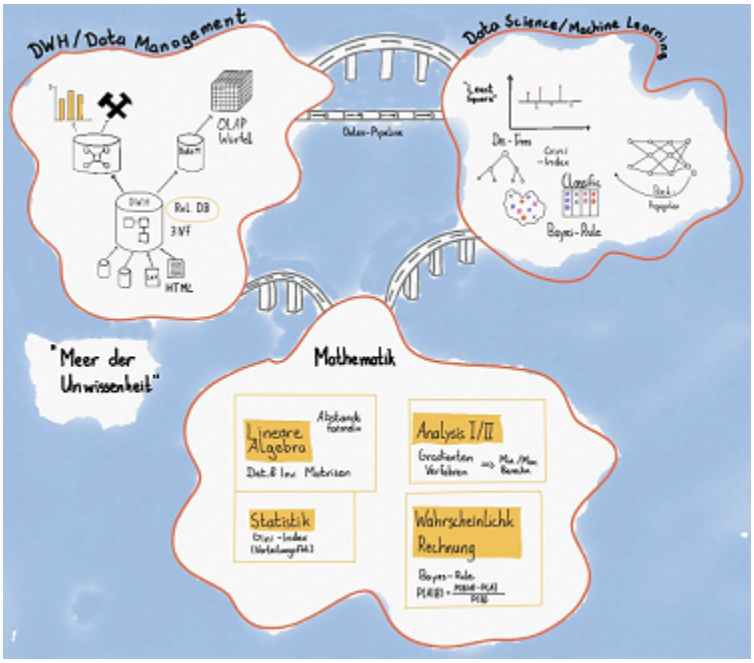
Mathematische Grundlagen des Maschinellen Lernens (ML)

—- Ausarbeitung zu einer Vorlesung (”Skript”) —



”3-Insel-Bild”

Dr. Hermann Völlinger, Mathematik und IT Architektur
Stuttgart, Herbst 2024

Diese Version V1.0 (Datum: 2.1.2025) 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)v1.0.pdf

Dieses Skript wurde generiert mit LaTeX

Inhaltsangabe

Contents

Bilderverzeichnis

List of Figures

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 im Zusammenhang mit 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. So würdigt die gängige Literatur den Zusammenhang zwischen mathematischen Erkenntnissen und deren Umsetzung in konkreten Anwendungen nicht ausreichend. Dies wird besonders bei Algorithmen wie etwa der Regression oder auch bei "Support Vector Machine" deutlich. Hier gibt das Skript anschauliche Beispiele für die Anwendung der Mathematik im ML Algorithmus. Auch die anderen Beispiele des Skriptes geben hier gute Beispiele.
Dies mag etwa auch damit zu tun haben, das heutzutage mathematische Ausbildung in der Schule und in der Universität etwas den Bezug zur Realität verloren haben, und deshalb viele Schüler und Studenten den Nutzen und die Schönheit der Mathematik nicht erkennen können. Es gehört ja manchmal schon "zum guten Ton", wenn technische Hochschulabsolventen fast schon scherzhaft sagen, dass Sie mit Mathematik "nichts am Hut haben".
Dies ist leider sehr schade und ist zudem beim Lösen von ML Fragestellungen 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 Brücke "DWH nach ML" im "3-Insel-Titelbild"

Neben der Brücke zwischen Mathematik und ML, die den Hauptfokus der Arbeit darstellt, ist jedoch auch die Brücke zwischen DWH/Data-Management und ML wichtig, die einen essentiellen Input für das Verständnis von ML Modellen liefert. Zu diesem Thema mache ich in diesem Unterkapitel einige wichtige Anmerkungen und Aussagen.

Über die Brücke "DWH nach ML" fließen nicht nur Daten sondern auch Metadaten. Metadaten sind für das Verständnis eines DWH sehr wichtig. Siehe auch dazu die Vorlesung zu Thema DWH, siehe [**HVö-LecDWH23**].
Diese Metadaten spielen jedoch auch ein wertvolle Rolle bei der Entwicklung von ML Algorithmen und Modellen. Insbesondere wenn wir hier an das Thema vom Aufbau grosser Sprachmodelle (aka LLM) denken. Die Idee besteht darin, das Wissen dieser Metadaten in die sogenannten "Knowledge Graphen" (KG) zu übertragen. Die KGs können LLMs verbessern, indem sie externes Wissen für Inferenz und Interpretier-

barkeit zur Verfügung zu stellen. Siehe hierzu in Fig.1 auch das Thema "Domain-specific Knowledge" in [LLMsAndKGs]. Die Rolle von KGs und ihr Zusammenspiel mit LLMs ist sehr gut in der Referenz [LLMsAndKGs] beschrieben.
KGs sind jedoch schwer zu konstruieren und entwickeln sich von Natur aus weiter, was eine Herausforderung für die bestehenden Methoden der KGs darstellt. Aus diesem Grund ist der Beitrag den die Metadaten des DWH/Data-Management leisten können so wichtig. Siehe hierzu auch das Kapitel im Anhang über LLM's (i.e. ChatGPT).
Wo stecken also diese Metadaten im DWH? Hier möchte ich beispielhaft zwei Werkzeuge erwähnen und zwar die Werkzeuge zur Datenmodellierung (z.B. Infosphere Data Architect -IDA) und Data Governance (z.B. Infosphere Governance Catalog - IGC). Beide Werkzeuge wurden ausführlich in der DWH Vorlesung (siehe [HVö-LecDWH23], Seite 25 und Seiten 130-132) besprochen.

Die Brücke zwischen DWH/Data-Management und ML kann jedoch auch in umgekehrter Richtung genutzt werden: Beispiele sind die Anwendung von KI zur Automatisierung oder Rationalisierung von Datenerfassung, Datenbereinigung, Datenanalyse, Datensicherheit und anderen Datenverwaltungsprozessen. Sowohl traditionelle, regelbasierte KI als auch fortschrittlichere generative KI-Modelle können bei der Datenverwaltung helfen.

1.2.1 Details zu IDA und IGC

IDA: IBM InfoSphere Data Architect ist eine kollaborative Lösung für die Modellierung und das Design von Unternehmensdaten, die das Integrationsdesign für Business Intelligence-, Master Data Management- und Service-orientierte Architekturinitiativen vereinfacht und beschleunigt. IDA ermöglicht die Zusammenarbeit mit Anwendern in jedem Schritt des Datendesignprozesses, vom Projektmanagement über das Anwendungsdesign bis hin zum Datendesign. Das Tool hilft dabei, Prozesse, Dienste, Anwendungen und Datenarchitekturen aufeinander abzustimmen. Sehen Sie in [DHBW-Moodle] das Video: 'Demo-Infosphere Data Architect (IDA).mp4'.
Als weiteres Hilfsmittel zur Erstellung von Domänen spezifischen Datenmodellen bietet die Firma IBM auch eine Reihe von Industriemodellen an, siehe etwa "BDW" (Banking Data Warehouse) oder "IIW" (Insurance Information Warehouse). Vergleiche hierzu auch die Referenz [BDW+IIW]. Weitere Industriemodelle gibt es aber auch für die Domänen "Financial Services (FSDM)", "Healthcare (HDM)", "Telecommunications (TDW)", "Retail (RDW)" und "Energy and Utilities" (EUDM).
Diese Datenmodelle sind darauf ausgelegt, die spezifischen fachlichen Anforderungen der jeweiligen Branche zu erfüllen und bieten vordefinierte Datenstrukturen, die

die Implementierung und Anpassung beschleunigen können. Sie unterstützen auch die Einhaltung von Branchenstandards und gesetzlichen Anforderungen, was für Unternehmen von großer Bedeutung ist. Siehe [GIM-BFMDM].

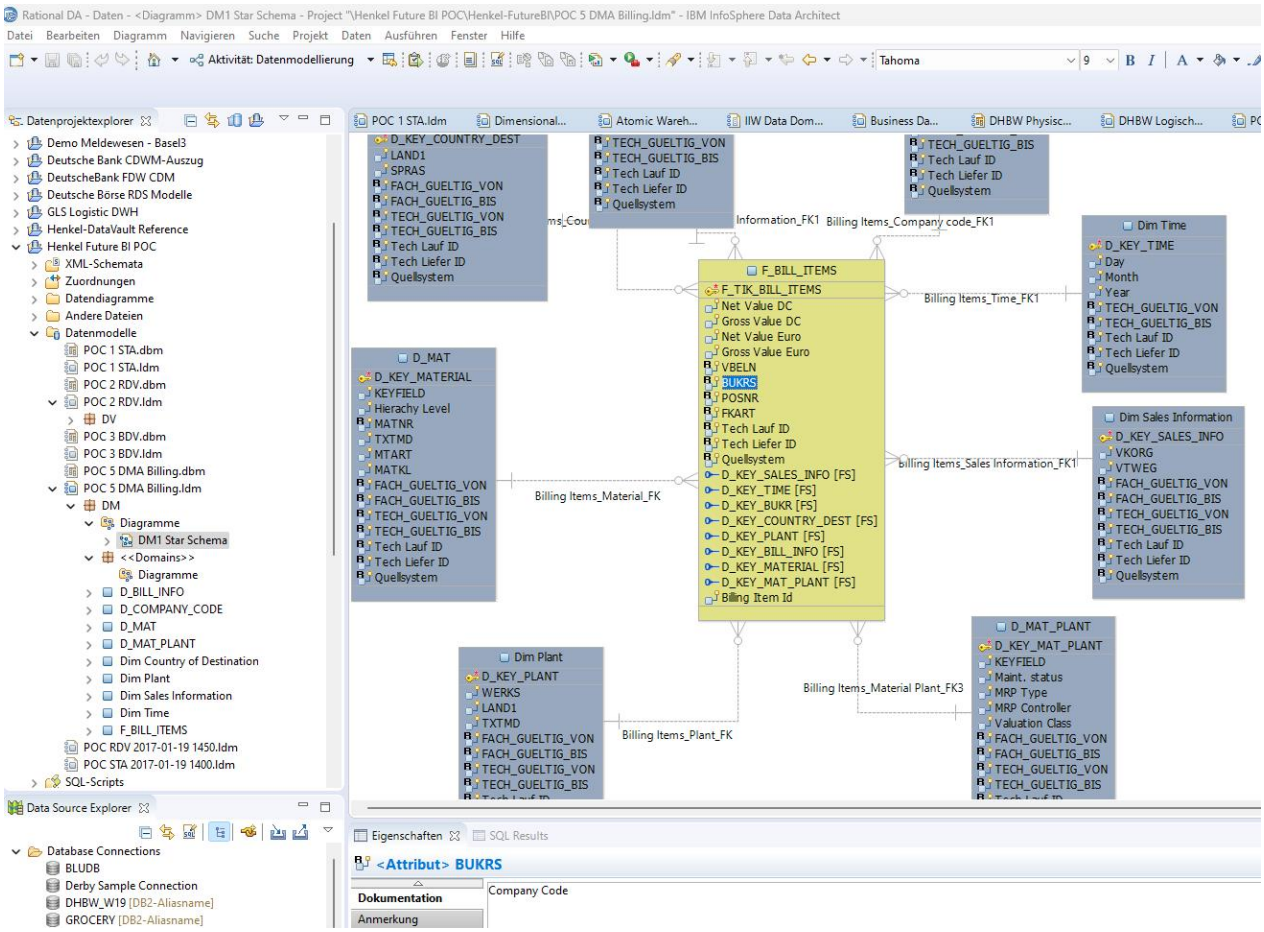


Figure 1: Beispiel für ein Datamart Modell mit Hilfe des Tools IDA

IGC: IBM InfoSphere Information Governance Catalog (IGC) ist ein interaktives, webbasiertes Tool, mit dem Benutzer ein Unternehmensvokabular und ein Klassifizierungssystem in einem zentralen Katalog erstellen, verwalten und gemeinsam nutzen können. Es hilft den Anwendern, die geschäftliche Bedeutung ihrer technischen Assets zu verstehen und bietet Such-, Browse- und Abfragefunktionen. Darüber hinaus können die Benutzer Asset-Sammlungen erstellen und Berichte über den Datenfluss zwischen Assets erstellen. Siehe in [DHBW-Moodle] die folgenden Dokumente und Videos: 'Geführte IGC Demo.pdf'; Geführte IGC Demo - Deutsch.pdf'; 'Demo-IGC-Teil1.mp4'; 'Demo-IGC-Teil2.mp4' und 'Demo Qutput-IGC-Teil2-Data Lineage.pdf'.
In allen oben erwähnten Industriemodellen werden auch komplette Business Glossare für IGC mitgeliefert.

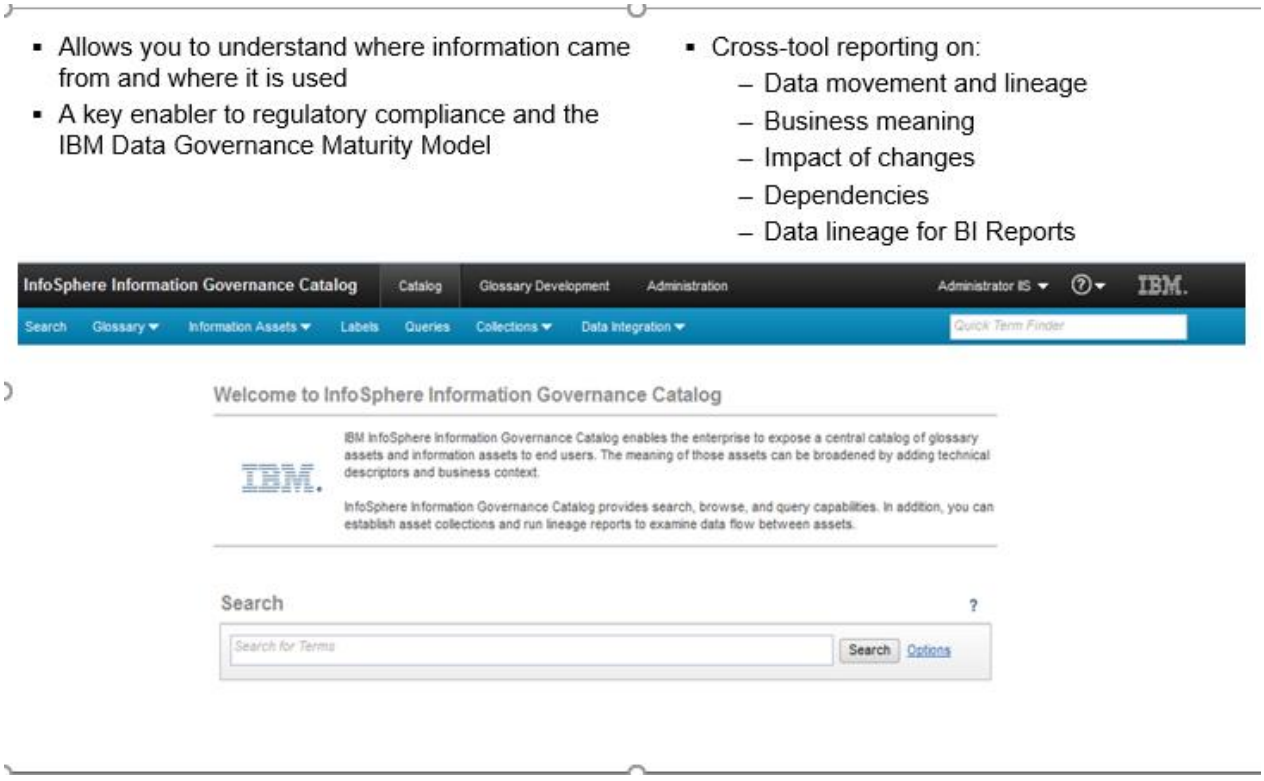


Figure 2: Einstiegsmaske im Tool IGC

1.2.2 Nutzen der IDA und IGC Strukturen für ML

Im nächsten Beispiel sehen wir wie Technische Assets (z.B. Spaltennamen einer Tabelle) mit fachlichen Begriffen ("Business Terms") verbunden werden. Dadurch erhalten wir eine fachliche Beschreibung und Kategorisierung/Strukturierung der technischen Assets im DWH.

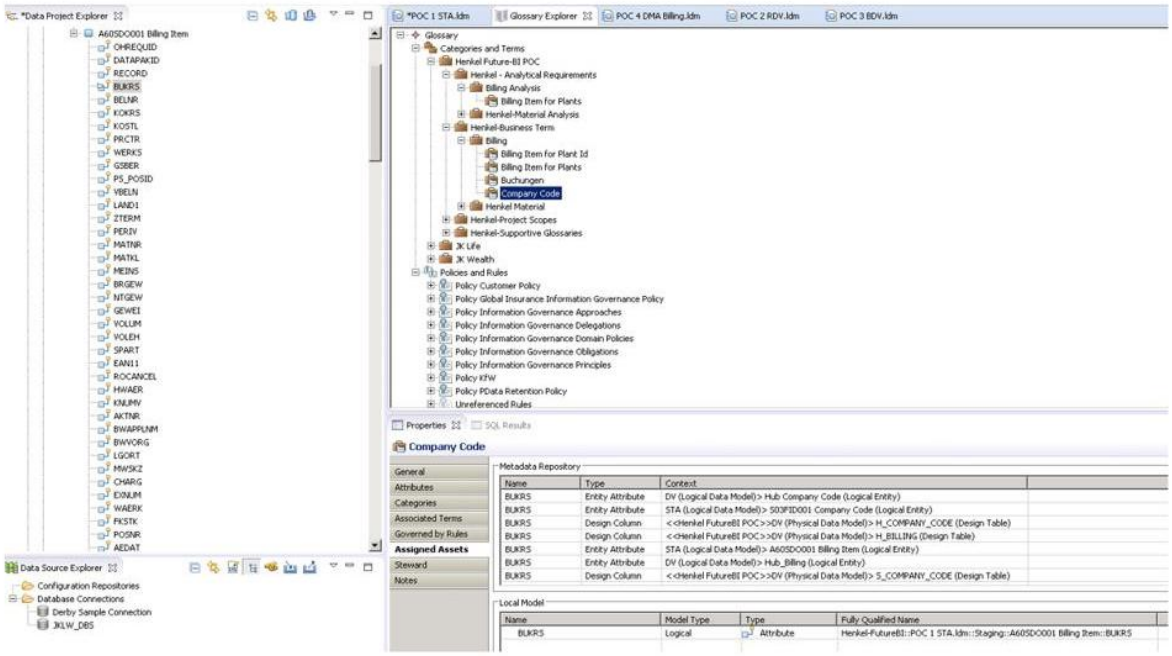


Figure 3: Business Term mit zugehörigen technischen Assets

Im obigen Beispiel sehen wir den Geschäftsbegriff „Buchungskreis“, zu dem das Feld mit dem Namen „BUKRS“ in 7 Tabellen in verschiedenen Datenschichten des DWH gehört. Diese Information erklärt die betriebswirtschaftliche Bedeutung dieses Feldes und gibt auch an, wo dieses Feld im DWH vorkommt. Dies ist sowohl für die Analyse/Auswertung der Daten als auch für den Datenpopulationsprozess (ETL) beim Laden der Daten in das DWH von entscheidender Bedeutung. Zudem ist diese Strukturierung bei Bau eines LLM Modells in ML sehr hilfreich.
Eine weitere nützliche Info ist die sogenannte "Data Lineage", die den kompletten Verarbeitungsprozess (inklusive und Datenablagen) im DWH abbildet. Die Data Lineage liefert auch eine Abhängigkeitsanalyse ("Impact Analysis") wenn sich etwa

Felder in den Quellsystemen eines DWHs verändern.
Die zugehörige Business Lineage zeigt nur die Datenablage ohne die ETL Prozesse dazwischen darzustellen. Damit gilt:
Data Lineage: Technisch orientiert, detaillierte Nachverfolgbarkeit der Datenflüsse und Transformationen innerhalb der IT-Infrastruktur.
Business Lineage: Geschäftlich orientiert, höher abstrahierte Darstellung der Datenflüsse und Nutzung in Geschäftsprozessen und -anwendungen.
Beide Konzepte sind wichtig für eine umfassende Daten-Governance-Strategie. Data Lineage bietet die technische Grundlage, während Business Lineage das Verständnis und die Transparenz aus der Geschäftsperspektive fördert.
Im nachfolgenden Bild sehen wir ein Beispiel für einen Data-Lineage Graphen für einen BI- Bericht in einem DWH.

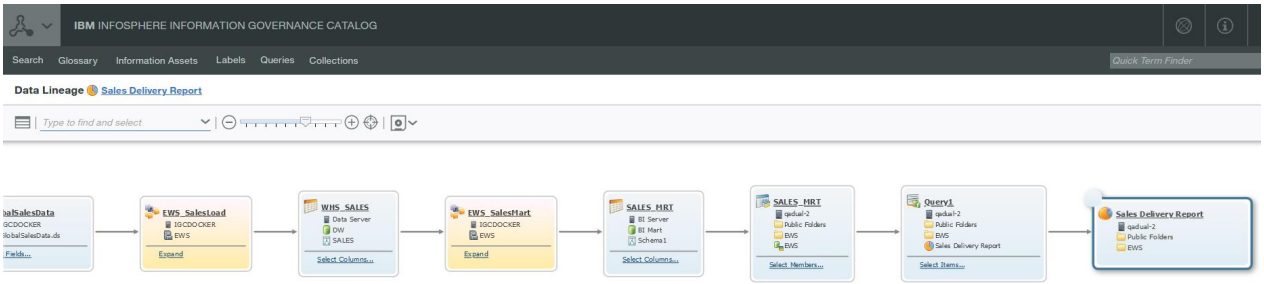


Figure 4: Data Lineage für einen BI Report mit Hilfe des Tools IGC

Nutzen für ein LLM: Zusammengefasst ist Data Governance beim Aufbau eines LLMs von zentraler Bedeutung, um sicherzustellen, dass die Datenqualität hoch ist, Bias minimiert wird, Datenschutz und -sicherheit gewährleistet sind, Transparenz und Verantwortlichkeit gefördert werden und letztlich ein leistungsfähiges, genaues und vertrauenswürdiges Modell entsteht.

Weiterführende bzw. vertiefende Literatur zu diesem Thema findet man auch unter den Referenzen [**TDWI-Feature Learning**] oder unter [**TDWI-Datenflüsse**].

1.3 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.
Einen guten Überblick über den Nutzen der Mathematik in Bereich der künstlichen

Intelligenz liefert auch der Übersichtsvortrag [MathAndAI]von Frau Prof. Gitta Kutyniok (Uni München) auf dem Internationalen Kongress der Mathematiker 2022.

1.4 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 neu ausformuliert auf Basis klassischer Methoden. Die lineare Regression nutzt mathematische Konzepte wie die Methode der kleinsten Quadrate, um Beziehungen zwischen Variablen zu modellieren.

2. Perzeptron (1957)
Der Perzeptron-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 ver-

bunden.

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 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 in der ersten Iteration ("$Iteration_O$") in der folgenden Figur.

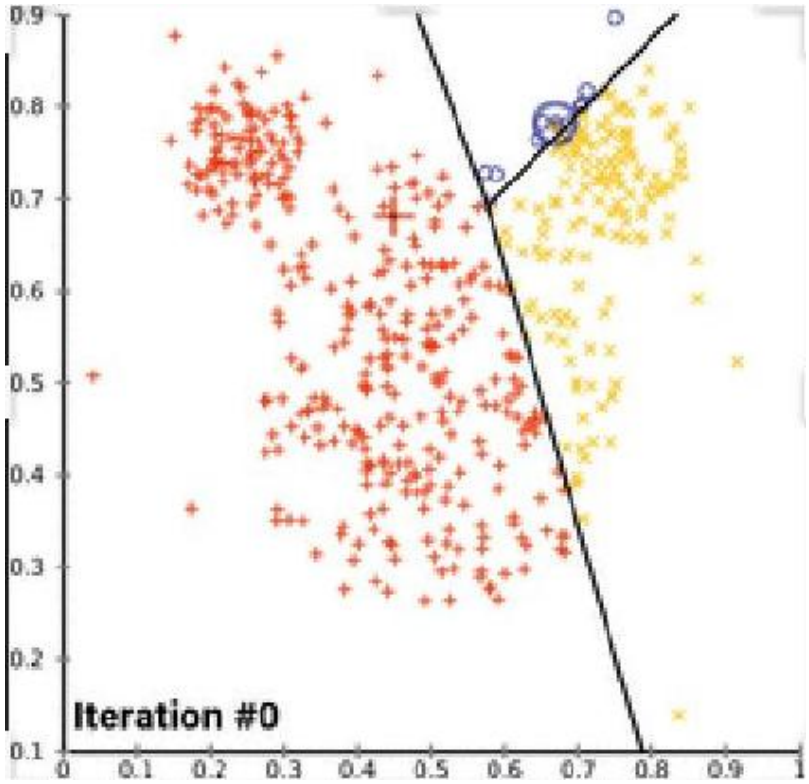


Figure 5: Erste 3-Means Konvergenz-Iteration0

Man kann nun relativ einfach daraus ein Python-Programm zu bauen um eine Animation dieser 3-Means-Konvergenz mit insgesamt 15 Iterationen (0...14) zu sehen.
Sie können diese Animation mit folgenden Link auf meinem 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.

3.2.1 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"): $\mathbf{m}_1^{(1)}, \ldots, \mathbf{m}_k^{(1)}$ aus dem Datensatz.
2. Zuordnung: Jedes Datenobjekt wird demjenigen Cluster zugeordnet, bei dem die Cluster-Varianz am wenigsten erhöht wird.

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

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 Beschreibung Python Programm zum k-Means-Clustering

Hier ist die Beschreibung des M-Means-Algorithmus zur Klassifikation der 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.

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.
Der **M-Means**-Algorithmus (oft als **K-Means** bezeichnet, wobei „K" hier „M" ist) ist ein gängiges unüberwachtes Lernverfahren zur Clusteranalyse, bei dem Datenpunkte in M Cluster unterteilt werden. In der Klassifikation der Iris-Blumen wird dieser Algorithmus verwendet, um die Blumen basierend auf ihren Merkmalen (z.B. Kelchblatt- und Kronblattlänge) in Gruppen zu unterteilen.

3.3.1 Laden der Iris-Daten

Zuerst wird das Iris-Dataset geladen. Das Iris-Dataset ist ein klassisches Beispiel in der Datenanalyse und enthält vier Merkmale für drei Arten von Iris-Blumen: *Iris Setosa*, *Iris Versicolor* und *Iris Virginica*:

$from \quad sklearn.datasets \quad import \quad load\_iris$
$import \quad pandas \quad as \quad pd$

Laden der Iris-Daten:

$iris = load\_iris()$
$df = pd.DataFrame(iris.data, columns = iris.feature\_names)$

3.3.2 Anwenden des k-Means-Algorithmus

Nun wird der M-Means-Algorithmus angewendet. Hierbei wird der Wert M (Anzahl der Cluster) angegeben, um die Daten in entsprechende Gruppen zu unterteilen:

$from \quad sklearn.cluster \quad import \quad KMeans$
$Anzahl \quad der \quad Cluster \quad M$
$M = 3$

M-Means (K-Means) initialisieren und anpassen:

$kmeans = KMeans(n\_clusters = M, random\_state = 42)$
$kmeans.fit(df)$

Zuweisung der Clusterlabels zu den Datenpunkten:

$df['Cluster'] = kmeans.labels\_$

3.3.3 Visualisierung der Cluster

Die resultierenden Cluster können visuell dargestellt werden, um die Verteilung der Iris-Daten zu analysieren. Dies kann mithilfe von Matplotlib oder Seaborn geschehen:

$import \quad matplotlib.pyplot \quad as \quad plt$
$import \quad seaborn \quad as \quad sns$

Paarweise Visualisierung der Daten mit den Cluster-Zuweisungen:

$sns.pairplot(df, \quad hue = \quad 'Cluster', \quad palette = \quad 'viridis')$
$plt.show()$

3.3.4 Evaluierung der Ergebnisse

Die Ergebnisse können durch Vergleichen der Cluster mit den tatsächlichen Arten der Iris-Blumen validiert werden, die im Datensatz enthalten sind:

$from \quad sklearn.metrics \quad import \quad accuracy_score$
$from \quad scipy.stats \quad import \quad mode$

Cluster-Labels mit den tatsächlichen Labels vergleichen:

$labels = kmeans.labels\_$
$true\_labels = iris.target$

Anpassung der Cluster-Labels an die tatsächlichen Labels (bei Bedarf):

$mapped\_labels = [mode(true\_labels[labels == i]) \quad [0] \quad [0] \quad for \quad i \quad in \quad range(M)]$

Accuracy berechnen:

$accuracy = accuracy_score(true\_labels, [mapped\_labels[label] \quad for \quad label \quad in \quad labels])$
$print(f"Accuracy : accuracy : .2f")$

3.3.5 Zusammenfassung

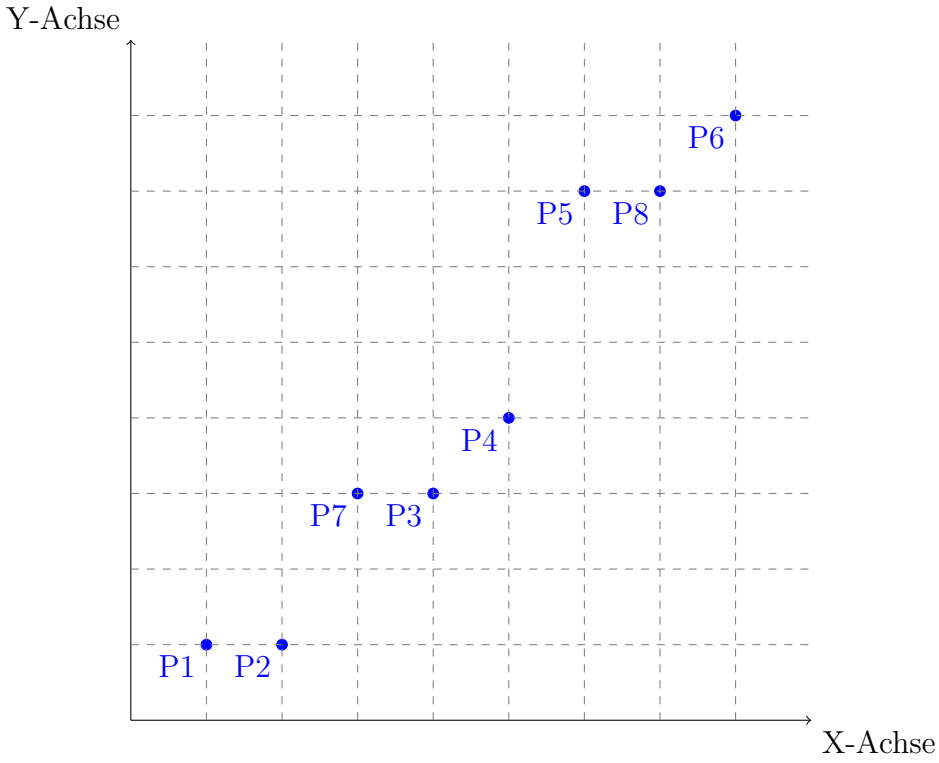
Das Python-Programm zur Klassifikation der Iris-Blumen mit dem M-Means-Algorithmus umfasst das Laden der Daten, das Anwenden des Algorithmus mit einer definierten Cluster-Anzahl, die Visualisierung der Resultate und eine Evaluierung der Cluster-Genauigkeit. Der M-Means-Algorithmus klassifiziert die Blumen anhand ihrer Merkmale in Gruppen, die mit den drei Arten von Iris-Blumen korrespondieren.

3.4 Manuelle Berechnung eines einfachen 3-Means-Clusters

Für sehr einfache Situationen lässt sich der K-Means-Algorithmus auch manuell berechnen, indem wir die entsprechenden Abstandsformeln (siehe oben) händisch berechned.
Durch ein solche kleine Übung wird der Algorithmus besser eintrainiert und auch besser verstanden.

3.4.1 Konkretes Beispiel

Angenommen, wir haben die folgenden acht Datenpunkte = ((1, 1), (2, 1), (4, 3), (5, 4), (6, 7), (8, 8), (3, 3), (7, 7) in einem zweidimensionalen Raum:



Wir möchten diese Punkte in k = 3 Cluster gruppieren. Diese Aufgabe wird in der Übung 3.1 durchgeführt.

3.5 Python-Beispiel für k-Means-Clustering für IRIS Blumen

Dieser Abschnitt beschreibt ein Code-Beispiele für den M-Means-Algorithmus zur Klassifikation der Iris-Blumen und enthält Python-Code für die Durchführung der Analyse. Es umfasst das Laden der Daten, die Anwendung des Algorithmus, die Visualisierung der Cluster und die Evaluierung der Ergebnisse.Siehe Lösungen in meinem Github:
https://github.com/HVoellinger/Mathematische-Grundlagen_von_ML/blob/main/Quellen_Math-ML/k-means-HVoe.pdf

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-Blumen 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%.

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.

3.6 Übungen zum Kapitel 3

3.6.1 Übung 3.1 - Manuelle Berechnung von "kleine" k-Means-Clustern

Berechnen Sie manuell das Beispiel im Unterabschnitt 3.4.1. Nach mehreren Iterationen (in diesem einfachen Beispiel könnten bereits 2 Iterationen ausreichen) würden wir eine stabile Zuordnung und endgültige Zentroiden haben.

3.6.2 Übung 3.2 - Erstelle ein Python Programm für Übung 3.1

Berechnen Sie erst manuell und anschließend per Python Programm (benutze den scikit k-Means Umsetzung) um die Cluster mit diesen Daten und berechnen.
Der Datenrahmen ist definiert durch die folgenden Punkten:
$df = pd.DataFrame \quad (x', y')$ wobei:
'x': [12, 20, 28, 18, 29, 33, 24, 45, 45, 52, 51, 52, 55, 53, 55, 61, 64, 69, 72], und
'y': [39, 36, 30, 52, 54, 46, 55, 59, 63, 70, 66, 63, 58, 23, 14, 8, 19, 7, 24] sind

Folgen Sie den Anweisungen aus https://benalexkeen.com/k-means-clustering-in-Python/ um zu den 3 Clustern zu kommen. Nutzen Sie dabei die Bibliothek "scikit-learn".

3.6.3 Übung 3.3 - Python Lösung zur Übung 3.1

Berechnen Sie per Python Pgm. analog zur Übung 3.2. auch die Datenpunkte von Übung 3.1 (nutze die scikit-learn Python Bibliothek).

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[1] 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.

[1] Der Gini-Index, auch bekannt als Gini-Koeffizient, wurde von dem italienischen Statistiker und Soziologen Corrado Gini entwickelt. Corrado Gini lebte von 1884 bis 1965 und war ein bedeutender Wissenschaftler, der sich intensiv mit statistischen Methoden und deren Anwendung in der Sozialwissenschaft befasste.

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

Das Gini-Index Verfahren ist eine Technik zur Aufteilung von Entscheidungsbäumen, insbesondere im Kontext von Klassifikationsproblemen. Der Gini-Index misst die Unreinheit oder den Mischungsgrad der Daten an einem bestimmten Punkt im Baum. Er wird oft für die Auswahl von Splits in einem Entscheidungsbaum verwendet.
Der "klassische" Gini-Index für zwei Zielvariablen i und j mit den entsprechenden

Wahrscheinlichkeiten p_i und p_j wird durch die folgende Formel dargestellt:

$$Gini = \sum_{i \neq j} (p_i) \cdot (p_j)$$

Betrachtet man den Fall mit binären Zielvariablen (YES=1 oder NO=0), so gibt es insgesamt vier mögliche Kombinationen der Wahrscheinlichkeiten sodass die Summe 1 ergibt:
$P(Target = 1) \cdot P(Target = 1) + P(Target = 1) \cdot P(Target = 0) + P(Target = 0) \cdot P(Target = 1) + P(Target = 0) \cdot P(Target = 0) = 1$.
Also folgt:
$P(Target = 1).P(Target = 0) + P(Target = 0) \cdot P(Target = 1) = 1 - [P(Target = 0)]^2 - [P(Target = 1)]^2$.

Damit wird der Gini-Index für einen bestimmten Knoten durch die folgende Formel dargestellt:

$$Gini(D) = 1 - \sum_{i=1}^{c} (p_i)^2$$

Hierbei steht D für den Datensatz im betrachteten Knoten, c ist die Anzahl der Klassen, und p_i ist der Anteil der Instanzen der Klasse i im Knoten.

Der Gini-Index ist eine Zahl zwischen 0 und 1, wobei 0 für einen reinen Knoten (alle Instanzen gehören derselben Klasse an) und 1 für einen maximal unreinen Knoten steht.
Wenn ein Entscheidungsbaum trainiert wird, wird der Gini-Index für potenzielle Splits berechnet, und der Split mit dem niedrigsten Gini-Index wird ausgewählt, um den Datensatz zu teilen. Der Prozess wird rekursiv für jeden entstehenden Teilbaum fortgesetzt.
Das Gini-Index-Verfahren ist eine beliebte Methode in Entscheidungsbäumen, weil es effizient ist und gut mit kategorialen und numerischen Daten umgehen kann. Es wird oft in Algorithmen wie dem CART (Classification and Regression Trees) verwendet.

Beschreibung des Verfahrens an einem einfachen Beispiel:
Angenommen, wir haben einen Datensatz mit einer Klassenzughörigkeit (A oder B) und zwei Merkmalen (Merkmal 1 und Merkmal 2). Die Zielvariable ist die Klassenzugehörigkeit (A oder B).

Wir möchten einen Entscheidungsbaum erstellen, um die Klasse vorherzusagen. Beginnen wir mit dem Wurzelknoten. Um den Gini-Index zu berechnen, betrachten

Table 1: Beispiel zur Knotenberechnung (GINI)

Beispiel	Merkmal 1	Merkmal 2	Klasse
1	2	3	A
2	1	4	A
3	3	2	B
4	4	1	B

wir jede mögliche Aufteilung und wählen diejenige mit dem geringsten Gini-Index. Angenommen, wir betrachten die Aufteilung nach Merkmal 1:

(a) Teilgruppe für (Merkmal 1) ≤ 2:

- Beispiele: 1, 2
- Gini-Index (≤ 2): $1 - (1/2)^2 - (1/2)^2 = 0.5$

(b) Teilgruppe für (Merkmal 1) > 2:

- Beispiele: 3, 4
- Gini-Index (> 2): $1 - (1/2)^2 - (1/2)^2 = 0.5$

Der gewichtete Durchschnitt der Gini-Indizes beträgt $0.5 \times \frac{2}{4} + 0.5 \times \frac{2}{4} = 0.5$. Das ist der Gini-Index für die Aufteilung nach Merkmal 1.

Wir würden denselben Prozess für Merkmal 2 durchführen und dann die Aufteilung wählen, die den geringsten gewichteten Gini-Index hat.
Das ist im Wesentlichen, wie das Gini-Index Verfahren in einem Entscheidungsbaum angewendet wird.

Wenn man nun allgemein k Merkmale hat, und ein Knoten p in k Partitionen ("Kindknoten") zerlegt ("split") werden könnte, wo wird der Gini-Index des Splits $Gini_{split}$ wie folgt berechnet:

$$Gini_{split} = \sum_{i=1}^{k} \frac{n_i}{n} \cdot Gini(i)$$

Hierbei ist:

$Gini_{split}$ ist der Gini-Index nach dem Split,
$Gini(i)$ ist der Gini-Index des Kindknoten i,
k ist die Anzahl der Kindknoten,
n_i ist der Anteil der Samples im i-ten Kindknoten.
n ist der Anteil der Samples im p-ten Knoten.

Man wählt als nächsten Knoten dann den Knoten mit dem geringsten $Gini_{split}$. Für ein besseres Verständnis dieser Formeln und ihrer genauen Anwendung verweisen wir hier auf den nächsten Abschnitt mit konkreten Beispielen und mit konkreten Zahlen. Wenn es sich bei der Zielvariablen um eine kategoriale Variable mit mehreren Ebenen handelt, ist der Gini-Index immer noch ähnlich. Wenn die Zielvariable k verschiedene Werte annimmt, lautet der Gini-Index:

$$Gini_{kat} = 1 - \sum_{i=1}^{k} p_i^2$$

Der **minimale** Gini-Index ist = 0, wenn alle Beobachtungen zu einem Label gehören. Der **maximale** Gini-Wert ist $1 - \frac{1}{k}$. Bei k = 2 ist $Gini_{max} = 0.5$.

4.2.2 Zusammenfassung: Algorithmus zum Gini-Index Verfahren

Hier ist eine Zusammenfassung, die die Schritte zur Erstellung eines Entscheidungsbaums unter Verwendung des Gini-Index beschreibt, inklusive der notwendigen Formeln und Erklärungen. Der Algorithmus setzt sich aus den folgenden vier Schritten zusammen:

1. Gini-Index

Der Gini-Index ist ein Maß für die Unreinheit eines Knotens. Für einen Knoten t mit mehreren Klassen wird der Gini-Index wie folgt berechnet:

$$\text{Gini}(t) = 1 - \sum_{i=1}^{C} p_i^2$$

wobei C die Anzahl der Klassen ist und p_i der Anteil der Instanzen der Klasse i im Knoten t.

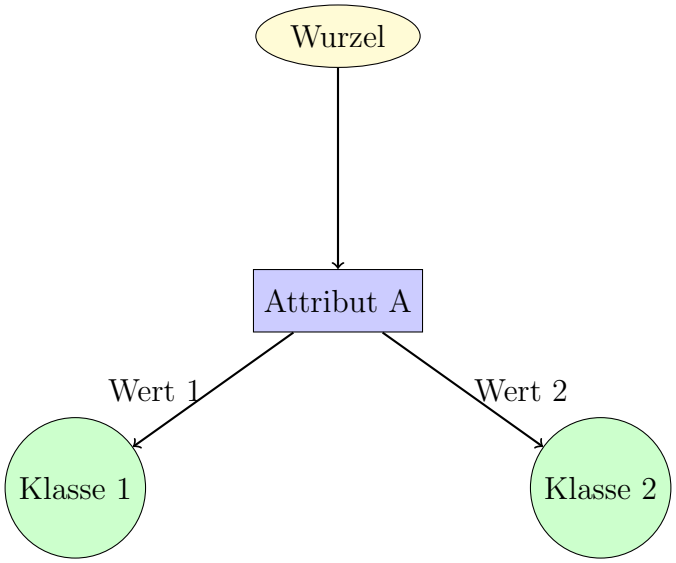
2. Splitten basierend auf dem Gini-Index

Für jedes Attribut und jeden möglichen Split berechnen wir den gewichteten Gini-Index der resultierenden Teilmengen. Der gewichtete Gini-Index nach einem Split ist gegeben durch:

$$\text{Gini}_{\text{split}} = \frac{n_{\text{left}}}{n_{\text{total}}} \cdot \text{Gini}_{\text{left}} + \frac{n_{\text{right}}}{n_{\text{total}}} \cdot \text{Gini}_{\text{right}}$$

Hierbei sind: - n_{left} und n_{right}: Anzahl der Instanzen in den linken und rechten Teilmengen, - n_{total}: Gesamtanzahl der Instanzen im aktuellen Knoten.
Der Split, der den geringsten gewichteten Gini-Index erzeugt, wird ausgewählt, um den Knoten zu teilen.

3. Beispiel für einen Entscheidungsbaum
Im Folgenden ist ein einfacher Entscheidungsbaum dargestellt, der auf dem Gini-Index basiert:



4. Rekursive Aufteilung
Der Entscheidungsbaum wird rekursiv aufgeteilt, bis eine der folgenden Bedingungen erfüllt ist:

- Alle Instanzen in einem Knoten gehören zur gleichen Klasse.
- Es gibt keine weiteren Attribute, um den Knoten weiter aufzuteilen.
- Eine maximale Tiefe des Baumes wurde erreicht.

.
Als nächstes sehen wir ein **Python-Beispiel** für eine Anwendung dieses Verfahrens auf den bekannten Iris-Blumen Datensatz. Erklärung des Codes:

- Gini-Index wird verwendet, um die "Unreinheit" eines Knotens zu messen.
- Splitting erfolgt basierend auf dem Split, der den niedrigsten gewichteten Gini-Index liefert.
- Der Entscheidungsbaum wird rekursiv erstellt, bis eine Stopping-Bedingung erreicht wird.
- Blätter des Baums repräsentieren die endgültigen Klassifikationen.

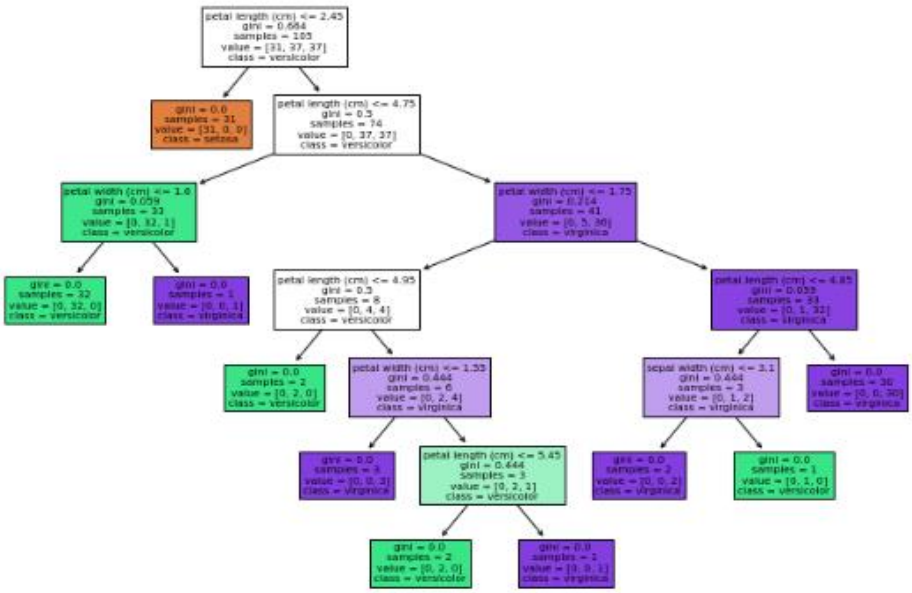
Der Gini-Index ist effizienter zu berechnen als Entropie und wird daher oft in Entscheidungsbäumen verwendet.
Erklärung der Schritte im Code:
1. Laden des Iris-Datensatzes:
Der Iris-Datensatz enthält Merkmale von Iris-Blumen und deren Klassifikation in drei Klassen.
2. Trainings- und Testdatensatz:
Der Datensatz wird in Trainings- und Testdaten aufgeteilt. 70 Prozent der Daten werden zum Trainieren verwendet, 30 Prozent zum Testen.
3. Erstellung des Entscheidungsbaummodells mit Gini-Index:
Das Entscheidungsbaum-Modell wird mit criterion="gini" erstellt, um den Gini-Index als Kriterium zu verwenden.
4. Modellbewertung:
Die Genauigkeit des Modells wird berechnet, indem das trainierte Modell auf die Testdaten angewendet wird.
5. Visualisierung:
Der Entscheidungsbaum wird mithilfe der plot_tree-Funktion von matplotlib visualisiert. Die Visualisierung zeigt die Aufteilungen basierend auf den Features des Iris-Datensatzes.

Voraussetzungen: Man benötigst die Bibliotheken scikit-learn und matplotlib. Sind diese noch nicht installiert sind, kann man dies mit dem Befehl "pip install scikit-learn matplotlib" tun.

In [2]:

```
# Entscheidungsbaum mit Gini-Index in Python
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
import matplotlib.pyplot as plt

# Iris-Datensatz laden
iris = load_iris()
X = iris.data
y = iris.target

# Daten in Trainings- und Testdaten aufteilen
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Entscheidungsbaum-Modell mit Gini-Index als Kriterium trainieren
clf = DecisionTreeClassifier(criterion="gini", random_state=42)
clf.fit(X_train, y_train)

# Modell evaluieren
accuracy = clf.score(X_test, y_test)
print(f"Genauigkeit des Modells: {accuracy * 100:.2f}%")

# Entscheidungsbaum visualisieren
plt.figure(figsize=(12,8))
tree.plot_tree(clf, feature_names=iris.feature_names, class_names=iris.target_names, filled=True)
plt.show()
```

```
Genauigkeit des Modells: 100.00%
```



In [ ]:

```
1
```

Figure 6: Python Pgm. zu IRIS-Daten mit GINI-Index-Verfahren

4.2.3 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 $Z = \{z_1, z_2, \ldots, z_m\}$ 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:

$$\mathrm{H}_1 = E[I] = \sum_{z \in Z} p_z I(z) = -\sum_{z \in Z} p_z \log_2 p_z.$$

Sei $z \in Z$, dann ist $p_z = P(X = z)$ die Wahrscheinlichkeit, mit der das Zeichen z

des Alphabets auftritt, oder gleichwertig:

$$\mathrm{H}_1 = -\sum_{i=1}^{m} p_i \log_2 p_i \text{ mit } p_i = p_{z_i}$$

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

Die Entropie $\mathrm{H_n}$ für Wörter w der Länge n ergibt sich durch:

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

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

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

$$(2) \qquad \mathrm{H} = \lim_{n \to \infty} \frac{\mathrm{H_n}}{n}.$$

Wenn die einzelnen Zeichen stochastisch voneinander unabhängig sind, dann gilt

$$\mathrm{H_n} = n\mathrm{H}_1 \text{ für alle } n, \text{ also } \mathrm{H} = \mathrm{H}_1.$$

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

Definition des ID3-Verfahrens und 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

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

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

4.2.4 Zusammenfassung: Algorithmus zum Endropie Verfahren

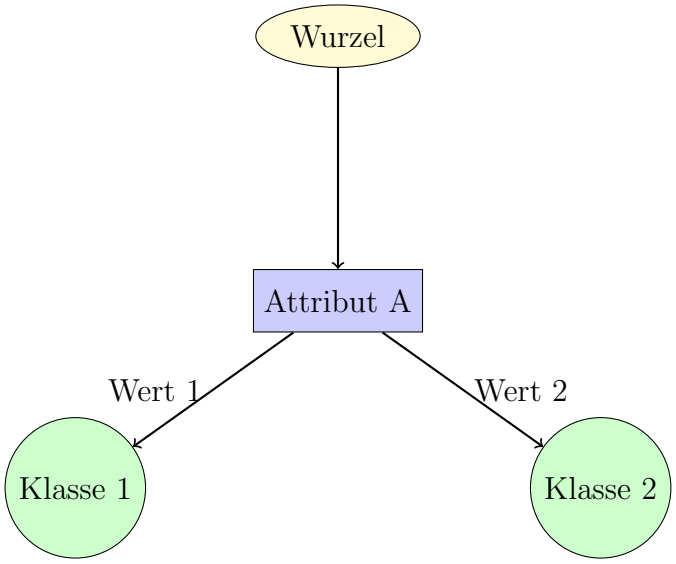
Der Algorithmus setzt sich aus den folgenden drei Schritten zusammen:
1. Entropie
Die Entropie $H(S)$ eines Datensatzes S für eine Binärklassifikation kann wie folgt berechnet werden:

$$H(S) = -p_1 \log_2(p_1) - p_2 \log_2(p_2)$$

wobei p_1 der Anteil der positiven Klassen und p_2 der Anteil der negativen Klassen ist.
2. Informationsgewinn
Der Informationsgewinn $IG(S, A)$ eines Merkmals A bei der Aufteilung des Datensatzes S wird berechnet durch:

$$IG(S, A) = H(S) - \sum_{v \in \text{Werte}(A)} \frac{|S_v|}{|S|} H(S_v)$$

wobei S_v die Teilmenge der Daten ist, die den Wert v für das Attribut A hat.
3. Entscheidungsbaum-Beispiel Im Folgenden ist ein einfacher Entscheidungsbaum dargestellt:



.

Als nächstes sehen wir ein **Python-Beispiel** für eine Anwendung dieses Verfahrens auf den bekannten Iris-Blumen Datensatz. Die Schritte sind analog wie beim GINI-Index-Verfahren.

In [1]:

```python
# Entscheidungsbaum mit Entropie in Python
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
import matplotlib.pyplot as plt

# Datensatz laden (Iris-Datensatz)
iris = load_iris()
X = iris.data
y = iris.target

# Datensatz in Trainings- und Testdaten aufteilen
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Entscheidungsbaum-Modell mit Entropie als Kriterium trainieren
clf = DecisionTreeClassifier(criterion="entropy", random_state=42)
clf.fit(X_train, y_train)

# Modell evaluieren
accuracy = clf.score(X_test, y_test)
print(f"Genauigkeit des Modells: {accuracy * 100:.2f}%")

# Entscheidungsbaum visualisieren
plt.figure(figsize=(12,8))
tree.plot_tree(clf, feature_names=iris.feature_names, class_names=iris.target_names, filled=True)
plt.show()
```

```
Genauigkeit des Modells: 97.78%
```

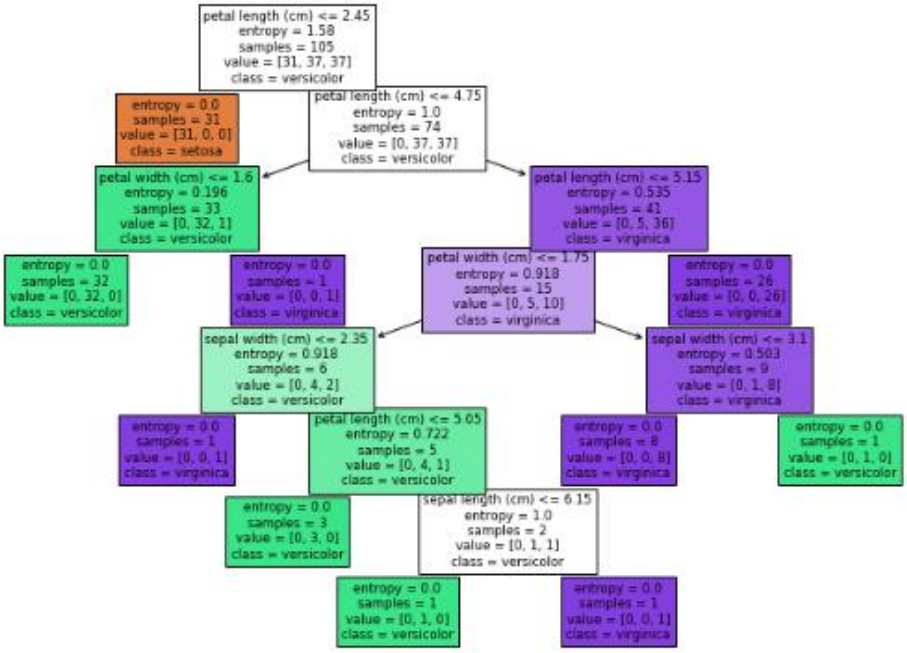


Figure 7: Python Pgm. zu IRIS-Daten mit Endropie-Verfahren

4.3 Beispiele für Gini-Index- und Entropie-Verfahren

Wir führen nun die beiden Verfahren an einem Datensatz "Playing Tennis Game" durch. Wir wollen zeigen, dass mit diesen Testdaten identische Ergebnisse erzielt werden.
Bei den Trainingsdaten handelt es sich um einen berühmten Datensatz aus der Welt des maschinellen Lernens, nämlich den Wetterdatensatz "Playing Tennis Game". Die Entscheidung (Spiel Y oder N) ist dabei basierend auf den Wetterbedingungen. Wir bauen einen entsprechenden Entscheidungsbaum auf.
In der ersten Zeile stehen die entsprechenden vier Bedingungen ("Features"): OUTLOOK, TEMP, HUMIDITY und WINDY und die Zielvariable ("Label") PLAY.
In den anschließenden 14 Zeilen stehen die Trainingsdaten.

Table 2: "Playing Tennis Game" Datensatz

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	cold	normal	false	yes
rainy	cold	normal	true	no
overcast	cold	normal	true	yes
sunny	mild	high	false	no
sunny	cold	normal	false	yes
rainy	mild	normal	false	yes
sunny	mild	normal	true	yes
overcast	mild	high	true	yes
overcast	hot	normal	false	yes
rainy	mild	high	true	no

Diese zwei Aufgaben sind analog zu den oben beschriebenen Algorithmen für das GINI-Verfahren und das Entropie-Verfahren als Übungen zu erledigen (siehe nächstes Kapitel).

Als Hilfe siehe auch die Lösung im Github:
https://github.com/HVoellinger/Mathematische-Grundlagen_von_ML/blob/main/Quellen_Math-ML/Homework_H4.5-DecTree_ID3.pdf

4.4 GINI-Index "Predictive Maintenance"

Im Spezialfalll "Predictive Maintennace" (deutsch: "Vorher-schauende Wartung") betrachten wir stetige Wertebereiche. In diesem Fall werden "Grenzwerte" berechnet, siehe Zeile 4 in der folgenden Wertetabelle:

Cheat	No		No		No		Yes		Yes		Yes		No		No		No		No		No	
	Taxable Income																					
	60		70		75		85		90		95		100		120		125		220			
	55		65		72		80		87		92		97		110		122		172		230	
	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>
Yes	0	3	0	3	0	3	0	3	1	2	2	1	3	0	3	0	3	0	3	0	3	0
No	0	7	1	6	2	5	3	4	3	4	3	4	3	4	4	3	5	2	6	1	7	0
Gini	0.420		0.400		0.375		0.343		0.417		0.400		*0.300*		0.343		0.375		0.400		0.420	

Figure 8: Stetige Wertetabelle für Entscheidungsbaum "Predictive Maintenance"

Wir berechnen den Gini-Index der ersten Zelle **Gini(55)**:

Gini(55) = $Frq(\leq 55) \cdot Gini(\leq 55)$+ Frq($>55$)* Gini($>55$)
wobei Frq(X):= $\frac{Anzahl\ aller\ Werte\ in\ der\ Zelle}{Anzahl\ aller\ Werte\ in\ den\ 4\ Zellen}$.
Wir berechnen: $Gini(\leq 55) = 1 - 0 - 0 = 1$
Gini(>55)=1-(3/10)^2-(7/10)^2=(100-9-49)/100=42/100=0.42.
Also folgt: Gini(55)=0/10*1+10/10*0.42=**0.420**.

Analog bei der zweiten Zelle **Gini(65)**:
$Gini(\leq 65) = 1 - 0 - 1 = 0$
Gini(> 65)=1-(3/9)^2-(6/9)^2=(81-9-36)/81=36/81=4/9
Also folgt Gini(65)=1/10*0+9/10*4/9=**0.400**.

Nachdem alle Gini Werte berechnet wurden, suchen wir die Zelle mit dem geringsten Gini-Wert, dies ist Gini(97). Hier wird der Entscheidungsbaum geteilt.
Rest analog wie bei nicht stetigen Werten.

4.4.1 Konkretes Beispiel für "Predictive Maintenance" in der Industrie

Dieser Anwendungsfall für Predictive Analysis (Maintenance) ist ein konkretes Projekt von IBM und BMW mit dem Ziel, die Qualität der Produktionsprozesse zu verbessern.
Es handelt sich hierbei um Daten einer Metallguss-Gießerei für den Guss von Metallteilen. Die Features für die Qualität der Gießerei sind dabei "Temperatur", "Druck und "Füllstand". Durch dieses Verfahren kann der "Ausschuss"(Abfall) der Gießerei um 20-30 Prozent verbessert werden.
Die konkrete Berechnung dieses Beispiels wir als Übung 4.3 durchgeführt.

Nr.	Anl	Typ	Temp.	Druck	Füllst.	Fehler
1001	123	TN	244	140	4600	nein
1002	123	TO	200	130	4300	nein
1009	128	TSW	245	108	4100	ja
1028	128	TS	250	112	4100	nein
1043	128	TSW	200	107	4200	nein
1088	128	TO	272	170	4400	ja
1102	128	TSW	265	105	4100	nein
1119	123	TN	248	138	4800	ja
1122	123	TM	200	194	4500	ja

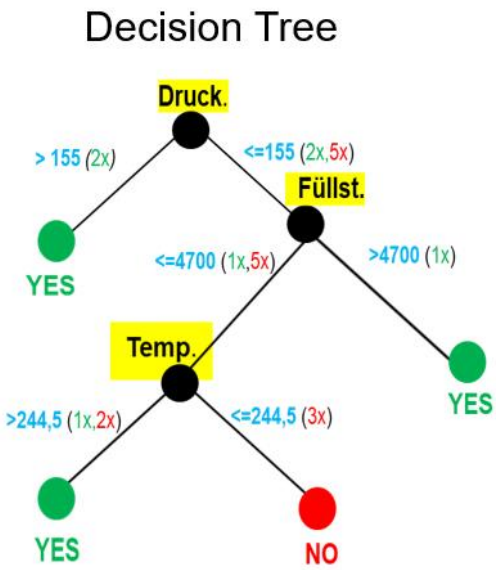


Figure 9: Daten und Ergebnisse einer Metallguss-Gießerei

Als nächstes folgen nun die Übungen zu Kapitel 4.

4.5 Übungen zu Kapitel 4

.

4.5.1 Übung 4.1 - Berechne das "Tennis Bsp." mit ID3- und GINI-Index Verfahren

Gruppenarbeit (2 Personen): Nutze die "Tennis" Daten aus Kapitel 4.3.1.
Teil A: Benutze das ID3 ("Iterative Dichotomoiser 3")-Verfahren zur Berechnung des Entscheidungsbaumes. Führe die Schritte des entsprechenden Algorithmus manuell durch.

Teil B: Benutze das GINI-Index-Verfahren zur Berechnung des Entscheidungsbaumes. Führe die Schritte des entsprechenden Algorithmus manuell durch.

4.5.2 Übung 4.2 - Schreibe ein Python Pgm. zur Umsetzung von Übung 4.1

Schreibe ein Python Programm das die Schritte der Übung 4.1 automatisiert.

4.5.3 Übung 4.3 - Berechne den Entscheidungsbaum "Predictive Maintenance"

Berechne mit den GINI-Index den Entscheidungsbaum für das "Predictive Maintenance" Beispiel aus Kapitel 4.4. Führe dabei manuell die folgenden Schritte durch:

- Berechne die "Frequenz-Matrix" für die Features "Temp", "Druck" und "Füllstand".
- Definiere den Root-Knoten (root node) durch Berechnung der Gini-Indizes aller drei Features und definiere dadurch dem optimalen "Split-Wert".
- Finalisiere den Entscheidungsbaum durch Berechnung der weiteren GINI-Indizes für die weiteren Features "Temp" und "Füllstand".

4.5.4 Übung 4.4 - Automatisiere die Schritte der obigen Übung als Python Pgm.

Schreibe ein Python Programm das die Schritte der Übung 4.3 automatisiert.

4.5.5 Übung 4.5 - Zusammenfassung einer Arbeit über "Predictive Maintenance"

Fasse die wesentlichen Ergebnisse der Artikels von Hans W. Dörmann Osuna: "Ansatz für ein prozessintegriertes Qualitätsregelungssystem für nicht stabile Prozesse" zusammen.
Hinweis: Die wesentlichen Ergebnisse findet man in den folgenden Kapiteln:
Kapitel 7.1 „Aufbau des klassischen Qualitätsregelkreises."
Kapitel 7.2. "Prädiktive dynamische Prüfung".

5 Lineare Regressionen(LR)

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 (simple Regression) oder mehreren unabhängigen (multiple Regression) (oder erklärenden) Variablen ("Features") zu analysieren und zu beschreiben. Das Hauptziel der Regression besteht darin, Vorhersagen für die abhängige Variable zu treffen, basierend auf den Werten der unabhängigen Variablen.
Die Grundidee hinter einem Regression ist es, eine Funktion zu finden, die die bestmögliche Anpassung an die gegebenen Daten bietet. Je nach Art der Daten und der Beziehung zwischen den Variablen gibt es verschiedene Arten von Regressionsmodellen, darunter:

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

2. **Polynominale - oder auch Multidimensionale 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.

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 $k \geqslant 2$. 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

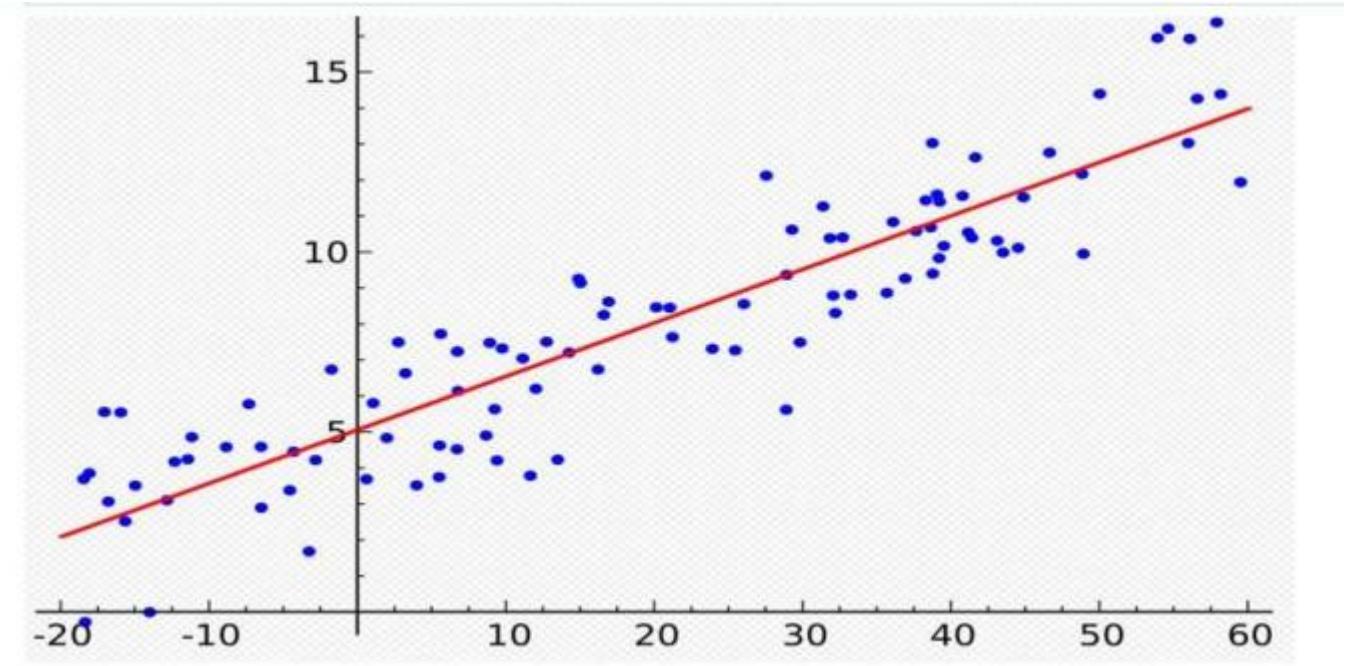
Mathematisch ("Geometrie") erhalten wir für k=1 eine Gerade in $\mathbb{R}^2$.
Wir schreiben die **Geradengleichung** als $y = a + b \cdot x$.
a ist dabei der Achsenabschnitt ("y-intercept" auf Englisch) und b die Steigung in x-Richtung ("x-slope").

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)

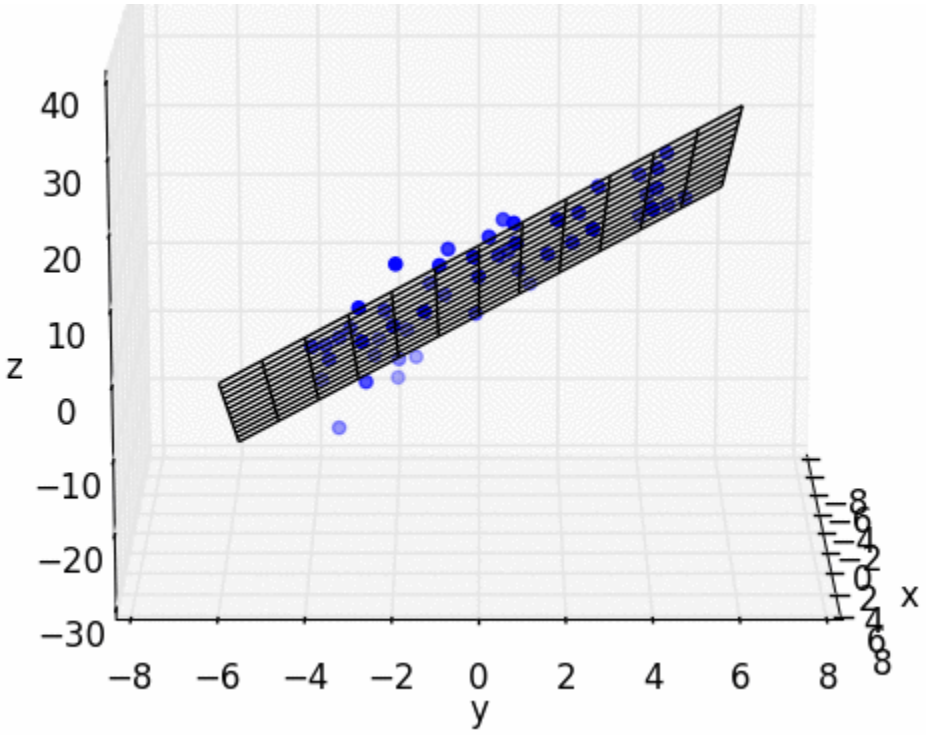
Für $k \geqslant 3$ erhalten wir eine Hyperebene in $\mathbb{R}^{k+1}$.

Die ersten zwei Fälle lassen sich leicht visualisieren:

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



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



Ein bewegtes Bild davon liegt in der Referenz [HVö-GitML20]: 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 ("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 R^2 , auch "Determinationskoeffizient" (lateinisch: "Determination", "Abgrenzung, Bestimmung" bzw. "determinare" "eingrenzen", "festlegen", "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 *Adj.*R^2 verwendet. Das adjustierte Bestimmtheitsmaß bestraft im Gegensatz zum unadjustierten Bestimmtheitsmaß die Aufnahme jeder neu hinzugenommenen abhängigen und unabhängigen Variable.

Die Regressionsgerade $f(x)$ als Schätzer (Modellfunktion) für den Zusammenhang von Größe und Gewicht der Probanden. $f(xi) = f_i$ 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 des Bestimmtheitsmaß 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:

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

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:

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

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

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

Für die Definition von R^2 brauchen wir Sum of Squares Error (SSE) und Sum of Squares Total (SST). Die Definition von R^2 ist gegeben durch die Differenz von 1 und dem Quotienten SSE/SST. Wir bezeichnen R^2 im deutschen Sprachgebrauch auch als "Bestimmtheitsgmaß":

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

Anmerkungen und Definition von MSE:

Bei der Nutzung von Python-Programmen, insbesondere bei der Nutzung der NumPy Bibliothek, wird oft noch eine weitere Kennzahl ("Measure") zur Messung der Güte der Regression benutzt. Diese Kennzahl ist "Mean Squared Error" (MSE) und ist eine Abwandlung der Definition von SSE (D-5.2). MSE ist definiert durch:

$$\texttt{(D-5.5): Mean Squared Error (MSE)} := \frac{1}{n} \cdot \sum_{i=1}^{n} \Big(y_i - f_i\Big)^2 = \frac{1}{n} \cdot \sum_{i=1}^{n} \varepsilon_i^2$$

Erläuterungen zu obigen Definitionen:

In der einfachen linearen Regression haben die Begriffe SSE, SSR, SST und MSE 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 (Sum of Squares Total)**:
  SST steht für die "Gesamtsumme der quadrierten Abweichungen" (Sum of Squares Total). 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 $\bar{y}$.

- **MSE (Mean Squared Error)**:
  Die MSE-Kennzahl ist eine weit verbreitete Metrik zur Beurteilung der Güte einer linearen Regression, einem statistischen Verfahren zur Vorhersage von abhängigen Variablen anhand von unabhängigen Variablen. Die MSE-Kennzahl quantifiziert, wie gut die Regressionsgerade die tatsächlichen Datenpunkte approximiert.Insgesamt ist der MSE eine nützliche Kennzahl, um die Genauigkeit eines linearen Regressionsmodells zu bewerten. Es ist jedoch wichtig zu beachten,

dass je nach Anwendungsfall auch andere Metriken in Betracht gezogen werden können, um die Vorhersagequalität zu beurteilen.

Die Beziehung zwischen den Kennzahlen SST, SSE und SSR 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 $\mathbf{R^2}$ 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 ein guten Eindruck von den Faktor $\frac{SSE}{SST}$.

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

Nun wollen wir die ersten offensichtlichen mathematische Aussagen zu $\mathbf{R^2}$ machen. Zur leichteren Schreibweise nutze ich die folgenden Notationen:

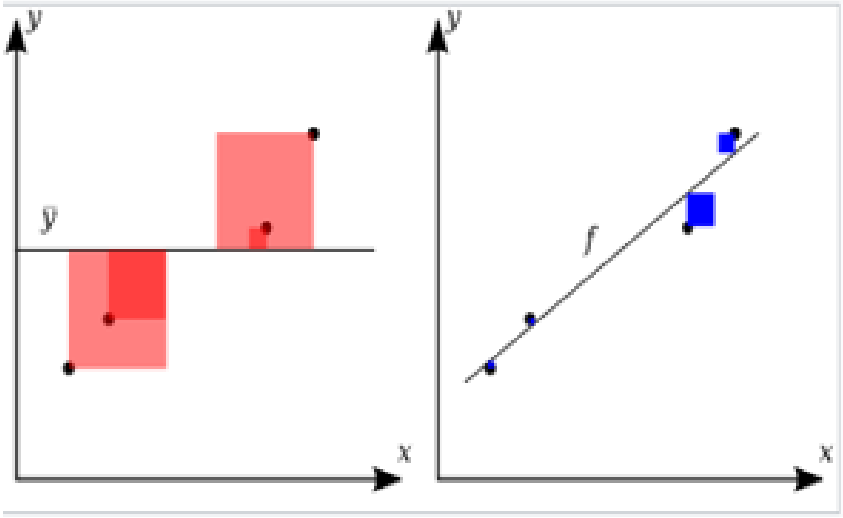


Figure 10: Geometrische Darstellung von SSE und SST

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

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) $0 \leqslant R^2 \leqslant 1$ "Begrenzung"

(ii) $R^2 = 0 \Leftrightarrow SSE = SST$ "Minimum"

(iii) $R^2 = 1 \Leftrightarrow SSE = 0$ "Maximum"

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

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

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

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

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

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

daraus folgt:

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

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):

Leicht können dann die folgenden 3 Aussagen bewiesen werden:

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

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

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

Beweis:
"straightforward":

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

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

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

5.3.3 Definition der mLR-Kennzahl $Adj.R^2$

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

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

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

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

Die Kennzahl $Adj.R^2$, 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 (Datenpunke) und k = Anzahl der unabhängigen Variablen (Prädiktoren) und definieren wir df := n-k-1 ("Anzahl der Freiheitsgrade"). Dann ergibt sich, unter Nutzung von (D-5.4), folgende Formel:

(D-5.5) $$Adj.R^2 := 1 - (1 - R^2) \cdot (\frac{n-1}{n-k-1}) = 1 - (\frac{SSE}{SST}) \cdot (\frac{n-1}{n-k-1})$$

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

Figure 11: Zusammenfassung der Formeln der Linearen Regression

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

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 Fälle (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. Beispielhaft 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

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

berechnet werden, wobei adj(A) die adjungierte (auch bekannt als die adjazente oder komplementäre) Matrix von A ist.
Die adjungierte Matrix von A (oft als 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)x(n-1)-Matrix erhalten.

Berechnung der Determinante $det(A)$

Die Determinante von einer (nxn)-Matrix A wird bezeichnet durch folgende Notation:[2]

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

und kann mit dieser allgemeinen Formel berechnet werden:

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

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

[2] Senkrechte Striche vor und hinter der Matrix A bedeutet Determinante von A

einer $n \times n$ Matrix so, wie dies 1690 von Gottfried Wilhelm Leibniz[3] vorgeschlagen 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 $\mathtt{det}(A)$) über die Formel:

$$\mathtt{det}(A = \sum_{\sigma\in\mathcal{S}_n} \mathtt{sgn}(\sigma) \cdot \prod_{i=1}^{n} a_{i,\sigma(i)} = \sum\Big\{\mathtt{sgn}(\sigma) \cdot \prod_{i=1}^{n} a_{i,\sigma(i)} \;\Big|\; \sigma \in \mathcal{S}_n\Big\} \tag{Det-2}$$

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

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

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

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

Insgesamt haben wir

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

Somit ergibt sich mit der Formel (Inv):

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

Es wird empfohlen bei solchen Rechnungen immer eine Probe zu machen, indem

[3] G. W. Leibniz war einer der bekanntesten Mathematiker, Philosophen und Naturwissenschafter seiner Epoche.

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. 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 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:

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

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

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

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 adjungierte 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:

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

Beispiel für eine symmetrische 3×3 Matrix:

In diesem Beispiel wird die Inverse und Determinante der 3x3-Matrix A mithilfe des Laplace'schen[4] 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 3x3-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:

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

[4] Pierre-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.

2. Berechne die Kofaktoren der Matrix A:

$$C_{11} = \begin{vmatrix} d & e \\ e & f \end{vmatrix} = df - e^2,$$

$$C_{12} = C_{21} = -\begin{vmatrix} b & c \\ e & f \end{vmatrix} = -bf + ce,$$

$$C_{13} = C_{31} = \begin{vmatrix} b & d \\ c & e \end{vmatrix} = be - dc,$$

$$C_{22} = \begin{vmatrix} a & c \\ c & f \end{vmatrix} = af - c^2,$$

$$C_{23} = C_{32} = -\begin{vmatrix} a & b \\ c & e \end{vmatrix} = -ae + bc,$$

$$C_{33} = \begin{vmatrix} a & b \\ b & d \end{vmatrix} = ad - b^2.$$

3. Erstelle die adjungierte Matrix 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 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: $A \cdot A^{(-1)} = E(Einheitsmatrix)$ 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.
Das Notebook erstellt eine symmetrische 3x3-Matrix A, berechnet Det(A) und Adj(A), woraus sich Inv(A) ergibt. Insgesamt besteht das Programm aus 4 Schritten.

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 libary 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]: $adf - ae^2 - b^2 f + 2bce - c^2 d$

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]])
```

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):

Inverse(A) = 1/det(A)*Adj(A), wobei det(A) im 1. und 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]:
$$\begin{bmatrix} \frac{df-e^2}{adf-ae^2-b^2f+2bce-c^2d} & \frac{-bf+ce}{adf-ae^2-b^2f+2bce-c^2d} & \frac{be-cd}{adf-ae^2-b^2f+2bce-c^2d} \\ \frac{-bf+ce}{adf-ae^2-b^2f+2bce-c^2d} & \frac{af-c^2}{adf-ae^2-b^2f+2bce-c^2d} & \frac{-ae+bc}{adf-ae^2-b^2f+2bce-c^2d} \\ \frac{be-cd}{adf-ae^2-b^2f+2bce-c^2d} & \frac{-ae+bc}{adf-ae^2-b^2f+2bce-c^2d} & \frac{ad-b^2}{adf-ae^2-b^2f+2bce-c^2d} \end{bmatrix}$$

5.4.2 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):

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

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

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

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

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

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

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

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:

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

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

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

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 :

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

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

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:

$a = \frac{1}{det} \cdot (\bar{y} \cdot \sum x_i^2 - \bar{x} \cdot \sum x_i \cdot y_i)$ (nach Formel (3))
$= \frac{1}{det} \cdot (\bar{y} \cdot (\sum (x_i - \bar{x})^2 + n \cdot \bar{x}^2) - \bar{x} \cdot \sum x_i \cdot y_i)$ (P4.1-(i))
$= \frac{1}{det} \cdot (\bar{y} \cdot (\sum (x_i - \bar{x})^2 + n \cdot \bar{x}^2) - \bar{x} \cdot (n \cdot \bar{x} \cdot \bar{y} + \sum [(x_i - \bar{x}) \cdot (y_i - \bar{y})]$ (P4.1-(iii))
$= \frac{1}{det} \cdot (\bar{y} \cdot (\sum (x_i - \bar{x})^2) - \bar{x} \cdot \sum [(x_i - \bar{x}) \cdot (y_i - \bar{y})]$ (Doppelten Faktor kürzen)

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 $\widetilde{x_i} := (x_i - \bar{x})$ und $\widetilde{y_i} := (y_i - \bar{y})$ 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):

$$det = \sum \widetilde{x_i}^2$$

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 $\widetilde{x_i} = (x_i - \bar{x})$ 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 $det = \sum \widetilde{x_i}^2$. Für die Achsenabschnitte a und die Steigung b einer optimalen sLR - Geraden $y = a + b \cdot x$ gelten folgende Formeln:

(sLSF-I): $a = \frac{1}{det} \cdot (\bar{y} \cdot \sum (\widetilde{x_i}^2) - \bar{x} \cdot \sum [\widetilde{x_i} \cdot \widetilde{y_i}])$

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

(sLSF-II): $b = \frac{1}{det} \cdot \sum [\widetilde{x_i} \cdot \widetilde{y_i}]$

Dabei steht:
$\sum$ für die Summe über alle n Datenpunkte $:= \sum_{i=1}^{n} x_i$,
$\bar{x}$ für den Durchschnittswert der x-Werte $:= \frac{1}{n} \cdot \sum_{i=1}^{n} x_i$,
$\bar{y}$ für den Durchschnittswert der y-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 $det = \sum \widetilde{x_i}^2$:

$a = \frac{1}{det} \cdot (\bar{y} \cdot \sum \widetilde{x_i}^2 - \bar{x} \cdot \sum [\widetilde{x_i} \cdot \widetilde{y_i}])$ (sLSF-I)

$= \frac{1}{\sum \widetilde{x_i}^2} \cdot (\bar{y} \cdot \sum \widetilde{x_i}^2 - \bar{x} \cdot \sum [\widetilde{x_i} \cdot \widetilde{y_i}])$ (Einsetzen (K4.2))

$= \bar{y} + \bar{x} \cdot [\frac{\sum [\widetilde{x_i} \cdot \widetilde{y_i}]}{\sum \widetilde{x_i}^2}]$ (Kürzen)

$= \bar{y} + \bar{x} \cdot b$ (sLF-II)

q.e.d.

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 $\Longrightarrow SST = SSE + SSR$

Beweis:

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

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 = 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 $\overline{f} := f(\bar{x})$ dann kann man mit einfachen Umrechnungen (Matrizenrechnung)zeigen:

$$\overline{f} = \overline{y}$$

Beweis: Man kann den Beweis auf zwei Arten durchführen:
1. Beweis-Methode:
In Theorem (Th-5.3) wurde gezeigt, dass gilt: (*) $\quad 0 = \sum_{i=1}^{n}(y_i - f_i)(f_i - \bar{y})$
Ohne Einschränkug der Allgemeinheit (OE) kann man annehmen: $(f_i - \bar{y}) \neq 0$
dann folgt direkt $0 = \sum_{i=1}^{n}(y_i - f_i)$
2. Beweis-Methode:
Um den den Zusammenhang zwischen den geschätzten Werten $\overline{f} = \frac{1}{n} \cdot \sum_{i=1}^{n}(f_i)$ und den tatsächlichen Werten $\overline{y}$ in einer einfachen linearen Regression zu zeigen, setzen wir die "Schätzung für den Achsenabschitt a" =: $\hat{a}$ und die "Schätzung der Steigung b" =: $\hat{b}$ ein und führen die Berechnungen durch:

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

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

5.4.4 Detailberechnung für mLR(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 mLR(k=2)-Ebene"
Sei $\quad det = \sum \widetilde{x_i}^2 \cdot \sum \widetilde{y_i}^2 - (\sum \widetilde{x_i} \cdot \widetilde{y_i})^2$
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:

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

(mLSF-II): $b = \frac{1}{det} \cdot \left[\sum \widetilde{y_i}^2 \cdot \sum (\widetilde{x_i} \cdot \widetilde{z_i}) - \sum (\widetilde{x_i} \cdot \widetilde{y_i}) \cdot \sum (\widetilde{y_i} \cdot \widetilde{z_i})\right]$

(sLSF-III): $c = \frac{1}{det} \cdot \left[\sum \widetilde{x_i}^2 \cdot \sum (\widetilde{y_i} \cdot \widetilde{z_i}) - \sum (\widetilde{x_i} \cdot \widetilde{y_i}) \cdot \sum (\widetilde{x_i} \cdot \widetilde{z_i})\right]$

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):

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

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

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

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

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 $k \geqq 3$ verstärkt sich sogar dieser Effekt

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

Also erhalten wir:

$$0 = \sum_{i=1}^{n} (\widetilde{x_i} \cdot \widetilde{z_i} + b \cdot \widetilde{x_i}^2 + c \cdot \widetilde{x_i} \cdot \widetilde{y_i}) \qquad (2)$$

Analog für die partielle Ableitung nach c:

$$0 = \sum_{i=1}^{n} (\widetilde{y_i} \cdot \widetilde{z_i} + b \cdot \widetilde{x_i} \cdot \widetilde{y_i} + c \cdot \widetilde{y_i}^2) \qquad (3)$$

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 Algebra"):

$$\begin{bmatrix} \sum \widetilde{x_i}^2 & \sum (\widetilde{x_i} \cdot \widetilde{y_i}) \\ \sum (\widetilde{x_i} \cdot \widetilde{y_i}) & \sum \widetilde{y_i}^2 \end{bmatrix} \cdot \begin{bmatrix} b \\ c \end{bmatrix} = \begin{bmatrix} \sum (\widetilde{x_i} \cdot \widetilde{z_i}) \\ \sum (\widetilde{y_i} \cdot \widetilde{z_i}) \end{bmatrix}$$

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)= detA = ad - bc. Mit obigen Werten eingesetzt erhält man:

$$det = \sum \widetilde{x_i}^2 \cdot \sum \widetilde{y_i}^2 - (\sum \widetilde{x_i} \cdot \widetilde{y_i})^2 \tag{4}$$

$$\begin{bmatrix} b \\ c \end{bmatrix} =$$

$$\frac{1}{det} \cdot \begin{bmatrix} \sum \widetilde{y_i}^2 & -(\sum (\widetilde{x_i} \cdot \widetilde{y_i})) \\ -(\sum (\widetilde{x_i} \cdot \widetilde{y_i})) & \sum \widetilde{x_i}^2 \end{bmatrix} \begin{bmatrix} \sum (\widetilde{x_i} \cdot \widetilde{z_i}) \\ \sum (\widetilde{y_i} \cdot \widetilde{z_i}) \end{bmatrix} \tag{5}$$

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 $\overline{f} := f(\bar{x}, \bar{y})$ dann kann man mit einfachen Umrechnungen (Matrizenrechnung) zeigen:

$$\overline{f} = \overline{z}$$

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.

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

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

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 $i \in \{1, \cdots, n\}$.
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:

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

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

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

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

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

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

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):

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

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

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

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

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

Setzen wir dies in die Definition von SSE)(**w**) ein und nutzen die uns schon bekannte **Notation**($\widetilde{x_i} := (x_i - \bar{x})$ und $\widetilde{y_i} := (y_i - \bar{y})$).
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)x(k+1)-Matrix invertieren, sondern nur eine kxk-Matrix. Dies führt zu Vereinfachung der Formeln und Ersparnissen in den Berechnungen. So erhalten wir:

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

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:

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

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 $\widetilde{\mathbf{x}^{(\mathbf{i})}}$ mit $1 \lneq i \lneq n$ wobei n = Anzahl der Trainingstupel. Schreiben wir die einzelnen Komponenten (Skalare) der (nxk)-Matrix X und (kxn)-Matrix $X^\top$ aus, so ergibt sich:

$$X := \begin{pmatrix} \widetilde{x_1^{(1)}} & \dots & \widetilde{x_k^{(1)}} \\ & \vdots & \\ \widetilde{x_1^{(n)}} & \dots & \widetilde{x_k^{(n)}} \end{pmatrix}; \quad X^\top := \begin{pmatrix} \widetilde{x_1^{(1)}} & \dots & \widetilde{x_1^{(n)}} \\ & \vdots & \\ \widetilde{x_k^{(1)}} & \dots & \widetilde{x_k^{(n)}} \end{pmatrix} \qquad (A')$$

Nun haben wir folgendes:

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

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

$$\texttt{SSE}(\mathbf{w}) = \left(X \cdot \mathbf{w} - \widetilde{\mathbf{y}}\right)^\top \cdot \left(X \cdot \mathbf{w} - \widetilde{\mathbf{y}}\right) \qquad (6)$$

5.5.2 Berechnung des Gradienten zu SSE

Um das minimale $\texttt{SSE}(\mathbf{w})$ zu bekommen, sagt uns die Mathematik (Analysis I), müssen wir den "Gradienten" der Funktion $\texttt{SSE}(\mathbf{w})$ 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 $\texttt{MSE}(\mathbf{w})$ anhand der Gleichung (3) berechnet.
Unser Ziel ist die Minimierung des $\texttt{SSE}(\mathbf{w})$ durch die Wahl des geeigneten Gewichtsvektors Vektors $\mathbf{w}$. Eine notwendige Bedingung, damit $\texttt{SSE}(\mathbf{w})$ minimal ist, ist:

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

d.h. der Gradient von $\texttt{SSE}(\mathbf{w})$ in Bezug auf $\mathbf{w}$ muss Null sein.
Zur Vorbereitung der Berechnung von $\nabla\texttt{SSE}(\mathbf{w})$, berechnen wir zunächst den Gradienten von zwei einfacheren Funktionen.

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

Angenommen die Funktion $f : \mathbb{R}^n \to \mathbb{R}$ 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 $C = (c_{i,j})_{\substack{i=1,\cdots,n\\ j=1,\cdots,n}}$ schreiben und den Vektor $\mathbf{x}$ als $\mathbf{x} = \langle x_1, \cdots, x_n\rangle^\top$, 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:

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

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 $g : \mathbb{R}^n \to \mathbb{R}$ 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):

$$\begin{aligned}
\texttt{SSE}(\mathbf{w}) &= \left(X \cdot \mathbf{w} - \widetilde{\mathbf{y}}\right)^\top \cdot \left(X \cdot \mathbf{w} - \widetilde{\mathbf{y}}\right) \\
&= \left(\mathbf{w}^\top \cdot X^\top - \widetilde{\mathbf{y}}^\top\right) \cdot \left(X \cdot \mathbf{w} - \widetilde{\mathbf{y}}\right) && (A \cdot B)^\top = B^\top \cdot A^\top \\
&= \left(\mathbf{w}^\top \cdot X^\top \cdot X \cdot \mathbf{w} - \widetilde{\mathbf{y}}^\top \cdot X \cdot \mathbf{w} - \widetilde{\mathbf{y}}^\top \cdot X^\top \cdot \widetilde{\mathbf{y}} + \widetilde{\mathbf{y}}^\top \cdot \widetilde{\mathbf{y}}\right) \\
&= \left(\mathbf{w}^\top \cdot X^\top \cdot X \cdot \mathbf{w} - 2 \cdot \widetilde{\mathbf{y}}^\top \cdot X \cdot \mathbf{w} + \widetilde{\mathbf{y}}^\top \cdot \widetilde{\mathbf{y}}\right) && \mathbf{w}^\top \cdot X^\top \cdot \widetilde{\mathbf{y}} = \widetilde{\mathbf{y}}^\top \cdot X \cdot \mathbf{w}
\end{aligned}$$

Die Tatsache das gilt

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

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

$$\texttt{SSE}(\mathbf{w}) = \left(\mathbf{w}^\top \cdot X^\top \cdot X \cdot \mathbf{w} - 2 \cdot \widetilde{\mathbf{y}}^\top \cdot X \cdot \mathbf{w} + \widetilde{\mathbf{y}}^\top \cdot \widetilde{\mathbf{y}}\right) \tag{7}$$

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

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

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

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

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

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

Dies führt zur Gleichung

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

Dies kann umgeschrieben werden zu

$$\boxed{\left(X^\top \cdot X\right) \cdot \mathbf{w} = X^\top \cdot \widetilde{\mathbf{y}}} \tag{8}$$

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

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

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:

$$\boxed{\mathbf{w} = \left(X^\top \cdot X\right)^{-1} \cdot X^\top \cdot \widetilde{\mathbf{y}}}.$$

Insgesamt erhält man das folgende Theorem:

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

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

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

$$\text{(LSF-II):} \qquad \mathbf{w} = \left(X^\top \cdot X\right)^{-1} \cdot X^\top \cdot \widetilde{\mathbf{y}}$$

Anmerkung (A-5.1): Gleichung (LSF-II) gilt nur falls $(X^\top \cdot X)$ invertierbar[5] 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 (kxk)-Matrix $X^\top \cdot X$ oft invertierbar ist, können wir (wenn dies nicht zutrifft) auch die ursprüngliche Gleichung : $\left(X^\top \cdot X\right) \cdot \mathbf{w} = X^\top \cdot \widetilde{\mathbf{y}}$ 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 (kxk)-Matrix $X^\top \cdot X$ und danach den (kx1)-Vektor $X^\top \cdot \widetilde{\mathbf{y}}$
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 erweitern:

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

weiterhin definieren wir:

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

Die Herleitung der Normalform ist dabei analog wie mit dem "Bias-Trick". Wir er-

[5]Matrix A ist genau dann invertierbar (auch regulär oder nicht singulär genannt), wenn ihre Determinante $Det(A)$) ungleich 0 ist

halten 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:

Example of a "Least Squares Fitting" calculation for simple LR

Find the "least square fit" $y = b0 + b1x$ for the experimental data points: {(1 , 2) , (3 , 4) , (2 , 6) , (4 , 8) , (5 , 12) , (6 , 13) , (7 , 15)}

Solution:

Number of Point N=7 Mean-Values ("Mittelwerte"): [M(x),M(y)] = [28/7; 60/7] ~ [4; 8,5714]

Set up a table with the quantities included in the above formulas for b0 and b1 and also the quantities for the calculation of R^2:

	needed for calculation of b0 and b1				needed for calculation of R^2			SST = SSE + SSR ?
i	x_i	y_i	$x_i * y_i$	x_i^2	$y(x_i)$	$SSE=sum(yi-y(x_i))^2$	$SST=sum(yi-M(y))^2$	$SSR=sum(y(xi) - M(y))^2$
1	1	2	2	1	2,0357	0,001274	43,1833	42,7101
2	3	4	12	9	6,3929	5,726	20,8977	4,7459
3	2	6	12	4	4,2143	3,1887	6,6121	18,9843
4	4	8	32	16	8,5715	0,3266	0,3265	0
5	5	12	60	25	10,7501	1,5623	11,7553	4,7467
6	6	13	78	36	12,9287	0,00661	19,6125	18,9861
7	7	15	105	49	15,1073	0,01151	41,3269	42,718
sum	**28**	**60**	**301**	**140**		**10,822994**	**143,7143**	**132,8911**

Substitute these values into Formula I and II:

$b_0 = (140*60/7 - 4*301)/(140 - 7*16) = -28/196 = -1/7 \sim -0,14286$

$b_1 = (301 - 7*4*(60/7))/28 = 61/28 \sim 2,1786$

----> Regression-Line: $y = -1/7 + (61/28)*x$

$R^2 = 1 - Sum((yi-y(xi))^2)/Sum((yi-M(y))^2) = 1 - (10,822994 / 143,7143) \sim 0,9247$

Compare with Python

intercept: - 0.14285714285714057

slope: [2.17857143]

coeff. determination: 0.9247017892644135

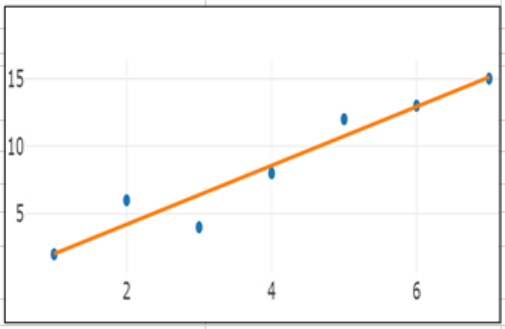
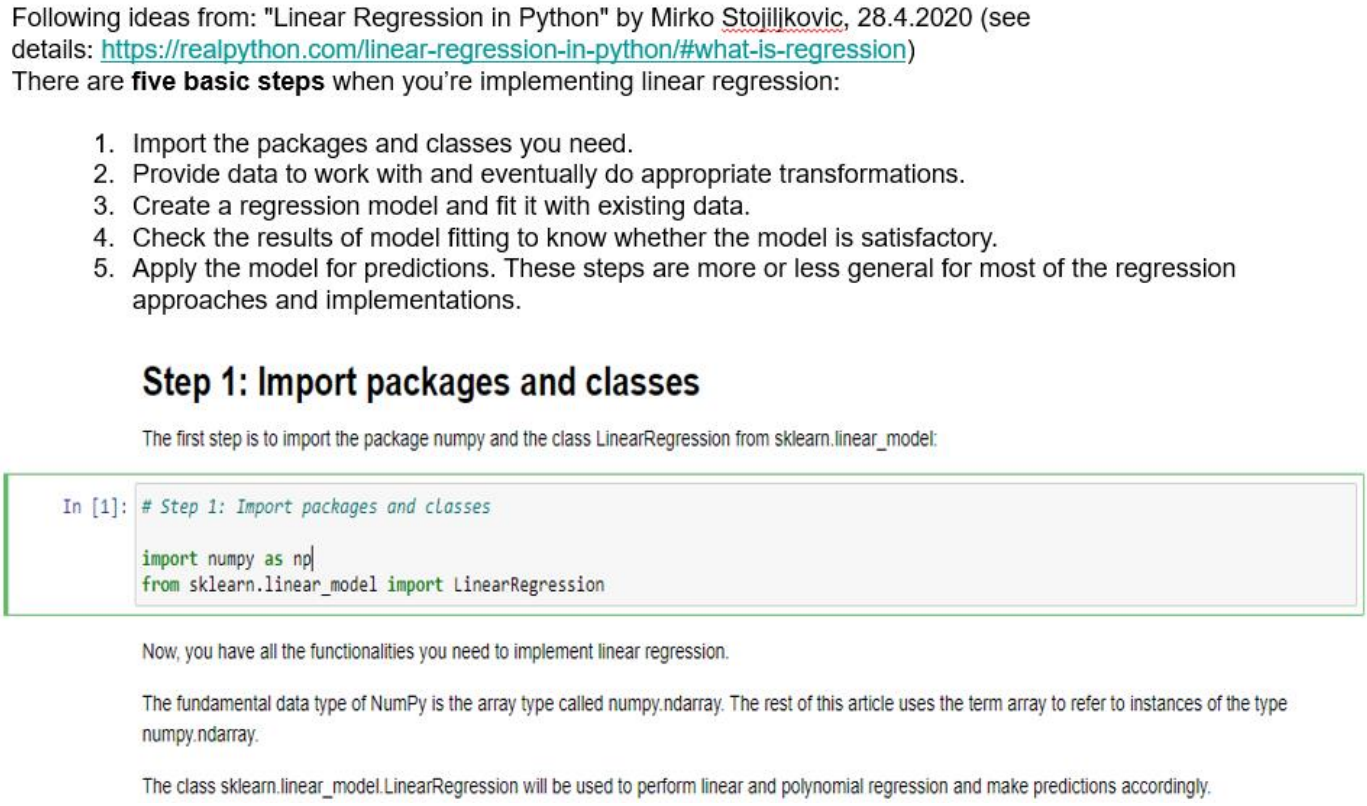


Figure 12: Manuelle Berechnung eines sLR Beispiels

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 Beispieles 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 13: 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.

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 14: sLR-Beispiel-Python-2/3

3. Ausgabe der Ergebnisse + Datenpunkte:
Bestimmungsgrat R^2 und Achsenabschnitt a und Steigung b der Regressiosgeraden 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.

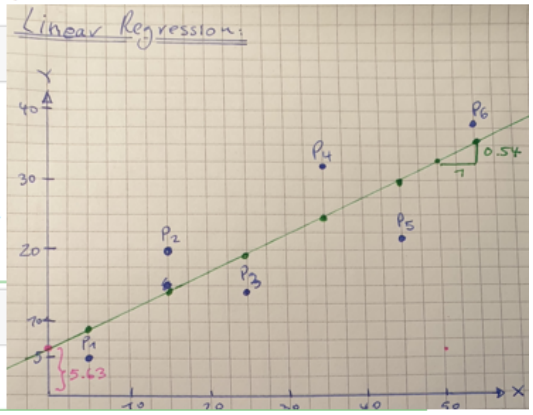
You can obtain the coefficient of determination (R^2) with .score() called on model:

```
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 15: sLR-Beispiel-Python-3/3

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 $det = \sum \widetilde{x_i}^2 \cdot \sum \widetilde{y_i}^2 - (\sum \widetilde{x_i} \cdot \widetilde{y_i})^2$
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:

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

$$b = \tfrac{1}{det} \cdot \left[\sum \widetilde{y_i}^2 \cdot \sum (\widetilde{x_i} \cdot \widetilde{z_i}) - \sum (\widetilde{x_i} \cdot \widetilde{y_i}) \cdot \sum (\widetilde{y_i} \cdot \widetilde{z_i})\right]$$

$$c = \tfrac{1}{det} \cdot \left[\sum \widetilde{x_i}^2 \cdot \sum (\widetilde{y_i} \cdot \widetilde{z_i}) - \sum (\widetilde{x_i} \cdot \widetilde{y_i}) \cdot \sum (\widetilde{x_i} \cdot \widetilde{z_i})\right]$$

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

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:

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:

	nedded for the calculation of a, b and c										
i	*xi*	*y i*	*z i*	*Xi:=xi-M(x)*	*Yi:=yi-M(y)*	*Zi:=zi-M(z)*	*Xi*Yi*	*Xi*Zi*	*Yi*Zi*	*Xi²*	*Yi²*
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(Xi²)*sum(Yi²)-(sum(Xi*Yi))²=3550*1406-(2070) 2=706400

a = Mean(z)-b*Mean(x)-c*Mean(y) ~ 22,25-0,4471*30 + 0.25500 *13* ~ 5, 522

b = (1/det)*(sum(Yi²)*sum(XiZi) sum(XiYi)*sum(YiZi))=(1/det)*(1406*2115-2070*1284) ~ 0,4471

c = (1/det)*(sum(Xi²)*sum(YiZi)- sum(XiYi)*sum(XiZi))=(1/det)*(3550*1284-2070*2115) ~ 0,2550

Compare with Python-Pgm (next slides):
intercept: 5.52257927519819
coefficients: [0.44706965 0.25502548]

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

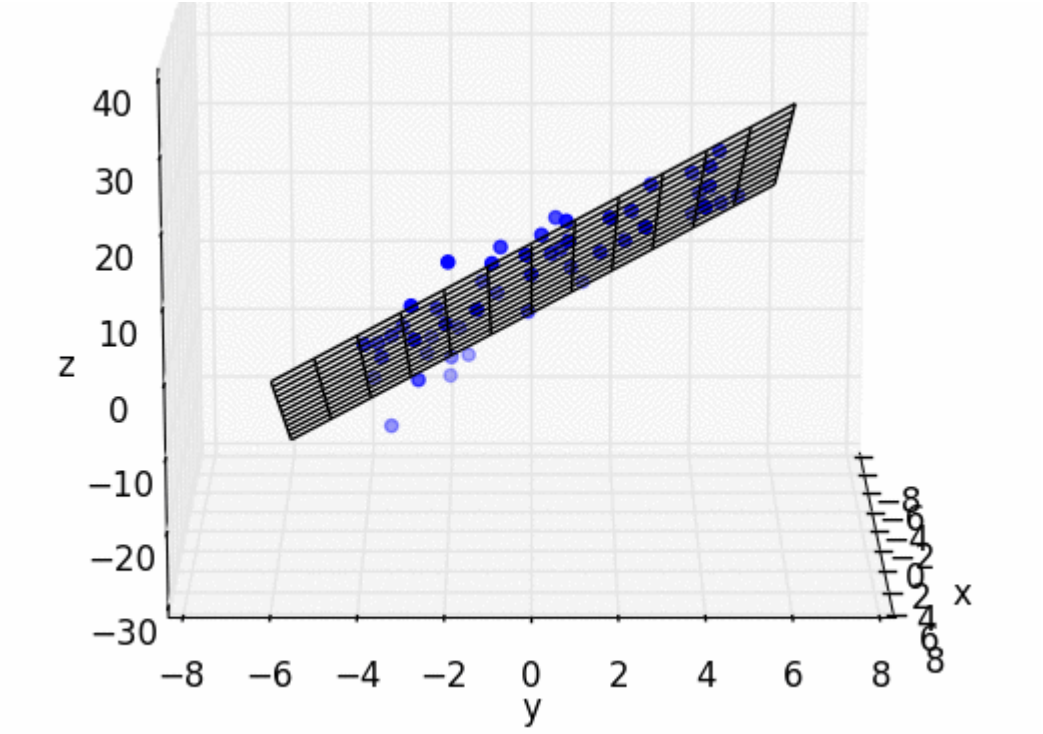
R^2 = 1 - SSE/SST ~ 0,86159 (details see notes-page)

coefficient of determination: 0.8615939258756776

--> Adj.R^2= 1 -(1-R^2)*(7/5) ~ 0,8062 (details see notepage)

Figure 16: Manuelle Berechnung mLR(k=2)

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



Ein bewegtes Bild davon liegt in der Referenz [HVö-GitML20]: https://github.com/HVoellinger/Lecture-Notes-to-ML-WS2020/blob/master/ML5-QYuIc.gif

Hinweis: Im Kapitel 7 ("Support Vector Machines") werden wir ein weitere Methode kennenlernen mit der man Regressions-Kurven oder -Ebenen bestimmen kann (siehe den Begriff "Support Vector Regression (SVR)" im Index-Verzeichnis.

5.8 Übungen zum Kapitel 5

Diese Übungen waren teilweise Bestandteil der 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 (*):

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

5.8.2 Übung 5.2 - Vermutung (V-5.1)

Beweisen Sie oder Widerlegen Sie folgende **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. $\mathbf{R^2}$ = 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:=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 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 - mLR(k=2) Beispiel "Studentisches Examen"

Berechne das mLR(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 mLR(x,y;z). Vergleiche und prüfe das Ergebnis mit einem Python-Programm.

Hinweis: Die gesuchte Hyperebene in $\mathbb{R}^3$ hat die folg. Darstellung:
z = 13.264 +2.488*x + 1.382*y. Siehe auch Lösungshinweise

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)).

5.8.11 Übung 5.1 - "Python-Pgm zu sLR-Verfahren(Iowa Housing Prices)"

Erzeuge ein Python Pgm. (Jupyter Notebook) für eine sLR-Verfahren mit den Daten des Beispiels "Iowa Housing Prices". Nutze die Informationen aus dem Video:
https://www.youtube.com/watch?v=Mcs2x5-7bc0

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[6] ("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

Formell lautet der Satz von Bayes:

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

Hier sind die einzelnen Bestandteile:

- $P(A|B)$: Die bedingte Wahrscheinlichkeit, dass das Ereignis A eintritt, gegeben, dass B bereits eingetreten ist.
- $P(B|A)$: Die bedingte Wahrscheinlichkeit, dass das Ereignis B eintritt, gegeben, dass A bereits eingetreten ist.
- $P(A)$: Die *a-priori* Wahrscheinlichkeit, dass das Ereignis A eintritt.
- $P(B)$: Die *a-priori* Wahrscheinlichkeit, dass das Ereignis B eintritt.

Anwendung Der Satz von Bayes wird häufig in Situationen verwendet, in denen man Rückschlüsse auf Ursachen auf der Grundlage von beobachteten Ergebnissen ziehen möchte. Ein klassisches Beispiel ist die medizinische Diagnose:

[6] Die Bayes'sche Regel, auch bekannt als Bayes' Theorem, wurde nach dem englischen Mathematiker und presbyterianischen Pfarrer Thomas Bayes benannt. Bayes lebte von 1701 bis 1761 und formulierte das nach ihm benannte Theorem in einem Essay mit dem Titel "An Essay towards solving a Problem in the Doctrine of Chances," der posthum 1763 von seinem Freund Richard Price veröffentlicht wurde

- A könnte die Hypothese sein, dass eine Person eine bestimmte Krankheit hat.
- B könnte ein positiver Test auf diese Krankheit sein.

Mit dem Satz von Bayes kann man die Wahrscheinlichkeit berechnen, dass eine Person tatsächlich krank ist ($P(A|B)$), wenn bekannt ist, dass der Test positiv ausgefallen ist. **Beispiel** Angenommen, 1% der Bevölkerung hat eine bestimmte Krankheit ($P(A) = 0{,}01$), und ein Test auf diese Krankheit ist in 99% der Fälle positiv, wenn die Person die Krankheit hat ($P(B|A) = 0{,}99$). Der Test ist aber auch in 5% der Fälle positiv, wenn die Person die Krankheit nicht hat ($P(B|\neg A) = 0{,}05$). Wir wollen wissen, wie hoch die Wahrscheinlichkeit ist, dass eine Person, die positiv getestet wurde, tatsächlich krank ist. Der Satz von Bayes gibt uns:

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)} = \frac{0{,}99 \cdot 0{,}01}{P(B)}$$

Die Wahrscheinlichkeit $P(B)$ kann berechnet werden als:

$$P(B) = P(B|A) \cdot P(A) + P(B|\neg A) \cdot P(\neg A)$$

$$P(B) = 0{,}99 \cdot 0{,}01 + 0{,}05 \cdot 0{,}99 = 0{,}0099 + 0{,}0495 = 0{,}0594$$

Nun können wir $P(A|B)$ berechnen:

$$P(A|B) = \frac{0{,}99 \cdot 0{,}01}{0{,}0594} \approx 0{,}1667$$

Das bedeutet, dass die Wahrscheinlichkeit, dass eine Person tatsächlich krank ist, wenn der Test positiv ist, etwa 16,67% beträgt.

6.2 Anwendungen des naiven Bayes-Klassifikator

1. Naive-Bayes-Theorem:
Das Naive-Bayes-Theorem ist eine statistische Formel, die auf dem Satz von Wahrscheinlichkeitsregeln beruht, bekannt als **Bayes-Theorem**. Es gibt an, wie Wahrscheinlichkeiten nach der Anwendung neuer Informationen aktualisiert werden. Für die Textklassifikation bedeutet dies, dass wir die Wahrscheinlichkeit berechnen, mit der ein Dokument einer bestimmten Klasse (z. B. "Positiv" oder "Negativ") angehört, basierend auf den darin enthaltenen Wörtern.
2. Feature-Extraktion:

Für jeden Text werden Features (Wörter) extrahiert, die als Eingabe für den Naive-Bayes-Klassifikator dienen. In der Regel werden diese Wörter als "Bag of Words" betrachtet, d. h. die Reihenfolge der Wörter im Text wird ignoriert.

3. Berechnung der Wahrscheinlichkeiten:
Für jede Klasse wird die Wahrscheinlichkeit berechnet, dass ein gegebenes Dokument dieser Klasse angehört. Dies erfolgt, indem die Wahrscheinlichkeiten für jedes im Dokument vorkommende Wort (Feature) multipliziert werden. Die Wahrscheinlichkeiten werden aus Trainingsdaten abgeleitet.

4. Klassifikation:
Das Dokument wird der Klasse zugeordnet, für die die berechnete Wahrscheinlichkeit am höchsten ist. Dies wird oft durch den Satz von Klassenlabels erreicht, die im Trainingsprozess erlernt wurden.

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

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

6.3 Mathematische Grundlagen bei Textklassifikatoren

6.3.1 Anwendung in der Textklassifikation

Bei der Textklassifikation wird $P(T|C)$ als Produkt der Wahrscheinlichkeiten der einzelnen Wörter in T gegeben C berechnet. Das heißt:

$$P(T|C) = P(w_1|C) \cdot P(w_2|C) \cdot \ldots \cdot P(w_n|C)$$

Hierbei sind $w_1, w_2, \ldots, w_n$ die Wörter im Text T. Der Klassifikator wählt dann die Klasse C mit der höchsten posterioren Wahrscheinlichkeit $P(C|T)$.

Beispiel: Spam-Filter:

Stellen wir uns vor, wir möchten eine E-Mail als "Spam" oder "Nicht-Spam" klassifizieren. Angenommen, wir haben eine E-Mail mit den Wörtern "win", "money", "now". Wir wollen die Wahrscheinlichkeit berechnen, dass diese E-Mail Spam ist, d. h. $P(\text{Spam}|\text{win, money, now})$.

$$P(\text{Spam}|\text{win, money, now}) \propto P(\text{win}|\text{Spam}) \cdot P(\text{money}|\text{Spam}) \cdot P(\text{now}|\text{Spam}) \cdot P(\text{Spam})$$

Die Klasse, die die höchste Wahrscheinlichkeit $P(\text{Spam}|\text{win, money, now})$ oder $P(\text{Nicht-Spam}|\text{win, money, now})$ hat, wird ausgewählt.

Zusammenfassung
Bayessches Lernen für Texte, insbesondere in der Form des Naive Bayes-Klassifikators, ist eine effiziente und weit verbreitete Methode zur Textklassifikation. Es nutzt den Satz von Bayes, um auf der Grundlage von Wortwahrscheinlichkeiten und *a-priori* Klassenwahrscheinlichkeiten die wahrscheinlichste Klasse für einen Text zu bestimmen.

6.3.2 "Laplace-Glättung"

Die Laplace-Glättung[7] (auch als Laplace-Smoothing oder additive Glättung bekannt) ist eine Technik, die in der Wahrscheinlichkeitstheorie und im maschinellen Lernen verwendet wird, um das Problem der Nullwahrscheinlichkeiten zu vermeiden. Dieses Problem tritt auf, wenn bestimmte Ereignisse in den Trainingsdaten nicht beobachtet wurden und daher eine Wahrscheinlichkeit von Null haben, was in vielen Modellen unerwünscht ist.

Hintergrund
Betrachten wir ein Beispiel aus der Textklassifikation mit einem Naive Bayes-Klassifikator. Der Klassifikator berechnet die Wahrscheinlichkeit einer Klasse C gegeben einem Text T basierend auf der Häufigkeit von Wörtern in der Trainingsdatenmenge. Wenn ein Wort in den Trainingsdaten für eine bestimmte Klasse nicht vorkommt, würde der Naive Bayes-Klassifikator eine Wahrscheinlichkeit von Null für diese Klasse berechnen, was dazu führen könnte, dass die Klasse insgesamt ausgeschlossen wird, selbst wenn sie sonst sehr wahrscheinlich wäre.

Definition
Die Laplace-Glättung löst dieses Problem, indem sie zu jeder beobachteten Häufigkeit

[7] Die Laplace-Glättung wurde nach dem französischen Mathematiker und Astronomen Pierre-Simon Laplace benannt. Pierre-Simon Laplace lebte von 1749 bis 1827 und war einer der bedeutendsten Mathematiker und Wissenschaftler seiner Zeit. Er entwickelte viele Konzepte in der Wahrscheinlichkeitstheorie und der Statistik, die bis heute von grundlegender Bedeutung sind.

eine kleine positive Konstante hinzufügt. Wenn man eine Konstante von 1 verwendet, lautet die Formel für die berechnete Wahrscheinlichkeit eines Wortes w_i in einer Klasse C:

$$P(w_i|C) = \frac{\text{Anzahl der Vorkommen von } w_i \text{ in } C + 1}{\text{Gesamtanzahl der Wörter in } C + V}$$

Dabei ist V die Gesamtzahl der verschiedenen Wörter im Vokabular (also die Anzahl der möglichen Wörter).

Anwendung

Wenn wir die Laplace-Glättung auf alle Wörter anwenden, sorgt dies dafür, dass kein Wort eine Wahrscheinlichkeit von Null erhält, selbst wenn es in den Trainingsdaten nicht vorkommt. Dadurch wird sichergestellt, dass das Modell besser generalisiert und auch mit Wörtern umgehen kann, die während des Trainings nicht gesehen wurden.

Beispiel

Angenommen, wir haben zwei Klassen, "Spam" und "Nicht-Spam", und wir berechnen die Wahrscheinlichkeiten von Wörtern in diesen Klassen. Wenn das Wort "Gewinn" in der Klasse "Spam" dreimal und in der Klasse "Nicht-Spam" nie vorkommt, würde ohne Glättung $P(\text{Gewinn}|\text{Nicht-Spam}) = 0$ sein.
Mit Laplace-Glättung berechnen wir jedoch:

$$P(\text{Gewinn}|\text{Nicht-Spam}) = \frac{0 + 1}{\text{Anzahl der Wörter in Nicht-Spam} + V}$$

Das Wort "Gewinn" erhält also eine kleine, aber nicht-null Wahrscheinlichkeit in der Klasse "Nicht-Spam", was verhindert, dass die Klasse "Nicht-Spam" ausgeschlossen wird, nur weil "Gewinn" nicht in den Trainingsdaten vorkommt.

Zusammenfassung

Die Laplace-Glättung ist eine einfache, aber wirkungsvolle Methode, um die Robustheit von Modellen zu erhöhen, indem sie das Problem von Nullwahrscheinlichkeiten verhindert. Sie wird häufig in Naive Bayes-Klassifikatoren und anderen probabilistischen Modellen eingesetzt, insbesondere in Situationen mit spärlichen Daten.

6.4 Konkrete Anwendung des naiven Bayes-Klassifikator

Als Beispiel für eine Textklassifikation unter Anwendung der Laplace-Glättung betrachten wir sechs gelabelte Trainingsdatensätze die in den zwei Klassen "Sports" oder "Not Sports" zugeordnet sind.

Table 3: Trainingsdatensatz zur Textklassifikation

No.	Trainingsdatensatz	Klasse
0	A great game	Sports
1	The election was over	Not Sports
2	Very clean match	Sports
3	A clean but forgettable game	Sports
4	It was a close election	Not Sports
5	A very close game	Sports

Als **Zieldatensatz** haben wir den Satz "A very close game". Es ist nun eine Klassifikation dieses Satzes durchzuführen (siehe Übung 6.1)
Zusatzfrage (Übung 6.2): Was wird passieren wenn wir den Zieldatensatz ändern auf "Hermann plays a very clean game"
Die Lösung dieser Aufgabe wird einmal manuell als Übung 6.1 und Übung 6.2 ausgeführt und einmal als Python Programm (Übung 6.3)
Betrachte also die folgende Tabelle mit den Trainingsdaten:

6.5 Übungen zum Kapitel 6

6.5.1 Übung 6.1 - Bsp. einer Bayes-Texklassifikation

Lösen Sie manuell das Beispiel aus obigen Kapitel unter Benutzung der Bayes-Textklassifikation (beachte dabei die Laplace-Glättung).

6.5.2 Übung 6.2 - Bsp. einer Bayes-Texklassifikation mit neuem Zielsatz

Analog zu Trainingsdaten aus der Übung 6.1 soll die Bayes-Textklassifikation für einen weiteren Zielsatz mit den gleichen **"gelabelten" Trainingsdaten** durchgeführt werden. Wir suchen jetzt eine Klassifikation für den Zielsatz **"Hermann plays a TT match"**

6.5.3 Übung 6.3 - Bayes-Textklassifikation in Python mit neuem Zielsatz

Definiere einen Algorithmus in Python (benutze Jupyter Notebook) um obige Berechnung zu automatisieren. Tipp: Vergleich analoge Beispiele in [**HVö-GitML20**] - https://github.com/HVoellinger/Lecture-Notes-to-ML-WS2020

7 Verfahren der Support Vector Machines (SVM)

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

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.
Mathematische Schreibweise:
Angenommen, wir haben eine Menge von Trainingsdatenpunkten: $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$, wobei $x_i \in \mathbb{R}^n$ der Eingabevektor und y_i als Klasse von Datenpunkten nur die Werte +1 oder -1 annehmen kann.
Für eine lineare SVM ist die mathematische Darstellung der Hyperebene dann gegeben durch:

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

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

und der Hyperebene muss maximal sein. Der Abstand zwischen zwei parallelen Hyperebenen, die die Support-Vektoren berühren, wird als Margin bezeichnet.

2. Optimierung:

Das Ziel der SVM ist es, den Margin zu maximieren, indem die Länge des Gewichtsvektors "w" minimiert wird. Das führt zu einem quadratischen Optimierungsproblem. In der linearen SVM lautet das Optimierungsproblem:

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

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

3. Kernel-Trick:

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

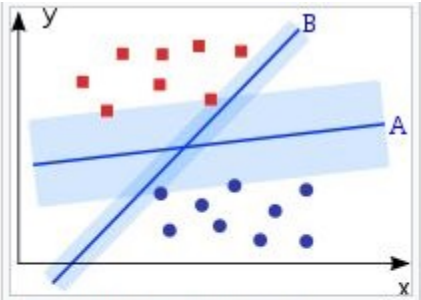
Anmerkung: Wie kommt es zu dem Namen "Kernel-Trick"?

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.

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.

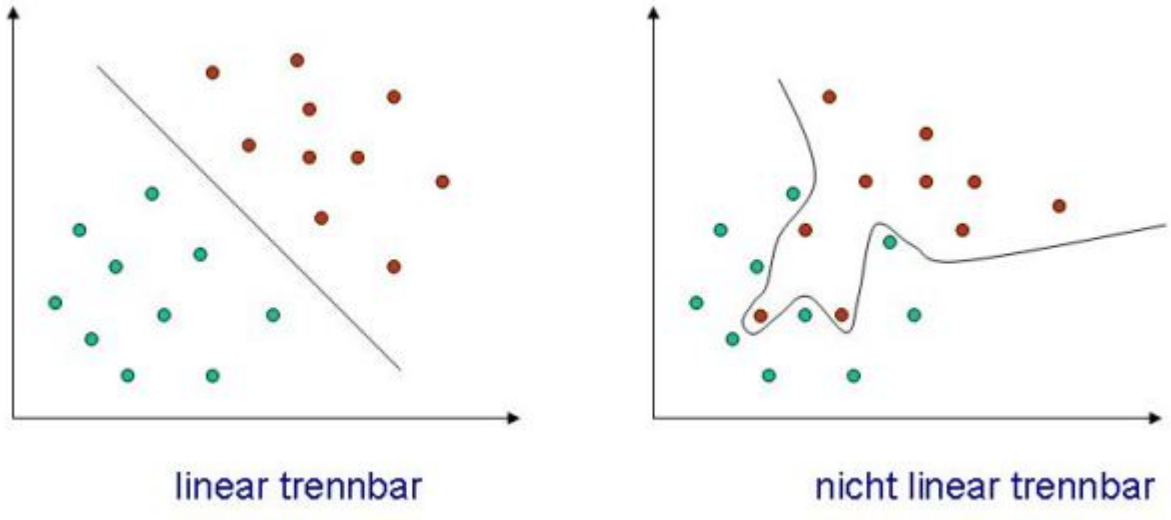


Zwei mögliche Trenngeraden mit verschiedenen Randgrößen

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

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.



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

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 Geometrisches 2-dim. Beispiel für Trennbarkeit

Stellen wir uns ein einfaches und anschauliches Beispiel in $\mathbb{R}^2$ vor, bei dem die Datenpunkte im ursprünglichen Raum (x_1, x_2) nicht linear trennbar sind.
Gesucht ist dann eine Abbildung $\Phi : \mathbb{R}^2 \to \mathbb{R}^3$, so das die Datenpunkte im höherdimensionalen Raum $(x, y, z) \in \mathbb{R}^3$ linear trennbar sind.
Der "Kernel-Trick" (siehe Kapitel über mathematische Grundlagen) ermöglicht es der SVM, die Entscheidungsgrenze in diesem höherdimensionalen Raum zu finden, ohne die zusätzlichen Merkmale z tatsächlich berechnen zu müssen:

7.3.1 Geometrische Darstellung der Abbildung Φ:

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 eine Transformation Φ an, um die Daten in $\mathbb{R}^3$ zu transformieren.

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

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

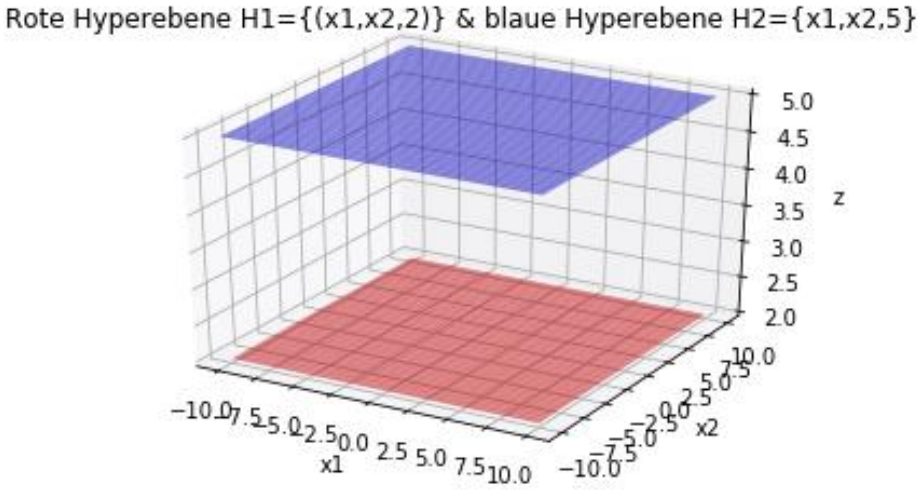


Figure 17: Hyperebenen zum Kernel-Trick Beispiel

Ein bewegtes Bild davon liegt in der Referenz [**HVö-GitML20**]:
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 $H := \{(x_1, x_2, z) | \cdot z = 3.5\}$ trennen. Das hier gesehene Vorgehen der Trennbarkeit in einem höher-dimensionalen Raum (als Zielbild der Abbildung Φ ist eine bekannte Tatsache der Mathematik. Wir nutzen dies um den "Kernel-Trick" (siehe Kapitel: "Mathematische Grundlagen von SVM") dort zu formulieren.
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 den Radius $r = \sqrt[2]{3,5} \approx 1.871$. Damit ergibt sich die visuelle Darstellung der Klassifikation-Trennlinie als Kreislinie. Wir lassen uns diesen Kreis via einem kleinem Python Programm zeichnen (siehe unten).
Als Übungsbeispiel (7.1) konstruieren wir uns jetzt ein Python-Programm der diese Daten visualisiert.
Dieses kleine Beispiel bildet die anschauliche Basis für die spätere Definition des "Kernel-Tricks" der SVM:

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

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

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

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

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

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

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

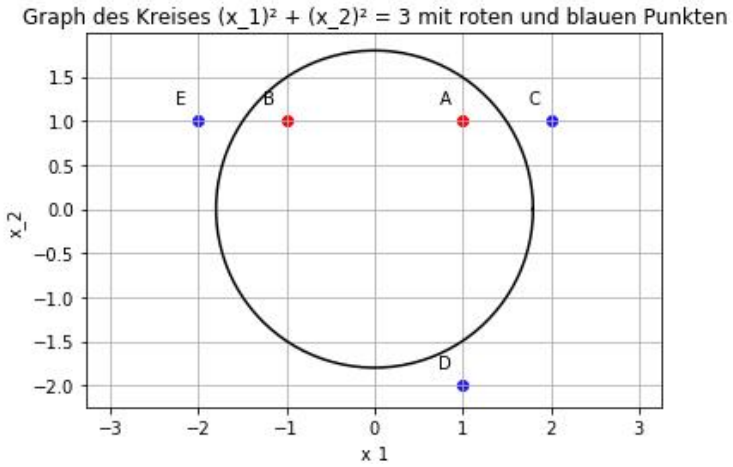


Figure 18: Python Code und Kernel-Trick Kreisbild

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 auch 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. Zur Programmierung in Python importieren wir zuerst die erforderlichen Bibliotheken. Ein mögliches Ergebnis könnte wie folgt aussehen:

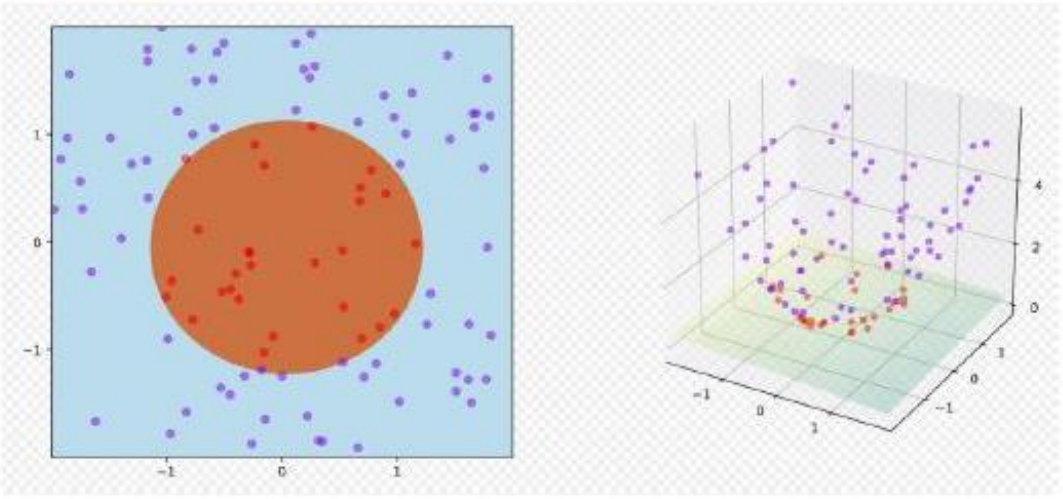


Figure 19: SVM Kernel-Trick(2D-3D) Beispiel

7.4 Mathematische Grundlagen der Support Vector Machines (SVM)

In diesem Kapitel werden die mathematischen Grundlagen gelegt. Die Bedeutung der mathematischen Formeln und Definitionen werden aber erst durch die Anwendungen im nächsten Kapitel klarer.

1. Einführung:
Support Vector Machines (SVMs) sind ein überwacht lernendes Modell, das vor allem für Klassifizierungsprobleme verwendet wird, aber auch für Regression anwendbar ist. Das Ziel ist, eine Hyperebene zu finden, die zwei Klassen in einem hochdimensionalen Raum trennt. Die mathematische Grundlage von SVM basiert auf Konzepten aus der linearen Algebra, Optimierungstheorie und Statistik.

2. Lineare Klassifikatoren:
Gegeben eine Trainingsmenge bestehend aus Paaren (x_i, y_i), wobei

$$x_i \in \mathbb{R}^n \quad \text{und} \quad y_i \in \{-1, 1\}$$

ist, besteht das Ziel darin, eine Hyperebene zu finden, die die Datenpunkte trennt. Eine Hyperebene in einem n-dimensionalen Raum wird durch

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

definiert, wobei w der Normalenvektor der Hyperebene ist und b der Bias.

3. Maximierung des Margins:
SVM strebt danach, den Abstand (Margin) zwischen den zwei Klassen zu maximieren. Der Margin wird als der Abstand zwischen der Hyperebene und den nächsten Punkten der beiden Klassen (Support-Vektoren) definiert. Für korrekt klassifizierte Punkte gilt:

$$y_i(w \cdot x_i + b) \geq 1$$

Das Ziel ist, die Norm des Vektors w zu minimieren, was zu folgendem Optimierungsproblem führt:

$$\min_w \frac{1}{2}\|w\|^2$$

unter der Nebenbedingung:

$$y_i(w \cdot x_i + b) \geq 1 \quad \forall i$$

4. Der Lagrange-Ansatz[8] und das Dualproblem:
Um das Problem zu lösen, werden Lagrange-Multiplikatoren α_i eingeführt. Die Lagrange-Funktion lautet:

$$L(w, b, \alpha) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{N} \alpha_i [y_i(w \cdot x_i + b) - 1]$$

Durch Optimierung der Lagrange-Funktion wird das Dualproblem[9]

$$\max_{\alpha} \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j (x_i \cdot x_j)$$

unter der folgenden Nebenbedingungen gelöst:

$$\sum_{i=1}^{N} \alpha_i y_i = 0, \quad \alpha_i \geq 0$$

Die Lösung dieses Dualproblems liefert die Lagrange-Multiplikatoren α_i, und der optimale Gewichtsvektor w kann als Linearkombination der Support-Vektoren ausgedrückt werden.

5. Nicht-lineare Klassifikation: Der Kernel-Trick:
Für nicht linear trennbare Daten erweitert man das SVM durch den sogenannten "Kernel-Trick". Hierbei werden die Eingabedaten in einen höherdimensionalen Raum abgebildet, in dem eine lineare Trennung möglich ist. Der Kernel $K(x_i, x_j)$ berechnet das Skalarprodukt in diesem Raum:

$$K(x_i, x_j) = \phi(x_i) \cdot \phi(x_j)$$

Gängige Kernel-Funktionen sind:

- **Linear Kernel:** $K(x_i, x_j) = x_i \cdot x_j$

[8] Joseph-Louis Lagrange (1736–1813) war ein bedeutender italienisch-französischer Mathematiker und Physiker, der entscheidende Beiträge zu zahlreichen Gebieten der Mathematik und Physik leistete.

[9] Das duale Problem ist eine Methode, um das ursprüngliche (primal) Optimierungsproblem in ein anderes Problem umzuwandeln, das oft einfacher zu lösen ist. Durch die Einführung von Lagrange-Multiplikatoren kann man sowohl die Zielfunktion als auch die Nebenbedingungen in eine kombinierte Funktion einbeziehen, die minimiert bzw. maximiert wird.

- **Polynom-Kernel (Grad=d):** $K(x_i, x_j) = (x_i \cdot x_j + 1)^d$
- **RBF (Gaussian) Kernel:** $K(x_i, x_j) = \exp(-\gamma \|x_i - x_j\|^2)$

Anmerkung: RBF = Radial-Basis-Funktion.

6. Soft Margin für nicht trennbare Daten:
Falls die Daten nicht perfekt trennbar sind, wird das Konzept des "Soft Margin" eingeführt. Hierbei werden sogenannte "Slack-Variablen" ξ_i für jedes Datenpunkt hinzugefügt, sodass auch Fehlklassifizierungen erlaubt sind:

$$\min_w \frac{1}{2}\|w\|^2 + C \sum_{i=1}^{N} \xi_i$$

unter der Nebenbedingung:

$$y_i(w \cdot x_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0$$

Der Parameter C reguliert dabei den Kompromiss zwischen der Maximierung des Margins und der Minimierung der Fehlklassifikationen.

7. Support Vector Regression (SVR):
SVM kann auch für Regressionsprobleme verwendet werden, was als "Support Vector Regression (SVR)" bekannt ist. Dabei wird versucht, eine Funktion zu finden, die innerhalb eines Toleranzbereichs ϵ von den Zielwerten abweicht. Das Optimierungsproblem lautet:

$$\min_w \frac{1}{2}\|w\|^2 + C \sum_{i=1}^{N} (\xi_i + \xi_i^*)$$

unter den Nebenbedingungen:

$$y_i - (w \cdot x_i + b) \leq \epsilon + \xi_i$$

$$(w \cdot x_i + b) - y_i \leq \epsilon + \xi_i^*, \quad \xi_i, \xi_i^* \geq 0$$

Hauptmerkmale:

- **Fehlertoleranz (ϵ):** Punkte innerhalb des Toleranzbereichs (ϵ-Tube) tragen nicht zur Optimierung der Funktion bei.

- **Maximale Marginalisierung:** SVR maximiert die Marginalisierung ähnlich wie bei SVM, jedoch für eine Regressionsebene oder -funktion.
- **Regularisierung (C):** Ein Hyperparameter C steuert den Kompromiss zwischen Modellgenauigkeit und Vermeidung von Überanpassung.
- **Kerntrick:** Durch die Verwendung von Kernfunktionen (z.B. polynomiale oder RBF-Kerne) kann SVR auch nichtlineare Beziehungen modellieren.

Vorteile:

- Gut geeignet für kleine, komplexe Datensätze.
- Flexibel durch verschiedene Kernfunktionen.
- Robust gegenüber Ausreißern, da diese oft außerhalb des ϵ-Tubes liegen.

Nachteile:

- Sorgfältige Abstimmung der Hyperparameter (ϵ, C, Kernparameter) erforderlich.
- Rechenintensiv bei großen Datensätzen.

Insgesamt eignet sich SVR besonders gut für präzise Vorhersagen in Szenarien mit nichtlinearen Beziehungen oder verrauschten Daten.

Hinweis: In den Übungsaufgaben (letztes des Kapitel) findet man einfache Beispiele für SVR mit den entsprechenden Jupyter Notenbooks (JB)

8. Fazit:
SVM ist ein robustes Klassifikations- und Regressionsverfahren mit soliden mathematischen Grundlagen. Es basiert auf der Maximierung des Margins, der Verwendung von Lagrange-Multiplikatoren und dem Kernel-Trick, um nicht-lineare und nicht-trennbare Daten zu behandeln.

7.5 Konkrete Beispiele für SVM

Ziel dieses Abschnitte ist es die mathematischen Formeln und Definitionen aus obigen Kapitel besser zu verstehen. Insbesondere für die 3 gängigen Kernel-Funktionen.

7.5.1 Lineare Trennbarkeit - Beispiel Irisblumen

Beispiel: Klassifikation von zwei Iris-Blumenarten mit SVM.
In diesem Beispiel betrachten wir einen Datensatz der linear trennbar ist. Wir verwenden dazu den bekannten Iris-Datensatz, der Informationen über drei Arten von Iris-Blumen enthält: Iris Setosa, Iris Versicolor und Iris Virginica. Jede Blume wird durch vier Merkmale beschrieben:
Sepallänge (in cm)
Sepalbreite (in cm)
Petallänge (in cm)
Petalbreite (in cm)

Zur Vereinfachung beschränken wir uns auf zwei Arten von Blumen (Iris Setosa und Iris Versicolor) und zwei Merkmale (Petallänge und Petalbreite).
Für diese zwei Ausprägungen von Iris-Blumen erkennen wir eine Lineare Trennbarkeit. Wir nutzen die Lineare Kernel-Funktion dazu um die "Maximierung des Margins" (Punkt 3) zu berechnen.
Wir erhalten eine Trennlinie: "Blütenbreite = -1.5 *Blütenlänge + 4" und bekommen drei Support-Vektoren (1.9, 0.2), (1.7, 0.5) und (3, 1.1).

Siehe dazu auch das folgende Python-Programm (Jupyter Notebook):
https://github.com/HVoellinger/Mathematische-Grundlagen_von_ML/blob/main/Quellen_Math-ML/SVM_Irisblume.pdf

7.5.2 Beispiele für drei gängige Kernel-Funktionen

In diesem Kapitel betrachten wir den nicht-linearen Fall und untersuchen die 3 gängigen Kernel-Funktionen $K(x_i, x_j)$.
In den Python-Programmen betrachten wir wieder den bekannten Fall das wir 5 Datenpunkte in zwei unterschiedlichen Klassen haben: Klasse +1: A(1,1), B(-1,1) und Klasse -1: C(2,1), D(1,-2) und E(-2,1) haben.

0. Allgemeine Erläuterungen der mathematischen Definitionen:

Im Kapitel über die Mathematik, sind folgende Definitionen wichtig:

- **Kernel Matrix:** Der Ausdruck $K(A, B)$ gibt an, wie "ähnlich" sich zwei Datenpunkte A und B im transformierten Raum sind. Die symmetrische (5x5)-Kernel-Matrix berechnet alle 25 Werte K(A,A) bis K(E,E).

- **Lagrange-Funktion:** Die Lagrange-Funktion in einem Support Vector Machine (SVM)-Problem kombiniert die Optimierungsziele der Maximierung des

Abstands (Margins) zwischen den Klassen und der Minimierung von Fehlern. Diese Funktion wird unter Berücksichtigung der Nebenbedingungen (Constraints) optimiert, um die optimale Trennlinie zu finden.

- **Wert der Lagrange Funktion:** Die Werte der Lagrange-Multiplikatoren α_i werden optimiert, um die maximale Margin-Trennlinie zu finden. Diese Optimierung erfolgt typischerweise mit einem quadratischen Programmierungsverfahren (QP). Die Lösung des dualen Problems liefert die optimalen α_i und jene x_i für die $\alpha_i > 0$ ist, sind die Support Vektoren.

Siehe dazu die folgenden Python-Programme (Jupyter Notebook):

- Linearer Kernel:
https://github.com/HVoellinger/Mathematische-Grundlagen_von_ML/blob/main/Quellen_Math-ML/SVM_LagrangeFkt-linear-5DP.pdf
- Polynom-Kernel (Grad = 2):
https://github.com/HVoellinger/Mathematische-Grundlagen_von_ML/blob/main/Quellen_Math-ML/SVM_LagrangeFkt-Polynom(2)-5DP.pdf
- Polynom-Kernel (Grad = 3):
https://github.com/HVoellinger/Mathematische-Grundlagen_von_ML/blob/main/Quellen_Math-ML/SVM_LagrangeFkt-Polynom(3)-5DP.pdf

1. Lineare Kernel-Fkt:
Ist eine lineare Trennbarkeit nicht mehr erreichbar, nutzen wir den Kernel-Trick (siehe Kapitel über Mathematische Grundlagen des SVM). Hier nutzen wir die Lineare Kernel-Fkt: $K(x_i, x_j) = x_i \cdot x_j$
Wie man leicht sieht, ist die Trennlinie in diesem Fall eine Gerade im 2D-Raum. Dies gilt allgemein für den linearen Kernel-Trick, da hier keine nichtlineare Transformation der Daten in einen höherdimensionalen Raum erfolgt, bleibt die Trennlinie immer eine gerade Linie in 2D (oder eine Hyperebene in höheren Dimensionen). Um gekrümmte Trennlinien zu erhalten, muss man einen nichtlinearen Kernel verwenden, wie z. B. den polynomialen Kernel (siehe nächste Beispiele) oder den RBF-Kernel (Radial Basis Function), die die Datenpunkte in einen höherdimensionalen Raum projizieren, wo dann eine lineare Trennung möglich wird, die sich im ursprünglichen Raum als gekrümmte Trennlinie manifestiert.

Siehe dazu das folgende Python-Programm (Jupyter Notebook):

https://github.com/HVoellinger/Mathematische-Grundlagen_von_ML/blob/main/Quellen_Math-ML/SVM_Trennlinie-LK.pdf

2. Polynom-Kernel vom Grad d=2:
Eine bessere lineare Trennbarkeit ist erreichbar, wenn wir Kernel-Trick mit einem Polynom vom Grad d= 2: $K(x_i, x_j) = (x_i \cdot x_j + 1)^d$ nutzen:

Der Kernel-Trick transformiert $(x_1, x_2) \in \mathbb{R}^2$ zu Daten $(z_1, z_2, z_3, z_4, z_5, z_6) \in \mathbb{R}^6$:

Die Transformation $\Phi : \mathbb{R}^2 \to \mathbb{R}^6$ ist definiert durch:

$$\Phi(x_1, x_2) = (x_1^2, x_2^2, \sqrt[2]{2} \cdot x_1 \cdot x_2, \sqrt[2]{2} \cdot x_1, \sqrt[2]{2} \cdot x_2, 1).$$

Wenn wir die Hyperebene in den ursprünglichen 2D-Raum zurückprojizieren, erhalten wir eine nichtlineare Trennkurve.
Durch das Einsetzen der neuen Merkmale in die ursprünglichen Variablen x_1, x_2 (durch Rücksubstitution von $z_1, z_2, ..., z_6$) ergibt sich eine quadratische Gleichung in x_1, x_2 die eine Kurve beschreibt, wie eine Ellipse, Parabel oder Hyperbel:

$$w_1 \cdot x_1^2 + w_2 \cdot x_2^2 + w_3 \cdot \sqrt[2]{2} \cdot x_1 \cdot x_2 + w_4 \cdot \sqrt[2]{2} \cdot x_1 + w_5 \cdot \sqrt[2]{2} \cdot x_2 + w_6.$$

Diese quadratische Gleichung definiert die nichtlineare Trennkurve im ursprünglichen 2D-Raum.

Fazit:

- Der polynomiale Kernel der Ordnung 2 transformiert die Datenpunkte aus dem 2D-Raum in einen 6-dimensionalen Raum mit den neuen Koordinaten $(z_1, ...z_6)$
- Im 6D-Raum wird eine lineare Hyperebene gefunden, die die Daten trennt.
- Bei der Rücktransformation in den ursprünglichen 2D-Raum entspricht diese lineare Hyperebene einer nichtlinearen Trennkurve, die durch obige quadratische Gleichung beschrieben wird.
- Die Trennlinie ist leider noch nicht optimal wie man im Plot des Python-Programmes sieht.

Siehe dazu das folgende Python-Programm (Jupyter Notebook):

https://github.com/HVoellinger/Mathematische-Grundlagen_von_ML/blob/main/Quellen_Math-ML/SVM-PolyKernel(2)-5DP.pdf

3. Polynom-Kernel vom Grad d=3:
Eine bessere lineare Trennbarkeit ist erreichbar, wenn wir ein Polynom vom Grad d= 3: $K(x_i, x_j) = (x_i \cdot x_j + 1)^d$ nutzen:

Siehe dazu das folgende Python-Programm (Jupyter Notebook):
Der Kernel-Trick transformiert $(x_1, x_2) \in \mathbb{R}^2$ zu Daten $(z_1, ...z_{10}) \in \mathbb{R}^{10}$:

Die Transformation $\Phi : \mathbb{R}^2 \to \mathbb{R}^{10}$ ist definiert durch:

$$\Phi(x_1, x_2) = (x_1^3, x_2^3, x_1^2 \cdot x_2, x_1 \cdot x_1^2, x_1^2, x_2^2, x_1 \cdot x_2, x_1, x_2, 1).$$

Siehe dazu das folgende Python-Programm (Jupyter Notebook):

https://github.com/HVoellinger/Mathematische-Grundlagen_von_ML/blob/main/Quellen_Math-ML/SVM-PolyKernel(3)-5DP.pdf

7.5.3 RBF-Beispiel für Nutzung des Kernel-Tricks

Ist eine Lineare Trennbarkeit nicht mehr erreichbar, nutzen wir den Kernel-Trick (siehe Kapitel über Mathematische Grundlagen des SVM).
Siehe dazu das folgende Python-Programm (Jupyter Notebook):

https://github.com/HVoellinger/Mathematische-Grundlagen_von_ML/blob/main/Quellen_Math-ML/SVM_RBF_Kernel.pdf

7.5.4 Einfaches Beispiel für Support Vector Regression (SVR)

Hier ist ein einfaches Beispiel, das zeigt, wie Support Vector Regression (SVR) funktioniert. Wir verwenden Python und die Bibliothek scikit-learn, um eine nichtlineare Funktion zu modellieren.
Beispiel: Vorhersage einer quadratischen Funktion.
Wir generieren Daten, die einer quadratischen Funktion $y = x^2$ folgen, fügen etwas Rauschen hinzu und verwenden SVR, um die Funktion zu approximieren.
Das Rauschen in dem oben angegebenen Python-Programm wird mit der Funktion np.random.randn() erzeugt
np.random.randn(50): erzeugt 50 Stichproben aus einer Standardnormalverteilung, d.h. einer Gauß-Verteilung mit Mittelwert $\mu = 0$ und Standardabweichung $\sigma = 1$.
Das erzeugte Rauschen wird dann skaliert (mit * 2), um die Amplitude des Rauschens zu erhöhen, und zu den ursprünglichen Werten der quadratischen Funktion $y = x^2$ addiert: *np.random.randn(50) * 2*
Das erzeugte Rauschen kann sowohl positive als auch negative Werte haben, was den Daten natürliche Schwankungen hinzufügt.
Hier ist dieses Beispiel als Jupyter Notebook (JB) umgesetzt:
https://github.com/HVoellinger/Mathematische-Grundlagen_von_ML/blob/main/Quellen_Math-ML/SVR_SimExample-JupyterNotebook.pdf

7.5.5 Vergleich von drei 3 unterschiedliche SVR Verfahren

Für einen einfachen Vergleich betrachte das folgende Jupyter Notenbook (JB), dass eine Regression mit den drei 3 SVM-Verfahren (linear, RBF und Polynom(Grad=3)) durchführt:
https://github.com/HVoellinger/Mathematische-Grundlagen_von_ML/blob/main/Quellen_Math-ML/SVR_Example-JupyterNotebook.pdf

Eine genaue Analyse der Ergebnisse soll als Übungsaufgabe 7.5 durchgeführt werden.

7.6 Übungen zum Kapitel 7

7.6.1 Übung 7.1 - Python Code zu Hyperebenen (7.3)

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

7.6.2 Übung 7.2 - Python Code zum geometrischen Beispiel (7.3)

Erstellen Sie den Python Code, der die Trennlinie der Klassen für die 5 Datenpunkte visualisiert. Benutzen Sie dazu die klassische Polynom-Kernel-Funktion vom Grad=2 von SVM. Visualisieren Sie die Punkte und Trennlinie in $\mathbb{R}^2$ (2D).

7.6.3 Übung 7.3 - Python-Pgm. zum Polynom-Kernel (Grad=d) mit 10 Datenpunkten

Erstellen Sie analog zum Kapitel (7.4) jetzt ein ein einfaches und anschauliches Beispiel für eine Support Vector Machine (SVM) mit 10 Datenpunkten. Benutzen Sie dazu eine Polynom-Kernel Funktion von Grad=d.
Sie nehmen wieder 2 Klassen +1 und -1 mit jetzt jeweils 5 Daten-Punkten (DP):
*** Definierte Datenpunkte und Labels: y = +1: A bis E und y = -1: F bis J *****
X = np.array([[1, 1],[-1, 1],[3, 2],[-2, -2],[3, 1],[2, 1],[1, -2],[-2, 1],[-1, -4],[1, 3]]) *** DP ***
y = np.array([1, 1, 1, 1, 1, -1, -1, -1, -1, -1]) *** Labels ***
Gegen Sie die SVM-Trennlinie mit einer Polynom (Grad=d) an. Visualisieren Sie die Stützvektoren und Trennlinie in 2D. Für welchen Grad=d sind die Trennlinien optimal?

7.6.4 Übung 7.4 - Vergleich Linearer Kernel mit Polynom-Kernel(Grad=d) und 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 Polynom-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.

7.6.5 Übung 7.5 - SVR-Verfahren: Vergleich dreier Ansätze im Jupyter Notebook

In Kapitel "Mathematische Grundlagen von SVM" (Kap. 7.4) wurde auch die SVM Anwendung als mögliche Lösung für Regression-Aufgaben (SVR) beschrieben (siehe Punkt 7.). Im verlinkten entsprechenden Jupyter Notebook (JB) wurde ein Beispiel mit den SVM-Verfahren Linear, RBF und Polym(Grad=3) berechnet (siehe Plots am Ende des JB). Vergleiche die Ergebnisse und gibt eine fachliche Begründung.

7.6.6 Übung 7.6 - Vergleichbarkeit der Ergebnisse mit Kapitel 5 (Simple Lineare Regression)

Im Kapitel 5 wird in der Übungsaufgabe (5.6)-Teil2 eine Simple Lineare Regressionsgerade für 10 Datenpunkte berechnet. Eine simple oder einfache Regression ist dadurch definiert, dass der Zielparameter nur aus einem Feature bestimmt wird. Damit liegt die Regressionskurve (bei Aufgabe (5.6)-Teil2 ist dies eine Gerade) in einer Ebene. Bestimme jetzt mit dem SVR-Verfahren für die gleichen Datenpunkte ebenfalls eine entsprechende Regressions-Kurve (oder -Gerade) und zeige, dass die Ergebnisse vergleichbar sind.

7.6.7 Übung 7.7 - Vergleichbarkeit der Ergebnisse mit Kapitel 5 (Multiple Lineare Regression)

In Aufgabe (5.7) werden die Datenpunkte für eine Multiple Lineare Regression angegeben. Bestimme die Regressionsebene mit dem SVR-Verfahren und vergleiche die Ergebnisse.

8 Neuronale Netzwerke, insbesondere Tiefe Neuronale Netzwerke

Neuronale Netzwerke sind eine spezielle Art von **künstlichen neuronalen Netzwerken (KNN)** [10]. Ein **Deep Neural Network (DNN)** ist ein allgemeiner Begriff, der für tiefe neuronale Netzwerke steht. Ein DNN besteht neben dem Eingabe- und Ausgabe-Schicht noch aus sehr vielen weiteren "versteckten" Schichten ("Hidden Layers"). Jede Schicht (Layer) enthält eine bestimmte Anzahl von Neuronen oder Aktivierungseinheiten. DNNs 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.
Eine spezielle Art von DNN ist ein **Convolutional Neural Network (CNN)** (in Deutsch "Faltungsnetz"), das insbesondere für Aufgaben wie die Bildverarbeitung entwickelt wurde. CNNs nutzen eine spezielle Schicht, die sogenannte Faltungsschicht (Convolutional Layer), um Muster in den Eingabedaten zu erkennen. Die Faltungsschichten extrahieren Merkmale aus Daten, indem sie kleine Filter verwenden, die über die Eingabedaten gleiten (falten), um Merkmale wie Kanten oder Texturen zu identifizieren. Sie sind besonders effektiv bei der Extraktion von Merkmalen aus solchen Daten, was sie ideal für Aufgaben wie Bilderkennung und Bildklassifikation macht.

8.0.1 Kann ein KNN von der Struktur eines menschlichen Gehirns lernen?

1. **Neuronen und Synapsen vs. Parameter**:

   - Das menschliche Gehirn hat etwa **86 Milliarden Neuronen** und bis zu **1 Billiarde Synapsen**. Diese Synapsen sind Verbindungen, durch die Neuronen Signale senden und empfangen. Diese enorme Zahl an Verbindungen ist verantwortlich für die Fähigkeit des Gehirns, Informationen zu verarbeiten, Erinnerungen zu speichern, zu lernen und komplexe kognitive Aufgaben auszuführen.

   - Ein großes KI-Modell wie GPT-3 hat **175 Milliarden Parameter**, was in etwa der Größenordnung von Neuronen im Gehirn entspricht. Diese Parameter

[10] Neuronale Netze sind dynamische Systeme, die aus einfach aufgebauten Einheiten, den Neuronen, bestehen, die über ein System von gerichteten Verbindungen miteinander wechselwirken. Sie wurden ursprünglich zur mehr oder weniger vereinfachten und idealisierten Modellierung biologischer Neuronenverbände entwickelt. Man spricht deshalb zur Unterscheidung von den natürlichen Systemen auch von künstlichen neuronalen Netzen, (Englisch: "Artificial Neural Networks")

stellen jedoch Gewichte in einem künstlichen neuronalen Netz dar und keine Verbindungen im biologischen Sinne.

2. **Flexibilität und Lernen**:
   - Das menschliche Gehirn ist in der Lage, **dynamisch zu generalisieren**, zu lernen und sich an neue Informationen und Umgebungen anzupassen. Lernen im Gehirn basiert auf plastischen Veränderungen der Synapsen.
   - KI-Modelle wie GPT-3 **lernen nur während des Trainingsprozesses**. Nach dem Training sind die Parameter festgelegt, und das Modell passt sich nicht an neue Daten an, ohne erneut trainiert zu werden.
3. **Energieeffizienz**:
   - Das menschliche Gehirn ist extrem energieeffizient und verbraucht etwa **20 Watt**. Es kann mit einer geringen Energiemenge komplexe Aufgaben ausführen.
   - GPT-3 benötigt erhebliche Rechenressourcen und viel Energie, besonders während des Trainings.
4. **Generalisation**:
   - Das menschliche Gehirn hat eine natürliche Fähigkeit zur **Generalisation** und kann aus wenigen Beispielen lernen und dieses Wissen auf neue Situationen anwenden.
   - KI-Modelle generalisieren oft basierend auf großen Mengen von Trainingsdaten und können bei neuen oder seltenen Szenarien Schwierigkeiten haben.

Zusammengefasst übersteigen die Parameter von GPT-3 zwar die Anzahl der Neuronen im menschlichen Gehirn, aber die **Komplexität, Flexibilität und Lernfähigkeit** des menschlichen Gehirns sind weitaus fortschrittlicher und effizienter.

8.1 Anwendung von DNN - "ChatGPT"

ChatGPT ist eine der bekanntesten Anwendungen von DNN's mit erstaunlichen Ergebnissen.
ChatGPT basiert auf einem tiefen neuronalen Netzwerk. Genauer gesagt, es ist ein Beispiel für ein sogenanntes **"Transformer-Modell"**, das eine spezielle Art von tiefem neuronalen Netzwerk darstellt. Transformer-Modelle sind dafür bekannt, sehr große und komplexe Datenmengen zu verarbeiten, was sie ideal für Aufgaben wie die

Verarbeitung und Generierung natürlicher Sprache macht.
Im Kern besteht das Modell aus mehreren Schichten von Neuronen, die in einer hierarchischen Struktur angeordnet sind. Diese Schichten sind in der Lage, komplexe Muster und Beziehungen in Daten zu erkennen und zu verarbeiten. In jedem Schritt der Verarbeitung durchläuft die Eingabe durch viele Schichten des Netzwerks, wobei jede Schicht die Daten auf eine andere Weise transformiert, bis am Ende eine Antwort generiert wird.

Die Entwicklung von ChatGPT besteht im Wesentlichen aus zwei Schritten:

1. Pretraining: Pretraining ist einer der wichtigsten Schritte im Training von ChatGPT. In diesem Schritt wird das Modell mit einer riesigen Menge von Textdaten gefüttert, damit es allgemeine Sprachmuster und -strukturen lernt.
2. Prompting: Prompting hingegen ist die Art und Weise, wie ein Benutzer nach dem Training mit dem Modell interagiert. Es handelt sich um den Eingabeprozess, bei dem der Benutzer dem Modell eine Eingabe gibt, auf die es basierend auf seinem vorherigen Training antwortet.
Anmerkung: Prompting ist in gewisser Weise subjektiv, da es stark von der Formulierung und Absicht des menschlichen Benutzers abhängt. Die subjektive Meinung und das Feedback von menschlichen Trainern während des Trainingsprozesses spielen ebenfalls eine Rolle bei der Gestaltung des Verhaltens des Modells.

Zusammenfassung: In diesem Sinne kann man sagen, dass ChatGPT auf einem sehr großen und tiefen neuronalen Netzwerk basiert, das speziell darauf trainiert wurde, menschliche Sprache zu verstehen und darauf zu reagieren.

8.1.1 Die Transformer-Architektur

Die Transformer-Architektur wurde erstmals in dem 2017 veröffentlichten Paper *"Attention is All You Need"* von Vaswani et al. vorgestellt. Sie hat die Verarbeitung natürlicher Sprache revolutioniert und weitgehend frühere Ansätze wie rekurrente neuronale Netze (RNNs) und Long Short-Term Memory-Netzwerke (LSTMs) ersetzt. Hier sind die Hauptkomponenten der Transformer-Architektur:

1. Encoder-Decoder-Struktur
Der Transformer besteht aus zwei Hauptteilen: einem **Encoder** und einem **Decoder**.

- **Encoder**: Der Encoder besteht aus mehreren identischen Schichten, die die Eingabesequenz verarbeiten. Jede Schicht hat zwei Hauptkomponenten: eine Selbstaufmerksamkeits-Schicht (*Self-Attention Layer*) und eine Feedforward-Schicht (*Fully Connected Layer*).

- **Decoder**: Der Decoder ist ähnlich aufgebaut wie der Encoder, aber mit einer zusätzlichen Schicht für die Aufmerksamkeit auf die Ausgabe des Encoders (*Encoder-Decoder Attention*). Der Decoder erzeugt die Ausgabe Sequenz für Sequenz.

2. Self-Attention Mechanismus
Der **Self-Attention Mechanismus** (oder *Scaled Dot-Product Attention*) ermöglicht es dem Modell, zu bestimmen, welche Teile der Eingabesequenz (oder der vorherigen Ausgaben) für die aktuelle Position am wichtigsten sind. Bei Self-Attention wird für jedes Wort in der Eingabesequenz berechnet, wie stark es auf alle anderen Wörter in der Sequenz achten sollte. Das Resultat ist eine gewichtete Summe, die diese Abhängigkeiten berücksichtigt.

3. Multi-Head Attention
Anstelle einer einzigen Self-Attention-Berechnung verwendet der Transformer **Multi-Head Attention**. Das bedeutet, dass mehrere Self-Attention-Berechnungen parallel durchgeführt werden, die dann zusammengeführt werden. Dies erlaubt es dem Modell, unterschiedliche Beziehungen und Aspekte der Sequenz gleichzeitig zu erfassen.

4. Position-Encoding
Da Transformer-Modelle keine rekurrente Struktur haben, wird die Reihenfolge der Eingabesequenz nicht explizit erfasst. Stattdessen verwendet das Modell **Position-Encoding**, um Informationen über die Reihenfolge der Sequenzpositionen hinzuzufügen.

5. Feedforward Neural Network
Nach der Multi-Head Attention folgt eine **Feedforward-Schicht**, die unabhängig auf jede Position in der Sequenz angewendet wird. Diese Schicht besteht aus zwei vollständig verbundenen Netzwerken mit einer nichtlinearen Aktivierungsfunktion dazwischen.

6. Residual Connections und Layer Normalization
Jede Subschicht im Encoder und Decoder ist von einer **Residual Connection** umgeben, die das Ergebnis der Subschicht mit ihrem Input kombiniert. Danach wird **Layer Normalization** angewendet, was die Konvergenz während des Trainings unterstützt.

7. Softmax Layer
Am Ende des Decoders steht eine **Softmax-Schicht**, die die Wahrscheinlichkeit für jedes Wort im Vokabular berechnet und das wahrscheinlichste Wort auswählt, um die Ausgabe zu erzeugen.

8.1.2 Vorteile der Transformer-Architektur

- **Parallelisierung**: Im Gegensatz zu RNNs, die sequenziell arbeiten, erlaubt der Transformer durch Self-Attention parallele Berechnungen, was das Training beschleunigt.
- **Langfristige Abhängigkeiten**: Self-Attention ermöglicht es dem Modell, langfristige Abhängigkeiten innerhalb der Daten besser zu erfassen.
- **Skalierbarkeit**: Transformer-Modelle können sehr groß skaliert werden, was ihre Leistung bei großen Datenmengen verbessert.

8.1.3 Anwendungen

Transformer-Modelle wie GPT (*Generative Pre-trained Transformer*), BERT (*Bidirectional Encoder Representations from Transformers*) und T5 (*Text-To-Text Transfer Transformer*) haben bahnbrechende Ergebnisse in vielen NLP-Aufgaben erzielt, darunter maschinelle Übersetzung, Textklassifikation, Fragebeantwortung und Textgenerierung.

**** Referenz: [**Math for DL**] ********

8.2 Mathematische Grundlagen von Faltungsnetzwerken

Die mathematischen Grundlagen des Faltungsnetzwerke, Englisch auch oft Convolutional Neural Networks (CNN) genannt, 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.

8.2.1 Liste der wichtigsten mathematische Methoden

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) [11] 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.

[11] Eine ReLU (Rectified Linear Unit) ist eine Aktivierungsfunktion, die häufig in neuronalen Netzen verwendet wird, insbesondere in Deep Learning-Modellen. Die ReLU-Funktion gibt den Eingabewert zurück, wenn dieser positiv ist, und 0, wenn der Eingabewert negativ ist.

8.2.2 Mathematische Definition eines tiefen Neuronalen Netzes (DNN)

Ein tiefes neuronales Netzwerk besteht aus L Schichten, wobei L normalerweise größer oder gleich 3 ist. Jede Schicht l, wobei $l = 1, 2, \ldots, L$, enthält eine bestimmte Anzahl von Neuronen oder Aktivierungseinheiten.
Jedes Neuron in einer Schicht l erhält Eingaben von den Neuronen in der vorherigen Schicht $(l-1)$ und berechnet eine gewichtete Summe dieser Eingaben. Diese gewichtete Summe wird normalerweise mit einer Aktivierungsfunktion ϕ aktiviert, um die Ausgabe des Neurons zu erzeugen.

Ein künstliches Neuron mit Gewichten $w_1, \ldots, w_n \in \mathbb{R}$, Bias $b \in \mathbb{R}$ und Aktivierungsfunktion $\rho : \mathbb{R} \to \mathbb{R}$ kann man also wie folgt definierten:

$$f(x_1, x_2, \ldots, x_n) = \rho\left(\sum_{i=1}^{n} w_i x_i + b\right)$$

In dieser Definition sind $x_1, x_2, \ldots, x_n$ die Eingabewerte, $w_1, w_2, \ldots, w_n$ die entsprechenden Gewichte und b ist der Bias-Term.
Die gewichtete Summe der Eingaben $(\sum_{i=1}^{n} w_i x_i + b)$ wird durch die Aktivierungsfunktion ρ geleitet, um die Ausgabe des künstlichen Neurons $f(x_1, x_2, \ldots, x_n)$ zu erzeugen.
Die Aktivierungsfunktion führt eine Nichtlinearität in die Reaktion des Neurons ein und bestimmt das Verhalten des Neurons. Häufig verwendete Aktivierungsfunktionen sind unter anderem die Sigmoid-, ReLU (Rectified Linear Unit) - und Tangens hyperbolicus (tanh)-Funktionen.
Die Gewichtungen und Schwellenwerte der Neuronen in einem tiefen neuronalen Netzwerk werden normalerweise in einem Trainingsprozess erlernt, der auf Daten basiert. Dies geschieht oft mithilfe von Verfahren wie dem Backpropagation-Algorithmus und Optimierungsalgorithmen wie dem Gradientenabstiegsverfahren .

Mathematische Definition eines DNN's:
Die mathematische Darstellung eines tiefen neuronalen Netzwerks kann komplex sein und hängt von den genauen Architekturen und Hyperparametern ab, die für ein bestimmtes Problem festgelegt wurden. In der Regel wird die gesamte Transformation eines Netzwerks von der Eingabe zur Ausgabe als eine Funktion $f(x; \phi)$ dargestellt, wobei x die Eingabe ist und ϕ die Parameter des Netzwerks sind.

Die Verkettung künstlicher Neuronen führt zu Kompositionen von affinen linearen Abbildungen und Aktivierungsfunktionen.
In den folgenden zwei Bildern sehen wir ein Beispiel für einer solche Komposition und daraus folgend die mathematische Definition eines (Deep) Neuronalen Netz (DNN):

Example: The following part of a neural network is given by

$$\Phi : \mathbb{R}^3 \to \mathbb{R}^2, \quad \Phi(x) = W^{(2)}\rho(W^{(1)}x + b^{(1)}) + b^{(2)}.$$

$$W^{(1)} = \begin{pmatrix} w_{11}^{(1)} & w_{12}^{(1)} & 0 \\ 0 & 0 & w_{23}^{(1)} \\ 0 & 0 & w_{33}^{(1)} \end{pmatrix}$$

$$W^{(2)} = \begin{pmatrix} w_{11}^{(2)} & w_{12}^{(2)} & 0 \\ 0 & 0 & w_{23}^{(2)} \end{pmatrix}$$

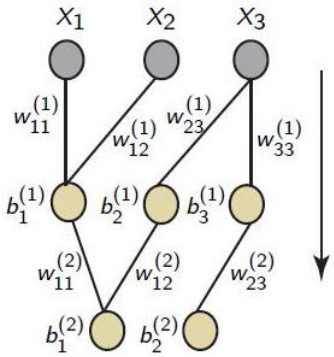


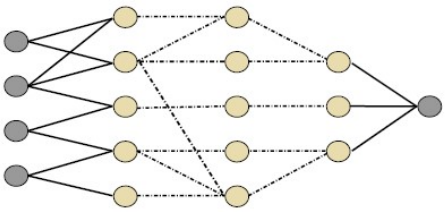
Figure 20: Affine Abbildungen und Gewichte zwischen Neuronen

.

****
ab hier werden weitere mathematischen Grundlagen bald eingefügt werden
****

8.3 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. Siehe das nachfolgende Diagramm mit Beispielwerten:



Definition:

Assume the following notions:

- $d \in \mathbb{N}$: Dimension of input layer.
- L: Number of layers.
- $\rho : \mathbb{R} \to \mathbb{R}$: (Non-linear) function called *activation function*.
- $T_\ell : \mathbb{R}^{N_{\ell-1}} \to \mathbb{R}^{N_\ell}$, $\ell = 1, \ldots, L$, where $T_\ell x = W^{(\ell)} x + b^{(\ell)}$

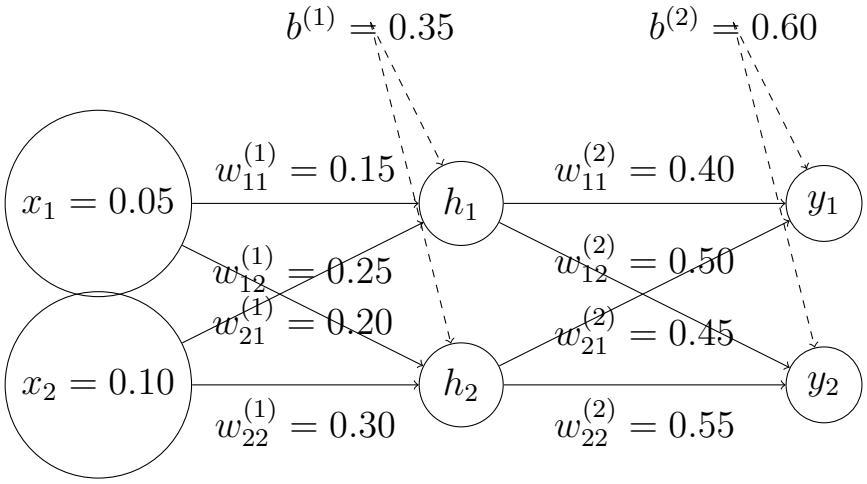
Then $\Phi : \mathbb{R}^d \to \mathbb{R}^{N_L}$ given by

$$\Phi(x) = T_L \rho(T_{L-1} \rho(\ldots \rho(T_1(x))), \quad x \in \mathbb{R}^d,$$

is called *(deep) neural network (DNN)*.

Figure 21: Definition eines Tiefen (Deep) Neuronalen Netzwerkes

.



Voraussetzungen des Beispiels:

- 2 Eingabeknoten: $x_1 = 0.05$, $x_2 = 0.10$

- 2 versteckte Knoten: h_1, h_2
- 2 Ausgabeknoten: y_1, y_2
- Lernrate: $\eta = 0.5$

Gewichte und Biases:

$$W^{(1)} = \begin{pmatrix} 0.15 & 0.25 \\ 0.20 & 0.30 \end{pmatrix}, \quad b^{(1)} = \begin{pmatrix} 0.35 \\ 0.35 \end{pmatrix}$$

$$W^{(2)} = \begin{pmatrix} 0.40 & 0.50 \\ 0.45 & 0.55 \end{pmatrix}, \quad b^{(2)} = \begin{pmatrix} 0.60 \\ 0.60 \end{pmatrix}$$

Zielausgaben:

$$y_{\text{target},1} = 0.01, \quad y_{\text{target},2} = 0.99$$

8.3.1 Berechnungen zum Vorwärtsdurchlauf ("Forward Pass")

Der Vorwärtsdurchlauf (englisch: forward pass) bei Convolutional Neural Networks (CNNs) ist der Prozess, bei dem die Eingabedaten durch das Netzwerk propagiert werden, um eine Vorhersage oder Ausgabe zu generieren. Während des Vorwärtsdurchlaufs werden die Daten Schicht für Schicht verarbeitet, wobei die Gewichtungen, Aktivierungen und Pooling-Operationen angewendet werden, um schließlich die Ausgabe zu erhalten.
Berechnen wir zunächst die gewichtete Summe $(z_1^{(1)}, z_2^{(1)})$ 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.

1. Berechnung der Nettoeingänge für die versteckte Schicht:

$$z_1^{(1)} = x_1 \cdot w_{11}^{(1)} + x_2 \cdot w_{21}^{(1)} + b_1^{(1)} = 0.05 \cdot 0.15 + 0.10 \cdot 0.20 + 0.35 = 0.3775$$

$$z_2^{(1)} = x_1 \cdot w_{12}^{(1)} + x_2 \cdot w_{22}^{(1)} + b_2^{(1)} = 0.05 \cdot 0.25 + 0.10 \cdot 0.30 + 0.35 = 0.3925$$

2. Aktivierung der versteckten Knoten mit der Sigmoid-Funktion:

$$h_1 = \sigma(z_1^{(1)}) = \frac{1}{1 + e^{-0.3775}} = 0.59327$$

$$h_2 = \sigma(z_2^{(1)}) = \frac{1}{1 + e^{-0.3925}} = 0.59688$$

Wir wiederholen diesen Vorgang für die Neuronen der Ausgabeschicht und verwenden die Ausgaben der Neuronen der versteckten Schicht als Eingaben.

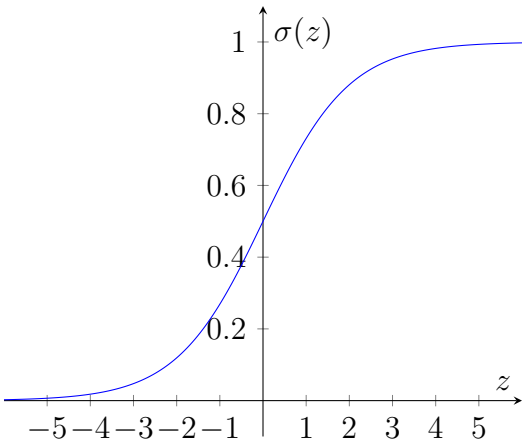
3. Berechnung der Nettoeingänge für die Ausgabeschicht:

$$z_1^{(2)} = h_1 \cdot w_{11}^{(2)} + h_2 \cdot w_{21}^{(2)} + b_1^{(2)} = 0.59327 \cdot 0.40 + 0.59688 \cdot 0.45 + 0.60 = 1.1059$$
$$z_2^{(2)} = h_1 \cdot w_{12}^{(2)} + h_2 \cdot w_{22}^{(2)} + b_2^{(2)} = 0.59327 \cdot 0.5 + 0.59688 \cdot 0.55 + 0.60 = 1.22492$$

4. Aktivierung der Ausgabeknoten:

$$y_1 = \sigma(z_1^{(2)}) = \frac{1}{1 + e^{-1.1059}} = 0.75136$$
$$y_2 = \sigma(z_2^{(2)}) = \frac{1}{1 + e^{-1.22492}} = 0.77293$$

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:



Warum verwendet man die Sigmoid-Funktion im Backpropagation-Prozess?

Die **Sigmoid-Funktion** wird aus mehreren Gründen im Backpropagation-Prozess neuronaler Netze verwendet:

1. **Glättung und Nichtlinearität**: Die Sigmoid-Funktion ist eine nichtlineare Aktivierungsfunktion, die die Eingabe auf einen Bereich zwischen 0 und 1 transformiert. Dies hilft, komplexe Muster und nichtlineare Beziehungen in den Daten zu modellieren, was entscheidend für neuronale Netze ist.

2. **Ableitbarkeit**: Die Sigmoid-Funktion ist leicht ableitbar, was für das Backpropagation-Verfahren, das auf dem Gradientenabstieg basiert, von zentraler Bedeutung ist. Die Ableitung der Sigmoid-Funktion lässt sich mit Hilfe der Kettenregel leicht berechnen. Durch Anwendung der Kettenregel ergibt sich:

$$\sigma'(x) = \frac{e^{-x}}{(1+e^{-x})^2}$$

Da $\sigma(x) = \frac{1}{1+e^{-x}}$, lässt sich die Ableitung auch wie folgt schreiben:

$$\sigma'(x) = \sigma(x) \cdot (1 - \sigma(x))$$

Diese einfache Form der Ableitung ermöglicht eine effiziente Berechnung während der Gewichtsaktualisierung im Backpropagation-Prozess.

3. **Wahrscheinlichkeitsinterpretation**: Da die Ausgaben der Sigmoid-Funktion im Bereich von 0 bis 1 liegen, können sie als Wahrscheinlichkeiten interpretiert werden, was besonders in binären Klassifikationsproblemen nützlich ist.

4. **Vermeidung von extremen Werten**: Die Sigmoid-Funktion beschränkt die Ausgaben auf einen festgelegten Bereich (zwischen 0 und 1), was hilft, extrem große Werte zu vermeiden, die das Lernen destabilisieren könnten.

Nachteile und moderne Alternativen
Ein Nachteil der Sigmoid-Funktion ist das Problem des **vanishing gradient**, bei dem die Ableitungen für sehr positive oder negative Eingaben zu kleinen Werten (nahe 0) führen können, was den Lernprozess verlangsamt. Aus diesem Grund werden in modernen neuronalen Netzen häufig alternative Aktivierungsfunktionen wie die **ReLU** (Rectified Linear Unit) verwendet, um dieses Problem zu umgehen.

8.3.2 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 ("Loss") pro Ausgabeneuron wieder mit Großbuchstaben (L_1, L_2) und den Gesamtfehler als L, so ergibt sich. Verwende die Mean Squared Error (MSE)E als Fehlerfunktion:

$$L = \frac{1}{2}\left((L_1)^2 + (L_2)^2\right) = \frac{1}{2}\left((y_{\text{target},1} - y_1)^2 + (y_{\text{target},2} - y_2)^2\right)$$

$$L = \frac{1}{2}\left((0.01 - 0.75136)^2 + (0.99 - 0.77293)^2\right) = 0.29837$$

8.3.3 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.
Insbesondere kommt die **Kettenregel von "geschachtelten" Funktionen** (siehe Diagramm unten) zur Anwendung (siehe auch Vorlesung "Analysis I + II").Insgesamt sind vier Schritte durchzuführen:

1. Als erstes berechnet man nun den Fehler in der Ausgabeschicht:

Zur Berechnung der Fehler und Gewichtsgradienten in der Ausgabeschicht nutzen wir die Logik der Prozesse im folgenden Diagramm:

.

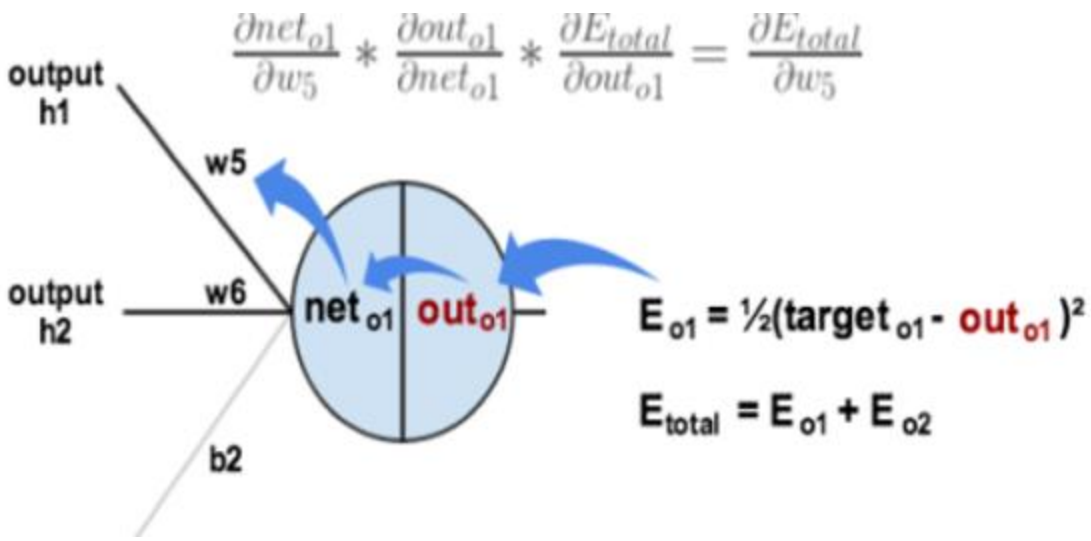


Figure 22: Backpropagation-Rückwärtslauf (Ausgabeschicht)

Wie man leicht sieht, ergibt sich der Fehler in der Ausgabeschicht $\delta_i^{(2)}$ durch:

$$\delta_1^{(2)} = (y_1 - y_{\text{target},1}) \cdot \sigma'(z_1^{(2)}) \quad wobei \quad gilt:$$

$$\sigma'(z_1^{(2)}) = y_1 \cdot (1 - y_1) = 0.75136 \cdot (1 - 0.75136) = 0.18681$$

$$Damit \quad gilt: \quad \delta_1^{(2)} = (0.75136 - 0.01) \cdot 0.18681 = 0.13812$$

Analog gilt auch für den zweiten Ausgabeknoten:

$$\delta_2^{(2)} = (y_2 - y_{\text{target},2}) \cdot \sigma'(z_2^{(2)}) \quad wobei \quad gilt:$$

$$\sigma'(z_2^{(2)}) = y_2 \cdot (1 - y_2) = 0.77293 \cdot (1 - 0.77293) = 0.17544$$

$$Damit \quad gilt: \quad \delta_2^{(2)} = (0.77293 - 0.99) \cdot 0.17544 = -0.03853$$

2. Gewichtsgradienten für die Ausgabeschicht:

Zur Berechnung der Gewichtsgradienten in der Ausgabeschicht nutzen wir die Kettenregel. Man erhält (siehe Übung 8.1):

$$\frac{\partial L}{\partial w_{ij}^{(2)}} = \delta_i^{(2)} \cdot h_j$$

Der Beweis dieser Aussage ist Teil der Übung 8.2. Somit ergibt sich für die Ausgabeschicht:

$$\frac{\partial L}{\partial w_{11}^{(2)}} = \delta_1^{(2)} \cdot h_1 = 0.13812 \cdot 0.59327 = 0.08195$$

$$\frac{\partial L}{\partial w_{12}^{(2)}} = \delta_1^{(2)} \cdot h_2 = 0.13812 \cdot 0.59688 = 0.08243$$

$$\frac{\partial L}{\partial w_{21}^{(2)}} = \delta_2^{(2)} \cdot h_1 = -0.03853 \cdot 0.59327 = -0.02287$$

$$\frac{\partial L}{\partial w_{22}^{(2)}} = \delta_2^{(2)} \cdot h_2 = -0.03853 \cdot 0.59688 = -0.02299$$

3. Fehler in der versteckten Schicht:

Zur Berechnung der Fehler und Gewichtsgradienten in der versteckten Schicht Ausgabeschicht nutzen wir die Logik der Prozesse im folgenden Diagramm:

.

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\downarrow$$

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$

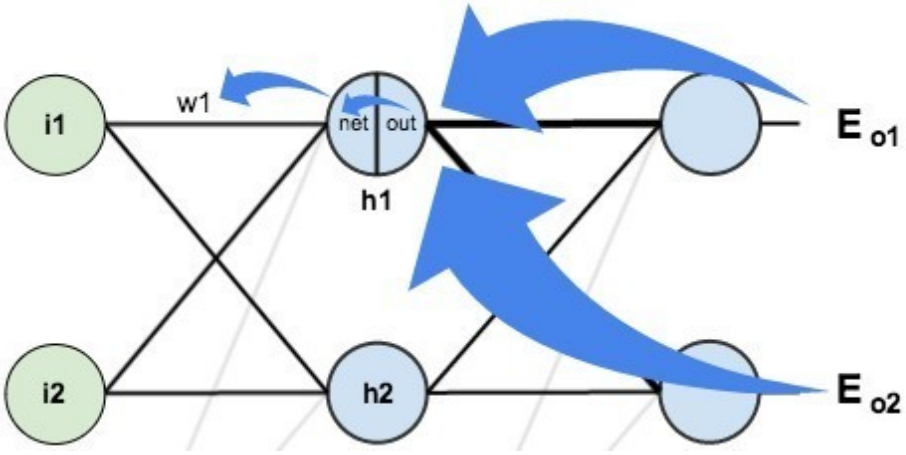


Figure 23: Backpropagation-Rückwärtslauf (versteckte Schicht)

Der Fehler in der versteckten Schicht $\delta_i^{(1)}$ ergibt sich nach obigem Diagramm:

$$\delta_1^{(1)} = (\delta_1^{(2)} \cdot w_{11}^{(2)} + \delta_2^{(2)} \cdot w_{21}^{(2)}) \cdot \sigma'(z_1^{(1)}) \quad wobei \quad gilt:$$

$$\sigma'(z_1^{(1)}) = h_1 \cdot (1 - h_1) = 0.59327 \cdot (1 - 0.59327) = 0.24106$$

$$Also: \quad \delta_1^{(1)} = (0.13812 \cdot 0.40 + -0.03853 \cdot 0.50) \cdot 0.24106 = 0.01367$$

Analog gilt auch für den zweiten versteckten Knoten:

$$\delta_2^{(1)} = (\delta_1^{(2)} \cdot w_{12}^{(2)} + \delta_2^{(2)} \cdot w_{22}^{(2)}) \cdot \sigma'(z_2^{(1)}) \quad wobei \quad gilt:$$

$$\sigma'(z_2^{(1)}) = h_2 \cdot (1 - h_2) = 0.59688 \cdot (1 - 0.59688) = 0.24062$$

$$Also: \quad \delta_2^{(1)} = (0.13812 \cdot 0.45 + -0.03853 \cdot 0.55) \cdot 0.24062 = 0.01427$$

4. Gewichtsgradienten für die versteckte Schicht:

Die Fehlerwerte der versteckten Schicht ($\delta_1^{(1)}$ und $\delta_2^{(1)}$) wurden bereits berechnet:

$$\delta_1^{(1)} = 0.01367, \quad \delta_2^{(1)} = 0.01427$$

Nun berechnen wir die Gradienten der Gewichte für die versteckte Schicht. Zur Berechnung der Gewichtsgradienten in der Ausgabeschicht nutzen wir die wieder die Kettenregel. Der Beweis ist wieder Teil der Übung 8.1:

$$\frac{\partial L}{\partial w_{ij}^{(1)}} = \delta_i^{(1)} \cdot x_j$$

Somit ergibt sich für die versteckte Schicht:

$$\frac{\partial L}{\partial w_{11}^{(1)}} = \delta_1^{(1)} \cdot x_1 = 0.01367 \cdot 0.05 = 0.0006835$$
$$\frac{\partial L}{\partial w_{12}^{(1)}} = \delta_1^{(1)} \cdot x_2 = 0.01367 \cdot 0.10 = 0.001367$$
$$\frac{\partial L}{\partial w_{21}^{(1)}} = \delta_2^{(1)} \cdot x_1 = 0.01427 \cdot 0.05 = 0.0007135$$
$$\frac{\partial L}{\partial w_{22}^{(1)}} = \delta_2^{(1)} \cdot x_2 = 0.01427 \cdot 0.10 = 0.001427$$

8.3.4 Schritt 4: Aktualisierung der Gewichte:

Verwenden wir nun die Lernrate $\eta = 0.5$, um die Gewichte zu aktualisieren, so erhält man:

Neue Gewichte in der Ausgabeschicht:

$$w_{11}^{(2)} = 0.40 - 0.5 \cdot 0.08195 = 0.359025$$
$$w_{12}^{(2)} = 0.45 - 0.5 \cdot 0.08243 = 0.408785$$
$$w_{21}^{(2)} = 0.50 - 0.5 \cdot (-0.02287) = 0.511435$$
$$w_{22}^{(2)} = 0.55 - 0.5 \cdot (-0.02299) = 0.561495$$

Neue Gewichte in der versteckten Schicht:

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

Schicht aktualisiert.

$$w_{11}^{(1)} = 0.15 - 0.5 \cdot 0.0006835 = 0.149658$$
$$w_{12}^{(1)} = 0.25 - 0.5 \cdot 0.001367 = 0.2493165$$
$$w_{21}^{(1)} = 0.20 - 0.5 \cdot 0.0007135 = 0.19964325$$
$$w_{22}^{(1)} = 0.30 - 0.5 \cdot 0.001427 = 0.2992865$$

8.3.5 Schritt 5: Aktualisierung der Biase:

Das Ignorieren der Bias-Anpassungen während des Trainings ist im Allgemeinen ein Fehler, da es das Netzwerk in seiner Lernfähigkeit stark einschränkt. Nach sehr vielen Iterationen könnte sich das Ergebnis zwar stabilisieren, aber es wäre wahrscheinlich nicht optimal, und der Loss bliebe höher als bei einer vollständigen Anpassung der Gewichte und Biases. Um die besten Ergebnisse zu erzielen, sollten sowohl die Gewichte als auch die Biases in jeder Iteration aktualisiert werden.

Biases nach der ersten Iteration
Dieser Abschnitt zeigt die Berechnung und Aktualisierung der Biases nach der ersten Iteration in einem neuronalen Netzwerk.
Sie enthält die Formeln für die Vorwärts- und Rückwärtsdurchläufe sowie die Aktualisierungen der Biases.

Vorwärtsdurchlauf (Iteration 1):

Hidden Layer Aktivierungen:

$$\begin{aligned} z_{\text{hidden},1} &= W_1[0] \cdot x + b_1[0] \\ &= (0.15 \times 0.05) + (0.20 \times 0.10) + 0.35 \\ &= 0.3775 \\ h_1 &= \sigma(0.3775) \approx 0.59327 \end{aligned}$$

$$\begin{aligned} z_{\text{hidden},2} &= W_1[1] \cdot x + b_1[1] \\ &= (0.25 \times 0.05) + (0.30 \times 0.10) + 0.35 \\ &= 0.3925 \\ h_2 &= \sigma(0.3925) \approx 0.59688 \end{aligned}$$

Output Layer Aktivierungen:

$$\begin{aligned} z_{\text{output},1} &= W_2[0] \cdot h + b_2[0] \\ &= (0.40 \times 0.59327) + (0.45 \times 0.59688) + 0.60 \\ &\approx 1.1059 \\ y_1 &= \sigma(1.1059) \approx 0.7514 \end{aligned}$$

$$\begin{aligned} z_{\text{output},2} &= W_2[1] \cdot h + b_2[1] \\ &= (0.50 \times 0.59327) + (0.55 \times 0.59688) + 0.60 \\ &\approx 1.2249 \\ y_2 &= \sigma(1.2249) \approx 0.7729 \end{aligned}$$

Rückwärtsdurchlauf (Backpropagation)

Fehler im Output Layer:

$$\begin{aligned} \text{error}_{\text{output},1} &= y_1 - y_{\text{target},1} \\ &= 0.7514 - 0.01 \\ &= 0.7414 \end{aligned}$$

$$\begin{aligned} \text{error}_{\text{output},2} &= y_2 - y_{\text{target},2} \\ &= 0.7729 - 0.99 \\ &= -0.2171 \end{aligned}$$

Delta für den Output Layer:

$$\begin{aligned} \delta_{\text{output},1} &= \text{error}_{\text{output},1} \times \sigma'(z_{\text{output},1}) \\ &= 0.7414 \times \sigma(1.1059) \times (1 - \sigma(1.1059)) \\ &= 0.7414 \times 0.7514 \times (1 - 0.7514) \\ &\approx 0.1385 \end{aligned}$$

$$\begin{aligned}\delta_{\text{output},2} &= \text{error}_{\text{output},2} \times \sigma'(z_{\text{output},2}) \\ &= -0.2171 \times \sigma(1.2249) \times (1 - \sigma(1.2249)) \\ &\approx -0.0381\end{aligned}$$

Fehler im Hidden Layer:

$$\begin{aligned}\text{error}_{\text{hidden},1} &= W_2[0,0] \times \delta_{\text{output},1} + W_2[1,0] \times \delta_{\text{output},2} \\ &= (0.40 \times 0.1385) + (0.50 \times -0.0381) \\ &\approx 0.0452\end{aligned}$$

$$\begin{aligned}\text{error}_{\text{hidden},2} &= W_2[0,1] \times \delta_{\text{output},1} + W_2[1,1] \times \delta_{\text{output},2} \\ &= (0.45 \times 0.1385) + (0.55 \times -0.0381) \\ &\approx 0.0500\end{aligned}$$

Delta für den Hidden Layer:

Der Gradient des Biases ergibt sich also zu:

$$\frac{\partial E}{\partial b} = \frac{\partial E}{\partial y} \cdot \frac{\partial y}{\partial z} \cdot \frac{\partial z}{\partial b}$$

Das bedeutet:

$$\boxed{\frac{\partial E}{\partial b} = -(y_{\text{target}} - y) \cdot y(1 - y)}$$

Da $\frac{\partial z}{\partial b} = 1$, fällt dieser Term weg, und es bleibt der Einfluss der Aktivierung und des Fehlers übrig.

Gradienten und Bias-Update

Zusammenfassung:

- Der Gradient des Biases zeigt an, wie der Fehler in Bezug auf den Bias ansteigt.
- Er wird durch die Ableitung der Fehlerfunktion und der Aktivierungsfunktion berechnet.

- Der Bias wird in jeder Iteration durch den Gradienten aktualisiert, um den Fehler zu minimieren.

Durch diesen Prozess lernt das neuronale Netzwerk, seine Biases so anzupassen, dass der Fehler minimiert wird und die Zielausgaben besser erreicht werden.

$$\begin{aligned}\delta_{\text{hidden},1} &= \text{error}_{\text{hidden},1} \times \sigma'(z_{\text{hidden},1}) \\ &= 0.0452 \times \sigma(0.3775) \times (1 - \sigma(0.3775)) \\ &\approx 0.0109\end{aligned}$$

$$\begin{aligned}\delta_{\text{hidden},2} &= \text{error}_{\text{hidden},2} \times \sigma'(z_{\text{hidden},2}) \\ &= 0.0500 \times \sigma(0.3925) \times (1 - \sigma(0.3925)) \\ &\approx 0.0120\end{aligned}$$

Aktualisierung der Biases

Im Backpropagation-Algorithmus wird der Bias mit einem Lernschritt basierend auf dem berechneten Gradienten aktualisiert:

Neue Biases für den Hidden Layer b_1:

$$\begin{aligned}b_1[0] &= b_1[0] - \eta \times \delta_{\text{hidden},1} \\ &= 0.35 - 0.5 \times 0.0109 \\ &\approx 0.3446\end{aligned}$$

$$\begin{aligned}b_1[1] &= b_1[1] - \eta \times \delta_{\text{hidden},2} \\ &= 0.35 - 0.5 \times 0.0120 \\ &\approx 0.3440\end{aligned}$$

Neue Biases für den Output Layer b_2:

$$\begin{aligned}b_2[0] &= b_2[0] - \eta \times \delta_{\text{output},1} \\ &= 0.60 - 0.5 \times 0.1385 \\ &\approx 0.5307\end{aligned}$$

$$\begin{aligned} b_2[1] &= b_2[1] - \eta \times \delta_{\text{output},2} \\ &= 0.60 - 0.5 \times (-0.0381) \\ &\approx 0.6191 \end{aligned}$$

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

Zusammenfassung (inklusive Ergebnisse der Übungen):
Wir haben nun alle unsere Gewichte aktualisiert! Als wir die Eingaben von 0,05 und 0,1 ursprünglich weiterleiteten, betrug der Fehler des Netzwerks 0,298371109 (siehe Unterkapitel 8.3.2). Nach dieser ersten Runde der Backpropagation ist der Gesamtfehler nun auf 0,291055 (Übung 8.2) gesunken.
Das mag nicht viel erscheinen, aber wenn man diesen Prozess beispielsweise 10.000 Mal wiederholt (siehe Übung 8.3), sinkt der Fehler auf 0,0000351085. Nach 10.000 Iterationen liegt der wert der oberen Neurone bei 0,015912196 (vs. Zielwert: 0,01) und der Wert der unteren Neurone bei 0,984065734 (vs. Zielwert: 0,99). Der totale Fehler lässt sich nun leicht mit der MSE Formel zu 0,0000351085 berechnen.

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

8.4 Probleme der Backpropagation

Die Backpropagation (Rückwärtsausbreitung) ist eine zentrale Methode zum Trainieren von neuronalen Netzwerken. Obwohl sie weit verbreitet und effektiv ist, gibt es mehrere Probleme und Herausforderungen im Zusammenhang mit Backpropagation, insbesondere in tiefen neuronalen Netzwerken:

8.4.1 Vanishing Gradients (verschwindende Gradienten)

In tiefen Netzwerken (mit vielen Schichten) werden die Gradienten bei der Rückwärtsausbreitung immer kleiner, je weiter sie durch das Netzwerk propagiert werden. Das bedeutet, dass die Gewichte in den frühen Schichten des Netzwerks nur minimal aktualisiert werden, was das Lernen verlangsamt oder sogar verhindert. Dies tritt häufig bei Aktivierungsfunktionen wie der Sigmoid- oder Tanh-Funktion auf, die Gradienten im Bereich von $(0, 1)$ haben.

Lösungsmöglichkeiten:

- Verwendung von alternativen Aktivierungsfunktionen wie der ReLU (Rectified Linear Unit), die das Problem des Vanishing Gradients mindern.
- Initialisierung der Gewichte mit spezielleren Methoden wie Xavier- oder He-Initialization.

8.4.2 Exploding Gradients (explodierende Gradienten)

Im Gegensatz zu den verschwindenden Gradienten können die Gradienten in tiefen Netzwerken manchmal extrem groß werden, was zu instabilen Gewichtsanpassungen führt. Dies kann dazu führen, dass die Gewichte nach nur wenigen Iterationen sehr hohe Werte annehmen, was das Training instabil oder unmöglich macht.

Lösungsmöglichkeiten:

- Anwendung von Gradient Clipping, um die Gradienten in einem bestimmten Bereich zu halten.
- Verwendung von kleineren Lernraten, um zu große Aktualisierungen zu vermeiden.

8.4.3 Lokale Minima

Backpropagation basiert auf Gradientenabstieg , der versucht, die Verlustfunktion zu minimieren. Allerdings kann dieser Prozess in lokale Minima oder Sattelpunkte geraten, anstatt das globale Minimum zu erreichen. Dies ist insbesondere bei komplexen, hochdimensionalen Verlustlandschaften der Fall.

Lösungsmöglichkeiten:

- Verwendung von Stochastic Gradient Descent (SGD) oder Varianten wie Adam, um die Wahrscheinlichkeit zu erhöhen, aus lokalen Minima zu entkommen.
- Erhöhte Modellkomplexität und mehr Daten können helfen, die Anzahl lokaler Minima zu verringern.

Lokales Minimum Problem: Ein einfaches Beispiel

Betrachten wir eine einfach konstruierte eindimensionale Funktion $f(x) = x^4 - 4x^2 + x + 3$ in $\mathbb{R}^2$ mit lokalem Minima bei ca. x = 1,34 und globales Minima bei x = -1,47.

Diese Funktion hat sowohl ein lokales Minimum als auch ein globales Minimum. Siehe auch:
https://github.com/HVoellinger/Mathematische-Grundlagen_von_ML/blob/main/Quellen_Math-ML/Plot_von_SGD-Beispiel.pdf

Wenn wir Backpropagation oder den Gradientenabstiegs-Algorithmus verwenden, um das Minimum der Funktion (f(x) zu finden, könnte der Algorithmus beim lokalen Minimum bei $x \approx 1.34$ steckenbleiben, anstatt weiter zu iterieren, um das globale Minimum bei $x \approx -1,47$ zu erreichen.

Lösungsmöglichkeiten

1. Stochastic Gradient Descent (SGD):

Eine Möglichkeit, aus einem lokalen Minimum herauszukommen, ist die Verwendung von *Stochastic Gradient Descent (SGD)* anstelle von reinem Gradientenabstieg. Beim SGD wird der Gradient auf der Grundlage eines zufällig ausgewählten Teils der Daten berechnet, was zu einer gewissen zufälligen Variation in den Aktualisierungen führt. Diese zufälligen Schwankungen können helfen, aus lokalen Minima herauszukommen, da der Algorithmus manchmal genug Schwung erhält, um das lokale Minimum zu überwinden und weiter nach tieferen (globalen) Minima zu suchen.

2. Weitere Ansätze:

- *Momentum*: Diese Technik hilft dabei, den Gradientenverlauf zu "glätten" und den Algorithmus über lokale Minima hinauszutragen, indem die Bewegung beschleunigt wird.
- *Simulated Annealing (Abkühlung)*: Hierbei handelt es sich um ein adaptives Verfahren, bei dem die Lernrate schrittweise verringert wird. Zu Beginn können größere Schritte unternommen werden, um aus lokalen Minima herauszukommen, während später kleinere Schritte genutzt werden, um sich dem globalen Minimum zu nähern.
- *RMSprop/Adam*: Diese Optimierungsalgorithmen passen die Lernrate dynamisch an, basierend auf den Gradientenverläufen, was oft zu einem effizienteren Auffinden von globalen Minima führt.

Anmerkung1: Das Stochastic Gradient Descent (SGD) wird bei neuronalen Netzen gegenüber dem klassischen (Batch) Gradient Descent (GD) bevorzugt.

GD führt bei jeden Schritt den Algorithmus über den gesamten Datensatz aus. Das ist bei sehr großen Datensätzen extrem rechenaufwändig und langsam.
Im Gegensatz dazu verwendet SGD zufällig ausgewählte Minibatches (meist nur ein kleiner Teil der Daten), um bei jedem Schritt den Gradienten zu berechnen. Dadurch kann es häufiger Updates durchführen, was zu einer schnelleren Konvergenz führt.

Anmerkung2: Die Lösung mit den SGD Verfahren soll für obiges Beispiel in Übung 8.5 durchgeführt werden.

8.4.4 Langsames Konvergieren

Backpropagation kann, besonders bei tieferen Netzwerken, sehr langsam konvergieren, da die Gewichtsaktualisierungen oft nur geringfügig sind. Dies führt zu langen Trainingszeiten, insbesondere bei großen und komplexen Datensätzen.

Lösungsmöglichkeiten:

- Nutzung von adaptiven Optimierungsverfahren wie Adam oder RMSprop, die die Lernrate dynamisch anpassen.
- Batch-Normalisierung kann helfen, die Konvergenz zu beschleunigen, indem sie die Aktivierungen normalisiert und den Gradientenfluss verbessert.

8.4.5 Anfälligkeit für Überanpassung (Overfitting)

Backpropagation, insbesondere bei tiefen Netzwerken, kann dazu führen, dass das Modell die Trainingsdaten zu gut lernt, sodass es nicht gut auf neue, ungesehene Daten generalisiert. Dies wird als Overfitting bezeichnet.

Lösungsmöglichkeiten:

- Anwendung von Regularisierungstechniken wie L2-Regularisierung oder Dropout, um Overfitting zu reduzieren.
- Mehr Trainingsdaten können ebenfalls helfen, Overfitting zu vermeiden.

8.4.6 Rechenaufwand und Speicherbedarf

Backpropagation erfordert eine große Anzahl von Berechnungen und viel Speicherplatz, insbesondere für sehr tiefe Netzwerke oder große Datensätze. Jede Iteration

umfasst mehrere Matrixmultiplikationen, und die Speicherung von Zwischenergebnissen (Aktivierungen, Gradienten) kann viel Speicher beanspruchen.

Lösungsmöglichkeiten:

- Verwendung von GPUs und TPUs, um die Berechnungen zu beschleunigen.
- Effizientere Speicherverwaltung und Ansätze wie Gradient Checkpointing können den Speicherbedarf reduzieren.

8.4.7 Abhängigkeit von der Lernrate

Backpropagation erfordert eine gut gewählte Lernrate. Eine zu kleine Lernrate führt zu sehr langsamen Fortschritten, während eine zu große Lernrate zu instabilen Aktualisierungen und schlechtem Konvergieren führen kann. Die Wahl der richtigen Lernrate ist daher entscheidend, aber oft schwierig.

Lösungsmöglichkeiten:

- Adaptive Lernratenverfahren wie Adam oder AdaGrad passen die Lernrate während des Trainings an und können die Notwendigkeit, eine feste Lernrate zu wählen, reduzieren.
- Lernraten-Decay oder das schrittweise Reduzieren der Lernrate während des Trainings kann ebenfalls hilfreich sein.

8.4.8 Schwierigkeiten bei unsupervised learning

Backpropagation ist hauptsächlich für *supervised learning* geeignet, da sie eine bekannte Zielausgabe benötigt, um den Fehler berechnen zu können. Für *unsupervised learning*-Aufgaben wie Clustering oder Dimensionenreduktion gibt es noch keine vollständig befriedigenden Backpropagation-Ansätze.

Lösungsmöglichkeiten:

- Ansätze wie Autoencoder oder Generative Adversarial Networks (GANs) versuchen, Backpropagation auch für *unsupervised learning* nutzbar zu machen.

Zusammenfassung

Obwohl Backpropagation ein mächtiger Algorithmus ist, bringt er auch verschiedene Herausforderungen mit sich:

- *Vanishing/Exploding Gradients*,
- *lokale Minima*,
- *langsames Konvergieren*,
- *Overfitting*,
- *hoher Rechenaufwand*,
- und die *Abhängigkeit von der Lernrate*.

Diese Probleme können jedoch durch verschiedene Optimierungstechniken und alternative Architekturen angegangen werden, die das Training tiefer neuronaler Netzwerke verbessern.

8.5 Übungen zum Kapitel 8

8.5.1 Übung 8.1 - Formeln per Kettenregel für Gewichtsgradienten

Nutze die Kettenregel der partiellen Ableitungen zur Herleitung der zwei Formeln für Ausgabeschicht

$$\frac{\partial L}{\partial w_{ij}^{(2)}} = \delta_i^{(2)} \cdot h_j$$

uns versteckte Schicht

$$\frac{\partial L}{\partial w_{ij}^{(1)}} = \delta_i^{(1)} \cdot x_j$$

8.5.2 Übung 8.2 - Berechnung des Fehlers nach einer Iteration

Zeigen Sie das der Fehler nach einer Iteration für unser Beispiel auf 0,291027924 sinkt.

8.5.3 Übung 8.3 - Jupyter Notebook zur Fehlerberechnung bei Backpropagation

Entwickle ein Jupyter Notebook zur Fehlerberechnung bei Backpropagation.
Wie groß ist der Fehler nach 10.000 Iterationen bei unserem obigen Beispiel?

8.5.4 Übung 8.4 - Berechnung/Beschreibung der zwei ersten Bias-Iterationen

Beschreibe das Vorgehen mit einem Latex Dokument.
Entwickle ein Jupyter Notebook zur Berechnung dieser beiden Iterationen.

8.5.5 Übung 8.5 - *Lokales Minimum Problem* mit SGD Verfahren

Beschreibe das Vorgehen für das Beispiel im Unterkapitel 8.4.3 mit einem Latex Dokument. Wie kann man dies in einem normalen Backpropagation Prozess anwenden?

9 Anhänge

In den Anhängen werden einige weitere Verfahren des Maschinellen Lernens der Vollständigkeit halber erwähnt - ohne zu sehr in die inhaltliche 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 niedrigdimensionalen Matrizen (Benutzermatrix und Objektmatrix) zerlegt, und Empfehlungen werden basierend auf den latenten Vektoren berechnet.
3. Content-Based Filtering (Inhaltsbasierte Filterung): Hier werden Empfehlungen basierend auf den Merkmalen (Eigenschaften) der Objekte und den Präferenzen der Benutzer generiert. Ähnlichkeiten zwischen Objekten werden durch Ähnlichkeitsmaße wie den Kosinusähnlichkeitswert der Merkmale berechnet.
4. Singular Value Decomposition (SVD): SVD ist eine Technik der linearen Algebra, die zur Dimensionsreduktion und Matrizenfaktorisierung verwendet wird. Im Zusammenhang mit Recommender Systems wird SVD häufig in der Matrixfaktorisierung für kollaborative Filterung angewendet.
5. Deep Learning: In den letzten Jahren haben Recommender Systems auch von den Fortschritten im Bereich des Deep Learning profitiert. Hier werden neuronale Netzwerke verwendet, um komplexe Muster in den Daten zu lernen und präzisere Empfehlungen zu generieren.

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

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

9.2 Regularisierungen und Lineare Algebra (LA)

9.2.1 Regularisierung in maschinellem Lernen

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

Es gibt zwei gängige Arten von Regularisierung:

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

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

9.2.2 Lineare Algebra in Bezug auf Regularisierung

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

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

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

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

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

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

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

9.3 Hauptkomponentenanalyse "Principal Component Analysis"

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

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

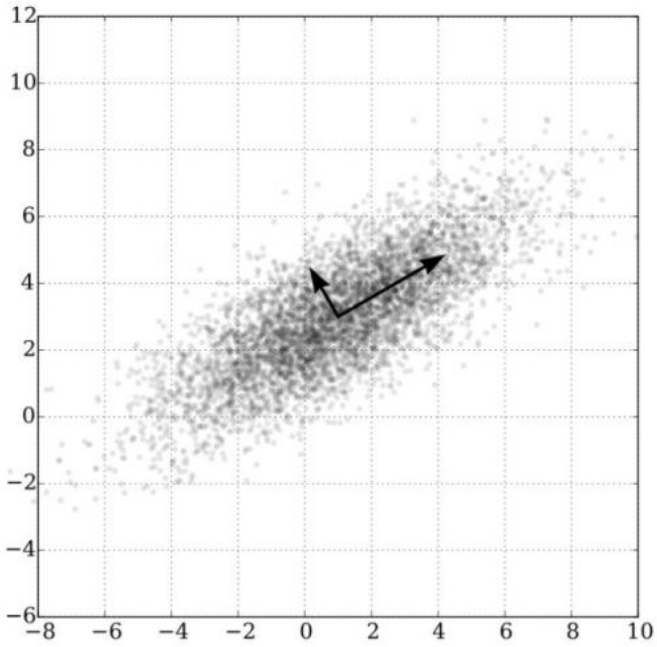


Figure 24: Schematische Darstellung der Hauptkomponentenanalyse

Zwei Hauptkomponenten einer zweidimensionalen Normalverteilung mit Mittelwert

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

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

9.4 Singulärwertzerlegung "Singular Value Decomposition" (SVD)

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

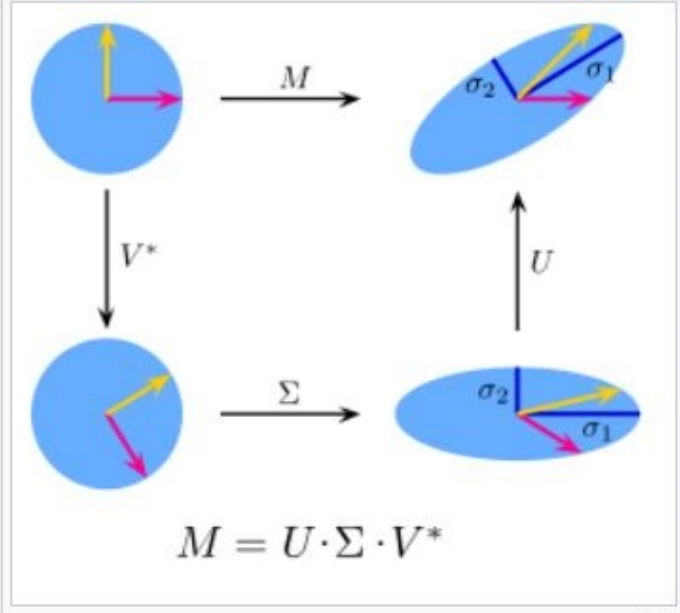


Figure 25: Schematische Darstellung des Singulärwertzerlegung

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-

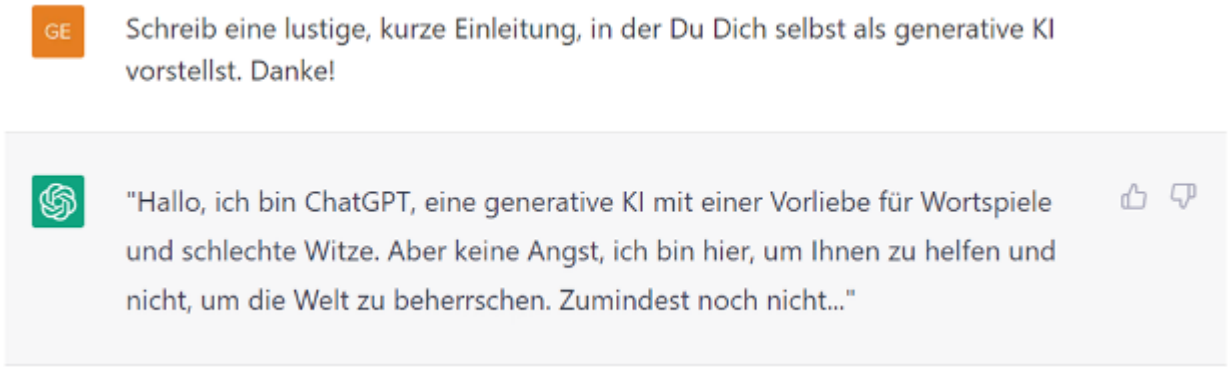
tungen und Pooling-Operationen zum Einsatz, um räumliche Muster in den Daten zu erfassen. Diese Operationen basieren auf mathematischen Konzepten aus der Signalverarbeitung.

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.



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.

10.1 Lösungshinweise zu Übungen Kapitel 3

Weitere Hilfe unter:
[**HVö-GitML20**] https://github.com/HVoellinger/Lecture-Notes-to-ML-WS2020

10.1.1 Lösungshinweise zu Übung 3.1

Manuelle Berechnung, siehe:
https://github.com/HVoellinger/Lecture-Notes-to-ML-WS2020/blob/master/Manuelle_Berechnung_zu_k-Means-Algorithmus.pdf

Die Plausibilität der Lösung ergibt sich sehr einfach aus der Anfangssituation. Siehe auch Lösung zur Übung 3.3.

10.1.2 Lösungshinweise zu Übung 3.2

Python Code zur Übung 3.2, siehe:
https://github.com/HVoellinger/Lecture-Notes-to-ML-WS2020/blob/master/Homework_H3.4_k-Means_Clustering.pdf

10.1.3 Lösungshinweise zu Übung 3.3

Python Code zur Übung 3.3, siehe:
https://github.com/HVoellinger/Lecture-Notes-to-ML-WS2020/blob/master/Beispiel-3-Means-Cluster.pdf

10.2 Lösungshinweise zu Übungen Kapitel 4

10.2.1 Lösungshinweise zu Übung 4.1

Teil A: https://github.com/HVoellinger/Lecture-Notes-to-ML-WS2020/blob/master/Solution_of_Homework_3.1_with_Entropy.pdf

Teil B: https://github.com/HVoellinger/Lecture-Notes-to-ML-WS2020/blob/master/Solution_of_Homework_3.1_with_Gini-Index.pdf

10.2.2 Lösungshinweise zu Übung 4.2

Teil A: https://github.com/HVoellinger/Lecture-Notes-to-ML-WS2020/blob/master/Homework_H4.5-DecTree_ID3.pdf

10.2.3 Lösungshinweise zu Übung 4.3

Für eine Lösung vergleiche:
https://github.com/HVoellinger/Lecture-Notes-to-ML-WS2020/blob/master/Solution_of_Homework_3.2-Version2.xlsx

10.2.4 Lösungshinweise zu Übung 4.4

Für eine Lösung vergleiche:
https://github.com/HVoellinger/Lecture-Notes-to-ML-WS2020/blob/master/ML4-Homework-H4_3-JupyterNotebook.pdf

10.2.5 Lösungshinweise zu Übung 4.5

Für eine Lösung vergleiche:
https://github.com/HVoellinger/Lecture-Notes-to-ML-WS2020/blob/master/Homework_H4.4-Summary_Article.pdf

10.3 Lösungshinweise zu Übungen Kapitel 5

Weitere Hilfe [**HVö-GitML20**]:
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

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

siehe auch konkrete Werte in der entsprechenden Tabelle im Skript.

10.3.5 Lösungshinweise zu Übung 5.5

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

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

. Siehe Berechnung zur animierten Darstellung dieser Hyperebene:
→ R^2 = 1 - Sum((yi-y(xi))²)/Sum((yi-M(y))²) = 1 -(134,2172/922,1) ⌣ 0,8544

Teil 2:
→ opt. Regression-Gerade: y = 15,164 + 3,479*x

→ R^2 = 1 - Sum((yi-y(xi))²)/Sum((yi-M(y))²) = 1 -(189,6999/922,1) ⌣ 0,7943

Zusatzfragen:

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

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[h]

Question 3': x= (25-15,164)/3,479=2,83[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:
Berechnungen als Excel zum Download: https://github.com/HVoellinger/Mathematische-Grundla
von_ML/blob/main/Quellen_Math-ML/LR-Calculation_of_Coeff.xlsx
PDF zu Anschauen der Berechnungen: https://github.com/HVoellinger/Mathematische-Grundlag
von_ML/blob/main/Quellen_Math-ML/LR-Calculation_of_Coeff.pdf

10.3.7 Lösungshinweise zu Übung 5.7

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

Siehe Berechnung zur animierten Darstellung dieser Hyperebene:
https://github.com/HVoellinger/Mathematische-Grundlagen_von_ML/blob/main/Quellen_Math-ML/Animation_Hyperebene-Aufg(5,7).pdf

Ein bewegtes Bild davon liegt in der Referenz: https://github.com/HVoellinger/Mathematische-Grundlagen_von_ML/blob/main/Images/hyperplane_animation1.gif

→ Adj.R^2= 1 -(1-R^2)*(9/7) ∽ 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[h]
Homework y = (25-13,264)/1,382 = 8,49[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.pdf

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 $X^\top \cdot \widetilde{y}$ die entsprechenden (kxk)-Matrix und (kx1)-Vektoren:
https://github.com/HVoellinger/Mathematische-Grundlagen_von_ML/blob/main/Images/Korollar-(K-5.5)-Seite1.jpg
Berechnen wird die notwendigen Matrixen-Multiplikaionenen so erhalten wir für k=1 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

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 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 Links:
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).

10.3.11 Lösungshinweise zu Übung 5.11

Die komplette Lösung zum Jupyter Notebook des Beispiels "Iowa Housing Prices" wird dargestellt im folgenden Video: https://www.youtube.com/watch?v=Mcs2x5-7bc0
Die sLR-Gerade ist im folgenden Bild sichtbar:

Figure 26: sLR-Gerade im Beispiel "Iowa-Housing-Prices"

10.4 Lösungshinweise zu Übungen Kapitel 6

10.4.1 Lösungshinweise zu Übung 6.1

Manuelle Berechnung siehe:
https://github.com/HVoellinger/Lecture-Notes-to-ML-WS2020/blob/master/Homework_H3.2-Bayes_Learning_for_Text_Classification-Folien.pdf

10.4.2 Lösungshinweise zu Übung 6.2

Zusatzfrage:
Das Ergebnis dreht sich um, wenn Sie als Zielsatz **"Hermann plays a very clean game"** eingeben. Berechnung analog wie bei Lösung von Übung 6.1

10.4.3 Lösungshinweise zu Übung 6.3

Lösung mit Python Code siehe:
https://github.com/HVoellinger/Lecture-Notes-to-ML-WS2020/blob/master/Homework_H3.2-Bayes_Learning_for_Text_Classification.pdf

10.5 Lösungshinweise zu Übungen Kapitel 7

10.5.1 Lösungshinweise zu Übung 7.1

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/Quellen_Math-ML/Kernel-Hyperebene-Animation.pdf

10.5.2 Lösungshinweise zu Übung 7.2

Hier ein Beispiel Notebook:

https://github.com/HVoellinger/Mathematische-Grundlagen_von_ML/blob/main/Quellen_Math-ML/SVM_EinfachBsp.pdf

10.5.3 Lösungshinweise zu Übung 7.3

Falls der Grad d des Polynoms im mittleren Bereich (d = 10 bis 30) ist, erhalten wir relativ gute Ergebnisse. Bei keineren d oder grösseren d werden die Ergebnisse entsprechend schlechter.
Setze für diesen Bereich eine Lösung der Polynom-Kernel Funktionen für den Grad d=10 und d=20 um mit vergleichbaren Ergebnissen um:

d=10: https://github.com/HVoellinger/Mathematische-Grundlagen_von_ML/blob/main/Quellen_Math-ML/SVM_Trennlinie-PK(10)-10DP.pdf

d=20:https://github.com/HVoellinger/Mathematische-Grundlagen_von_ML/blob/main/Quellen_Math-ML/SVM_Trennlinie-PK(20)-10DP.pdf

10.5.4 Lösungshinweise zu Übung 7.4

Hier ein Beispiel Notebook:

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

10.5.5 Lösungshinweise zu Übung 7.5

Wie nicht anders zu erwarten, ist das Lineare SVM Verfahren ungeeignet für die Regression. RBF und Polynom(Grad=3)passen aber vergleichbar gut als Lösung. Eine

Abschätzung welches Verfahren final besser ist, ist mit bloßen Auge nicht erkennbar. Dazu sind die entsprechenden Regression Qualitätsparameter zu berechnen.

https://github.com/HVoellinger/Mathematische-Grundlagen_von_ML/blob/main/Quellen_Math-ML/SVR_Example-JupyterNotebook.pdf

10.5.6 Lösungshinweise zu Übung 7.6

Wie nicht anders zu erwarten, sind die Ergebnisse vergleichbar. Die richtige Methode ist SVR(Typ=linear). Typ RBF oder Polyn(Grad=3) passen nicht. Hier die Ergebnisse als Jupyter Notebook:

https://github.com/HVoellinger/Mathematische-Grundlagen_von_ML/blob/main/Quellen_Math-ML/SVR_SimLinRegression-JupyterNotebook.pdf

10.5.7 Lösungshinweise zu Übung 7.7

Analog zu dem Lösungshinweis bei Übung (7.6)
https://github.com/HVoellinger/Mathematische-Grundlagen_von_ML/blob/main/Quellen_Math-ML/SVR_MultLinRegression.pdf

10.6 Lösungshinweise zu Übungen Kapitel 8

10.6.1 Lösungshinweise zu Übung 8.1

Siehe folgende Referenz: [**BackP-Exam**]
https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/
oder allgemeiner (Wikipedia über Backpropagation):
http://en.wikipedia.org/wiki/Backpropagation#Finding_the_derivative_of_the_error

10.6.2 Lösungshinweise zu Übung 8.2

Siehe:
https://github.com/HVoellinger/Mathematische-Grundlagen_von_ML/blob/main/Quellen_Math-ML/Forwardpropagation-Beispiel.pdf

10.6.3 Lösungshinweise zu Übung 8.3

Hier ein Beispiel Notebook:
https://github.com/HVoellinger/Mathematische-Grundlagen_von_ML/blob/main/Quellen_Math-ML/Backpropagation-JupyterNotebook.pdf

10.6.4 Lösungshinweise zu Übung 8.4

Latex-Beschreibung der Vorgehensweise:
https://github.com/HVoellinger/Mathematische-Grundlagen_von_ML/blob/main/Quellen_Math-ML/Backpropagation-Biasberechnung-Beispiel.pdf

Hier ein Beispiel Notebook:
https://github.com/HVoellinger/Mathematische-Grundlagen_von_ML/blob/main/Quellen_Math-ML/Iterationen_von_Biases-JupyterNotebook.pdf

10.6.5 Lösungshinweise zu Übung 8.5

Latex-Beschreibung der Vorgehensweise:
https://github.com/HVoellinger/Mathematische-Grundlagen_von_ML/blob/main/Quellen_Math-ML/SGD-Verfahren_Lokale-Minima-neu.pdf

Lösungsansatz mit Python:
https://github.com/HVoellinger/Mathematische-Grundlagen_von_ML/blob/main/Quellen_Math-ML/Plot_Beispiel_mit_SGD.pdf

11 Referenzen

Liste der Referenzen:

[**HVö-LecDWH23**] - Hermann Völlinger: Script of the Lecture "Introduction to Data Warehousing"; DHBW Stuttgart; WS2023; http://www.dhbw-stuttgart.de/~hvoellin/

[**HVö-ExcDWH23**] - Hermann Völlinger and Other: Exercises and Solutions of the Lecture "Introduction to Data Warehousing"; DHBW Stuttgart; WS2023; http://www.dhbw-stuttgart.de/~hvoellin/

[**HVö-ExcML20**] - 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-LecML20**] - Hermann Völlinger: Script of the Lecture "Machine Learning: Concepts and Algorithms"; DHBW Stuttgart; WS2020; http://www.dhbw-stuttgart.de/~hvoellin/

[**HVö-GitML20**] - 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)*

[**GIM-BFMDW**] - IBM General Information Manual (GIM): "IBM Banking and Financial Markets Data Warehouse"; https://www.ibm.com/downloads/cas/A3ZR8JQY.

[**BDW+IIW**] - H. Völlinger und A. Tarabrin: "Modelle mit Industriemuster: Moderne Informationsstrukturen für brachenspezifische Lösungen"; Übersichtsartikel in der Zeitschrift Objektspektrum 1/2013.

[**TDWI-DatenFlüsse**] - TDWI BI-Spektrum 05/2023 - Fachartikel "Der (Daten-)Weg ist das Ziel - Datenflüsse besser verstehen"; A. Weininger u.a.

[**TDWI-Feature Learning**] - TDWI BI-Spektrum 05/2023 - Fachartikel "KI in der relationalen Welt - Feature Learning"; F. Ullmann

[**LLMsAndKGs**] - "Unifying Large Language Models and Knowledge Graphs: A Roadmap"; JOURNAL OF LATEX CLASS FILES, VOL. 14, NO. 8, AUGUST 2021.

[**MathAndAI**] - Prof. Gitta Kutyniok (Uni München): "The Mathematics of Artifi

cial Intelligence"; Übersichtsartikel, aus einem eingeladenen Vortrag auf dem Internationalen Kongress der Mathematiker 2022. https://arxiv.org/pdf/2203.08890

[**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 Dokumennt; ResearchGate arXiv: 9 Mai 2021; Dieser Review wir als Buch-Kapitel im Buch "Theory of Deep Learning" in Cambridge University Press erscheinen. https://arxiv.org/abs/2105.04026

[**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://github.com/HVoellinger/Mathematische-Grundlagen_von_ML/blob/main/Quellen_Math-ML/SVM_Klassifikation.pdf

[**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_learningMachine Learning

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

Anmerkungen: Alle obigen Dokumente und Artikel, außer den direkten Links ins Internet, findet man auch in meiner GitHub: https://github.com/HVoellinger/

Mathematische-Grundlagen_von_ML/tree/main/Quellen_Math-ML.

Index