

Mathematische Grundlagen des Maschinellen Lernens (ML)

— Ausarbeitung zu einer Vorlesung ("Skript") —



"3-Insel-Bild"

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

Diese Version V0.68 (Datum: 13.10.2023) ist nicht final!!!
(Beachte: ***** Kommentare *****)

Aktuelle Versionen findet man in meinem GitHub

[https://github.com/HVoellinger/Mathematische-Grundlagen_von_ML/blob/main/Math+MaschLernen\(ML\)v0.65.pdf](https://github.com/HVoellinger/Mathematische-Grundlagen_von_ML/blob/main/Math+MaschLernen(ML)v0.65.pdf)

Dieses Skript wurde generiert mit L^AT_EX

Inhaltsangabe

Contents

1 Management Zusammenfassung - Motivation und Ziele	1
1.1 Meine Motivation und Ziele der Arbeit	1
1.2 Überblick über die Mathematischen Methoden	2
1.3 Historische Anmerkungen zu Mathematik und ML	3
2 Wichtige Anwendungen der Mathematik in ML	5
3 Mathematik zum "k-Means-Algorithmus"	6
3.1 Problemstellung und Mathematik	6
3.2 k-Means Algorithmen	7
3.3 Python Programm zum k-Means-Clustering	8
3.4 k-Means-Clustering für IRIS Blumen	12
3.5 Übungen zum Kapitel 3	15
4 Lernen von Entscheidungsbäumen ("Decision Tree")	16
4.1 Mathematik bei Entscheidungsbäumen	16
4.2 Mathematische Grundlagen Gini-Index- und Entropie-Verfahren	17
4.3 Beispiele für Gini-Index- und Entropie-Verfahren	20
4.4 GINI-Index "Production Maintenance"	27
4.5 Übungen zu Kapitel 4	28
5 Lineare Regression (LR) in ML	30
5.1 Allgemeine Einführung in Regressionsmodelle	30
5.2 Motivation und Beispiele der Linearen Regression	31
5.3 Kennzahlen R^2 und $adj.R^2$	33
5.4 "Least Square Fit" (LSF) Verfahren für LR	40
5.5 Allgemeine Berechnungen von LSF für LR ($k \geq 1$)	57
5.6 simple Lineare Regression (sLR) Beispiele	65
5.7 multiple Lineare Regression mLR($k=2$) Beispiele)	70
5.8 Übungen zum Kapitel 5	72

6 Text-Klassifikation via Naive-Bayes Verfahren	76
6.1 Eine Einführung in den naiven Bayes-Klassifikator	76
6.2 Mathematische Grundlagen bei Textklassifikatoren	78
6.3 Konkrete Anwendung des naiven Bayes-Klassifikator	80
6.4 Übungen zum Kapitel 6	86
7 Verfahren der "Support Vector Machines" (SVM)	88
7.1 Grundlegende mathematische Funktionsweise	88
7.2 Funktionsweise im Detail	89
7.3 Anschauliches 2-dim. Beispiel für Kernel-Trick	92
7.4 Übungen zum Kapitel 7	96
8 Neuronale Faltungsnetzwerke "Convolutional Neural Networks"	98
8.1 Mathematische Grundlagen von CNN	98
8.2 Einfaches Beispiel - Rückwärtspropagierung	100
8.3 Übungen zum Kapitel 8	105
9 Anhänge	106
9.1 Empfehlungssysteme "Recommender Systems" (LinAlgebra)	106
9.2 Regularisierungen und Lineare Algebra (LA)	108
9.3 Hauptkomponentenanalyse "Principal Component Analysis"	110
9.4 Singulärwertzerlegung "Singular Value Decomposition" (SVD)	112
9.5 Mathematische Verfahren im NLP, i.e. "Latent Semantic Analysis/Indexing"	113
9.6 Mathematik und grosse Sprachmodelle "Large Language Models" (LLMs)	113
10 Lösungen / Lösungshinweise zu den Übungen	118
10.1 Lösungshinweise zu Übungen Kapitel 3	118
10.2 Lösungshinweise zu Übungen Kapitel 4	119
10.3 Lösungshinweise zu Übungen Kapitel 5	120
10.4 Lösungshinweise zu Übungen Kapitel 6	124
10.5 Lösungshinweise zu Übungen Kapitel 7	126
10.6 Lösungshinweise zu Übungen Kapitel 8	129

11 Referenzen

130

Bilderverzeichnis

List of Figures

1	k-Means-Konvergenz-Iteration0	7
2	Die ersten 5 Entries im IRIS Datensatz	13
3	DT-Homework(1+2)	28
4	DT-Homework(3+4)	29
5	DT-Homework5	29
6	Geometrische Darstellung von SSE und SST	36
7	Manuelle Berechnung eines sLR Beispiels	66
8	sLR-Beispiel-Python-1/3	67
9	sLR-Beispiel-Python-2/3	68
10	sLR-Beispiel-Python-3/3	69
11	Manuelle Berechnung mLR($k=2$)	71
12	Screenshots zum Naive-Bayes-Learning	77
13	Hyperebenen zum Kernel-Trick Beispiel	93
14	Python Code und Kernel-Trick Kreisbild	94
15	Backpropagation-Rückwärtslauf	103
16	Schematische Darstellung der Hauptkomponentenanalyse	110
17	Schematische Darstellung des Singulärwertzerlegung	112
18	Codeblock(1+2) zum Kernel-Trick Beispiel	127
19	Codeblock(3) zum Kernel-Trick Beispiel	128

Tabellenverzeichnis

List of Tables

1 Management Zusammenfassung - Motivation und Ziele

Ziel der Ausarbeitung ist es, die [mathematischen Konzepte des Maschinellen Lernens \(ML\)](#) zu erkennen.

1.1 Meine Motivation und Ziele der Arbeit

Die Anregung zur Erstellung dieses Skriptes liegen in den Erfahrungen, die ich bei meinen DHBW Vorlesungen in Stuttgart zur "Einführung in das Maschinelle Lernen (ML)" festgestellt habe. Viele Studenten habe große Probleme, mathematische Aussagen zu Verstehen und deren Bedeutung für ML zu begreifen.

Dies führte in einigen Fällen auch zum Abbruch des "Data Science" Studienganges. Meine beruflichen Erfahrungen im Bereich der Projektberatung für Data Warehouse Projekte bestätigen dies.

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 scherhaft sagen, dass Sie mit Mathematik "nichts am Hut haben".

Dies ist leider sehr schade und ist zudem beim Lösen von ML Fragestellungen eine große Hürde. Viele Studenten im "Data Science" Bereich scheitern deshalb auch an mangelnden Mathematik Kenntnissen.

Das 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.

Hierfür werden wir mit vielen anschaulichen, aber nicht trivialen Beispielen starten, mit deren Hilfe wir die Arbeitsweise des mathematischen Verfahrens leichter erkennen können. Die Studenten können die prinzipiellen Arbeitsweisen und Methoden der Algorithmen somit erkennen. Die Studenten werden zudem durch geeignete Übungsbeispiele angeregt die Beispiele zu Verallgemeinern. 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 Skriptes. Die Umsetzung dieser Beispiele in allgemeine mathematische Aussagen

unter Nutzung des mathematischen Kalküls wird damit leichter und verständlicher vorbereitet (nach dem Prinzip: „Vom Speziellen zum Allgemeinen“).

Es werden zudem 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 Umsetzung dieser Beispiele in allgemeine mathematische Aussagen unter Nutzung des mathematischen Kalküls wird damit leichter und verständlicher vorbereitet.

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 den mathematischen Erkenntnissen ergeben, zu entwickeln.

1.2 Überblick über die Mathematischen Methoden

Die mathematischen Grundlagen des Maschinellen Lernens („Machine Learning“) umfassen mehrere Bereiche der Mathematik, die für das Verständnis und die Entwicklung von Machine Learning Algorithmen von Bedeutung sind. Hier sind die wichtigsten **mathematische Konzepte und Grundlagen**:

1. Lineare Algebra: Lineare Algebra ist ein grundlegender Bestandteil des Maschinellen Lernens „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.

1.3 Historische Anmerkungen zu Mathematik und ML

Vor etwa 250 Jahren begann man, die Mathematik 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.

Die Nutzung von Mathematik in Machine Learning ist eine grundlegende Komponente der Entwicklung von Algorithmen und Modellen. Hier sind einige historische Anmerkungen zur Verwendung von Mathematik in diesem Bereich:

1. Entwicklung der linearen Regression (19. Jahrhundert):

Die lineare Regression ist ein grundlegendes statistisches Modell, das in Machine Learning weit verbreitet ist. Es wurde im 19. Jahrhundert entwickelt und nutzt mathematische Konzepte wie die Methode der kleinsten Quadrate, um Beziehungen zwischen Variablen zu modellieren.

2. Perceptron (1957)

Der Perceptron-Algorithmus, entwickelt von Frank Rosenblatt, war einer der frühen Versuche, maschinelles Lernen formal mathematisch zu modellieren. Es basierte auf

der Idee von künstlichen Neuronen und wurde für die Klassifikation verwendet.

3. Gradientenabstiegsverfahren (1960er Jahre):

Das Gradientenabstiegsverfahren ist ein wichtiger Optimierungsalgorithmus in der Mathematik, der später in vielen Machine-Learning-Verfahren wie Neuronalen Netzen, Support Vector Machines und tiefen Lernalgorithmen verwendet wurde.

4. Bayes'sche Statistik (18. Jahrhundert):

Die Bayes'sche Statistik und das Bayes'sche Theorem, das auf dem Werk von Thomas Bayes basiert, sind grundlegende mathematische Konzepte, die in probabilistischen Machine-Learning-Modellen wie dem Naive Bayes-Klassifikator Anwendung finden.

5. Entwicklung von neuronalen Netzen (1940er Jahre - heute):

Neuronale Netze, die in vielen modernen Machine-Learning-Anwendungen eine zentrale Rolle spielen, basieren auf mathematischen Modellen von künstlichen Neuronen und sind eng mit der Linearen Algebra und dem Konzept der Backpropagation verbunden.

6. Entwicklung von tiefen neuronalen Netzen (2010er Jahre):

Die jüngste Entwicklung von tiefen neuronalen Netzen, die in Deep Learning eingesetzt werden, beruht auf fortgeschrittenen mathematischen Konzepten, insbesondere der Backpropagation, die es ermöglichen, komplexe Modelle zu trainieren.

Die Verwendung von Mathematik in Machine Learning ist also keine neue Entwicklung, sondern hat eine lange Geschichte. Mit der Zeit wurden jedoch immer leistungsfähigere mathematische Techniken entwickelt und angewendet, um komplexe Modelle zu erstellen und große Datenmengen zu verarbeiten. Diese Entwicklung setzt sich fort, da Machine Learning und künstliche Intelligenz weiterhin stark erforscht und weiterentwickelt werden.

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. Mathematik zum k-Means-Algorithmus ("Euklidische Geometrie")
2. Entscheidungsbäume mit GINI-Index und ID3-Verfahren(Statistik)
3. Lineare Regression und "Best Fit" Verfahren der Analysis (AN)
4. Text-Klassifikation mit Bayes-Verfahren (Bedingte Wahrscheinlichkeiten)
5. Verfahren der "Support Vector Machines" (SVM) via "Kernel-Trick"
6. Neuronale Netzwerke und Backpropagation via "Absteigende Gradienten"

Am Ende kommt noch ein Kapitel **Anhang** in den sechs 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)"?
6. Mathematik und grosse Sprachmodelle "Large Language Models" (LLMs)

3 Mathematik zum "k-Means-Algorithmus"

Ein **k-Means-Algorithmus** ist ein Verfahren der Euklidischen Geometrie. Dabei wird aus einer Menge von ähnlichen Objekten eine vorher bekannte Anzahl von k Gruppen ("Clustern") gebildet.

Der Algorithmus ist eine der am häufigsten verwendeten Techniken zur Gruppierung von Objekten, da er schnell die Zentren der Cluster findet. Dabei bevorzugt der Algorithmus Gruppen mit geringer Varianz und ähnlicher Größe.

Der Algorithmus hat starke Ähnlichkeiten mit dem "Erwartungs-Maximierungs-Algorithmus (EM-Algorithmus)" und zeichnet sich durch seine Einfachheit aus.

Erweiterungen sind der "*k-Median-Algorithmus*" und der "*k-Means++Algorithmus*".

3.1 Problemstellung und Mathematik

Ziel von k-Means ist es einen den Datensatz so in k Partitionen zu teilen, dass die Summe der quadrierten Abweichungen von den Cluster-Schwerpunkten minimal ist. Mathematisch entspricht dies der Optimierung der Funktion F :

$$F = \sum_{i=1}^k \sum_{\mathbf{x}_j \in S_i} \|\mathbf{x}_j - \boldsymbol{\mu}_i\|^2$$

mit den Datenpunkten \mathbf{x}_j und
den "Schwerpunkten" $\boldsymbol{\mu}_i$ der Cluster S_i .

Diese Zielfunktion basiert auf der **Methode der kleinsten Quadrate** und man spricht auch von "Clustering durch Varianzminimierung", da die Summe der Varianzen der Cluster minimiert wird.

Da zudem $\|\mathbf{x}_j - \boldsymbol{\mu}_i\|^2$ die quadrierte "Euklidische Distanz" ist, ordnet k-Means effektiv jedes Objekt dem nächstgelegenen (nach Euklidischer Distanz) Clusterschwerpunkt zu.

Eine Visualisierung der k-Means-Konvergenz Iterationen, sehen wir im der folgenden Figur.

3 MATHEMATIK ZUM "K-MEANS-ALGORITHMUS"

3.2 k-Means Algorithmen



Figure 1: k-Means-Konvergenz-Iteration0

Um eine Animation dieser 3-Means-Konvergenz mit insgesamt 15 Iterationen zu sehen können sie auch folgenden Link auf mein GitHub aufrufen:

https://github.com/HVoellinger/Mathematische-Grundlagen_von_ML/blob/main/Images/k-Means-Konvergenz-Animation.gif

3.2 k-Means Algorithmen

Da die Suche nach der optimalen Lösung schwer ist, wird im Normalfall ein approximativer Algorithmus verwendet wie die **Heuristiken von Lloyd oder MacQueen**. Da die Problemstellung von k abhängig ist, muss dieser Parameter vom Benutzer festgelegt werden.

3 MATHEMATIK ZUM "K-MEANS-ALGORITHMUS"

3.3 Python Programm zum k-Means-Clustering

3.2.1 Lloyd-Algorithmus

Der am häufigsten verwendete k-Means-Algorithmus ist der **Lloyd-Algorithmus**, der oft als "der k-Means-Algorithmus" bezeichnet wird, obwohl Lloyd diesen Namen nicht verwendet hat. Lloyds Algorithmus besteht aus drei Schritten:

1. **Initialisierung:** Wähle k zufällige Mittelwerte ("Means"): $\mathbf{m}_1^{(1)}, \dots, \mathbf{m}_k^{(1)}$ aus dem Datensatz.
2. **Zuordnung:** Jedes Datenobjekt wird demjenigen Cluster zugeordnet, bei dem die Cluster-Varianz am wenigsten erhöht wird.

$$S_i^{(t)} = \left\{ \mathbf{x}_j : \|\mathbf{x}_j - \mathbf{m}_i^{(t)}\|^2 \leq \|\mathbf{x}_j - \mathbf{m}_{i^*}^{(t)}\|^2 \text{ für alle } i^* = 1, \dots, k \right\}$$

3. **Aktualisieren:** Berechne die Mittelpunkte der Cluster neu.

$$\mathbf{m}_i^{(t+1)} = \frac{1}{|S_i^{(t)}|} \sum_{\mathbf{x}_j \in S_i^{(t)}} \mathbf{x}_j$$

Die Schritte 2–3 werden dabei so lange wiederholt, bis sich die Zuordnungen nicht mehr ändern.

3.2.2 MacQueen'sLloyd-Algorithmus

MacQueen führte mit dem Begriff k-Means einen anderen Algorithmus ein:

1. Wähle die ersten k Elemente als Clusterzentren.
2. Weise jedes neue Element dem Cluster zu, bei dem sich die Varianz am wenigsten erhöht, und aktualisiere das Clusterzentrum.

Während es ursprünglich – vermutlich – nicht vorgesehen war, kann man auch diesen Algorithmus iterieren, um ein besseres Ergebnis zu erhalten.

3.3 Python Programm zum k-Means-Clustering

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

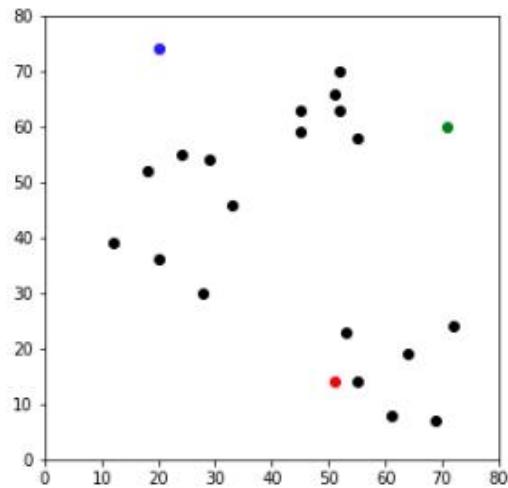
3 MATHEMATIK ZUM "K-MEANS-ALGORITHMUS"

3.3 Python Programm zum k-Means-Clustering

Display dataset

Print the centroids and the values of the data frame in a two-dimensional coordinate system.

```
1 fig = plt.figure(figsize=(5, 5))
2 plt.scatter(df['x'], df['y'], color='k')
3 colmap = {1: 'r', 2: 'g', 3: 'b'}
4 for i in centroids.keys():
5     plt.scatter(*centroids[i], color=colmap[i])
6 plt.xlim(0, 80)
7 plt.ylim(0, 80)
8 plt.show()
```



Assignment Stage

Assign each Datapoint to its closest centroid. Since the step will be repeated, we will program a function.

The distance is calculated as the difference between the two points $[x_1, y_1]$ and $[x_2, y_2]$ by the following formula:

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

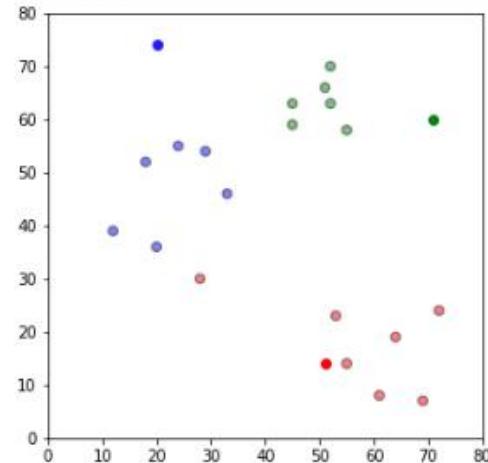
3 MATHEMATIK ZUM "K-MEANS-ALGORITHMUS"

3.3 Python Programm zum k-Means-Clustering

Display modified dataset with color assigned to closest centroid.

Create a function to display the new data frame with the additional information. Draw each cluster in a different color.

```
1 # Function to display the data frame
2 def displayDataset(df, centroids):
3     fig = plt.figure(figsize=(5, 5))
4
5     # display data frame
6     plt.scatter(df['x'], df['y'], color=df['color'], alpha=0.5, edgecolor='k')
7
8     # display each centroid
9     for i in centroids.keys():
10         plt.scatter(*centroids[i], color=colmap[i])
11
12     plt.xlim(0, 80)
13     plt.ylim(0, 80)
14     plt.show()
15
16 # invoke display function
17 displayDataset(df, centroids)
```



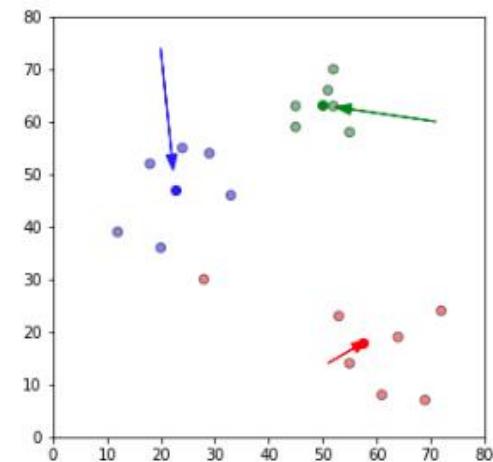
3 MATHEMATIK ZUM "K-MEANS-ALGORITHMUS"

3.3 Python Programm zum k-Means-Clustering

Display updated centroids

Display the new positions of the centroids. The change of positions is indicated with arrows.

```
1 fig = plt.figure(figsize=(5, 5))
2 ax = plt.axes()
3
4 # draw datapoints
5 plt.scatter(df['x'], df['y'], color=df['color'], alpha=0.5, edgecolor='k')
6
7 # draw centroids
8 for i in centroids.keys():
9     plt.scatter(*centroids[i], color=colmap[i])
10 plt.xlim(0, 80)
11 plt.ylim(0, 80)
12
13 # add arrows
14 for i in old_centroids.keys():
15     old_x = old_centroids[i][0]
16     old_y = old_centroids[i][1]
17     dx = (centroids[i][0] - old_centroids[i][0]) * 0.75
18     dy = (centroids[i][1] - old_centroids[i][1]) * 0.75
19     ax.arrow(old_x, old_y, dx, dy, head_width=2, head_length=3, fc=colmap[i], ec=colmap[i])
20 plt.show()
```



3 MATHEMATIK ZUM "K-MEANS-ALGORITHMUS"

3.4 k-Means-Clustering für IRIS Blumen

Repeat Assignment and Update Steps

Repeat the previous steps until there is no more modification in the assignment of the closest centroids.

```
1 # Create endless loop
2 while True:
3
4     # copy old centroid points
5     closest_centroids = df['closest'].copy(deep=True)
6
7     # calculate new means of each cluster
8     centroids = update(centroids)
9
10    # assign each datapoint to nearest centroid
11    df = assignment(df, centroids)
12
13    # if the old centroids equals the new ones => no modification made => exit loop
14    if closest_centroids.equals(df['closest']):
15        break
16
17
18 # display result
19 displayDataset(df, centroids)
```



3.4 k-Means-Clustering für IRIS Blumen

Der in diesem Beispiel verwendete Datensatz ist der berühmte **IRIS-Blumen Datensatz**. Er besteht aus 150 Einträgen der Irisblume, die jeweils durch vier Merkmale beschrieben werden: Kelchblattlänge, Kelchblattbreite, Blütenblattlänge und Blütenblattbreite.

3 MATHEMATIK ZUM "K-MEANS-ALGORITHMUS"

3.4 k-Means-Clustering für IRIS Blumen

Zusätzlich ist der spezifische Typ der Irisblume angegeben. Es gibt drei Arten der Schwertlilienpflanze: Iris Setosa, Iris Versicolor und Iris Virginica.

Jede Art ist mit mit 50 Einträgen im Datensatz vertreten. Die Merkmale sind in Zentimetern angegeben. Die Abbildung unten zeigt wie die Daten im Datenrahmen dargestellt werden. Der Datensatz wurde erstmals von Ronald Fisher, einem britischen Biologen, in einem Aufsatz aus dem Jahr 1936 vorgestellt. Er ist eines der am häufigsten verwendeten Beispiele für statistischen Klassifizierung-Algorithmen beim Maschinellen Lernen.

Die Abbildung unten zeigt die ersten fünf Einträge des Datensatzes. Der Datensatz ist in der Python-Bibliothek scikit-learn enthalten.

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

Figure 2: Die ersten 5 Entries im IRIS Datensatz

Siehe Lösungen in meinem Github:

https://github.com/HVoellinger/Mathematische-Grundlagen_von_ML/blob/main/Quellen_Math-ML/k-means-HVoe.pdf

https://github.com/HVoellinger/Mathematische-Grundlagen_von_ML/blob/main/Quellen_Math-ML/k-means-HVoe.tex

Die Konfusion-Matrix zeigt, dass alle 50 setosa-Blüten richtig klassifiziert wurden. 48 versicolor-Blüten wurden richtig klassifiziert, aber 14 wurden falsch erkannt. 36 virginica-Blüten wurden richtig beschriftet und 2 virginica-Blüten wurden fälschlicherweise als virginica-Blüten klassifiziert.

Insgesamt funktionieren Clustering und Klassifizierung mit k-Means im IRIS-Datensatz gut. Mit einer Genauigkeit von 89 Prozent ist der Algorithmus ein gutes Hilfsmittel, um ein Modell für die Vorhersage der genauen Art der Irisblüte zu erstellen.

Wie die Konfusion-Matrix zeigt, ist die Klassifikation zu 100% genau für die Setosa-Blüten und über 94% für die Versicolor-Arten. Es scheitert nur bei der genauen Virginica-Blüten mit einer Genauigkeit von "nur" 77%.

3 MATHEMATIK ZUM "K-MEANS-ALGORITHMUS"

3.4 *k*-Means-Clustering für IRIS Blumen

Das Modell von k-Means kann jedoch noch weiter verbessert werden, wie andere Implementierungen wie k-means++ zeigen. Es verbessert außerdem die Wahl der anfänglichen Zentren.

3 MATHEMATIK ZUM "K-MEANS-ALGORITHMUS"

3.5 Übungen zum Kapitel 3

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

3.5 Übungen zum Kapitel 3

.....

3.5.1 Übung 3.1 - nnn1

TEXT

3.5.2 Übung 3.2 - nnn2

.....

3.5.3 Übung 3.3 -

4 Lernen von Entscheidungsbäumen ("Decision Tree")

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

4.1 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.

4. 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.

5. 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).

6. 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.

6. 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.

4.2 Mathematische Grundlagen Gini-Index- und Entropie-Verfahren

4.2.1 Gini-Index Verfahren

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

4 LERNEN VON ENTSCHEIDUNGSBÄUMEN ("DECISION TREE")

4.2 Mathematische Grundlagen Gini-Index- und Entropie-Verfahren

4.2.2 Entropie: ID3-Verfahren

Iterative Dichotomiser3 (ID3) ist ein Algorithmus, der zur Entscheidungsfindung bei Entscheidungsbäumen eingesetzt wird.

Der australische Forscher J. Ross Quinlan publizierte diesen Algorithmus erstmals im Jahr 1986. ID3 war in seinen ersten Jahren sehr einflussreich. Er findet auch heute noch in einigen Produkten Verwendung. ID3 gilt als Vorgänger des [C4.5]-Algorithmus.

ID3 wird verwendet, wenn bei großer Datenmenge viele verschiedene Attribute von Bedeutung sind und deshalb ein Entscheidungsbaum ohne große Berechnungen generiert werden soll. Somit entstehen meist einfache Entscheidungsbäume. Es kann aber nicht garantiert werden, dass keine besseren Bäume möglich wären.

Mathematischer Algorithmus

Die Basisstruktur von ID3 ist iterativ. Es werden zu jedem noch nicht benutzten Attribut Entropien bezüglich der Trainingsmenge berechnet. Das Attribut mit dem höchsten Informationsgewinn (Englisch: *Information Gain IG*) bzw. der kleinsten Entropie, wird gewählt und daraus ein neuer Baum-Knoten generiert.

Das Verfahren terminiert, wenn alle Trainingsinstanzen klassifiziert wurden, d.h. wenn jedem Blattknoten eine Klassifikation zugeordnet ist.

Der Informationstheoretische Verständnis des Begriffes **Entropie** geht auf Claude Elwood Shannon zurück und existiert seit etwa 1948. In diesem Jahr veröffentlichte Shannon seine fundamentale Arbeit:

”A Mathematical Theory of Communication”

(<http://math.harvard.edu/~ctm/home/text/others/shannon/entropy/entropy.pdf>)

und prägte damit die moderne Informationstheorie.

Die Entropie wird üblicherweise mit einem großen griechischen Eta H bezeichnet. Claude Elwood Shannon definierte die Entropie H einer diskreten, gedächtnislosen Quelle (diskreten Zufallsvariable) X über einem endlichen, aus Zeichen bestehenden Alphabet $Z = \{z_1, z_2, \dots, z_m\}$ wie folgt:

Zunächst ordnet man jeder Wahrscheinlichkeit p eines Ereignisses seinen Informationsgehalt $I(z) = -\log_2 p_z$ zu. Dann ist die **Entropie eines Zeichens** definiert als der Erwartungswert des Informationsgehalts:

$$H_1 = E[I] = \sum_{z \in Z} p_z I(z) = - \sum_{z \in Z} p_z \log_2 p_z.$$

Sei $z \in Z$, dann ist $p_z = P(X = z)$ die Wahrscheinlichkeit, mit der das Zeichen z des Alphabets auftritt, oder gleichwertig:

4 LERNEN VON ENTSCHEIDUNGSBÄUMEN ("DECISION TREE")

4.2 Mathematische Grundlagen Gini-Index- und Entropie-Verfahren

$$H_1 = - \sum_{i=1}^m p_i \log_2 p_i \text{ mit } p_i = p_{z_i}$$

Dabei wird $0 \cdot \log_2 0 = 0$ gesetzt (entsprechend dem Grenzwert $\lim_{x \rightarrow 0} x \log_2 x$). Summanden mit verschwindender Wahrscheinlichkeit tragen daher aufgrund der Definition nicht zur Summe bei.

Die Entropie H_n für Wörter w der Länge n ergibt sich durch:

$$(1) \quad H_n = - \sum_{w \in Z^n} p_w \log_2 p_w$$

wobei $p_w = P(X = w)$ die Wahrscheinlichkeit ist, mit der das Wort w auftritt.

Die Entropie H ist dann der Grenzwert der Folge $n \rightarrow \infty$ davon:

$$(2) \quad H = \lim_{n \rightarrow \infty} \frac{H_n}{n}.$$

Wenn die einzelnen Zeichen stochastisch voneinander unabhängig sind, dann gilt

$$H_n = nH_1 \text{ für alle } n, \text{ also } H = H_1.$$

Vergleiche auch Blockentropie: https://de.wikipedia.org/wiki/Bedingte_Entropie#Blockentropie.

Auswahl der Attribute ("Merkmale"):

Sei T die Menge der Trainingsbeispiele mit ihrer jeweiligen Klassifizierung, $a \in A$ das zu prüfende Attribut aus der Menge der verfügbaren Attribute, $V(a)$ die Menge der möglichen Attributwerte von a und T_v die Untermenge von T , für die das Attribut a den Wert v annimmt.

Der Informationsgewinn, der durch Auswahl des Attributs a erzielt wird, errechnet sich dann als Differenz der Entropie von T und der erwarteten durchschnittlichen Entropie von T bei Fixierung von a :

$$IG(T, a) = \text{Entropie}(T) - \sum_{v \in V(a)} \frac{|T_v|}{|T|} \text{Entropie}(T_v).$$

Schließlich wählt man ein Attribut mit dem größtmöglichen Gewinn aus der Menge

$$\{a_{next} \in A | IG(T, a_{next}) = \max_{a \in A}(IG(T, a))\}$$

4 LERNEN VON ENTSCHEIDUNGSBÄUMEN ("DECISION TREE")

4.3 Beispiele für Gini-Index- und Entropie-Verfahren

als das nächste Attribut.

Diese Wahl führt zur Bevorzugung von Attributen mit vielen Wahlmöglichkeiten und damit zu einem breiten Baum. Um dem entgegenzuwirken kann eine Normalisierung über die Anzahl der Wahlmöglichkeiten durchgeführt werden.

Siehe auch folgenden Weblink:

https://github.com/HVoellinger/Mathematische-Grundlagen_von_ML/blob/main/Quellen_Math-ML/Homework_H4.5-DecTree_ID3_Praesentation.pdf

4.3 Beispiele für Gini-Index- und Entropie-Verfahren

4.3.1 Beispiele für Gini-Index Verfahren

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

4 LERNEN VON ENTSCHEIDUNGSBÄUMEN ("DECISION TREE")

4.3 Beispiele für Gini-Index- und Entropie-Verfahren

Oberster Knoten berechnen:

outlook:

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

$$GINI(\text{outlook}) = \sim 0,343$$

humidity:

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

$$GINI(\text{humidity}) = 0,367$$

temperature:

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

$$GINI(\text{temperatur}) = 0,444$$

windy:

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

$$GINI(\text{windy}) = 0,429$$



Für sunny: temperature |sunny

	hot	mild	cool	
Y	0	1	1	2
N	2	1	0	3
				5
				21/132

$$GINI(\text{temp|outlook=sunny}) = 0,2$$

Für windy: temperature |windy | windy |sunny

	FALSE	TRUE	
Y	1	1	2
N	2	1	3
			5

$$GINI(\text{windy|sunny}) = 0,267$$

4 LERNEN VON ENTSCHEIDUNGSBÄUMEN ("DECISION TREE")

4.3 Beispiele für Gini-Index- und Entropie-Verfahren

Für windy :

$\text{windy} \mid \text{train}$

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

$$\text{GINI}(\text{windy} \mid \text{train}) = \frac{3}{5}(1 - \frac{3}{3}^2 - \frac{0}{3}^2) + \frac{2}{5}(1 - \frac{0}{2}^2 - \frac{2}{2}^2) = 0$$

Für $\text{humidity} \mid \text{train}$

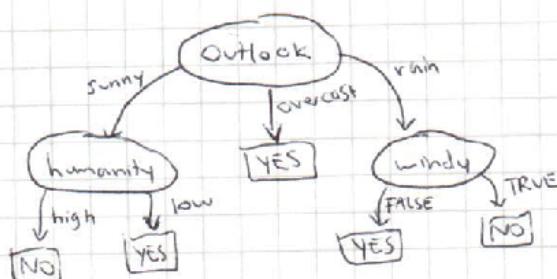
	high	normal	
Y	1	2	3
N	1	1	2
	2	3	5

$$\text{GINI}(\text{humidity} \mid \text{train}) = \frac{3}{5}(1 - \frac{1}{2}^2 - \frac{1}{2}^2) + \frac{2}{5}(1 - \frac{2}{3}^2 - \frac{1}{3}^2) > 0$$

Für $\text{temp} \mid \text{train}$

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

$$\text{GINI}(\text{temp} \mid \text{train}) = \frac{3}{5}(1 - \frac{2}{3}^2 - \frac{1}{3}^2) + \frac{2}{5}(1 - \frac{1}{2}^2 - \frac{1}{2}^2) > 0$$



4 LERNEN VON ENTSCHEIDUNGSBÄUMEN ("DECISION TREE")

4.3 Beispiele für Gini-Index- und Entropie-Verfahren

4.3.2 Beispiel für Entropie-Verfahren

1 Entropie

1.1 Begriffserklärung

Die Entropie gibt die Unreinheit von Daten an. Wenn der Wert geringer ist, dann können die Daten einfacher klassifiziert werden. Bei einem höheren Wert können die Daten schlecht klassifiziert werden. Bei einer hohen Entropie werden mehr Bits benötigt, um die Information zu beschreiben.

1.2 Formel

$$H(S) = - \sum_{c \in C} p(c) \log_2(p(c))$$

H - griechisches E (Eta), steht für Entropie

S - Datensatz

C - Menge aller Kategorien

c - Kategorie

2 Aufgabe

Entscheidungsbaum für einen Datensatz mithilfe des ID3-Algorithmus berechnen.

outlook	temp	humidity	windy	play
sunny	hot	high	false	no
sunny	hot	high	true	no
overcast	hot	high	false	yes
rainy	mild	high	false	yes
rainy	cool	normal	false	yes
rainy	cool	normal	true	no
overcast	cool	normal	true	yes
sunny	mild	high	false	no
sunny	cool	normal	false	yes
rainy	mild	normal	false	yes
sunny	mild	normal	true	yes
overcast	mild	high	true	yes
rainy	mild	high	true	no

1

Tabelle 1: Playing Tennis Game - data set

1. Schritt: Gesamtentropie berechnen

Hierfür muss die Gesamtzahl an ja/nein Ereignissen gezählt werden.

$$\begin{aligned} H(S) &= - \left(\frac{9}{14} \log_2 \left(\frac{9}{14} \right) + \frac{5}{14} \log_2 \left(\frac{5}{14} \right) \right) \\ &\approx 0.940 \end{aligned}$$

2. Schritt: Information Gain für jedes Feature berechnen

Entropie für jede Klassifizierung berechnen:

	outlook	overcast	sunny	rainy	sum
YES	4	2	3	9	
NO	0	3	2	5	
sum	4	5	5	14	

$$H(\text{outlook} = \text{overcast}) = - \left(\frac{4}{4} \log_2 \left(\frac{4}{4} \right) + 0 \log_2 (0) \right) = 0$$

$$H(\text{outlook} = \text{sunny}) = - \left(\frac{2}{5} \log_2 \left(\frac{2}{5} \right) + \frac{3}{5} \log_2 \left(\frac{3}{5} \right) \right) \approx 0.971$$

$$H(\text{outlook} = \text{rainy}) = - \left(\frac{3}{5} \log_2 \left(\frac{3}{5} \right) + \frac{2}{5} \log_2 \left(\frac{2}{5} \right) \right) \approx 0.971$$

Information Gain des Features:

$$\begin{aligned} IG(S, A_{\text{outlook}}) &= 0.94 - \left(\frac{4}{14} \cdot 0 + \frac{5}{14} \cdot 0.971 + \frac{5}{14} \cdot 0.971 \right) \\ &= 0.246 \end{aligned}$$

2

4 LERNEN VON ENTSCHEIDUNGSBÄUMEN ("DECISION TREE")

4.3 Beispiele für Gini-Index- und Entropie-Verfahren

temperature	hot	mild	cool	sum
YES	2	4	3	9
NO	2	2	1	5
sum	4	6	4	14

$$H(\text{temp} = \text{hot}) = -\left(\frac{2}{4}\log_2\left(\frac{2}{4}\right) + \frac{2}{4}\log_2\left(\frac{2}{4}\right)\right) = 1$$

$$H(\text{temp} = \text{mild}) = -\left(\frac{4}{6}\log_2\left(\frac{4}{6}\right) + \frac{2}{6}\log_2\left(\frac{2}{6}\right)\right) \approx 0.918$$

$$H(\text{temp} = \text{cool}) = -\left(\frac{3}{4}\log_2\left(\frac{3}{4}\right) + \frac{1}{4}\log_2\left(\frac{1}{4}\right)\right) \approx 0.811$$

Information Gain des Features:

$$\begin{aligned} IG(S, A_{\text{outlook}}) &= 0.94 - \left(\frac{4}{14} \cdot 1 + \frac{6}{14} \cdot 0.918 + \frac{4}{14} \cdot 0.811\right) \\ &= 0.029 \end{aligned}$$

humidity	high	normal	sum
YES	3	6	9
NO	4	1	5
sum	7	7	14

$$H(\text{humidity} = \text{high}) = -\left(\frac{3}{7}\log_2\left(\frac{3}{7}\right) + \frac{4}{7}\log_2\left(\frac{4}{7}\right)\right) \approx 0.985$$

$$H(\text{humidity} = \text{normal}) = -\left(\frac{6}{7}\log_2\left(\frac{6}{7}\right) + \frac{1}{7}\log_2\left(\frac{1}{7}\right)\right) \approx 0.592$$

3

Information Gain des Features:

$$\begin{aligned} IG(S, A_{\text{outlook}}) &= 0.94 - \left(\frac{7}{14} \cdot 0.985 + \frac{7}{14} \cdot 0.592\right) \\ &= 0.152 \end{aligned}$$

windy	FALSE	TRUE	sum
YES	6	3	9
NO	2	3	5
sum	8	6	14

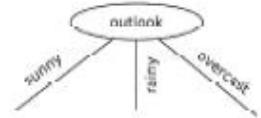
$$H(\text{windy} = \text{TRUE}) = -\left(\frac{3}{6}\log_2\left(\frac{3}{6}\right) + \frac{3}{6}\log_2\left(\frac{3}{6}\right)\right) = 1$$

$$H(\text{windy} = \text{FALSE}) = -\left(\frac{6}{8}\log_2\left(\frac{6}{8}\right) + \frac{2}{8}\log_2\left(\frac{2}{8}\right)\right) \approx 0.811$$

Information Gain des Features:

$$\begin{aligned} IG(S, A_{\text{outlook}}) &= 0.94 - \left(\frac{8}{14} \cdot 0.811 + \frac{6}{14} \cdot 1\right) \\ &= 0.049 \end{aligned}$$

3. Schritt: Das Feature mit dem größten IG wird als Wurzelknoten gewählt.
Daraus ergibt sich folgender Baum:



4

Für jeden Zweig muss rekursiv wieder ein neuer Wurzelknoten bestimmt werden.

1. Gesamtentropie berechnen:

Für das Subset S_{sunny} ergibt sich folgender Datensatz:

outlook	temp	humidity	windy	play
sunny	hot	high	false	no
sunny	hot	high	true	no
sunny	mild	high	false	no
sunny	cool	normal	false	yes
sunny	mild	normal	true	yes

$$\begin{aligned} H(S_{\text{sunny}}) &= -\left(\frac{2}{5}\log_2\left(\frac{2}{5}\right) + \frac{3}{5}\log_2\left(\frac{3}{5}\right)\right) \\ &\approx 0.971 \end{aligned}$$

2. Information Gain für jedes Feature berechnen:

temperature	hot	mild	cool	sum
YES	0	1	1	2
NO	2	1	0	3
sum	2	2	1	5

$$H(\text{temp} = \text{hot}) = 0$$

$$H(\text{temp} = \text{mild}) = 1$$

$$H(\text{temp} = \text{cool}) = 0$$

$$IG(S_{\text{sunny}}, A_{\text{temp}}) = 0.971 - \left(\frac{2}{5} \cdot 0 + \frac{2}{5} \cdot 1 + \frac{1}{5} \cdot 0\right)$$

— 24/132 —

$$H(\text{humidity} = \text{high}) = 0$$

$$H(\text{humidity} = \text{normal}) = 0$$

$$\begin{aligned} IG(S_{\text{sunny}}, A_{\text{humidity}}) &= 0.971 - \left(\frac{3}{5} \cdot 0 + \frac{2}{5} \cdot 0\right) \\ &\approx 0.971 \end{aligned}$$

windy	FALSE	TRUE	sum
YES	1	1	3
NO	1	2	3
sum	2	3	5

$$H(\text{windy} = \text{FALSE}) = 1$$

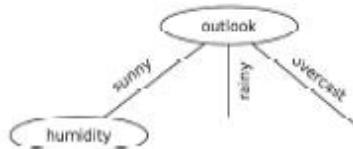
$$H(\text{windy} = \text{TRUE}) = -\left(\frac{1}{3}\log_2\left(\frac{1}{3}\right) + \frac{2}{3}\log_2\left(\frac{2}{3}\right)\right) \approx 0.918$$

$$\begin{aligned} IG(S_{\text{sunny}}, A_{\text{windy}}) &= 0.971 - \left(\frac{2}{5} \cdot 1 + \frac{3}{5} \cdot 0.918\right) \\ &\approx 0.020 \end{aligned}$$

3. Schritt: Das Feature mit dem größten IG wird als Wurzelknoten gewählt.
Daraus ergibt sich folgender Baum:

4 LERNEN VON ENTSCHEIDUNGSBÄUMEN ("DECISION TREE")

4.3 Beispiele für Gini-Index- und Entropie-Verfahren



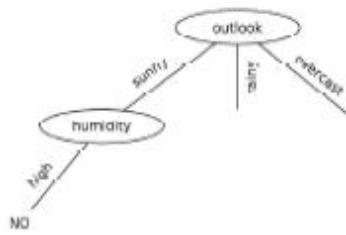
1. Gesamtentropie berechnen:

Für das Subset $S_{\text{sunny}, \text{A}_{\text{outlook}}}$ ergibt sich folgender Datensatz:

outlook	temp	humidity	windy	play
sunny	hot	high	false	no
sunny	hot	high	true	no
sunny	mild	high	false	no

Es muss keine Entropie berechnet werden, da alle Einträge das Ergebnis „no“ aufweisen.

Daraus ergibt sich folgender Baum:



7

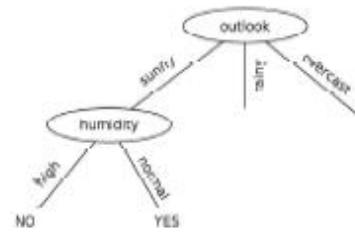
1. Gesamtentropie berechnen:

Für das Subset $S_{\text{sunny}, \text{A}_{\text{outlook}}}$ ergibt sich folgender Datensatz:

outlook	temp	humidity	windy	play
sunny	cool	normal	false	yes
sunny	mild	normal	true	yes

Es muss keine Entropie berechnet werden, da alle Einträge das Ergebnis „yes“ aufweisen.

Daraus ergibt sich folgender Baum:



8

1. Gesamtentropie berechnen:

outlook	temp	humidity	windy	play
rainy	mild	high	false	yes
rainy	cool	normal	false	yes
rainy	cool	normal	true	no
rainy	mild	normal	false	yes
rainy	mild	high	true	no

$$H(S_{\text{overcast}=\text{rainy}}) = -\left(\frac{2}{5}\log_2\left(\frac{2}{5}\right) + \frac{3}{5}\log_2\left(\frac{3}{5}\right)\right) \approx 0.971$$

2. Information Gain für jedes Feature berechnen:

temperature	mild	cool	sum
YES	2	1	3
NO	1	1	2
sum	3	2	5

$$H(\text{temp} = \text{mild}) = -\left(\frac{2}{3}\log_2\left(\frac{2}{3}\right) + \frac{1}{3}\log_2\left(\frac{1}{3}\right)\right) \approx 0.918$$

$$H(\text{temp} = \text{cool}) = 1$$

$$IG(S_{\text{rainy}}, A_{\text{temp}}) = 0.971 - \left(\frac{3}{5}0.92 + \frac{2}{5}1\right)$$

— 25/132 —

$$H(\text{humidity} = \text{high}) = 1$$

$$H(\text{humidity} = \text{normal}) = -\left(\frac{2}{3}\log_2\left(\frac{2}{3}\right) + \frac{1}{3}\log_2\left(\frac{1}{3}\right)\right) \approx 0.918$$

$$IG(S_{\text{rainy}}, A_{\text{humidity}}) = 0.971 - \left(\frac{3}{5}0.92 + \frac{2}{5}1\right) \approx 0.019$$

windy	TRUE	FALSE	sum
YES	0	3	3
NO	2	0	2
sum	2	3	5

$$H(\text{windy} = \text{TRUE}) = 0$$

$$H(\text{windy} = \text{FALSE}) = 0$$

$$IG(S_{\text{rainy}}, A_{\text{windy}}) = 0.971 - \left(\frac{3}{5}0 + \frac{2}{5}1\right) \approx 0.971$$

3. Schritt: Das Feature mit dem größten IG wird als Wurzelknoten gewählt.
Daraus ergibt sich folgender Baum:

4 LERNEN VON ENTSCHEIDUNGSBÄUMEN ("DECISION TREE")

4.3 Beispiele für Gini-Index- und Entropie-Verfahren



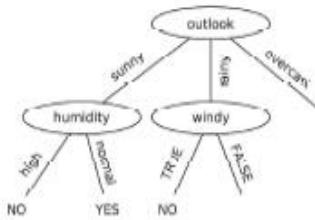
1. Gesamtentropie berechnen:

Für das Subset $S_{\text{rainy}, \text{TRUE}}$ ergibt sich folgender Datensatz:

outlook	temp	humidity	windy	play
rainy	cool	normal	true	no
rainy	mild	high	true	no

Es muss keine Entropie berechnet werden, da alle Einträge das Ergebnis „no“ aufweisen.

Daraus ergibt sich folgender Baum:



11

1. Gesamtentropie berechnen:

Für das Subset $S_{\text{rainy}, \text{FALSE}}$ ergibt sich folgender Datensatz:

outlook	temp	humidity	windy	play
rainy	cool	normal	false	yes
rainy	normal	high	false	yes
rainy	normal	normal	false	yes

Es muss keine Entropie berechnet werden, da alle Einträge das Ergebnis „yes“ aufweisen.

Daraus ergibt sich folgender Baum:



12

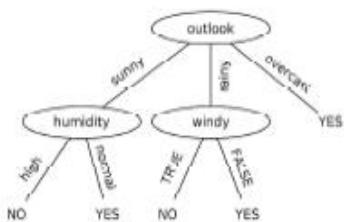
1. Gesamtentropie berechnen:

Für das Subset $S_{\text{outlook}=\text{overcast}}$ ergibt sich folgender Datensatz:

outlook	temp	humidity	windy	play
overcast	hot	high	false	yes
overcast	cool	normal	true	yes
overcast	mild	high	true	yes

Es muss keine Entropie berechnet werden, da alle Einträge das Ergebnis „yes“ aufweisen.

Daraus ergibt sich folgender Baum:



4 LERNEN VON ENTSCHEIDUNGSBÄUMEN ("DECISION TREE")

4.4 GINI-Index "Production Maintenance"

https://github.com/HVoellinger/Mathematische-Grundlagen_von_ML/blob/main/Quellen_Math-ML/Homework_H4.5-DecTree_ID3.pdf

4.4 GINI-Index "Production Maintenance"

Ab hier bis Ende Subsection sind die Folien der Vorlesung ML zu nutzen und diese sind in Latex umzusetzen...

4.5 Übungen zu Kapitel 4

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

4.5.1 Übung 4.1

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 3: DT-Homework(1+2)

4.5.2 Übung 4.2

4.5.3 Übung 4.3

4.5.4 Übung 4.4

4.5.5 Übung 4 .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 4: 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 5: DT-Homework5

5 Lineare Regression (LR) in ML

5.1 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. **Polynomiale - 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.

5.2 Motivation und Beispiele der Linearen Regression

Zusammenfassung: 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.

5.2 Motivation und Beispiele der Linearen Regression

In unserem Skript fokussieren wir uns auf die **Lineare 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 $k \geq 2$. von einer "**Multidimensionalen (multiplen) Linearen Regression**" (mLR).

Eine Trainingsmenge von n Datenpunkten $\{P_j\}$ in \mathbb{R}^{k+1} wird dargestellt als Vektor:

$$\{(x_{ij}, y_j)\}, \text{ wobei } 1 \leq i \leq k \text{ und } 1 \leq j \leq n.$$

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 (aus fachlicher Sicht) sollen aus den k Eigenschaften $\{x_{ij}\}$ eine Aussage über die Zieleigenschaft $\{y_j\}$ gemacht werden.

5.2.1 Beispiele von LR Lösungen für $k=1, k=2$

Mathematisch ("Geometrie") erhalten wir für $k=1$ eine Gerade in \mathbb{R}^2 .

Wir schreiben die **Geradengleichung** als $y = a + b \cdot x$.

a ist dabei der Achsenabschnitt ("y-intercept" auf Englisch) und b die Steigung in x-Richtung ("x-slope").

Für $k = 2$ eine Ebene in \mathbb{R}^3 mit der **Ebenengleichung** $z = a + b \cdot x + c \cdot y$.

a ist dabei der Achsenabschnitt mit der z-Achse (z-intercept), b die Steigung in x-Richtung (x-slope) und c die Steigung in y-Richtung (y-slope)

5 LINEARE REGRESSION (LR) IN ML

5.2 Motivation und Beispiele der Linearen Regression

Für $k \geq 3$ erhalten wir eine Hyperebene in \mathbb{R}^{k+1} .

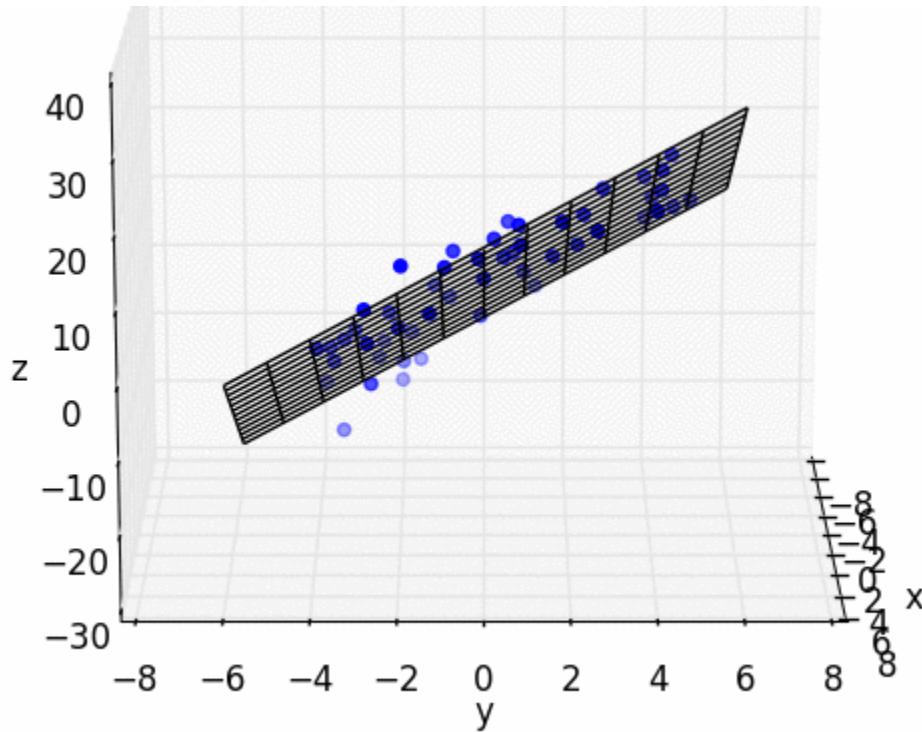
Die ersten zwei Fälle lassen sich leicht visualisieren:

k=1: Das folgende Bild zeigt die „*Regressionsgerade*“ für die n blauen Beobachtungspunkte $\{P_1 = (x_1, y_1), \dots, P_n = (x_n, y_n)\}$



k=2: Das nächste Bild zeigt die „*Regressionsebene*“ für die n blauen Beobachtungspunkte $\{P_1 = (x_1, y_1, z_1), \dots, P_n = (x_n, y_n, z_n)\}$

5.3 Kennzahlen R^2 und adj. R^2



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

5.3 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)**.

5.3.1 Definition der sLR Kennzahl R^2

Das Bestimmtheitsmaß, auch "Determinationskoeffizient" (von lateinisch: "Determinatio" "Abgrenzung, Bestimmung" bzw. "determinare" "eingrenzen", "festlegen",

5 LINEARE REGRESSION (LR) IN ML

5.3 Kennzahlen R^2 und adj. R^2

”bestimmen“ und ”coefficere“ ”mitwirken“), bezeichnet mit R^2 , ist in der Statistik eine Kennzahl zur Beurteilung der Anpassungsgüte einer Regressionsanalyse.

Das Bestimmtheitsmaß beruht auf der ”Quadratsummenzerlegung“, bei der die totale Quadratsumme in die durch das Regressionsmodell erklärte Quadratsumme einerseits und in die Residuenquadratsumme (oder auch ”Fehlerquadratsummen“) andererseits zerlegt wird.

Weil das Bestimmtheitsmaß durch die Aufnahme zusätzlicher Variablen wächst und die Gefahr der Überanpassung besteht, wird für praktische Anwendungen meist das ”adjustierte Bestimmtheitsmaß“ verwendet. Das adjustierte Bestimmtheitsmaß bestraft im Gegensatz zum unadjustierten Bestimmtheitsmaß die Aufnahme jeder neu hinzugenommenen abhängigen und unabhängigen Variable.

Die Regressiongerade $f(x)$ als Schätzer (Modellfunktion) für den Zusammenhang von Größe und Gewicht der Probanden. $f(x_i) = f_i$ ist das geschätzte Gewicht des Probanden bei einer gegebenen Größe x_i .

Der Restfehler (das Residuum) ε_i stellt die Differenz zwischen dem Messwert y_i und dem Schätzwert f_i dar.

Die Definition der Kennzahl R^2 benutzt die Kennzahlen SST und SSE). SSR wird für die eigentliche Definition von R^2 nicht genutzt. Wir brauchen diese Kennzahl aber später im Kapitel. Sei n = (Anzahl der Beobachtungen) dann definieren wir der Einfachheit halber:

$$f_i = f(x_i) \quad \text{und} \quad \bar{x} = \frac{1}{n} \sum_{i=1}^n (x_i); \quad \bar{y} = \frac{1}{n} \sum_{i=1}^n (y_i)$$

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

$$(D-5.1): \text{ Sum of Squares Total (SST)} := \sum_{i=1}^n (y_i - \bar{y})^2$$

$$(D-5.2): \text{ Sum of Squares Error (SSE)} := \sum_{i=1}^n (y_i - f_i)^2 = \sum_{i=1}^n \varepsilon_i^2$$

$$(D-5.3): \text{ Sum of Squares Regression (SSR)} := \sum_{i=1}^n (f_i - \bar{y})^2$$

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“:

5.3 Kennzahlen R^2 und adj. R^2

$$(D-5.4): R^2 := 1 - \frac{SSE}{SST}$$

Erläuterungen zu obigen Definition:

In der einfachen linearen Regression haben die Begriffe SSE, SSR und SST folgende Bedeutungen:

- **SSE (Sum of Squares Error):** SSE steht für die "Summe der quadrierten Fehler ("Error")". In der Literatur wird dies manchmal auch als "Sum of Squares Residuals" bezeichnet. Wir wählen aber lieber den Begriff "Error" um auch mit der Abkürzung SSE konsistent zu sein. SSE misst die Gesamtmenge der Abweichungen zwischen den beobachteten Datenpunkten y_i und den vorhergesagten Werten f_i der Regressionsgeraden. Es repräsentiert die nicht erklärte Variation in den Daten.
- **SSR (Sum of Squares Regression):** SSR steht für die "Summe der quadrierten Abweichungen der Regression" (Sum of Squares Regression). Die SSR misst die Variation, die durch die Regression erklärt wird. Sie quantifiziert die Summe der quadrierten Distanzen zwischen den vorhergesagten Werten \hat{y}_i und dem Mittelwert \bar{y} der abhängigen Variable.
- **SST (Total Sum of Squares):** SST steht für die "Gesamtsumme der quadrierten Abweichungen" (Total Sum of Squares). Die SST misst die Gesamtvariation der beobachteten Werte y_i um ihren Mittelwert \bar{y} . Es ist die Summe der quadrierten Abweichungen jedes Datenpunkts von \bar{y} .

Die Beziehung zwischen diesen Begriffen kann durch die Gleichung $SST = SSR + SSE$ für optimale Regressionsgeraden ausgedrückt werden. Diese Gleichung zeigt, wie die Gesamtvariation der abhängigen Variable in die Variation, die durch die Regression erklärt wird (SSR), und die nicht erklärte Variation (SSE) aufgeteilt wird.

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".

Im **folgenden Bild** sehen wir die Kennzahlen **SSE** und **SST** geometrisch mit 4 Beobachtungspunkten. Sehr gut sind die Größen der Quadrate für **SSE** und **SST** zu erkennen. Man bekommt zudem auch einen Eindruck von den Faktor $\frac{SSE}{SST}$

5.3 Kennzahlen R^2 und adj. R^2 

Figure 6: Geometrische Darstellung von SSE und SST

5.3.2 Eigenschaften und Theoreme zu R^2 ($k \geq 2$)

Nun wollen wir die ersten offensichtlichen mathematischen Aussagen zu R^2 machen. Zur leichteren Schreibweise nutze ich die folgenden Notationen:

$$\text{sum}(xi) := \sum_{i=1}^n (x_i) \text{ und } \bar{x} := \frac{1}{n} \cdot \sum_{i=1}^n (x_i)$$

Wir beginnen mit einigen einfachen mathematischen Aussagen ("Theoreme", "Propositionen", "Korollare", etc.) über die Eigenschaften der Kennzahl R^2 .

Anmerkung: Ohne Einschränkung der Allgemeinheit ("OE") können wir annehmen, dass $SST > 0$ ist.

Theorem (Th-5.1): "Eigenschaften von R^2 "

Es gelten die folgenden Aussagen:

- (i) $0 \leq R^2 \leq 1$ "Begrenzung"
- (ii) $R^2 = 0 \Leftrightarrow SSE = SST$ "Minimum"

5.3 Kennzahlen R^2 und adj. R^2

$$(iii) \quad R^2 = 1 \Leftrightarrow SSE = 0 \quad \text{"Maximum"}$$

Beweis: Der Beweis dieser Aussagen ist trivial. Um aber das mathematische Kalkül einzuüben führen wir hier die mathematische Beweisargumentation explizit aus:

$$\text{ad (i): per Definition gilt } SST \geqslant SSE \Leftrightarrow 1 \geqslant \frac{SSE}{SST} \Leftrightarrow 1 - \frac{SSE}{SST} \geqslant 0 \Leftrightarrow R^2 \geqslant 0$$

$$\text{per Definition gilt } \frac{SSE}{SST} \geqslant 0 \Leftrightarrow 0 \geqslant -\frac{SSE}{SST} \Leftrightarrow 1 \geqslant 1 - \frac{SSE}{SST} \Leftrightarrow R^2 \leqslant 1$$

$$\text{ad (ii): } SSE = SST \Leftrightarrow \frac{SSE}{SST} = 1 \Leftrightarrow 1 - \frac{SSE}{SST} = 0 \Leftrightarrow R^2 = 0$$

$$\text{ad (iii): } SSE = 0 \Leftrightarrow \frac{SSE}{SST} = 0 \Leftrightarrow 1 + \frac{SSE}{SST} = 1 \Leftrightarrow 1 = 1 - \frac{SSE}{SST} \Leftrightarrow R^2 = 1 \quad \text{q.e.d.}$$

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-5.1): “opt. sLR ohne Achsenabschnitt($a=0$)”

$$\text{Aus } a = 0 \Rightarrow b = \frac{\overline{x \cdot y}}{\overline{x^2}}$$

Beweis:

$$a = 0 \Rightarrow y = b \cdot x$$

Da die sLR-Gerade optimal ist, muss die Ableitung von R^2 nach b gleich Null. Berechne nun diese. Da die Ableitung von SST konstant ist und die Gleichung später = 0 gesetzt wird, gilt:

$$0 = \frac{\partial}{\partial b} \left(1 - \frac{SSE}{SST} \right) = \frac{\partial}{\partial b} [SSE] = \frac{\partial}{\partial b} \left[\sum_{i=1}^n [(y_i - b \cdot x_i)^2] \right] = -2 \cdot \sum (y_i \cdot x_i - b \cdot (x_i)^2)$$

daraus folgt:

$$\sum (y_i \cdot x_i) = b \cdot \sum (x_i)^2 \Rightarrow b = \frac{\sum (y_i \cdot x_i)}{\sum (x_i)^2} \Rightarrow m = \frac{\overline{x \cdot y}}{\overline{x^2}} \quad \text{q.e.d.}$$

Wir brauchen für später auch noch weitere hilfreiche Formeln über Summen und Mittelwerte. Diese werden für die Berechnung der “optimalen” Koeffizienten a und b gebraucht (siehe: “Least Square Fit” (LSF) Methode):

Proposition (P-5.1):

5.3 Kennzahlen R^2 und adj. R^2

Leicht können dann die folgenden 3 Aussagen bewiesen werden:

$$(a) \quad (i:) \quad \sum(x_i - \bar{x})^2 = \sum(x_i^2) - n \cdot \bar{x}^2$$

$$(b) \quad (ii:) \quad \sum(y_i - \bar{y})^2 = \sum(y_i^2) - n \cdot \bar{y}^2$$

$$(c) \quad (iii:) \quad \sum((x_i - \bar{x}) \cdot (y_i - \bar{y})) = \sum(x_i \cdot y_i) - n \cdot \bar{x} \cdot \bar{y}$$

Beweis:

”straightforward”:

$$\begin{aligned} (a) \quad & \sum(x_i - \bar{x})^2 = \sum(x_i^2 - 2 \cdot \bar{x} \cdot x_i + \bar{x}^2) \\ &= \sum(x_i^2) - 2 \cdot \bar{x} \cdot \sum(x_i) + \sum(\bar{x}^2) \\ &= \sum(x_i^2) - 2 \cdot n \cdot \bar{x}^2 + n \cdot \bar{x}^2 \quad (\text{weil: } \sum(x_i) = n \cdot \bar{x}) \end{aligned}$$

$$(b) \quad \text{Analog: } \sum(y_i - \bar{y})^2 = \sum(y_i^2) - n \cdot \bar{y}^2$$

$$(c) \quad \text{Analog: } \sum((x_i - \bar{x})(y_i - \bar{y})) = \sum(x_i \cdot y_i) - n \cdot \bar{x} \cdot \bar{y} \quad \text{q.e.d.}$$

5.3.3 Definition der mLR-Kennzahl adj.R²

In diesem Abschnitt definieren wir die Theorie der ”multiple 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 $i \in \{1, \dots, n\}$.

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:

$F : \mathbb{R}^k \rightarrow \mathbb{R}$ zu berechnen, so dass

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

5 LINEARE REGRESSION (LR) IN ML

5.3 Kennzahlen R^2 und adj. R^2

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

Die Kennzahl adj. R^2 , wobei "adj." für "adjustiert" (angepasst) steht, soll nun genauer definiert werden. Die adj. R^2 ist eine modifizierte Version des gewöhnlichen R^2 ("Bestimmtheitsmaß").

Während das normale R^2 die Proportion der abhängigen Variabilität erklärt, kann das adj. R^2 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 (Datenpunkte) und k = Anzahl der unabhängigen Variablen (Prädiktoren) und definieren wir $df := n-k-1$ ("Anzahl der Freiheitsgrade"). Dann ergibt sich, unter Nutzung von (D-5.4), folgende Formel:

$$(D-5.5) \quad \text{adj.}R^2 := 1 - (1 - R^2) \cdot \left(\frac{n-1}{n-k-1}\right) = 1 - \left(\frac{SSE}{SST}\right) \cdot \left(\frac{n-1}{n-k-1}\right)$$

Das adj. R^2 berücksichtigt die Anzahl der unabhängigen Variablen ("Prädiktoren") im Modell und passt das gewöhnliche R^2 an, um zu verhindern, dass es aufgrund von Überanpassung ("Overfitting") zu optimistisch wird. Ein höheres adj. R^2 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 observations	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

5 LINEARE REGRESSION (LR) IN ML

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

in der Literatur auch der Zusatz "optimale" weggelassen und man sagt nur "mLR-Hyperebene".

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

Mit den LSF Verfahren oder in Deutsch "Methode der kleinsten Quadrate" lassen sich nun die optimale sLR-Gerade ($k=1$) und im Fall ($k=2$) die optimale mLR-Ebene berechnen. Wir nutzen das "Max-Min" Kriterium der Analysis.

Wir berechnen dazu explizit die Ableitung der Funktion R^2 nach ihren Veränderlichen in den Fällen ($k=1$) und ($k=2$).

Um dies Durchführen zu können brauchen wir noch einige mathematische Grundlagen der Matrizenrechnung aus der Linearen Algebra. Dies sind die Berechnung einer Inversen einer Matrix und die Berechnung der Determinante einer Matrix. Beispielsweise zeigen wir jetzt nur die Formel für die Berechnung einer Determinante. Der Rest ist Wiederholung der Linearen Algebra Vorlesung.

5.4.1 Lineare Algebra - Inverse und Determinante einer Matrix

In diesem Unterkapitel machen wie eine kleine Wiederholung der Vorlesung über Lineare Algebra. Wir definieren die Inverse und die Determinante einer Matrix und geben ein Beispiel dazu.

Berechnung der Inversen Matrix A^{-1}

Du kannst die Matrixgröße und die Elemente entsprechend deiner spezifischen Matrix anpassen. Die inverse Matrix von A :

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}$$

kann mit der Formel

$$A^{-1} = \frac{1}{\det(A)} \cdot \text{adj}(A) \quad (\text{Inv})$$

berechnet werden, wobei $\text{adj}(A)$ die adjugierte (auch bekannt als die adjazente oder

5 LINEARE REGRESSION (LR) IN ML

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

komplementäre) Matrix von A ist.

Die adjungierte Matrix von A (oft als $\text{adj}(A)$ abgekürzt) wird berechnet, indem man die Kofaktormatrix transponiert:

$$\text{adj}(A) = \begin{bmatrix} C_{11} & C_{21} & \cdots & C_{n1} \\ C_{12} & C_{22} & \cdots & C_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ C_{1n} & C_{2n} & \cdots & C_{nn} \end{bmatrix}$$

wobei C_{ij} die Kofaktoren der Elemente von A sind. Die Kofaktoren werden durch das Entfernen der i-ten Zeile und j-ten Spalte aus A und berechnen der Determinante der verbleibenden $(n-1) \times (n-1)$ -Matrix erhalten.

Berechnung der Determinante $\det(A)$

Die Determinante von einer (nxn) -Matrix A wird bezeichnet durch folgende Notation:¹

$$\begin{vmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{vmatrix}$$

und kann mit dieser allgemeinen Formel berechnet werden:

$$\det(A) = \sum_{i=1}^n (-1)^{i+j} \cdot a_{ij} \cdot \det(A_{ij}) \quad (\text{Det-1})$$

wobei A_{ij} die Matrix ist, die durch Entfernen der i-ten Zeile und j-ten Spalte aus A entsteht.

Bemerkung: Definition der Determinante nach Leibniz (1690):

In dieser Bemerkung definieren wir etwas allgemeiner und eleganter die Determinante einer $n \times n$ Matrix so, wie dies 1690 von [Gottfried Wilhelm Leibniz](#)² vorgeschlagen

¹Senkrechte Striche vor und hinter der Matrix A bedeutet Determinante von A

²G. W. Leibniz war einer der bekanntesten Mathematiker, Philosophen und Naturwissenschaftler seiner Epoche.

5 LINEARE REGRESSION (LR) IN ML

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

wurde. Wir verzichten hier aus Zeitgründen auf die Einführung von Permutationsgruppen. Dies kann auch in Wikipedia leicht nachgeschaut werden.

Ist $A = (a_{i,j}) \in \mathbb{K}^{n \times n}$, wobei entweder $\mathbb{K} = \mathbb{R}$ oder $\mathbb{K} = \mathbb{C}$ gilt, so definieren wir die **Determinante** von A (geschrieben $\det(A)$) über die Formel:

$$\det(A) = \sum_{\sigma \in \mathcal{S}_n} \operatorname{sgn}(\sigma) \cdot \prod_{i=1}^n a_{i,\sigma(i)} = \sum \left\{ \operatorname{sgn}(\sigma) \cdot \prod_{i=1}^n a_{i,\sigma(i)} \mid \sigma \in \mathcal{S}_n \right\}$$

(Det-2)

Beispiel für eine 2×2 Matrix:

Es sei A eine 2×2 Matrix, es gilt also

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{pmatrix}$$

Zur Berechnung von $\det(A)$ (nach Leibniz, Formel (Det-2)) müssen wir uns überlegen, wie \mathcal{S}_2 aussieht. Es gilt

$$\mathcal{S}_2 := \{[1, 2], [2, 1]\}.$$

Für die Permutation $[1, 2]$ finden wir: $\operatorname{sgn}([1, 2]) = (-1)^0 = +1$.

Und für die Permutation $[2, 1]$ ergibt sich: $\operatorname{sgn}([2, 1]) = (-1)^1 = -1$.

Insgesamt haben wir

$$\det(A) = \det \begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{pmatrix} = (+1) \cdot a_{1,1} \cdot a_{2,2} + (-1) \cdot a_{1,2} \cdot a_{2,1} = a_{1,1} \cdot a_{2,2} - a_{1,2} \cdot a_{2,1}$$

Somit ergibt sich mit der Formel (Inv):

$$A^{-1} = \frac{1}{\det A} \cdot \begin{pmatrix} a_{2,2} & -a_{2,1} \\ -a_{1,2} & a_{1,1} \end{pmatrix}$$

Es wird empfohlen bei solchen Rechnungen immer eine Probe zu machen, indem man jetzt mit den konkreten Werten die Gleichung: $A \cdot A^{(-1)} = E$ (Einheitsmatrix) nachrechnet.

Beispiel für eine 3×3 Matrix:

Für eine 3×3 Matrix A lassen sich Inverse und Determinante analog berechnen.

5 LINEARE REGRESSION (LR) IN ML

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

Eine manuelle Berechnung ist jedoch sehr aufwendig und es ist ratsam einen entsprechenden ML Algorithmus zu benutzen. Du kannst dazu etwa Jupyter Notebook ausführen. Stelle sicher, dass die benötigte NumPy-Bibliothek installiert ist. Im entsprechenden Codebeispiel (weiter unten) entwickeln wird ein Notebook erstellt, dass für eine symmetrische 3x3-Matrix A, ihre Determinante und Adj. Matrix berechnet, woraus sich die inverse Matrix von A sich ergibt.

Gegeben sei die 3x3-Matrix A:

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

Berechnung der adjungierten Matrix $\text{adj}(A)$:

$$\text{adj}(A) = \begin{bmatrix} C_{11} & C_{21} & C_{31} \\ C_{12} & C_{22} & C_{32} \\ C_{13} & C_{23} & C_{33} \end{bmatrix}$$

wobei C_{ij} die Kofaktoren der Elemente von A sind:

$$\begin{aligned} C_{11} &= \begin{vmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{vmatrix}, & C_{12} &= -\begin{vmatrix} a_{21} & a_{23} \\ a_{31} & a_{33} \end{vmatrix}, & C_{13} &= \begin{vmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{vmatrix} \\ C_{21} &= -\begin{vmatrix} a_{12} & a_{13} \\ a_{32} & a_{33} \end{vmatrix}, & C_{22} &= \begin{vmatrix} a_{11} & a_{13} \\ a_{31} & a_{33} \end{vmatrix}, & C_{23} &= -\begin{vmatrix} a_{11} & a_{12} \\ a_{31} & a_{32} \end{vmatrix} \\ C_{31} &= \begin{vmatrix} a_{12} & a_{13} \\ a_{22} & a_{23} \end{vmatrix}, & C_{32} &= -\begin{vmatrix} a_{11} & a_{13} \\ a_{21} & a_{23} \end{vmatrix}, & C_{33} &= \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} \end{aligned}$$

Die Kofaktoren werden durch die Berechnung der Determinanten von 2x2-Untermatrizen erstellt. Du kannst die Werte der Matrix A anpassen, um die Kofaktoren und die adjugierte Matrix für deine spezifische Matrix zu berechnen.

Berechnung der Determinante $\det(A)$:

$$\det(A) = a_{11} \cdot \begin{vmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{vmatrix} - a_{12} \cdot \begin{vmatrix} a_{21} & a_{23} \\ a_{31} & a_{33} \end{vmatrix} + a_{13} \cdot \begin{vmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{vmatrix}$$

Berechne die Determinanten der 2x2-Untermatrizen, indem du die Differenz der Produkte der Hauptdiagonalen und Nebendiagonalen verwendest:

$$\det(A) = a_{11} \cdot ((a_{22} \cdot a_{33}) - (a_{23} \cdot a_{32})) - a_{12} \cdot ((a_{21} \cdot a_{33}) - (a_{23} \cdot a_{31})) + a_{13} \cdot ((a_{21} \cdot a_{32}) - (a_{22} \cdot a_{31}))$$

5 LINEARE REGRESSION (LR) IN ML

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

Beispiel für eine symmetrische 3×3 Matrix:

In diesem Beispiel wird die Inverse und Determinante der 3×3 -Matrix A mithilfe des Laplace'schen³ Entwicklungssatzes berechnet.

Dieser Satz ermöglicht es, die Determinante einer Matrix durch die Berechnung von Determinanten kleinerer Matrizen zu finden, was besonders nützlich ist, wenn die Matrix groß ist.

Um die Inverse einer Matrix zu berechnen, könnte man den Laplace-Entwicklungssatz verwenden, um die Determinante der Matrix zu finden und dann die Cofaktormethode oder eine andere Methode zur Berechnung der Inversen anwenden.

Du kannst die Werte der Matrix A anpassen, um die Determinante für deine spezifische Matrix zu berechnen.

Ist die Matrix **symmetrisch** so vereinfachen sich die Formel wesentlich. Gegeben sei die symmetrische 3×3 -Matrix A :

$$A = \begin{bmatrix} a & b & c \\ b & d & e \\ c & e & f \end{bmatrix}$$

Um die inverse Matrix A^{-1} zu berechnen, führen wir die folgenden Schritte aus:

1. Berechne die Determinante von A :

$$\det(A) = a(df - ce) - b(ef - cd) + c(be - dc)$$

³Pierre-Simon Laplace war einer der führenden Mathematiker und Wissenschaftler des 18. und 19. Jahrhunderts und hinterließ einen erheblichen Einfluss auf verschiedene Bereiche der Mathematik und Naturwissenschaften.

5 LINEARE REGRESSION (LR) IN ML

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

2. Berechne die Kofaktoren der Matrix A :

$$\begin{aligned} C_{11} &= \begin{vmatrix} d & e \\ e & f \end{vmatrix} = df - e^2, \\ C_{12} = C_{21} &= -\begin{vmatrix} b & c \\ e & f \end{vmatrix} = -bf + ce, \\ C_{13} = C_{31} &= \begin{vmatrix} b & d \\ c & e \end{vmatrix} = be - dc, \\ C_{22} &= \begin{vmatrix} a & c \\ c & f \end{vmatrix} = af - c^2, \\ C_{23} = C_{32} &= -\begin{vmatrix} a & b \\ c & e \end{vmatrix} = -ae + bc, \\ C_{33} &= \begin{vmatrix} a & b \\ b & d \end{vmatrix} = ad - b^2. \end{aligned}$$

3. Erstelle die adjungierte Matrix $\text{adj}(A)$ durch Transposition der Kofaktormatrix:

$$\text{adj}(A) = \begin{bmatrix} C_{11} & C_{21} & C_{31} \\ C_{12} & C_{22} & C_{32} \\ C_{13} & C_{23} & C_{33} \end{bmatrix}$$

4. Berechne die inverse Matrix A^{-1} , indem du $\text{adj}(A)$ durch die Determinante von A teilst:

$$A^{-1} = \frac{1}{\det(A)} \cdot \text{adj}(A)$$

Die inverse Matrix A^{-1} ist dann die resultierende 3x3-Matrix.

Dieser LaTeX-Code zeigt die korrekte Berechnung der inversen Matrix für die gegebene symmetrische 3x3-Matrix A unter Verwendung des Laplace'schen Entwicklungssatzes (siehe oben).

Auch hier wird empfohlen bei dieser Rechnung eine Probe zu machen, indem man jetzt mit den konkreten Werten die Gleichung: $A \cdot A^{(-1)} = E(\text{Einheitsmatrix})$ nachrechnet.

Python-Programm zur Berechnung der obigen Inversen:

Wir entwickeln ein Python Programm als Jupyter Notebook. Du kannst dieses Code-Snippet in einem Jupyter Notebook ausführen. Importiere die benötigte NumPy-Bibliothek.

5 LINEARE REGRESSION (LR) IN ML

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

Das Notebook erstellt eine symmetrische 3x3-Matrix A, berechnet $\text{Det}(A)$ und $\text{Adj}(A)$, woraus sich $\text{Inv}(A)$ ergibt. Insgesamt besteht das Programm aus 4 Schritten.

5 LINEARE REGRESSION (LR) IN ML

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

1. Schritt: Vorbereitungen und Berechnung+ Ausgabe von Det(A):

Importiere NumPy-Bibliothek. Definiere symmetrische (3x3)-Matrix. Berechne Det(A) und zeige das Ergebniss.

```
In [1]: # Importiere die NumPy-Bibliothek
import numpy as np

# Importiere das sympy-Modul
import sympy as sp

# Import Library time to check execution date+time
import time

# Definiere die reellen Variablen als Symbole
a, b, c, d, e, f = sp.symbols('a b c d e f')

# Definiere die 3x3-Matrix A mit den Symbolen
A=sp.Matrix([[a,b,c],
             [b,d,e],
             [c,e,f]])

# Berechne die Determinante von A und klammere sie aus
det_A=sp.det(A)

print("Die Determinante von A = Det(A) ist gegeben durch:")
det_A
```

Die Determinante von A = Det(A) ist gegeben durch:

Out[1]: $adf - ae^2 - b^2f + 2bce - c^2d$

2. Schritt: Berechne Kofaktoren als einzelne Zeilen und Adj(A):

Adj(A) = symmetrisch ($C_{12}=C_{21}$; $C_{13}=C_{31}$) mit 6 Kofaktoren: $C_{11}, C_{12}, C_{13}, C_{22}, C_{23}$ und C_{33} .

```
In [2]: # Berechne die 6 Kofaktoren
C_11 = (A[1, 1] * A[2, 2] - A[2, 1] * A[1, 2])
C_12 = -(A[1, 0] * A[2, 2] - A[2, 0] * A[1, 2])
C_13 = (A[1, 0] * A[2, 1] - A[2, 0] * A[1, 1])
C_22 = (A[0, 0] * A[2, 2] - A[2, 0] * A[0, 2])
C_23 = -(A[0, 0] * A[2, 1] - A[2, 0] * A[0, 1])
C_33 = (A[0, 0] * A[1, 1] - A[1, 0] * A[0, 1])

# Berechne die adjungierte Matrix Adj(A)
Adj_A = sp.Matrix([[C_11, C_12, C_13],
                  [C_12, C_22, C_23],
                  [C_13, C_23, C_33]])
```

5 LINEARE REGRESSION (LR) IN ML

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

3. Schritt: Ausgabe Kofaktoren als einzelne Zeilen und Adj(A):

```
In [3]: # Ausgabe der Kofaktoren als einzelne Zeilen und adjungierten Matrix
print("Die Kofaktoren sind:")
print("C_11 =", C_11)
print("C_12 =", C_12)
print("C_13 =", C_13)
print("C_22 =", C_22)
print("C_23 =", C_23)
print("C_33 =", C_33)
print(" ")
# Darstellung der adjungierten Matrix Adj(A) in Matrixschreibweise einer 3x3-Matrix
print("Die adjungierte Matrix ist somit Adj(A):")
Adj_A
```

Die Kofaktoren sind:

```
C_11 = d*f - e**2
C_12 = -b*f + c*e
C_13 = b*e - c*d
C_22 = a*f - c**2
C_23 = -a*e + b*c
C_33 = a*d - b**2
```

Die adjungierte Matrix ist somit Adj(A):

```
Out[3]: 
$$\begin{bmatrix} df - e^2 & -bf + ce & be - cd \\ -bf + ce & af - c^2 & -ae + bc \\ be - cd & -ae + bc & ad - b^2 \end{bmatrix}$$

```

4. Schritt: Berechnung und Ausgabe von Inv(A):

$\text{Inverse}(A) = 1/\det(A) * \text{Adj}(A)$, wobei $\det(A)$ im 1. und $\text{Adj}(A)$ im 2. Schritt berechnet wurde.

```
In [4]: # Berechne die inverse Matrix von A
A_inv = (1 / det_A) * Adj_A

# Beschreibung der Formel für Inverse(A)
print("Inv(A) = 1/det(A)*Adj(A) ist somit gegeben durch:")
A_inv
```

Inv(A) = 1/det(A)*Adj(A) ist somit gegeben durch:

```
Out[4]: 
$$\begin{bmatrix} \frac{df - e^2}{adf - ae^2 - b^2f + 2bce - c^2d} & \frac{-bf + ce}{adf - ae^2 - b^2f + 2bce - c^2d} & \frac{be - cd}{adf - ae^2 - b^2f + 2bce - c^2d} \\ \frac{-bf + ce}{adf - ae^2 - b^2f + 2bce - c^2d} & \frac{af - c^2}{adf - ae^2 - b^2f + 2bce - c^2d} & \frac{-ae + bc}{adf - ae^2 - b^2f + 2bce - c^2d} \\ \frac{be - cd}{adf - ae^2 - b^2f + 2bce - c^2d} & \frac{-ae + bc}{adf - ae^2 - b^2f + 2bce - c^2d} & \frac{ad - b^2}{adf - ae^2 - b^2f + 2bce - c^2d} \end{bmatrix}$$

```

5 LINEARE REGRESSION (LR) IN ML

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

5.4.2 Detailberechnung für sLR

Sei die optimale Regression-Gerade gegeben durch $y = a + b \cdot x$ dann ist R^2 eine Funktion in den Veränderlichen a und b : " $R^2 = R^2(a, b)$ ". Somit muss für ein Maximum gelten, dass der Gradient $\Delta R^2 = 0$ ist (Analysis I):

$$dR^2 = \frac{\partial R^2}{\partial a} \cdot da + \frac{\partial R^2}{\partial b} \cdot db = 0 \iff \frac{\partial R^2}{\partial a} = \frac{\partial R^2}{\partial b} = 0$$

Wir rechnen als erstes die **Ableitung nach a** :

$$\frac{\partial R^2}{\partial a} = \frac{\partial}{\partial a} \left(1 - \frac{SSE}{SST} \right) = \frac{\partial}{\partial a} (1) - \frac{\partial}{\partial a} \left(\frac{SSE}{SST} \right) = 0 - \frac{\partial}{\partial a} (SSE)$$

Die letzte Gleichung ist gültig, da das Gesamtergebnis ebenfalls = 0 gesetzt wird und $\frac{\partial}{\partial a} \left(\frac{1}{SST} \right) = const.$ ist. Damit kann diese Ableitung vernachlässigt werden. Somit erhält man:

$$0 = \frac{\partial}{\partial a} (SSE) = \frac{\partial}{\partial a} \left[\sum_{i=1}^n ((y_i - a - b \cdot x_i)^2) \right] = -(2) \cdot \left(\sum (y_i - a - b \cdot x_i) \right)$$

Durch Benutzung der Definition des Mittelwertes und Kürzen mit (-2) erhält man:

$$\begin{aligned} 0 &= n \cdot \bar{y} - n \cdot a - n \cdot b \cdot \bar{x} \iff 0 = \bar{y} - a - b \cdot \bar{x} \\ &\iff a = \bar{y} - b \cdot \bar{x} \quad \iff \bar{y} = a + b \cdot \bar{x} \end{aligned} \quad (1)$$

Analog berechnen wir die **Ableitung nach b** :

$$\frac{\partial R^2}{\partial b} = \frac{\partial [1 - \frac{SSE}{SST}]}{\partial b} = \frac{\partial 1}{\partial b} - \frac{\partial [\frac{1}{SST}]}{\partial b} \cdot \frac{\partial SSE}{\partial b} = 0 - \frac{\partial [\frac{1}{SST}]}{\partial b} \cdot \frac{\partial SSE}{\partial b}$$

Da diese Gleichung = 0 gesetzt wird und $\frac{\partial [\frac{1}{SST}]}{\partial b} = const.$ ist, kann diese Ableitung vernachlässigt werden. Somit erhält man:

$$0 = \frac{\partial (SSE)}{\partial b} = \frac{\partial}{\partial b} \left[\sum_{i=1}^n [(y_i - a - b \cdot x_i)^2] \right] = -2 \cdot \sum [y_i \cdot x_i - a \cdot x_i - b \cdot x_i^2]$$

Durch Benutzung der Definition des Mittelwertes und Kürzen durch (-2) erhält man:

$$\begin{aligned} 0 &= \sum (y_i \cdot x_i) - a \cdot n \cdot \bar{x} - b \cdot \sum (x_i^2) \\ &\iff \sum (y_i \cdot x_i) = a \cdot n \cdot \bar{x} + b \cdot \sum (x_i^2) \end{aligned} \quad (2)$$

5 LINEARE REGRESSION (LR) IN ML

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

Schreiben wir nun die beiden Gleichungen (1) und (2) mit einer 2x2- Matrix um, so erhält man nach den Regeln der Matrizenmultiplikation (siehe Vorlesung "Lineare Algebra"):

$$\begin{bmatrix} 1 & \bar{x} \\ n \cdot \bar{x} & \sum x_i^2 \end{bmatrix} \cdot \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} \bar{y} \\ \sum x_i \cdot y_i \end{bmatrix}$$

In der Lin. Algebra wird die Inverse einer 2x2- Matrix: $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ berechnet als:

$$A^{-1} = \frac{1}{ad-bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

wobei Determinante(A)= detA = ad - bc. Mit obigen Werten eingesetzt erhält man:

$$det = \sum x_i^2 - n \cdot \bar{x}^2 \quad (3)$$

$$\begin{bmatrix} a \\ b \end{bmatrix} = \frac{1}{det} \cdot \begin{bmatrix} \sum x_i^2 & -\bar{x} \\ -n \cdot \bar{x} & 1 \end{bmatrix} \cdot \begin{bmatrix} \bar{y} \\ \sum x_i \cdot y_i \end{bmatrix}$$

Unter Berücksichtigung von ($det = \sum x_i^2 - n \cdot \bar{x}^2$) ist dies äquivalent zu den folgenden zwei Formeln :

$$a = \frac{1}{det} \cdot (\bar{y} \cdot \sum x_i^2 - \bar{x} \cdot \sum x_i \cdot y_i) \quad (4)$$

$$b = \frac{1}{det} \cdot (\sum x_i \cdot y_i - n \cdot \bar{x} \cdot \bar{y}) \quad (5)$$

Weitere Umformulierungen von (4) und (5):

Zur Berechnung des Achsenabschnittes ("intercept") a setzt man nun (P4.1-(i)) und (P5.1-(iii)) in Formel (3). Man kürzt anschließend doppelte Komponenten, so erhält man:

$$\begin{aligned} a &= \frac{1}{det} \cdot (\bar{y} \cdot \sum x_i^2 - \bar{x} \cdot \sum x_i \cdot y_i) && \text{(nach Formel (3))} \\ &= \frac{1}{det} \cdot (\bar{y} \cdot (\sum (x_i - \bar{x})^2 + n \cdot \bar{x}^2) - \bar{x} \cdot \sum x_i \cdot y_i) && \text{(P4.1-(i))} \end{aligned}$$

5 LINEARE REGRESSION (LR) IN ML

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

$$\begin{aligned}
 &= \frac{1}{det} \cdot (\bar{y} \cdot (\sum (x_i - \bar{x})^2 + n \cdot \bar{x}^2) - \bar{x} \cdot (n \cdot \bar{x} \cdot \bar{y} + \sum [(x_i - \bar{x}) \cdot (y_i - \bar{y})]) \quad (\text{P4.1-(iii)}) \\
 &= \frac{1}{det} \cdot (\bar{y} \cdot (\sum (x_i - \bar{x})^2) - \bar{x} \cdot \sum [(x_i - \bar{x}) \cdot (y_i - \bar{y})]) \quad (\text{Doppelten Faktor kürzen})
 \end{aligned}$$

Analog lässt sich auch die es Steigung ("slope") b aus Formel (4) umformen und man erhält:

$$b = \frac{1}{det} \cdot \sum [(x_i - \bar{x}) \cdot (y_i - \bar{y})].$$

Notation und "Bias-Trick": Zur Vereinfachung der Formeln und besserer Übersichtlichkeit führen wir das "Tilde" Symbol $\tilde{x}_i := (x_i - \bar{x})$ und $\tilde{y}_i := (y_i - \bar{y})$ ein.

Der **Trick** besteht darin, daß wir das Ergebnis der Optimierung des "Biases" a in die Gleichung zur Optimierung von der "Steigung" b einsetzen.

Dadurch vereinfachen sich die Gleichungen, denn wir keine 2x2-Matrix invertieren, sondern können mit Skalaren rechnen. Aus der Linearen Algebra wissen wir, daß dies eine enorme Vereinfachung und Ersparnisse in den Berechnungen bedeutet. Ich nenne diesen Trick **"Bias-Trick"**, da wir diesen Effekt durch das separate Vorziehen der Berechnung des optimalem "Biases" erreicht haben.

Damit ergibt sich auch das folgende Korollar:

Korollar (K-5.2):

$$det = \sum \tilde{x}_i^2$$

Beweis:

Formel (3) von oben besagt: $det = \sum x_i^2 - n \cdot \bar{x}^2$

Nach Proposition (P-5.1)(i) gilt: $\sum (x_i - \bar{x})^2 = \sum (x_i^2) - n \cdot \bar{x}^2$

Setzen wir die Definition von $\tilde{x}_i = (x_i - \bar{x})$ ein, so erhalten wir das gewünschte Ergebnis.

q.e.d.

Wir erhalten wir als Zusammenfassung der obigen Berechnungen das folgende Theorem:

Theorem (Th-5.2): "LSF Komponenten für optimale sLR-Gerade"

Sei $det = \sum \tilde{x}_i^2$. Für die Achsenabschnitte a und die Steigung b einer optimalen sLR - Geraden $y = a + b \cdot x$ gelten folgende Formeln:

$$(\text{sLSF-I}): \quad a = \frac{1}{det} \cdot (\bar{y} \cdot \sum (\tilde{x}_i^2) - \bar{x} \cdot \sum [\tilde{x}_i \cdot \tilde{y}_i])$$

$$(\text{sLSF-I}^*): \quad a = \bar{y} - b \cdot \bar{x}$$

5 LINEARE REGRESSION (LR) IN ML

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

$$(\text{sLSF-II}): \quad b = \frac{1}{\det} \cdot \sum [\tilde{x}_i \cdot \tilde{y}_i]$$

Dabei steht:

$$\sum \text{ für die Summe über alle } n \text{ Datenpunkte} := \sum_{i=1}^n x_i,$$

$$\bar{x} \text{ für den Durchschnittswert der } x\text{-Werte} := \frac{1}{n} \cdot \sum_{i=1}^n x_i,$$

$$\bar{y} \text{ für den Durchschnittswert der } y\text{-Werte} := \frac{1}{n} \cdot \sum_{i=1}^n y_i,$$

Beweis:

Der Beweis der Formeln (sLSF-I) und (sLSF-II) wurde durch Berechnungen oben gezeigt. Der Beweis (sLSF-I*) wurde schon in Formel (1) gezeigt. Wir beweisen in jedoch noch einmal direkt durch Einsetzen von $\det = \sum \tilde{x}_i^2$:

$$\begin{aligned} a &= \frac{1}{\det} \cdot (\bar{y} \cdot \sum \tilde{x}_i^2 - \bar{x} \cdot \sum [\tilde{x}_i \cdot \tilde{y}_i]) && (\text{sLSF-I}) \\ &= \frac{1}{\sum \tilde{x}_i^2} \cdot (\bar{y} \cdot \sum \tilde{x}_i^2 - \bar{x} \cdot \sum [\tilde{x}_i \cdot \tilde{y}_i]) && (\text{Einsetzen (K4.2)}) \\ &= \bar{y} + \bar{x} \cdot \left[\frac{\sum [\tilde{x}_i \cdot \tilde{y}_i]}{\sum \tilde{x}_i^2} \right] && (\text{Kürzen}) \\ &= \bar{y} + \bar{x} \cdot b && (\text{sLF-II}) \\ &\text{q.e.d.} \end{aligned}$$

5.4.3 Weitere Folgerungen für optimale sLR Geraden

Theorem (Th-5.3): "SST = SSE + SSR"

Sei $y = a + b \cdot x$ optimale sLR-Gerade $\implies SST = SSE + SSR$

Beweis:

$$\begin{aligned} SST &= \sum_{i=1}^n (y_i - \bar{y})^2 \\ &= \sum_{i=1}^n (y_i - f_i + f_i - \bar{y})^2 \\ &= \sum_{i=1}^n [(y_i - f_i)^2 + 2(y_i - f_i)(f_i - \bar{y}) + (f_i - \bar{y})^2] \\ &= \sum_{i=1}^n (y_i - f_i)^2 + \sum_{i=1}^n (f_i - \bar{y})^2 + 2 \sum_{i=1}^n (y_i - f_i)(f_i - \bar{y}) \end{aligned}$$

5 LINEARE REGRESSION (LR) IN ML

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

Beachte, dass der mittlere Ausdruck $\sum_{i=1}^n (f_i - \bar{y})^2$ der SSR entspricht und der letzte Ausdruck $2 \sum_{i=1}^n (y_i - f_i)(f_i - \bar{y})$ (*) durch algebraische Umformungen gleich null wird (da die Residuen und die Abweichungen vom Mittelwert orthogonal zueinander sind). Der Beweis von (*) ist eine Übungsaufgabe.

Somit bleibt übrig:

$$SST = SSR + SSE$$

q.e.d

Wenn man viele Beispiele rechnet könnte man vermuten, dass auch die Umkehrung gilt. Diese ist jedoch nicht trivialerweise berechenbar.

Andererseits ist auch ein Gegenbeispiel nicht einfach zu konstruieren.

Somit kann man an dieser Stelle nur eine Vermutung aussprechen:

Vermutung (V-5.1)

Aus der Bedingung $SST = SSE + SSR$ folgt für eine sLR-Gerade, dass diese auch optimal ist in unserer Definition (d.h. $R^2 = \text{maximal}$).

Anmerkung zu einem möglichen Beweis:

Kann zur Zeit noch nicht bewiesen werden.

Auch ein Gegenbeispiel ist nicht trivial konstruierbar.

Somit bleibt diese Frage offen, bis ein Beweis gefunden ist.

Anmerkung: (V-5.1) wird als Übungsaufgabe mit (*) definiert.

Korollar (K-5.3): "Massenzentrum"

Sei $f(x) = a + b \cdot x$ eine opt. sLR-Gerade. Definiere $\bar{f} := f(\bar{x})$ dann kann man mit einfachen Umrechnungen (Matrizenrechnung) zeigen:

$$\bar{f} = \bar{y}$$

Beweis: Man kann den Beweis auf zwei Arten durchführen:

1. Beweis-Methode:

In Theorem (Th-5.3) wurde gezeigt, dass gilt: (*) $0 = \sum_{i=1}^n (y_i - f_i)(f_i - \bar{y})$

Ohne Einschränkung der Allgemeinheit (OE) kann man annehmen: $(f_i - \bar{y}) \neq 0$

5 LINEARE REGRESSION (LR) IN ML

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

dann folgt direkt $0 = \sum_{i=1}^n (y_i - f_i)$

2. Beweis-Methode:

Um den Zusammenhang zwischen den geschätzten Werten $\bar{f} = \frac{1}{n} \cdot \sum_{i=1}^n (f_i)$ und den tatsächlichen Werten \bar{y} in einer einfachen linearen Regression zu zeigen, setzen wir die "Schätzung für den Achsenabschnitt a" $=: \hat{a}$ und die "Schätzung der Steigung b" $=: \hat{b}$ ein und führen die Berechnungen durch:

$$\begin{aligned}\bar{f} &= \frac{1}{n} \cdot \sum_{i=1}^n (\hat{a} + \hat{b} \cdot x_i) \implies \bar{f} = \hat{a} + \hat{b} \cdot \bar{x} \\ (\text{sLSF-I}^*) &: \hat{a} = \bar{y} - \hat{b} \cdot \bar{x} \implies \bar{y} = \hat{a} + \hat{b} \cdot \bar{x}\end{aligned}$$

Vergleichen wir die Formeln für \bar{f} und \bar{y} , so sind diese gleich.
q.e.d.

5.4.4 Detailberechnung für mLRL(k=2)

Analog wie für die simple Lineare Regression (SLR) lässt sich das LSF-Verfahren auch auf die multiple Lineare Regression(mLR) anwenden. Für den Fall k=2 erhält man das folgende Theorem:

Theorem (Th-5.4): "LSF Komponenten für optimale mLRL(k=2)-Ebene"

Sei $det = \sum \tilde{x}_i^2 \cdot \sum \tilde{y}_i^2 - (\sum \tilde{x}_i \cdot \tilde{y}_i)^2$

Für den Achsenabschnitt a und die Steigungen b und c einer optimalen mLRL(k=2)-Ebene $z = a + b \cdot x + c \cdot y$ gelten folgende Formeln:

$$(\text{mLSF-I}): \quad a = \bar{z} - b \cdot \bar{x} - c \cdot \bar{y}$$

$$(\text{mLSF-II}): \quad b = \frac{1}{det} \cdot [\sum \tilde{y}_i^2 \cdot \sum (\tilde{x}_i \cdot \tilde{z}_i) - \sum (\tilde{x}_i \cdot \tilde{y}_i) \cdot \sum (\tilde{y}_i \cdot \tilde{z}_i)]$$

$$(\text{sLSF-III}): \quad c = \frac{1}{det} \cdot [\sum \tilde{x}_i^2 \cdot \sum (\tilde{y}_i \cdot \tilde{z}_i) - \sum (\tilde{x}_i \cdot \tilde{y}_i) \cdot \sum (\tilde{x}_i \cdot \tilde{z}_i)]$$

Beweis:

Sei die optimale Regression-Ebene gegeben durch $z = a + b \cdot x + c \cdot y$ dann muss gelten $R^2 = R^2(a, b, c)$ ist maximal $\Leftrightarrow SSE$ ist minimal. Sei $S = S(a, b, c)$ die Abkürzung für $SSE(a, b, c)$. S ist eine Funktion in den Veränderlichen a und b und c . Somit muss für ein Minimum gelten, dass der Gradient $\Delta S(a, b, c) = 0$ ist (Analysis I):

$$dS(a, b, c) = \frac{\partial S}{\partial a} \cdot da + \frac{\partial S}{\partial b} \cdot db + \frac{\partial S}{\partial c} \cdot dc = 0 \iff \frac{\partial S}{\partial a} = \frac{\partial S}{\partial b} = \frac{\partial S}{\partial c} = 0$$

5 LINEARE REGRESSION (LR) IN ML

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

Wir rechnen als erstes die **partielle Ableitung nach a** :

$$\frac{\partial S}{\partial a} = \frac{\partial}{\partial a} \left[\sum_{i=1}^n (z_i - a - b \cdot x_i - c \cdot y_i)^2 \right] = 2 \cdot \sum (z_i - a - b \cdot x_i - c \cdot y_i)$$

Durch Benutzung der Definition des Mittelwertes und Kürzen mit 2 erhält man:

$$0 = n \cdot \bar{z} - n \cdot a - n \cdot b \cdot \bar{x} - n \cdot c \cdot \bar{y}$$

Daraus folgt:

$$a = \bar{z} - b \cdot \bar{x} - c \cdot \bar{y} \quad (1)$$

Nun nutzen wir wieder den "**Bias-Trick**": Wir setzen den optimalen Bias a in die partiellen Ableitungen nach b und c ein. Damit müssen wir später keine 3x3-Matrix invertieren, sondern nur eine 2x2-Matrix. Dies führt zu Ersparnissen in den Berechnungen. Für größere $k \geq 3$ verstärkt sich sogar dieser Effekt

$$\begin{aligned} 0 &= \frac{\partial}{\partial b} \cdot \left[\sum_{i=1}^n (z_i - \bar{z} + b \cdot \bar{x} + c \cdot \bar{y} - b \cdot x_i - c \cdot y_i)^2 \right] \\ &= \frac{\partial}{\partial b} \cdot \left[\sum_{i=1}^n (\tilde{z}_i - b \cdot \tilde{x}_i - c \cdot \tilde{y}_i)^2 \right] \\ &= 2 \cdot \sum_{i=1}^n ((\tilde{z}_i - b \cdot \tilde{x}_i - c \cdot \tilde{y}_i) \cdot (-\tilde{x}_i)) \\ &= (-2) \cdot \sum_{i=1}^n ((\tilde{x}_i \cdot \tilde{z}_i + b \cdot \tilde{x}_i^2 + c \cdot \tilde{x}_i \cdot \tilde{y}_i) \end{aligned}$$

Also erhalten wir:

$$0 = \sum_{i=1}^n (\tilde{x}_i \cdot \tilde{z}_i + b \cdot \tilde{x}_i^2 + c \cdot \tilde{x}_i \cdot \tilde{y}_i) \quad (2)$$

Analog für die partielle Ableitung nach c :

$$0 = \sum_{i=1}^n (\tilde{y}_i \cdot \tilde{z}_i + b \cdot \tilde{x}_i \cdot \tilde{y}_i + c \cdot \tilde{y}_i^2) \quad (3)$$

Schreiben wir nun die beiden Gleichungen (2) und (3) mit einer 2x2-Matrix um, so erhält man nach den Regeln der Matrizenmultiplikation (siehe Vorlesung "Lineare

5 LINEARE REGRESSION (LR) IN ML

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

Algebra"):

$$\begin{bmatrix} \sum \tilde{x}_i^2 & \sum (\tilde{x}_i \cdot \tilde{y}_i) \\ \sum (\tilde{x}_i \cdot \tilde{y}_i) & \sum \tilde{y}_i^2 \end{bmatrix} \cdot \begin{bmatrix} b \\ c \end{bmatrix} = \begin{bmatrix} \sum (\tilde{x}_i \cdot \tilde{z}_i) \\ \sum (\tilde{y}_i \cdot \tilde{z}_i) \end{bmatrix}$$

In der Lin. Algebra wird die Inverse einer 2x2- Matrix: $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ berechnet als:

$$A^{-1} = \frac{1}{ad-bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

wobei Determinante(A) = $\det A = ad - bc$. Mit obigen Werten eingesetzt erhält man:

$$\det = \sum \tilde{x}_i^2 \cdot \sum \tilde{y}_i^2 - (\sum \tilde{x}_i \cdot \tilde{y}_i)^2 \quad (4)$$

$$\begin{bmatrix} b \\ c \end{bmatrix} = \frac{1}{\det} \cdot \begin{bmatrix} \sum \tilde{y}_i^2 & -(\sum (\tilde{x}_i \cdot \tilde{y}_i)) \\ -(\sum (\tilde{x}_i \cdot \tilde{y}_i)) & \sum \tilde{x}_i^2 \end{bmatrix} \begin{bmatrix} \sum (\tilde{x}_i \cdot \tilde{z}_i) \\ \sum (\tilde{y}_i \cdot \tilde{z}_i) \end{bmatrix} \quad (5)$$

Durch Auflösen der Gleichung (5) erhalten wir zusammen mit (1) und (4) die Aussagen von Theorem (Th-5.4)
q.e.d.

5.4.5 Weitere Folgerungen für opt. mLR(k=2)-Ebene

Korollar (K-5.4): "Massenzentrum"

Sei $f(x) = a + b \cdot x + c \cdot y$ eine opt. mLR-Ebene. Definiere $\bar{f} := f(\bar{x}, \bar{y})$ dann kann man mit einfachen Umrechnungen (Matrizenrechnung) zeigen:

$$\bar{f} = \bar{z}$$

Beweis: Man kann den Beweis analog wie bei sLR durchführen.

In Theorem (Th-5.4) wurde gezeigt, dass gilt: (1) $\bar{z} = a + b \cdot \bar{x} + c \cdot \bar{y} = f(\bar{x}, \bar{y})$
q.e.d.

5 LINEARE REGRESSION (LR) IN ML

5.5 Allgemeine Berechnungen von LSF für LR ($k \geq 1$)

***** Weitere analoge Aussagen wie bei sLR-Geraden formulieren (i.e. "SST = SSE + SSR?" etc. ... *****

5.5 Allgemeine Berechnungen von LSF für LR ($k \geq 1$)

Die Notation ist analog zu den obigen Definitionen bei $adj.R^2$:

In einem **Linearen Regression (LR) 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 $i \in \{1, \dots, n\}$.

Die Zahl k ist die Anzahl der unabhängigen **Features ("Prädiktoren")** $\mathbf{x}^{(i)}$. Weiterhin nennen wir Zielvariable $y^{(i)}$ abhängige **Zielvariable ("target column")**. Die Datenpaare bilden für das LR Problem die **Trainings-Menge**. Unser Ziel ist es die folgende Funktion:

$$f : \mathbb{R}^k \rightarrow \mathbb{R} \text{ zu berechnen, so dass}$$

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

$$\forall i \in \{1, \dots, n\} : f(\mathbf{x}^{(i)}) \approx y^{(i)}.$$

Um die Gleichung $f(\mathbf{x}^{(i)}) \approx y^{(i)}$ zu berechnen nutzen wir wieder die **Summe der quadratischen Fehler "Sum of Squared Errors"**:

$$SSE := \sum_{i=1}^n (f(\mathbf{x}^{(i)}) - y^{(i)})^2. \quad (1)$$

Für eine Liste von Trainingsdaten $[\langle \mathbf{x}^{(1)}, y^{(1)} \rangle, \dots, \langle \mathbf{x}^{(n)}, y^{(n)} \rangle]$, haben wir das Ziel der Minimierung von **SSE**.

5.5.1 Matrixdarstellung von SSE

Um dies zu erreichen brauchen wir eine Modell für die Funktion f . Das einfachste Modell ist das lineare Modell, i.e. wir nehmen an das f gegeben ist durch:

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

Hier bezeichnet der Ausdruck $\mathbf{x}^\top \cdot \mathbf{w}$ das Matrix Produkt des Vektors \mathbf{x}^\top , welcher eine 1-zu- k Matrix ist, mit dem Vektor \mathbf{w} . Alternativ könnte man diesen Ausdruck auch als das Punktprodukt des Vektors \mathbf{x} und des Vektors \mathbf{w} interpretieren.

5 LINEARE REGRESSION (LR) IN ML

5.5 Allgemeine Berechnungen von LSF für LR ($k \geq 1$)

An dieser Stelle werden Sie sich vielleicht fragen, warum es sinnvoll ist, hier die Matrixschreibweise einzuführen. Der Grund ist dass diese Notation die Formel verkürzt und darüber hinaus effizienter zu implementieren ist, da die meisten Programmiersprachen, die im Bereich des maschinellen Lernens verwendet werden, über spezielle Bibliotheksunterstützung für Matrixoperationen verfügen.

Sofern der Computer mit einer Grafikkarte ausgestattet ist, wären einige Programmiersprachen sogar in der Lage, Matrixoperationen an die Grafikeinheit zu delegieren. Dies führt zu einer erheblichen Geschwindigkeitszuwachs.

Die oben angegebene Definition von f ist das Modell, das in der [linearen Regression](#) genutzt wird. Führen wir \mathbf{w} als der [Gewichtsvektor](#) und b als [Bias](#) ein, so erhalten wir in vereinfachter Matrixschreibweise:

$$f(\mathbf{x}) = \mathbf{x}^\top \cdot \mathbf{w} + b.$$

Als erstes wollen wir den Bias b berechnen. Wir Arbeiten dabei mit der umgeschriebenen Gleichung von (1):

$$\text{SSE}(\mathbf{w}) = \sum_{i=1}^n \left((\mathbf{x}^{(i)})^\top \cdot \mathbf{w} + b - y^{(i)} \right)^2 \quad (2)$$

Wir berechnen die partielle Ableitung $\frac{\partial}{\partial b}(\text{MSE})$ und setzen diese Null. In Matrixschreibweise erhalten wir ann :

$$0 = \frac{\partial}{\partial b}(\text{SSE}(\mathbf{w})) = \sum_{i=1}^n \left((\mathbf{x}^{(i)})^\top \cdot \mathbf{w} + b - y^{(i)} \right) \quad (3)$$

Lösen wir die Gleichung nach b auf und kürzen mit n , so erhalten wir mit der Matrixschreibweise:

$$b = \bar{\mathbf{y}} - \mathbf{w} \cdot \bar{\mathbf{x}}^\top \quad (4)$$

Setzen wir dies in die Definition von $\text{SSE}(\mathbf{w})$ ein und nutzen die uns schon bekannte **Notation** ($\tilde{x}_i := (x_i - \bar{x})$ und $\tilde{y}_i := (y_i - \bar{y})$).

Nun nutzen wir wieder den **"Bias-Trick"**: Wir setzen den optimalen Bias b in die partiellen Ableitungen der Gewichte w ein. Damit müssen wir später keine $(k+1) \times (k+1)$ -Matrix invertieren, sondern nur eine $k \times k$ -Matrix. Dies führt zu Vereinfachung der Formeln und Ersparnissen in den Berechnungen. So erhalten wir:

$$\text{SSE}(\mathbf{w}) = \sum_{i=1}^n \left((\widetilde{\mathbf{x}}^{(i)})^\top \cdot \mathbf{w} - \widetilde{y}^{(i)} \right)^2 \quad (5)$$

5 LINEARE REGRESSION (LR) IN ML

5.5 Allgemeine Berechnungen von LSF für LR ($k \geq 1$)

Beachten Sie, dass $\mathbf{y} \in \mathbb{R}^n$ enthalten ist, da es eine Komponente für alle n Trainings Beispiele hat. Als nächstes definieren wir die **Designmatrix** X wie folgt:

$$X := \begin{pmatrix} (\widetilde{\mathbf{x}}^{(1)})^\top \\ \vdots \\ (\widetilde{\mathbf{x}}^{(n)})^\top \end{pmatrix} \quad (A)$$

In der Literatur heisst X auch oft **Feature Matrix**. X ist eine (nxk) -Matrix. So definiert sind die Zeilenvektoren der Matrix X die transponierten Vektoren $\mathbf{x}^{(i)}$ mit $1 \leq i \leq n$ wobei $n = \text{Anzahl der Trainingstupel}$. Schreiben wir die einzelnen Komponenten (Skalare) der (nxk) -Matrix X und (kxn) -Matrix X^\top aus, so ergibt sich:

$$X := \begin{pmatrix} \widetilde{x}_1^{(1)} & \dots & \widetilde{x}_k^{(1)} \\ \vdots & & \vdots \\ \widetilde{x}_1^{(n)} & \dots & \widetilde{x}_k^{(n)} \end{pmatrix}; \quad X^\top := \begin{pmatrix} \widetilde{x}_1^{(1)} & \dots & \widetilde{x}_1^{(n)} \\ \vdots & & \vdots \\ \widetilde{x}_k^{(1)} & \dots & \widetilde{x}_k^{(n)} \end{pmatrix} \quad (A')$$

Nun haben wir folgendes:

$$X \cdot \mathbf{w} - \tilde{\mathbf{y}} = \begin{pmatrix} (\widetilde{\mathbf{x}}^{(1)})^\top \\ \vdots \\ (\widetilde{\mathbf{x}}^{(n)})^\top \end{pmatrix} \cdot \mathbf{w} - \tilde{\mathbf{y}} = \begin{pmatrix} (\widetilde{\mathbf{x}}^{(1)})^\top \cdot \mathbf{w} - \widetilde{y}^{(1)} \\ \vdots \\ (\widetilde{\mathbf{x}}^{(n)})^\top \cdot \mathbf{w} - \widetilde{y}^{(n)} \end{pmatrix}$$

Wenn wir das Quadrat des Vektors $X \cdot \mathbf{w} - \tilde{\mathbf{y}}$ nehmen, stellen wir fest, dass wir die Gleichung (2) wie folgt umschreiben können:

$$\text{SSE}(\mathbf{w}) = (X \cdot \mathbf{w} - \tilde{\mathbf{y}})^\top \cdot (X \cdot \mathbf{w} - \tilde{\mathbf{y}}) \quad (6)$$

5.5.2 Berechnung des Gradienten zu SSE

Um das minimale $\text{SSE}(\mathbf{w})$ zu bekommen, sagt uns die Mathematik (Analysis I), müssen wir den "Gradienten" der Funktion $\text{SSE}(\mathbf{w})$ null setzen. Die folgenden Berechnungen dienen dazu. Nützlich dazu ist die Matrizenrechnung aus der Linearen Algebra.

Im letzten Abschnitt haben wir den mittleren quadratischen Fehler $\text{MSE}(\mathbf{w})$ anhand der Gleichung (3) berechnet.

Unser Ziel ist die Minimierung des $\text{SSE}(\mathbf{w})$ durch die Wahl des geeigneten Gewichtsvektors Vektors \mathbf{w} . Eine notwendige Bedingung, damit $\text{SSE}(\mathbf{w})$ minimal ist, ist:

5 LINEARE REGRESSION (LR) IN ML

5.5 Allgemeine Berechnungen von LSF für LR ($k \geq 1$)

$$\nabla \text{SSE}(\mathbf{w}) = \mathbf{0},$$

d.h. der **Gradient** von $\text{SSE}(\mathbf{w})$ in Bezug auf \mathbf{w} muss Null sein.

Zur Vorbereitung der Berechnung von $\nabla \text{SSE}(\mathbf{w})$, berechnen wir zunächst den Gradienten von zwei einfacheren Funktionen.

******* Zwischenrechnung *******

Angenommen die Funktion $f : \mathbb{R}^n \rightarrow \mathbb{R}$ ist definiert als:

$f(\mathbf{x}) := \mathbf{x}^\top \cdot C \cdot \mathbf{x}$ where $C \in \mathbb{R}^{n \times n}$ Wenn wir die Matrix C als $C = (c_{i,j})_{\substack{i=1, \dots, n \\ j=1, \dots, n}}$ schreiben und den Vektor \mathbf{x} als $\mathbf{x} = \langle x_1, \dots, x_n \rangle^\top$, dann kann $f(\mathbf{x})$ wie folgt berechnet werden:

$$f(\mathbf{x}) = \sum_{i=1}^n x_i \cdot \sum_{j=1}^n c_{i,j} \cdot x_j = \sum_{i=1}^n \sum_{j=1}^n x_i \cdot c_{i,j} \cdot x_j.$$

Wir berechnen die partielle Ableitung von f nach x_k und verwenden die Produktregel zusammen mit der Definition des **Kronecker-Delta** $\delta_{i,j}$, das als 1 definiert ist, wenn $i = j$ und sonst als 0:

$$\delta_{i,j} := \begin{cases} 1 & \text{if } i = j; \\ 0 & \text{andernfalls.} \end{cases}$$

Dann wird die partielle Ableitung von f nach x_k , die als $\frac{\partial f}{\partial x_k}$ geschrieben wird, wie folgt berechnet:

$$\begin{aligned} \frac{\partial f}{\partial x_k} &= \sum_{i=1}^n \sum_{j=1}^n \left(\frac{\partial x_i}{\partial x_k} \cdot c_{i,j} \cdot x_j + x_i \cdot c_{i,j} \cdot \frac{\partial x_j}{\partial x_k} \right) \\ &= \sum_{i=1}^n \sum_{j=1}^n \left(\delta_{i,k} \cdot c_{i,j} \cdot x_j + x_i \cdot c_{i,j} \cdot \delta_{j,k} \right) \\ &= \sum_{j=1}^n c_{k,j} \cdot x_j + \sum_{i=1}^n x_i \cdot c_{i,k} \cdot c_{i,k} \\ &= (C \cdot \mathbf{x})_k + (C^\top \cdot \mathbf{x})_k \end{aligned}$$

Wir haben also gezeigt, dass

$$\nabla f(\mathbf{x}) = (C + C^\top) \cdot \mathbf{x}.$$

Wenn die Matrix C **symmetrisch** ist, d.h. wenn $C = C^\top$, vereinfacht sich dies zu:

$$\nabla f(\mathbf{x}) = 2 \cdot C \cdot \mathbf{x}.$$

Als nächstes, wenn die Funktion $g : \mathbb{R}^n \rightarrow \mathbb{R}$ definiert ist als

$$g(\mathbf{x}) := \mathbf{b}^\top \cdot A \cdot \mathbf{x}, \quad \text{wobei } \mathbf{b} \in \mathbb{R}^n \text{ und } A \in \mathbb{R}^{n \times n},$$

dann zeigt eine ähnliche Berechnung, dass

$$\nabla g(\mathbf{x}) = A^\top \cdot \mathbf{b}$$

Der Beweis dieser Aussage ist eine Übungsaufgabe "Übung 5.3".

***** Ende der Zwischenrechnung *****

5.5.3 Ableitung der Normalform

Als nächstes leiten wir die so genannte **Normalform** für die Lineare Regression ab. Zu diesem Zweck erweitern wir zuerst das Produkt im der Gleichung (6):

$$\begin{aligned} \text{SSE}(\mathbf{w}) &= (X \cdot \mathbf{w} - \tilde{\mathbf{y}})^\top \cdot (X \cdot \mathbf{w} - \tilde{\mathbf{y}}) \\ &= (\mathbf{w}^\top \cdot X^\top - \tilde{\mathbf{y}}^\top) \cdot (X \cdot \mathbf{w} - \tilde{\mathbf{y}}) \quad (A \cdot B)^\top = B^\top \cdot A^\top \\ &= (\mathbf{w}^\top \cdot X^\top \cdot X \cdot \mathbf{w} - \tilde{\mathbf{y}}^\top \cdot X \cdot \mathbf{w} - \tilde{\mathbf{y}}^\top \cdot X^\top \cdot \tilde{\mathbf{y}} + \tilde{\mathbf{y}}^\top \cdot \tilde{\mathbf{y}}) \\ &= (\mathbf{w}^\top \cdot X^\top \cdot X \cdot \mathbf{w} - 2 \cdot \tilde{\mathbf{y}}^\top \cdot X \cdot \mathbf{w} + \tilde{\mathbf{y}}^\top \cdot \tilde{\mathbf{y}}) \quad \mathbf{w}^\top \cdot X^\top \cdot \tilde{\mathbf{y}} = \tilde{\mathbf{y}}^\top \cdot X \cdot \mathbf{w} \end{aligned}$$

Die Tatsache das gilt

$$\mathbf{w}^\top \cdot X^\top \cdot \tilde{\mathbf{y}} = \tilde{\mathbf{y}}^\top \cdot X \cdot \mathbf{w}$$

mag auf den ersten Blick nicht offensichtlich sein. Aber dies folgt aus zwei Fakten:

- (a) Für zwei Matrizen A and B wo das Matrix Produkt $A \cdot B$ ist definiert ist, gilt

$$(A \cdot B)^\top = B^\top \cdot A^\top.$$
- (b) Das Matrix Produkt $\mathbf{w}^\top \cdot X^\top \cdot \mathbf{y}$ ist eine reelle Zahl r . Das Transponierte r^\top einer reellen Zahl ist selbst wieder eine reelle Zahl r , i.e. $r^\top = r$ for all $r \in \mathbb{R}$.

Also haben wir letztendlich gezeigt, dass

$$\text{SSE}(\mathbf{w}) = (\mathbf{w}^\top \cdot X^\top \cdot X \cdot \mathbf{w} - 2 \cdot \tilde{\mathbf{y}}^\top \cdot X \cdot \mathbf{w} + \tilde{\mathbf{y}}^\top \cdot \tilde{\mathbf{y}}) \quad (7)$$

gilt. Die benutzte Matrix $X^\top \cdot X$ im ersten Term ist symmetrisch weil

$$(X^\top \cdot X)^\top = X^\top \cdot (X^\top)^\top = X^\top \cdot X.$$

5 LINEARE REGRESSION (LR) IN ML

5.5 Allgemeine Berechnungen von LSF für LR ($k \geq 1$)

Benutzen wir die Resultate der vorherigen Abschnitte so können wir nun den Gradienten von $\text{MSE}(\mathbf{w})$ mit Bezug auf \mathbf{w} . berechnen zu

$$\nabla \text{MSE}(\mathbf{w}) = X^\top \cdot X \cdot \mathbf{w} - X^\top \cdot \tilde{\mathbf{y}}.$$

Falls der quadratische Fehler $\text{SSE}(\mathbf{w})$ ein Minimum für die Gewichte \mathbf{w} hat, dann gilt

$$\nabla \text{SSE}(\mathbf{w}) = \mathbf{0}.$$

Dies führt zur Gleichung

$$X^\top \cdot X \cdot \mathbf{w} - X^\top \cdot \tilde{\mathbf{y}} = \mathbf{0}.$$

Dies kann umgeschrieben werden zu

$$(X^\top \cdot X) \cdot \mathbf{w} = X^\top \cdot \tilde{\mathbf{y}}$$

(8)

Diese Gleichung nennt man **Normalform**. Nutzen wir die Gleichungen (A') so können wir jetzt leicht per Matrizen-Multiplikation die Komponenten der ($k \times k$)-Produktmatrix $X^\top \cdot X$ und den ($1 \times k$)-Vektor $X^\top \cdot \tilde{\mathbf{y}}$ berechnen. Damit sind alle Faktoren der **Normalform** bekannt:

$$X^\top \cdot X := \begin{pmatrix} \sum_{i=1}^n [\tilde{x}_1^{(i)}]^2 & \dots & \sum_{i=1}^n [\tilde{x}_1^{(i)}] \cdot \tilde{x}_k^{(i)} \\ \vdots & & \vdots \\ \sum_{i=1}^n [\tilde{x}_k^{(i)}] \cdot \tilde{x}_1^{(i)} & \dots & \sum_{i=1}^n [\tilde{x}_k^{(i)}]^2 \end{pmatrix} \quad X^\top \cdot \tilde{\mathbf{y}} := \begin{pmatrix} \sum_{i=1}^n [\tilde{x}_1^{(i)}] \cdot \tilde{y}^{(i)} \\ \vdots \\ \sum_{i=1}^n [\tilde{x}_k^{(i)}] \cdot \tilde{y}^{(i)} \end{pmatrix} \quad (B)$$

Wie man leicht an der Matrix $X^\top \cdot X$ erkennt, ist die Matrix symmetrisch, da das Produkt zweier reellen Zahlen kommutativ ist.

Angenommen die Matrix $X^\top \cdot X$ ist invertierbar, so lässt sich obige Matrix umschreiben zu:

$$\mathbf{w} = (X^\top \cdot X)^{-1} \cdot X^\top \cdot \tilde{\mathbf{y}}.$$

Insgesamt erhält man das folgende Theorem:

Theorem (Th-5.5): "Bias und Gewichte bei LR ($k \geq 1$)"

5 LINEARE REGRESSION (LR) IN ML

5.5 Allgemeine Berechnungen von LSF für LR ($k \geq 1$)

Für $k=1$ (opt. sLR-Geraden), $k=2$ (opt. mLR-Ebenen) und $k \geq 3$ (opt. mLR-Hyperebenen) lassen sich "Bias" und "Gewichte" durch die folgenden zwei Gleichungen bestimmen:

$$(\text{LSF-I}): \quad b = \bar{\mathbf{y}} - \mathbf{w} \cdot \bar{\mathbf{x}}^\top$$

$$(\text{LSF-II}): \quad \mathbf{w} = (X^\top \cdot X)^{-1} \cdot X^\top \cdot \tilde{\mathbf{y}}$$

Anmerkung (A-5.1): Gleichung (LSF-II) gilt nur falls $(X^\top \cdot X)$ invertierbar⁴ ist. Dies ist eine wichtige Eigenschaft von invertierbaren Matrizen. $\Leftrightarrow \det(X^\top \cdot X) \neq 0$ ist. Ist dies nicht der Fall lösen wir die Normalform (8) auf.

Beweis:

Siehe obige Berechnungen.
q.e.d.

Bemerkung: Obwohl die ($k \times k$)-Matrix $X^\top \cdot X$ oft invertierbar ist, können wir (wenn dies nicht zutrifft) auch die ursprüngliche Gleichung : $(X^\top \cdot X) \cdot \mathbf{w} = X^\top \cdot \tilde{\mathbf{y}}$ lösen.

In der Tat werden wir wenn wir die Normaform auflösen die *Python* Funktion `numpy.linalg.solve(A, b)` nutzen, welche die Matrix $A \in \mathbb{R}^{n \times n}$ und den Vektor $\mathbf{b} \in \mathbb{R}^n$ berechnet mit der folgenden Gleichung:

$$A \cdot \mathbf{x} = \mathbf{b}.$$

Korollar (K-5.5):

Für $k=1$ und $k=2$ folgen die Theoreme (Th-5.2) und (Th-5.4) direkt aus dem Theorem (Th-5.5).

Beweis:

Einsetzen der Definitionen in die zwei Formeln von (Th-5.5) für beide Fälle.

Hinweis: Berechne als erstes die ($k \times k$)-Matrix $X^\top \cdot X$ und danach den ($k \times 1$)-Vektor $X^\top \cdot \tilde{\mathbf{y}}$

Diese Berechnungen sind als (Übung-5.8) durchzuführen. Eine komplette Lösung dazu finden sie auch im Abschnitt "Lösungshinweise zu den Übungen Kapitel 5".

Anmerkung (A-5.2): Beweis von (Th-5.5) ohne "Bias-Trick":

Wenn man ohne den Bias-Trick arbeiten will, kann man die Notation so anpassen, dass wir den k -dimensionalen Feature-Vektor zu einem $(k+1)$ -dimensionalen Vektor

⁴Matrix A ist genau dann invertierbar (auch regulär oder nicht singulär genannt), wenn ihre Determinante $\det(A)$ ungleich 0 ist

5 LINEARE REGRESSION (LR) IN ML

5.5 Allgemeine Berechnungen von LSF für LR ($k \geq 1$)

erweitern:

$$x'_j := x_j \quad \text{für alle } j \in \{1, \dots, k\} \quad \text{and} \quad x'_{k+1} := 1.$$

weiterhin definieren wir:

$$\mathbf{w}' := \langle w_1, \dots, w_k, b \rangle^\top \quad \text{wobei } \langle w_1, \dots, w_k \rangle = \mathbf{w}^\top.$$

Die Herleitung der Normalform ist dabei analog wie mit dem "Bias-Trick". Wir erhalten für die Faktoren der [Normalform](#):

$$X^\top \cdot X := \begin{pmatrix} \sum_{i=1}^n [x_1^{(i)}]^2 & \dots & \sum_{i=1}^n [x_1^{(i)} \cdot x_k^{(i)}] & \sum_{i=1}^n x_1^{(i)} \\ \vdots & & & \\ \sum_{i=1}^n [x_k^{(i)} \cdot x_1^{(i)}] & \dots & \sum_{i=1}^n [x_k^{(i)}]^2 & \sum_{i=1}^n x_k^{(i)} \\ \sum_{i=1}^n x_1^{(i)} & \dots & \sum_{i=1}^n x_k^{(i)} & n \end{pmatrix} \quad X^\top \cdot y := \begin{pmatrix} \sum_{i=1}^n [x_1^{(i)} \cdot y^{(i)}] \\ \vdots \\ \sum_{i=1}^n [x_k^{(i)} \cdot y^{(i)}] \\ \sum_{i=1}^n y^{(i)} \end{pmatrix} \quad (B')$$

Wie man leicht an der Matrix $X^\top \cdot X$ erkennt, ist die Matrix symmetrisch, da das Produkt zweier reellen Zahlen kommutativ ist.

Anwendung für den Fall $k=1$ (simple lineare Regression)

Einsetzen der beiden Formeln (B') in die aus in die Formeln von (Th-5.5) ergibt die Ergebnisse des Theorem (Th-5.2) für die simple lineare Regression.

Hinweis: Die konkrete Berechnungen sind als (Übung-5.9) durchzuführen. Eine komplette Lösung dazu finden sie auch im Abschnitt "Lösungshinweise zu den Übungen Kapitel 5".

Anwendung für den Fall $k=2$ (mult. lineare Regression ($k=2$))

Analog wie oben bei $k=1$ ist hier vorzugehen. Die einzige Schwierigkeit ist die Berechnung der Inversen zu einer (3x3)-Matrix, da hier etwas längliche Ausdrücke zu berechnen sind. Hier zeigt sich auch der Vorteil des "Bias-Trick", da hier nur eine (2x2)-Matrix zu invertieren ist. Wir benutzen hier die bekannten Formeln aus der Linearen Algebra. Am Ende erhalten wir die Formeln des Theorem (Th-5.4).

Hinweis: Die konkrete Berechnungen sind als (Übung-5.9) durchzuführen. Eine komplette Lösung dazu finden sie auch im Abschnitt "Lösungshinweise zu den Übungen Kapitel 5".

q.e.d.

5.6 simple Lineare Regression (sLR) Beispiele

Um ein besseres Gefühl für die lineare Regression zu bekommen, wollen wir sie noch einige Beispiel mit konkreten Zahlen berechnen. Erstes manuell und dann auch noch mit Hilfe der Scikit-learn Software Bibliothek zum maschinellen Lernen für die Programmiersprache Python.

5.6.1 Manuelle Berechnungen zu einem sLR Beispiel

Bei der manuellen Berechnung für einfache Beispiele mit wenigen Trainingspunkten entwickeln wir einen Excel-Tabelle, wo wir die wichtigsten Parameter/Komponenten für die Berechnung von der optimalen sLR-Geraden und dem Bestimmungsgrad R^2 per Tabellen-Kalkulation bestimmen lassen.

1. Berechne opt. Regressionsgerade:

Nach (Th-5.2): Mit $\det = \sum x_i^2 - n \cdot \bar{x}^2$ gelten für den Achsenabschnitt a und die Steigung b :

$$a = \frac{1}{\det} \cdot (\bar{y} \cdot \sum x_i^2 - \bar{x} \cdot \sum x_i \cdot y_i) \text{ und } b = \frac{1}{\det} \cdot (\sum x_i \cdot y_i - n \cdot \bar{x} \cdot \bar{y})$$

Also brauchen wir: n , \bar{x} , \bar{y} , $n \cdot \bar{x} \cdot \bar{y}$, \bar{x}^2 , $\sum x_i^2$ und $\sum (x_i \cdot y_i)$

Alle diese Werte werden nun in die Excel-Tabelle eingetragen. Siehe die blauen Felder im nächsten Bild.

2. Berechne R^2 :

Nachdem die opt. Regressionsgerade $y(x)$ berechnet ist lassen sich auch die Werte SSE und SST berechnen. Siehe grünen Felder im nächsten Bild:

5 LINEARE REGRESSION (LR) IN ML

5.6 simple Lineare Regression (sLR) Beispiele

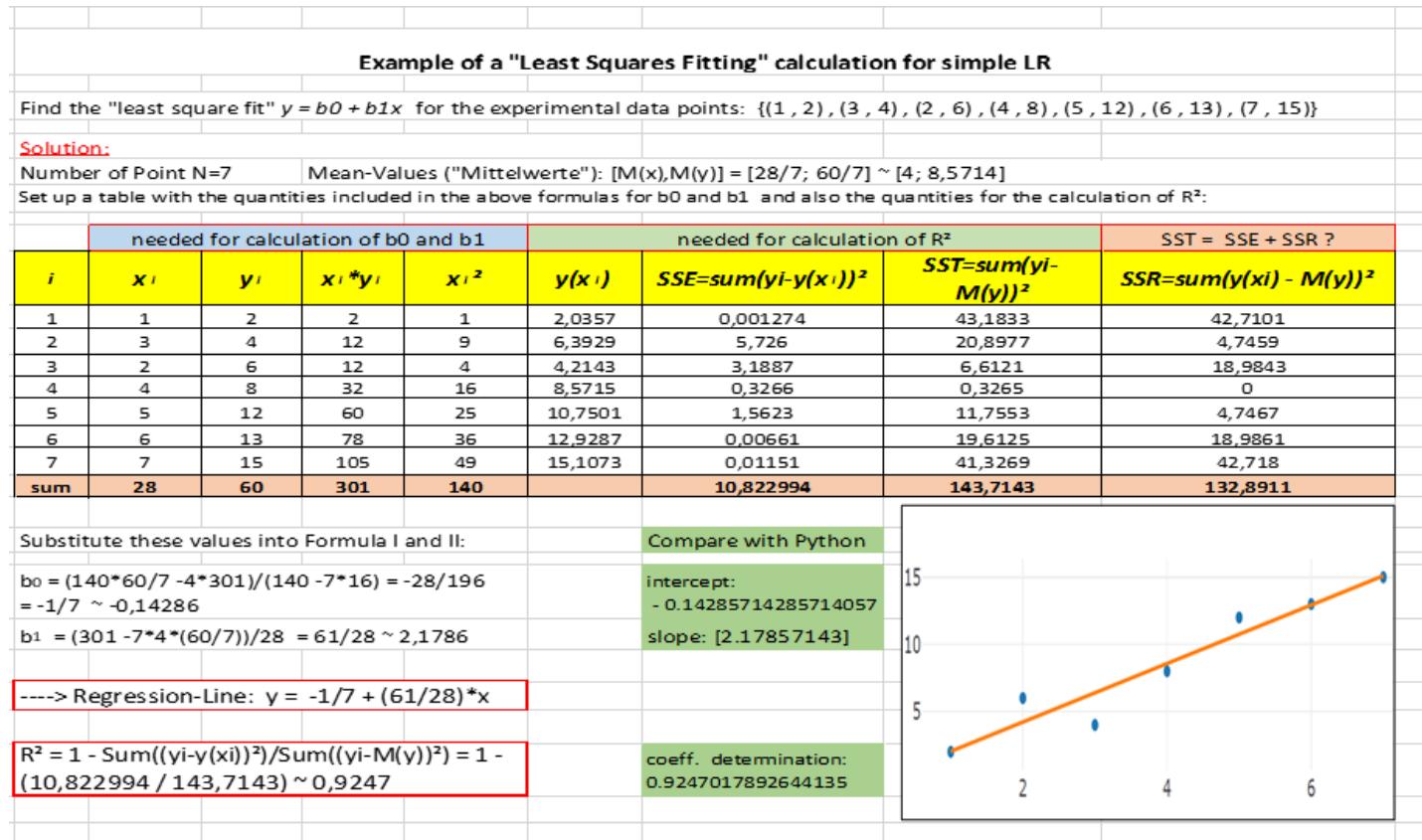


Figure 7: Manuelle Berechnung eines sLR Beispiels

5 LINEARE REGRESSION (LR) IN ML

5.6 simple Lineare Regression (sLR) Beispiele

5.6.2 Python-Programm zur Berechnung eines sLR Beispiels

In den folgenden 3 Screenshots eines Jupyter Notebooks sind die expliziten Python Codeblöcke zur Berechnung eines weiteren sLR Beispiels aufgelistet.

1. Vorbereitung:

Notwendige Python Bibliotheken zur Verfügung stellen, insbesondere Scikit-learn.

Following ideas from: "Linear Regression in Python" by Mirko Stojiljkovic, 28.4.2020 (see details: <https://realpython.com/linear-regression-in-python/#what-is-regression>)

There are **five basic steps** when you're implementing linear regression:

1. Import the packages and classes you need.
2. Provide data to work with and eventually do appropriate transformations.
3. Create a regression model and fit it with existing data.
4. Check the results of model fitting to know whether the model is satisfactory.
5. Apply the model for predictions. These steps are more or less general for most of the regression approaches and implementations.

Step 1: Import packages and classes

The first step is to import the package numpy and the class LinearRegression from sklearn.linear_model:

```
In [1]: # Step 1: Import packages and classes
import numpy as np
from sklearn.linear_model import LinearRegression
```

Now, you have all the functionalities you need to implement linear regression.

The fundamental data type of NumPy is the array type called numpy.ndarray. The rest of this article uses the term array to refer to instances of the type numpy.ndarray.

The class sklearn.linear_model.LinearRegression will be used to perform linear and polynomial regression and make predictions accordingly.

Figure 8: sLR-Beispiel-Python-1/3

2. Daten und Modell:

Die sechs Datenpunkte werden definiert (Abm. andere wie im manuellen Beispiel). Das Modell LinearRegression() aus der Scikit Bibliothek wird aktiviert.

5 LINEARE REGRESSION (LR) IN ML

5.6 simple Lineare Regression (sLR) Beispiele

Step 2: Provide data

The second step is defining data to work with. The inputs (regressors, x) and output (predictor, y) should be arrays (the instances of the class `numpy.ndarray`) or similar objects. This is the simplest way of providing data for regression:

```
In [2]: # Step 2: Provide data  
x = np.array([ 5, 15, 25, 35, 45, 55]).reshape((-1, 1))  
y = np.array([ 5, 20, 14, 32, 22, 38])
```

Now, you have two arrays: the input x and output y . You should call `.reshape()` on x because this array is required to be two-dimensional, or to be more precise, to have one column and as many rows as necessary. That's exactly what the argument `(-1, 1)` of `.reshape()` specifies.

```
In [3]: print ("This is how x and y look now:")  
print("x=",x)  
print("y=",y)
```

Step 3: Create a model and fit it

The next step is to create a linear regression model and fit it using the existing data. Let's create an instance of the class `LinearRegression`, which will represent the regression model:

```
In [4]: model = LinearRegression()
```

This statement creates the variable `model` as the instance of `LinearRegression`. You can provide several optional parameters to `LinearRegression`:

---> `fit_intercept` is a Boolean (True by default) that decides whether to calculate the intercept b_0 (True) or consider it equal to zero (False).
---> `normalize` is a Boolean (False by default) that decides whether to normalize the input variables (True) or not (False).
---> `copy_X` is a Boolean (True by default) that decides whether to copy (True) or overwrite the input variables (False).
---> `n_jobs` is an integer or None (default) and represents the number of jobs used in parallel computation. None usually means one job and -1 to use all processors.

This example uses the default values of all parameters.

It's time to start using the model. First, you need to call `.fit()` on `model`:

```
In [5]: model.fit(x, y)  
Out[5]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

Figure 9: sLR-Beispiel-Python-2/3

5 LINEARE REGRESSION (LR) IN ML

5.6 simple Lineare Regression (sLR) Beispiele

3. Ausgabe der Ergebnisse + Datenpunkte:

Bestimmungsgrad R^2 und Achsenabschnitt a und Steigung b der Regressionsgeraden werden ausgegeben. Für die sechs Datenpunkte werden auch die Werte $y(x_i)$ berechnet.

Step 4: Get results

Once you have your model fitted, you can get the results to check whether the model works satisfactorily and interpret it.

You can obtain the coefficient of determination (R^2) with `.score()` called on model:

```
In [7]: r_sq = model.score(x, y)
print('coefficient of determination:', r_sq)

coefficient of determination: 0.7158756137479542
```

When you're applying `.score()`, the arguments are also the predictor x and regressor y , and the return value is R^2 .

The attributes of model are `.intercept_`, which represents the coefficient, b_0 and `.coef_`, which represents b_1 :

```
In [8]: print('intercept:', model.intercept_)
print('slope:', model.coef_)

intercept: 5.633333333333329
slope: [0.54]
```

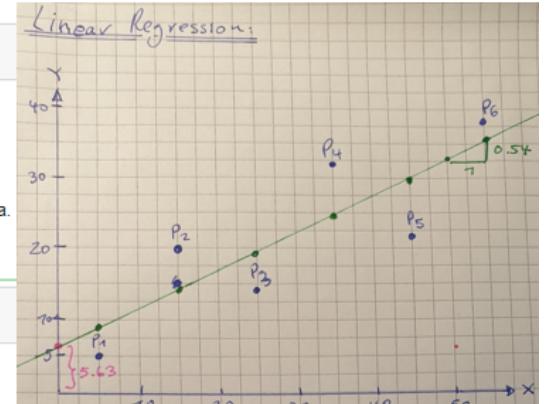
Step 5: Predict response

Once there is a satisfactory model, you can use it for predictions with either existing or new data.

To obtain the predicted response, use `.predict()`:

```
In [10]: y_pred = model.predict(x)
print('predicted response:', y_pred, sep='\n')

predicted response:
[ 8.33333333 13.73333333 19.13333333 24.53333333 29.93333333 35.33333333]
```



When applying `.predict()`, you pass the regressor as the argument and get the corresponding predicted response.

Figure 10: sLR-Beispiel-Python-3/3

5 LINEARE REGRESSION (LR) IN ML

5.7 multiple Lineare Regression mLR($k=2$) Beispiele)

5.7 multiple Lineare Regression mLR($k=2$) Beispiele)

5.7.1 Manuelle Berechnungen zu einem mLR($k=2$) Beispiel

Bei der manuellen Berechnung für einfache Beispiele mit wenigen Trainingspunkten entwickeln wir einen Excel-Tabelle, wo wir die wichtigsten Parameter/Komponenten für die Berechnung von der optimalen mLR($k=2$)-Ebene und dem Bestimmungsgrad $adj.R^2$ per Tabellen-Kalkulation bestimmen lassen.

1. Berechne opt. Regressionsgerade:

Nach (Th-5.4): Sei $det = \sum \tilde{x}_i^2 \cdot \sum \tilde{y}_i^2 - (\sum \tilde{x}_i \cdot \tilde{y}_i)^2$

Für den Achsenabschnitt a und die Steigungen b und c einer optimalen mLR($k=2$)-Ebene $z = a + b \cdot x + c \cdot y$ gelten folgende Formeln:

$$a = \bar{z} - b \cdot \bar{x} - c \cdot \bar{y}$$

$$b = \frac{1}{det} \cdot [\sum \tilde{y}_i^2 \cdot \sum (\tilde{x}_i \cdot \tilde{z}_i) - \sum (\tilde{x}_i \cdot \tilde{y}_i) \cdot \sum (\tilde{y}_i \cdot \tilde{z}_i)]$$

$$c = \frac{1}{det} \cdot [\sum \tilde{x}_i^2 \cdot \sum (\tilde{y}_i \cdot \tilde{z}_i) - \sum (\tilde{x}_i \cdot \tilde{y}_i) \cdot \sum (\tilde{x}_i \cdot \tilde{z}_i)]$$

Also brauchen wir: n , $\bar{x}, \bar{y}, \bar{z}$, $\sum \tilde{x}_i^2, \sum \tilde{y}_i^2$, $\sum (\tilde{x}_i \cdot \tilde{y}_i), \sum (\tilde{x}_i \cdot \tilde{z}_i), \sum (\tilde{y}_i \cdot \tilde{z}_i)$

Alle diese Werte werden nun in die Excel-Tabelle eingetragen. Siehe die blauen Felder im nächsten Bild.

2. Berechne $adj.R^2$:

Nachdem die opt. Regressionsebene $z(x, y)$ berechnet ist, lassen sich auch die Werte SSE und SST berechnen. Siehe grünen Felder im nächsten Bild:

5 LINEARE REGRESSION (LR) IN ML

5.7 multiple Lineare Regression mLR($k=2$) Beispiele)

Solution:

Number of Point N=8

Mean-Values ("Mittelwerte") = : [M(x), M(y), M(z)] ~ [240/8, 104/8, 178/8] = [30; 13; 22,25]

Set up a table with the quantities included in the above LSF formulas (I) and (II) for simple LR:

i	xi	yi	zi	needed for the calculation of a, b and c							
				$Xi := xi - M(x)$	$Yi := yi - M(y)$	$Zi := zi - M(z)$	$Xi * Yi$	$Xi * Zi$	$Yi * Zi$	Xi^2	Yi^2
1	0	1	4	-30	-12	-18,25	360	547,50	219,00	900	144
2	5	1	5	-25	-12	-17,25	300	431,25	207,00	625	144
3	15	2	20	-15	-11	-2,25	165	33,75	24,75	225	121
4	25	5	14	-5	-8	-8,25	40	41,25	66,00	25	64
5	35	11	32	5	-2	9,75	-10	48,75	-19,50	25	4
6	45	15	22	15	2	-0,25	30	-3,75	-0,50	225	4
7	55	34	38	25	21	15,75	525	393,75	330,75	625	441
8	60	35	43	30	22	20,75	660	622,50	456,50	900	484
Sum	240	104	178	0	0	0,00	2070	2115,00	1284,00	3550	1406

Substitute the values to the formulas (I) and (II) of LSF for mLR:

$det = sum(Xi^2) * sum(Yi^2) - (sum(Xi * Yi))^2 = 3550 * 1406 - (2070)^2 = 706400$

Compare with Python-Pgm (next slides):

$a = Mean(z) - b * Mean(x) - c * Mean(y) \sim 22,25 - 0,4471 * 30 + 0,25500 * 13 \sim 5,522$

Intercept: 5.52257927519819

$b = (1/det) * (sum(Yi^2) * sum(XiZi) - sum(XiYi) * sum(YiZi)) = (1/det) * (1406 * 2115 - 2070 * 1284) \sim 0,4471$

coefficients: [0.44706965 0.25502548]

$c = (1/det) * (sum(Xi^2) * sum(YiZi) - sum(XiYi) * sum(XiZi)) = (1/det) * (3550 * 1284 - 2070 * 2115) \sim 0,2550$

So we get the optimal mLR line: $z = 5,522 + 0,4471 * x + 0,255 * y$

$R^2 = 1 - SSE/SST \sim 0,86159$ (details see notes-page)

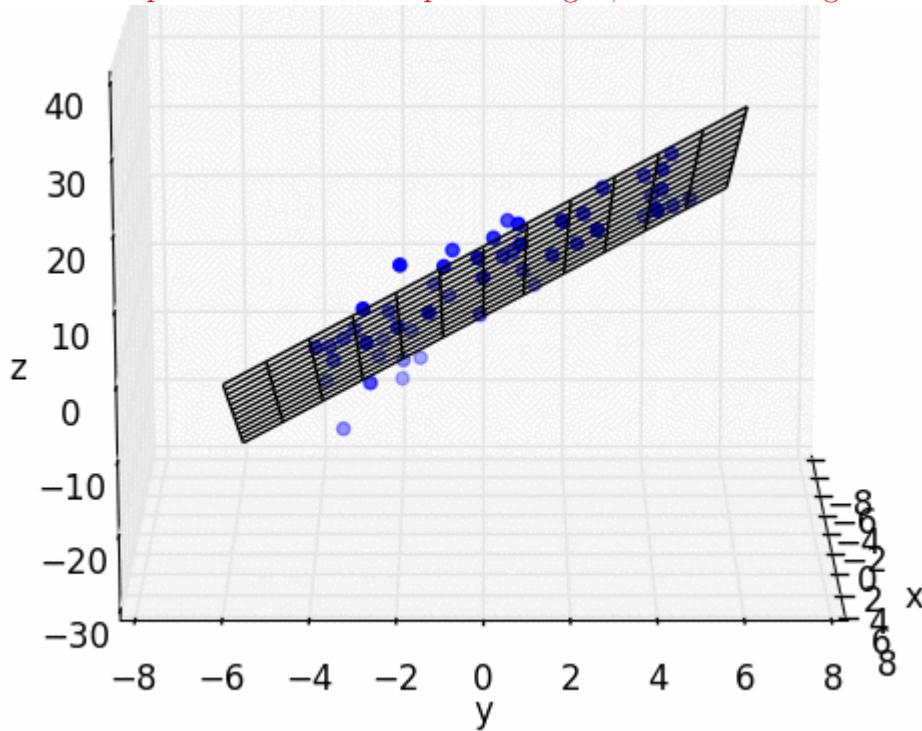
coefficient of determination: 0.8615939258756776

--> Adj.R² = 1 - (1-R²) * (7/5) ~ 0,8062 (details see notepage)

Figure 11: Manuelle Berechnung mLR($k=2$)

5.7.2 Python-Programm zur Berechnung eines mLR(k=2) Beispiels

***** Weitere Kapitel Texte und Beispiele einfügen, siehe Vorlesung *****



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

5.8 Übungen zum Kapitel 5

Diese Übungen waren Teil einer Vorlesung "Einführung in Maschinelles Lernen (ML)" im WS 2020 an der DHBW Stuttgart.

5.8.1 Übung 5.1 - Zwischenergebnis bei (Th-5.3)

Zeige die im Theorem (Th-5.3) noch nicht bewiesene Behauptung (*):

$$(*) \quad 0 = \sum_{i=1}^n (y_i - f_i)(f_i - \bar{y}) \text{ für optimale sLR Geraden.}$$

5.8.2 Übung 5.2 - Vermutung (V-5.1)

Beweisen Sie oder Widerlegen Sie folgende **Vermutung (V-5.1)**:

5.8 Übungen zum Kapitel 5

Aus der Bedingung $SST = SSE + SSR$ folgt für eine sLR-Gerade, dass diese auch optimal ist in unserer Definition (d.h. $R^2 = \text{maximal}$).

5.8.3 Übung 5.3 - Zwischenergebnis bei (Th-5.4)

Zeige die im Theorem (Th-5.4) noch nicht bewiesene Behauptung:

$$\nabla g(\mathbf{x}) = A^\top \cdot \mathbf{b}$$

5.8.4 Übung 5.4 - Manuelles sLR Beispiel

Führen Sei die Schritte analog der Vorlesung zur manuellen Befüllung der Excel-Tabelle zur Bestimmung der optimalen sLR-Gerade durch. Prüfen Sie da Ergebnis mit einem kleinen Python Programm. Plotten Sie die optimale sLR-Gerade mit den Ausgangsdatenpunkten. Stimmt das Bild mit Ihrer Erwartungshaltung überein (Datenpunkt 6 und 7)?

5.8.5 Übung 5.5 - Manuelles mLR(k=2) Beispiel

Führen Sei die Schritte analog der Vorlesung zur manuellen Befüllung der Excel-Tabelle zur Bestimmung der optimalen mLR-Ebene durch. Prüfen Sie da Ergebnis mit einem kleinen Python Programm. Plotten Sie die opt. mLR-Ebene mit den Ausgangsdatenpunkten. Stimmt das Bild mit Ihrer Erwartungshaltung überein?

5.8.6 Übung 5.6 - sLR Beispiel "Studentisches Examen"

1. Teil: Finde eine "least square fit" gerade $y = a + b*x$ für $y:=\text{erreichte Punktzahl (Score)}$ des Examens [pt.] abhängig vom Parameter: $x:=$ "Aufwand der Examens Vorbereitung in Std.[h]" . Daten der Trainings-Sets TS = $\{(examvorb.[h]; score[pt.])\} = \{(7; 41), (3; 27), (5; 35), (3; 26), (8; 48), (7; 45), (10; 46), (3; 27), (5; 29), (3; 19)\}$
Baue das Modell sLR(x,y). Vergleiche und prüfe das Ergebnis mit einem Python-Programm.

2. Teil: Wiederhole obige Aufgabe mit zehn anderen Eingabewerten:

$$\{(homework[h]; score[pt.]) \\ = (5; 41), (4; 27), (5; 35), (3; 26), (9; 48), (8; 45), (10; 46), (5; 27), (3; 29), (3; 19)\}$$

Zusatzaufgaben: Beantworte die folgenden Fragen:

Q1: Wie viele Punkte würde ein Student ohne Examens Vorbereitung erreichen? Ist dies realistisch (Hinweis: Sinnhaftigkeit von "Bias")?

Q1': Wie viele Punkte würde ein Student ohne Hausaufgabe/Homework Aufwand

5.8 Übungen zum Kapitel 5

erreichen? Ist dies realistisch (Hinweis: Sinnhaftigkeit von "Bias")?

Q2: Wie viele Punkte würde ein Student mit 10 Stunden für Examens Vorbereitung erreichen?

Q2': Wie viele Punkte würde ein Student mit 10 Stunden Homework erreichen?

Q3: Wie viel Aufwand Examens Vorbereitung ist notwendig um genug Punkte (=25) zum Bestehen des Examens?

Q3': Wie viel Aufwand Homework ist notwendig um genug Punkte (=25) zum Bestehen des Examens?

Q4: Welche Aufwandsart ist aufgrund unserer Trainingsdaten effektiver?

5.8.7 Übung 5.7 - mL(k=2) Beispiel "Studentisches Examen"

Berechne das mL(k=2)-Modell studentische Examens-Ergebnisse mit zwei 2 unabhängigen Parametern (Vergleiche auch Übung 5.6):

Finde eine "least square fit" Ebene $z = a + b*x + c*y$ für z :=erreichte Punktzahl (Score) des Examens [pt.] abhängig von den zwei Parametern: x := "Aufwand der Examens Vorbereitung in Std.[h]" and y :=Aufwand für Hausaufgaben in Std.[h].

Daten der Trainings-Sets TS :

$$= \{(x, y; z) | (examvorb.[h], homework[h]; score[pt.])\} = \{(7, 5; 41), (3, 4; 27), (5, 5; 35), (3, 3; 26), (8, 9; 48), (7, 8; 45), (10, 10; 46), (3, 5; 27), (5, 3; 29), (3, 3; 19)\}$$

Baue das Modell $mL(x,y;z)$. Vergleiche und prüfe das Ergebnis mit einem Python-Programm.

Zusatz: Beantworte die folgenden Fragen:

Q1: Wie viele Punkte würde ein Student ohne Examens Vorbereitung und ohne Hausaufgabe Aufwand erreichen?

Q2: Wie viele Punkte würde ein Student mit 10 Stunden für Examens Vorbereitung und auch mit 10 Std. Hausaufgabe Aufwand erreichen?

Q3: Wie viel Aufwand für Examensvorbereitung ist notwendig um genug Punkte (=25) zum Bestehen des Examens? Wie gross ist der Aufwand bei Homework?

Zusätzliche Frage/Anmerkung: Unsere Berechnung nutzt beide Variablen zur Berechnung. Was ist der Unterschied zum sLR-Model Resultat? Vergleiche $adj.R^2$ (wie hier genutzt) zu den zwei R^2 - Lösungen von Übung 5.6.

5.8.8 Übung 5.8 - "Beweis zu Korollar (K-5.5)"

Beweise die Aussage von Korollar (K-5.5) und lerne dabei die "Allgemeinen Berechnungen zu LSF-Verfahren" dadurch kennen. Beachte die Hinweise.

5.8.9 Übung 5.9 - "Beweis zur Anmerkung (A-5.2)-Fall (k=1)"

Beweise die Aussage von Anmerkung (A-5.2) im Fall K=1 und lerne dabei die "Allgemeinen Berechnungen zum LSF-Verfahren (ohne Bias-Trick)" besser kennen.

Beachte die Hinweise im Skript. Nutze die Ergebnisse des Beispieles der Berechnung der inversen Matrix zu einer symmetrischen (2x2)-Matrix (Unterkapitel (5.4.1)).

5.8.10 Übung 5.10 - "Beweis zur Anmerkung (A-5.2) Fall (k=2)"

Beweise die Aussage von Anmerkung (A-5.4) im Fall k=2 und lerne dabei die "Allgemeinen Berechnungen zum LSF-Verfahren (ohne Bias-Trick)" besser kennen.

Beachte die Hinweise im Skript. Nutze die Ergebnisse des Beispieles der Berechnung der inversen Matrix zu einer symmetrischen (3x3)-Matrix (Unterkapitel (5.4.1)).

6 Text-Klassifikation via Naive-Bayes Verfahren

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.

6.1 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

6 TEXT-KLASSIFIKATION VIA NAIVE-BAYES VERFAHREN

6.1 Eine Einführung in den naiven Bayes-Klassifikator

Bayes Learning (Conditional Probability):

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

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$$

LIKELIHOOD
The probability of "B" being True, given "A" is True

PRIOR
The probability "A" being True. This is the knowledge.

POSTERIOR
The probability of "A" being True, given "B" is True

MARGINALIZATION
The probability "B" being True.

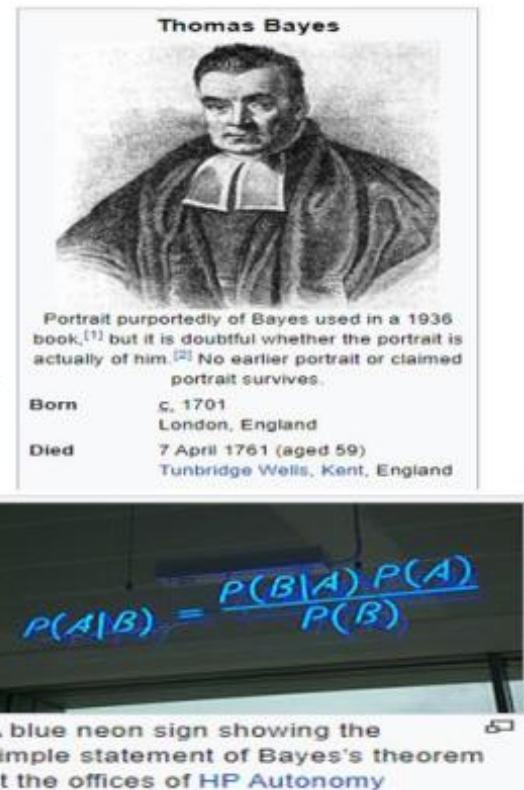


Figure 12: Screenshots zum Naive-Bayes-Learning

Trainingsprozess erlernt wurden.

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 Unabhängigkeitssannahme zwischen den Features in einigen Fällen zu weniger genauen

6 TEXT-KLASSIFIKATION VIA NAIVE-BAYES VERFAHREN

6.2 Mathematische Grundlagen bei Textklassifikatoren

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.

6.2 Mathematische Grundlagen bei Textklassifikatoren

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

6.2.1 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:

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

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

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

with

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

6 TEXT-KLASSIFIKATION VIA NAIVE-BAYES VERFAHREN

6.2 Mathematische Grundlagen bei Textklassifikatoren

6.2.2 "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 $\mathbf{x} = \langle x_1, x_2, \dots, x_d \rangle$ from a **multinomial distribution** with N trials, a "smoothed" version of the data gives the estimator:

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

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.

6 TEXT-KLASSIFIKATION VIA NAIVE-BAYES VERFAHREN

6.3 Konkrete Anwendung des naiven Bayes-Klassifikator

6.3 Konkrete Anwendung des naiven Bayes-Klassifikator

Naive Bayes Algorithm

Sentence Classification

What is Bayes Algorithm

-  Simple algorithm to classify text
-  Low training time and resources
-  Requires a set of labeled training data
-  Will be used to classify new sentences

Our 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

* Training data consists of two classes
+ sport or not sport

A = class
B = sentence

The probability OF "B" BEING TRUE GIVEN THAT "A" IS TRUE

The probability OF "A" BEING TRUE

$$P(A|B) = \frac{P(B|A) P(A)}{P(B)}$$

The probability OF "A" BEING TRUE GIVEN THAT "B" IS TRUE

The probability OF "B" BEING TRUE

Bayes' rule

6 TEXT-KLASSIFIKATION VIA NAIVE-BAYES VERFAHREN

6.3 Konkrete Anwendung des naiven Bayes-Klassifikator

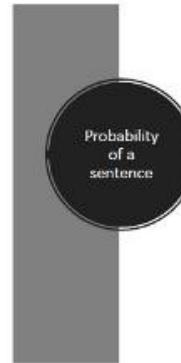


How does it work?

- Comparing the probabilities:

$$P(\text{Sport} | \text{Hermann played a TT match}) = \frac{P(\text{A very close game}|\text{Sports}) * P(\text{Sports})}{P(\text{A very close game})}$$

$$P(\text{Not Sport} | \text{Hermann played a TT match}) = \frac{P(\text{A very close game}|\text{not sports}) * P(\text{Not Sports})}{P(\text{A very close game})}$$



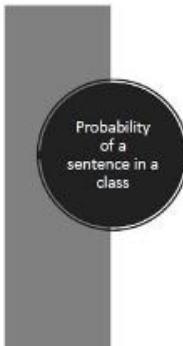
- Likelihood a sentence is a sport sentence:

- $P(\text{Sport}) = \frac{\text{Number of sentences in class "Sports"}}{\text{total number of sentences in the training set}}$
- Similarly calculate $P(\text{Not Sports})$

- „Naive“ Bayes because we think each word is independent from the other ones

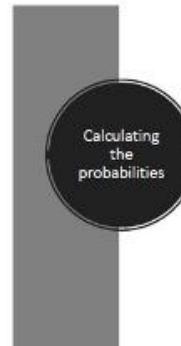
- Possibility of a sentence „A very close game“ is calculated like this:

- $P(\text{A very close game}) = P(A) * P(\text{very}) * P(\text{close}) * P(\text{game})$



- Applying the probabilities of the words to Bayes formula:

$$P(\text{A very close game} | \text{Sports}) = \frac{P(\text{A}|\text{Sports}) * P(\text{very}|\text{Sports}) * P(\text{close}|\text{Sports}) * P(\text{game}|\text{Sports})}{P(\text{Sports})}$$



- Now all we have to do is calculate all the different probabilities 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 close match”	Sports
4	“A clean but forgettable game”	Sports
5	“It was a close election”	Not Sports

- $P(\text{Sports}) = 3/5 \quad P(\text{Not Sports}) = 2/5$

- Probability of a word in class Sports:

- $P(\text{game}|\text{Sports}) = \frac{\text{amount of "game" in Sports sentences}}{\text{total number of words in Sports sentences}} = \frac{2}{11}$

- Repeat for other words and other classes

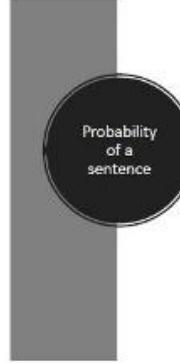
6 TEXT-KLASSIFIKATION VIA NAIVE-BAYES VERFAHREN

6.3 Konkrete Anwendung des naiven Bayes-Klassifikator

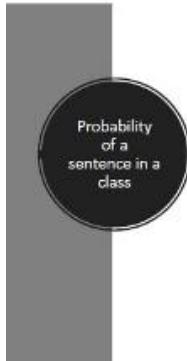


How does it work?

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

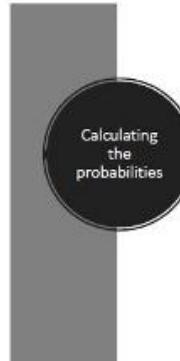


- Likelihood a sentence is a sport sentence:
 - $P(\text{Sport}) = \frac{\text{Number of sentences in class 'Sports'}}{\text{Total number of sentences in the training set}}$
 - Similarly calculate $P(\text{Not Sports})$
- „Naive“ Bayes because we think each word is independent from the other ones
 -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



- Applying the probabilities of the words to Bayes formula:

$$P(\text{A very close game}|\text{Sports}) = \frac{P(\text{A}|\text{Sports}) \cdot P(\text{very}|\text{Sports}) \cdot P(\text{close}|\text{Sports}) \cdot P(\text{game}|\text{Sports})}{P(\text{Sports})}$$



- Now all we have to do is calculate all the different probabilities 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
- $P(\text{Sports}) = 3/5 \quad P(\text{Not Sports}) = 2/5$
- Probability of a word in class Sports:
 - $P(\text{game}|\text{Sports}) = \frac{\text{amount of "game" in Sports sentences}}{\text{total number of words in Sports sentences}} = \frac{3}{11}$
- Repeat for other words and other classes

6 TEXT-KLASSIFIKATION VIA NAIVE-BAYES VERFAHREN

6.3 Konkrete Anwendung des naiven Bayes-Klassifikator

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:

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

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

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

with

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

6 TEXT-KLASSIFIKATION VIA NAIVE-BAYES VERFAHREN

6.3 Konkrete Anwendung des naiven Bayes-Klassifikator

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:

$$P(\text{a very close game}) = P(a) \times P(\text{very}) \times P(\text{close}) \times P(\text{game})$$

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:

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

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* = $P(\text{Sports})=3/5$. Then, $P(\text{Not Sports})=2/5$. That's easy enough.

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

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

6 TEXT-KLASSIFIKATION VIA NAIVE-BAYES VERFAHREN

6.3 Konkrete Anwendung des naiven Bayes-Klassifikator

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 $P(\text{game}|\text{Sports}) = (2+1)/(11+14) = 3/25$. The full results are:

$P(\text{Sports}) = 3/5$
 $P(\text{Not Sports}) = 2/5$
Anzahl (Words|Sports) = 11
 $\# (\text{Words|Not Sports}) = 9$
 $\# (\text{possible words}) = 14 \text{ -- see notes}$



Word	$P(\text{word} \text{Sports})$	$P(\text{word} \text{Not Sports})$
a	3/25	2/23
very	2/25	1/23
close	1/25	2/23
game	3/25	1/23

$$\Rightarrow P(a|\text{Sports}) * P(\text{very}|\text{Sports}) * P(\text{close}|\text{Sports}) * P(\text{game}|\text{Sports}) * P(\text{Sports}) = 3/25 * 2/25 * 1/25 * 3/25 * 3/5 = 18/25 * 1/25^3 * 3/5 = 54/125 * 1/25^3 \approx 2,7648e-5$$

$$\text{Analog: } P(a|\text{Not Sports}) * P(\text{very}|\text{Not Sports}) * P(\text{close}|\text{Not Sports}) * P(\text{game}|\text{Not Sports}) * P(\text{Not Sports}) = 2/23 * 1/23 * 2/23 * 1/23 * 2/5 = 8/115 * 1/23^3 \approx 0,57176e-5$$

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

6 TEXT-KLASSIFIKATION VIA NAIVE-BAYES VERFAHREN

6.4 Übungen zum Kapitel 6

6.4 Übungen zum Kapitel 6

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

.....

6.4.1 Übung 6.1 - Bsp. einer Bayes-Texklassifikation

Analog dem Beispiel aus obigen Kapitel soll die Bayes- Texklassifikation 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	<i>"Hermann plays a TT match"</i>	???????????

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

6.4.2 Übung 6.2 - Bayes-Texklassifikation 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>

6.4.3 Übung 6.3

: *** Weitere Kapitel Texte ***

7 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-Klassifikation hat sehr viel mit SVM zu tun. Vergleichen Sie dazu die Beispiele die in der Referenz [TK + SVM] zu sehen sind.

7.1 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: $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$, 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 **mathematischen 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 ($y_i = -1$): $w \cdot x_i + b \leq -1$

7 VERFAHREN DER "SUPPORT VECTOR MACHINES" (SVM)

7.2 Funktionsweise im Detail

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:

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

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.

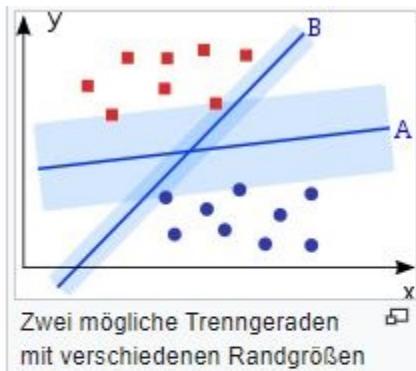
*** Weitere Kapitel Texte ***

7.2 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.

7 VERFAHREN DER "SUPPORT VECTOR MACHINES" (SVM)

7.2 Funktionsweise im Detail



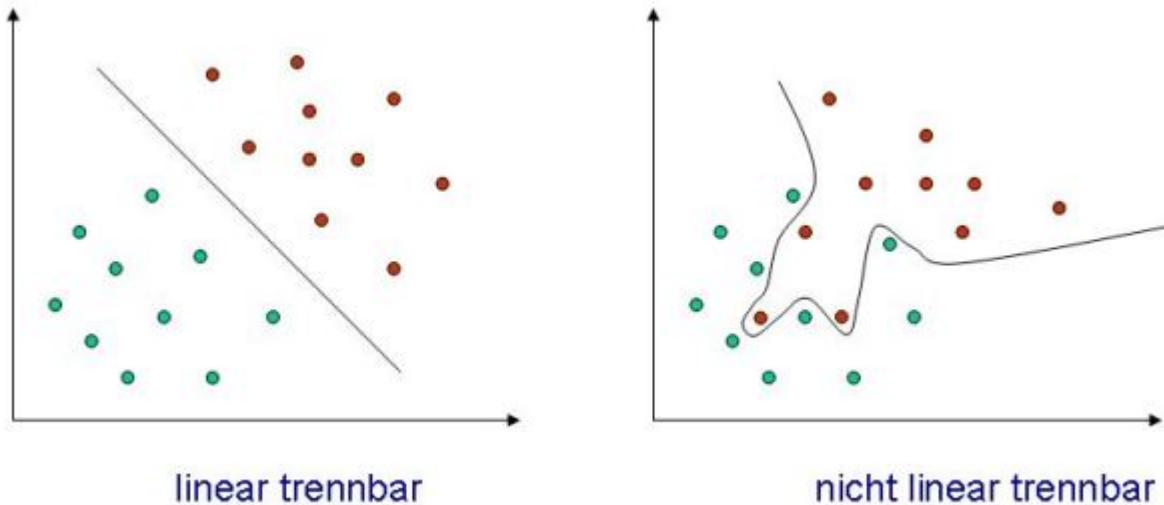
Beim Einsetzen der Hyperebene ist es nicht notwendig, alle Trainingsvektoren zu beachten. Vektoren, die weiter von der Hyperebene entfernt liegen und gewissermaßen hinter einer Front anderer Vektoren "versteckt" sind, beeinflussen Lage und Position der Trennebene nicht. Die Hyperebene ist nur von den 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.

7.2.1 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.

7 VERFAHREN DER "SUPPORT VECTOR MACHINES" (SVM)

7.2 Funktionsweise im Detail



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.

7.2.2 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

7 VERFAHREN DER "SUPPORT VECTOR MACHINES" (SVM)

7.3 Anschauliches 2-dim. Beispiel für Kernel-Trick

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.

7.3 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 $\Phi : \mathbb{R}^2 \rightarrow \mathbb{R}^3$, so dass 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:

7.3.1 Definition des Kernel-Trick Beispieles

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:

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

Die transformierten Punkte $A' = \Phi(A)$ $E' = \Phi(E)$ 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:

7 VERFAHREN DER "SUPPORT VECTOR MACHINES" (SVM)

7.3 Anschauliches 2-dim. Beispiel für Kernel-Trick

Rote Hyperebene $H_1 = \{(x_1, x_2, 2)\}$ & blaue Hyperebene $H_2 = \{x_1, x_2, 5\}$

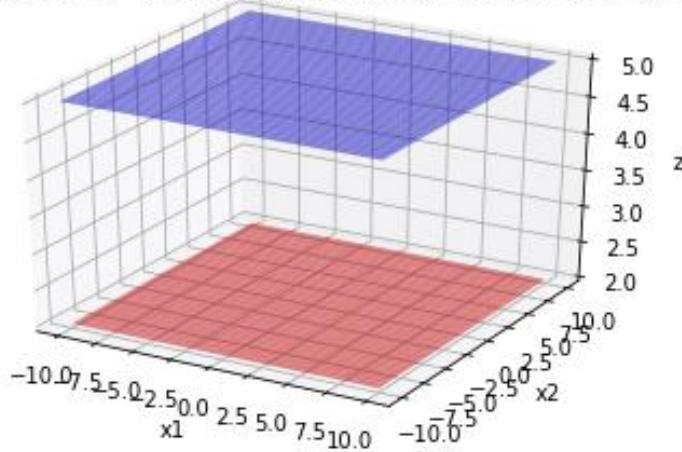


Figure 13: Hyperebenen zum Kernel-Trick Beispiel

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. Der Normalenvektor dieser Ebene ist in z-Richtung also $(0, 0, 1)$ in \mathbb{R}^3 . Da alle Punkte A' bis E' den gleichen Abstand 1.5 zu dieser Hyperebene haben, sind diese alle "Stützvektoren im Sinne von SVM (siehe oben).

Durch eine "Zurücktransformation" erhält man den **Kreis**: $x_1^2 + x_2^2 = 3,5$ mit dem Radius $r = \sqrt{3,5} \approx 1.871$. Damit ergibt sich die visuelle Darstellung der Klassifikation-Trennlinie als Kreislinie. Wir lassen uns diesen Kreis via einem kleinen Python Programm zeichnen:

7 VERFAHREN DER "SUPPORT VECTOR MACHINES" (SVM)

7.3 Anschauliches 2-dim. Beispiel für Kernel-Trick

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 theta = np.linspace(0, 2*np.pi, 100)
5 r = 1.8
6
7 x = r * np.cos(theta)
8 y = r * np.sin(theta)
9
10 plt.plot(x, y, color='black', label='(x_1)^2 + (x_2)^2 = 3')
11
12 points = {'A': (1, 1), 'B': (-1, 1), 'C': (2, 1), 'D': (1, -2), 'E': (-2, 1)}
13 colors = {'A': 'red', 'B': 'red', 'C': 'blue', 'D': 'blue', 'E': 'blue'}
14
15 for point, coords in points.items():
16     plt.scatter(coords[0], coords[1], color=colors[point], label=point)
17     plt.annotate(point, coords, textcoords="offset points", xytext=(-10,10), ha='center')
18
19 plt.xlabel('x_1')
20 plt.ylabel('x_2')
21 plt.title('Graph des Kreises (x_1)^2 + (x_2)^2 = 3 mit roten und blauen Punkten')
22 plt.grid(True)
23 plt.axis('equal')
24 #plt.legend()
25 plt.show()

```

Graph des Kreises $(x_1)^2 + (x_2)^2 = 3$ mit roten und blauen Punkten

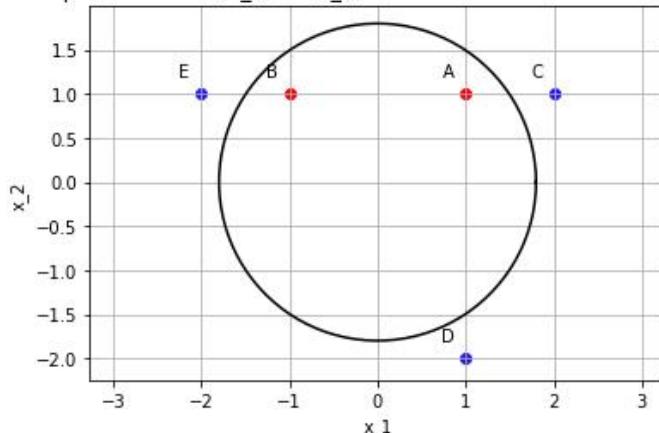


Figure 14: Python Code und Kernel-Trick Kreisbild

7.3.2 Berechnung per Jupyter Notebook

...

Wenn es mehr Daten-Punkte gibt (was die Regel ist) müssen wir Algorithmen einsetzen. Die Beispiele werden dann auch entsprechend komplizierter und können nicht mehr manuell berechnet werden. Die Normalvektoren $(0, 0, 1)$ in R^3 sind sicherlich nicht senkrecht, sondern haben Beiträge in x- und y-Richtung.

Ein solches Beispiel soll als Übungsbeispiel in Übung(7.1) gerechnet werden

Um ein einfaches und anschauliches Beispiel für eine Support Vector Machine (SVM) mit einem Kernel-Trick von 2D nach 3D in einem Jupyter Notebook zu erstellen, können Sie den folgenden Python-Code verwenden. Der Code verwendet die scikit-learn Bibliothek für die SVM und die matplotlib Bibliothek zum Plotten. Zunächst importieren Sie die erforderlichen Bibliotheken. Hier das komplette Notebook:

https://github.com/HVoellinger/Mathematische-Grundlagen_von_ML/blob/main/Quellen_Math-ML/SVM-Kernel-2D-3D.pdf

7.4 Übungen zum Kapitel 7

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

7.4.1 Übung 7.1 - Einfaches Kernel-Trick(2D nach 3D)- Beispiel

Berechnen Sie analog zum Kapitel (7.3) ein einfaches und anschauliches Beispiel für eine Support Vector Machine (SVM) mit einem Kernel-Trick von 2D nach 3D in einem Jupyter Notebook.

Sie nehmen wieder 2 Klassen +1 und -1 mit jetzt 5 Daten-Punkten. Wählen Sie eine passende Transformation von 2D nach 3D.

Gegen Sie die SVM-Ebene an mit Ebenen-Gleichung (Normalvektor), Stützvektoren an. Visualisieren Sie die Punkte und Trennlinie in 2D und in 3D

7.4.2 Übung 7.2 - Vergleich Polymonischer Kernel mit RBF Kernel

In dieser Übung erstellen wir einen Datensatz mit Punkten, die konzentrische Kreise bilden, die im ursprünglichen Merkmalsraum nicht linear separierbar sind. Wir verwenden dann zwei SVM-Modelle: eines mit einem Polynomischen Kernel und ein anderes mit einem RBF-Kernel (Radial Basis Function). Der RBF-Kernel wendet effektiv den Kernel-Trick an, indem er die Daten in einen höherdimensionalen Raum transformiert, in dem sie linear trennbar werden. Die Entscheidungsgrenzen und Datenpunkte werden grafisch dargestellt, um den Unterschied zwischen den beiden Kerneln zu verdeutlichen. Der RBF-Kernel wird in der Praxis häufig verwendet, um nicht linear trennbare Daten zu verarbeiten, und ist eine der wichtigsten Anwendungen des Kernel-Tricks in der SVM. Hier das Beispiel Notebook:

https://github.com/HVoellinger/Mathematische-Grundlagen_von_ML/blob/main/Quellen_Math-ML/SVM-RBF+Polyn-Kernel.pdf

7.4.3 Übung 7.3 - 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.

8 Neuronale Faltungsnetzwerke "Convolutional Neural Networks"

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] *****

8.1 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.

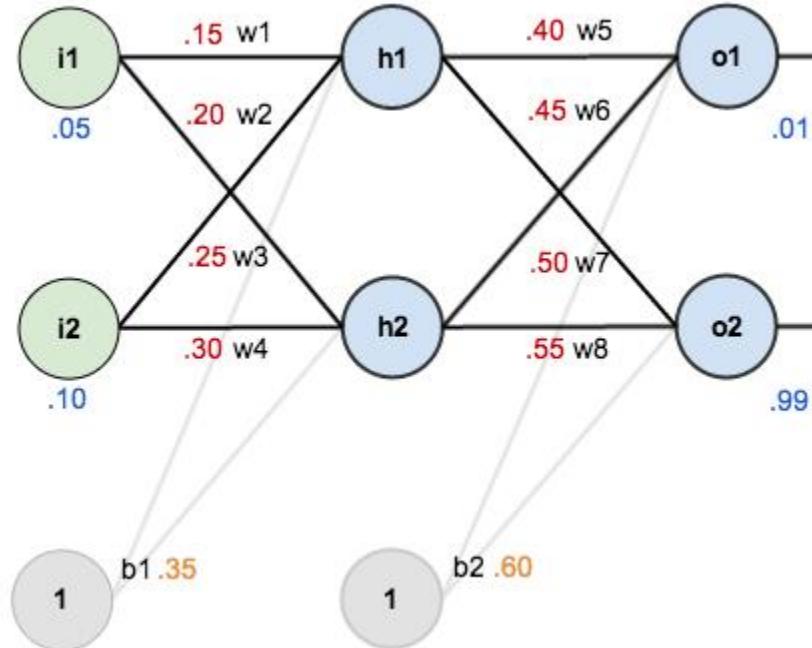
8.2 Einfaches Beispiel - Rückwärtspropagierung

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)) \Rightarrow Verarbeitungsschicht ("Hidden Layer" mit 2 "Hidden Nodes" (h_1, h_2)) \Rightarrow 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:



8.2.1 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 propagierte 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):

$$h_1 = (i_1 \cdot w_1) + (i_2 \cdot w_2) + b_1 = (0.05 \cdot 0.15) + (0.1 \cdot 0.25) + 0.35 = 0.3775$$

$$h_2 = (i_1 \cdot w_3) + (i_2 \cdot w_4) + b_1 = (0.05 \cdot 0.2) + (0.1 \cdot 0.3) + 0.35 = 0.3925$$

$$H_1 = \sigma(h_1) = \frac{1}{1 + e^{-0.3775}} \approx 0.5933$$

$$H_2 = \sigma(h_2) = \frac{1}{1 + e^{-0.3925}} \approx 0.5969$$

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)

$$o_1 = (a_1 \cdot w_5) + (a_2 \cdot w_6) + b_2 = (0.5933 \cdot 0.4) + (0.5969 \cdot 0.45) + 0.6 \approx 1.1059$$

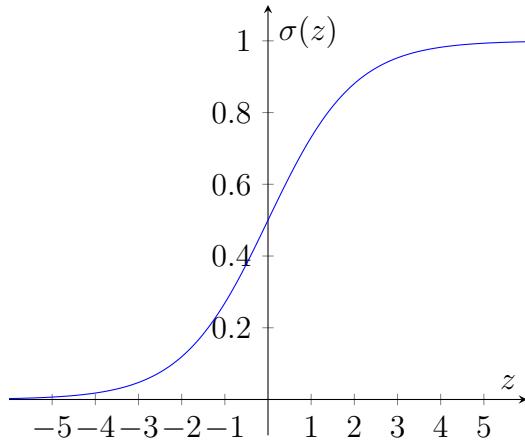
$$O_1 = \sigma(o_1) \approx 0.7514$$

$$o_2 = (a_1 \cdot w_7) + (a_2 \cdot w_8) + b_2 = (0.5933 \cdot 0.5) + (0.5969 \cdot 0.55) + 0.6 \approx 1.2249$$

$$O_2 = \sigma(o_2) \approx 0.7729$$

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.

8.2.2 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:

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

8.2.3 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"):

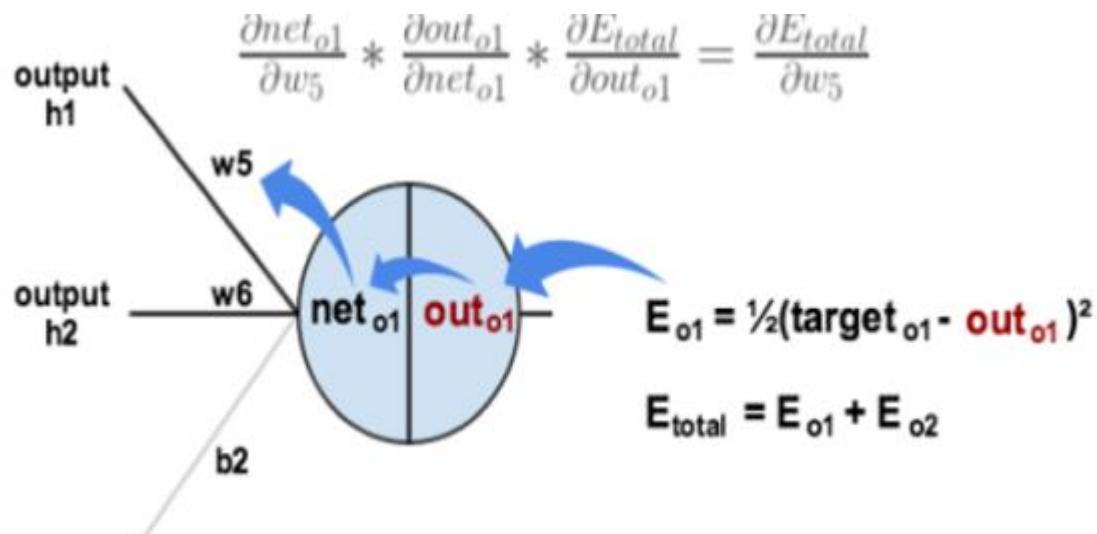


Figure 15: Backpropagation-Rückwärtslauf

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

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

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

$$\begin{aligned}w'_5 &= w_5 - \eta \cdot \frac{\partial \text{loss}}{\partial w_5} \approx 0.2 - 0.1 \cdot (-0.0664) \approx 0.3066 \\ w'_6 &= w_6 - \eta \cdot \frac{\partial \text{loss}}{\partial w_6} \approx 0.4 - 0.1 \cdot (-0.0497) \approx 0.7050 \\ b'_3 &= b_3 - \eta \cdot \frac{\partial \text{loss}}{\partial b_3} \approx 0.1 - 0.1 \cdot (-0.0743) \approx 0.2074\end{aligned}$$

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]
 ****=
 ***** Ende Anpassung für diese Subsection *****
 ****=

8.3 Übungen zum Kapitel 8

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

.....

8.3.1 Übung 8.1 - CNN Architekut

TEXT

8.3.2 Übung 8.2 - Beispiel Backpropagation

.....

8.3.3 Übung 8.3 -

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 einen niedrigdimensionalen latenten 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.

9.1 Empfehlungssysteme "Recommender Systems" (LinAlgebra)

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-

9.2 Regularisierungen und Lineare Algebra (LA)

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"

9.3 Hauptkomponentenanalyse "Principal Component Analysis"

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 Faktorenanalyse zu unterscheiden, mit der sie formale Ähnlichkeit hat und in der sie als Näherungsmethode zur Faktorextraktion verwendet werden kann (der Unterschied der beiden Verfahren kann in Wikipedia nachgelesen werden).

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

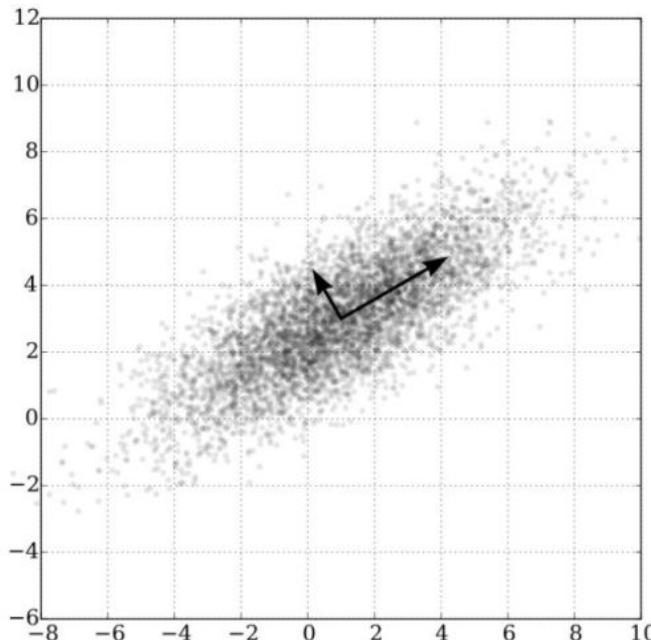


Figure 16: Schematische Darstellung der Hauptkomponentenanalyse

Zwei Hauptkomponenten einer zweidimensionalen Normalverteilung mit Mittelwert

9 ANHÄNGE

9.3 Hauptkomponentenanalyse "Principal Component Analysis"

(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)

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 σ_1 und σ_2 sind die Längen der großen bzw. kleinen Halbachse der Ellipse.

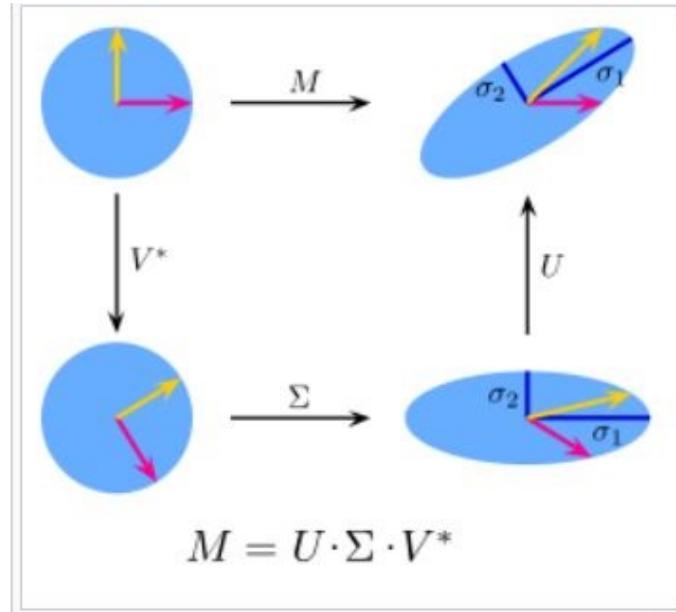


Figure 17: Schematische Darstellung des Singulärwertzerlegung

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

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.

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

9.6.1 Wie funktionieren LLMs?

Große bzw. generative KI-Modelle werden auch als "Basismodelle" ("Foundation Models") oder, als wichtigster Unterfall, ""große Sprachmodelle" ("Large Language Models" – LLMs) bezeichnet.

Obwohl das Aufkommen dieser Modelle in den letzten Jahren einen bedeutenden technologischen Fortschritt darstellt, nutzen sie weitestgehend bestehende Technologien – jedoch in einem viel größeren Maßstab und Umfang. Generative KI-Modelle werden typischerweise mit mehreren Milliarden, wenn nicht Hunderten von Milliarden Parametern trainiert und benötigen große Mengen an Trainingsdaten und Rechenleistung.

Generative KI-Modelle sind fortgeschrittene, auf maschinellem Lernen beruhende Modelle, die darauf ausgelegt sind, neue, zur Zeit so bislang nicht existierende Inhalte zu erzeugen.

Sie basieren hauptsächlich auf zwei speziellen Typen von neuronalen Netzen: "Generative Adversarial Networks" (GANs) oder "Transformer-Netzwerken".

Damit unterscheiden sie sich von anderen KI-Modellen, die nur für Vorhersagen oder Klassifikationen konzipiert sind oder andere spezifische Funktionen erfüllen. Die beiden derzeit bedeutendsten generativen KI-Modelle, ChatGPT und GPT-4, sind Transformer-Modelle. Vereinfacht ausgedrückt, lernen texterzeugende Transformer die Muster und Strukturen aus einem großen Datensatz menschlicher Sprache und nutzen dieses Wissen, um neuen, zusammenhängenden Text zu erzeugen.

Dazu wird der Eingabetext durch mehrere Schichten von neuronalen Netzwerken verarbeitet, welche die relevanten Merkmale und Muster in der Eingabe extrahieren und verarbeiten. Der Eingabetext wird dabei als Matrix von Wort-Umgebungen repräsentiert, die Bedeutungen und Beziehungen zwischen den Wörtern in der Eingabe erfassen.

Die Ausgabe des Transformers ist eine Wahrscheinlichkeitsverteilung über die Wörter im Vokabular, die die Wahrscheinlichkeit jedes Worts im erzeugten Text darstellt. Das Modell verwendet diese Wahrscheinlichkeitsverteilung, um das nächste zu generierende Wort auszuwählen, und dieser Prozess wird wiederholt, bis die gewünschte Länge des Textes erzeugt ist.

Durch das Ziehen von Stichproben aus den Daten und das Mischen dieser Stichproben kann das Modell Inhalte erzeugen, die über den Trainingsdatensatz hinausgehen – die man mithin als "neu" ansehen kann. Generative KI-Modelle sind oft in der Lage, menschliche Texteingaben zu verarbeiten und auf dieser Grundlage eine Ausgabe (Text, Bild, Audio, Video) zu erzeugen.

Aufgrund der großen Datenmengen, die benötigt werden, müssen sich die Entwickler von generativen KI-Modellen allerdings häufig auf Trainingsdaten verlassen, die frei im Internet verfügbar sind und deren Datenqualität zweifelhaft ist. Die von diesen Modellen generierten Inhalte können personenbezogene Daten enthalten, und auch verzerrt, verfälscht oder schädlich sein.

9.6.2 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-4.

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 Netzwerken 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 Fal-

tungen 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.

.....

9.6.3 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 Medienecho.

GE

Schreib eine lustige, kurze Einleitung, in der Du Dich selbst als generative KI vorstellst. Danke!



"Hallo, ich bin ChatGPT, eine generative KI mit einer Vorliebe für Wortspiele und schlechte Witze. Aber keine Angst, ich bin hier, um Ihnen zu helfen und nicht, um die Welt zu beherrschen. Zumindest noch nicht..."



10 Lösungen / 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.

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

10.1 Lösungshinweise zu Übungen Kapitel 3

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

10.1.1 Lösungshinweise zu Übung 3.1

.....

10.1.2 Lösungshinweise zu Übung 3.2

10 LÖSUNGEN / LÖSUNGSHINWEISE ZU DEN ÜBUNGEN
10.2 Lösungshinweise zu Übungen Kapitel 4

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:

10 LÖSUNGEN / LÖSUNGSHINWEISE ZU DEN ÜBUNGEN

10.3 Lösungshinweise zu Übungen Kapitel 5

10.3 Lösungshinweise zu Übungen Kapitel 5

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

10.3.1 Lösungshinweise zu Übung 5.1

Siehe analoge Teilberechnungen im Skript.

10.3.2 Lösungshinweise zu Übung 5.2

Siehe Hinweise im entsprechenden Kapitel des Skriptes.

10.3.3 Lösungshinweise zu Übung 5.3

Siehe analoge Teilberechnungen im Skript.

10.3.4 Lösungshinweise zu Übung 5.4

$$a = -\frac{1}{7}, \quad b = \frac{61}{28}, \quad R^2 \sim 0.9247$$

siehe auch konkrete Werte in der entsprechenden Tabelle im Skript.

10.3.5 Lösungshinweise zu Übung 5.5

$$a \sim 5.522, \quad b \sim 0.4471, \quad c \sim 0.225, \quad adj.R^2 \sim 0.8062$$

Siehe auch konkrete Werte in der entsprechenden Tabelle im Skript.

10.3.6 Lösungshinweise zu Übung 5.6

Teil 1:

→ opt. Regression-Gerade: $y = 14,117 + 3,7376*x$

$$\rightarrow R^2 = 1 - \text{Sum}((y_i - y(x_i))^2) / \text{Sum}((y_i - M(y))^2) = 1 - (134,2172 / 922,1) \sim 0,8544$$

Teil 2:

→ opt. Regression-Gerade: $y = 15,164 + 3,479*x$

$$\rightarrow R^2 = 1 - \text{Sum}((y_i - y(x_i))^2) / \text{Sum}((y_i - M(y))^2) = 1 - (189,6999 / 922,1) \sim 0,7943$$

Zusatzfragen:

Question 1: 14,117 points. Bias = unrealistisch (zu wenig Daten und zudem fiktiv).

10 LÖSUNGEN / LÖSUNGSHINWEISE ZU DEN ÜBUNGEN

10.3 Lösungshinweise zu Übungen Kapitel 5

Question 1': 15,114 points. Bias = unrealistisch (zu wenig Daten und zudem fiktiv).

Question 2: $14,117 + 37,38 = 51,497$ points.

Question 2': $15,164 + 34,79 = 49,954$ points.

Question 3: $x = (25-14,117)/3,7376 = 2,91[\text{h}]$

Question 3': $x = (25-15,164)/3,479 = 2,83[\text{h}]$

Question 4: Zum Bestehen ist Homework besser (höherer Bias) und um jedoch insgesamt ein besseres Ergebnis zu erhalten ist die Examensvorbereitung besser (höherer "Slope" = "Lernkurve") und der Bestimmungswert R^2 ist besser.

Für eine komplette Lösung vergleiche:

https://github.com/HVoellinger/Mathematische-Grundlagen_von_ML/blob/main/Quellen_Math-ML/LR-Calculation_of_Coeff.xlsx

10.3.7 Lösungshinweise zu Übung 5.7

→ opt. mLR-Ebene : $z = 13,264 + 2,488*x + 1,382*y$

→ Adj. $R^2 = 1 - (1-R^2)*(9/7) \sim 0.8512$

Question 1: 13,264 points.

Question 2: $13,264 + 24,88 + 13,82 = 51,96$ points.

Question 3: Examensvorbereitung $x = (25-13,264)/2,488 = 4,72[\text{h}]$

Homework $y = (25-13,264)/1,382 = 8,49[\text{h}]$

Anmerkung: $adj.R^2$ ist geringer als R^2 für Examensvorbereitung, aber weit besser als R^2 bei Homework. Insgesamt ist somit diese Modell besser ("ausbalancierter" und geringerer Bias).

Für eine komplette Lösung vergleiche:

https://github.com/HVoellinger/Mathematische-Grundlagen_von_ML/blob/main/Quellen_Math-ML/LR-Calculation_of_Coeff.xlsx

10.3.8 Lösungshinweise zu Übung 5.8

Setze die Gleichungen (B) für $k=1$ und $K=2$ um, so erhalten wir für $(X^\top \cdot X)^{(-1)}$ und $X^\top \cdot \tilde{y}$ die entsprechenden ($k \times k$)-Matrix und ($k \times 1$)-Vektoren:

[https://github.com/HVoellinger/Mathematische-Grundlagen_von_ML/blob/main/Images/Korollar-\(K-5.5\)-Seite1.jpg](https://github.com/HVoellinger/Mathematische-Grundlagen_von_ML/blob/main/Images/Korollar-(K-5.5)-Seite1.jpg)

Berechnen wird die notwendigen Matrixen-Multiplikationen so erhalten wir für $k=1$

10 LÖSUNGEN / LÖSUNGSHINWEISE ZU DEN ÜBUNGEN

10.3 Lösungshinweise zu Übungen Kapitel 5

und k=2 die Ergebnisse der Theoreme (Th-5.2) und (Th-5.4):

[https://github.com/HVoellinger/Mathematische-Grundlagen_von_ML/blob/main/Images/Korollar-\(K-5.5\)-Seite2.jpg](https://github.com/HVoellinger/Mathematische-Grundlagen_von_ML/blob/main/Images/Korollar-(K-5.5)-Seite2.jpg)

10.3.9 Lösungshinweise zu Übung 5.9

Setze die Gleichungen (B') für k=1 um, so erhalten wir für $(X^\top \cdot X)$ und $X^\top \cdot y$ die folgenden (2x2)- und (2x1)-Matrizen :

$$X^\top \cdot X = \begin{pmatrix} \sum_{i=1}^n [x_1^{(i)}]^2 & \sum_{i=1}^n x_1^{(i)} \\ \sum_{i=1}^n x_1^{(i)} & n \end{pmatrix} \quad X^\top \cdot y = \begin{pmatrix} \sum_{i=1}^n [x_1^{(i)} \cdot y^{(i)}] \\ \sum_{i=1}^n y^{(i)} \end{pmatrix} \quad (B'_{k=1})$$

Für die Inverse Matrix $(X^\top \cdot X)^{(-1)}$ erhalten wir:

$$(X^\top \cdot X)^{(-1)} = \begin{pmatrix} n & -\sum_{i=1}^n x_1^{(i)} \\ -\sum_{i=1}^n x_1^{(i)} & \sum_{i=1}^n [x_1^{(i)}]^2 \end{pmatrix}$$

Multiplizieren wir diese jetzt noch mit $X^\top \cdot y$ und berechnen das Ergebnis für w_1 und b , so erhalten wir das Ergebnis von Theorem (Th-5.2). Siehe auch den folgenden Link:

https://github.com/HVoellinger/Mathematische-Grundlagen_von_ML/blob/main/Images/Anmerkung-A5.2-Blatt2.jpg

10.3.10 Lösungshinweise zu Übung 5.10

Setze die Gleichungen (B') für k=2 um, so erhalten wir für $(X^\top \cdot X)$ eine (3x3)-Matrix analog wie im Falls k=1 wo wir eine (2x2)-Matrix erhalten hatten:

Die Herleitung der Normalform ist dabei analog wie mit dem "Bias-Trick". Wir erhalten für die Faktoren der [Normalform](#):

$$X^\top \cdot X = \begin{pmatrix} \sum_{i=1}^n [x_1^{(i)}]^2 & \sum_{i=1}^n [x_1^{(i)} \cdot x_2^{(i)}] & \sum_{i=1}^n x_1^{(i)} \\ \sum_{i=1}^n [x_2^{(i)} \cdot x_1^{(i)}] & \sum_{i=1}^n [x_2^{(i)}]^2 & \sum_{i=1}^n x_2^{(i)} \\ \sum_{i=1}^n x_1^{(i)} & \sum_{i=1}^n x_2^{(i)} & n \end{pmatrix} X^\top \cdot y = \begin{pmatrix} \sum_{i=1}^n [x_1^{(i)} \cdot y^{(i)}] \\ \sum_{i=1}^n [x_2^{(i)} \cdot y^{(i)}] \\ \sum_{i=1}^n y^{(i)} \end{pmatrix} \quad (B'_{k=2})$$

Wie man leicht an der Matrix $X^\top \cdot X$ erkennt, ist die Matrix symmetrisch, da das

10 LÖSUNGEN / LÖSUNGSHINWEISE ZU DEN ÜBUNGEN

10.3 Lösungshinweise zu Übungen Kapitel 5

Produkt zweier reellen Zahlen kommutativ ist. Als nächstes ist $(X^\top \cdot X)^{(-1)}$ zu berechnen.

Hinweis: Eine manuelle Berechnung einer solchen inversen Matrix ist sehr aufwendig und fehleranfällig. Wir empfehlen deshalb dies über ein Programm zu machen. Siehe dazu auch das Codebeispiel in Python im Unterkapitel (5.4.1). Das komplette Jupyter Notebook (Code und PDF-Version) dazu finden sie auch unter folgendem Link:

Code: https://github.com/HVoellinger/Mathematische-Grundlagen_von_ML/blob/main/Quellen_Math-ML/Inv-Matrix-Berech+Ausgaben.ipynb

PDF-Version: https://github.com/HVoellinger/Mathematische-Grundlagen_von_ML/blob/main/Quellen_Math-ML/Inv-Matrix-Berech+Ausgaben.pdf

Bilden wir noch das Matrix-Produkt dieser Matrix mit $X^\top \cdot y$ und berechnen das Ergebnis für w_1, w_2 und b , so erhalten wir das Ergebnis von Theorem (Th-5.4).

10 LÖSUNGEN / LÖSUNGSHINWEISE ZU DEN ÜBUNGEN

10.4 Lösungshinweise zu Übungen Kapitel 6

10.4.1 Lösungshinweise zu Übung 6.1

- Sports:

$$P(\text{Hermann plays a TT match} \mid \text{Sports}) =$$

$$\begin{aligned} P(\text{Hermann|Sports}) * P(\text{plays|Sport}) * P(a|\text{Sports}) * P(\text{TT|Sports}) * P(\text{match|Sports}) * P(\text{Sports}) \\ = \frac{1}{29} * \frac{1}{29} * \frac{4}{29} * \frac{1}{29} * \frac{2}{29} * \frac{4}{6} = 2,6002\text{e-}7 \end{aligned}$$

- Not Sports:

$$P(\text{Hermann plays a TT match} \mid \text{Not Sports}) =$$

$$\begin{aligned} P(\text{Hermann|Not Sports}) * P(\text{plays|Not Sport}) * P(a|\text{Not Sports}) * P(\text{TT|Not Sports}) * P(\text{match|Not Sports}) \\ * P(\text{Not Sports}) \\ = \frac{1}{23} * \frac{1}{23} * \frac{2}{23} * \frac{1}{23} * \frac{1}{23} * \frac{2}{6} = 1,0358\text{e-}7 \end{aligned}$$

➔ It will be classified as a Sports-sentence

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

.....

10 LÖSUNGEN / LÖSUNGSHINWEISE ZU DEN ÜBUNGEN

10.4 Lösungshinweise zu Übungen Kapitel 6

10.4.2 Lösungshinweise zu Übung 6.2

```
In [17]: 1 # multiplying the probabilities of each word
2 new_sport = list(prob_new_sport)
3 sport_multiply_result = 1
4 for i in range(0, len(new_sport)):
5     sport_multiply_result *= new_sport[i]
6
7 # multiplying the result with the ratio of sports sentences to the total amount of sentences (here its 4/6)
8 sport_multiply_result *= ( len(tdm_sport) / len(rows) )
9
10 # multiplying the probabilities of each word
11 new_not_sport = list(prob_new_not_sport)
12 not_sport_multiply_result = 1
13 for i in range(0, len(new_not_sport)):
14     not_sport_multiply_result *= new_not_sport[i]
15
16 # multiplying the result with the ratio of sports sentences to the total amount of sentences (here its 2/6)
17 not_sport_multiply_result *= ( len(tdm_not_sport) / len(rows) )
18
19
```

```
In [18]: 1 # comparing what's more likely
2
3 print(f'The probability of the sentence "{new_sentence}":\nSport vs not sport\n{sport_multiply_result} vs {not_sport_multiply_result}')
4
5 if not_sport_multiply_result < sport_multiply_result:
6     print('Verdict: It\'s probably a sports sentence!')
7 else:
8     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!
```

10.4.3 Lösungshinweise zu Übung 6.3

10 LÖSUNGEN / LÖSUNGSHINWEISE ZU DEN ÜBUNGEN
10.5 Lösungshinweise zu Übungen Kapitel 7

10.5 Lösungshinweise zu Übungen Kapitel 7

.....

10.5.1 Lösungshinweise zu Übung 7.1

.....

10.5.2 Lösungshinweise zu Übung 7.2

.....

10.5.3 Lösungshinweise zu Übung 7.3

Hier ist ein beispielhafter Python-Code zur Generierung der zwei Hyperebenen im 2-dim. Beispiel zum Kernel-Trick.

Eine animierte Darstellung des Ergebnisses kann man in meinem GitHub anschauen:

[https://github.com/HVoellinger/Mathematische-Grundlagen_von_ML/blob/main/
Images/Kernel-Trick_Animation.gif](https://github.com/HVoellinger/Mathematische-Grundlagen_von_ML/blob/main/Images/Kernel-Trick_Animation.gif)

10 LÖSUNGEN / LÖSUNGSHINWEISE ZU DEN ÜBUNGEN

10.5 Lösungshinweise zu Übungen Kapitel 7

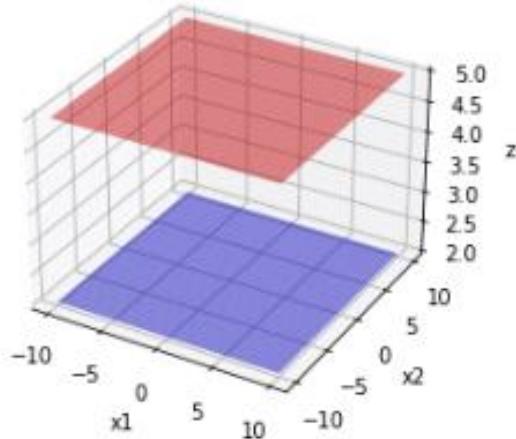
```
In [1]: 1 import numpy as np
2 import matplotlib.pyplot as plt
3 from mpl_toolkits.mplot3d import Axes3D
4 from matplotlib.animation import FuncAnimation
5
6 # Import Library time to check execution date+time
7 import time
8 # print the date & time of the notebook
9 print('*****')
10 print("Actual date & time of the notebook:",time.strftime("%d.%m.%Y %H:%M:%S"))
11 print('*****')
12
*****
Actual date & time of the notebook: 25.08.2023 14:41:36
*****
```

```
In [2]: 1 # Definition der Hyperebenen
2 def hyperplane1(x1, x2):
3     return 5 * np.ones_like(x1)
4
5 def hyperplane2(x1, x2):
6     return 2 * np.ones_like(x1)
7
8 # Erzeugung der Datenpunkte für den Plot
9 x1 = np.linspace(-10, 10, 100)
10 x2 = np.linspace(-10, 10, 100)
11 x1, x2 = np.meshgrid(x1, x2)
12 z1 = hyperplane1(x1, x2)
13 z2 = hyperplane2(x1, x2)
14
15 # Erstellen des 3D-Plots
16 fig = plt.figure()
17 ax = fig.add_subplot(111, projection='3d')
18
19 # Plotten der Hyperebenen
20 ax.plot_surface(x1, x2, z1, color='r', alpha=0.5, label='H1: z=5')
21 ax.plot_surface(x1, x2, z2, color='b', alpha=0.5, label='H2: z=2')
22
23 # Achsenbeschriftung
24 ax.set_xlabel('x1')
25 ax.set_ylabel('x2')
26 ax.set_zlabel('z')
27
28 # Legende
29 # ax.legend()
30
31
```

Figure 18: Codeblock(1+2) zum Kernel-Trick Beispiel
— 127/132 —

10 LÖSUNGEN / LÖSUNGSHINWEISE ZU DEN ÜBUNGEN
10.5 Lösungshinweise zu Übungen Kapitel 7

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



In [3]:

```
1 # Animieren der Ansicht (Drehen)
2 def animate(angle):
3     ax.view_init(elev=20, azim=angle)
4
5 # Erstellen der Animation
6 ani = FuncAnimation(fig, animate, frames=np.arange(0, 360, 2), interval=50)
7
8 # Animation als GIF speichern
9 ani.save('hyperebenen_animation.gif', writer='pillow')
10
11 # Anzeigen der Animation
12 plt.show()
13
14 # print current date and time
15
16 print("Date & Time:", time.strftime("%d.%m.%Y %H:%M:%S"))
17 # end of import test
18 print ("*** End of Hyperplan Example ***")
```

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

Figure 19: Codeblock(3) zum Kernel-Trick Beispiel

10 LÖSUNGEN / LÖSUNGSHINWEISE ZU DEN ÜBUNGEN
10.6 Lösungshinweise zu Übungen Kapitel 8

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>

[KS-GitHub] Karl Stroetmann ”Lecture Notes on Artificial Intelligence“ - GitHub/Artificial Intelligence; <https://github.com/karlstroetmann/Artificial-Intelligence>, 2019

[Wiki-ML] - Wikipedia: ”Machine learning ML“; https://en.wikipedia.org/wiki/Machine_learning Machine Learning

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

*** Weitere Referenzen ***

Indexverzeichnis