Data Mining und Maschinelles Lernen

Zusammenfassung

Fabian Damken

8. November 2023

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Inhaltsverzeichnis

1 Einleitung **8**
- 1.1 Geschichte . . . 8
  - 1.1.1 Das Perzeptron . . . 9
- 1.2 KI Heute . . . 9
- 1.3 Was ist Maschinelles Lernen? . . . 10
  - 1.3.1 Kurzgefasst . . . 11

2 Grundlagen **13**
- 2.1 CRISP-DM: Verlaufsmodell des Data Mining . . . 13
- 2.2 Klassifikation und Regression . . . 13
- 2.3 Statistik . . . 14
  - 2.3.1 Erwartungswert, Varianz und Standardabweichung . . . 14
  - 2.3.2 Bias . . . 15
  - 2.3.3 Normalverteilung . . . 15
  - 2.3.4 Bedingte Wahrscheinlichkeiten . . . 15
  - 2.3.5 Bayes-Statistik . . . 15
  - 2.3.6 Konfidenzintervalle . . . 16
  - 2.3.7 Entropie . . . 16

3 k-Nächste Nachbarn (kNN) **18**
- 3.1 Ähnlichkeitsmaße . . . 18
- 3.2 Auswahlfunktion . . . 19
- 3.3 Überanpassung . . . 20
- 3.4 Asymptotische Ergebnisse und Fluch der hohen Dimension . . . 20

4 Lineare Modelle und Funktionsapproximation **21**
- 4.1 Lineare Modelle . . . 21
- 4.2 Fehler . . . 22
  - 4.2.1 Bias und Varianz . . . 22
- 4.3 Gütekriterien . . . 23
  - 4.3.1 Verlustfunktionen . . . 24
  - 4.3.2 Likelihood . . . 24
- 4.4 Logistische Regression . . . 25

5 Modellselektion und Evaluation **26**
- 5.1 Aufteilung in Trainings- und Testmenge: Kreuzvalidierung . . . 26
- 5.2 Bayes'sche Modellselektion . . . 26
  - 5.2.1 Approximation der Modell-Likelihood und Bayes'sches Informationskriterium . . . 27
  - 5.2.2 Minimale Beschreibungslänge . . . 27

5.3 Evaluierungsmaße . . . 28
5.3.1 Konfusionsmatrix und Gütemaße . . . 28
5.3.2 ROC-Analyse und -Kurve . . . 29
5.3.3 Präzision, Sensitivität und F1-Wert . . . 30
5.4 Vergleichen von Algorithmen . . . 30

6 Baumbasierte Verfahren 32
6.1 Top-Down Induction of Decision Trees (TDIDT): ID3 . . . 32
6.2 Gütemaße . . . 33
6.2.1 Informationsgewinn . . . 33
6.2.2 Gini-Index . . . 33
6.2.3 Regression . . . 34
6.3 Stutzen (Pruning) des Baumes . . . 34

7 Ensemble-Methoden 35
7.1 Zufallswälder (Random Forests) . . . 35
7.1.1 Bagging Allgemein . . . 36
7.2 Boosting . . . 37
7.2.1 AdaBoost . . . 37
7.3 Gradienten Boosting . . . 39
7.3.1 Funktionaler Gradient . . . 40

8 Probabilistische Graphische Modelle und Stützvektormethode 41
8.1 (Naiver) Bayes-Klassifikator . . . 41
8.2 Bayes'sche Netzwerke . . . 42
8.2.1 Variablenelimination . . . 42
8.3 Parameterschätzung . . . 44
8.3.1 Vollständige Daten: Maximum Likelihood . . . 44
8.3.2 Unvollständige Daten: Expectation Maximization (EM) . . . 45
8.4 Diskriminative Ansätze . . . 45
8.4.1 Stützvektormethode . . . 46
8.4.2 Nicht linear trennbare Daten . . . 49

9 Cluster-Analyse 50
9.1 Dendrogramme und Hierarchische Cluster-Analyse . . . 50
9.1.1 (Vorgetäuschte) Strukturen, Anzahl Cluster und Ausreißer . . . 51
9.2 Partitionierung und K-Means . . . 51

10 Tiefes Lernen und Faltende Neuronale Netzwerke 54
10.1 Faltungsschichten . . . 54
10.2 Räumliche Zusammenfassung: (Max) Pooling . . . 55
10.3 Aktivierungsfunktionen . . . 55
10.4 Training . . . 57
10.4.1 Initialisierung . . . 57
10.4.2 Stochastischer Gradientenabstieg . . . 57
10.4.3 Rückwärtspropagation . . . 58
10.5 Vermeidung von Überanpassung . . . 59
10.6 Fine-Tuning und Transfer Learning . . . 60

10.7 Täuschen von CNNs 60

11 Data Mining: Apriori und PageRank **61**
11.1 Apriori 61
11.1.1 Regelbewertung 62
11.2 Web Mining 62
11.2.1 PageRank 62

Abbildungsverzeichnis

1.1 Zusammenhang von künstlicher Intelligenz, maschinellem Lernen und tiefem Lernen. 8
1.2 Darstellung eines einschichtigen Perzeptrons. 10
1.3 Darstellung eines mehrschichtigen Perzeptrons. 10
1.4 Traditionelle Programmierung (links) im Vergleich zu maschinellem Lernen (rechts). 11
1.5 Kreislauf der Erstellung von ML-Komponenten. 11
1.6 Arten des maschinellen Lernens. 12

2.1 Grafische Übersicht über das CRISP-DM. Autor: Kenneth Jensen Lizenz: CC BY-SA 3.0 Quelle: Wikipedia Commons, Datei CRISP-DM_Process_Diagram.png 14

7.1 Illustration von Bagging und Boosting. 35

8.1 Beispiel für ein Bayes'sches Netzwerk mit den Zufallsvariablen A, B, C, D und E. 43
8.2 Bayes-Netz als Beispiel zur Variablenelimination. 44
8.3 Durchführung der Variablenelimination für das Bayes-Netz aus Abbildung 8.2. 44
8.4 Illustration eines einfachen Klassifikationsalgorithmus. 47

9.1 Beispiel für ein einfaches Dendrogramm. 51
9.2 Dendrogramm für einige Flaggen. 52

10.1 Berechnungsgraph für $\big((x+y)z\big)^2$, wobei die Werte der Vorwärtspropagation über und die der Rückwärtspropagation unter den Kanten stehen. Für einige Ableitungen ist zusätzlich der Rechenweg notiert, der für die anderen Kanten analog gilt. 58

Tabellenverzeichnis

Liste der Algorithmen

1 Erstellung einer ROC-Kurve 29

2 ID3-Algorithmus 33

3 AdaBoost (für zwei Klassen) 38

4 K-Means 53

5 Apriori: Berechnung der häufigen Mengen 63
6 Apriori: Regelerzeugung 64
7 Apriori 64

1 Einleitung

Die Veranstaltung „Data Mining und Maschinelles Lernen“ behandelt, ebenso wie diese Zusammenfassung, den Teilbereich des maschinellen Lernens der künstlichen Intelligenz. Dabei werden Algorithmen entwickelt, durch die ein Computer sich selbstständig verbessert. Ein Teilgebiet dieses maschinellen Lernens ist das *tiefe Lernen* (DL, für *Deep Learning*), bei dem tiefe künstliche neuronale Netzwerke genutzt werden (dies wird im Kapitel 10 näher behandelt). Dieser Zusammenhang ist in Abbildung 1.1 dargestellt.

Diese Zusammenfassung wird in die folgenden Bereiche einführen: k-Nächste Nachbarn, Lineare Modelle und Funktionsapproximation, Modellselektion und Evaluierung, Entscheidungsbäume, Ensemble-Methoden, Naive Bayes und Bayes-Netzwerke, die Stützvektormethode, Clusteranalyse und Assoziationsregeln und (Tiefe) Neuronale Netzwerke. Viele andere Bereiche werden allerdings auch nicht abgedeckt, bspw. Variational Learning, Details des Deep Learning, Gaussian Processes, Graphische Modelle, Kausalität, . . . Diese Inhalte sind zu Teilen in den Zusammenfassungen für die Kurse „Statistical Machine Learning“, „Probabilistic Graphical Models“ (noch nicht verfügbar, voraussichtlich im Wintersemester 2021/22), „Statistical Relational AI“ (noch nicht verfügbar, voraussichtlich im Wintersemester 2021/22) sowie „Deep Learning: Architectures and Methods“ (bald verfügbar) zu finden.

1.1 Geschichte

Im Gegensatz zu den meisten Vermutungen hat die künstliche Intelligenz bereits eine lange Vergangenheit:

1950er Geburt der künstlichen Intelligenz

1960er Ära der Perzeptrons

1970er Erster KI-Winter

1980er Ära der Expertensysteme

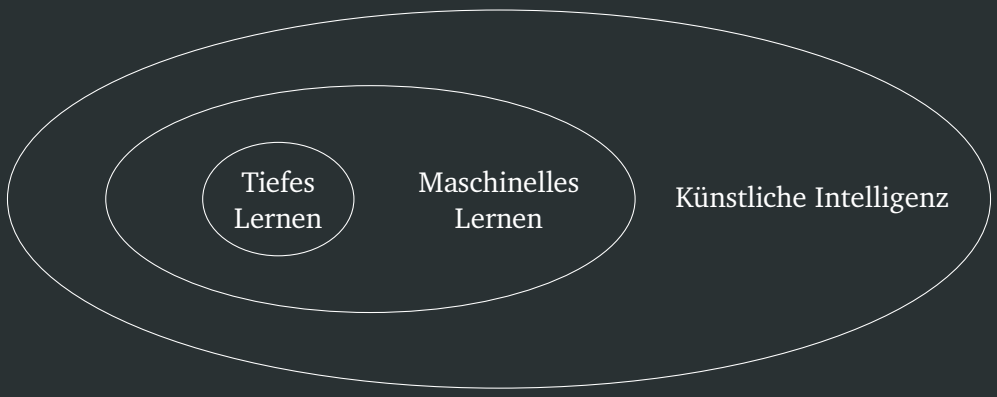

Abbildung 1.1: Zusammenhang von künstlicher Intelligenz, maschinellem Lernen und tiefem Lernen.

1990er Zweiter KI-Winter

2000er Ära des statistischen maschinellen Lernens

2010er Ära des tiefen Lernens

Das Perzeptron, welches im folgenden Abschnitt vorgestellt wird, hat die ersten großen Ergebnisse in der KI-Forschung produziert. Es hat zwar eine sehr mächtige Vorhersagekraft, es gibt jedoch Probleme, die nicht mit einem Perzeptron lösbar sind (Minsky, 1969). Diese Tatsache hat anschließend zu dem ersten KI-Winter geführt, in dem wenig geforscht wurde und das öffentliche Interesse abgeebbt ist. Gefolgt ist die Ära der Expertensysteme, Fall- und Regal-basierte KI-Systeme, die durch einen Menschen und logisches Schlussfolgern erstellt wurden. Jedoch haben auch diese Systeme zu viel versprochen und das öffentliche Interesse ist schnell abgeebbt.

Nun betrat das moderne maschinelle Lernen mit dem *statistischen* maschinellen Lernen das Feld, welches durch komplizierte statistische Modelle angetrieben und motiviert wurde. Die Ergebnisse haben sehr viele Erfolge gebracht, es gab jedoch kein großes Pressecho. Dies könnte unter anderem daher kommen, dass der Grundsatz der Forschung quantitative und messbare Ergebnisse waren und keine großen Ansprüche zur „Intelligenz“ gestellt wurden. Darauf folgte die Ära des tiefen Lernens, also neuronale Netzwerke mit „vielen“ Schichten, welches ein große Aufmerksamkeit von der Presse und Politik bekommt.

1.1.1 Das Perzeptron

Das von Frank Rosenblatt entwickelte *Perzeptron* war das erste Modell, welches durch das menschliche Gehirn motiviert war, ein künstliches neuronales Netzwerk. Dabei werden viele kleine und einfache Einheiten (Neuronen) zu einem größeren Modell verbunden und das Lernen findet durch Anpassung der Verbindungsstärken (Synapsen) und -gewichten statt. Eine Darstellung eines solchen Perzeptrons ist in Abbildung 1.2 gegeben. Dabei werden die Eingaben in der Neuronenschicht gewichtet und die Neuronen *feuern*. Diese Ausgabe wird anschließend an das Ausgabeneuron weitergegeben, welches die *Aktivierungen* akkumuliert und, sofern der akkumulierte Wert über einen gewissen Schwellenwert liegt, feuert. So kann eine binäre Klassifikation durchgeführt werden.

Zum trainieren eines Perzeptrons werden bekannte Daten verwendet, in das Perzeptron eingegeben und die vorhergesagten Ergebnisse mit den echten verglichen. War die Vorhersage korrekt, wird nicht geändert. War die Vorhersage jedoch falsch, so werden die Verbindungsstärken so geändert, dass das Richtige vorhergesagt wird. Dies wird so lange wiederholt, bis keine Fehler mehr gemacht werden.

Mit dem Wechsel zu tiefem Lernen werden diese einschichtigen nicht mehr verwendet, sondern es werden mehrere Neuronenschichten verwendet, die aufeinander Aufbauen (in Abbildung 1.3 in ein solches Netzwerk gezeigt). Neben der Nachteile des höheren Speicherverbrauchs, mehr benötigter Rechenkraft und der Anforderung an mehr Daten haben solche Modelle den großen Vorteil, dass sie eine deutlich höhere Vorhersagekraft besitzen.

1.2 KI Heute

Heute gibt es vier entscheidende Unterschiede zu vergangenen KI-Systemen:

1. Die Modelle sind größer.
    - Früher wurden neuronale Netzwerke mit ein bis drei Schichten und hunderte bis tausende Neuronen verwendet.

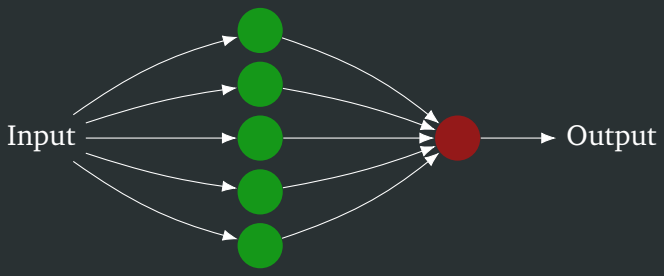

Abbildung 1.2: Darstellung eines einschichtigen Perzeptrons.

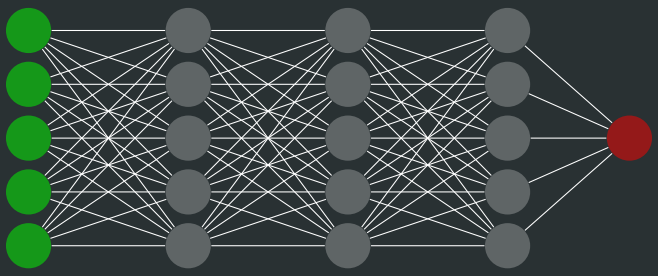

Abbildung 1.3: Darstellung eines mehrschichtigen Perzeptrons.

- Heutige Modelle haben hunderte Schichten und hunderttausende Neuronen.

2. Es sind mehr Daten verfügbar.
   - Früher waren tausende Bilder, hunderte Stunden Audiomaterial und hunderttausende Wörter verfügbar.
   - Heute sind es Milliarden Bilder, Milliarden Stunden Audiomaterial und hunderte Milliarden Wörter.
   - Diese Daten werden dabei in vielen großen Firmen gesammelt, bspw. hat YouTube mehr als zehn Milliarden Videos, Alibaba tätigt mehr als zwölf Milliarden Verkäufe pro Jahr, Facebook-Nutzer laden hunderte Milliarden Bilder pro Jahr hoch und Google kennt mehr als hundert Billionen Webseiten.
3. Heute Computer sind Leistungsfähiger.
   - Früher konnte eine CPU ca. eine Millionen Operationen pro Sekunde ausführen und es gab keine GPUs.
   - Heutige CPUs können mehr als eine Billionen Operationen pro Sekunde und heutige GPUs können mehr als zehn Billionen Operationen pro Sekunde ausführen.
4. Die Systeme funktionieren und lösen viele Aufgaben.

Dabei ist der Hauptmotor der künstlichen Intelligenz aktuell das maschinelle Lernen.

1.3 Was ist Maschinelles Lernen?

Maschinelles Lernen ist die Automatisierung von Automatisierung, es werden Teile des Computers so programmiert, dass sie sich anschließend selbstständig „programmieren“. Das ist nötig, da das Schreiben von Software oftmals der Flaschenhals in der Entwicklung ist, da die Daten so schnell mehr werden. Es ist also klug, die

(a) Traditionelle Programmierung

(b) Maschinelles Lernen

Abbildung 1.4: Traditionelle Programmierung (links) im Vergleich zu maschinellem Lernen (rechts).

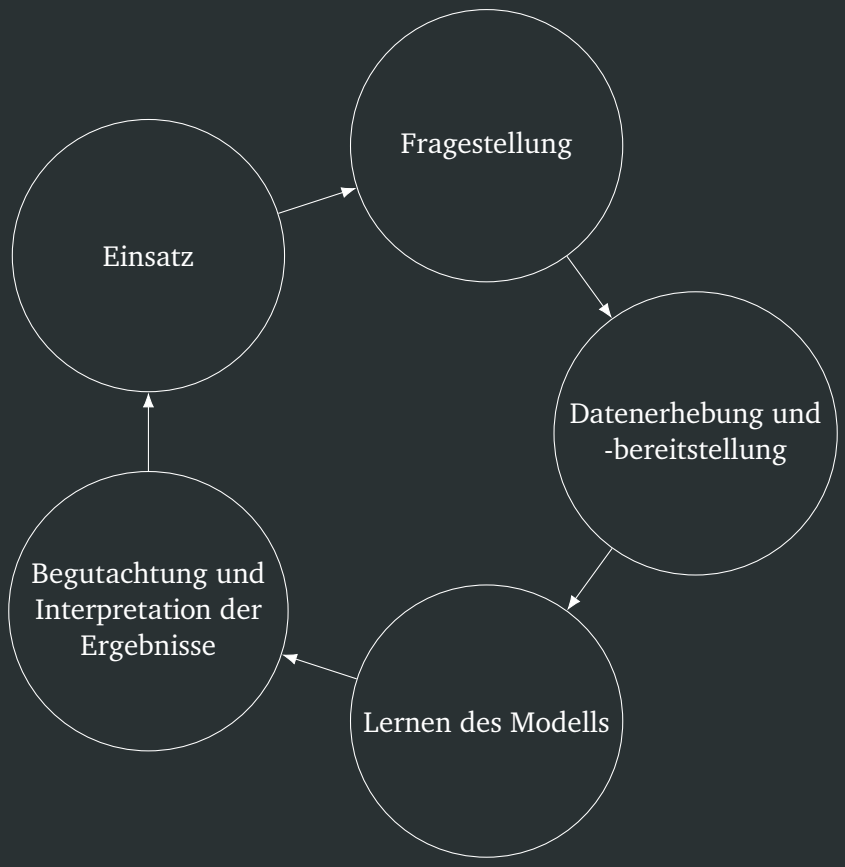

Abbildung 1.5: Kreislauf der Erstellung von ML-Komponenten.

Daten zu nutzen, um die Software selbst zu erstellen. Im Gegensatz zur traditionellen Programmierung werden also keine Ausgaben durch ein Programm erstellt, sondern es wird ein Programm aus Ausgaben erstellen (siehe Abbildung 1.4). Die Entwicklung von ML-Komponenten ist dabei ein Kreislauf, dargestellt in Abbildung 1.5.

Anwendungsgebiete von maschinellem Lernen sind beispielsweise Websuche, Computational Biologie/Cognitive Science/Social Science/..., Finanzwelt, E-Commerce, Robotik, Debugging, Industrie 4.0 und viele mehr.

1.3.1 Kurzgefasst

Im Bereich des maschinellen Lernens gibt es viele tausende Algorithmen und hunderte neue Algorithmen pro Jahr. Dabei adressiert jeder Algorithmus die folgenden drei Fragestellungen:

1. Wie wird das Modell *repräsentiert*?
   Entscheidungsbäume, Regeln/logische Programme, Instanzen, Probabilistische Graphische Modelle, Neuronale Netzwerke, Stützvektormaschinen, Ensembles, ...

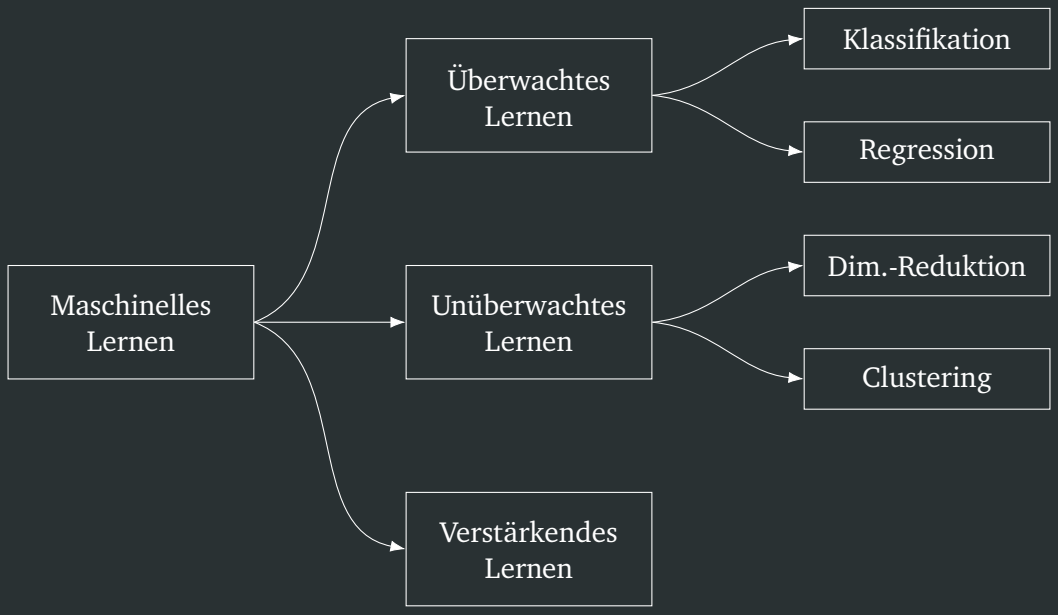

Abbildung 1.6: Arten des maschinellen Lernens.

2. Wie wird das Modell *optimiert*?
   Kombinatorische Optimierung (z. B. Greedy-Suche), Konvexe und nichtlineare Optimierung (z. B. Gradientenabstieg), Optimierung unter Randbedingungen (z. B. lineare Programmierung), ...

3. Wie wird das Modell *evaluiert* und *beurteilt*?
   Korrektklassifikationsrate (Accuracy), Genauigkeit (Precision) und Trefferquote (Recall), Quadrierter Fehler, Likelihood, A-Posteriori Wahrscheinlichkeit, Kosten/Nutzen, Margin, Entropie, KL-Divergenz, ...

Dabei gibt es vier große Kategorien des maschinellen Lernens:

Überwacht (Induktiv) Die Trainingsdaten erhalten neben Eingaben auch gewünschten Ausgaben.
Englisch: *Supervised Learning*

Unüberwacht Die Trainingsdaten erhalten nur Eingaben und *keine* Ausgaben.
Englisch: *Unsupervised Learning*

Teilweise Überwacht Die Trainingsdaten *einige* gewünschte Ausgaben, aber nicht alle.
Englisch: *Semi-supervised Learning*

Verstärkend Der Algorithmus wird belohnt nachdem er eine Reihe an Aktionen ausgeführt hat und die Belohnung wird maximiert.
Englisch: *Reinforcement Learning*

Die Bereiche des überwachten und unüberwachten Lernens lassen sich dann noch weiter einordnen, was in Abbildung 1.6 gezeigt ist.

2 Grundlagen

Dieses Kapitel behandelt einige Grundlagen, die für das maschinelle Lernen benötigt werden.

2.1 CRISP-DM: Verlaufsmodell des Data Mining

Die Schritte des CRISP-DM (Cross-Industry Standard Process for Data Mining), dem Verlaufsmodell des Data Mining, sind:

Problem Verstehen Analyseziele, Situationsbewertung, Datenanalyseziele, Projektplan

Daten Verstehen Sammeln, Beschreiben, Untersuchen, Qualität von Rohdaten

Daten Aufbereiten Ein- und Ausschluss, Bereinigung, Transformation von Variablen

Modellierung Methoden- und Testdesignwahl, Schätzung, Modellqualität

Evaluierung Modell akzeptieren, Prozess überprüfen, Nächste Schritte festlegen

Nachbereitung Anwendungs- und Wartungsplan, Präsentation, Bericht

Diese Schritte sind grafisch in Abbildung 2.1 dargestellt.

2.2 Klassifikation und Regression

Bei überwachtem maschinellen Lernen wird jeder Beobachtung $\boldsymbol{x}$ ein Label y zugeordnet, d. h. die Datenpunkte sind gegeben als $(\boldsymbol{x}, y) \in X \times Y$, wobei X und Y die Mengen aller möglichen Beobachtungen/Labels sind. Im Teilbereich der *Klassifikation* sind die Labels diskret, d. h. es wird eine *qualitative* Beschreibung gesucht. Im Gegensatz dazu steht die *Regression*, bei der die Labels kontinuierlich sind und eine *quantitative* Beschreibung gesucht wird.

Das Ziel ist immer eine *wahre Funktion* $f : X \to Y$ zu finden, die für alle Eingaben $\boldsymbol{x} \in X$ und Labels $y \in Y$ den korrekten Wert liefert, d. h. $f(\boldsymbol{x}) = y$. Das Problem ist im Allgemeinen, dass nur eine Teilmenge aller Beobachtungen gegeben ist, die *Trainingsdaten*. Auf Basis dieser Trainingsdaten wird nun eine Annäherung $\hat{f}$ an die wahre Funktion f gesucht, welche als *Modell* bezeichnet wird. Wurde ein Modell gefunden, so liefert dieses über

$$\hat{y} = \hat{f}(\boldsymbol{x})$$

eine *Vorhersage* $\hat{y} \in Y$ für ein Datum $\boldsymbol{x} \in X$.

Dabei kann zur Klassifikation sowohl ein spezifisches als auch ein Regressionsmodell genutzt werden. Es können zum Beispiel alle Werte oberhalb eines Schwellenwertes θ der einen und alle anderen Werte der anderen Klasse (im Fall von binären Klassifikationsproblemen) zugeordnet werden:

$$\hat{y} = \begin{cases} +1 & \text{falls } \hat{f}(\boldsymbol{x}) \geq \theta \\ -1 & \text{sonst} \end{cases}$$

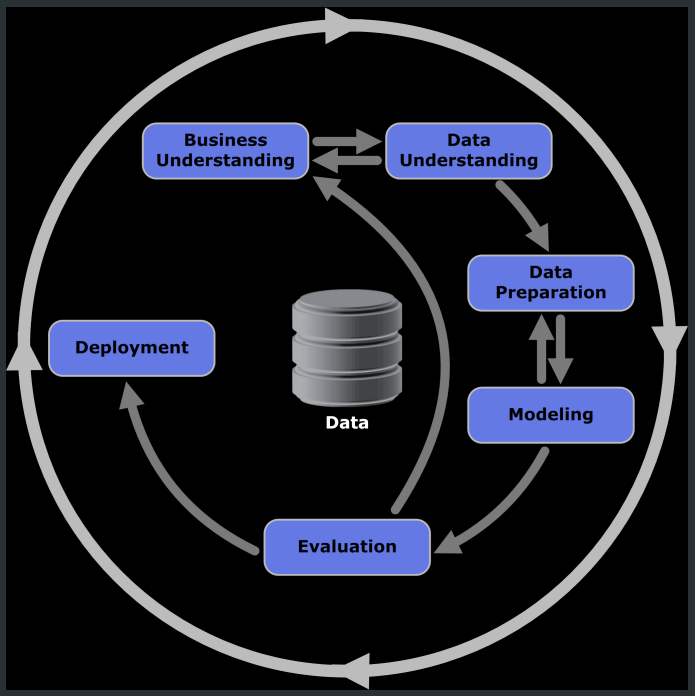

Abbildung 2.1: Grafische Übersicht über das CRISP-DM.
Autor: Kenneth Jensen
Lizenz: CC BY-SA 3.0
Quelle: Wikipedia Commons, Datei CRISP-DM_Process_Diagram.png

Diese Art der Zuordnung ist nur eine Möglichkeit, ein Regressionsmodell als Klassifikationsmodell zu nutzen. Eine weitere ist z. B. die logistische Regression, siehe Abschnitt 4.4.

2.3 Statistik

In diesem Abschnitt werden benötigte grundlegenden Begriffe der Statistik eingeführt.

2.3.1 Erwartungswert, Varianz und Standardabweichung

Für eine diskrete Zufallsvariable X mit Werten x_i ist der *Erwartungswert* gegeben durch

$$\mathbb{E}[X] = \sum_i x_i P(X = x_i).$$

Für eine kontinuierliche Zufallsvariable X mit der Wahrscheinlichkeitsdichte p wird die Summe durch ein Integral ersetzt:

$$\mathbb{E}[X] = \int_{-\infty}^{\infty} x f(x)\,\mathrm{d}x$$

Oftmals werden dabei die Integrationsgrenzen weggelassen.

Eine wichtige Eigenschaft des Erwartungswerts ist, dass dieser linear ist, d. h. es gilt

$$\mathbb{E}[aX + bY] = a\mathbb{E}[X] + b\mathbb{E}[Y]$$

für zwei Zufallsvariablen X, Y und skalare a, b. Sind die Zufallsvariablen X, Y stochastisch unabhängig, so gilt

$$\mathbb{E}[XY] = \mathbb{E}[X]\mathbb{E}[Y]$$

Die *Varianz* einer Zufallsvariablen ist gegeben durch die mittlere quadratische Abweichung vom Mittelwert,

$$\mathrm{Var}[X] = \mathbb{E}\big[(X - \mathbb{E}[X])^2\big].$$

Sie wird oft auch mit σ^2 bezeichnet. Die *Standardabweichung* einer Zufallsvariable ist dann $\sigma = \sqrt{\mathrm{Var}[X]}$. Nach dem *Verschiebungssatz* gilt, wie einfach herzuleiten ist:

$$\mathrm{Var}[X] = \mathbb{E}\big[(X - \mathbb{E}[X])^2\big] = \mathbb{E}\big[X^2\big] - \big(\mathbb{E}[X]\big)^2.$$

2.3.2 Bias

Der *Bias* eines Schätzers $\hat{y}$ für eine Zufallsvariable Y bezeichnet die mittlere Verzerrung

$$\mathrm{Bias}[\hat{y}] = \mathbb{E}[Y - \hat{y}] = \mathbb{E}[Y] - \hat{y}$$

des Schätzers. Ein Schätzer mit einem Bias von Null wird als *Erwartungstreu* oder *Unbiased* bezeichnet.

Oftmals kann bei einem Schätzer nur entweder die Varianz oder der Bias reduziert werden, was als *Bias-Varianz Trade-Off* bezeichnet wird.

2.3.3 Normalverteilung

Die *Normalverteilung* (auch *Gauß-Verteilung*) ist eine der wichtigsten Wahrscheinlichkeitsverteilungen in der Statistik. Dabei heißt eine Zufallsvariable X *normalverteilt* mit Mittelwert μ und Varianz σ^2, wenn sie die Wahrscheinlichkeitsdichte

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}\right\}$$

besitzt.

2.3.4 Bedingte Wahrscheinlichkeiten

Bedingte Wahrscheinlichkeiten, geschrieben $P(X = x \,|\, Y = y)$, beschreiben die Wahrscheinlichkeit einer Zufallsvariable X den Wert x anzunehmen, wenn eine andere Zufallsvariable Y den Wert y hat. Zufallsvariablen X, Y heißen *stochastisch unabhängig*, wenn $P(X = x \,|\, Y = y) = P(X = x)$ gilt. Im Allgemeinen gilt

$$P(X \,|\, Y = y) = \frac{P(X = x, Y = y)}{P(Y = y)},$$

wobei $P(X = x, Y = y)$ die *Verbundwahrscheinlichkeit* ist.

2.3.5 Bayes-Statistik

Eine der wichtigsten Formeln in der Statistik in die Bayes-Formel

$$p(x \,|\, y) = \frac{p(y \,|\, x)\, p(x)}{p(y)},$$

die den Zusammenhang zwischen der Likelihood $p(y \mid x)$, der A-Priori Wahrscheinlichkeit $p(x)$ und der A-Posteriori Wahrscheinlichkeit $p(x \mid y)$ beschreibt. Der Faktor $p(y)$ dient dabei nur der Normalisierung, weshalb die Bayes-Formel oft auch als

$$p(x \mid y) \propto p(y \mid x)\, p(x)$$

geschrieben wird.

2.3.6 Konfidenzintervalle

Ein *Konfidenzintervall* ist ein Intervall $[\ell, u]$, welches aus einer gegebenen Irrtumswahrscheinlichkeit α erzeugt wird. Es gibt den Bereich an, in dem der Wert der Zufallsvariable mit Wahrscheinlichkeit $1 - \alpha$ liegt:

$$P(\ell \leq X \leq u) = 1 - \alpha$$

Eine häufige Wahl ist die Bildung eines Konfidenzintervalls für den Erwartungswert. So können Aussagen wie „Mit $99\,\%$ Wahrscheinlichkeit liegt der Mittelwert im Intervall $[\ell, u]$." getätigt werden (hier mit $\alpha = 0.01$).

Konfidenzintervalle sind ein mächtiges Werkzeug in der Bewertung von Verfahren, was in Kapitel 6 weiter behandelt wird.

2.3.7 Entropie

Die Entropie einer Wahrscheinlichkeitsverteilung $P(X)$ gibt an, wie viel Informationsgehalt in dieser steckt. Für eine diskrete Zufallsvariable X mit den möglichen Werten $\mathcal{X}$ wird die Entropie wie folgt berechnet:

$$H(X) = -\sum_{x \in \mathcal{X}} P(x)\, \log P(x)$$

Für eine kontinuierliche Zufallsvariable X mit Wahrscheinlichkeitsdichte $p(X)$ gilt

$$H(X) = -\int p(x)\, \log p(x)\, \mathrm{d}x,$$

wobei wie vorher beschrieben die Integrationsgrenzen zur Übersichtlichkeit weggelassen werden. Wird der Logarithmus zur Basis 2 verwendet, so kann die Entropie einer Verteilung auch als die Anzahl Bits angesehen werden, die mindestens zur Übertragung benötigt werden.

Beispiel Es werden Nachrichten X mit den Werten a, b, c und d versendet. Eine Aufzeichnung des Netzwerkverlaufs stellt fest, dass die Nachrichten mit den folgenden Häufigkeiten gesendet werden:

a	b	c	d
25	7	14	54

Bei einer Gesamtanzahl von $25 + 7 + 14 + 54 = 100$ ergeben sich also die folgenden relativen Häufigkeiten, bzw. die folgende empirische Verteilung:

$$\begin{aligned} P(X = a) &= 25\% \quad P(X = b) = 7\% \\ P(X = c) &= 14\% \quad P(X = d) = 54\% \end{aligned}$$

Der Informationsgehalt $-\log_2 P(X)$ je Symbol ist:

$$\begin{aligned} -\log_2 P(X = a) &= 2 & -\log_2 P(X = b) &= 3.8365 \\ -\log_2 P(X = c) &= 2.8365 & -\log_2 P(X = d) &= 0.888969 \end{aligned} \tag{2.1}$$

Daraus ergibt sich die folgende Gesamtentropie:

$$\begin{aligned} H(X) = -\big(\quad & P(X=a)\, \log P(X=a) \\ & + P(X=b)\, \log P(X=b) \\ & + P(X=c)\, \log P(X=c) \\ & + P(X=d)\, \log P(X=d) \quad \big) \\ &= 0.25 \cdot 2 + 0.007 \cdot 3.8365 + 0.14 \cdot 2.8365 + 0.54 \cdot 0.888969 \\ &= 1.40401 \end{aligned}$$

Es werden also mindestens 2 Bits benötigt, um die Nachricht zu kodieren.

An dem Informationsgehalt (2.1) ist zu sehen, dass as Symbol d die wenigsten Informationen überträgt, da es am häufigsten vorkommt. Dies kann Interpretiert werden als „Überraschungseffekt“ beim Empfänger: Kommt ein d an, ist dies nicht überraschen, da dies in $54\,\%$ der Fälle passiert. Kommt jedoch ein b an, ist dies sehr ungewöhnlich.

3 k-Nächste Nachbarn (kNN)

Im Allgemeinen können Verfahren des maschinellen Lernens in *globale* und *lokale* Modelle unterteilt werden: Lokale Modelle Klassifizieren einen Datenpunkt nur Anhand seiner Umgebung, globale Modelle hingegen finden ein global Gültiges Entscheidungskriterium, beispielsweise eine trennende Hyperebene. Das verfahren der *k-Nächsten Nachbarn* (kNN) gehört zu den lokalen Modelle. Bei diesem Ansatz wird ein Beispiel anhand der k nächsten Nachbarn (nach irgendeiner Ähnlichkeitsmetrik) klassifiziert: Ein Datenpunkt gehört einer bestimmten Klasse an, wenn der Großteil der umliegenden Datenpunkte auch dieser Klasse angehören. Leicht formalisiert ergibt sich folgendes Vorgehen (wobei $f(\cdot)$ die tatsächliche Kategorie beschreibt):

1. Berechne den Abstand zwischen dem Datenpunkt $\boldsymbol{x}_*$ und jedem Trainingsdatenpunkt.
2. Wähle die k nächsten Nachbarn $\boldsymbol{n}_1, \cdots, \boldsymbol{n}_k$ bezüglich einem Ähnlichkeitsmaß $dist(\boldsymbol{x}, \boldsymbol{y})$.
3. Berechne die Vorhersage $A\big(\boldsymbol{x}_*;\, f(\boldsymbol{n}_1), \cdots, f(\boldsymbol{n}_k)\big)$.

Hier werden sofort einige Schwierigkeiten ersichtlich:

- Welches Ähnlichkeitsmaß $dist(\cdot, \cdot)$ sollte verwendet werden?
- Wie viele Nachbarn k sollten verwendet werden?
- Was passiert, wenn die Werte der Nachbarn nicht überein stimmen oder es ein Unentschieden gibt?
- Wie kann die Suche effizient gestaltet werden?

3.1 Ähnlichkeitsmaße

Wie bereits beschrieben ist ein Problem die Auswahl eines Ähnlichkeitsmaßes. Die zwei wichtigsten Eigenschaften des Ähnlichkeitsmaßes sind, dass dieses kleiner werden sollte, wenn zwei Datenpunkte sich ähnlicher sind und genau dann Null ist, wenn zwei Datenpunkte identisch sind. Eine Möglichkeit ist beispielsweise der euklidische Abstand

$$dist(\boldsymbol{x}, \boldsymbol{y}) = \|\boldsymbol{x} - \boldsymbol{y}\|_2 = \sqrt{\textstyle\sum_{i=1}^{n} (x_i - y_i)^2}$$

oder der Kosinus-Abstand

$$dist(\boldsymbol{x}, \boldsymbol{y}) = \cos(\boldsymbol{x}, \boldsymbol{y}) = \frac{\langle \boldsymbol{x} | \boldsymbol{y} \rangle}{|\boldsymbol{x}|\,|\boldsymbol{y}|} = \frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} x_i^2}\,\sqrt{\sum_{i=1}^{n} y_i^2}},$$

welcher Invariant gegenüber Skalierungen der Vektoren ist. Andere Ähnlichkeitsmaße für binäre (0/1) Daten sind z. B.:

- Matching Koeffizient:

$$|\boldsymbol{x} \cap \boldsymbol{y}|$$

- Dice Koeffizient:

$$\frac{2|\boldsymbol{x} \cap \boldsymbol{y}|}{|\boldsymbol{x}| + |\boldsymbol{y}|}$$

- Jaccard Koeffizient.

$$\frac{|\boldsymbol{x} \cap \boldsymbol{y}|}{|\boldsymbol{x} \cup \boldsymbol{y}|}$$

- Overlap Koeffizient:

$$\frac{|\boldsymbol{x} \cap \boldsymbol{y}|}{\min\{|\boldsymbol{x}|, |\boldsymbol{y}|\}}$$

- Kosinus:

$$\frac{|\boldsymbol{x} \cap \boldsymbol{y}|}{\sqrt{|\boldsymbol{x}|}\,\sqrt{|\boldsymbol{y}|}}$$

Dabei beschreibt $\cap$ eine komponentenweise Und-Aggregation und $\cup$ eine Oder-Aggregation:

$$\begin{bmatrix}0\\0\\1\\1\end{bmatrix} \cap \begin{bmatrix}0\\1\\0\\1\end{bmatrix} = \begin{bmatrix}0\\0\\0\\1\end{bmatrix} \qquad \begin{bmatrix}0\\0\\1\\1\end{bmatrix} \cup \begin{bmatrix}0\\1\\0\\1\end{bmatrix} = \begin{bmatrix}0\\1\\1\\1\end{bmatrix}$$

Trotz dieser vielen Ähnlichkeitsmaße ist es im Allgemeinen aber sehr schwer zu entscheiden, ob sich zwei Datenpunkte (z. B. Bilder) ähnlich sind – die Bedeutung von „ähnlich“ scheint an dieser Stelle eher ein philosophisches Problem zu sein, im maschinellen Lernen wird der Begriff hingegen eher pragmatisch genutzt.

3.2 Auswahlfunktion

Wie auch für das Ähnlichkeitsmaß gibt es für die Auswahlfunktion verschiedene Möglichkeiten. Bei der Klassifikation wird üblicherweise eine einfache Mehrheitsentscheidung verwendet, bei der Regression gibt es hingegen mehrere vernünftige Ansätze. Die einfachste ist eine Mittlung

$$A\big(\boldsymbol{x}_*;\, f(\boldsymbol{n}_1), \cdots, f(\boldsymbol{n}_k)\big) = \frac{1}{k}\sum_{i=1}^{k} f(\boldsymbol{n}_i),$$

wobei $f(\boldsymbol{n}_i)$ den Funktionswert des Trainingsbeispiels $\boldsymbol{n}_i$ darstellt. Andere Möglichkeiten sind gewichtete Varianten

$$A\big(\boldsymbol{x}_*;\, f(\boldsymbol{n}_1), \cdots, f(\boldsymbol{n}_k)\big) = \sum_{i=1}^{k} sim(\boldsymbol{n}_i, \boldsymbol{x}_*)\, f(\boldsymbol{n}_i)$$

oder

$$A\big(\boldsymbol{x}_*;\, f(\boldsymbol{n}_1), \cdots, f(\boldsymbol{n}_k)\big) = \frac{1}{\sum_{i=1}^{k} w_i}\sum_{i=1}^{k} w_i\, f(\boldsymbol{n}_i)$$

mit $w_i = dist^{-2}(\boldsymbol{n}_i, \boldsymbol{x}_*)$, wobei $sim(\boldsymbol{n}_i, \boldsymbol{x}_*)$ ein Ähnlichkeitsmaß ist.

3.3 Überanpassung

Die Wahl eines „guten“ k ist im Allgemeinen schwierig, aber auch sehr wichtig. Der Wert $k = 1$ liefert beispielsweise eine Perfekte vorhersage auf den Trainingsdaten, da der nächste Datenpunkt immer der Abfragedatenpunkt ist, es wird also immer der selbe Wert zugeordnet. Werden jedoch andere Daten verwendet, ist der Fehler vermutlich sehr groß! Die ist bekannt als das Problem der *Überanpassung* (Englisch: *Overfitting*) und stellt eine große Herausforderung im maschinellen Lernen dar. Dieses Problem wird neben weiteren Möglichkeiten zur Evaluation eines Modells im Kapitel 5 behandelt.

3.4 Asymptotische Ergebnisse und Fluch der hohen Dimension

Konvergiert k/N gegen 0 während k und N (die Anzahl Trainingsdatenpunkte) gegen unendlich laufen, d. h.

$$\lim_{\substack{k \to \infty \\ N \to \infty}} \frac{k}{N} = 0,$$

dann konvergiert die Vorhersage von kNN gegen die zu erwartende Vorhersage (Hastie et al., 2001). Das bedeutet für unendlich viele Datenpunkte und unendliche viele vergleiche ist kNN perfekt. Dies birgt aber ein Problem: Die Dichte der Beispiele ist reziprok proportional zu N^n, wobei n die Dimension der Datenpunkte ist. Daher steigt die Menge benötigter Daten *exponentiell* mit der Dimension des Zustandsraums. Dies ist bekannt als der *Fluch der hohen Dimension*, der *Curse of Dimensionality*. Diesem Fluch unterliegen sehr viel (wenn nicht gar alle) Modelle des maschinellen Lernens!

4 Lineare Modelle und Funktionsapproximation

Die Grundlage des maschinellen Lernens ist im Allgemeinen die Approximation einer „wahren“ Funktion durch ein (einfacheres) Modell. Dabei wird das Modell in Bezug auf ein *Gütekriterium* optimiert. Dieses Gütekriterium kann beispielsweise ein Fehler, z. B. der quadratische Fehler, oder eine Wahrscheinlichkeit, z. B. die Likelihood, sein. Diese Gütekriterien werden in Abschnitt 4.3 vorgestellt.

Dieses Kapitel behandelt grundlegende Begriffe zur Funktionsapproximation und stellt als eines der ersten Modelle lineare Modelle vor, die sehr einfach in geschlossener Form optimiert werden können. Damit stellen sie den Grundbaustein zu mächtigeren Modellen wie Neuronalen Netzwerken und Gauß-Prozessen dar. Neuronale Netzwerke werden in Kapitel 10 behandelt, Gauß-Prozesse werden in dieser Zusammenfassung nicht behandelt.

4.1 Lineare Modelle

Bei linearen Modellen

$$\hat{y} = \hat{f}(\boldsymbol{x}) = \sum_{i=1}^{k} \beta_i h_i(\boldsymbol{x}) \tag{4.1}$$

hängt der Ausgabewert nur in linearer Form von den Parametern $\boldsymbol{\beta} \coloneqq \begin{bmatrix}\beta_1 & \beta_2 & \cdots & \beta_k\end{bmatrix}^T \in \mathbb{R}^k$ und beliebigen *Basisfunktionen* $h_{1:k}(\cdot)$ ab. Wird $h_i(\boldsymbol{x}) = x_i$ mit $k = n$, wobei n die Dimension des Eingaberaums ist, gewählt, so hängen die Ausgabewerte sogar nur linear von den Eingabewerten ab. Oftmals wird ein *Bias* eingeführt, der ausschließlich auf das Ergebnis addiert wird (als Achsenverschiebung). Dies kann z. B. durch eine Basisfunktion $h(\boldsymbol{x}) = 1$ modelliert werden.

Zur Optimierung dieses Modells werden nun optimale Parameter $\boldsymbol{\beta}$ gesucht, die ein bestimmtes Gütekriterium minimieren oder maximieren. Zur einfacheren Darstellung sei im Folgenden $\boldsymbol{h}(\boldsymbol{x})$ der Vektor aller Basisfunktionswerte (*Features*) für eine Eingabe $\boldsymbol{x}$, d. h. $\boldsymbol{h} : \mathbb{R}^n \to \mathbb{R}^k$. Dadurch vereinfacht sich (4.1) zu

$$\hat{y} = \boldsymbol{h}^T(\boldsymbol{x})\boldsymbol{\beta}.$$

Um die optimalen Parameter $\boldsymbol{\beta}$ zu finden muss nun ein Gütekriterium ausgewählt werden. Oft wird hierbei der quadratische Fehler (die *Sum of Squared Residuals*, RSS)

$$RSS(\boldsymbol{\beta}) = \sum_{i=1}^{N} \left(y_i - \boldsymbol{h}^T(\boldsymbol{x}_i)\boldsymbol{\beta}\right)^2 = (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})^T(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})$$

verwendet, wobei $\left\{(\boldsymbol{x}_i, y_i)\right\}_{i=1,\cdots,N}$ die Trainingsbeispiele sind. Dieses Kriterium hat die schöne Eigenschaft, dass es an jeder Stelle differenzierbar ist (in Bezug auf $\boldsymbol{\beta}$) und ein eindeutiges Minimum besitzt. Im letzten Schritt wurde die Summe mit $\boldsymbol{y} \coloneqq \begin{bmatrix}y_1 & y_2 & \cdots & y_N\end{bmatrix}^T \in \mathbb{R}^N$ und $\boldsymbol{X} \coloneqq \begin{bmatrix}\boldsymbol{h}(\boldsymbol{x}_1) & \boldsymbol{h}(\boldsymbol{x}_2) & \cdots & \boldsymbol{h}(\boldsymbol{x}_N)\end{bmatrix}^T \in \mathbb{R}^{N\times k}$ zusammengefasst, um die folgende Herleitung zu vereinfachen. Zur Minimierung des Fehlers wird nun die Ableitung bezüglich $\boldsymbol{\beta}$ gebildet und Null gesetzt, um die optimalen Parameter $\boldsymbol{\beta}^*$ zu finden:

$$\frac{\partial}{\partial\boldsymbol{\beta}} RSS(\boldsymbol{\beta}) = 2\boldsymbol{X}^T(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}) = 2\boldsymbol{X}^T\boldsymbol{y} - 2\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{\beta} \stackrel{!}{=} \boldsymbol{0} \quad \Longrightarrow \quad \boldsymbol{\beta}^* = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}$$

Die Invertierung der Matrix $\boldsymbol{X}^T\boldsymbol{X}$ ist natürlich nur möglich, wenn diese regulär ist. Ansonsten existiert kein eindeutiges Minimum. Der der Praxis ist dies jedoch meistens der Fall und Falls nicht kann die Matrix durch Addieren kleiner Werte auf der Diagonale regularisiert werden.

Dies Vorgehen hat zwar ein großes Potential und kann durch die Freiheit in den Basisfunktionen $\boldsymbol{h}(\cdot)$ weitreichend eingesetzt werden, jedoch gibt es einige Limitierungen:

- Die Matrix $\boldsymbol{X}^T\boldsymbol{X}$ hat die Dimension $k \times k$, wobei k die Anzahl Basisfunktionen ist (die üblicherweise über der Dimension der Eingabedaten, n, liegt). Die Invertierung dieser Matrix hat die Komplexität $\mathcal{O}(k^3)$, d. h. die benötigte Zeit steigt kubisch mit der Anzahl Basisfunktionen. Auch dies ist ein Auftreten des Fluches der hohen Dimension.
- Bei linearen Basisfunktionen $\boldsymbol{h}(\cdot)$ kann es auftreten, dass die Daten überhaupt nicht linear modellierbar sind. Dann müssen stärkere Basisfunktionen oder ein anderes Verfahren eingesetzt werden.
- Es ist nicht ausreichend, den Fehler zu minimieren. Dies kann zu Überanpassung und instabilen Lösungen führen.

Der letzte Punkt wird im folgenden Abschnitt weiter betrachtet.

4.2 Fehler

Der bisher betrachtete Fehler betrachtet nur die aktuell vorliegenden Trainingsdaten – es wäre jedoch deutlich interessanter, den Fehler über *alle* Daten zu betrachten. Dies ist durch die Betrachtung des Erwartungswertes über die Ein- und Ausgabedaten möglich. Ein gegebener Trainingsdatensatz stellt dann eine Stichprobe dar.

Der erwartete quadratische Fehler eines beliebigen Modells $\hat{f}$, ist

$$EPE(\boldsymbol{X}) = \mathbb{E}\Big[(Y - \hat{f}(\boldsymbol{X}))^2\Big], \tag{4.2}$$

wobei $\boldsymbol{X}$ und Y die Zufallsvariablen der Ein- bzw. Ausgabewerte sind. Nun wird für jeden Eingabewert $\boldsymbol{x}$ als Realisierung von $\boldsymbol{X}$ ein Optimierungproblem formuliert:

$$\hat{f}(\boldsymbol{x}) = \arg\min_y \mathbb{E}\big[(Y - y)^2 \,\big|\, \boldsymbol{X} = \boldsymbol{x}\big]$$

Dabei ist y anschließend die Vorhersage des Modells. Die Lösung dieses Optimierungsproblems ist gegeben durch den Erwartungswert von Y gegeben die Realisierung $\boldsymbol{x}$ von $\boldsymbol{X}$:

$$\hat{f}(\boldsymbol{x}) = \mathbb{E}\big[Y \,\big|\, \boldsymbol{X} = \boldsymbol{x}\big]$$

Dieser Erwartungswert ist jedoch i. A. nicht berechenbar, weshalb der Weg über die Stichproben gemacht wird. Das liegt daran, dass die Wahrscheinlichkeitsverteilungen von $\boldsymbol{X}$ und Y nicht bekannt sind. Während sie bekannt, könnte man sich den gesamten Lernprozess sparen.

4.2.1 Bias und Varianz

Der erwartete quadratische Fehler (4.2) kann in einen Bias- und einen Varianz-Term aufgespalten werden, die eine größere Interpretation liefern. Dafür wird zunächst ein beliebiger Testpunkt $\boldsymbol{x}_0$ (mit Wert y_0) ausgewählt, an dem der Fehler berechnet werden soll. Es wird außerdem davon ausgegangen, dass die Trainingsdaten

eines beliebigen Datensatzes $\mathcal{T}$ verrauscht sind, d. h. es gibt einen Messfehler $\epsilon \sim \mathcal{N}(0, \sigma_\epsilon^2)$ mit Varianz σ_ϵ^2 und Erwartungswert 0. Der erwartete quadratische Fehler teilt sich dann wie folgt auf:

$$EPE(\boldsymbol{x}_0) = \underbrace{\mathbb{E}_Y\Big[(Y - y_0)^2\Big]}_{\text{Rauschen}} + \underbrace{\mathbb{E}_\mathcal{T}\Big[\big(y_0 - \mathbb{E}_\mathcal{T}\big[\hat{f}(\boldsymbol{x}_0)\big]\big)^2\Big]}_{\text{Bias}^2} + \underbrace{\mathbb{E}_\mathcal{T}\Big[\big(\mathbb{E}_\mathcal{T}\big[\hat{f}(\boldsymbol{x}_0)\big] - \hat{f}(\boldsymbol{x}_0)\big)^2\Big]}_{\text{Varianz}}$$

Der Fehler setzt sich also aus dem Rauschen, dem quadratischen Bias sowie der Varianz zusammen. Der Bias ist dabei unabhängig von dem Datensatz und Null bei einem perfektem lernen. Die Varianz hingegen ist nicht abhängig vom wahren Wert und ebenfalls Null bei einem perfektem Lerner. Ein Modell ohne Bias und ohne Varianz ist jedoch im Allgemeinen nicht erreichbar!

Da der Bias quadratisch in den Gesamtfehler einfließt und somit stärker als die Varianz, erklärt, wieso Modelle sehr schnell zu Überanpassung[1] neigen: Die Optimierung des Bias ist „lohnenswerter“, da dieser so stark gewichtet wird. Dem kann dadurch entgegengewirkt werden, indem beispielsweise das Rauschen in den Ursprungsdaten erhöht wird oder indem mehr Daten verwendet werden.

Für ein lineares Modell ist der erwartete Fehler gegeben durch

$$EPE(\boldsymbol{x}_0) = \sigma_\epsilon^2 + \Big[y_0 - \mathbb{E}\big[\hat{f}(x_0)\big]\Big]^2 + \operatorname{Var}\big[\hat{f}(\boldsymbol{x}_0)\big],$$

wobei die Varianz von x_0 abhängt. Im Mittel über alle Trainingseingabewerte $\boldsymbol{x}_i$ hat die Varianz den Wert

$$\frac{1}{N}\sum_{i=1}^{N} \operatorname{Var}\big[\hat{f}(\boldsymbol{x})_i\big] = \frac{k}{N}\sigma_\epsilon^2.$$

Wird der Trainingsfehler über einen Trainingsdatensatz mit N Beispielen gemittelt, ergibt sich der folgende erwartete Fehler:

$$\frac{1}{N}\sum_{i=1}^{N} EPE(\boldsymbol{x}_0) = \frac{1}{N}\sigma_\epsilon^2 + \frac{1}{N}\sum_{i=1}^{N}\Big[y_0 - \mathbb{E}\big[\hat{f}(x_0)\big]\Big]^2 + \frac{k}{N}\sigma_\epsilon^2$$

Der Fehler nimmt also im Allgemeinen mit steigender Trainingsdatensatzgröße ab und mit steigender Dimension (steigendem k) zu.

Da nicht bekannt ist, welches Modell und welche Basisfunktionen gut zu den Daten passt, muss meist eine hohe Anzahl an Basisfunktionen genutzt werden. Durch die kubische Berechnungskomplexität können lineare Modelle dann trotz ihrer Einfachheit schnell langsam werden.

4.3 Gütekriterien

Das Gütekriterium, bzw. das Optimierungsziel[2], ist eine der wichtigsten Modellentscheidungen. Wird das falsche Kriterium optimiert, lernt ein Modell eventuell keine sinnvolle Repräsentation. Dabei gibt es grundlegend zwei Typen von Gütekriterien:

Verlustfunktion Das Gütekriterium beschreibt den Fehler, also den Abstand, des vorhergesagten Ergebnisses im Vergleich zu den tatsächlichen (Trainings-) Daten. Ein Beispiel ist der quadratische Fehler. Eine Verlustfunktion wird immer minimiert.

[1] Bei einem überangepasstem Modell liegt ein niedriger Bias aber eine hohe Varianz vor.
[2] Englisch: *Objective*

Likelihood Es wird die Wahrscheinlichkeit der Parameter oder der Trainingsdaten maximiert. Dies ist z. B. der Ansatz bei einem Maximum Likelihood-Schätzer. Eine Likelihood wird immer maximiert.

Im allgemeinen kann eine Likelihood auch immer als Verlustfunktion gesehen werden, wenn das Vorzeichen geändert wird. Dadurch wird aus einem Maximierungs- ein Minimierungsproblem[3].

4.3.1 Verlustfunktionen

Es gibt sehr viele unterschiedliche Verlustfunktionen, die für unterschiedliche Dinge gut sind. Für die Regression ist der quadratische Fehler

$$\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

üblich, aber auch der absolute Fehler

$$\sum_{i=1}^{N}|y_i - \hat{y}_i|$$

wird gelegentlich verwendet. Dieser hat den zentralen Nachteil, dass er nicht überall Differenzierbar ist.

Zur Klassifikation werden meistens andere Verlustfunktionen eingesetzt, die die diskreten Eigenschaften der Klassifizierung berücksichtigen. Ein Beispiel ist der 0-1-Fehler

$$\sum_{i=1}^{N}\mathbb{1}[y = \hat{y}],$$

wobei $\mathbb{1}[\cdot]$ die Auswahlfunktion ist (sie ist genau dann 1, wenn die Aussage in den eckigen Klammern wahr ist, sonst ist sie 0). Wie bei dem absoluten Fehler besteht auch hier das Problem, dass die Verlustfunktion nicht differenzierbar ist. Eine bessere Verlustfunktion ist in diesem Fall die *Kreuzentropie*

$$-\sum_{i=1}^{N}P(Y = y_i \mid \boldsymbol{X} = \boldsymbol{x}_i)\,\log(P(Y = y_i \mid \boldsymbol{X} = \boldsymbol{x}_i)),$$

wobei dabei ein stochastisches Modell angenommen werden muss, um die Wahrscheinlichkeiten zu erhalten.

4.3.2 Likelihood

Beschreibt das Modell eine Wahrscheinlichkeitsverteilung $P_\theta(Y \mid \boldsymbol{X})$ über die Ausgaben (mit den Parameter θ), so kann statt der Minimierung einer Verlustfunktion auch die Wahrscheinlichkeit der Daten maximiert werden. Die Gesamtwahrscheinlichkeit aller Trainingsdaten ist, unter der Annahme, dass die Trainingsdaten stochastisch unabhängig und identisch verteilt sind, gegeben durch die Faktorisierung

$$P_\theta(y_1, \cdots, y_N \mid \boldsymbol{x}_1, \cdots, \boldsymbol{x}_N) = \prod_{i=1}^{N}P_\theta(y_i \mid \boldsymbol{x}_i).$$

[3] Dies ist beispielsweise notwendig wenn zur Implementierung eines Modells eine Bibliothek verwendet wird, die ausschließlich Minimierungsprobleme unterstützt (z. B. SciPy und PyTorch).

An dieser Stelle wird zur kürze die genaue Zuweisung zur Zufallsvariablen (also $Y = y_i$ und $\boldsymbol{X} = \boldsymbol{x}_i$) weggelassen. Da dieses Produkt schwer zu optimieren ist, wird üblicherweise die Log-Likelihood

$$L(\theta) = \log P_\theta(y_1, \cdots, y_N \mid \boldsymbol{x}_1, \cdots, \boldsymbol{x}_N) = \sum_{i=1}^{N} \log P_\theta(y_i \mid \boldsymbol{x}_i). \tag{4.3}$$

verwendet, wodurch das Produkt zu einer Summe wird. Das Ziel ist nun jene Parameter θ^* zu finden, die diese Summe maximal werden lassen. Dafür muss eine Verteilung angenommen werden, da die wahre Verteilung nicht bekannt ist.

Es kann beispielsweise eine Normalverteilung $p(Y \mid \boldsymbol{X}) = \mathcal{N}\big(f_\theta(\boldsymbol{X}), \sigma^2\big)$ angenommen werden, die nur eine Varianz hinzufügt. Der Ausgabe des Modells $\hat{f}(\boldsymbol{X})$ wird somit ein Rauschen $\epsilon \sim \mathcal{N}(0, \sigma^2)$ hinzugefügt, wodurch die Ausgabe eine Wahrscheinlichkeitsverteilung darstellt. Einsetzen in die Log-Likelihood (4.3) liefert nun

$$\begin{aligned}
L(\theta) &= \sum_{i=1}^{N} \log\left(\frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{1}{2}\frac{\big(y_i - \hat{f}_\theta(\boldsymbol{x}_i)\big)^2}{\sigma^2}\right\}\right) \\
&= -\sum_{i=1}^{N} \log\Big(\sqrt{2\pi\sigma^2}\Big) + \frac{1}{2\sigma^2}\big(y_i - \hat{f}_\theta(\boldsymbol{x}_i)\big)^2 = \underbrace{-N \log\Big(\sqrt{2\pi\sigma^2}\Big)}_{C_2 :=} - \underbrace{\frac{1}{2\sigma^2}}_{C_1 :=} \sum_{i=1}^{N} \big(y_i - \hat{f}_\theta(\boldsymbol{x}_i)\big)^2 \\
&= C_2 - C_1 \sum_{i=1}^{N} \big(y_i - \hat{f}_\theta(\boldsymbol{x}_i)\big)^2 = C_2 - C_1\, RSS(\theta),
\end{aligned}$$

wobei die Konstanten C_1 und C_2 unabhängig von den Parametern θ des Modells sind. Die Minimierung des quadratischen Fehlers ist also äquivalent zur Maximierung der Likelihood unter Annahme einer Normalverteilung mit konstanter Varianz!

4.4 Logistische Regression

Im allgemeinen kann jedes Regressionsmodell auch zur Klassifikation verwendet werden, indem eine Entscheidungsfunktion an das Ende gehängt wird. Ein Beispiel ist die logistische Funktion (auch *Sigmoid*)

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

welche durch die Annahme hergeleitet werden kann, dass der Quotient der Klassenwahrscheinlichkeiten als log-lineares Modell berechnet werden kann:

$$y = \log\left(\frac{P(Y = +1)}{P(Y = -1)}\right) = \log\left(\frac{P(Y = +1)}{1 - P(Y = +1)}\right) \quad \Longleftrightarrow \quad P(Y = +1) = \frac{1}{1 + e^{-y}}$$

Das Ergebnis gibt dann also die Wahrscheinlichkeit an, dass der Eingabedatenpunkt zur Klasse 1 gehört. Eine andere Möglichkeit zur binären Klassifikation basiert auf einem Schwellenwert θ. Liegt die Ausgabe des Regressionsmodells über diesem Schwellenwert, wird dem Datenpunkt die Klasse 1 und sonst die Klasse -1 zugeordnet:

$$y = \begin{cases} +1 & \text{falls } \hat{f}(\boldsymbol{x}) \geq \theta \\ -1 & \text{sonst} \end{cases}$$

Dabei muss nun ein Wert für θ festgelegt werden, was z. B. basierend auf den Kosten für ein falsch-positives, bzw. falsch-negatives, Ergebnis geschehen kann. Wird θ erhöht, treten mehr falsch-negative Ergebnisse auf, wird θ verringert, mehr falsch-positive.

5 Modellselektion und Evaluation

Eine Problematik (wenn nicht die größte) ist, dass immer nur eine endliche Menge an Beispielen vorhanden und die wahre Verteilung der Beispiele nicht bekannt ist. Daher ist es wichtig, einen geeigneten Weg zur Beurteilung der Lernergebnisse zu finden. Eine Bewertung auf den Trainingsdaten selbst ist nicht ausreichend, da das Modell dann „einfach“ alle Beispiele auswendig lernen könnte und so immer perfekte Vorhersagen trifft, auf echten Daten aber versagt. In diesem Fall wurde dann kein logischer Zusammenhang zwischen Ein- und Ausgabe gelernt. Ein Beispiel hierfür ist die Verwendung in kNN mit $k = 1$. Dieses Auswendig lernen wird auch *Überanpassung* genannt und äußert sich durch einen niedrigen Bias und eine hohe Varianz des Modells. Dies ist eine Äußerung des *Bias-Varianz Trade-Off* von Schätzern (siehe Unterabschnitt 2.3.2). Die Evaluation von Modellen ist auch bei der Selektion von Modellen wichtig: Wie wird aus Modellen das beste ausgewählt? Dieses Kapitel wird sich mit diesen beiden Themen, Modellselektion und Evaluation, beschäftigen.

5.1 Aufteilung in Trainings- und Testmenge: Kreuzvalidierung

Zur Bewertung eines Modells ist es zunächst hilfreich, die Daten in eine Trainings- und eine Testmenge aufzuteilen. Die Daten in der Trainingsmenge werden dem Algorithmus zum Trainieren übergeben, die restlichen Daten (die Testmenge) werden zum Vergleich der Vorhersage der Modelle verwendet. Im Allgemeinen wird allerdings nicht nur eine Aufteilung verwendet, sondern die Daten werden mehrmals geteilt, um eine möglichst hohe Varianz in den verwendeten Daten zu erhalten. Dadurch wird vermieden zufällig ausschließlich auf Ausnahmen gelernt zu haben und anschließend auf den „normalen“ Fällen zu testen.

Aus Zeitgründen ist es allerdings meist zu aufwendig, auf allen möglichen Kombinationen der Daten zu trainieren, bzw. zu evaluieren. Daher wird häufig die *Kreuzvalidierung* eingesetzt, bei der die Lernmenge zufällig in n Mengen aufgeteilt wird. Der Algorithmus wird anschließend auf $n-1$ dieser Mengen trainiert und die verbleibende Menge wird zur Evaluation verwendet. Dies wird n mal wiederholt, sodass jede Menge einmal als Testmenge verwendet wurde. Ein Extremfall von Kreuzvalidierung ist die Leave-One-Out Kreuzvalidierung, bei der $n = N$ gesetzt wird, wobei N die Anzahl Trainingsbeispiele ist. Es wird also immer auf nur einem Datenpunkt evaluiert.

5.2 Bayes’sche Modellselektion

Ein sehr anderer und statistisch motivierter Ansatz zur Modellselektion ist die *Bayes’sche Modellselektion*, bei der die Modelle anhand der A-Posteriori Wahrscheinlichkeit verglichen werden. Sei dazu $\{\mathcal{M}_m\}_{m=1,\cdots,M}$ die Menge aller Modelle und sei $\mathcal{T}$ die Trainingsdaten. Dann ist die A-Posteriori Wahrscheinlichkeit eines Modells $\mathcal{M}_m$ gegeben durch

$$P(\mathcal{M}_m \mid \mathcal{T}) \propto P(\mathcal{T} \mid \mathcal{M}_m)\, P(\mathcal{M}_m) \tag{5.1}$$

mit der Likelihood $P(\mathcal{T} \mid \mathcal{M}_m)$ und der A-Priori Wahrscheinlichkeit $P(\mathcal{M}_m)$. Zum Vergleich von zwei Modellen $\mathcal{M}_m$ und $\mathcal{M}_{m'}$ werden nun die A-Posteriori Wahrscheinlichkeiten verglichen:

$$\begin{aligned} P(\mathcal{M}_m) > P(\mathcal{M}_{m'}) \quad &\Longleftrightarrow \quad P(\mathcal{T} \mid \mathcal{M}_m)\, P(\mathcal{M}_m) > P(\mathcal{T} \mid \mathcal{M}_{m'})\, P(\mathcal{M}_{m'}) \\ &\Longleftrightarrow \quad \frac{P(\mathcal{T} \mid \mathcal{M}_m)}{P(\mathcal{T} \mid \mathcal{M}_{m'})} \frac{P(\mathcal{M}_m)}{P(\mathcal{M}_{m'})} > 1 \end{aligned} \tag{5.2}$$

Die erste Umformung ist dabei gültig, der in (5.1) ausgelassene Normalisierungsfaktor $P(\mathcal{T})$ für beide Modelle gleich ist. In (5.2) ist zu sehen, dass die A-Priori Wahrscheinlichkeiten verschwinden, wenn sie für beide Modelle gleich ist.

5.2.1 Approximation der Modell-Likelihood und Bayes'sches Informationskriterium

Hat das Modell $\mathcal{M}_m$ die Parameter θ_m mit dem Maximum Likelihood Ergebnis $\hat{\theta}_m$, so kann die Log-Likelihood des Modells durch

$$\log P(\mathcal{T} \mid \mathcal{M}_m) = \underline{\log P(\mathcal{T} \mid \mathcal{M}_m, \hat{\theta}_m)} - \frac{d_m}{2} \log N + \mathcal{O}(1) \tag{5.3}$$

angenähert werden. Dabei beschreibt d_m die Anzahl Parameter in dem Modell $\mathcal{M}_m$ und N ist die Anzahl der Trainingsbeispiele. Die unterklammerte Wahrscheinlichkeit ist dabei die Log-Likelihood, die durch die Parameter $\hat{\theta}_m$ bei gegebenem Modell maximiert wurde.

Aus dieser Approximation der Modell-Likelihood ergibt sich das *Bayes'sche Informationskriterium* (Englisch: *Bayes Information Criterion*, BIC). Dies entspricht den Approximationstermen der Log-Modell-Likelihood (5.3) mit gedrehtem Vorzeichen und skalierten Faktoren:

$$BIC(\mathcal{M}_m) = -2P(\mathcal{T} \mid \mathcal{M}_m, \hat{\theta}_m) + d_m \log N$$

Die Wahl eines Modells mit kleinstem BIC entspricht also der Wahl des Modells mit der höchsten A-Posteriori Wahrscheinlichkeit. Dabei bevorzugt das BIC durch den Einfluss der Parameteranzahl d_m einfachere Modelle. Die Auswahl unter Nutzung des BIC ist dabei auch zuverlässig: Aus einer Familie an Modellen, unter denen sich das richtige befindet, konvergiert die Wahrscheinlichkeit, dass das BIC das korrekte Modell identifiziert mit $N \to \infty$ gegen 1.

Wird das BIC für jedes Modell $\mathcal{M}_m$ berechnet, so können (wie bei der Kreuzvalidierung), die Modelle relativ zueinander bewertet werden:

$$\frac{\exp\{-\frac{1}{2} BIC(\mathcal{M}_m)\}}{\sum_{i=1}^{M} \exp\{-\frac{1}{2} BIC(\mathcal{M}_i)\}}$$

5.2.2 Minimale Beschreibungslänge

Einen anderen Weg zur Bewertung von Modellen stellt die *Minimale Beschreibungslänge* (Englisch: *Minimum Description Length*, MDL) dar. Dabei wird die Entropie der Modellverteilung $P(\mathcal{T} \mid \mathcal{M}_m, \theta_m)$ verwendet. Das Kriterium der minimalen Beschreibungslänge lautet dann

$$MDL(\mathcal{M}_m) = -\log P(\mathcal{T} \mid \mathcal{M}_m, \theta_m) - \log P(\theta_m \mid \mathcal{M}_m),$$

welche es zu minimieren gilt. Eine Minimierung dieses Kriteriums entspricht der Minimierung der aufgewandten Modellgröße unter Einbeziehung der Vorhersagekraft. Dabei wird implizit die A-Posteriori Wahrscheinlichkeit maximiert, weshalb auch durch Minimierung des BIC das Modell mit der kleinsten Beschreibungslänge gefunden werden kann. Bei normalverteilten Ausgabewerten und Parametern führt das MDL-Kriterium zu einer Minimierung der Varianz.

5.3 Evaluierungsmaße

In diesem Abschnitt werden weitere (empirische) Evaluierungsansätze beschrieben, die zusätzlich zu den bereits vorgestellten Ansätzen zur Modellselektion genutzt werden (sollten).

Eine offensichtliche Möglichkeit ist die Evaluierung durch Experten, d. h. die Vorhersagen des Modells werden von Menschen geprüft, die sich sehr gut mit der Anwendungsdomäne auskennen. Dies ist leider oft die einzige Möglichkeit, aber subjektiv, kostspielig und zeitaufwendig. Eine andere Möglichkeit ist die online-Evaluierung, wo das Modell direkt eingesetzt und in der Praxis evaluiert wird. Dies ergibt die beste Schätzung für die Funktionalität des Modells, kann aber kostspielig werden, wenn das Modell falsche Vorhersagen trifft. Ein Beispiel ist die direkte Anwendung eines Aktienhandel-Modells an der Börse. Dies kann sehr gut funktionieren, kann aber im schlimmsten Fall zu dem kompletten finanziellen Ruin führen.

In der Praxis sollten nach einer Evaluation immer empirische Gütemaße angegeben werden. Einige dieser werden in den folgenden Abschnitten vorgestellt.

5.3.1 Konfusionsmatrix und Gütemaße

In der *Konfusionsmatrix* (Englisch: *Confusion Matrix*) werden fast alle wichtigen Informationen (in der Klassifikation) zusammengefasst, indem die Anzahl korrekt-positiven, korrekt-negativen, falsch-positiven und falsch-negativen Klassifikationen zusammengefasst wird:

	Klassifiziert als Positiv	Klassifiziert als Negativ	$\sum$
Ist Positiv	Korrekt-Positive (TP)	Falsch-Negative (FN)	Alle Positiven (P)
Ist Negativ	Falsch-Positive (FP)	Korrekt-Negative (TN)	Alle Negativen (N)
$\sum$	Alle als Positiv klassifizierten	Alle als Negativ klassifizierte	Anzahl Bespiele

Aus dieser Matrix lassen sich fast alle Gütemaße ablesen, wie beispielsweise:

- *Korrekt-Positiv Rate* (TPR): $\dfrac{\mathrm{TP}}{(\mathrm{TP}+\mathrm{FN})}$
  Anteil der korrekt klassifizierten positiven Beispiele.
- *Falsch-Positiv Rate* (FPR): $\dfrac{\mathrm{FP}}{\mathrm{FP}+\mathrm{TN}}$
  Anteil der negativen Beispiele, die als positiv Klassifiziert werden.
- *Falsch-Negativ Rate* (FNR): $\dfrac{\mathrm{FN}}{\mathrm{TP}+\mathrm{FN}} = 1 - \mathrm{TPR}$
  Anteil der positiven Beispiele, die als negativ Klassifiziert werden.
- *Korrekt-Negativ Rate* (TNR): $\dfrac{\mathrm{TN}}{\mathrm{FP}+\mathrm{TN}} = 1 - \mathrm{FPR}$
  Anteil der korrekt klassifizierten negativen Beispiele.
- *Genauigkeit/Accuracy* (Acc): $\dfrac{\mathrm{TP}+\mathrm{TN}}{\mathrm{N}+\mathrm{P}}$
  Anteil der korrekt klassifizierten Beispiele.
- *Fehler* (Err): $\dfrac{\mathrm{FP}+\mathrm{FN}}{\mathrm{N}+\mathrm{P}} = 1 - \mathrm{Acc}$
  Anteil der falsch klassifizierten Beispiele.

Auch für mehrere Klassen kann eine Konfusionsmatrix erstellt werden, in diesem Fall werden die Zeilen und Spalten entsprechend erweitern. Die Genauigkeit ergibt sich dann als Summe der Diagonaleinträge geteilt durch die Gesamtanzahl.

5.3.2 ROC-Analyse und -Kurve

Häufig ist es nicht aussagekräftig, die Fehlklassifikationsrate zu verwenden, da die Gesamtzahl der positiven Datenpunkte im Vergleich zu den negativen sehr klein ist. Dies führt zu einer guten Genauigkeit, bzw. einem kleinen Fehler, und suggeriert ein gutes Modell. Dennoch ist es möglich, dass viele positiven Fälle nicht entdeckt werden, was bspw. bei einer Krebserkrankung fatale Auswirkungen haben kann. Ein ausführliches Beispiel hierzu wird in Unterunterabschnitt 5.3.3 beschrieben.

Es ist natürlich bei jedem Lernverfahren möglich, gewisse Klassen höher zu Gewichten, um eine falsche Einschätzung aufgrund der Klassenhäufigkeiten zu vermeiden.

Eine Alternative bietet die *Receiver Operating Characteristic*, die direkt die Entscheidungsfunktion bewertet (dies ergibt die sogenannte *ROC-Kurve*). Bei dieser Kurve entspricht jeder Punkt einem Schwellenwert θ, d. h. es muss ein Klassifikator

$$y = \begin{cases} +1 & \text{falls } \hat{f}(\boldsymbol{x}) \geq \theta \\ -1 & \text{sonst} \end{cases}$$

mit Regressionsmodell $\hat{f}$ genutzt werden. Der Fehler hängt also vom Schwellenwert ab: Bei einem hohen Schwellenwert sind mehr positive Beispiele falsch, bei einem kleinen Schwellenwert sind mehr negative Beispiele falsch. Die ROC-Analyse verschafft hier Abhilfe, indem sie unabhängig von dem Schwellenwert arbeitet, d. h. das Verhalten des Klassifikators wird für alle möglichen Schwellenwerte untersucht.

Zur Erstellung der ROC-Kurve wird auf der x-Achse die FPR und auf der y-Achse die TPR angegeben. Ein perfekter Klassifikator stellt dann einen einzigen Punkt in der oberen linken Ecke (mit einer TPR von Eins und einer FPR von Null) dar, ein zufälliger Klassifikator eine diagonale Linie von unten links nach oben rechts. Die Fläche unter der ROC-Kurve gibt dann die Wahrscheinlichkeit an, dass ein positives Beispiel einen höheren Wert $\hat{f}(\boldsymbol{x})$ als ein negatives Beispiel hat.

In Algorithmus 1 ist schematisch der Ablauf der Erstellung einer ROC-Kurve gezeigt. Anschließend kann der Flächeninhalt der ROC-Kurve durch numerische Integration ermittelt werden. Für ein Positivbeispiel $\boldsymbol{x}_+$ und ein Negativbeispiel $\boldsymbol{x}_-$ gibt der Flächeninhalt zwischen diesen Werten die Wahrscheinlichkeit $P\big(\hat{f}(\boldsymbol{x}_+) > \hat{f}(\boldsymbol{x}_-)\big)$ an.

Algorithmus 1 : Erstellung einer ROC-Kurve

```
1  Generiere eine Liste L mit allen Beispielen x, aufsteigend sortiert nach f̂(x). ;
2  Sei P die Anzahl positiver und N die Anzahl negativer Bespiele. ;
3  Setze TP ← 0 und FP ← 0. ;
4  for i = 1 bis zure Länge von L do
5      Sie x_i das i-te Element von L. ;
6      if x positive Instanz then
7          TP ← TP + 1 ;
8      else
9          FP ← FP + 1 ;
10     Zeichne einen neuen Punkt mit den Koordinaten (FP/N, TP/P). ;
```

5.3.3 Präzision, Sensitivität und F1-Wert

Eine weitere Alternative zur Ergebnisanalyse mit der Genauigkeit neben der ROC-Analyse stellen die häufig verwendeten Gütemaße *Präzision* und *Sensitivität* (Englisch: *Recall*) dar. Diese werden wie folgt berechnet:

$$\text{Präzision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \qquad \text{Sensitivität} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

Sie beschreiben also die Wahrscheinlichkeiten $P(\text{positiv} \mid \text{positiv vorhergesagt})$ und $P(\text{positiv vorhergesagt} \mid \text{positiv})$. Da im allgemeinen nicht beide dieser Werte vollständig optimiert werden können (dies ist der sogenannte Precision-Recall Trade-Off), ist es hilfreich, die beiden Werte in einen einzigen zusammenzufassen. Eine Möglichkeit der Zusammenfassung ist das harmonische Mittel von Präzision und Sensitivität, welche *F1-Wert* genannt wird:

$$\text{F1} = \frac{2 \cdot \text{Präzision} \cdot \text{Sensitivität}}{\text{Präzision} + \text{Sensitivität}}$$

Werden Präzision und Sensitivität gegeneinander als Scatter-Plot aufgetragen, bilden sich zwei Kurven, die sich immer schneiden dieser Schnittpunkt ist der *Breakeven-Punkt* von Präzision und Sensitivität, an dem beide Maße den gleichen Wert annehmen.

Probleme des Genauigkeits-Maß und Nutzen des F1-Werts

Im Vergleich zu der naheliegenden Genauigkeit liefert der F1-Wert ein sehr viel besseres Maß für die Güte eines Modells, da die Genauigkeit eine sehr hohe Güte suggerieren kann, da die Gesamtzahl positiver/negativer Fälle stark überwiegt. Beispiel: Weltweit sind ca. $T = 14$ Millionen Menschen an Krebs erkrankt bei einer Grundmenge von $T + N = 7500$ Millionen Menschen (d. h. $N = 7486$). Klassifiziert ein Modell nun eine Millionen Menschen korrekt als erkrankt, d. h. positiv, und alle anderen negativ, gibt keine falsch-positiven Ergebnisse. Die Genauigkeit ist demnach

$$\text{Genauigkeit} = \frac{\text{TP} + \text{TN}}{\text{P} + \text{N}} = \frac{1 + 7486}{14 + 7486} = \frac{7487}{7500} \approx 99.83\%.$$

Jedoch ist der Schaden durch 13 Millionen nicht erkannte Erkrankungen sehr hoch, das Genauigkeits-Maß suggeriert aber, dass das Modell sehr gut ist. Präzision und Sensitivität sind

$$\text{Präzision} = \frac{\text{TP}}{\text{TP} + \text{FP}} = \frac{1}{1 + 0} = 100\% \qquad \text{und} \qquad \text{Sensitivität} = \frac{\text{TP}}{\text{TP} + \text{FN}} = \frac{1}{1 + 13} \approx 7.14\%,$$

das heißt alle als positiv klassifizierten Personen sind tatsächlich krebs-erkrankt, es werden jedoch nicht ansatzweise alle erkrankten Personen erkannt. Daraus ergibt sich ein F1-Wert von

$$\text{F1} = \frac{2 \cdot \text{Präzision} \cdot \text{Sensitivität}}{\text{Präzision} + \text{Sensitivität}} = \frac{2 \cdot 1 \cdot 0.0714}{1 + 0.0714} \approx 13.32\%,$$

was das Modell deutlich schlechter bewertet. Daher ist der F1-Wert im Allgemeinen besser geeignet, um die Güte eines Modells abzuschätzen.

5.4 Vergleichen von Algorithmen

Um Algorithmen fair zu vergleichen, kann die Nutzung eines statistischen Tests hilfreich sein, da die bereits vorgestellten Maße häufig trügerisch sein können. Bei einem statistischen Test wird eine *Nullhypothese*

aufgestellt, welche dann durch ein stochastisches Verfahren verworfen oder akzeptiert wird. In diesem Abschnitt wird ein Vorzeichen-Test vorgestellt, der für eine Nullhypothese der Form „A und B sind gleich“ geeignet ist. Eine Beispielhypothese adressiert die Wahrscheinlichkeiten für Kopf (A) und Zahl (A) bei einem Münzwurf. In diesem Fall ist die Nullhypothese „Die Münze ist fair, d. h. $P(A) = P(B)$.“ Ähnlich kann die Nullhypothese formuliert werden als „Die Algorithmen A und B sind gleich“. Insgesamt werden N Versuche durchgeführt, i mal gewinnt A und $N - i$ mal gewinnt B.

Nach der Binomialverteilung mit $p = 0.5$ ist die Wahrscheinlichkeit für einen i-fachen Erfolg gegeben durch

$$P(i) = \binom{N}{i} p^i (1-p)^{N-1},$$

wobei $\binom{N}{i}$ den Binomialkoeffizienten darstellt. Bei einem einseitigen Test wird die Wahrscheinlichkeit genutzt, dass *höchstens* k Erfolge eintreten:

$$P(i \leq k) = \sum_{i=1}^{k} \binom{N}{i} \frac{1}{2^i} \frac{1}{2^{N-i}} = \frac{1}{2^N} \sum_{i=1}^{k} \binom{N}{i}$$

Analog kann auch ein zweiseitiger Test verwendet werden, bei dem die Wahrscheinlichkeit genutzt wird, dass die Anzahl Erfolge höchstens k und mindestens $N - k$ ist:

$$P(N-k \leq i \leq k) = \frac{1}{2^N} \sum_{i=1}^{k} \binom{N}{i} + \frac{2}{N} \sum_{i=1}^{k} \binom{N}{N-i} = \frac{1}{2^{N-1}} \sum_{i=1}^{k} \binom{N}{i}$$

Ist N groß, so kann für den Test eine Normalverteilung genutzt werden.

Der Vorzeichen-Tests ist insbesondere wegen seiner Einfachheit und der nicht-Annahme einer zugrundeliegenden Verteilung gut geeignet für derlei statistische Tests. Allerdings ist er eher konservativ, dass bedeutet, er lehnt die Nullhypothese nicht schnell ab. Dies hat einerseits den Vorteil, dass das Ergebnis im Falle der Ablehnung sehr sicher ist. Andererseits wird aber oft auch kein Unterschied detektiert, d. h. ein anderer Test könnte einen signifikanten Unterschied feststellen. Eine Alternative ist der zweiseitige t-Test, bei dem die Größe des Unterschieds unter der Annahme einer Normalverteilung derselben betrachtet wird.

Als Faustregel gilt: Der Vorzeichen-Test beantwortet die Frage „Wie häufig sind A und B unterschiedlich?“ und der t-Test beantwortet die Frage „Um wie viel sind A und B unterschiedlich?“

6 Baumbasierte Verfahren

Baumbasierte Verfahren stellen einen grundlegend anderen Ansatz zur Klassifikation dar als bisheriger Verfahren. Dabei wird ein Baum, der sogenannte *Entscheidungsbaum*, konstruiert, dessen Knoten abfragen der Form „Hat das Attribut X den Wert x?“ oder „Ist der Wert des Attributs X kleiner als x?“ darstellen. Die Blätter des Baumes repräsentieren dann die abschließende Klassifikation. Entscheidungsbäume gehören dabei zu den *globalen Modelle*, d. h. sie Teilen des gesamten Merkmalsraum in verschieden Gruppen (Klassen) ein. Diese Aufteilung erfolgt rekursiv und die Aufteilung wird automatisch aus den Trainingsdaten bestimmt. Die Entscheidungen werden dabei so gewählt, dass die Blätter möglichst „reine“ Klassen darstellen, d. h. dass sich keine Beispiele mehrerer Klassen in den Blättern wiederfinden.

Das Lernen und Trainieren von Entscheidungsbäumen ist dabei sehr effizient und skaliert gut. Soll ein Entscheidungsbaum für p kategorische (nicht-numerische) Merkmale aus N Trainingsbeispielen erstellt werden, so liegt die Komplexität des im nächsten Abschnitt vorgestellten ID3-Algorithmus in $\mathcal{O}(pN \log N)$.

Das resultierende Modell hat den großen Vorteil, dass es direkt eine Begründung für eine Entscheidung mitliefert, indem die Entscheidungsreihenfolge betrachtet wird. Dies hat den Anschein als seien Baumlernverfahren im Allgemeinen gut interpretierbar, allerdings lässt diese Interpretierbarkeit mit der Größe des Baumes nach.

Es ist auch möglich, Baumverfahren zur Regression einzusetzen und sie bspw. mit linearen Modellen zu kombinieren.

6.1 Top-Down Induction of Decision Trees (TDIDT): ID3

Ein einfacher Algorithmus zur Erstellung von Entscheidungsbäumen ist der *ID3-Algorithmus*, welcher ein TDIDT-Verfahren darstellt. In jeder Iteration des ID3-Algorithmus wird genau eine Entscheidung erstellt, d. h. es wird ein Merkmal ausgewählt, an dem der Baum geteilt wird. Dabei ist die Idee stets das Merkmal zu wählen, welches die höchste Güte hat. Die Güte kann bspw. der Informationsgewinn sein (einige Gütemaße werden in Abschnitt 6.2 vorgestellt).

Ein Merkmal M_j mit k möglichen Werten (bspw. das Merkmal „Luftfeuchtigkeit“ mit den zwei Werten „hoch“ und „niedrig“) teilt die Gesamtdatenmenge $\mathcal{X}$ in genau k Mengen $\mathcal{X}_1, \cdots, \mathcal{X}_k$ auf. Für jede dieser Teildatenmengen wird der ID3-Algorithmus nun erneut ausgeführt, wobei das Merkmal M_j entfernt wird. Dieser Algorithmus ist in Algorithmus 2 skizziert.

Einer besonderen Behandlung bedürfen numerische Werte, da diese nicht direkt in endlich viele Kategorien eingeteilt werden können. Die einfachste Lösung ist hier jedoch, die Werte in Töpfe einzuteilen, für das Merkmal „Temperatur“ beispielsweise die Intervall $(-\infty, 0)$, $[0, 30)$ und $[30, \infty)$. Im Allgemeinen ist diese Einteilung jedoch willkürlich und muss oft händisch optimiert werden, was bei der Baumbildung ein Problem darstellt.

Algorithmus 2 : ID3-Algorithmus

Input : Datenmenge $\mathcal{X}$, Merkmalsmenge $\{M_1, \cdots, M_p\}$
Output : Entscheidungsbaum T

if *$\mathcal{X}$ enthält nur Datenpunkt einer Klasse y* **then**
    return Terminalknoten für Klasse y.
else
    Wähle $M_j \leftarrow \arg\max_{M \in \{M_1, \cdots, M_p\}}$ *Güte*$(M, \mathcal{X})$
    Teile $\mathcal{X}$ in $\mathcal{X}_1, \cdots, \mathcal{X}_k$ auf
    for $i = 1, \cdots, k$ **do**
        Rufe den ID3-Algorithmus erneut auf: *ID3*$\Big(\mathcal{X}_i, \{M_1, \cdots, M_p\} \setminus \{M_j\}\Big)$
    return Entscheidungsknoten für Merkmal M_j mit den Teilbäumen $\{T_i\}_{i=1, \cdots, k}$

6.2 Gütemaße

Das Herz des ID3-Algorithmus ist die Wahl eines Gütemaßes zur Auswahl des (nächsten) Entscheidungsmerkmals. In diesem Abschnitt werden die zwei wichtigsten Kriterien, der Informationsgewinn und der Gini-Index, beschrieben.

6.2.1 Informationsgewinn

Das naheliegendste Maß für die Güte einer bestimmten Aufteilung ist der durch das Merkmal induzierte Informationsgewinn. Das Maß des Informationsgewinns basiert auf der Entropie $H(\mathcal{X})$ des Datensatzes in Bezug auf die Klasse y. Dabei ist die Information I, die durch ein Merkmal M_j induziert wird, gegeben durch

$$I(M_j, \mathcal{X}) = -\sum_{i=1}^{k} \frac{|\mathcal{X}_i|}{|\mathcal{X}|} H(\mathcal{X}_i).$$

Das bedeutet die Entropie des Datensatzes wird für jede durch das Merkmal hervorgerufene Aufteilung $\mathcal{X}_i$, $i = 1, \cdots, k$, berechnet. Der Informationsgewinn ist dann die Differenz zwischen der Information mit und ohne die Aufteilung, also

$$\Delta I(M_j, \mathcal{X}) = I(M_j, \mathcal{X}) - I(-, \mathcal{X}),$$

wobei $I(-, \mathcal{X}) = -H(\mathcal{X})$ ist.

Im ID3-Algorithmus wird dann das Merkmal verwendet, welches den Informationsgewinn $\Delta I(M_j, \mathcal{X})$ maximiert.

6.2.2 Gini-Index

Eine andere Möglichkeit zur Bewertung der Güte ist der *Gini-Index*, welcher quantifiziert, wie „rein“ die Klasse ist, die ein Knoten darstellt. Er wird berechnet als

$$\mathit{Gini}(\mathcal{X}) = 1 - S(\mathcal{X}) \qquad S(\mathcal{X}) = \sum_{y \in \mathcal{Y}} \big(P_{\mathcal{X}}(y)\big)^2,$$

wobei S die *Reinheitsfunktion* darstellt. Die Wahrscheinlichkeit $P_{\mathcal{X}}(y \mid t)$ stellt den Anteil der Beispiele dar, die im Datensatz $\mathcal{X}$ der Klasse y angehören. Die Menge $\mathcal{Y}$ bezeichnet dabei alle möglichen Klassen.

Der Gini-Index nimmt sein Maximum an, wenn alle Klassen gleich wahrscheinlich sind und sein Minimum, wenn ausschließlich eine Klasse vorkommt. Ein hoher Gini-Index spricht charakterisiert also einen unreinen und ein niedriger einen reinen Knoten.

Ein baumbasiertes Verfahren, welches den Gini-Index verwendet, ist *Classification and Regression Trees* (CART). Aber auch für ID3 kann der Gini-Index angewandt werden. In diesem Fall wird er als Unreinheitsmaß verwendet und die Reduzierung der Unreinheit

$$\Delta Gini(M_j, \mathcal{X}) = Gini(\mathcal{X}) - \sum_{i=1}^{k} \frac{|\mathcal{X}_i|}{|\mathcal{X}|} Gini(\mathcal{X}_i).$$

Dabei beschreiben $\mathcal{X}_i$, $i = 1, \cdots, k$ die durch das Merkmal M_j hervorgerufenen Teilmengen. Im Gegensatz zum Informationsgewinn soll der Gini-Index also minimiert, d. h. die Reinheit maximiert, werden. Dies ist bei der Implementierung zu beachten und kann bspw. durch Negierung des Informationsmaßes geschehen.

6.2.3 Regression

Bei der Regression ist das Ziel die Minimierung der Fehlerquadrate

$$SSE(t) = \sum_{(\boldsymbol{x},y)\,\in\, t} (y - \bar{y})^2 \qquad \text{mit} \qquad \bar{y} = \frac{1}{|t|} \sum_{(\boldsymbol{x},y)\,\in\, t} y,$$

wobei t ein Knoten, bzw. die Menge der Trainingsdaten in diesem Knoten, ist. Werden Knoten der Form $x \leq z$, also mit einem Schwellenwert z bzgl. eines Merkmals x, verwendet, ist die SSE-Reduktion gegeben durch

$$\Delta SSE(t) = SSE(t) - \big(SSE(t_-) + SSE(t_+)\big),$$

wobei t_- und t_+ den linken, bzw. rechten, Teilbaum beschreiben.

Es ist außerdem Sinnvoll eine Mindestreduktion einzuführen, um Blättern zu erzeugen und den Baum nicht zu stark wachsen zu lassen.

6.3 Stutzen (Pruning) des Baumes

Ein Problem der vorgestellten Baumlernverfahren ist, dass die Bäume sehr groß werden können. Dies kann zu zwei Problem führen:

- Sie erzeugen eine hohe Fehlerrate auf neuen Datensätzen, d. h. der Baum wird auf die Trainingsdaten überangepasst.
- Die Interpretierbarkeit ist bei großen Bäumen mit vielen Blättern schwer bis nicht möglich.

Das heißt die „beste“ Größe eines Baumes liegt irgendwo zwischen nur einem Blatt und dem größtmöglichen Baum. Ein gelernter Baum muss also eventuell gestutzt werden, d. h. es müssen Blätter und Entscheidungsknoten entfernt werden. Ein Stutzungsschritt führt die folgenden zwei Operationen aus: Zunächst wird ein Knoten an die Stelle eines Teilbaums gesetzt, anschließend wird der Teilbaum eine Ebene höher gezogen.

Die Suche nach dem Baum mit der „richtigen“ Größe beginnt dabei mit der Stutzung der Äste aus der Richtung der Blätter („von unten nach oben“). Die Unterknoten eines Knotens können dabei weggestutzt werden, wenn die Summe der Fehler der Unterknoten größer ist als der Fehler des betrachteten Knotens.

Dieser Fehler kann jedoch nur geschätzt werden, weshalb hier auch die Sicherheit (Konfidenz) der Schätzung berücksichtigt werden sollte. Dies kann beispielsweise dadurch geschehen, keinen Punktwert zu schätzen, sondern ein Konfidenzintervall. Anschließend kann die obere Schranke des Konfidenzintervalls verwendet werden um den Baum zu Schätzen.

7 Ensemble-Methoden

In diesem Kapitel werden *Ensemble-Methoden* behandelt, die im Vergleich zu den bisher vorgestellten Verfahren eine Meta-Ebene höher liegen, indem sie mehrere Verfahren kombinieren. Die Idee hinter Ensembles ist, mehrere „schwache“ Lerner zu einem mächtigeren Lerner zu kombinieren. Dabei gibt es in der Grundform zwei Möglichkeiten: Bagging und Boosting. Beim Bagging werden mehrere Modelle parallel trainiert die Vorhersagen anschließend bspw. mittels Mehrheitsentscheid kombiniert. Beim Boosting werden die Modelle nacheinander trainiert und der Fehler des vorherigen Modells wird zur Gewichtung der Trainingsdaten für das nächste Modell verwendet. Dabei wird ein Mitspracherecht errechnet, mit dem anschießend Vorhersagen getroffen werden.

Der Ausgangspunkt von Ensembles ist der Zusammenhang von Bias und Varianz: Dabei weißt ein überangepasstes Modell einen niedrigen Bias, aber eine hohe Varianz auf und ein unterangepasstes Modell einen hohen Bias, aber eine niedrige Varianz. Das bedeutet, dass ein unterangepasstes Modell nicht flexibel und ein überangepasstes Modell zu flexibel ist. Ensemble-Methoden versuchen, diese Probleme zu bekämpfen. Dabei geht Adaptives Boosting (AdaBoost) gegen Unteranpassung, und Bagging gegen Überanpassung vor. Weitere verbesserte Ensembles sind beispielsweise Gradienten Boosting, was sowohl Über- als auch Unteranpassung verbessert, und Zufallswälder, bei denen Baumlernverfahren verwendet werden.

Im Folgenden werden zunächst Zufallswälder und an diesem Beispiel Bagging und anschließend Boosting behandelt.

7.1 Zufallswälder (Random Forests)

Zufallswälder (Englisch: *Random Forests*) sind eine Variante verschiedener *Bagging*-Algorithmen und klassifizieren Daten auf Basis von mehreren Klassifikationsbäumen, wobei das Lernen der Bäume randomisiert geschieht. Um die Entscheidungen der Bäume zu aggregieren, wird ein einfacher Mehrheitsentscheid verwendet, d. h.

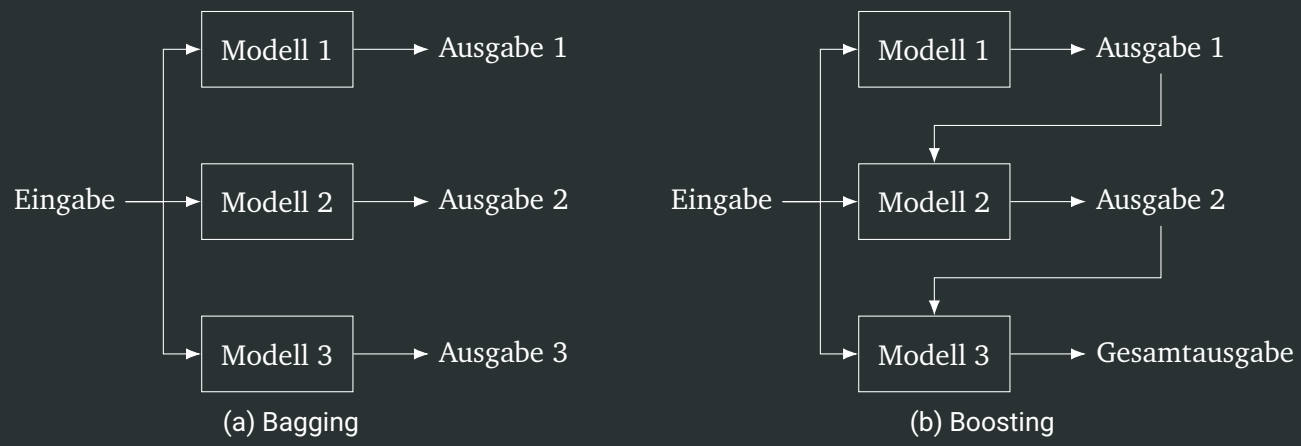

Abbildung 7.1: Illustration von Bagging und Boosting.

ein Datenpunkt wird in die Klasse gesteckt, in die sie die meisten Bäume einordnen.

Um unterschiedliche einzelne Bäume zu erhalten werden nicht jedem Baum die gleichen Merkmale zur Verfügung gestellt, sondern jeder Baum erhält nur eine zufällige Stichprobe aus den Merkmalen. Ebenfalls werden, als weiterer Zufallsaspekt, nicht alle Merkmale in der Bestimmung der nächsten Verzweigung verwendet, sondern ebenfalls nur eine Stichprobe aus den dem Baum zur Verfügung stehenden Merkmalen. Im Gegensatz zur Verwendung von nur einem Baum wird bei Zufallswäldern der Baum nicht gestutzt, d. h. jeder Baum hat (auf den gezogenen Stichproben) den kleinstmöglichem Trainingsfehler.

Der offensichtliche Hyperparameter bei Zufallswäldern ist die zu verwendende Anzahl an Entscheidungsbäumen. Im Allgemeinen ist die Gesamtklassifikationskraft um so besser, je mehr einzelne Bäume verwendet werden. Da die Überanpassung aufgrund des Zufallsaspekts nicht mit der Anzahl der Bäume steigt, spricht also aus theoretischer Sicht nichts gegen die Wahl von sehr vielen Bäumen. Praktisch ist diese Anzahl jedoch einerseits durch die Rechenkraft limitiert, auch wenn dieses Limit sehr hoch liegt, andererseits kann die Anzahl möglicher Merkmalskombinationen schnell ausgeschöpft sein, wenn nicht viele Merkmale vorliegen. Daher ist es in der Praxis oft ratsam, nicht allzu viele Bäume zu verwenden.

Im Vergleich zu nur einem Entscheidungsbäumen haben Zufallswälder den Vorteil, dass jede Variable, die zur Klassentrennung beiträgt, irgendwann auch verwendet wird. Ein großer Nachteil ist jedoch, dass die Verständlichkeit der Regeln verloren geht, was insbesondere an dem abschließenden Mehrheitsentscheid liegt. Dieser Verlust der Verständlichkeit offenbart jedoch wieder eine andere Möglichkeit zur Interpretation der Daten: Da jedes Merkmal auf verschiedenste Weise zu der Klassifikation beitragen kann, kann die Wichtigkeit der Merkmale abgeschätzt werden. Eine Möglichkeit ist

$$I = \frac{VG}{NV},$$

wobei VG die Summe der Verminderungen des Gütemaßes für das Merkmal und NV die Anzahl an Verzweigungen im gesamten Wald ist.

7.1.1 Bagging Allgemein

Im Allgemeinen ist *Bagging* (*Bootstrap Aggregation*) immer dann besser, wenn jeder einzelne Klassifikator etwas besser ist als zufälliges Raten. Beispiel: Hat bei 25 Modellen jeder Klassifikator eine Fehlerrate von $\epsilon = 0.35$, d. h. es werden nur $65\,\%$ der Daten korrekt klassifiziert, dann liegt die Wahrscheinlichkeit, dass das Mehrheitsentscheid-Ensemble einen Fehler macht bei nur $6\,\%$:

$$P(\text{falsche Vorhersage}) = \sum_{i=13}^{25} \binom{25}{i} \epsilon^i (1-\epsilon)^{25-i} = 0.06$$

Aber auch für Regression ist Bagging nützlich: Werden M verschiedene Regressionsmodelle genutzt, die alle sehr flexibel, d. h. überangepasst, sind, haben alle einen geringen Bias, aber eine hohe Varianz. Durch Mittlung der Ergebnisse bleibt der Erwartungswert nach dem zentralen Grenzwertsatz gleich, die Varianz wird aber um den Faktor $\sqrt{M}$ reduziert. Im Fall von $M = 25$ Modellen wird die Varianz also schon um den Faktor $\sqrt{M} = \sqrt{25} = 5$ reduziert.

Bagging-Verfahren sind also insgesamt geeignet, um die Aussagestärke einer Modellfamilie zu erhöhen, indem mehrere einfachere Modelle auf einem zufälligem Anteil der Daten trainiert werden. Die Ergebnisse werden anschließend durch einen Mehrheitsentscheid (in der Klassifikation) oder mittels Durchschnittsbildung (in der Regression) zusammengefasst.

7.2 Boosting

Im *Boosting* werden die einzelnen Modelle im Gegensatz zum Bagging nicht nebeneinander, sondern hintereinander trainiert. Die Idee ist, dass es leichter ist, ein paar Faustregeln statt einer Gesamtlösung für ein Problem zu finden. Beim Boosting werden also keine mächtigen Lerner kombiniert, sondern es werden einfache Verfahren wie Naive Bayes, Logistische Regression, Entscheidungsstümpfe oder flache Entscheidungsbäume kombiniert. Das Ergebnis ist dann ein starker Lerner.

In jeder des Boosting werden die Trainingsbeispiele neu gewichtet, je nach, „wie falsch“ es im vorherigen Schritt klassifiziert wurde. Der nächste Lerner wird dann auf den gewichteten Beispielen trainiert und es wird ein Mitspracherecht auf Basis des Fehlers berechnet. Kann der Lerner nicht mit gewichteten Beispielen umgehen, so können die (auf Eins normierten) Gewichte als Wahrscheinlichkeiten betrachtet werden, sodass zum trainieren jedes Lerners nur eine Stichprobe der eigentlichen Daten verwendet wird. Das funktioniert allerdings nur dann, wenn der schwache Lerner mit einer unterschiedlichen Anzahl an Trainingsdaten umgehen kann.

Boosting ist also im allgemeinen sehr mächtig und funktioniert bei vielen Problemen, unter einigen technischen Voraussetzungen gibt es bei einigen Algorithmen sogar Konvergenzgarantien. So konvergiert bspw. AdaBoost zu einem Fehler von Null, wenn die Anzahl an Iterationen steigt und die entsprechenden Voraussetzungen erfüllt sind.

7.2.1 AdaBoost

In diesem Abschnitt wird *AdaBoost*, ein Boosting-Algorithmus beschrieben und Teile des Algorithmus hergeleitet. Die Grundform zur Klassifikation von Daten in die zwei Klassen -1 und $+1$ ist in Algorithmus 3 gezeigt. Die Auswahl von M, der maximalen Anzahl AdaBoost-Iterationen, geschieht empirische wie die Wahl sämtlicher anderer Hyperparameter im maschinellen Lernen auch (z. B. auf Basis eines Testdatensatzes).

Herleitung der Gewichte und des Mitspracherechts

Bei einem Blick auf den AdaBoost-Algorithmus stellt sich die Frage, warum die Gewichte und das Mitspracherecht gerade so gewählt werden. Dies ist darauf zurück zu führen, dass AdaBoost (in der Grundform) des Exponentialverlust $\exp\left\{-y\hat{f}(\boldsymbol{x})\right\}$ annimmt, welcher eine obere Schranke für den 0-1-Verlust darstellt. Der Vorteil des Exponentialverlusts ist, dass dieser Differenzierbar ist. Es wäre auch möglich, andere differenzierbare Verlustfunktionen zu verwenden, wodurch AdaBoost zu Gradienten Boosting verallgemeinert wird (siehe Abschnitt 7.3).

Im m-ten Schritt hat der Gesamtverlust des starken Lerners, also der aggregierten Teilmodelle, die Form

$$E = \sum_{i=1}^{N} \exp\left\{-y_i \sum_{j=1}^{m} \alpha^{(j)} \hat{f}^{(j)}(\boldsymbol{x}_i)\right\},$$

Algorithmus 3 : AdaBoost (für zwei Klassen)

Input : Trainingsdaten $\{(\boldsymbol{x}_i, y_i)\}_{i=1,\cdots,N}$ mit binären Labels $y_i \in \{-1, +1\}$

Output : Starker Klassifikator

$w_i^{(1)} \leftarrow 1/N$ für $i = 1, \cdots, N$

for $m = 1, \cdots, M$ **do**

  Anpassen des Modells $\hat{f}^{(m)}(\boldsymbol{x})$ unter Einbeziehung der Gewichte $\boldsymbol{w}_{1:N}^{(m)}$.

  // Berechnung des Fehlers:

  $\epsilon^{(m)} \leftarrow \sum_{i=1}^{N} w_i^{(m)} \cdot \mathbb{1}\big[y_i \neq \hat{f}^{(m)}(\boldsymbol{x}_i)\big]$

  // Berechnung des Mitspracherechts:

  $\alpha^{(m)} \leftarrow \frac{1}{2} \log\left(\frac{1-\epsilon^{(m)}}{\epsilon^{(m)}}\right)$

  // Berechung der neuen (nicht normalisierten) Gewichte:

  $\tilde{w}_i^{(m+1)} \leftarrow w_i^{(m)} \exp\Big(\alpha^{(m)} \cdot \mathbb{1}\big[y_i \neq \hat{f}^{(m)}(\boldsymbol{x}_i)\big]\Big)$ für $i = 1, \cdots, N$

  // Normalisieren der Gewichte:

  $w_i^{(m+1)} \leftarrow \frac{\tilde{w}_i^{(m+1)}}{\sum_{i=1}^{N} \tilde{w}_i^{(m+1)}}$

return Klassifikator $\hat{F}^{(M)}(\boldsymbol{x}) = \operatorname{sign}\Big(\sum_{m=1}^{M} \alpha^{(m)} \hat{f}^{(m)}(\boldsymbol{x})\Big)$

wobei das aktuell zu lernende Modell $\hat{f}^{(m)}$ wie folgt separiert werden kann:

$$
\begin{aligned}
&= \sum_{i=1}^{N} \exp\left\{-y_i \textstyle\sum_{j=1}^{m-1} \alpha^{(j)} \hat{f}^{(j)}(\boldsymbol{x}_i) - y_i \alpha^{(m)} \hat{f}^{(m)}\right\} \\
&= \sum_{i=1}^{N} \underbrace{\exp\left\{-y_i \textstyle\sum_{j=1}^{m-1} \alpha^{(j)} \hat{f}^{(j)}(\boldsymbol{x}_i)\right\}}_{w_i^{(m)} :=} \exp\left\{-y_i \alpha^{(m)} \hat{f}^{(m)}\right\} \\
&= \sum_{i=1}^{N} w_i^{(m)} \exp\left\{-y_i \alpha^{(m)} \hat{f}^{(m)}\right\}
\end{aligned}
$$

Es ergeben sich also die in AdaBoost beschriebenen Gewichte aus der Wahl des Exponentialverlusts als obere Schranke für den 0-1-Verlust.

Zur Herleitung des neuen Mitspracherechts $\alpha^{(m)}$ wird nun der Fehler nochmals weiter umgeschrieben und

anschließend nach $\alpha^{(m)}$ abgeleitet und Null gesetzt, um den Fehler zu minimieren:

$$
\begin{aligned}
E &= \sum_{i=1}^{N} w_i^{(m)} \exp\left\{-y_i \alpha^{(m)} \hat{f}^{(m)}\right\} \\
&= \sum_{i=1}^{N} w_i^{(m)} e^{-\alpha^{(m)}} \cdot \mathbb{1}\big[y_i = \hat{f}^{(m)}(\boldsymbol{x}_i)\big] + \sum_{i=1}^{N} w_i^{(m)} e^{\alpha^{(m)}} \cdot \mathbb{1}\big[y_i \neq \hat{f}^{(m)}(\boldsymbol{x}_i)\big] \\
&= \sum_{i=1}^{N} w_i^{(m)} e^{-\alpha^{(m)}} \Big(1 - \mathbb{1}\big[y_i \neq \hat{f}^{(m)}(\boldsymbol{x}_i)\big]\Big) + \sum_{i=1}^{N} w_i^{(m)} e^{\alpha^{(m)}} \cdot \mathbb{1}\big[y_i \neq \hat{f}^{(m)}(\boldsymbol{x}_i)\big] \\
&= \sum_{i=1}^{N} w_i^{(m)} e^{-\alpha^{(m)}} - \sum_{i=1}^{N} w_i^{(m)} e^{-\alpha^{(m)}} \cdot \mathbb{1}\big[y_i \neq \hat{f}^{(m)}(\boldsymbol{x}_i)\big] + \sum_{i=1}^{N} w_i^{(m)} e^{\alpha^{(m)}} \cdot \mathbb{1}\big[y_i \neq \hat{f}^{(m)}(\boldsymbol{x}_i)\big] \\
&= \Big(e^{\alpha^{(m)}} - e^{-\alpha^{(m)}}\Big) \underbrace{\sum_{i=1}^{N} w_i^{(m)} \cdot \mathbb{1}\big[y_i \neq \hat{f}^{(m)}(\boldsymbol{x}_i)\big]}_{=\epsilon^{(m)}} + e^{-\alpha^{(m)}} \underbrace{\sum_{i=1}^{N} w_i^{(m)}}_{=1} \\
&= \Big(e^{\alpha^{(m)}} - e^{-\alpha^{(m)}}\Big)\epsilon^{(m)} + e^{-\alpha^{(m)}}
\end{aligned}
$$

Ableiten nach $\alpha^{(m)}$ und Null setzen:

$$
\begin{aligned}
& \frac{\partial E}{\partial \alpha^{(m)}} = \Big(e^{\alpha^{(m)}} + e^{-\alpha^{(m)}}\Big)\epsilon^{(m)} - e^{-\alpha^{(m)}} \stackrel{!}{=} 0 \\
\implies \quad & 0 = e^{\alpha^{(m)}}\epsilon^{(m)} + e^{-\alpha^{(m)}}\epsilon^{(m)} - e^{-\alpha^{(m)}} \\
\iff \quad & e^{\alpha^{(m)}}\epsilon^{(m)} = e^{-\alpha^{(m)}}\Big(1 - \epsilon^{(m)}\Big) \\
\iff \quad & \alpha^{(m)} + \log \epsilon^{(m)} = -\alpha^{(m)} + \log\Big(1 - \epsilon^{(m)}\Big) \\
\iff \quad & \alpha^{(m)} = \frac{1}{2} \log\left(\frac{1 - \epsilon^{(m)}}{\epsilon^{(m)}}\right)
\end{aligned}
$$

Das ergibt genau das Mitspracherecht in AdaBoost.

7.3 Gradienten Boosting

Bei AdaBoost wird der Gradient des Verlusts verwendet, um das Mitbestimmungsrecht Herzuleiten. Als Verallgemeinerung davon werden bei *Gradienten Boosting* nicht die Daten gewichtet neu zum Lernen genutzt, sondern es werden die Residuen trainiert. Der Vorgang ist dabei wie folgt:

1. Trainiere ein einfaches Regressionsmodell $\hat{f}^{(1)}(\boldsymbol{x})$.
2. Trainiere ein weiteres Regressionsmodell $\hat{f}^{(2)}(\boldsymbol{x})$ auf den Residuen $y - \hat{y}$, wobei $\hat{y}$ die vom Modell vorhergesagten Labels sind. Daraus ergibt sich das folgende Regressionsmodell $\hat{F}^{(2)}(\boldsymbol{x}) = \hat{f}^{(1)}(\boldsymbol{x}) + \eta\gamma_2\hat{f}^{(2)}(\boldsymbol{x})$, wobei γ_2 eine Lernrate und $\eta \in (0, 1]$ eine Regularisierungs-Lernrate ist.
3. Anschließend werden weitere Modelle auf den Residuen von $\hat{F}^{(2)}$ trainiert, bis die Daten gut dargestellt werden.

Das finale Modell ist dann gegeben als die gewichtete Summe der nacheinander gelernten Modelle:

$$\hat{F}^{(M)}(\boldsymbol{x}) = \hat{f}^{(1)}(\boldsymbol{x}) + \eta \sum_{m=1}^{M} \hat{f}^{(m)}(\boldsymbol{x})$$

Je näher die Lernrate η auf 1 gesetzt wird, desto schneller geschieht das Lernen und desto höher ist die Gefahr von Überanpassung. Analog kann auch bei AdaBoost eine Lernrate eingeführt werden.

7.3.1 Funktionaler Gradient

Als weitere Verallgemeinerung kann statt der Residuen der negative Gradient einer Verlustfunktion in Richtung des Modells verwendet werden. In diesem Fall werden die nachfolgenden Modelle auf den negativen Wert des Gradienten angepasst. Diese Idee stammt daher, dass der negative (funktionale) Gradient des quadratischen Fehlers gerade die Residuen sind:

$$L = \frac{1}{2} \sum_{i=1}^{N} \big(y_i - \hat{f}(\boldsymbol{x}_i)\big) \quad \Longrightarrow \quad \frac{\partial L}{\partial \hat{f}(\boldsymbol{x}_i)} = -\big(y_i - \hat{f}(\boldsymbol{x}_i)\big)$$

Analog können andere Gradienten Boosting-Verfahren hergeleitet werden, indem ein anderer Verlust verwendet wird. Beispiele sind unter anderem:

- Fehlklassifikation: $L(y, F) = \mathbb{1}\big[y = \operatorname{sign} F\big]$
- Exponentiell/AdaBoost: $L(y, F) = \exp\{-yF\}$
- Binomial: $L(y, F) = \log\big(1 + \exp\{-2yF\}\big)$
- Quadratisch/L_2: $L(y, F) = (y - F)^2$
- Stützvektormaschine: $L(y, F) = y(1 - yF)$

Eine Realisierung von Gradienten Boosting ist *Extreme Gradient Boosting* (xgboost), die Regularisierung etwas mehr betont. Bei der Variante ist auch eine verteilte Berechnung möglich, wodurch der Algorithmus etwas zehn mal schneller ist als auf einer einzelnen Maschine.

8 Probabilistische Graphische Modelle und Stützvektormethode

In diesem Abschnitt werden Probabilistische Graphische Modelle (PGMs), eine allgemeine Methode zur Darstellung von bedingten Wahrscheinlichkeiten, vorgestellt. Als Motivation dient dabei der naive Bayes-Klassifikator, bei dem die exponentielle Komplexität zur Schätzung der Wahrscheinlichkeiten eine starke Einschränkung darstellt. Insbesondere in der statistischen Inferenz, der Wissensgewinnung aus Daten, sind PGMs die Haupttriebkraft, da sie einen strukturierten Weg der Inferenz darstellen.

8.1 (Naiver) Bayes-Klassifikator

Der naive Bayes-Klassifikator ist eine Methode zur Klassifizierung, die nicht nur auf der Likelihood aufbaut, sondern zusätzlich eine A-Priori Wahrscheinlichkeit in die Klassifikation mit einbezieht. Dabei steht der Satz von Bayes,

$$P(y \mid \boldsymbol{x}) \propto P(\boldsymbol{x} \mid y)\, P(y),$$

im Mittelpunkt. Dieser stellt einen Zusammenhang zwischen der A-Posteriori Wahrscheinlichkeit $P(y \mid \boldsymbol{x})$, der Likelihood $P(\boldsymbol{x} \mid y)$ und der A-Priori Wahrscheinlichkeit $P(y)$ her. Die Proportionalitätskonstante $1/P(\boldsymbol{x})$ ist dabei nur von $\boldsymbol{x}$ abhängig und somit für unterschiedliche y, die die Klassen darstellen, gleich. Daher kann dieser i. A. ignoriert werden.

Zur Klassifikation wird nun die A-Posteriori Wahrscheinlichkeit, bzw. das Produkt aus Likelihood und A-Priori Wahrscheinlichkeit, maximiert:

$$y^* = \arg\max_y \; P(y \mid \boldsymbol{x}) = \arg\max_y \; P(\boldsymbol{x} \mid y)\, P(y)$$

Zur Angabe der Konfidenz, also der Wahrscheinlichkeit, dass y^* die korrekte Klasse ist, kann $P(\boldsymbol{x})$ durch Marginalisierung berechnet werden:

$$P(\boldsymbol{x}) = \sum_{i=1}^{k} P(\boldsymbol{x} \mid y_i)$$

Zum „Lernen“ dieses Klassifikators muss also die Likelihood $P(\boldsymbol{x} \mid y)$ geschätzt werden. Dabei nimmt der *naive* Bayes-Klassifikator an, dass die einzelnen Merkmale $x_1, \cdots, x_n$, die einen Datenpunkt $\boldsymbol{x}$ ausmachen, stochastisch unabhängig sind. Unter dieser Annahme ist die Likelihood $P(\boldsymbol{x} \mid y)$ gegeben durch die Faktorisierung

$$P(\boldsymbol{x} \mid y) = \prod_{i=1}^{n} P(x_i \mid y),$$

d. h. die einzelnen Verteilungen können einzeln geschätzt werden.

Der (naive) Bayes-Klassifikator hat dabei drei zentrale Vorteile:

- Es kann Vorwissen in Form der A-Priori Wahrscheinlichkeit verwendet werden.

- Unter den getroffenen Annahmen liefert er optimale Ergebnisse im Sinne der Fehlklassifikationswahrscheinlichkeit $P(y \neq \hat{y})$.
- Die Konfidenz der Klassifikation kann als Wahrscheinlichkeit angegeben werden.

Die Annahme der Unabhängigkeit der Merkmale, gegeben der Klasse, ist jedoch im Allgemeinen nicht korrekt und oftmals falsch! Der naive Bayes-Klassifikator liefert dennoch oft gute Ergebnisse, diese sind jedoch statistisch nicht immer einwandfrei erklärbar.

Ein Problem der Schätzung der Klassenwahrscheinlichkeiten aus Daten ist auch, dass dies (für diskrete Merkmale) einer exponentiellen Laufzeit- und Speicherkomplexität unterliegt. Der Bayes-Klassifikator ist also für eine große Anzahl Merkmale nicht geeignet.

Hinweis: Trotz seines Namens ist der Bayes-Klassifikator keine Methode der Bayes'schen Statistik im Sinne des maschinellen Lernens. Bei Bayes'schen Methoden werden A-Priori Wahrscheinlichkeiten genutzt, um die Parameter eines Modells im Vorhinein einzuschränken, bspw. um die Gewichte eines linearen Klassifikators klein zu halten. Da der Bayes-Klassifikator keine solche Wahrscheinlichkeiten nutzt (und nicht einmal Parameter hat), zählt er nicht zu den Bayes'schen Methoden. Die vorhandene A-Priori Wahrscheinlichkeit trifft direkt Aussagen über die Daten.

8.2 Bayes'sche Netzwerke

Bayes'sche Netzwerke (oder auch *Bayes-Netzwerke*) sind eine Ausprägung von der allgemeinen Klasse der *Probabilistischen Graphischen Modelle*. Die bestehen aus einem qualitativem Teil, dem Graph, der die Abhängigkeiten zwischen den Zufallsvariablen darstellt, und einem quantitativem Teil, den Mengen der Wahrscheinlichkeitsverteilungen. Zusammengenommen bilden sie eine faktorisierte Wahrscheinlichkeitsverteilung als Produkt aller bedingten Wahrscheinlichkeiten. Dabei repräsentiert eine Kante immer eine Abhängigkeit, d. h. geht eine Kante von Zufallsvariable X zu Zufallsvariable Y, so ist Y von X abhängig. Ein Bayes'sches Netz ist dabei immer gerichtet und azyklisch. In Abbildung 8.1 in ein Beispiel für einen solchen Graph gegeben, der die Verbundwahrscheinlichkeit

$$P(A, B, C, D, E) = P(A)\, P(B)\, P(C \mid A, B)\, P(D \mid B)\, P(E \mid C)$$

charakterisiert. Im Allgemeinen ist die Verbundwahrscheinlichkeit eines Bayes'schen Netzwerks mit den Zufallsvariablen $\boldsymbol{X} = \{X_1, \cdots, X_n\}$ gegeben durch das Produkt

$$P(\boldsymbol{X}) = \prod_{i=1}^{n} P\big(X_i \mid \mathrm{Pa}(X_i)\big), \tag{8.1}$$

wobei $\mathrm{Pa}(X_i)$ die Menge der Elternknoten von X_i darstellt. Im obigen Beispiel ist also $\mathrm{Pa}(C) = \{A, B\}$.

8.2.1 Variablenelimination

Die *Variablenelimination* ein Algorithmus zur Berechnung von marginalisierten Abfragen innerhalb eines Bayes'schen Netzwerkes. Die Grundlagen für den Algorithmus bietet die allgemeine Form der Marginalisierung

$$P(\boldsymbol{Y}) = \sum_{X_i \notin \boldsymbol{Y}}^{\text{marg}} P(X_1, \cdots, X_i, \cdots X_n),$$

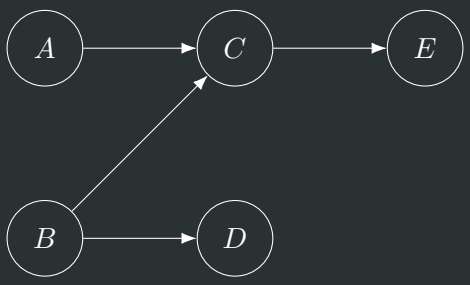

Abbildung 8.1: Beispiel für ein Bayes'sches Netzwerk mit den Zufallsvariablen A, B, C, D und E.

wobei die Summe über eine Zufallsvariable die Summe über die einzelnen Ausprägungen der entsprechenden Zufallsvariable entspricht (dies wird durch das „marg“ über der Summe angezeigt). Wird die Faktorisierung der Verbundwahrscheinlichkeit (8.1) eingesetzt,

$$P(\boldsymbol{Y}) = \sum_{X_i \notin \boldsymbol{Y}}^{\text{marg}} \prod_{j=1}^{n} P\big(X_j \,|\, \text{Pa}(X_j)\big),$$

so sind Summe und Produkt vertauschbar, sofern der entsprechende Summand in einem Faktor nicht auftaucht. Dann kann dieser Faktor vor die Summe gezogen werden: $\sum_i (2i) = 2 \sum_i i$. Es muss zur Berechnung einer marginalisierten Wahrscheinlichkeit also nicht die gesamte Verbundwahrscheinlichkeit berechnet werden sondern es reicht aus, mit den bedingten Wahrscheinlichkeiten zur arbeiten und nach und nach zu summieren. Durch die Summation wird die entsprechende Variable aus dem Produkt eliminiert, was den Namen des Algorithmus erklärt.

Trotz dass dieser Algorithmus effizienter ist als die gesamte Verbundwahrscheinlichkeit, ist das Schlussfolgern in Bayes'schen NP-schwer.

Beispiel Es wird das Bayes-Netz aus Abbildung 8.2 betrachtet. Die Verbundwahrscheinlichkeit ist demnach gegeben durch

$$P(G, A, R, V, K, L, S) = P(G)\, P(A)\, P(R \,|\, G, A)\, P(V \,|\, A)\, P(K \,|\, R, V)\, P(L \,|\, K)\, P(S \,|\, K).$$

Daraus werden nun nacheinander die Variablen eliminiert, um $P(L)$ zu berechnen. Im ersten Schritt bietet es sich an, die Verbundwahrscheinlichkeit $P(R, G \,|\, A)$ zu berechnen und G zu eliminieren:

$$P(R, G \,|\, A) = P(G)\, P(R \,|\, G, A) \qquad \Longrightarrow \qquad P(R \,|\, A) = \sum_{G}^{\text{marg}} P(R, G \,|\, A)$$

Anschließend kann A eliminiert werden:

$$P(A, R, V) = P(A)\, P(V \,|\, A)\, P(R \,|\, A) \qquad \Longrightarrow \qquad P(R, V) = \sum_{A}^{\text{marg}} P(A, R, V)$$

Weiter geht es mit der Elimination von R und V:

$$P(K, R, V) = P(K \,|\, R, V)\, P(R, V) \qquad \Longrightarrow \qquad P(K) = \sum_{R,V}^{\text{marg}} P(K, R, V)$$

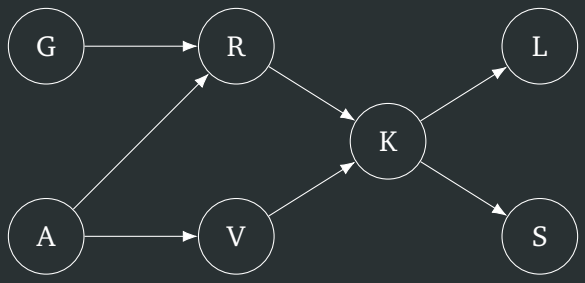

Abbildung 8.2: Bayes-Netz als Beispiel zur Variablenelimination.

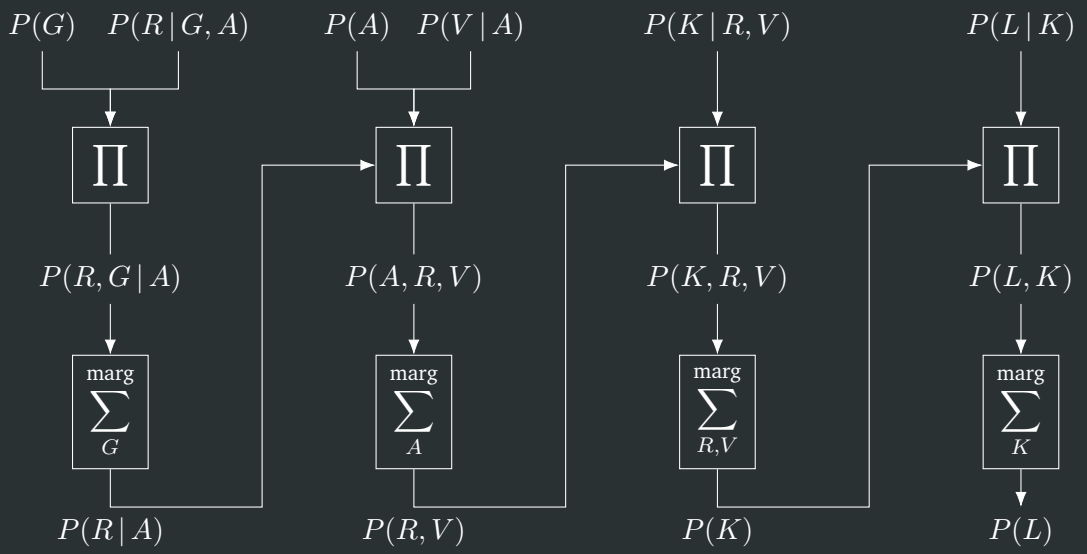

Abbildung 8.3: Durchführung der Variablenelimination für das Bayes-Netz aus Abbildung 8.2.

Durch die finale Elimination von K wird $P(L)$ berechnet:

$$P(L, K) = P(L \mid K)\, P(K) \qquad \Longrightarrow \qquad P(L) = \sum_K^{\text{marg}} P(L, K)$$

Diese Berechnung kann auch grafisch dargestellt werden, siehe Abbildung 8.3.

8.3 Parameterschätzung

Ein Problem von Bayes'schen Netzen ist, dass diese zum Schlussfolgern zunächst erstellt werden müssen (sowohl der qualitative als auch der quantitative Teil). Eine Möglichkeit ist, die Netze durch Experten aufbauen zu lassen, allerdings ist dies oft sehr teuer oder sogar unmöglich. Eine andere Möglichkeit besteht in der Nutzung von Algorithmen des maschinellen Lernens, also der Nutzung von Daten, zum Aufbau des Netzwerkes. Dieser Abschnitt beschäftigt sich dabei mit zwei möglichen Fällen – vollständige und unvollständige Daten.

8.3.1 Vollständige Daten: Maximum Likelihood

Eine Möglichkeit der Parameterschätzung bei vollständigen Daten sind Maximum Likelihood-Schätzer. Dabei wird angenommen, dass der qualitative Teil des Bayes-Netzes gegeben ist und nur der quantitative Teil, also

die Wahrscheinlichkeiten selbst, geschätzt werden müssen. Dazu werden einfach die Häufigkeiten gezählt und die Wahrscheinlichkeiten so abgeschätzt.

Sei $\theta_{x_i|\mathrm{Pa}_i(X)}$ der Parameter, der die Wahrscheinlichkeit $P\big(X = x_i \,|\, \mathrm{Pa}(X) = \mathrm{Pa}_i(X)\big)$ beschreibt, wobei X die betrachtete Zufallsvariable, $\mathrm{Pa}(X)$ die übergeordneten Zufallsvariablen und x_i und $\mathrm{Pa}_i(X)$ ihre entsprechenden (bzw. eine der) Ausprägungen sind. Sei dann $N\big(x_i, \mathrm{Pa}_i(X)\big)$ die Anzahl der Beispiele mit dieser Ausprägung und sei $N\big(\mathrm{Pa}_i(X)\big)$ die Anzahl der Beispiele mit der Ausprägung der übergeordneten Zufallsvariablen. Dann wird der Parameter $\theta_{x_i|\mathrm{Pa}_i(X)}$ durch

$$\hat{\theta}_{x_i|\mathrm{Pa}_i(X)} = \frac{N\big(x_i, \mathrm{Pa}_i(X)\big)}{N\big(\mathrm{Pa}_i(X)\big)}$$

abgeschätzt. Dieser Schätzer ist asymptotisch konsistent und kann auch für große Datenmengen berechnet werden.

8.3.2 Unvollständige Daten: Expectation Maximization (EM)

In der Praxis kommt es leider häufiger vor, dass nicht alle Daten vorhanden sind. Dies können bspw. einzelne Merkmale sein, die komplett Fehlen oder einzelne Datenpunkte, bei denen nicht alle Merkmale vorhanden sind. Ein Algorithmus, bzw. eine Familie von Algorithmen, um damit umzugehen sind *Expectation Maximization* (EM) Algorithmen. Diese basieren auf der Intuition, dass, wenn die wahren Zählungen bekannt wären, die Parameter direkt bestimmt werden könnten. Die wahren Zählungen sind jedoch nicht bekannt. Daher iteriert der EM-Algorithmus zwischen zwei Schritten, dem E- und dem M-Schritt:

E-Schritt Die Zählungen werden durch statistisches Schlussfolgern vervollständigt, d. h. die Daten werden aus einem vorhandenen Modell abgeleitet.

M-Schritt Die vervollständigten Zählungen werden als korrekt angenommen und das Modell wird verbessert.

Diese zwei Schritte werden bis zur Konvergenz wiederholt.

8.4 Diskriminative Ansätze

Der (naive) Bayes-Klassifikator gehört, da die darunterliegenden Verteilungen geschätzt werden, zu den sogenannten *generativen* Modelle. Im Gegensatz dazu stehen die *diskriminativen*, bei denen keine Wahrscheinlichkeitsverteilung, sondern direkt eine Entscheidungsregel, geschätzt wird. Beispielsweise gilt bei einem Bayes-Klassifikator die Entscheidungsregel, dass ein Datenpunkt $\boldsymbol{x}$ der Klasse y_1 angehört, wenn

$$P(\boldsymbol{x} \,|\, y_1)\, P(y_1) > P(\boldsymbol{x} \,|\, y_2)\, P(y_2)$$

gilt. Daraus motiviert existiert also ein Schwellenwert $\boldsymbol{x}_0$, bei dem

$$P(\boldsymbol{x}_0 \,|\, y_1)\, P(y_1) = P(\boldsymbol{x}_0 \,|\, y_2)\, P(y_2)$$

gilt. Dieser Schwellenwert stellt ein lineares diskriminatives Modell dar, welches zur Klassifikation dient. Bei einem Merkmal ist der Diskriminator ein einfacher Schwellenwert, bei zwei Merkmalen eine Gerade, bei drei Merkmalen eine Ebene und bei n Merkmalen eine eine n-dimensionale Hyperebene.

Diskriminative Modelle versuchen direkt den Schwellenwert zu finden ohne die darunterliegende Verteilung zu schätzen. Dabei wird, da weder die echte Funktion noch die echte Verteilung bekannt ist, Auch die

empirische Risikominimierung (ERM) zurück gegriffen. Das bedeutet, vereinfacht, dass eine hinreichend große Lernmenge genommen und der Fehler auf dieser minimiert wird. Dies führt allerdings auf das Problem, dass potentiell mehrere Funktionen für den minimalen Fehler existieren. Außerdem kann immer das Problem der Überanpassung auftreten, sodass verrauschte oder neue Daten zu falschen Ergebnissen führen.

8.4.1 Stützvektormethode

Die *Stützvektormethode* ist ein Weg zur Findung einer optimalen Trennung bei der Klassifikation. Dabei wird ein Ähnlichkeitsmaß $k(\boldsymbol{x}, \boldsymbol{x}')$ verwendet, welches zwei Trainingspunkten $\boldsymbol{x}$ und $\boldsymbol{x}'$ vergleicht. Dieses Ähnlichkeitsmaß wird auch *Kernel* oder *Kernfunktion* genannt. Ein Beispiel ist das Skalarprodukt $k(\boldsymbol{x}, \boldsymbol{x}') = \langle \boldsymbol{x} | \boldsymbol{x}' \rangle$. Es ist auch möglich, zusätzliche eine Transformation $\phi : X \to \mathcal{H}$ vom Merkmalsraum X in einen Vektorraum $\mathcal{H}$ zu verwenden, bspw. wenn der Merkmalsraum keinen Vektorraum darstellt. Es kann aber auch hilfreich sein, eine (nichtlineare) Transformation anzuwenden, bevor die Stützvektormethode angewandt wird.

Ein einfacher diskriminativer Ansatz ist z. B. die Zuweisung eines neuen Datenpunkts zu der Klasse, derer Durchschnitt am nächsten an dem neuen Punkt liegt. Dazu werden die Klassenzentroiden

$$\boldsymbol{c}_+ = \frac{1}{m_+} \sum_{\substack{i=1 \\ y_i = +1}}^{n} \boldsymbol{x}_i \qquad\qquad \boldsymbol{c}_- = \frac{1}{m_-} \sum_{\substack{i=1 \\ y_i = -1}}^{n} \boldsymbol{x}_i$$

verwendet, die jeweils den Durchschnitt der Datenpunkte von Klasse $+1$ und -1 darstellen. Zwischen den beiden Zentroiden liegt dann der Punkt $\boldsymbol{c} = \frac{1}{2}(\boldsymbol{c}_+ + \boldsymbol{c}_-)$, der zur Vorhersage verwendet werden kann. Dazu wird der Winkel zwischen $\boldsymbol{w} = \boldsymbol{c}_+ - \boldsymbol{c}_-$ und $\boldsymbol{x} - \boldsymbol{c}$ verwendet, wobei $\boldsymbol{x}$ der zuzuweisende Datenpunkt ist:

$$\hat{y} = \operatorname{sign} \langle \boldsymbol{w} | \boldsymbol{x} - \boldsymbol{c} \rangle$$

Intuition: Der Vektor $\boldsymbol{w}$ verbindet genau die Zentroiden $\boldsymbol{c}_+$ und $\boldsymbol{c}_-$, während der Vektor $\boldsymbol{c}$ auf den Mittelpunkt zwischen diesen zeigt. Durch die Verschiebung $\boldsymbol{x} - \boldsymbol{c}$ des neuen Datenpunkts wird der Vektor in den Mittelpunkt der beiden Zentroiden verschoben. Das Skalarprodukt $\langle \boldsymbol{w} | \boldsymbol{x} - \boldsymbol{c} \rangle$ ist proportional zum Kosinus des Winkels φ zwischen den Vektoren:

$$\langle \boldsymbol{w} | \boldsymbol{x} - \boldsymbol{c} \rangle \propto \cos(\boldsymbol{w},\, \boldsymbol{x} - \boldsymbol{c}) = \frac{\langle \boldsymbol{w} | \boldsymbol{x} - \boldsymbol{c} \rangle}{|\boldsymbol{w}|\, |\boldsymbol{x} - \boldsymbol{c}|}$$

Dabei ist der Kosinus genau dann negativ, wenn der Winkel im Bereich $(-\pi/2, +\pi/2)$ liegt. Dies entspricht genau der Region, die der Klasse -1 zugewiesen wird. Dieser Ansatz entspricht schon fast der Stützvektormethode, jedoch ist die Wahl des Mittelpunkts zwischen den beiden Klassen zur Zuweisung der Klassen nicht optimal. Der nächste Abschnitt wird sich mit der optimalen Wahl eines Hyperebene beschäftigen, die die beiden Klassen trennt.

Optimalität der Hyperebene

Eine Hyperebene wird als optimal bezeichnet, wenn der Abstand zu dem nächsten positivem/negativem Beispiel maximal ist. existiert für jeden linear trennbaren Datensatz[1] eine eindeutig bestimmte optimale Hyperebene, die es zu finden gilt. Seien $\boldsymbol{w}$ die Gewichte der Hyperebene, $\boldsymbol{x}$ die Merkmale (bzw. die Werte der Features, wenn eine Feature-Transformation verwendet wird) und b ein Bias. Dann lautet das Modell

[1] An sich müssen die Daten nicht selbst linear trennbar sein. Es ist ausreichend, wenn sie in dem gewählten Feature-Raum, der durch eine Feature-Funktion ϕ aufgespannt wird, linear separierbar sind.

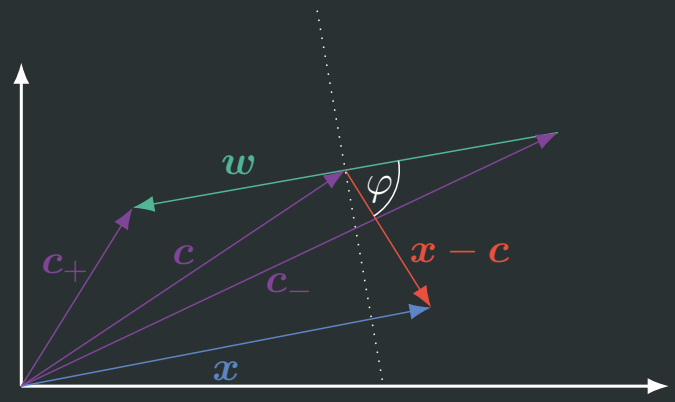

Abbildung 8.4: Illustration eines einfachen Klassifikationsalgorithmus.

$\hat{f}(\boldsymbol{x}) = \langle \boldsymbol{w} | \boldsymbol{x} \rangle + b$ und das Vorzeichen des Modells wird als Klassifikation verwendet. Ist $\hat{f}(\boldsymbol{x}_0) = 0$, so liegt der Datenpunkt $\boldsymbol{x}_0$ direkt auf der Hyperebene und es kann keine Entscheidung getroffen werden.

Um nun die optimale Hyperebene zu finden wird verlangt, dass für die am nächsten an der Hyperebene liegenden Beispiele $|\langle \boldsymbol{w} | \boldsymbol{x} \rangle + b| = 1$ gilt. Dies ergibt zwei neue parallele Hyperebenen, die durch die nächstliegenden Punkte gehen. Sei nun x_- ein Punkt auf der Hyperebene $\langle \boldsymbol{w} | \boldsymbol{x} \rangle + b = -1$. Da $\boldsymbol{w}$ der Normalvektor zu dieser Hyperebene ist, ist auch $\boldsymbol{w}/\|\boldsymbol{w}\|_2$ ein Normalvektor zu der Hyperebene, wobei letzterer auf die Länge 1 normiert ist. Sei d die Distanz zwischen der positiven und der negativen Hyperebene. Dann liegt der Punkt $\boldsymbol{x}_+ = \boldsymbol{x}_- + d\boldsymbol{w}/\|\boldsymbol{w}\|_2$ genau auf der Hyperebene $\langle \boldsymbol{w} | \boldsymbol{x} \rangle + b = +1$. Daraus folgt:

$$\langle \boldsymbol{w} | \boldsymbol{x}_+ \rangle + b = 1 \quad \Longleftrightarrow \quad \left\langle \boldsymbol{w} \middle| \boldsymbol{x}_- + d\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|_2} \right\rangle + b = 1 \quad \Longleftrightarrow \quad \langle \boldsymbol{w} | \boldsymbol{x}_- \rangle + d\frac{\langle \boldsymbol{w} | \boldsymbol{w} \rangle}{\|\boldsymbol{w}\|_2} + b = 1$$

$$\Longleftrightarrow \quad \langle \boldsymbol{w} | \boldsymbol{x}_- \rangle + d\frac{\|\boldsymbol{w}\|_2^2}{\|\boldsymbol{w}\|_2} + b = 1 \quad \Longleftrightarrow \quad \underbrace{\langle \boldsymbol{w} | \boldsymbol{x}_- \rangle + b}_{=-1} + d\,\|\boldsymbol{w}\|_2 = 1 \quad \Longleftrightarrow \quad d = \frac{2}{\|\boldsymbol{w}\|_2}$$

Es gilt also $\|\boldsymbol{w}\|_2 = 2/d$, wobei d die Breite des optimalen Margins ist.

Zur Maximierung des Margins d muss also die Norm der Gewichte, $\|\boldsymbol{w}\|_2$, maximiert werden. Da jedoch auch die Trainingsdaten richtig klassifiziert werden sollen, müssen die Nebenbedingungen

$$\langle \boldsymbol{w} | \boldsymbol{x}_i \rangle + b \geq +1 \qquad \text{und} \qquad \langle \boldsymbol{w} | \boldsymbol{x}_i \rangle + b \leq -1$$

gelten, wobei die erste für Beispiele mit $y_i = +1$ und die zweite für Beispiele mit $y_i = -1$ herangezogen werden. Diese Nebenbedingungen lassen sich kombinieren zu der Bedingung

$$y_i(\langle \boldsymbol{w} | \boldsymbol{x}_i \rangle + b) - 1 \geq 0.$$

Der nächste Abschnitt wird die Lösung dieses Optimierungsproblems behandeln.

Optimierungsproblem

Wie im vorangehenden Abschnitt gezeigt ist das zu lösende Optimierungsproblem für die Stützvektormethode gegeben durch[2]:

$$\begin{aligned} \min_{\boldsymbol{w},b} \quad & \frac{1}{2}\|\boldsymbol{w}\|_2^2 \\ \text{sodass} \quad & y_i(\langle \boldsymbol{w}|\boldsymbol{x}_i\rangle + b) - 1 \geq 0, \quad i = 1, \cdots, N \end{aligned}$$

Dies ist ein konvexes Optimierungsproblem, dass eindeutig in $\mathcal{O}(N^3)$ lösbar ist, wobei N die Anzahl Trainingsbeispiele (und damit Nebenbedingungen) darstellt.

Statt des primalen Problems ist es jedoch hilfreich, dass duale Probleme zu lösen, da dies mehr Einsichten in die Lösung offenbart. Dazu wird zunächst die Lagrange-Funktion

$$L_P(\boldsymbol{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\|\boldsymbol{w}\|_2^2 - \sum_{i=1}^{M} \alpha_i \big(y_i(\langle \boldsymbol{w}|\boldsymbol{x}_i\rangle + b) - 1\big) \tag{8.2}$$

aufgestellt. Um das duale Problem zu erhalten, wird diese Funktion nach $\boldsymbol{w}$ und b abgeleitet und die Ableitungen Null gesetzt. Das ergibt:

$$\frac{\partial L_P}{\partial \boldsymbol{w}} = \boldsymbol{w} - \sum_{i=1}^{N} \alpha_i y_i \boldsymbol{x}_i \stackrel{!}{=} \boldsymbol{0} \quad \Longrightarrow \quad \boldsymbol{w}^* = \sum_{i=1}^{N} \alpha_i y_i \boldsymbol{x}_i$$

$$\frac{\partial L_P}{\partial b} = -\sum_{i=1}^{N} \alpha_i y_i \stackrel{!}{=} 0$$

Werden diese Bedingungen zurück in (8.2) eingesetzt, ergibt sich die folgende duale Lagrange-Funktion:

$$\begin{aligned} L_D(\boldsymbol{\alpha}) &= \frac{1}{2}\|\boldsymbol{w}\|_2^2 - \sum_{i=1}^{M} \alpha_i \big(y_i(\langle \boldsymbol{w}|\boldsymbol{x}_i\rangle + b) - 1\big) \\ &= \frac{1}{2}\|\boldsymbol{w}\|_2^2 - \sum_{i=1}^{M} \alpha_i y_i \langle \boldsymbol{w}|\boldsymbol{x}_i\rangle - b \underbrace{\sum_{i=1}^{M} \alpha_i y_i}_{=0} + \sum_{i=1}^{M} \alpha_i \\ &= \frac{1}{2}\langle \boldsymbol{w}|\boldsymbol{w}\rangle - \sum_{i=1}^{M} \alpha_i y_i \langle \boldsymbol{w}|\boldsymbol{x}_i\rangle + \sum_{i=1}^{M} \alpha_i \\ &= \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \langle \boldsymbol{x}_i|\boldsymbol{x}_j\rangle - \sum_{i=1}^{M}\sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \langle \boldsymbol{x}_i|\boldsymbol{x}_j\rangle + \sum_{i=1}^{M} \alpha_i \\ &= \sum_{i=1}^{M} \alpha_i - \frac{1}{2}\sum_{i=1}^{M}\sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \langle \boldsymbol{x}_i|\boldsymbol{x}_j\rangle \end{aligned}$$

Die Maximierung dieser Funktion bzgl. $\boldsymbol{\alpha}$ unter den Nebenbedingungen $\alpha_i \geq 0$, $i = 1, \cdots, N$ und $\sum_{i=1}^{N} \alpha_i y_i = 0$ liefert die duale Lösung und somit auch die Lösung für das Optimierungsproblem der Stützvektormaschine.

[2] Hier wird statt der 2-Norm die halbierte quadrierte 2-Norm verwendet, da dies das optimale $\boldsymbol{w}$ nicht ändert, aber zu einer angenehmeren Lösung des Optimierungsproblems führt. Es muss beispielsweise keine Wurzelfunktion differenziert werden.

Die optimalen Gewichte

$$\boldsymbol{w}^* = \sum_{i=1}^{N} \alpha_i y_i \boldsymbol{x}_i$$

sind dabei eine Linearkombination aus den Trainingsdaten. Ist der Lagrange-Multiplikator α_i für den i-ten Datenpunkt Null, so liegt dieser innerhalb einer Entscheidungsregion ist der Multiplikator größer Null, so liegt das Beispiel auf einer der beiden Hyperebenen und stellt somit einen *Stützvektor* dar.

8.4.2 Nicht linear trennbare Daten

Es ist möglich, dass die Daten eines Problems nicht linear separierbar sind, d. h. es kann keine Hyperebene gefunden werden, die die Daten ohne Verlust in zwei Klassen einteilt. Liegt das Problem nur darin, dass die Daten verrauscht sind, aber größtenteils einer linearen Trennung folgen, so können *Schlupfvariablen* (Englisch: *Slack Variables*) $\boldsymbol{\xi}$ eingeführt werden, die das Problem relaxieren. Das Optimierungsproblem lautet dann

$$\begin{aligned} \min_{\boldsymbol{w},b,\boldsymbol{\xi}} \quad & \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C\sum_{i=1}^{N}\xi_i \\ \text{sodass} \quad & y_i(\langle \boldsymbol{w}|\boldsymbol{x}_i\rangle + b) - 1 \geq 1 - \xi_i, \quad i = 1, \cdots, N \\ & \xi_i \geq 0, \quad i = 1, \cdots, N \end{aligned}$$

wobei der Faktor C zur Bestrafung der Variablen eingeführt wird. Ein hoher C-Faktor führt somit dazu, dass weniger Schlupf erlaubt wird.

Kernel-Trick

Dies ist jedoch keine Lösung für Daten, die tatsächlich nicht linear trennbar sind. Hier gibt es einerseits die Möglichkeit einer Feature-Transformation vom Merkmalsraum X einen Feature-Raum $\mathcal{H}$ vorzunehmen mit einer Funktion $\phi : X \to \mathcal{H}$, andererseits ist es Möglich eine Kernfunktion (Englisch: *Kernel*)

$$k(\boldsymbol{x}, \boldsymbol{x}') = \big\langle \phi(\boldsymbol{x}) \big| \phi(\boldsymbol{x}') \big\rangle$$

zu definieren, die direkt das Skalarprodukt im Feature-Raum berechnet. Dabei ist es allerdings nicht zwingend notwendig, die Transformation in den Feature-Raum tatsächlich zu berechnen! Es reicht aus eine Funktion zu finden, die das Skalarprodukt berechnet. Dafür muss die Funktion die folgende Bedingung erfüllen (*Mercer's Theorem*): Eine Funktion $k : X \times X \to \mathbb{R}$ ist genau dann eine (positiv definite) Kernfunktion, wenn die Gram-Matrix $\boldsymbol{K} = \big(k(\boldsymbol{x}_i, \boldsymbol{x}_j)\big)_{i,j=1,\cdots,N}$ für alle möglichen Datenpunkte symmetrisch und positiv semi-definit ist. Umgekehrt gilt auch, dass für jede Kernfunktion ein Feature-Raum existiert, in der die Kernfunktion das Skalarprodukt berechnet. Dieser Feature-Raum kann jedoch auch unendlich-dimensional sein. Das bedeutet durch eine Kernfunktion kann eine implizite Transformation in einen hoch- und sogar unendlich-dimensionalen Feature-Raum berechnet werden, in dem die Daten linear separierbar sind! Dies ist bekannt als der *Kernel-Trick*.

9 Cluster-Analyse

In der *Cluster-Analyse* wird versucht, Muster in Daten zu erkennen, ohne dass diese Labels aufweisen. Daher gehören diese Verfahren in die Gruppe der unüberwachten Lernmethoden. Dabei gibt es grundsätzlich zwei Ansätze: Partitionierung und Hierarchie. Bei der Partitionierung werden die Daten direkt in einem Schritt aufgeteilt, bei dem Ansatz der Hierarchie werden viele kleine Partitionierungen vorgenommen und ein Baum aufgebaut, bei dem die Verzweigungen die Cluster und die Blätter die Datenpunkte darstellen. Bei beiden Ansätzen wird, wie schon bei kNN, eine Distanzfunktion $D(A, B)$ benötigt, die die folgenden Eigenschaften erfüllt:

- Symmetrie: $D(A, B) = D(B, A)$
- Dreiecksungleichung: $D(A, B) + D(B, C) \geq D(A, C)$
- Positive Definitheit: $D(A, B) \geq 0$ und $D(A, B) = 0$ gdw. $A = B$
- Reflexivität: $D(A, A) = 0$

Die Bedingung der Reflexivität und $D(A, B) \geq 0$ können aus der Definition ausgeklammert werden, da diese bereits aus den restlichen Axiomen folgen. Zur Übersichtlichkeit bleiben sie jedoch inkludiert. Es ist jedoch auch mit einer sehr guten Distanzfunktion nicht klar, wie die Distanz zwischen einem Datenpunkt und einem Cluster an Datenpunkten definiert wird. Dabei gibt es grundlegend vier Ansätze:

Single Linkage Es wird die Distanz zu den Punkten genutzt, die sich (in dem entsprechenden Cluster) am nächsten zum betrachteten Datenpunkt befinden.

Complete Linkage Es wird die Distanz zu den Punkten genutzt, die sich (in dem entsprechenden Cluster) am weitesten entfernt vom betrachteten Datenpunkt befinden.

Group Average Linkage Es wird der Durchschnitt der Distanzen zu allen Punkten (in dem entsprechenden Cluster) genutzt.

Wards Linkage Es wird versucht, die Varianz zwischen den Clustern zu minimieren.

Dabei beziehen sich die Aussagen „am nächsten“ und „am weitesten entfernt“ wiederum selbst auf das genutzte Distanzmaß.

In den folgenden Abschnitten werden nun konkrete Verfahren sowie Darstellungsweisen zur Cluster-Analyse vorgestellt. Dabei ist der Erfolg eines Verfahrens hängt sehr stark mit der Auswahl eines guten Distanzmaßes zusammen.

9.1 Dendrogramme und Hierarchische Cluster-Analyse

In einem *Dendrogramm* wird eine vorliegende hierarchische Aufteilung in Cluster als Baum dargestellt. Dabei beschreiben die Blätter einzelne Datenpunkte und die Verzweigungen können als Cluster angesehen werden.

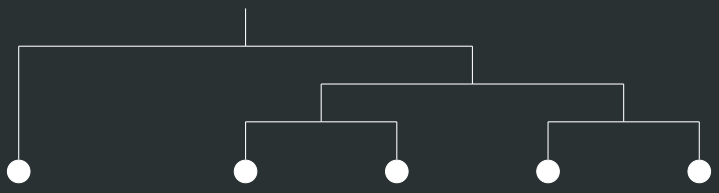

Abbildung 9.1: Beispiel für ein einfaches Dendrogramm.

Ein einfaches Dendrogramm ist in Abbildung 9.1 gezeigt. Für ein Dendrogramm mit n Blättern, d. h. für n Trainingsbeispiele, gibt es

$$\frac{(2n-3)!}{2^{n-2}(n-2)!}$$

unterschiedliche Möglichkeiten zur Aufteilung. Da dies viel zu viele sind, um sämtliche Aufteilungen zu testen, müssen Heuristiken verwendet werden. Dazu gibt es zwei wesentliche Ansätze:

Bottom-Up Zu Beginn befindet sicher jeder Datenpunkt in einem eigenen Cluster. Anschließend werden die beiden ähnlichsten Cluster gesucht und vereinigt, was eine Verzweigung im Dendrogramm darstellt. Dies wird so lange wiederholt, bis es nur noch einen Cluster – den gesamten Datensatz – gibt. Diese Ansätze werden auch *agglomerativ* genannt.

Top-Down Zu Beginn befinden sich alle Datenpunkte im gleichen Cluster. Anschließend wird die bestmögliche Trennung gesucht und die Datenpunkte in mehrere Cluster aufgeteilt, d. h. es wird eine Verzweigung im Dendrogramm eingeführt. Dies wird so lange wiederholt, bis sich jeder Datenpunkt in einem eigenen Cluster befindet, d. h. ein Blatt im Dendrogramm bildet. Diese Ansätze werden auch *aufteilend* genannt.

9.1.1 (Vorgetäuschte) Strukturen, Anzahl Cluster und Ausreißer

Die durch ein Dendrogramm erzeugte Struktur innerhalb der Daten kann sowohl Strukturen aufdecken, allerdings auch vortäuschen. In Abbildung 9.2 ist ein Dendrogramm basierend auf den Flaggen von Australien, St. Helena, Anguilla, Südgeorgien, Vereinigtes Königreich, Serbien, Frankreich, Niger, Indien, Irland und Brasilien gegeben. Dabei ist auffällig, dass es eine enge Gruppe um Länder gibt, die aus den ehemaligen britischen Kolonien hervorgegangen sind. Jedoch wird auch eine Nähe zwischen bspw. Niger, Indien und Irland angedeutet, die so nicht existiert.

Ein Dendrogramm kann über diese Strukturerkennung auch bei anderen Fragestellungen helfen, bspw. der Frage danach, wie viele Cluster es in den Daten gibt: Gibt es zwei (oder mehr) stark getrennte Teilbäume, so legt dies nahe, dass es ebenso viele Cluster in den Daten gibt. Auch legt ein einzelner Ast (wie im Beispiel in Abbildung 9.2 Brasilien) nahe, dass es sich um einen Ausreißer handelt.

9.2 Partitionierung und K-Means

Bei Partitionierungsclustering wird jedes Objekt genau einem Cluster zugeordnet und es gibt keine Überlappung oder Hierarchie der Cluster. Ein Algorithmus der dies umsetzt ist *K-Means*, wobei das K die Anzahl der Cluster im Vorhinein festlegt. Bei diesem Algorithmus werden die Mittelwerte in einem Cluster betrachtet und jeder Datenpunkt dem Cluster zugeordnet, dessen Mittelpunkt am nächsten (bzgl. irgendeines Abstandsmaßes) ist.

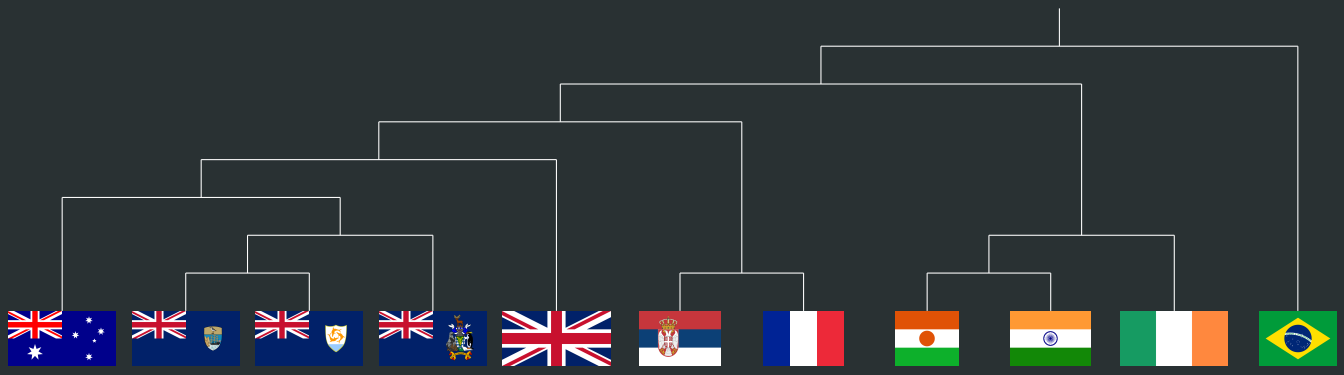

Abbildung 9.2: Dendrogramm für einige Flaggen.

Anschließend werden die Mittelwerte aus den Zuordnungen neu gebildet und der Algorithmus beginnt von vorn. Dies ist in Algorithmus 4 skizziert.

Trotz seiner Einfachheit ist K-Means ein sehr mächtiges Clusteringverfahren, welches Strukturen in den Daten findet. Dabei ist ein gutes Clustering mit kleinem K oftmals besser als ein schlechtes Clustering mit großem K, d. h. zum sinnvollen finden von Strukturen müssen nicht immer viele Strukturen gefunden werden. K-Means hat jedoch starke Probleme, wenn die Cluster unterschiedlich groß, dicht oder nicht sphärische sind. Außerdem ergeben sich Probleme bei Ausreißern, leeren Clustern (diese können nicht gefunden werden) und hochdimensionalen Daten, da hier exponentiell viele Daten benötigt werden. Auch ist K-Means stark von der Initialisierung der Zentroide abhängig. Die wichtigsten Probleme von K-Means sind also:

Initialisierung Zur Initialisierung gibt es im Grunde drei Möglichkeiten: Educated Guess, Zufällig und K-Means++. Bei ersterem wird die Initialisierung aufgrund einer Ahnung gesetzt, wo die Zentroide sein könnten. Im zweiten Fall werden die Zentroide rein zufällig gewählt, was zu unterschiedlichen Ergebnissen führen kann. Bei letzterer Methode werden die Zentroide weit von den Daten entfernt initialisiert, was zu besseren Ergebnissen führt als eine zufällige Initialisierung.

Clusteranzahl Werden die Anzahl von Clustern, also der K-Wert, falsch (zu klein/groß) gewählt, so findet K-Means eventuell kein aussagekräftiges Clustering.

Nicht-sphärische Cluster Wird der euklidische Abstand als Distanzmaß verwendet, so können nur sphärische Cluster gefunden werden. Andere Abstandsmaße können dem entgegen wirken, das Problem bleibt allerdings weiterhin bestehen, nur die Definition von „sphärisch“ ändert sich.

Cluster mit ungleicher Varianz K-Means nimmt an, dass alle Cluster die gleiche Varianz vorweisen. Ist dies nicht der Fall, wird K-Means falsche Cluster finden oder die Daten den Clustern falsch zuordnen.

Erzwungene Clusterbildung K-Means wird immer genau K Cluster finden, auch wenn keine Cluster vorhanden sind. Dieses Phänomen sollte bedacht werden, wenn K-Means verwendet wird.

Möglichkeiten zur Bekämpfung dieser Schwierigkeiten sind z. B. mehrfaches Durchführen des Clusterings mit unterschiedlichen Initialisierungen, „Over-Clustering“ und Nachverarbeitung zum Zusammenfügen der

Cluster und probabilistische oder Kernel-Varianten. Andere Verfahren zur Cluster-Analyse sind z. B. Spectral Clustering, Random Projections, Cluster-Analyse mit Randbedingungen, DBSCAN, Bi-Clustering, LDA und Mean Shift Clustering.

Algorithmus 4 : K-Means

1 Initialisiere jeden Zentroid $\bar{\boldsymbol{x}}_i$, $i = 1, \cdots, K$, bspw. zufällig.
2 **while** *nicht konvergiert* **do**
3 Ordne jeden Datenpunkt dem Cluster zu, dessen Mittelwert am nächsten liegt.
4 Ersetze die Zentroide durch die Mittelwerte der den Clustern zugeordneten Datenpunkte.

10 Tiefes Lernen und Faltende Neuronale Netzwerke

Dieses Kapitel behandelt das *tiefe Lernen*, eine Art des maschinellen Lernens bei dem sogenannte *tiefe Neuronale Netzwerke* verwendet werden. Diese bauen auf der einfachen Idee des Perzeptrons (siehe Unterabschnitt 1.1.1) auf und schichten mehrere Perzeptron-Modelle aufeinander. Dieses Kapitel wird nur sehr knapp in die Thematik einführen, eine intensivere Behandlung ist in der Zusammenfassung von „Deep Learning: Architectures and Methods“[1] zu finden.

Die Idee der mehrschichtigen Modelle ist, dass Hierarchien von Features gelernt werden. Dabei dient also jede Schicht als Feature-Transformation in einen anderen Raum, auf dem dann die nächsten Schichten besser klassifizieren können. Dabei werden zwischen den Schichten nichtlineare Funktionen, die sogenannten *Aktivierungsfunktionen*, eingebaut, um eine bessere Vorhersagekraft zu erreichen. Ein einzelnes Neuron mit den Eingabewerten $x_1, \cdots, x_m$ aus der vorherigen Schicht, den Gewichten $w_1, \cdots, w_m$, einem Bias b und der Aktivierungsfunktion f wird durch

$$a = f\left(\sum_{i=1}^{m} w_i x_i + b\right) = f\left(\boldsymbol{w}^T \boldsymbol{x} + b\right)$$

beschrieben, wobei a die *Aktivierung* des Neurons genannt wird. Eine Schicht besteht üblicherweise aus mehreren Neuronen, was für eine Schicht mit n Neuronen durch

$$\boldsymbol{a} = \boldsymbol{f}\left(\boldsymbol{W}^T \boldsymbol{x} + \boldsymbol{b}\right)$$

kompakt dargestellt werden kann. Dabei ist der Bias $\boldsymbol{b} \in \mathbb{R}^n$ nun ein Vektor und die Gewichte $\boldsymbol{W} \in \mathbb{R}^{m \times n}$ eine Matrix. Die Aktivierungsfunktion $\boldsymbol{f} : \mathbb{R}^n \to \mathbb{R}^n$ ist in den meisten Fällen Komponentenweise zu verstehen (z. B. bei Sigmoid, ReLU, Tanh), es gibt aber auch Aktivierungsfunktionen, die die Informationen aller Neuronen kombinieren (z. B. Softmax).

10.1 Faltungsschichten

Werden durch ein neuronales Netzwerk Bilder verarbeitet, so ist die Nutzung von voll verbundenen Schichten (Englisch: *Fully-Connected Layers*), bei denen jeder Eingabedatenpunkt mit jedem Neuron verbunden ist, oftmals zu aufwendig und nicht sinnvoll. Eine alternative Architektur sind *Faltungsschichten* (Englisch: *Convolutional Layers*), die wesentlich weniger Parameter haben und somit einfacher zu trainieren sind. Eine Faltungsschicht besteht dabei aus mehreren Filtern, die durch Kernel aus der Bildverarbeitung dargestellt werden. Dabei werden die Bilder nicht wie bei voll verbundenen Schichten flach, also eindimensional, dargestellt, sondern es werden die räumlichen Dimensionen (Höhe und Breite) sowie die *Kanäle*, bspw. Farben, beachtet.

Ein Filter wird durch so viele $F_x \times F_y$-Matrizen dargestellt wie die Eingabe Kanäle hat. Dieser Filter wird anschließend Pixel für Pixel über das Bild geschoben. Es ist auch möglich den Filter nicht nur um einen, sondern mehrere, Pixel zu verschieben. Diese Eigenschaft wird durch den *Stride* S_x/S_y charakterisiert und kann für die x- bzw. y-Richtung (d. h. die Breite- bzw. die -Höhe) unterschiedlich sein. Da durch dieses

[1] `https://fabian.damken.net/summaries/cs/elective/iws/dlam/`

verschieben die Daten kleiner werden und dies nicht immer gewünscht ist, kann ein Padding mit Nullen am Rand durchgeführt werden, d. h. das Bild wird um P_x/P_y Nullen links, rechts, oben und unten erweitert. Die Breite, bzw. Höhe, des Ausgabevolumens ist dann

$$O_{x/y} = \frac{I_{x/y} - F_{x/y} + 2P_{x/y}}{S_{x/y}} + 1,$$

wobei I die Eingabebreite/-höhe ist. Ein Filter ist also nur anwendbar, wenn diese Formel einen ganzzahligen Wert ausgibt. In einer Schicht werden dabei üblicherweise mehrere Filter eingesetzt, deren Anzahl dann die Anzahl Kanäle des Ausgabevolumens festlegt. Ein Filter besteht dabei aus mehreren einzelnen Filtern, einem für jeden Kanal, und die Ergebnisse werden addiert. Eine Filterschicht mit D hat demnach bei d Eingabekanälen

$$\#\text{Parameter} = dDF_xF_y + O_x\,O_y$$

Parameter, wobei $O_x\,O_y$ den Bias darstellt. Da Eingabedaten, Bilder, Filter, Stride und Zero-Padding üblicherweise quadratisch sind, gilt in dem Fall auch vereinfacht:

$$O = \frac{I - F + 2P}{S} + 1 \qquad\qquad \#\text{Parameter} = dDF^2 + O^2$$

Eine gute Visualisierung des Vorgehens ist hier zu finden:

`https://cs231n.github.io/assets/conv-demo/index.html`

In einem Faltungsnetzwerk (Englisch: *Convolutional Neural Network*, CNN) werden mehrere dieser Faltungsschichten (mit nachgelagerten Aktivierungsfunktionen) aufeinander geschichtet und bilden so das neuronale Netzwerk. Häufig werden diese anschließend mit einem Klassifizierungs- oder Regressions-„Kopf“ versehen, d. h. einem voll verbundenem Netzwerk, welches das letzte Aktivierungsvolumen des CNNs annimmt, auf eine Dimension „quetscht“, und auf den so erstellten Features Klassifikation/Regression durchführt.

10.2 Räumliche Zusammenfassung: (Max) Pooling

Als weitere Maßnahme zur Reduktion der Datengröße wird *Pooling* angewendet. Beim Pooling werden wird das Aktivierungsvolumen verkleinert indem beieinander liegende Pixel zusammengefasst werden. Dabei arbeiten sie auf jedem Kanal unabhängig von den anderen Kanälen und haben häufig keine Parameter. Eine Möglichkeit des Pooling ist *Max Pooling*, bei dem ein Filter über das Bild geschoben wird, der nur den maximalen Wert behält. Oft wird ein 2x2-Filter mit Stride 2 in jede Richtung verwendet, der die räumlichen Dimensionen (Höhe und Breite) des Bildes halbiert. Offensichtlich ist dies nur auf Daten mit gerader räumlichen Dimension anwendbar.

Andere Möglichkeiten des Poolings sind z. B. Sum Pooling, Mean Pooling oder überlappendes Pooling.

10.3 Aktivierungsfunktionen

Die Wahl der Aktivierungsfunktion ist ein entscheidender Faktor für den Erfolg des Netzwerkes. In diesem Abschnitt werden die bekanntesten Aktivierungsfunktionen zusammengefasst, eine sehr ausführliche Liste ist unter

`https://en.wikipedia.org/wiki/Activation_function`

zu finden.

Identität

$$f : \mathbb{R} \to (-\infty, \infty) : x \mapsto x \qquad\qquad f'(x) = 1$$

Diese Aktivierungsfunktion wird im Allgemeinen nicht verwendet, da sie keinen wirklichen Mehrwert in Form von Nichtlinearität liefert.

Sigmoid

$$f : \mathbb{R} \to (0, 1) : x \mapsto \frac{1}{1 + e^{-x}} \qquad\qquad f'(x) = f(x)\big(1 - f(x)\big)$$

Ein großer Nachteil der Sigmoid-Funktion ist, dass der Gradient nur im $x = 0$ von Null verschieden ist, sie selbst aber nur Werte $f(x) > 0$ produziert und an die nächsten Schichten weiter gibt. Dadurch kommt es durch Multiplikation mit Null zu „sterbende Gradienten“ und es kann kein Lernen stattfinden.

Tanh

$$f : \mathbb{R} \to (-1, 1) : x \mapsto \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \qquad\qquad f'(x) = 1 - \big(f(x)\big)^2$$

Der hyperbolische Tangens ist der Sigmoid-Funktion sehr ähnlich, produziert aber auch negative Werte. Da die Ableitung aber immer Werte im Intervall $(0, 1)$ annimmt, werden die Gradienten immer kleiner, wenn sie durch eine solche Schicht fließen. Daher kann auch diese Aktivierungsfunktion zu sterbenden Gradienten führen.

Rectified Linear Unit (ReLU)

$$f : \mathbb{R} \to [0, \infty) : x \mapsto \max\{0,\, x\} \qquad\qquad f'(x) = \begin{cases} 0 & \text{falls } x < 0 \\ 1 & \text{falls } x > 0 \\ \text{undefiniert} & \text{falls } x = 0 \end{cases}$$

Die ReLU-Aktivierungsfunktion ist die bekannteste und meistgenutzte Aktivierungsfunktion im tiefen Lernen. Sie hat den Nachteil dass der Gradient für die Hälfte der Eingabewerte $x \leq 0$ auf Null gesetzt wird, wodurch kein Lernen stattfinden kann. Außerdem ist sie an der Stelle $x = 0$ nicht Differenzierbar, was in der Praxis aber kein großes Problem darstellt.

Leaky ReLU Um die Null-Ableitungen der ReLU-Funktion zu vermeiden, kann setzt die Leaky ReLU-Funktion die Werte links von der Null nicht komplett auf Null:

$$f : \mathbb{R} \to [0, \infty) : x \mapsto \max\{0.01x,\, x\} \qquad\qquad f'(x) = \begin{cases} 0.01 & \text{falls } x < 0 \\ 1 & \text{falls } x > 0 \\ \text{undefiniert} & \text{falls } x = 0 \end{cases}$$

Dadurch wird eine Sättigung des Gradienten vermieden.

Softmax und Maxout Die Softmax-Aktivierungsfunktion arbeitet im Gegensatz zu den herkömmlichen Funktionen nicht nur auf der Eingabe eines Neurons, sondern auf allen aus der Schicht. Sie weißt dabei jedem Eingabewert eines Ausgabewert im Intervall $(0, 1)$ zu, die insgesamt auf 1 aufsummieren. Die Ausgabe kann demnach zur Klassifikation als Konfidenz verwendet werden, dass die Eingaben dieser Klasse zuzuordnen sind:

$$f : \mathbb{R}^n \to (0,1)^n : \boldsymbol{x} \mapsto \left[\frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}\right]_{i=1,\cdots,n} \qquad \frac{\partial f_i(\boldsymbol{x})}{\partial x_j} = f_i(\boldsymbol{x})\big(\delta_{ij} - f_j(\boldsymbol{x})\big)$$

Dabei ist δ_{ij} das Kronecker-Delta. Ein großer Vorteil dieser Terminalaktivierungsfunktion ist, dass sie überall differenzierbar ist.

10.4 Training

In diesem Abschnitt wird das Training eins neuronalen Netzes, d. h. die Optimierung der Parameter behandelt. Dabei wird zunächst auf die verwendeten numerischen Verfahren und anschließend auf die Berechnung der benötigten Gradienten eingegangen. Numerische Tricks zur Verbesserung des Lernprozesses werden dabei nicht ausführlich behandelt. Diese werden in der Zusammenfassung von „Deep Learning: Architectures and Methods“[2] behandelt.

10.4.1 Initialisierung

Die Initialisierung eines neuronalen Netzwerk ist einer der wichtigsten Faktoren dafür, ob das Netz nach dem Training vernünftig arbeitet [3]. Dabei ist es sehr wichtig, nicht einfach alle Gewichte mit Null oder einem anderen konstanten Wert (wobei Null der schlechteste ist, da die Gradienten sterben) zu initialisieren, da sich diese sonst in die gleiche Richtung entwickeln. Eine Möglichkeit der Initialisierung ist Stichproben aus einer Gleichverteilung $U(-b, b)$ oder einer Normalverteilung zu ziehen. Viele Initialisierungsverfahren sind z. B. in der Dokumentation von PyTorch zu finden:

`https://pytorch.org/docs/stable/nn.init.html`

10.4.2 Stochastischer Gradientenabstieg

Beim *stochastischen Gradientenabstieg* (Englisch: *Stochastic Gradient Descent*) werden nun die folgenden Schritte so lange wiederholt, bis der Verlust konvergiert ist:

1. Wählen einer Stichprobe von Datenpunkten aus der Gesamtmenge.
2. Vorwärtspropagation der Daten durch das neuronale Netzwerk und Berechnung der Verlusts E durch Vergleich mit den vorliegenden Trainingsdaten.
3. Rückwärtspropagation zur Berechnung des Gradienten.
4. Aktualisierung der Parameter mittels des Gradienten (mit der Lernrate $\alpha \in (0, 1)$):

$$\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} - \alpha \nabla_{\boldsymbol{\theta}} E\big(\boldsymbol{\theta}^{(k)}\big)$$

[2] `https://fabian.damken.net/summaries/cs/elective/iws/dlam/`
[3] Jonathan Frankle und Michael Carbin (2018): „The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks“

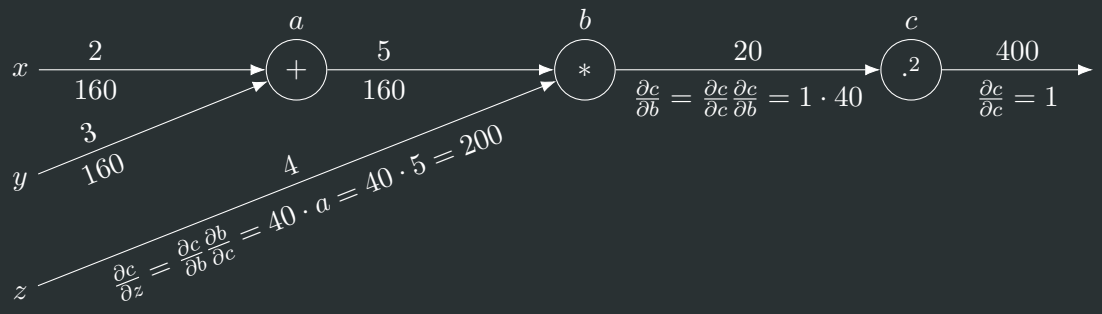

Abbildung 10.1: Berechnungsgraph für $\big((x+y)z\big)^2$, wobei die Werte der Vorwärtspropagation über und die der Rückwärtspropagation unter den Kanten stehen. Für einige Ableitungen ist zusätzlich der Rechenweg notiert, der für die anderen Kanten analog gilt.

Dies ist das Grundverfahren beim stochastischen Gradientenabstieg und dem Trainieren von neuronalen Netzwerken. Dazu kommen viele, viele weitere Tricks und Hilfsmittel, um das Lernen zu beschleunigen und zu verbessern.

10.4.3 Rückwärtspropagation

Während die Vorwärtspropagation zur Bestimmung der Ausgaben des Netzwerkes einfach ausgerechnet werden kann, bedarf es bei der Rückwärtspropagation, also der Berechnung der Gradienten in Richtung der Gewichte/Biases, mehr Denkarbeit. Die Grundlage besteht dabei in der mehrfachen Anwendung der Kettenregel

$$\frac{\mathrm{d}}{\mathrm{d}x} f\big(g(x)\big) = \frac{\partial f}{\partial g}\frac{\mathrm{d}g}{\mathrm{d}x}$$

durch die ein Baustein-System zur Berechnung von Vorwärts- und Rückwärtspropagation möglich ist. Dieses Baustein-System entspricht einem Berechnungsgraphen, der aus primitiven Operationen wie Summe und Multiplikation aufgebaut ist. Entlang diesem können schließlich die Gradienten sowie sämtliche Zwischenergebnisse berechnet werden. Ein Beispielhafter Berechnungsgraph inklusive Vor- und Rückwärtspropagation ist in Abbildung 10.1 gegeben.

Im folgenden werden einige der Bausteine vorgestellt, die zum aufbauen eines neuronalen Netzwerkes genutzt werden können.

Baustein: Sequenz

Dieser Baustein setzt andere Bausteine $B_1, \cdots, B_n$ zusammen, indem diese sequentiell angewandt werden. Vorwärts- und Rückwärtspropagation sind dabei durch eine Verkettung der Funktionen, bzw. der Kettenregel, gegeben.

Vorwärtspropagation:

$$B(x) = B_1\big(B_2\big(\cdots B_n(x)\big)\big)$$

Rückwärtspropagation:

$$\nabla B(x) = B_1'(x)B_2'\big(B_1(x)\big)\cdots B_n'\big(B_{n-1}(\cdots)\big)$$

Baustein: Verlust

Dieser Baustein implementiert die Verlustfunktion, beispielsweise den quadratischen Verlust. Dabei ist neben der Ausgabe $\hat{y}$ des Netzwerkes auch der Sollwert y gegeben. Viele andere Verlustfunktionen zur Klassifikation sind unter

`https://en.wikipedia.org/wiki/Loss_functions_for_classification`

zu finden, zur Regression wird meistens der quadratische Verlust oder eine Likelihood verwendet.

Vorwärtspropagation:

$$E(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$$

Rückwärtspropagation:

$$\frac{\partial E}{\partial \hat{y}} = \hat{y} - y$$

Baustein: Linear

Dieser Baustein implementiert eine Schicht eines neuronalen Netzwerkes (ohne Aktivierungsfunktion) und ist einer der meist genutzten Blöcke.

Vorwärtspropagation:

$$\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$$

Rückwärtspropagation:

$$\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}} = \boldsymbol{W} \qquad \frac{\partial y_i}{\partial W_{jk}} = \delta_{ij} x_k \qquad \frac{\partial \boldsymbol{y}}{\partial \boldsymbol{b}} = \boldsymbol{I}$$

Da die Ableitung des Blocks in Richtung der Gewichtsmatrix einen dreidimensionalen Tensor ergibt, ist die Ableitung Komponentenweise angegeben.

Baustein: Aktivierungsfunktion

Der Baustein der Aktivierungsfunktion wird üblicherweise nach einer linearen Schicht, bzw. einer Faltungsschicht, eingesetzt, um Nichtlinearitäten in das Netzwerk einzufügen. Dabei gibt es viele verschiedene Aktivierungsfunktionen, die bereits in Abschnitt 10.3 vorgestellt wurden.

10.5 Vermeidung von Überanpassung

Da neuronale Netzwerke häufig viele Parameter haben kann es schnell passieren, dass diese auf die Daten überangepasst werden. Zur Vermeidung dessen gibt es viele unterschiedliche Möglichkeiten. Die einfachste ist die *Datenaugmentierung*, bei der die ursprünglichen Trainingsdaten erweitert werden, bspw. durch Spiegelung oder Rotation der Bilder, das wählen Zufälliger Ausschnitte, Änderung der RGB-Wert oder andere Bildtransformationen.

Eine andere Möglichkeit ist *Dropout*, wobei während des Lernens die Eingaben von Neuronen zufällig (üblicherweise mit der Wahrscheinlichkeit $1/2$) Null gesetzt werden, sodass diese nicht zum Ergebnis beitragen. Dadurch sind die anderen Neuronen gezwungen, Robustere Merkmale zu lernen um ohne diese Neuronen auszukommen. Der Nachteil ist, dass das Training des Netzes in etwas doppelt so lange dauert. Eine Alternative sind Standardregularisierungen wie z. B. L2-Regularisierung.

10.6 Fine-Tuning und Transfer Learning

Eine Möglichkeit das Training zu beschleunigen ist, gar nicht erst zu trainieren. Es können vortrainierte Modelle (z. B. auf ImageNet) genutzt werden, denen anschließend nur noch ein Klassifikationskopf angesetzt wird, welcher dann auf den vorliegenden Daten Fine-Tuned wird. Dieses vorgehen wird *Transfer Learning* genannt.

10.7 Täuschen von CNNs

Ein Problem von CNNs und neuronalen Netzwerken allgemein ist, dass der Grund, wieso die Klassifikation so gut funktioniert, nicht bekannt ist: Die Netzwerke und deren Entscheidungen sind nicht erklär- und interpretierbar. Es gab in der Vergangenheit immer wieder Fälle, in denen ein Netzwerk getäuscht werden konnte, indem bspw. zufälliges Rauschen auf ein Bild addiert wurde.

11 Data Mining: Apriori und PageRank

Im *Data Mining* wird versucht aus großen Datenmengen Wissen zu gewinnen, beispielsweise *Assoziationsregeln* bei Einkäufen. Dabei ist eine Menge von Einkäufen, also eine Menge von Mengen deren Inhalt Produkte sind, gegeben, und es sollen Regeln der Form „Wenn A gekauft wird, dann auch B“ gefunden werden.

Sei R eine Menge von Objekten, die binäre Werte haben („wurde gekauft“ und „wurde nicht gekauft“). Sei $t \subseteq R$ eine Transaktion und r eine Menge von Transaktionen. Dies ist die Menge, die oben informell Beschrieben wurde. Dann sollen alle Regeln der Form $X \to Y$ mit $X \subseteq R$, $Y \subseteq R$ und $X \cap Y = \emptyset$ gefunden werden, für die gilt:

$$s(X \to Y) \equiv s(X \cup Y) \coloneqq \frac{\left|\{t \in r \mid X \cup Y \subseteq t\}\right|}{|r|} \geq s_{\min}$$
$$c(X \to Y) \coloneqq \frac{\left|\{t \in r \mid X \cup Y \subseteq t\}\right|}{\left|\{t \in r \mid X \subseteq t\}\right|} \geq c_{\min}$$

Dabei ist $s_{\min} \in [0, 1]$ die minimale Unterstützung und $c_{min} \in [0, 1]$ die minimale Konfidenz. Unterstützung und Konfidenz können dabei verstanden werden als der Anteil der Transaktionen, die der Assoziationsregel $X \to Y$ genügen, an den Gesamttransaktionen (Unterstützung) und an den Transaktionen, die mindestens die „Prämisse“ X enthalten (Konfidenz). Durch den Anspruch einer minimalen Unterstützung können irrelevante Assoziationsregeln ausgeschlossen werden, da diese nur selten anwendbar sind. Durch die minimale Konfidenz können unsichere Assoziationsregeln ausgeschlossen werden, da die Konklusion nur selten zutrifft.

11.1 Apriori

Der *Apriori-Algorithmus* ist ein Verfahren zur Extraktion von Assoziationsregeln aus einer Menge von Daten. Dabei werden *häufige Mengen*[1] in einer Menge $L_k \subseteq R^k$ gesammelt, die die häufigen Mengen mit k Elementen enthält. Dabei gilt:

- Ist eine Menge häufig, so sind auch alle ihre Teilmengen häufig (Anti-Monotonie).
- Ist eine Menge nicht häufig, so sind auch alle Obermengen nicht häufig (Monotonie).

Der Kern des Algorithmus ist nun die Kandidatengenerierung, um aus einer Menge L_k die Menge L_{k+1} zu bilden. Dazu werden alle Mengen aus L_k, die $k - 1$ Objekte gemeinsam haben, erschöpfend vereinigt, und bilden anschließend die Menge L_{k+1}. Daraus folgt für eine Menge $X \in L_k$, dass alle echten Teilmengen $S_i \subsetneq X$ auch ein L_{k-1} liegen, d. h. $S_i \in L_{k-1}$.

Zusammengesetzt besteht der Apriori-Algorithmus also aus den Schritten der Kandidatenerzeugung, die aus der Kandidatengenerierung und -kürzung, d. h. entfernen der Kandidaten, welche die Anforderung an die Unterstützung nicht erfüllen, besteht, sowie der Regelerzeugung aus den häufigen Mengen.

Bei der Generierung der häufigen Mengen wird zunächst aus der Menge L_k eine Kandidatenmenge C_{k+1} erstellt, die anschließend im Kürzungsschritt zu der Menge L_{k+1} weiter verarbeitet wird. Dieser Schritt ist

[1] Eine Menge ist genau dann häufig, wenn ihre Unterstützung über dem Unterstützungs-Minimum $s_{\min}$ liegt.

in Algorithmus 5 skizziert. Zusammen mit der Regelerzeugung in Algorithmus 6 ergibt dies den Apriori-Algorithmus in Algorithmus 7.

Trotz seiner Einfachheit hat der Apriori-Algorithmus allerdings gravierende Nachteile: Im schlimmsten Fall ist der exponentiell in R, da möglicherweise alle Teilmengen gebildet werden. In der Praxis ist dies aber oft nicht der Fall und die Beschneidung für die Mindestunterstützung und -konfidenz reicht meist aus. Des weiteren liefert Apriori unheimliche viele, stark redundante, Regeln. Dies erschwert die Interpretierbarkeit (Beispiel: Kaufen alle Kunden Produkt A, kaufen auch Kunden, die Produkt B kaufen, Produkt A).

11.1.1 Regelbewertung

Ein relevanter Teil bei der Findung von Assoziationsregeln ist die Bewertung dieser. Dabei baut Apriori auf die Unterstützung und die Konfidenz auf, letzte erfüllt aber eine Ansprüche nicht, die an eine sinnvolle Bewertungsmetrik gestellt werden sollten:

- Unabhängige Mengen (d. h. Prämisse und Konklusion sind unabhängig) sollten mit 0 bewertet werden.
- Die Bewertung einer Regel sollte höher werden, wenn die Regel mehr Belege hat.
- Die Bewertung einer Regel sollte niedriger werden, wenn die Regel weniger Belege hat.

11.2 Web Mining

Das Internet, bzw. konkret das World Wide Web, haben im Bereich des Data Mining zu vielen interessanten Forschungsfragen und Analysezielen geführt. Das ist zum Beispiel die Indexierung von Webseiten zur Suche, Analyse von Klick-Strömen, Ko-Zitations-Netzwerke, finden häufiger Muster in Informationsquellen und das Ranking von Webseiten. Dabei kann das Web als Graph interpretiert werden, dessen Knoten Webseiten und Kanten Verlinkungen sind.

11.2.1 PageRank

Eine besonders wichtige spielt das Ranking und die Bewertung von Webseiten, bspw. um die Reihenfolge der Suchergebnisse bei einer Suchmaschine festzulegen. Dabei geht es insbesondere um die Feststellung, wann eine Seite besonders *wichtig* ist. Eine Seite, die besonders viele andere Seiten verlinkt, heißt dabei *expansiv* und eine Seite, die von besonders vielen anderen Seiten verlinkt wird heißt *beliebt*. Es kann nun die Frage gestellt werden, mit welcher Wahrscheinlichkeit ein Nutzer auf die Seite i kommen würde, wenn er auf einer zufälligen Seite startet und den folgenden Regeln folgt:

- Mit der Wahrscheinlichkeit α folgt der Nutzer einer Kante der aktuellen Seite (Übergangswahrscheinlichkeit).
- Mit der Wahrscheinlichkeit $1 - \alpha$ springt er auf eine zufällige Seite unter der Annahme, dass die Seiten gleich verteilt sind (Sprungwahrscheinlichkeit).

Der Rang einer Seite nach dem *PageRank* (benannt nach dem Erfinder Larry Page) ist dabei der Anteil von i an den besuchten Knoten.

Zur Berechnung des PageRank wird eine Matrix $\boldsymbol{M}$ aufgebaut, bei der der Eintrag M_{ij} die Wahrscheinlichkeit angibt, von Knoten j zum Knoten i über eine Kante überzugehen (Übergangswahrscheinlichkeit). Sei $n(j)$ die Anzahl von j ausgehender Kanten, dann gilt $M_{ij} = 1/n(j)$. Eine zweite $N \times N$-Matrix $\left[1/N\right]$ mit den Einträgen $1/N$ gibt die Sprungwahrscheinlichkeiten für N Webseiten an. Die Wahrscheinlichkeit, eine Webseite

Algorithmus 5 : Apriori: Berechnung der häufigen Mengen

1 **Function** *Erzeuge-Kandidaten*(L_k)**:**
  Input : Häufige Mengen L_k
  Output : Kandidatenmenge C_{k+1}
  // Initialisiere die Kandidatenmenge:
2 $C_{k+1} \leftarrow \{\}$
3 **foreach** $I_1, I_2 \in L_k$*, die* $k-1$ *Objekte gemeinsam haben* **do**
4 Seien $\{i_1, \cdots, i_{k-1}\}$ die gemeinsamen Objekte von I_1 und I_2.
5 Seien i_k und i'_k die unterschiedlichen Objekte von I_1, bzw. I_2.
  // Konstruiere den potentiellen neuen Kandidaten:
6 $I \leftarrow \{i_1, \cdots, i_{k-1}, i_k, i'_k\}$
7 **if** *alle* k*-elementigen Teilmengen von* I *in* L_k *sind* **then**
  // Erweitere die Kandidatenmenge:
8 $C_{k+1} \leftarrow C_{k+1} \cup I$
9 **return** C_{k+1}

10 **Function** *Kürze*$(C_{k+1}, r, s_{\min})$**:**
  Input : Kandidatenmenge C_{k+1}, Transaktionsmenge r, Mindestunterstützung $s_{\min}$
  Output : Häufige Mengen L_{k+1}
  // Initialisiere die häufigen Mengen:
11 $L_{k+1} \leftarrow \{\}$
12 **foreach** $I \in C_{k+1}$ **do**
  // Berechne die Unterstützung:
13 $s \leftarrow \dfrac{\left|\{t \in r \mid I \subseteq t\}\right|}{|r|}$
14 **if** $s \geq s_{\min}$ **then**
  // Nimm den Kandidaten in die häufigen Mengen auf:
15 $L_{k+1} \leftarrow L_{k+1} \cup \{I\}$
16 **return** L_{k+1}

17 **Function** *Häufige-Mengen*$(R, r, s_{\min})$**:**
  Input : Objektgesamtheit R, Transaktionsmenge r, Mindestunterstützung $s_{\min}$
  Output : Häufige Mengen
  // Initialisiere die häufigen Mengen mit allen einelementigen Kandidaten:
18 $C_1 \leftarrow \bigcup_{i \in R} i$
19 $L_1 \leftarrow$ *Kürze*$(C_1, r, s_{\min})$
20 $k \leftarrow 1$
21 **while** $L_k \neq \emptyset$ **do**
22 $C_{k+1} \leftarrow$ *Erzeuge-Kandidaten*(L_k)
23 $L_{k+1} \leftarrow$ *Kürze*$(C_{k+1}, r, s_{\min})$
24 $k \leftarrow k + 1$
25 **return** $\bigcup_{i=1}^{k} L_k$

Algorithmus 6 : Apriori: Regelerzeugung

Function *Erzeuge-Regeln*$(L, c_{\min})$**:**
  Input : Häufige Mengen L, Mindestkonfidenz $c_{\min}$
  Output : Assoziationsregeln $\mathcal{H}$
  // Initialisiere die Assoziationsregeln:
  $\mathcal{H} = \{\}$
  foreach $I \in L$ **do**
    // Erzeuge alle einelementigen Konklusion:
    $H_1 \leftarrow \{\}$
    foreach $i \in I$ **do**
      if $c(I \setminus \{i\} \rightarrow \{i\}) \geq c_{\min}$ **then**
        $H_1 \leftarrow H_1 \cup \big\{\{i\}\big\}$
    // Nutze die Kandidatengenerierung zur Generierung der nächsten Konklusionen:
    $k \leftarrow 1$
    while $H_k \neq \emptyset$ **do**
      $H_{k+1} \leftarrow$ *Erzeuge-Kandidaten*(H_k)
      foreach $h \in H_{k+1}$ **do**
        if $c(I \setminus h \rightarrow h) \geq c_{\min}$ **then**
          // Entferne h aus H_{k+1}:
          $H_{k+1} \leftarrow H_{k+1} \setminus \{h\}$
      $k \leftarrow k + 1$
    // Erzeuge die Assoziationsregeln mit Prämisse:
    $H = \bigcup_{i=1}^{k} H_k$
    foreach $h \in H$ **do**
      $\mathcal{H} \leftarrow \mathcal{H} \cup \big\{(I \setminus h) \rightarrow h\big\}$
  return $\mathcal{H}$

Algorithmus 7 : Apriori

Input : Objektgesamtheit R, Transaktionsmenge r, Mindestunterstützung $s_{\min}$, Mindestkonfidenz $c_{\min}$
Output : Assoziationsregeln $\mathcal{H}$

$L \leftarrow$ *Häufige-Mengen*$(R, r, s_{\min})$
$\mathcal{H} \leftarrow$ *Erzeuge-Regeln*$(L, c_{\min})$
return $\mathcal{H}$

zu besuchen, ist dann die Summe von Sprung- und Übergangswahrscheinlichkeit, zusammengefasst in der Matrix $\boldsymbol{M}'$:

$$\boldsymbol{M}' = (1-\alpha)\left[1/N\right] + \alpha\boldsymbol{M}$$

Der PageRank ist dann der rekursive Algorithmus

$$\text{Rang}(i) = \frac{1-\alpha}{N} + \alpha \sum_{j \in \{(i,j)\}} \frac{\text{Rang}(i)}{n(j)},$$

wobei mit $j \in \{(i,j)\}$ die Nachbarn von i gemeint sind. Die Lösungen dieses Algorithmus sind durch den Eigenvektor von $\boldsymbol{M}'$ mit dem Eigenwert $\lambda = 1$ gegeben. Dieser Eigenwert ist nach dem Satz von Perron-Frobenius der eindeutig bestimmte Eigenwert, der (betragsmäßig) größer als alle anderen Eigenwerte ist.

Ein großer Nachteil des PageRank ist, dass nicht das Verhalten des Nutzers, sondern die Konfiguration der Webseite, also der Wille des Webseitenbetreibers, ausschlaggebend für das Ranking ist. So können sich große Firmen oder Webseitenbetreiber zum Beispiel Verlinkungen einkaufen, um den PageRank zu erhöhen.