2.1 Bayes-Learning für Texte

Bayes' Theorem is useful when working with conditional probabilities (like we are doing here), because it provides us with a way to reverse them. In our case, we have, so using this theorem we can reverse the conditional probability:

$$P(sports|a\ very\ close\ game) = \frac{P(a\ very\ close\ game|sports) \times P(sports)}{P(a\ very\ close\ game)}$$

Since for our classifier we're just trying to find out which tag has a bigger probability, we can discard the divisor —which is the same for both tags— and just compare

$$P(a\ very\ close\ game|Sports) \times P(Sports)$$

with

$$P(a\ very\ close\ game|Not\ Sports) \times P(Not\ Sports)$$

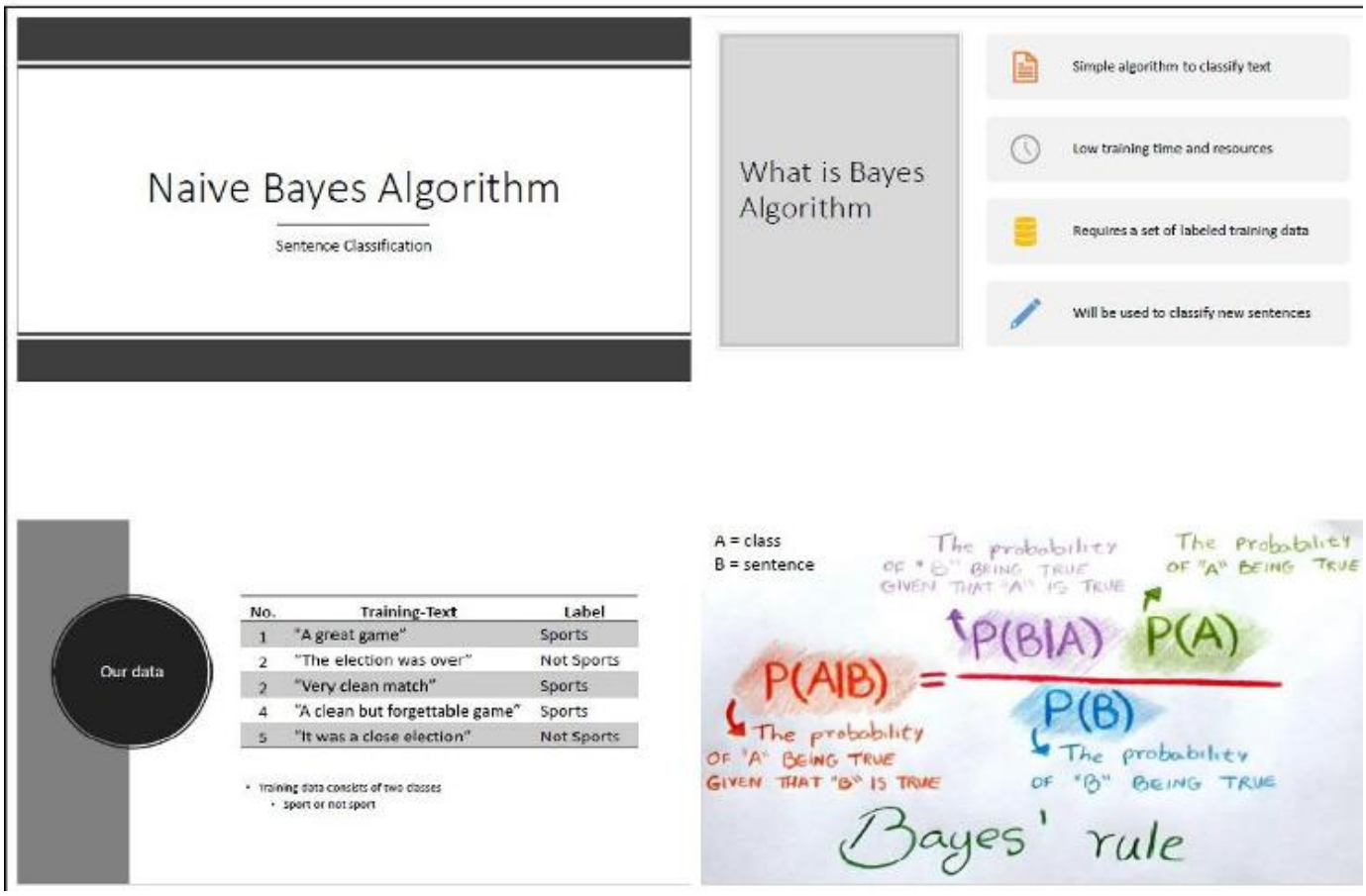
5.2.2 "Laplace Glättung"

In statistics, **additive smoothing**, also called **Laplace smoothing**[1] (not to be confused with Laplacian smoothing as used in image processing), or **Lidstone smoothing**, is a technique used to smooth categorical data. Given an observation $\mathbf{x} = \langle x_1, x_2, \ldots, x_d \rangle$ from a multinomial distribution with N trials, a "smoothed" version of the data gives the estimator:

$$\hat{\theta}_i = \frac{x_i + \alpha}{N + \alpha d} \qquad (i = 1, \ldots, d),$$

where the "pseudocount" $\alpha > 0$ is a smoothing parameter. $\alpha = 0$ corresponds to no smoothing. (This parameter is explained in § Pseudocount below.) Additive smoothing is a type of shrinkage estimator, as the resulting estimate will be between the empirical probability (relative frequency) x_i/N, and the uniform probability $1/d$. Invoking Laplace's rule of succession, some authors have argued[*citation needed*] that α should be 1 (in which case the term **add-one smoothing**[2][3] is also used)[*further explanation needed*], though in practice a smaller value is typically chosen.

5.3 Konkrete Anwendung des naiven Bayes-Klassifikator



How does it work?

- Comparing the probabilities:
  - P (Sport| Hermann played a TT match) = $\frac{P(A\ very\ close\ game|Sports)*P(Sports)}{P(A\ very\ close\ game)}$
  - P (Not Sports | Hermann played a TT match) = $\frac{P(A\ very\ close\ game|not\ Sports)*P(Not\ Sports)}{P(A\ very\ close\ game)}$

Probability of a sentence

- Likelyhood a setence is a sport sentence:
  - P(Sport) = $\frac{Number\ of\ sentences\ in\ class\ "Sports"}{Total\ number\ of\ sentences\ in\ the\ training\ set}$
    - Similarly calculate P(Not Sports)
- „Naive" Bayes because we think each word is indipendent from the other ones
  - Possibility of a sentence „A very close game" is calculated like this:
    - $P(A\ very\ close\ game) = P(A) * P(very) * P(close) * P(game)$

Probability of a sentence in a class

- Applying the probabilites of the words to Bayes formula:

$$P(A\ very\ close\ game\ |Sports) = P(A|Sports) * P(very|Sports) * P(close|Sports) * P(game|Sports) * P(Sports)$$

Calculating the probabilities

- Now all we have to do is calculate all the different probabilites by counting everything in our training data

No.	Training-Text	Label
1	"A great game"	Sports
2	"The election was over"	Not Sports
3	"Very clean match"	Sports
4	"A clean but forgettable game"	Sports
5	"It was a close election"	Not Sports

- P(Sports) = 3/5 | P(Not Sports) = 2/5
- Probability of a word in class Sports:
  - $P(game|Sports) = \frac{amount\ of\ "game"\ in\ Sports\ sentences}{total\ number\ of\ words\ in\ Sports\ sentences} = \frac{2}{11}$
  - Repeat for other words and other classes

How does it work?

- Comparing the probabilities:
  - P (Sport| Hermann played a TT match) = $\frac{P(A\ very\ close\ game|Sports)*P(Sports)}{P(A\ very\ close\ game)}$
  - P (Not Sports | Hermann played a TT match) = $\frac{P(A\ very\ close\ game|not\ Sports)*P(Not\ Sports)}{P(A\ very\ close\ game)}$

Probability of a sentence

- Likelyhood a setence is a sport sentence:
  - P(Sport) = $\frac{Number\ of\ sentences\ in\ class\ "Sports"}{Total\ number\ of\ sentences\ in\ the\ training\ set}$
    - Similarily calculate P(Not Sports)
- „Naive" Bayes because we think each word is indipendent from the other ones
  - Possibility of a sentence „A very close game" is calculated like this:
    - $P(A\ very\ close\ game) = P(A) * P(very) * P(close) * P(game)$

Probability of a sentence in a class

- Applying the probabilites of the words to Bayes formula:

$$P(A\ very\ close\ game\ |Sports) = P(A|Sports) * P(very|Sports) * P(close|Sports) * P(game|Sports) * P(Sports)$$

Calculating the probabilities

- Now all we have to do is calculate all the different probabilites by counting everything in our training data

No.	Training-Text	Label
1	"A great game"	Sports
2	"The election was over"	Not Sports
3	"Very clean match"	Sports
4	"A clean but forgettable game"	Sports
5	"It was a close election"	Not Sports

- P(Sports) = 3/5 | P(Not Sports) = 2/5
- Probability of a word in class Sports:
  - $P(game|Sports) = \frac{amount\ of\ "game"\ in\ Sports\ sentences}{total\ number\ of\ words\ in\ Sports\ sentences} = \frac{2}{11}$
    - Repeat for other words and other classes

Task: Let's see how this works in practice with a simple example. Suppose we are **building a classifier that says whether a text is about sports or not.** Our training data has 5 sentences:

Training-Text and Target-Text:

No.	Training-Text	Label
1	"A great game"	Sports
2	"The election was over"	Not Sports
2	"Very clean match"	Sports
4	"A clean but forgettable game"	Sports
5	"It was a close election"	Not Sports
	Target-Text	
new	*"A very close game"*	??????????

Bayes' Theorem is useful when working with conditional probabilities (like we are doing here), because it provides us with a way to reverse them. In our case, we have, so using this theorem we can reverse the conditional probability:

$$P(sports|a\ very\ close\ game) = \frac{P(a\ very\ close\ game|sports) \times P(sports)}{P(a\ very\ close\ game)}$$

Since for our classifier we're just trying to find out which tag has a bigger probability, we can discard the divisor —which is the same for both tags— and just compare

$$P(a\ very\ close\ game|Sports) \times P(Sports)$$

with

$$P(a\ very\ close\ game|Not\ Sports) \times P(Not\ Sports)$$

So here comes the *Naive* part: we assume that every word in a sentence is **independent** of the other ones. This means that we're no longer looking at entire sentences, but rather at individual words. So for our purposes, "this was a fun party" is the same as "this party was fun" and "party fun was this".

We write this as:

$$P(a\ very\ close\ game) = P(a) \times P(very) \times P(close) \times P(game)$$

This assumption is very strong but super useful. It's what makes this model work well with little data or data that may be mislabeled. The next step is just applying this to what we had before:

$$P(a\ very\ close\ game|Sports) = P(a|Sports) \times P(very|Sports) \times P(close|Sports) \times P(game|Sports)$$

Calculating Probabilities:

The final step is just to calculate every probability and see which one turns out to be larger. Calculating a probability is just counting in our training data. First, we calculate the *a priori* probability of each tag: for a given sentence in our training data, the probability that it is *Sports* = P(Sports)=3/5. Then, P(Not Sports)= 2/5. That's easy enough.

Then, calculating P(game|Sports) means counting how many times the word "game" appears in *Sports* texts (2) divided by the total number of words in *sports* (11). Therefore, P(game|Sports)=2/11.

However, we run into a problem here: "close" doesn't appear in any *Sports* text! That means that P(close|Sports)=0. This is rather inconvenient since we are going to be multiplying it with the other probabilities, so we'll end up with zero.

How do we do it? By using something called Laplace smoothing: we add 1 to every count so it's never zero. To balance this, we add the number of possible words to the divisor, so the division will never be greater than 1. In our case, the possible words are (see notespage):
'a' 'great' 'very' 'over' 'it' 'but' 'game' 'election' 'clean' 'close' 'the' 'was' 'forgettable' 'match' .
Since the number of possible words is 14 (I counted them!), applying smoothing we get
that P(game|Sports)=(2+1)/(11+14)=3/25. The full results are:

P(Sports)=3/5
P(Not Sports)=2/5
Anzahl (Words|Sports)=11
(Words|Not Sports)=9
(possible words)=14 <-- see notes

Word	P(word | Sports)	P(word | Not Sports)
a	3/25	2/23
very	2/25	1/23
close	1/25	2/23
game	3/25	1/23

==> P(a|Sports)* P(very|Sports)* P(close|Sports)* P(game|Sports)* P(Sports) = 3/25*2/25*1/25*3/25*3/5 = 18/25*1/25³*3/5 = 54/125*1/25³~ **2,7648e-5**

Analog: P(a|Not Sports)*P(very|Not Sports)*P(close|Not Sports)*P(game|Not Sports)*P(Not Sports) = 2/23*1/23*2/23*1/23*2/5= 8/115*1/23³**~0,57176e-5**

=➔ the Sentence '*a very close game*' is tagged as Sports.

5.4 Übungen zum Kapitel 5

****

ab hier bis Ende der Übungen sind die Folien der Vorlesung ML zu nutzen und diese sind in Latex umzusetzen

****

.....

5.4.1 Übung 5.1 - Bsp. einer Bayes-Texklassifikation

Analog dem Beispiel aus obigen Kapitel soll die Bayes- Textklassifikation für einen weiteren Zielsatz mit den gleichen **"gelabelten" Trainingsdaten** durchgeführt werden. Wir suchen jetzt eine Klassifikation für den Zielsatz **"Hermann plays a TT match"**:

No.	Training-Text	Label
1	"A great game"	Sports
2	"The election was over"	Not Sports
2	"Very clean match"	Sports
4	"A clean but forgettable game"	Sports
5	"It was a close election"	Not Sports
6	"A very close game"	Sports
	Target-Text	
new	*"Hermann plays a TT match"*	??????????

Zusatzfrage: Wie verändert sich das Ergebnis wenn ich als Zielsatz: **"Hermann plays a very clean game"** eingebe?

5.4.2 Übung 5.2 - Bayes-Textklassifikation in Python

Definiere einen Algorithmus in Python (benutze Jupyter Notebook) um obige Berechnung zu automatisieren. Tipp: Vergleich analoge Beispiele in **[HVö-5]** - https://github.com/HVoellinger/Lecture-Notes-to-ML-WS2020

5.4.3 Übung 5.3

: *** Weitere Kapitel Texte ***

6 Verfahren der "Support Vector Machines" (SVM)

*** Referenzen: [Wiki-SVM]; [SVM-Def] und [TK + SVM]*****

Wichtig zum Verständnis von SVM ist die Beschreibung einiger mathematischen Grundlagen der SVM. Diese sind in der Tat traditionelle Lineare Algebra und keine "Rocket Science". Es geht bei diesem Klassifikationsproblem darum optimale Hyperebenen zu finden die die Klassen "gut" trennen. Dies kann man etwa auch gut am Beispiel von Text-Klassifikationen sehen. Text-Klassifikatiion hat sehr viel mit SVM zu tun. Vergleichen Sie dazu die Beispiele die in der Referenz [TK + SVM] zu sehen sind.

6.1 Grundlegende mathematische Funktionsweise

Das Hauptziel der SVM besteht darin, eine optimale Trennung (Klassifikation) zwischen zwei Klassen von Datenpunkten in einem mehrdimensionalen Raum zu finden. Die SVM sucht eine Hyperebene, die die beiden Klassen maximal voneinander trennt, wobei der Abstand zwischen der Hyperebene und den nächsten Datenpunkten (den sogenannten Support-Vektoren) maximiert wird.
Mathematisch Schreibweise:
Angenommen, wir haben eine Menge von Trainingsdatenpunkten: $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$, wobei $x_i \in \mathbb{R}^n$ der Eingabevektor und y_i als Klasse von Datenpunkten nur die Werte +1 oder -1 annehmen kann.
Für eine lineare SVM ist die mathematische Darstellung der Hyperebene dann gegeben durch:

$$w \cdot x + b = 0$$

Hier ist "w" der Gewichtsvektor, der senkrecht zur Hyperebene zeigt und die Richtung der Trennung bestimmt. "b" ist der Bias (auch Verschiebungsparameter genannt), der die Verschiebung der Hyperebene entlang der "w"-Achse steuert. Es gelten dabei die folgenden **matematischen Bedingungen**
1. Lineare Trennbarkeit:
Die SVM hat bestimmte Trennbedingungen, die während des Trainingsprozesses erfüllt werden sollen: Die Punkte jeder Klasse müssen auf unterschiedlichen Seiten der Hyperebene liegen.
Mathematisch ausgedrückt gilt für positive Beispiele ($y_i = 1$): $w \cdot x_i + b \geq 1$
und für negative Beispiele ($y_i = -1$): $w \cdot x_i + b \leq -1$
Der Abstand (Margin) zwischen den nächsten Datenpunkten (Support-Vektoren)

und der Hyperebene muss maximal sein. Der Abstand zwischen zwei parallelen Hyperebenen, die die Support-Vektoren berühren, wird als Margin bezeichnet.
2. Optimierung:
Das Ziel der SVM ist es, den Margin zu maximieren, indem die Länge des Gewichtsvektors "w" minimiert wird. Das führt zu einem quadratischen Optimierungsproblem. In der linearen SVM lautet das Optimierungsproblem:

$$\textbf{Minimiere } ||w||^2$$

Unter den Bedingungen: $y_i \cdot (w \cdot x_i + b) \geq 1$ für alle Datenpunkte (x_i, y_i)

3. Kernel-Trick:
In Fällen, in denen die Daten nicht linear separierbar sind, kann der sogenannte Kernel-Trick angewendet werden. Der Kernel-Trick ermöglicht es, die Daten in einen höherdimensionalen Raum zu transformieren, in dem sie linear separierbar werden. Beliebte Kernel-Funktionen sind beispielsweise der lineare Kernel, der polynomiale Kernel und der RBF (Radial Basis Function) Kernel.

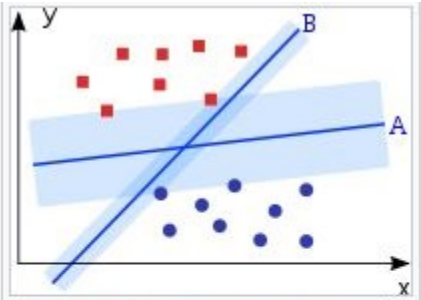
Anmerkung: Wie kommt es zu dem Namen "Kernel-Trick"?
Der Name "Kernel" kommt daher, dass in der mathematischen Darstellung der SVM die Funktion, die die Datenpunkte in den höherdimensionalen Raum transformiert, als Kernel-Funktion bezeichnet wird.

Dies sind die grundlegenden mathematischen Grundlagen der Support Vector Machine. SVM ist eine äußerst flexible und leistungsfähige Methode, die in vielen Anwendungen erfolgreich eingesetzt wird.

*** Weitere Kapitel Texte ***

6.2 Funktionsweise im Detail

Ausgangsbasis für den Bau einer Support Vector Machine ist eine Menge von Trainingsobjekten, für die jeweils bekannt ist, welcher Klasse sie zugehören. Jedes Objekt wird durch einen Vektor in einem Vektorraum repräsentiert. Aufgabe der Support Vector Machine ist es, in diesen Raum eine Hyperebene einzupassen, die als Trennfläche fungiert und die Trainingsobjekte in zwei Klassen teilt. Der Abstand derjenigen Vektoren, die der Hyperebene am nächsten liegen, wird dabei maximiert. Dieser breite, leere Rand soll später dafür sorgen, dass auch Objekte, die nicht genau den Trainingsobjekten entsprechen, möglichst zuverlässig klassifiziert werden.

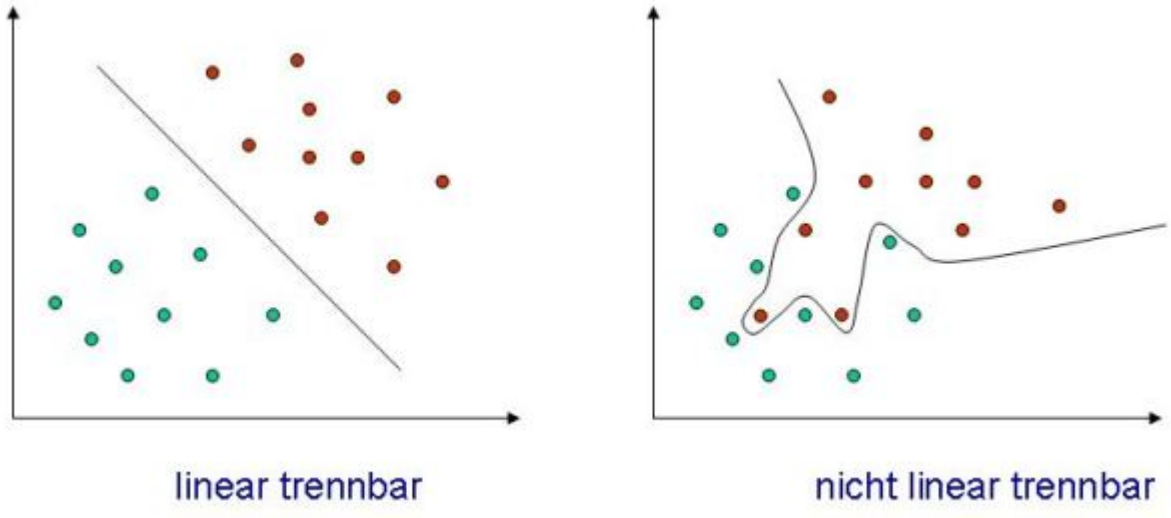


Zwei mögliche Trenngeraden
mit verschiedenen Randgrößen

Beim Einsetzen der Hyperebene ist es nicht notwendig, alle Trainingsvektoren zu beachten. Vektoren, die weiter von der Hyperebene entfernt liegen und gewissermaßen hinter einer Front anderer Vektoren "versteckt" sind, beeinflussen Lage und Position der Trennebene nicht. Die Hyperebene ist nur von den ihr am nächsten liegenden Vektoren abhängig – und auch nur diese werden benötigt, um die Ebene mathematisch exakt zu beschreiben. Diese nächstliegenden Vektoren werden nach ihrer Funktion Stützvektoren (engl. support vectors) genannt und verhalfen den Support Vector Machines zu ihrem Namen.

6.2.1 Lineare Trennbarkeit

Eine Hyperebene kann nicht "verbogen" werden, sodass eine saubere Trennung mit einer Hyperebene nur dann möglich ist, wenn die Objekte linear trennbar sind.



Diese Bedingung ist für reale Trainingsobjektmengen im Allgemeinen nicht erfüllt. Support Vector Machines verwenden im Fall nichtlinear trennbarer Daten den **Kernel-Trick**, um eine nichtlineare Klassengrenze einzuziehen.

6.2.2 Kernel-Trick

Die Idee dahinter ist, den Vektorraum in einen höherdimensionalen Raum zu überführen, wo die Objekte linear trennbar sind und dort eine Hyperebene zu definieren. In einem Raum mit genügend hoher Dimensionsanzahl – im Zweifelsfall unendlich – wird auch die verschachtelteste Vektormenge linear trennbar. In diesem höherdimensionalen Raum wird nun die trennende Hyperebene bestimmt. Bei der Rücktransformation in den niedrigerdimensionalen Raum wird die lineare Hyperebene zu einer nichtlinearen, unter Umständen sogar nicht zusammenhängenden Hyperfläche, welche die Trainingsvektoren sauber in zwei Klassen trennt.
Bei diesem Vorgang stellen sich zwei Probleme: Die Hochtransformation ist enorm rechenintensiv und die Darstellung der Trennfläche im niedrigdimensionalen Raum im Allgemeinen unwahrscheinlich komplex und damit praktisch unbrauchbar. An dieser Stelle setzt der **Kernel-Trick** an.
Verwendet man zur Beschreibung der Trennfläche geeignete Kernelfunktionen, die im Hochdimensionalen die Hyperebene beschreiben und trotzdem im Niedrigdimensionalen "gutartig" bleiben, so ist es möglich, die Hin- und Rücktransformation umzusetzen, ohne sie tatsächlich rechnerisch ausführen zu müssen. Auch hier genügt ein Teil der Vektoren, nämlich wiederum die Stützvektoren, um die Klassengrenze vollständig

zu beschreiben.
Sowohl lineare als auch nichtlineare Support Vector Machines lassen sich durch zusätzliche **Schlupfvariablen** flexibler gestalten. Die Schlupfvariablen erlauben es dem Klassifikator, einzelne Objekte falsch zu klassifizieren, "bestrafen" aber gleichzeitig jede derartige Fehleinordnung. Auf diese Weise wird zum einen Überanpassung vermieden, zum anderen wird die benötigte Anzahl an Stützvektoren gesenkt.

6.3 Anschauliches 2-dim. Beispiel für Kernel-Trick

Stellen wir uns ein einfaches und anschauliches Beispiel in $\mathbb{R}^2$ vor, bei dem die Datenpunkte im ursprünglichen Raum (x_1, x_2) nicht linear trennbar sind.
Gesucht ist dann eine Abbildung $\Phi : \mathbb{R}^2 \to \mathbb{R}^3$, so das die Datenpunkte im höherdimensionalen Raum $(x, y, z) \in \mathbb{R}^3$ linear trennbar sind.
Der Kernel-Trick ermöglicht es der SVM, die Entscheidungsgrenze in diesem höherdimensionalen Raum zu finden, ohne die zusätzlichen Merkmale z tatsächlich berechnen zu müssen:
Gegeben sind die folgenden Datenpunkte: Klasse +1: A(1,1), B(-1,1) und Klasse -1: C'(2,1), D(1,-2) und E(-2,1).
In diesem Beispiel sind die Datenpunkte nicht linear trennbar (siehe auch nachfolgende Skizze). Es gibt keine gerade Linie, die die Punkte der Klasse +1 von den Punkten der Klasse -1 perfekt trennen kann.
Jetzt wenden wir den Kernel-Trick an, um die Daten in $\mathbb{R}^3$ zu transformieren. Hier verwenden wir den polynomiellen Kernel über die folgende Transformation an:

$$\Phi : \mathbb{R}^2 \to \mathbb{R}^3 \text{ definiert durch } \Phi(x_1, x_2) = (x_1, x_2, x_1^2 + x_2^2).$$

Die transformierten Punkte A' = $\Phi(A)$ ....E' = $\Phi(A)$ werden dann berechnet als: Klasse +1: A'(1,1,2), B'(-1,1,2) und Klasse -1: C'(2,1,5), D'(1,-2,5) und E'(-2,1,5).
Die roten Punkte liegen dabei alle auf der Höhe z=2 und die blauen Punkte auf z=5.
Siehe dazu den folgenden Python Plot:

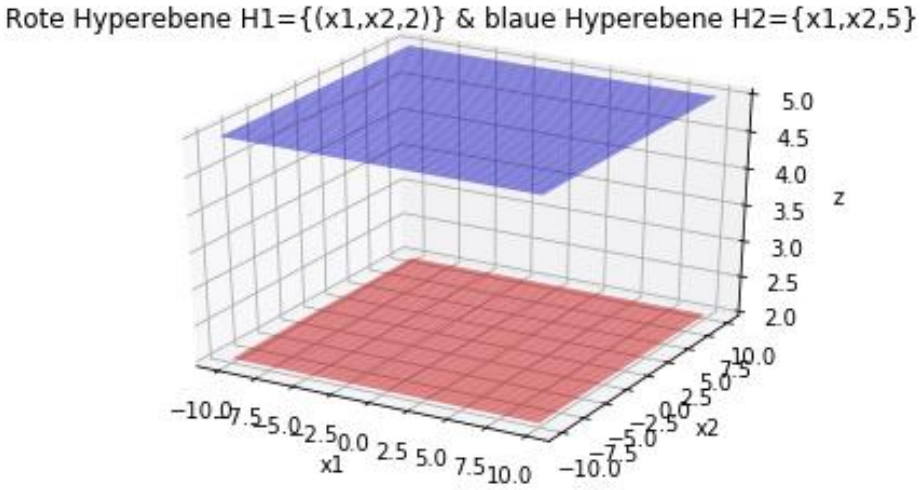


Figure 5: Hyperebenen zum Kernel-Trick Beispiel

Ein bewegtes Bild davon liegt in der Referenz [**HVö-5**]: https://github.com/HVoellinger/Mathematische-Grundlagen_von_ML/blob/main/Images/hyperebenen_animation.gif

Wie man leicht aus dem Plot sieht, lassen sich diese Punkte optimal durch eine Hyperebene $H := \{(x_1, x_2, z) | z = 3,5\}$ trennen.
Durch eine "Zurücktransformation" erhält man den **Kreis:** $x_1^2 + x_2^2 = 3,5$ mit den Radius $r = \sqrt[2]{3,5} \approx 1.871$ Damit ergibt sich die visuelle Darstellung der Klassifikation-Trennlinie als Kreislinie. Wir lassen uns diesen Kreis via einem kleinem Python Programm ausploten:

```python
import numpy as np
import matplotlib.pyplot as plt

theta = np.linspace(0, 2*np.pi, 100)
r = 1.8

x = r * np.cos(theta)
y = r * np.sin(theta)

plt.plot(x, y, color='black', label='(x_1)² + (x_2)² = 3')

points = {'A': (1, 1), 'B': (-1, 1), 'C': (2, 1), 'D': (1, -2), 'E': (-2, 1)}
colors = {'A': 'red', 'B': 'red', 'C': 'blue', 'D': 'blue', 'E': 'blue'}

for point, coords in points.items():
    plt.scatter(coords[0], coords[1], color=colors[point], label=point)
    plt.annotate(point, coords, textcoords="offset points", xytext=(-10,10), ha='center')

plt.xlabel('x_1')
plt.ylabel('x_2')
plt.title('Graph des Kreises (x_1)² + (x_2)² = 3 mit roten und blauen Punkten')
plt.grid(True)
plt.axis('equal')
#plt.legend()
plt.show()
```

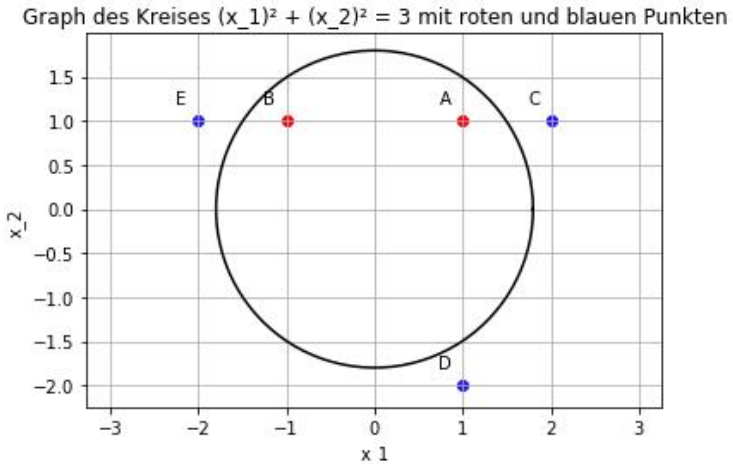


Figure 6: Python Code und Kernel-Trick Kreisbild

6.4 Übungen zum Kapitel 6

.....

6.4.1 Übung 6.1 - .....

TEXT

6.4.2 Übung 6.2 - .....

6.4.3 Übung 6.3 - Python Code zu Hyperebenen

Schreiben Sie den Python Code zur Generierung der zwei Hyperebenen im 2-dim. Beispiel zum Kernel-Trick. Animieren Sie die Grafik, so dass sie sich dreht.

7 Neuronale Faltungsnetzwerke "Convolutional Neural Networks" (CNN)

Neuronale Faltungsnetzwerke sind eine spezielle Art von künstlichen neuronalen Netzwerken, die hauptsächlich für die Verarbeitung von strukturierten Daten wie Bildern oder anderen Gitterdaten entwickelt wurden. Sie sind besonders effektiv bei der Extraktion von Merkmalen aus solchen Daten, was sie ideal für Aufgaben wie Bilderkennung und Bildklassifikation macht.
Der Name "Faltungsnetzwerke" leitet sich von der **Faltungsoperation** ab, die in diesen Netzwerken eine wichtige Rolle spielt. Diese Operation ermöglicht es, lokale Merkmale oder Muster in den Eingabedaten zu erkennen, indem sie über die Daten "geschoben" wird. Diese Merkmale werden dann auf höheren Ebenen des Netzwerks kombiniert, um komplexere Muster und Strukturen zu identifizieren.
Convolutional Neural Networks (CNNs) sind eine spezielle Art von **Deep-Learning-Modellen**. Insgesamt bezieht sich "Deep Learning" auf den Einsatz von neuronalen Netzwerken mit mehreren Schichten (daher "tief") für das Lernen von Darstellungen und Mustern in Daten. CNNs sind nur eine von vielen Architekturen im Bereich des Deep Learning.
CNNs haben in den letzten Jahren große Fortschritte erzielt und sind zum Beispiel in der Bilderkennung, Gesichtserkennung, medizinischen Bildverarbeitung und sogar in der natürlichen Sprachverarbeitung (mit Modellen wie den sogenannten "Convolutional Seq2Seq" Modellen) weit verbreitet.
***
***

Weitere Kapitel Texte
***
**** Referenz: [Math for DL] ********

7.1 Mathematische Grundlagen von CNN

Die mathematischen Grundlagen des Convolutional Neural Networks (CNN) beruhen auf Konzepten aus linearen Algebra, partiellen Ableitungen und Faltung (Convolution). Ein CNN ist eine spezielle Art von neuronalem Netzwerk, das häufig in der Bild- und Sprachverarbeitung eingesetzt wird, aufgrund seiner Fähigkeit, räumliche Strukturen in Daten zu erkennen.
Hier sind die wichtigen mathematischen Grundlagen eines Convolutional Neural Networks:

1. **Lineare Algebra:**

CNNs verwenden Matrizenoperationen, um Gewichte und Aktivierungen zu berechnen. Ein typischer Schritt in einem CNN ist die lineare Transformation, bei der Eingabedaten durch eine Gewichtsmatrix multipliziert und ein Bias addiert wird. Dies erzeugt die Aktivierungen der nächsten Schicht.

2. Faltung (Convolution):

Die Faltung ist ein zentrales Konzept in CNNs. In einem CNN werden Filter (auch Kernel genannt) verwendet, um räumliche Merkmale aus den Eingabedaten zu extrahieren. Die Faltung erfolgt durch das Verschieben des Filters über die Eingabedaten und die Berechnung des Punktprodukts zwischen dem Filter und dem überlappenden Teil der Daten. Das Ergebnis ist ein sogenanntes Aktivierungsmuster oder Feature-Map, das die erkannten Merkmale anzeigt.

3. Nichtlinearität (Aktivierungsfunktionen):

Nach der Faltung wird in den meisten Schichten des CNN eine Nichtlinearität eingeführt, um die Expressivität des Netzwerks zu erhöhen. Eine Aktivierungsfunktion wie die ReLU (Rectified Linear Unit) wird oft verwendet, um negative Werte zu eliminieren und nichtlineare Verzerrungen in den Daten zu erzeugen.

4. Pooling:

Pooling ist ein weiterer wichtiger Schritt in CNNs, um die räumliche Dimension der Daten zu reduzieren und die Invarianz gegenüber leichten Translationen zu erreichen. Typischerweise wird Max-Pooling angewendet, bei dem der maximalste Wert aus einem kleinen Ausschnitt der Daten beibehalten wird.

5. Backpropagation:

Wie bei anderen neuronalen Netzwerken werden CNNs mit dem Backpropagation-Algorithmus trainiert. Backpropagation verwendet die Kettenregel der partiellen Ableitungen, um die Gewichte des Netzwerks so anzupassen, dass der Fehler zwischen den vorhergesagten und tatsächlichen Ausgaben minimiert wird.

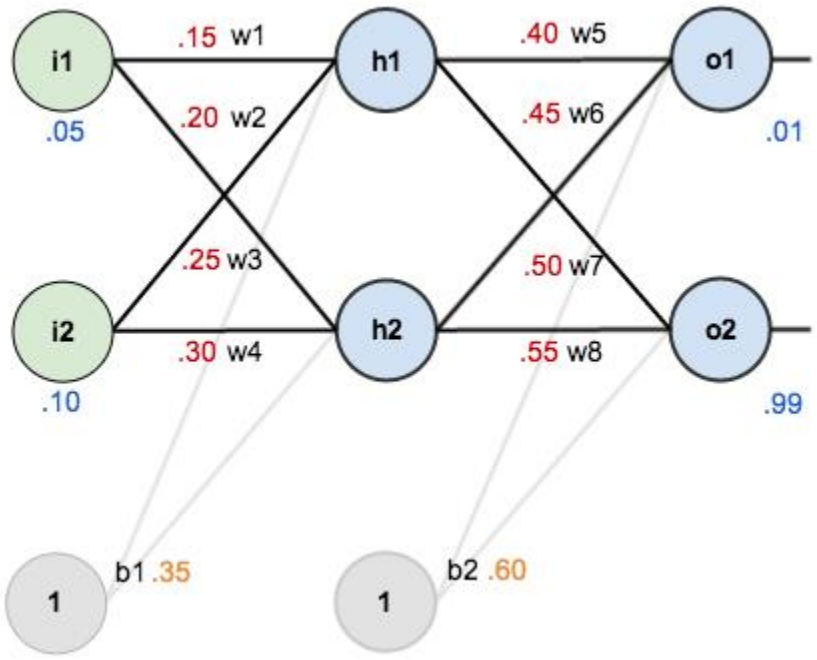
6. Mehrschichtige Struktur:

CNNs bestehen aus mehreren Schichten, darunter Eingabeschicht, Faltungsschichten, Aktivierungsschichten, Pooling-Schichten und einer Ausgabeschicht. Die tieferen Schichten sind in der Lage, einfache Merkmale zu lernen, während die höheren Schichten komplexere Merkmale und Kombinationen von Merkmalen erfassen können.

Diese mathematischen Grundlagen ermöglichen es Convolutional Neural Networks, Merkmale aus Eingabedaten zu extrahieren und komplexe Muster zu lernen, was zu einer effektiven Verarbeitung von Bildern, Videos, Sprache und anderen räumlichen Daten führt.

7.2 Rückwärtspropagierung für ein anschauliches Beispiel

Die Rückwärtspropagierung ("Backpropagation") erfolgt um die Gewichte zu aktualisieren und den Fehler zu minimieren. Durch diese Schritte werden die Gewichte des neuronalen Netzwerks iterativ angepasst, um den **Fehler zwischen den vorhergesagten und tatsächlichen Ausgaben zu minimieren** und bessere Vorhersagen zu machen. Der Vorgang wird für jeden Eingabedatensatz und seine zugehörige Ausgabe wiederholt, bis das Netzwerk trainiert ist.
Lassen Sie uns einfaches Beispiel für den Backpropagation-Prozess für ein sehr einfaches neuronales Netz mit nur einer versteckten Schicht ("Hidden Layer") durchgehen. Insgesamt haben ein neuronales Netzwerk mit zwei Eingängen, zwei versteckten Neuronen und zwei Ausgangsneuronen. Zusätzlich werden die versteckten Neuronen und die Ausgangsneuronen einen Bias (aka: Verzerrungen") enthalten. An diesem 3-Schichten CNN können wir den kompletten Backpropagation Prozess beispielhaft berechnen:
Eingabeschicht ("Input Layer" mit 2 "Input Nodes" (i_1, i_2)**) ⇒ Verarbeitunsschicht ("Hidden Layer" mit 2 "Hidden Nodes "**(h_1, h_2) **) ⇒ Ausgabeschicht ("Output Layer" mit 2 "Output Nodes"** (o_1, o_2)**).**
Um ein paar Zahlen zu haben, mit denen man arbeiten kann, sind hier die anfänglichen Gewichte, die Bias("Verzerrungen") und die Trainingsinputs/-outputs angegeben:



7.2.1 Berechnungen zum Vorwärtsdurchlauf ("Forward Pass")

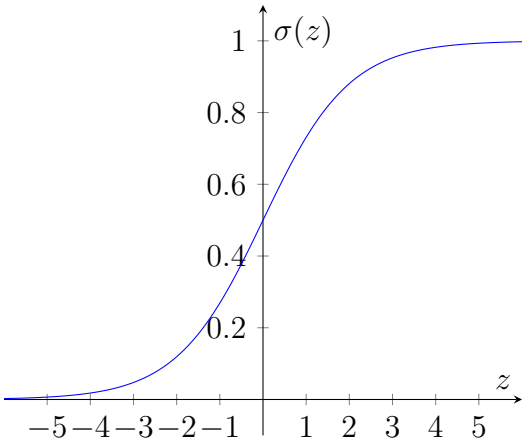
Der Vorwärtsdurchlauf (englisch: forward pass) bei Convolutional Neural Networks (CNNs) ist der Prozess, bei dem die Eingabedaten durch das Netzwerk propagiert werden, um eine Vorhersage oder Ausgabe zu generieren. Während des Vorwärtsdurchlaufs werden die Daten Schicht für Schicht verarbeitet, wobei die Gewichtungen, Aktivierungen und Pooling-Operationen angewendet werden, um schließlich die Ausgabe zu erhalten.
Berechnen wir zunächst die gewichtete Summe (h_1, h_2) und glätten dann den Wertebereich durch die **logistische Funktion** $\sigma(z) := \frac{1}{1+e^{-z}}$) (siehe auch die Anmerkung dazu weiter unten im Text) um die Ausgaben der versteckten Schicht zu erhalten. Wir bezeichne diese mit Großbuchstaben (H_1, H_2):

$$
\begin{aligned}
h_1 &= (i_1 \cdot w_1) + (i_2 \cdot w_2) + b_1 = (0.05 \cdot 0.15) + (0.1 \cdot 0.25) + 0.35 = 0.3775 \\
h_2 &= (i_1 \cdot w_3) + (i_2 \cdot w_4) + b_1 = (0.05 \cdot 0.2) + (0.1 \cdot 0.3) + 0.35 = 0.3925 \\
H_1 &= \sigma(h_1) = \frac{1}{1 + e^{-0.3775}} \approx 0.5933 \\
H_2 &= \sigma(h_2) = \frac{1}{1 + e^{-0.3925}} \approx 0.5969
\end{aligned}
$$

Wir wiederholen diesen Vorgang für die Neuronen der Ausgabeschicht und verwenden die Ausgaben der Neuronen der versteckten Schicht als Eingaben. Wir bezeichnen die Ergebnisse Ausgabeschicht mit Großbuchstaben (O_1, O_2):(i_1, i_2)

$$
\begin{aligned}
o_1 &= (a_1 \cdot w_5) + (a_2 \cdot w_6) + b_2 = (0.5933 \cdot 0.4) + (0.5969 \cdot 0.45) + 0.6 \approx 1.1059 \\
O_1 &= \sigma(o_1) \approx 0.7514 \\
o_2 &= (a_1 \cdot w_7) + (a_2 \cdot w_8) + b_2 = (0.5933 \cdot 0.5) + (0.5969 \cdot 0.55) + 0.6 \approx 1.2249 \\
O_2 &= \sigma(o_2) \approx 0.7729
\end{aligned}
$$

Anmerkung zur "logistischen Funktion":
Die Sigmoid-Funktion $\sigma(z)$ glättet den Wertebereich und ist aufgrund ihres kontinuierlichen Verlaufs gut für die Berechnung von Gradienten bei der Rückwärtspropagation (Backpropagation) geeignet. Sie wird in CNNs und anderen neuronalen Netzwerken häufig verwendet, um die Aktivierungsstärke eines Neurons zu modulieren. Die Sigmoid-Funktion ist eine nichtlineare Funktion, die eine kontinuierliche Ausgabe zwischen 0 und 1 erzeugt. Der folgende Graph zeigt den S-förmigen Verlauf der logistischen Funktion $\sigma(z)$ über den Bereich von z=-6 bis z=+6:



Bemerkung: Die Sigmoid-Aktivierung wurde früher häufiger verwendet, hat jedoch in einigen Situationen Probleme wie das Verschwinden von Gradienten verursacht. Aus diesem Grund verwenden moderne CNN-Architekturen oft ReLU (Rectified Linear Unit) und ihre Varianten als Aktivierungsfunktionen, da sie das Problem des Verschwindens von Gradienten verringern und zur schnelleren Konvergenz des Lernprozesses beitragen können.

7.2.2 Berechnungen des Fehlers/Verlustes "Error/Loss":

Wir können nun den Fehler für jedes Ausgangsneuron mit Hilfe der quadratischen Fehlerfunktion berechnen und sie addieren, um den Gesamtfehler zu erhalten.
Anmerkung: Die $\frac{1}{2}$ ist enthalten, damit der Exponent beim späteren Differenzieren aufgehoben wird. Das Ergebnis wird schließlich ohnehin mit einer Lernrate η multipliziert, so dass es keine Rolle spielt, dass wir hier eine Konstante einführen.
Bezeichne den Fehler ("Error") pro Ausgabeneuron wieder mit Großbuchstaben (E_1, E_2) und den Gesamtfehler als E, so ergibt sich:

$$E_1 = \frac{1}{2} \cdot (b_1 - y_{\text{target1}})^2 \approx \frac{1}{2} \cdot (0.7514 - 0.01)^2 \approx 0.2748$$
$$E_2 = \frac{1}{2} \cdot (b_2 - y_{\text{target2}})^2 \approx \frac{1}{2} \cdot (0.7729 - 0.99)^2 \approx 0.0236$$
$$E = E_1 + E_2 \approx 0.2748 + 0.0236 \approx 0.2983$$

7.2.3 Rückwärtsdurchlauf "Backward Pass":

Unser Ziel bei der Backpropagation ist es, die einzelnen Gewichte im Netz so zu aktualisieren, dass die tatsächliche Ausgabe näher an der Zielausgabe liegt, wodurch der Fehler für jedes Ausgangsneuron und das Netz als Ganzes minimiert wird.
In der nachfolgenden Grafik ist das Vorgehen schematisch dargestellt. Insbesondere kommt die **Kettenregel von "geschachtelten" Funktionen** zur Anwendung (siehe Vorlesung "Analysis I + II"):

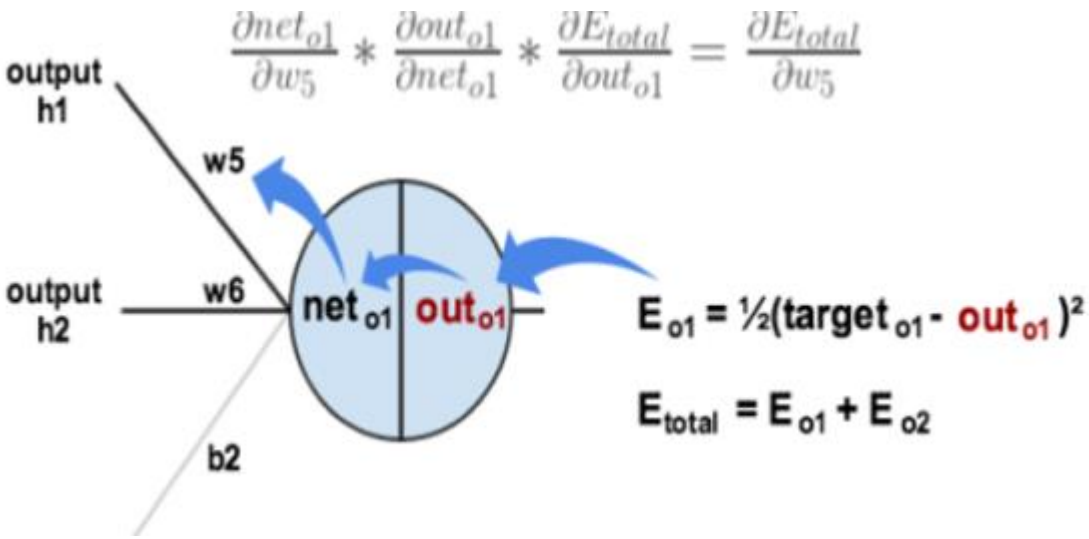


Figure 7: Backpropagation-Rückwärtslauf

******* ab hier bis Ende subsection ist dass noch anzupassen *******

$$\frac{\partial \text{loss}}{\partial a_3} = a_3 - y_{\text{target}} \approx 0.6685 - 1 \approx -0.3315$$
$$\frac{\partial a_3}{\partial z_3} = a_3 \cdot (1 - a_3) \approx 0.6685 \cdot (1 - 0.6685) \approx 0.2241$$
$$\frac{\partial \text{loss}}{\partial w_5} = \frac{\partial \text{loss}}{\partial a_3} \cdot \frac{\partial a_3}{\partial z_3} \cdot a_1 \approx -0.3315 \cdot 0.2241 \cdot 0.6682 \approx -0.0664$$
$$\frac{\partial \text{loss}}{\partial w_6} = \frac{\partial \text{loss}}{\partial a_3} \cdot \frac{\partial a_3}{\partial z_3} \cdot a_2 \approx -0.3315 \cdot 0.2241 \cdot 0.8909 \approx -0.0497$$
$$\frac{\partial \text{loss}}{\partial b_3} = \frac{\partial \text{loss}}{\partial a_3} \cdot \frac{\partial a_3}{\partial z_3} \approx -0.3315 \cdot 0.2241 \approx -0.0743$$

Wir multiplizieren die Gewichte w_i und die Biases b_i mit einer Lernrate (e.g., $\eta = 0.1$):

$$w_5' = w_5 - \eta \cdot \frac{\partial \text{loss}}{\partial w_5} \approx 0.2 - 0.1 \cdot (-0.0664) \approx 0.3066$$
$$w_6' = w_6 - \eta \cdot \frac{\partial \text{loss}}{\partial w_6} \approx 0.4 - 0.1 \cdot (-0.0497) \approx 0.7050$$
$$b_3' = b_3 - \eta \cdot \frac{\partial \text{loss}}{\partial b_3} \approx 0.1 - 0.1 \cdot (-0.0743) \approx 0.2074$$

In ähnlicher Weise werden die Gradienten berechnet und die Gewichte und Verzerrungen für die Verbindungen zwischen der Eingabeschicht und der verborgenen Schicht aktualisiert.

Dieser Prozess des Vorwärtsdurchlaufs, der Verlustberechnung und des Rückwärtsdurchlaufs wird iterativ über den gesamten Trainingsdatensatz wiederholt, um die Gewichte und Verzerrungen zu aktualisieren, bis das Netzwerk lernt, genaue Vorhersagen zu treffen.

Weitere Details unter folgenden **Referenzen:** [BackP-Exam] und [BackP-Scikit]

7.3 Übungen zum Kapitel 7

.....

7.3.1 Übung 7.1 - CNN Architekur

TEXT

7.3.2 Übung 7.2 - Beispiel Backpropagation

.....

7.3.3 Übung 7.3 - .....

8 Mathematik und grosse Sprachmodelle "Large Language Models" (LLMs)

8.1 Einsatz von Mathematik in LLMs

Insgesamt beruht die Funktionsweise von LLMs auf einer Kombination verschiedener mathematischer Konzepte und Techniken. Die Modelle werden trainiert, um Muster in großen Textkorpora zu erkennen und Texte zu generieren, die auf den erlernten statistischen Zusammenhängen basieren.
Mathematik ermöglicht es, diese komplexen Modelle zu verstehen, zu entwickeln und weiterzuentwickeln.
Mathematik spielt eine zentrale Rolle bei der Funktionsweise von großen Sprachmodellen wie den "Large Language Models" (LLMs) wie GPT-3.5.
Hier sind einige Schlüsselaspekte, wie Mathematik in Bezug auf LLMs relevant ist:

1. Lineare Algebra: LLMs verarbeiten Textdaten in Form von Matrizen. Die zugrunde liegenden Berechnungen involvieren Operationen wie Matrixmultiplikation, Addition, Subtraktion und Skalierung. Lineare Algebra ist daher entscheidend, um die Transformationen und Berechnungen in den Modellen zu verstehen.

2. Tensorrechnung: LLMs verwenden oft Tensoren, die eine Erweiterung von Matrizen auf höhere Dimensionen darstellen. Die meisten Daten in LLMs werden in Form von Tensoren dargestellt, und die Manipulation dieser Tensoren durch mathematische Operationen ist entscheidend für die Generierung von Text.

3. Wahrscheinlichkeit und Statistik: Wahrscheinlichkeitstheorie und Statistik sind unverzichtbar, um die Unsicherheit in Textdaten zu modellieren. LLMs verwenden oft Methoden wie Bayes'sche Wahrscheinlichkeit und Maximum-Likelihood-Schätzungen, um die Wahrscheinlichkeit von Wörtern oder Sätzen zu bestimmen.

4. Optimierung: Die Trainingsphasen von LLMs sind Optimierungsprobleme, bei denen das Modell so angepasst wird, dass es die beste Leistung erzielt. Hier kommen mathematische Optimierungsverfahren wie Gradientenabstieg zum Einsatz, um die Modellparameter zu optimieren.

5. Neuronale Netzwerke: LLMs verwenden tiefe neuronale Netzwerke, um komplexe Muster in den Daten zu erfassen. Die Mathematik hinter neuronalen Net-

zwerken umfasst Aktivierungsfunktionen, Gewichtungen, Verknüpfungen und Schichten, die zusammenarbeiten, um Eingabedaten in sinnvolle Ausgaben zu transformieren.

6. Informationstheorie: Die Theorie der Information ist relevant, um zu verstehen, wie effektiv Informationen in einem LLM codiert und übertragen werden. Konzepte wie Entropie, Kullback-Leibler-Divergenz und Informationsgewinn sind hier von Bedeutung.

7. Convolutions und Pooling: In manchen LLM-Architekturen kommen Faltungen und Pooling-Operationen zum Einsatz, um räumliche Muster in den Daten zu erfassen. Diese Operationen basieren auf mathematischen Konzepten aus der Signalverarbeitung.

8. Aufmerksamkeitsmechanismen: Aufmerksamkeitsmechanismen sind zentral für LLMs. Mathematische Modelle von Aufmerksamkeit ermöglichen es dem Modell, relevante Teile des Textes zu identifizieren und sich darauf zu konzentrieren.

9. Rekursive und iterative Prozesse: In einigen Textgenerationsaufgaben verwenden LLMs rekursive oder iterative Prozesse. Mathematische Konzepte wie Rekursion und Iteration sind hier von Bedeutung.

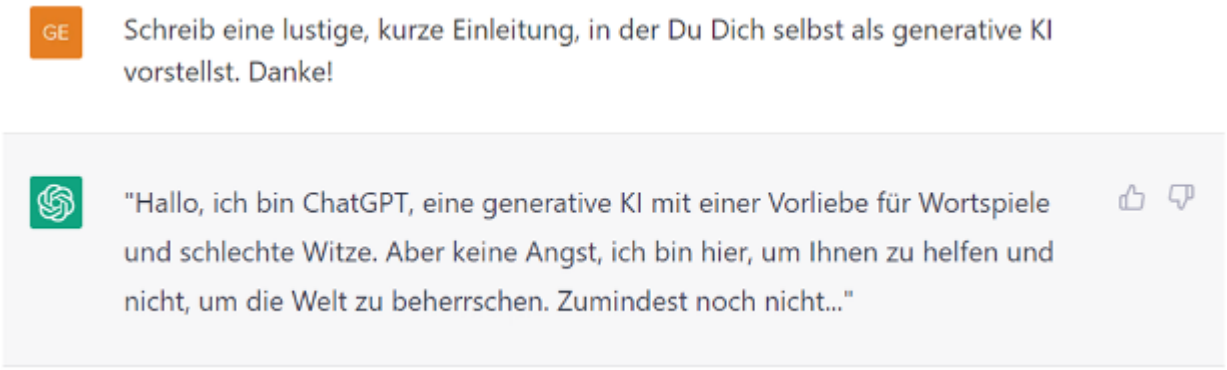
.....

8.2 ChatGPT

Es schreibt Gedichte, Raptexte, es besteht Zertifizierungen von Softwareherstellern, absolviert MBA-Prüfungen, kann Referate und Reiseberichte schreiben, die Stringtheorie erklären und Computerspiele programmieren. Es schreibt Aufsätze basierend auf Fakten, erfindet manches vollkommen frei und halluziniert gelegentlich.
Beim Weltwirtschaftsforum in Davos sprechen führende CEOs leidenschaftlich darüber, und Google hat ernsthaft Angst davor.

Die Rede ist natürlich von ChatGPT, einem Dialogsystem, das die Errungenschaften der generativen Künstlichen Intelligenz auf einfachste Weise der breiten Öffentlichkeit zugänglich macht und damit wie ein Schaufenster in die Zukunft wirkt. Mit ChatGPT wird das bereits seit längerem existierende GPT-3 (bzw. genau genommen GPT-3.5) plötzlich für jedermann nutzbar und erfährt ein gewaltiges Me-

dienecho.



TEXT

8.3 Weitere Beispiele

.....

8.4 Übungen zum Kapitel 8

.....

8.4.1 Übung 8.1 - .....

TEXT

8.4.2 Übung 8.2 - .....

.....

9 Anhänge

In den Anhängen werden einige weitere Verfahren des Maschinellen Lernens der Vollständigkeit halber erwähnt - ohne zu sehr in die inhaltlich Tiefe zu gehen. Zukünftig können jedoch diesem Themen in einer erweiterten Version des Skriptes durchaus nochmals aufgegriffen werden.

9.1 Empfehlungssysteme "Recommender Systems" (LinAlgebra)

Die mathematischen Grundlagen von Recommender Systems (auch als Empfehlungssysteme bezeichnet) hängen von der spezifischen Art des Empfehlungssystems ab. Es gibt verschiedene Ansätze und Modelle, die in Recommender Systems verwendet werden, um Empfehlungen für Benutzer zu generieren.
Hier sind einige der wichtigsten mathematischen Grundlagen:

1. Collaborative Filtering (Kollaborative Filterung): Bei der kollaborativen Filterung werden Empfehlungen basierend auf der Ähnlichkeit zwischen Benutzern oder Objekten generiert. Diese Ähnlichkeit kann durch Berechnung von Ähnlichkeitsmetriken wie der Kosinusähnlichkeit oder der Pearson-Korrelation zwischen Benutzern oder Objekten ermittelt werden.

2. Matrix Factorization (Matrizenfaktorisierung): Dies ist eine Technik, bei der die Bewertungen von Benutzern für Objekte in eine niedrigdimensionale latente Raum eingebettet werden. Diese Einbettung ermöglicht es, die Bewertungen als Produkte von latenten Benutzer- und Objektvektoren darzustellen. Die Matrix wird in zwei niedrigdimensionale Matrizen (Benutzermatrix und Objektmatrix) zerlegt, und Empfehlungen werden basierend auf den latenten Vektoren berechnet.

3. Content-Based Filtering (Inhaltsbasierte Filterung): Hier werden Empfehlungen basierend auf den Merkmalen (Eigenschaften) der Objekte und den Präferenzen der Benutzer generiert. Ähnlichkeiten zwischen Objekten werden durch Ähnlichkeitsmaße wie den Kosinusähnlichkeitswert der Merkmale berechnet.
4. Singular Value Decomposition (SVD): SVD ist eine Technik der linearen Algebra, die zur Dimensionsreduktion und Matrizenfaktorisierung verwendet wird. Im Zusammenhang mit Recommender Systems wird SVD häufig in der Matrixfaktorisierung für kollaborative Filterung angewendet.

5. ** Deep Learning **: In den letzten Jahren haben Recommender Systems auch von den Fortschritten im Bereich des Deep Learning profitiert. Hier werden neuronale Netzwerke verwendet, um komplexe Muster in den Daten zu lernen und präzisere Empfehlungen zu generieren.

6. Evaluation Metrics (Auswertungsmetriken): Um die Leistung von Recommender Systems zu bewerten, werden verschiedene Auswertungsmetriken verwendet, wie zum Beispiel Genauigkeit, Trefferquote, Mittlere absolute Fehler (MAE) oder Mittlere quadratische Abweichung (MSE).

Zusammenfassung: Je nach Art des Recommender Systems können weitere mathematische Konzepte und Modelle involviert sein. Die Grundlagen der Mathematik und Statistik spielen jedoch eine entscheidende Rolle bei der Modellierung, Berechnung und Bewertung von Empfehlungssystemen, um nützliche und relevante Empfehlungen für die Benutzer zu generieren.

9.2 Regularisierungen und Lineare Algebra (LA)

9.2.1 Regularisierung in maschinellem Lernen

Regularisierung ist eine Technik im maschinellen Lernen, die verwendet wird, um Overfitting zu vermeiden. Overfitting tritt auf, wenn ein Modell zu stark auf die Trainingsdaten passt und dadurch auf neuen, bisher ungesehenen Daten schlecht abschneidet. Regularisierung fügt eine zusätzliche Bedingung zur Verlustfunktion hinzu, um die Gewichtungen im Modell zu begrenzen oder einzuschränken.

Es gibt zwei gängige Arten von Regularisierung:

1. **L2-Regularisierung (Ridge-Regularisierung)**: Hierbei wird der Verlustfunktion ein Ausdruck hinzugefügt, der proportional zur Quadratsumme der Gewichtungen ist. Dies zwingt das Modell dazu, die Gewichtungen kleiner zu halten.

2. **L1-Regularisierung (Lasso-Regularisierung)**: Bei dieser Methode wird der Verlustfunktion ein Ausdruck hinzugefügt, der proportional zur Summe der absoluten Werte der Gewichtungen ist. Dadurch neigen viele der Gewichtungen im Modell dazu, genau null zu werden, was zu einer Art von Feature-Auswahl führt.

9.2.2 Lineare Algebra in Bezug auf Regularisierung

Lineare Algebra ist ein wichtiger Teilbereich der Mathematik, der sich mit Vektoren, Matrizen und linearen Gleichungssystemen befasst. Sie spielt eine bedeutende Rolle in der Vorstellung und Berechnung von Regularisierungstechniken. Hier sind einige Punkte, wie Lineare Algebra mit Regularisierung zusammenhängt:

1. **Gewichtungen als Vektoren:** In maschinellen Lernalgorithmen werden die Gewichtungen oft als Vektoren dargestellt. Diese Vektoren werden mithilfe von Linearer Algebra manipuliert, um die Regularisierungsbedingungen zu erfüllen.

2. **Matrixformulierung:** Viele Regularisierungstechniken können in der Matrixformulierung ausgedrückt werden. Zum Beispiel kann die L2-Regularisierung als Hinzufügen eines Ausdrucks zur Verlustfunktion betrachtet werden, der mit der Quadratwurzel der Gewichtungsmatrix multipliziert wird.

3. **Optimierung:** Bei der Anwendung von Regularisierungstechniken wird häufig eine Verlustfunktion optimiert, um die besten Gewichtungen zu finden. Lineare Al-

gebra spielt eine Rolle bei der Ableitung und Lösung dieser Optimierungsprobleme.

4. **Eigenschaften von Matrizen:** In einigen Fällen können Eigenschaften von Matrizen genutzt werden, um Regularisierungsverfahren zu analysieren und zu verstehen.

Insgesamt ist die **Lineare Algebra** ein unverzichtbares Werkzeug, wenn es darum geht, Regularisierung in maschinellem Lernen zu verstehen, zu implementieren und anzuwenden. Sie ermöglicht es, die mathematischen Grundlagen hinter diesen Techniken zu verstehen und effektiv mit komplexen Modellen zu arbeiten.

9.3 Hauptkomponentenanalyse "Principal Component Analysis" (PCA)

Die Hauptkomponentenanalyse (Englisch: "Principal Component Analysis") kurz: PCA) ist auch als Hauptachsentransformation bekannt. Das PCA ein Verfahren der multivariaten Statistik.
Sie strukturiert umfangreiche Datensätze durch Benutzung der Eigenvektoren der Kovarianzmatrix. Dadurch können Datensätze vereinfacht und veranschaulicht werden, indem eine Vielzahl statistischer Variablen durch eine geringere Zahl möglichst aussagekräftiger Linearkombinationen (die Hauptkomponenten) genähert wird.
Speziell in der Bildverarbeitung wird die Hauptkomponentenanalyse, auch *Karhunen-Loève-Transformation* genannt, benutzt. Sie ist von der Faktoren-Analyse zu unterscheiden, mit der sie formale Ähnlichkeit hat und in der sie als Näherungsmethode zur Faktorenextraktion verwendet werden kann (der Unterschied der beiden Verfahren kann in Wikipedia nachgelesen werden).

Das Bild unten zeigt die **Hauptkomponentenanalyse als Faktorenanalyse**.

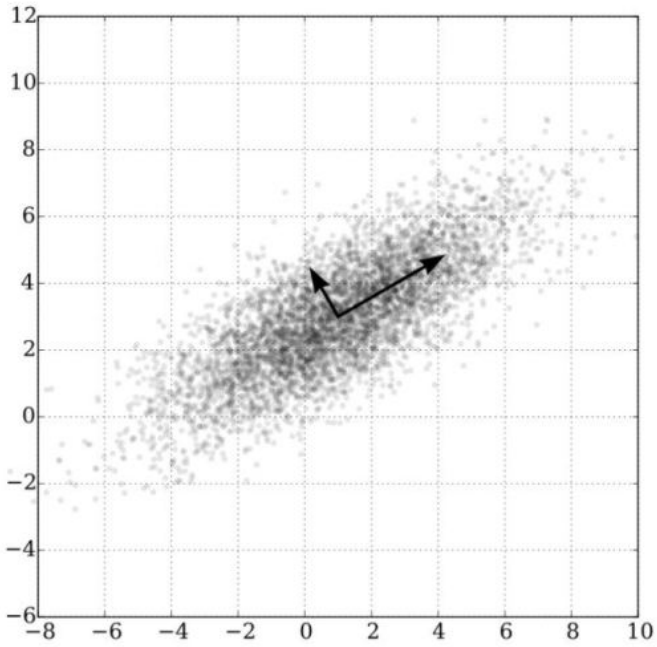


Figure 8: Schematische Darstellung der Hauptkomponentenanalyse

Zwei Hauptkomponenten einer zweidimensionalen Normalverteilung mit Mittelwert

(1,3) und Standardabweichung circa 3 in (0.866, 0.5)-Richtung und 1 in die dazu orthogonale Richtung. Die Vektoren sind die Eigenvektoren der Kovarianzmatrix und haben als Länge die Wurzel des zugehörigen Eigenwertes. Sie sind so verschoben, dass sie am Mittelwert ansetzen.

Es gibt verschiedene **Verallgemeinerungen** der Hauptkomponentenanalyse, z. B. die Hauptkurven ("Principal Curves"), die Hauptflächen ("Principal Surfaces"), t-verteilte stochastische Nachbarschaftseinbettung ("t-distributed stochastic neighbor embedding") oder die kernbasierte Hauptkomponentenanalyse ("kernel Principal Component Analysis", kurz: kernel PCA).

9.4 Singulärwertzerlegung "Singular Value Decomposition" (SVD)

Eine **Singulärwertzerlegung** (engl. Singular Value Decomposition; abgekürzt SWZ oder SVD) einer Matrix bezeichnet deren Darstellung als Produkt dreier spezieller Matrizen. Daraus kann man die Singulärwerte der Matrix ablesen. Diese charakterisieren, ähnlich den Eigenwerten, Eigenschaften der Matrix. Singulärwertzerlegungen existieren für jede Matrix – auch für nichtquadratische Matrizen.
Singulärwertzerlegung am Beispiel einer zweidimensionalen, reellen Scherung M: Diese Transformation verzerrt den blauen Einheitskreis oben links zur Ellipse rechts oben im Bild. M kann zerlegt werden in zwei Drehungen U und V* sowie eine Dehnung/Stauchung Σ entlang der Koordinatenachsen. Die Singulärwerte $\sigma 1$ und $\sigma 2$ sind die Längen der großen bzw. kleinen Halbachse der Ellipse.

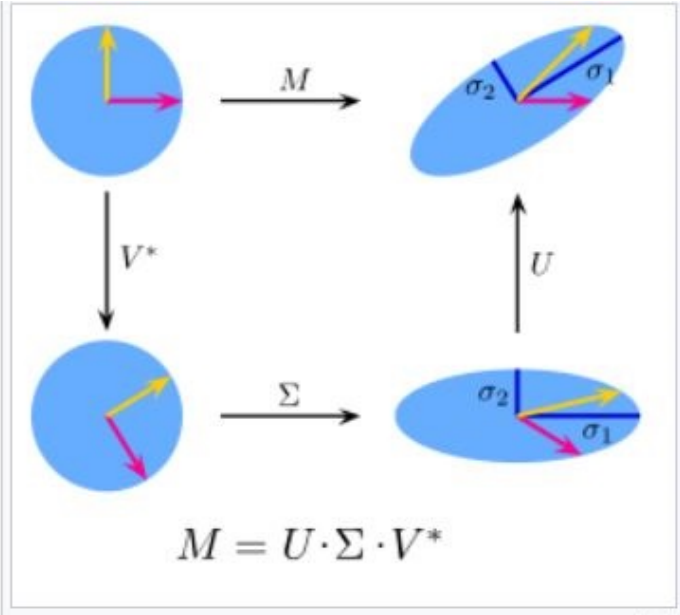


Figure 9: Schematische Darstellung des Singulärwertzerlegung

9.5 Mathematische Verfahren im NLP,i.e. "Latent Semantic Analysis/Indexing" (LSI)

Latent Semantic Indexing (kurz *LSI*) ist ein (nicht mehr patentgeschütztes Verfahren des Information Retrieval, das 1990 zuerst von Deerwester et al.erwähnt wurde.

Verfahren wie das LSI sind insbesondere für die Suche auf großen Datenmengen wie dem Internet von Interesse. Das Ziel von LSI ist es, Hauptkomponenten von Dokumenten zu finden. Diese Hauptkomponenten (Konzepte) kann man sich als generelle Begriffe vorstellen.
So ist Pferd zum Beispiel ein Konzept, das Begriffe wie Mähre, Klepper oder Gaul umfasst. Somit ist dieses Verfahren zum Beispiel dazu geeignet, aus sehr vielen Dokumenten (wie sie sich beispielsweise im Internet finden lassen), diejenigen herauszufinden, die sich thematisch mit 'Autos' befassen, auch wenn in ihnen das Wort Auto nicht explizit vorkommt.
Des Weiteren kann LSI dabei helfen, Artikel, in denen es wirklich um Autos geht, von denen zu unterscheiden, in denen nur das Wort Auto erwähnt wird (wie zum Beispiel bei Seiten, auf denen ein Auto als Gewinn angepriesen wird.
Die **Singulärwertzerlegung** (siehe Kapitel 9.4) ist der Kern der "Latent Semantic Analysis", eines Verfahrens des Information Retrieval, das hilft, in großen Textkollektionen latente Konzepte aufzudecken, anhand derer dann z. B. unterschiedlich bezeichnete Informationen zum gleichen Thema gefunden werden können.

10 Lösungen und Lösungshinweise zu den Übungen

Bei vielen Aufgaben werden Lösungshinweise gegeben mit den die Übungen dann selbstständig gelöst werden können. Bei einfachen Rechenaufgaben, deren Lösungsweg klar vorgegeben ist, wird lediglich das Ergebnis der Rechnung genannt.
***

10.1 Lösungshinweise zu Übungen Kapitel 3

10.1.1 Lösungshinweise zu Übung 3.1

.....

10.1.2 Lösungshinweise zu Übung 3.2

10.2 Lösungshinweise zu Übungen Kapitel 4

10.2.1 Lösungshinweise zu Übung 4.1

.....

10.2.2 Lösungshinweise zu Übung 4.2

.....

10.2.3 Lösungshinweise zu Übung 4.3

Für eine Lösung vergleiche:

https://github.com/HVoellinger/Lecture-Notes-to-ML-WS2020/blob/master/LR-Calculation_of_Coeff.xlsx

10.3 Lösungshinweise zu Übungen Kapitel 5

10.3.1 Lösungshinweise zu Übung 5.1

- Sports:

  P(Hermann plays a TT match | Sports) =

  $$P(Hermann|Sports) * P(plays|Sport) * P(a|Sports) * P(TT|Sports) * P(match|Sports) * P(Sports)$$

  $$= \frac{1}{29} * \frac{1}{29} * \frac{4}{29} * \frac{1}{29} * \frac{2}{29} * \frac{4}{6} = 2{,}6002\text{e-}7$$

- Not Sports:

  P(Hermann plays a TT match | Not Sports) =

  $$P(Hermann|Not\ Sports) * P(plays|Not\ Sport) * P(a|Not\ Sports) * P(TT|Not\ Sports) * P(match|Not\ Sports) * P(Not\ Sports)$$

  $$= \frac{1}{23} * \frac{1}{23} * \frac{2}{23} * \frac{1}{23} * \frac{1}{23} * \frac{2}{6} = 1{,}0358\text{e-}7$$

  ➔ It will be classified as a Sports-sentence

Zusatzfrage: Das Ergebnis dreht sich rum wenn Sie als Zielsatz **"Hermann plays a very clean game"** eingeben.

10.3.2 Lösungshinweise zu Übung 5.2

....

Die letzten Python Codeblöcke sehen Sie im folg Screenshot:

In [17]:
```
# multiplying the probabilities of each word
new_sport = list(prob_new_sport)
sport_multiply_result = 1
for i in range(0, len(new_sport)):
    sport_multiply_result *= new_sport[i]

# multiplying the result with the ratio of sports senteces to the total amount of sentences (here its 4/6)
sport_multiply_result *= ( len(tdm_sport) / len(rows) )

# multiplying the probabilities of each word
new_not_sport = list(prob_new_not_sport)
not_sport_multiply_result = 1
for i in range(0, len(new_not_sport)):
    not_sport_multiply_result *= new_not_sport[i]

# multiplying the result with the ratio of sports senteces to the total amount of sentences (here its 2/6)
not_sport_multiply_result *= ( len(tdm_not_sport) / len(rows) )


```

In [18]:
```
# comparing whats more likely

print(f'The probability of the sentence "{new_sentence}":\nSport vs not sport\n{sport_multiply_result} vs {not_sport_multipl

if not_sport_multiply_result < sport_multiply_result:
    print('Verdict: It\'s probably a sports sentence!')
else:
    print('Verdict: It\'s probably not a sport sentence!')
```

```
The probability of the sentence "Hermann plays a TT match":
Sport vs not sport
2.6002118815154297e-07 vs 1.0357848652047699e-07


Verdict: It's probably a sports sentence!
```

Weitere Hilfe unter [**HVö-5**] https://github.com/HVoellinger/Lecture-Notes-to-ML-WS2020

10.4 Lösungshinweise zu Übungen Kapitel 6

10.4.1 Lösungshinweise zu Übung 6.1

.....

10.4.2 Lösungshinweise zu Übung 6.2

10.4.3 Lösungshinweise zu Übung 6.3

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.animation import FuncAnimation

# Import libary time to check execution date+time
import time
# print the date & time of the notebook
print('**************************************************************************')
print("Actual date & time of the notebook:",time.strftime("%d.%m.%Y %H:%M:%S"))
print('**************************************************************************')

```

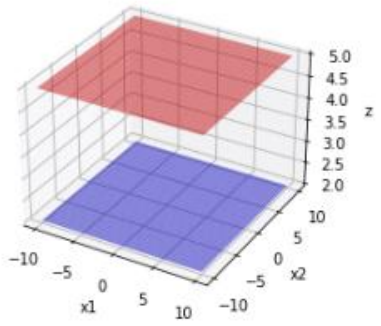
```
**************************************************************************
Actual date & time of the notebook: 25.08.2023 14:41:36
**************************************************************************
```

In [2]:

```
# Definition der Hyperebenen
def hyperplane1(x1, x2):
    return 5 * np.ones_like(x1)

def hyperplane2(x1, x2):
    return 2 * np.ones_like(x1)

# Erzeugung der Datenpunkte für den Plot
x1 = np.linspace(-10, 10, 100)
x2 = np.linspace(-10, 10, 100)
x1, x2 = np.meshgrid(x1, x2)
z1 = hyperplane1(x1, x2)
z2 = hyperplane2(x1, x2)

# Erstellen des 3D-Plots
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Plotten der Hyperebenen
ax.plot_surface(x1, x2, z1, color='r', alpha=0.5, label='H1: z=5')
ax.plot_surface(x1, x2, z2, color='b', alpha=0.5, label='H2: z=2')

# Achsenbeschriftung
ax.set_xlabel('x1')
ax.set_ylabel('x2')
ax.set_zlabel('z')

# Legende
# ax.legend()


```

Figure 10: Codeblock(1+2) zum Kernel-Trick Beispiel

Out[2]: Text(0.5, 0, 'z')



In [3]:

```
# Animieren der Ansicht (Drehen)
def animate(angle):
    ax.view_init(elev=20, azim=angle)

# Erstellen der Animation
ani = FuncAnimation(fig, animate, frames=np.arange(0, 360, 2), interval=50)

# Animation als GIF speichern
ani.save('hyperebenen_animation.gif', writer='pillow')

# Anzeigen der Animation
plt.show()

# print current date and time

print("Date & Time:",time.strftime("%d.%m.%Y  %H:%M:%S"))
# end of import test
print ("*** End of Hyperplan Example ***")
```

```
Date & Time: 25.08.2023  14:43:45
*** End of Hyperplan Example ***
```

Figure 11: Codeblock(3) zum Kernel-Trick Beispiel

10.5 Lösungshinweise zu Übungen Kapitel 7

.....

10.5.1 Lösungshinweise zu Übung 7.1

.....

10.5.2 Lösungshinweise zu Übung 7.2

10.6 Lösungshinweise zu Übungen Kapitel 8

10.6.1 Lösungshinweise zu Übung 8.1

.....

10.6.2 Lösungshinweise zu Übung 8.2

Weitere Kapitel Texte
***
***
***

11 Referenzen

Liste der Referenzen:

[**HVö-1**] - Hermann Völlinger: Script of the Lecture "Introduction to Data Warehousing"; DHBW Stuttgart; WS2021; http://www.dhbw-stuttgart.de/~hvoellin/

[**HVö-2**] - Hermann Völlinger and Other: Exercises and Solutions of the Lecture "Introduction to Data Warehousing"; DHBW Stuttgart; WS2021; http://www.dhbw-stuttgart.de/~hvoellin/

[**HVö-3**] - Hermann Völlinger and Other: Exercises and Solutions of the Lecture "Machine Learning: Concepts and Algorithms"; DHBW Stuttgart; WS2020; http://www.dhbw-stuttgart.de/~hvoellin/

[**HVö-4**] - Hermann Völlinger: Script of the Lecture "Machine Learning: Concepts and Algorithms"; DHBW Stuttgart; WS2020; http://www.dhbw-stuttgart.de/~hvoellin/

[**HVö-5**] - Hermann Völlinger: GitHub to the Lecture "Machine Learning: Concepts and Algorithms"; https://github.com/HVoellinger/Lecture-Notes-to-ML-WS2020

[**DHBW-Moodle**] - DHBW-Moodle for TINF19D: "Directory of supporting Information for the DWH Lecture"; *Kurs: T3INF4304-3-Data Warehouse (dhbw-stuttgart.de)*

[**Hands-on ML**] - Aurelien Geron "Hands-on Machine Learning with SciKit-Learn, Keras and Tensorflow"; Paperback, O'Reilly (2nd edition), 2019

[**DL for NLP**] - Kamath, Liu and Whitaker "Deep Learning for NLP and Speech Recognition"; Paperback, Springer Verlag, 2019.

[**LA and Opt for ML**] - Charu C. Aggarwal "Linear Algebra and Optimization for Machine Learning"; Paperback, Springer Verlag, 2020.

[**Math for DL**] - Berner, Grohs, Kutyniok, and Petersen: "The Modern Mathematics of Deep Learning"; Review Paper; ResearchGate arXiv:2105.04026v1 [cs.LG] 9 May 2021; This review paper will appear as a book chapter in the book "Theory

of Deep Learning" by Cambridge University Press

[**SVM-Def**] - "Was ist eine Support Vector Machine?"; 06.11.2019; Autor, Redakteur: Dipl.-Ing. (FH) Stefan Luber / Nico Litzel; https://www.bigdata-insider.de/was-ist-eine-support-vector-machine-a-880134/

[**TK + SVM**] - Seminararbeit von Alena Tabea Geduldig: "Textklassifikation mit Support Vector Machines" im Hauptseminar "Linguistic Software Engineering" bei Prof. Dr. Jürgen Rolshoven (Uni Köln); WS 2014/2015; https://docplayer.org/9656155-Textklassifikation-mit-support-vector-machines.html

[**BackP-Exam**] - "A Step by Step Backpropagation Example" by Matt Mazur (2018); https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

[**BackP-Scikit**] - "Scikit-learn 0.23.1; Ch. 1.17. Neural network models (supervised)";https://github.com/mattm/simple-neural-network

[**Wiki-ML**] - Wikipedia: "Machine learning ML); https://en.wikipedia.org/wiki/Machine_learningMachine Learning

[**Wiki-SVM**] - Wikipedia: Support Vector Machine (SVM); https://de.wikipedia.org/wiki/SupportVectorMachine(SVM)

*** Weitere Referenzen
***