Medizininformatik

Ein Kompendium für Studium und Praxis

Bearbeitet von
Martin Dugas

1. Auflage 2017. Buch. X, 259 S. Hardcover
ISBN 978 3 662 53327 7
Format (B x L): 16,8 x 24 cm
Gewicht: 664 g

Grundbegriffe aus der Informatik

2

Zusammenfassung

Im folgenden Kapitel werden einige Grundbegriffe aus der Informatik kurz erläutert, die für das Verständnis von Medizininformatik wichtig sind. Es handelt sich dabei um eine kleine Auswahl, die Nicht-Informatikern das Verständnis der Folgekapitel erleichtern soll. Für detaillierte Darstellungen wird auf Sekundärliteratur verwiesen.

2.1 Datenstrukturen

Es gibt verschiedene Verfahren, Daten im Computer abzulegen. Diese haben darauf Einfluss, wie schnell ein Programm arbeiten kann und wie viel Speicherplatz benötigt wird. Dazu ein praktisches Beispiel: Wenn die Bücher in einer Bibliothek ohne jegliche Ordnung in den Regalen aufgestellt wären, müsste man immer den Großteil der Regale durchsehen, bis man ein bestimmtes Buch gefunden hat. Dadurch, dass man einen alphabetisch geordneten Autoren- und Titelkatalog zur Verfügung hat, wird das Auffinden der Bücher deutlich erleichtert. Im Folgenden werden einige wesentliche Datenstrukturen kurz vorgestellt.

Der Speicherplatzbedarf von realen Programmen wird meist in **Byte** angegeben bzw. den dazugehörigen Zehnerpotenzen: 1 Kilobyte (kB) = 1000 Byte, 1 Megabyte (MB) = 1 Million Byte, 1 Gigabyte (GB) = 1 Milliarde Byte, 1 Terabyte (TB) = 1 Billion Byte. Ein Byte entspricht einem Zeichen, das einen Informationsgehalt von 8 Bit hat. Ein **Bit** ist die kleinste Informationseinheit, die nur die Werte 0 und 1 annehmen kann.

2.1.1 Liste

Eine **Liste** ist eine endliche Sequenz von Elementen. Im Gegensatz zur Menge ist die Reihenfolge der Elemente wichtig. Beispiel: Die DNA-Sequenz ATTAGCAA ist eine Liste.

M. Dugas, *Medizininformatik*,
DOI 10.1007/978-3-662-53328-4_2

Die Elemente dieser Liste sind die Nukleotidbasen. Es gibt verschiedene Möglichkeiten, Listen im Computer zu repräsentieren, z. B. als einfach verkettete Liste, bei der von jedem Listenelement auf das Folgeelement verwiesen wird.

Ein spezieller Listentyp ist die **Queue** (**Warteschlange**), bei der neue Elemente am Beginn der Liste eingetragen werden und Elemente am Ende der Liste ausgelesen werden (**FIFO** = First In, First Out). Beim **Stack** werden neue Elemente an das Listenende angehängt und das Auslesen erfolgt ebenfalls vom Listenende her (**LIFO** = Last In, First Out).

2.1.2 Graph

Ein **Graph** G = (V,E) besteht aus einer Menge von **Knoten** V (Vertices) und **Kanten** E (Edges). Graphen eignen sich sehr gut, um Netzwerke in Medizin und Biologie zu repräsentieren, z. B. die verschiedenen Stoffwechselpfade in einer Zelle. Eine Kante besteht aus zwei Knoten, zwischen denen eine Verbindung besteht. Wenn die Kante eine Richtung aufweist, dann liegt ein gerichteter Graph vor (siehe Abb. 2.1), ansonsten ein ungerichteter. Ein Pfad ist eine Liste von Kanten, die zwei Knoten verbindet.

Die Distanzen zwischen den Knoten im Graphen können in einer **Distanzmatrix** dargestellt werden. Den n Knoten werden in der Distanzmatrix n Zeilen und n Spalten zugeordnet. Die Entfernung zwischen den Knoten v_x und v_y ist in Spalte x und Zeile y eingetragen. Knoten in einem Graphen können auch multidimensionale Objekte sein, z. B. Vektoren mit den Ergebnissen von Genexpressionsmessungen. In diesem Fall kann man die Distanz von zwei Knoten durch verschiedene Distanzmaße festlegen. Beispiele hierfür sind:

- Die Differenzen in jeder Vektorkomponente werden ermittelt, quadriert und aufsummiert. Die Quadratwurzel dieser Summe ist die **Euklidische Distanz**.
- Die Absolutwerte der Differenzen in jeder Vektorkomponente werden ermittelt. Die Summe wird als **Manhattan-Distanz** bezeichnet.
- Der **Korrelationskoeffizient** der beiden Vektoren x und y mit den Vektorkomponenten x_i bzw. y_i und den Mittelwerten $\bar{x}$ bzw. $\bar{y}$ wird berechnet als: $r = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum (x_i - \bar{x})^2 \sum (y_i - \bar{y})^2}}$

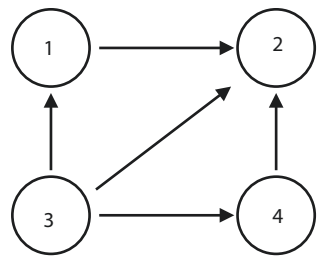


Abb. 2.1 Gerichteter Graph mit vier Knoten und fünf Kanten

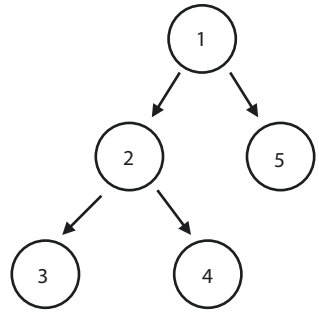


Abb. 2.2 Baum mit drei Blattknoten (3,4,5). Knoten Nr. 1 ist der Wurzelknoten

2.1.3 Baum

Ein besonders wichtiger Typ des gerichteten Graphen ist der **Baum** (Abb. 2.2). Dieser zeichnet sich dadurch aus, dass er keine Zyklen hat (directed acyclic graph, abgekürzt **dag**) und einen so genannten **Wurzelknoten**, zu dem keine Kanten hinführen. Zu jedem Knoten außer dem Wurzelknoten führt genau eine Kante. Vom Wurzelknoten führt zu jedem Knoten genau ein Pfad. Knoten, von denen keine Kanten ausgehen, werden als Blattknoten bezeichnet. Wenn eine Kante vom Knoten 2 zum Knoten 3 vorliegt, dann bezeichnet man Knoten 2 als den Vater und Knoten 3 als das Kind. Als Höhe eines Baumes wird die maximale Länge eines Pfades vom Wurzelknoten zu einem Blatt bezeichnet. Eine wichtige Sonderform des Baumes ist der Binärbaum, bei dem von einem Knoten maximal zwei Kanten ausgehen. Bäume eignen sich besonders gut, um hierarchische Zusammenhänge in Medizin und Biologie darzustellen, wie z. B. einen Stammbaum oder einen Entscheidungsbaum.

Mit der **CART**-Methode (Classification And Regression Trees) [Breiman] können binäre Entscheidungsbäume konstruiert werden. Die Daten werden hierbei rekursiv anhand eines Schwellenwerts in zwei Teilmengen zerlegt. Die Güte des Baumes kann durch Kreuzvalidierung geschätzt werden. Im Fall von Überanpassung an die Lerndaten (Over-Fitting) kann ein zu komplizierter Baum durch Entfernen von Zweigen (Pruning) verbessert werden.

2.1.4 Hash

Viele Anwendungen benötigen nicht komplexe Baumstrukturen, sondern es genügt eine einfache Verzeichnisstruktur mit den Basisoperationen insert(), delete() und search(). Eine effiziente Datenstruktur zum Aufbau solcher Verzeichnisse sind **Hash**-Tabellen. Hash-Tabellen werden verwendet, um Datenelemente in einer großen Datenmenge zu finden. Über einen Schlüsselwert (z. B. Name) ist ein direkter Zugriff auf das Feldelement (z. B. Telefonnummer) möglich.

2.2 Algorithmus

Ein **Algorithmus** ist eine endliche Folge von Instruktionen, die alle eindeutig interpretierbar und mit endlichem Aufwand in endlicher Zeit ausführbar sind. Die Informatik unterscheidet somit Verfahren zur Lösung von Problemen (Algorithmen = Lösungsverfahren) und die Implementation der Verfahren in einer bestimmten Programmiersprache auf bestimmten Rechnern.

Im Hinblick auf die Effizienz von Algorithmen ist das Laufzeitverhalten von besonderem Interesse. Man unterscheidet z. B. Algorithmen, deren **Zeitbedarf** linear mit der Größe n der Eingabedaten ansteigt (geschrieben als **O(n)**), beispielsweise das Durchsuchen einer Liste vom Anfang bis zum Ende. Durch eine geeignete Indexstruktur kann man diesen Algorithmus so verbessern, dass die Laufzeit O(log n) beträgt, d. h., die für das Auffinden eines bestimmten Listenelements benötigte Zeit nimmt proportional zum Logarithmus der Anzahl von Listenelementen zu.

Viele Algorithmen lassen sich **rekursiv** formulieren, d. h., sie enthalten Funktionen, die sich selbst so lange aufrufen, bis ein Abbruchkriterium erfüllt ist. Rekursive Algorithmen haben häufig einen relativ hohen Speicherbedarf für die noch nicht abgeschlossenen Schritte und werden daher nach Möglichkeit in eine **iterative Form** (ohne Rekursion) umgearbeitet.

Für viele Optimierungsprobleme ist bekannt, dass sie **NP-schwer** sind. Dies bedeutet, dass kein Algorithmus mit polynomialer Laufzeit $O(n^k)$ bekannt ist (NP = nicht polynomial), sondern der Zeitbedarf mit der Größe des Problems exponentiell zunimmt. Dies führt dazu, dass derartige Aufgaben für große n nur näherungsweise gelöst werden können, weil die Exponentialfunktion sehr stark ansteigt. Analog zum Laufzeitbedarf kann man auch den **Speicherplatzbedarf** eines Algorithmus beschreiben. Abbildung 2.3 zeigt ein Beispiel für einen Algorithmus.

```
Quadratwurzel (a, eps, x):

  begin
    x := a / 5

    while |a − x²| > eps * a  do
      x := (x + a / x) / 2
    end

end Quadratwurzel
```

Abb. 2.3 Beispiel für Algorithmus: Berechnung der Quadratwurzel x von a. Die while-Schleife wird so lange ausgeführt, bis die durch eps angegebene Genauigkeit erreicht ist

2.2.1 Sortieralgorithmen

Sortierte Daten sind von vielfältiger Bedeutung, z. B. kann man in einer alphabetisch sortierten Liste einen bestimmten Eintrag deutlich schneller auffinden.

Beim **Selection-Sort-Algorithmus** wird zunächst das kleinste Element aus der gesamten Liste gesucht und mit dem ersten Element vertauscht. Analog wird mit dem zweitkleinsten, drittkleinsten usw. verfahren. Beim **Bubble-Sort-Algorithmus** beginnt man mit dem ersten Element der Liste und vergleicht es mit allen weiteren. Wenn ein kleineres Element gefunden wird, dann vertauscht man es mit dem ersten. Man erhält so das kleinste Element in der ersten Position. Man schreitet mit der zweiten Position fort und vergleicht auch dieses mit allen weiteren (ohne die vorderen) und erhält so das zweitkleinste in der zweiten Position. Man fährt mit allen weiteren Elementen fort bis zum vorletzten.

Quicksort arbeitet nach dem Prinzip „**Divide-and-Conquer**“. Dies bedeutet, dass das Problem so lange in kleinere Teilprobleme zerlegt wird, bis diese auf einfache Weise lösbar sind. Man wählt aus der Mitte der Liste ein Vergleichselement. Links und rechts wird auf die Randelemente der Liste je ein Zeiger gesetzt. Der linke Zeiger wird so lange nach rechts verschoben, bis das darüber stehende Listenelement größer oder gleich dem Vergleichselement ist. Der rechte Zeiger wird so lange nach links verschoben, bis das darüber stehende Listenelement kleiner oder gleich dem Vergleichselement ist. Die beiden den Zeigern zugeordneten Elemente werden vertauscht und die Zeiger eine Position weiter bewegt (linker Zeiger nach rechts, rechter Zeiger nach links). Das Verschieben der Zeiger und Vertauschen der zugeordneten Listenelemente wird so lange wiederholt, bis die Zeiger sich überlappen. Zu diesem Zeitpunkt befinden sich links vom ausgewählten Vergleichselement nur noch Elemente kleiner oder gleich diesem Vergleichselement, rechts davon nur noch solche größer oder gleich dem Vergleichselement. Man kann die Liste jetzt zwischen den Zeigern trennen, weil beim späteren Zusammensetzen in der linken Teilliste keine Elemente vorkommen, die größer als Elemente der rechten Teilliste sind. Die Teillisten werden nun genauso verarbeitet wie die ursprüngliche Liste. Sobald eine Teilliste nur noch zwei Elemente enthält, werden diese verglichen und gegebenenfalls vertauscht. Die sortierte Liste erhält man durch Zusammensetzen der Teillisten von links nach rechts.

Man kann zeigen, dass Selection-Sort und Bubble-Sort einen Zeitbedarf von $O(n^2)$ benötigen, d. h., für eine doppelt so lange Liste wird die vierfache Sortierzeit benötigt. Hingegen erfordert Quicksort eine Laufzeit von $O(n \log n)$, was bei langen Listen deutliche Vorteile bringt.

2.2.2 Greedy-Algorithmen

Greedy-Algorithmen (engl. greedy = gierig) werden häufig für die Lösung von Optimierungsproblemen verwendet. Ein Greedy-Algorithmus konstruiert eine beste Lösung Komponente

für Komponente, wobei der Wert einer gesetzten Komponente nie zurückgenommen wird. Die zu setzende Komponente und ihr Wert wird nach einfachen Kriterien bestimmt.

Ein Beispiel für dieses Vorgehen ist der **Algorithmus von Dijkstra** zur Bestimmung kürzester Wege in einem Graphen. Hierbei wird ausgehend von einem Startknoten ein Baum konstruiert, der den kürzesten Weg vom Startknoten zu den Knoten des Baumes beschreibt. Beim Aufbau dieses Baumes wird die Distanz der Baumknoten zu den jeweiligen Nachbarknoten berücksichtigt.

2.2.3 Dynamische Programmierung

In der Bioinformatik wird oft das algorithmische Prinzip des **dynamischen Programmierens** eingesetzt, das mit Greedy-Algorithmen verwandt ist. Hierbei wird die Lösung für ein Problem durch Kombination oder Erweiterung geeigneter Lösungen verwandter Probleme kleinerer Größe gewonnen.

Voraussetzungen

1. Gesucht ist eine Lösung für ein Problem der Größe n. Diese Lösung ist, ebenso wie die Lösungen verwandter Probleme kleinerer Größe, durch eine geeignete Optimalitätsbedingung charakterisiert.
2. Die Lösung für ein Problem einer bestimmten Größe kann durch Erweiterung oder Kombination von geeigneten Lösungen kleinerer Probleme gewonnen werden.
3. Unter all den Möglichkeiten, Lösungen kleinerer Probleme zu erweitern oder zu kombinieren, lassen sich Möglichkeiten erkennen, die nicht auf Lösungen größerer Probleme führen, weil sie die entsprechende Optimalitätsbedingung nicht erfüllen.

Algorithmus dynamische Programmierung

Erzeuge zunächst alle Lösungen für Probleme der Größe 0 (kleinste Problemgröße). Hierauf erzeuge für k = 1, …, n Lösungen für alle Probleme der Größe k. Diese werden durch Erweiterung oder Kombination von Lösungen zu Problemen der Größe < k bestimmt. Lösungen, welche die Optimalitätsbedingung nicht erfüllen, werden verworfen.

2.3 Datenbanksystem

Ein **Datenbanksystem** (DBS) lässt sich wie folgt charakterisieren:

Die zentrale Kontrollinstanz eines DBS ist das Datenbankmanagementsystem (**DBMS**) in Form eines Softwarepaketes, über das Anwendungsprogramme (Applikationen) sowie Benutzer-Dialoge auf eine große Datensammlung, nämlich die Datenbank, zugreifen können. Es ist im Allgemeinen sogar in der Lage, mehrere verschiedene Datenbanken gleichzeitig zu verwalten.

Ein DBMS weist drei Ebenen auf:

- Externe Ebene, z. B. die Sicht eines bestimmten Anwenders
- Konzeptionelle Ebene: logische Gesamtsicht auf die Daten und deren Beziehungen untereinander
- Interne Ebene: physische Datenorganisation, d. h., es wird festgelegt, wie die Daten und deren Beziehungen gespeichert werden und welche Zugriffsmöglichkeiten bestehen

Bei den relationalen Datenbanken werden die Informationen in Form von Relationen (Tabellen) gespeichert. Die Spalten der Tabelle heißen **Attribute**, die Zeilen Tupel. Das **Datenbankschema** legt fest, welche Attribute in einer Tabelle gespeichert werden, und dient somit zur Beschreibung der Struktur der Datenbanktabelle. Jedes Attribut hat einen bestimmten Datentyp, der den Wertebereich festlegt, z. B. **String** (= Text), **Integer** (= Ganzzahl), **Float** (= Gleitkommazahl), **Date** (= Datum), **Time** (= Zeit) oder **Boolean** (= ja/nein, **Checkbox**). Eine minimale Kombination von Attributen, anhand der alle Tupel einer Tabelle eindeutig unterscheidbar sind, heißt **Schlüssel**. Abbildung 2.4 zeigt ein einfaches Beispiel für eine Datenbanktabelle.

Der Industriestandard für relationale Datenbanksprachen ist **SQL** (Structured Query Language). Mittels SQL können Tabellen erzeugt, mit Daten gefüllt sowie Abfragen erstellt werden. Abbildung 2.5 zeigt einfache Beispiele für die Basisoperationen Create, Insert, Delete, Update und Select. Ein **Index** ermöglicht das rasche Wiederauffinden von Information und besteht z. B. aus einer aufsteigend geordneten Liste der Werte eines Attributes mit Verweisen auf den kompletten Eintrag.

Neben einer Fülle von kommerziellen Datenbanksystemen (z. B. Oracle Database, Microsoft SQL Server u.v.a.) gibt es auch frei verfügbare SQL-Datenbanken (z. B. MySQL [MySQL], PostgreSQL [PostgreSQL]).

Die Beziehungen zwischen verschiedenen Datenbanktabellen können im sog. **ER-Modell** (**Entity-Relationship-Modell**) dargestellt werden (Abb. 2.6). Man unterscheidet bei den Beziehungen zwischen den Tabellen 1:1-Relation, 1:n-Relation und n:m-Relation. 1:1-Relation bzw. 1:n-Relation bedeutet, dass zu einem Eintrag aus der 1. Tabelle höchstens ein Eintrag bzw. mehrere Einträge aus der zweiten Tabelle zugeordnet werden können. Eine n:m-Relation liegt vor, wenn die 1:n-Relation auch in umgekehrter Richtung besteht, z. B. zwischen „Studenten“ und „Kursen“: Ein Student kann mehrere Kurse belegen, ein Kurs enthält aber auch mehrere Studenten.

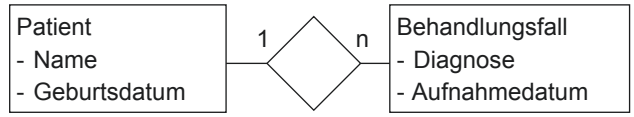
Abb. 2.4 Tabelle „Patient“ mit den Attributen „Name“, „Vorname“ und „Geburtsdatum“

Patient	Name	Vorname	Geburtsdatum
	Mustermann	Peter	4.5.1965
	Mustermann	Anna	6.7.1970
	...		

Abb. 2.5 Beispiele für SQL-Kommandos

```
CREATE TABLE patient (
patnr int4,
name varchar(30), vorname varchar(30),
geschlecht char(1), geburtsdatum date, klinik varchar(30)
);
insert into patient (patnr,name,geschlecht) values (2,'Huber','w');
update patient set geburtsdatum='18.4.1950' where patnr=2;
select patnr,name from patient where geschlecht='w' AND patnr > 1;
delete from patient where patnr=2;
```

Abb. 2.6 ER-Modell: Die Rechtecke entsprechen Datenbanktabellen, die Raute stellt die Beziehung zwischen den Tabellen dar



2.4 Data Warehouse und Data Mining

Datenbanken in der Medizin sind häufig durch ein hohes Maß an Komplexität gekennzeichnet, sodass eine manuelle Analyse nur mit hohem Aufwand möglich ist. Auswertungsrelevante Daten sind oft auf verschiedene Systeme verteilt. Ein **Data Warehouse** (**DWH**) ist eine zentrale Datenbank, die Daten aus mehreren Quellen zusammenführt und für die Datenanalyse optimiert ist. Auf diese Weise wird eine globalisierte Sicht auf heterogene und verteilte Datenbestände möglich.

Der Datentransfer aus den Datenquellen in das DWH erfolgt dabei als so genannter Extract-Transform-Load (**ETL**)-Prozess (Abb. 2.7). Die Quelldaten werden dazu in der Regel temporär zwischengespeichert (Staging), bereinigt (Data Cleaning) und für das DWH transformiert. ETL-Prozesse können sowohl durch das Datenvolumen als auch durch die Komplexität der Datenstrukturen sowie Transformationsprozesse sehr aufwändig sein. ETL-Prozesse müssen angepasst werden, wenn sich die Datenstruktur der Datenquellen ändert, zum Beispiel bei neuen Softwareversionen (Softwarereleases). Es gibt vielfältige IT-Tools für ETL, darunter auch Open Source Software wie zum Beispiel **Talend** Open Studio [Talend].

Für eine bestimmte Anwendung oder einen bestimmten Organisationsbereich kann der jeweilige Teildatenbestand aus dem DWH kopiert werden. Diesen nennt man **Data Mart**. **OLAP** (Online Analytical Processing)-Systeme können zusammen mit DWHs für komplexe Analysevorhaben (z. B. unternehmensweites Berichtswesen/Reporting) eingesetzt werden, die ein hohes Datenaufkommen verursachen. Hierbei können multidimensionale Analysefragestellungen bearbeitet werden, ohne die Leistungsfähigkeit der Primärsysteme zu beeinträchtigen.

In der Medizin ist es manchmal aus Datenschutzgründen (ärztliche Schweigepflicht) notwendig, dass Patientendaten zwar gemeinsam ausgewertet werden sollen, aber nicht direkt zusammengeführt werden dürfen. In dieser Situation kann man **föderierte**

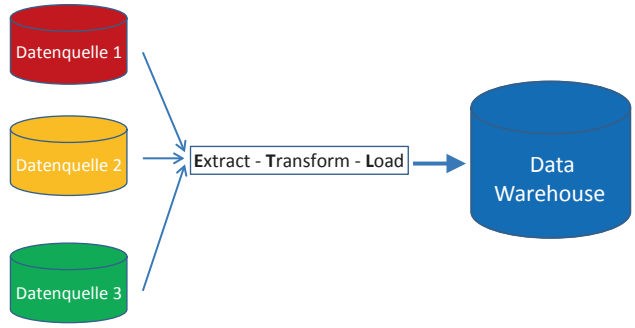


Abb. 2.7 ETL-Prozess: Daten werden aus mehreren Datenquellen extrahiert, transformiert und dann in das DWH geladen

Datenbanksysteme oder föderierte DWHs einsetzen. Es werden in allen Datenquellen die gleichen Abfragen ausgeführt und nur die Ergebnisse dieser Abfragen zusammengeführt. Dies setzt voraus, dass die verschiedenen Datenquellen ausreichend kompatibel sind, um die gleichen Abfragen ausführen zu können. Dies erreicht man durch ein gemeinsames Datenmodell (englisch **Common Data Model**, abgekürzt **CDM**); dies wird auch als **globales Datenbankschema** bezeichnet. **i2b2** [i2b2] ist ein Beispiel für ein DWH mit föderierten Abfragemöglichkeiten in der Medizin. Beispiele für standortübergreifende Datenmodelle in der Medizin sind **OMOP** CDM [OMOP] und **PCORnet** CDM [PCORnet], die in der klinischen Forschung eingesetzt werden.

Data Mining bezeichnet Auswerteverfahren auf große Datenbestände mit dem Ziel, neue Zusammenhänge und Trends zu erkennen. Es geht also um die Extraktion von Wissen, das im statistischen Sinne gültig ist. Typische Aufgaben des Data Mining sind unter anderem:

- **Clustering**: Partitionierung einer Datenbank in Cluster (= Gruppen) von Objekten, sodass Objekte eines Clusters möglichst ähnlich, Objekte verschiedener Cluster möglichst unähnlich sind.
- **Klassifikation**: Ausgehend von Trainingsobjekten, die einer Klasse zugeordnet sind, werden zukünftige Objekte klassifiziert. Im Gegensatz dazu werden durch **Regression** ausgehend von Trainingsdaten Parameter von statistischen Modellen geschätzt.
- **Ausreißererkennung**: Suche nach Datenobjekten, die inkonsistent zu den restlichen Daten sind.
- **Assoziationsregeln**: In einer Datenbank sollen Regeln ermittelt werden, die häufig auftretende und starke Zusammenhänge innerhalb der Transaktionen beschreiben.
- **Zusammenfassung (Aggregation)**: Eine Menge von Daten soll möglichst kompakt beschrieben werden, indem die Attributwerte generalisiert und die Zahl der Datensätze reduziert werden.
- **Evaluation**: Die vom System gefundenen Muster werden geeignet präsentiert und von einem Experten der Anwendung evaluiert. Für die Präsentation wird häufig eine Visualisierung verwendet, weil sie verständlicher ist als eine rein textuelle Ausgabe.

Für die Aufgaben des Data Mining werden u. a. Verfahren des maschinellen Lernens eingesetzt; dabei handelt es sich um Programme, deren Ein-/Ausgabeverhalten sich mit zunehmender Verwendung und „Erfahrung“ automatisch verbessert. Die Daten werden zunächst ohne Hintergrundwissen über deren Bedeutung analysiert und müssen daher sorgfältig interpretiert werden. Wichtige Fehlerquellen für Data Mining sind Probleme der Datenqualität (z. B. unvollständige Daten) und systematische Fehler bei der Datenerfassung (nicht repräsentative Daten). Verfahren des Data Mining werden insbesondere bei großen Datensätzen eingesetzt, bezeichnet als **Big Data**. Diese Datenmengen sind gekennzeichnet durch hohes Datenvolumen, Geschwindigkeit der Datengenerierung, Vielfalt der Datentypen und Datenquellen, variable Konsistenz und Qualität der Daten. Im Englischen werden diese Eigenschaften von Big Data bezeichnet als Volume, Velocity, Variety, Variability, Veracity (**5 V**).

2.5 Internet und Rechnernetze

Das **Internet** ist ein globaler Rechnerverbund. Mit dem Internet-Protokoll (IP) wird jedem an das Netz angebundenen Gerät mindestens eine **IP-Adresse** zugewiesen, über die es erreichbar ist. Das Domain Name System (**DNS**) ist ein Verzeichnisdienst, der **Domainnamen** (zum Beispiel www.uni-muenster.de) die entsprechende IP-Adresse (zum Beispiel 128.176.6.250) zuordnet. World Wide Web (**WWW**) und **E-Mail** sind wichtige Services, die durch das Internet bereitgestellt werden. Für den Versand von E-Mails wird das Simple Mail Transfer Protocol (**SMTP**) eingesetzt.

Mit einem **Internet-Browser** (z. B. Mozilla Firefox, Google Chrome oder Internet Explorer) kann man durch das Hypertext Transfer Protocol (**http**, bzw. als verschlüsselte Version **https**) Internet-Seiten abrufen. Diese Seiten werden meist in der Seitenbeschreibungssprache Hyper Text Markup Language (**HTML**) [HTML] übertragen. Über Querverweise (**Internet-Links**) können HTML-Seiten miteinander verknüpft werden. Internet-Seiten können aktive Komponenten enthalten und dynamisch erzeugt werden; dabei findet häufig ein Datenaustausch mit entfernten Servern statt. Dieser Mechanismus kann äußerst vielfältig genutzt werden, z. B. für **Suchmaschinen** im Internet (beispielsweise Google), aber auch für Anfragen an medizinische Literaturdatenbanken. Abbildung 2.8 zeigt zwei einfache Beispiele für HTML-Seiten.

Reine HTML-Seiten bieten nur begrenzte Interaktionsmöglichkeiten. Mittels **JavaScript** [JavaScript] können diese erweitert werden, zum Beispiel durch **AJAX** (Asynchronous JavaScript and XML). In HTML-Seiten können **Java-Applets** integriert werden. Es handelt sich dabei um Java-Programme [Java], die über das Internet auf den lokalen Rechner geladen und dort ausgeführt werden. Alternativ können Java-Programme auch als so genannte **Servlets** auf einem Internet-Server ausgeführt werden.

Neben dem globalen Internet gibt es eine Vielzahl von **Rechnernetzen**, die nur für Mitglieder der jeweiligen Organisationen (zum Beispiel eines Unternehmens) zugänglich sind. Diese Computernetzwerke werden meist durch eine **Firewall** abgesichert, um

```
<html>
<head>
<title>Testseite</title>
</head>
<body>
<h1>Testseite</h1>
Dieser Text ist <b>fett</b>.
</body>
</html>
```


```
<html>
<body>
<H1>Formular</H1>
<FORM ACTION=formular.cgi> Name
<INPUT NAME="Name"
        TYPE="text" SIZE="3"> <BR>
OP-Status <SELECT NAME="OP-Status">
  <OPTION VALUE="1"> praeoperativ
  <OPTION VALUE="2"> postoperativ
</SELECT>
<BR>
<INPUT TYPE="submit" VALUE="Daten abschicken">
</form>
</body>
</html>
```

Abb. 2.8 Beispiele für HTML-Seiten: eine einfache Textseite sowie ein Formular mit Auswahlliste

einerseits Kommunikation nach außen (zum Beispiel E-Mail) zu ermöglichen und andererseits die Daten und IT-Systeme der jeweiligen Organisation zu schützen. Mit Virtual Private Network (**VPN**) bezeichnet man geschlossene Kommunikationsnetze, die ein bestehendes Netzwerk (insbesondere das Internet) als Transportmedium nutzen. Durch Verschlüsselungsverfahren wird erreicht, dass nur die Mitglieder des VPN die jeweiligen Daten entschlüsseln und verwenden können. Auf diese Weise kann ein Mitarbeiter beispielsweise von zuhause aus über das Internet auf das Firmennetz zugreifen. Man spricht hier auch von einem **VPN-Tunnel**.

2.6 XML und JSON

Das World Wide Web Consortium, die weltweit führende herstellerunabhängige Organisation für die Weiterentwicklung von Internet-Technologie mit mehr als 400 Mitgliedsorganisationen, bezeichnet **XML** (= eXtensible Markup Language) [XML] als das universelle Format für strukturierte Dokumente und Daten im Internet. Abbildung 2.9 zeigt ein Beispiel für ein XML-Dokument aus der Medizininformatik.

```
<?xml version="1.0" encoding="UTF-8" ?>
 <patient>
   <patnr>012345</patnr>
   <name>Mustermann</name>
   <vorname>Peter</vorname>
   <geburtsdatum>1.1.1950</geburtsdatum>
   <diagnose>
     <code>I21</code>
     <diagnosetext>Akuter Myokardinfarkt</diagnosetext>
   </diagnose>
</patient>
```

Abb. 2.9 Patientendaten im XML-Format

XML ist wie HTML eine so genannte Markup Language, kann aber im Unterschied zu HTML praktisch beliebig erweitert werden. Mit einer XML Schema Definition (**XSD**) kann die Struktur eines XML-Dokuments beschrieben werden. Es kann automatisiert geprüft werden, ob ein XML-Dokument dieser Definition entspricht und damit wohlgeformt ist. XML eignet sich besonders für die von verschiedenen Rechnertypen (= Plattformen) unabhängige Beschreibung von komplexen, hierarchischen Strukturen und wird aus diesem Grund auch für die Beschreibung von medizinischen Datenmodellen eingesetzt.

Über **Webservices** können Programme über Internet-Protokolle direkt miteinander kommunizieren (**Maschine-zu-Maschine-Kommunikation**). Es handelt sich also bei Webservices um eine spezielle Form von Programmierschnittstellen (englisch Application Programming Interface, abgekürzt **API**). Es gibt hierfür verschiedene Programmierparadigma, beispielsweise **REST** (Representational State Transfer) oder **SOAP** (Simple Object Access Protocol). Als Dateiformate werden bei Webservices **XML** (extensible Markup Language) und **JSON** (JavaScript Object Notation) [JSON] eingesetzt. Abbildung 2.10 zeigt ein Beispiel für eine Datei im JSON-Format.

```
"patient":
{
  "patnr": "012345",
  "name": "Mustermann",
  "vorname": "Peter",
  "geburtsdatum": "1.1.1950",
  "diagnose": {,
     "code": "I21",
     "diagnosetext": "Akuter Myokardinfarkt"
  }
}
```

Abb. 2.10 Patientendaten der vorigen Abbildung im JSON-Format

2.7 Compilerbau

Ein Compiler im engeren Sinne ist ein Übersetzer für höhere Programmiersprachen. Nach Analyse eines Quelltextes (z. B. ein Java-Programm) wird eine Zwischendarstellung erzeugt, aus der der jeweilige rechnerspezifische **Maschinencode** generiert wird. In diesem Kontext werden Verfahren zur automatisierten Analyse hierarchisch strukturierter Dokumente eingesetzt; diese Methoden sind in der Informatik von allgemeiner Bedeutung, z. B. für Systeme wie Web-Browser.

Die **Syntaxanalyse** eines Dokuments umfasst die **lexikalische Analyse** (sog. **Scanner**), bei der Schlüsselwörter und Operatoren erkannt werden, und die **Strukturanalyse** (sog. **Parser**), bei der der hierarchische Aufbau erfasst wird. Bei der **semantischen Analyse** wird z. B. geprüft, ob die verwendeten Typen korrekt verwendet werden (z. B. ob einer numerischen Variable ein Text zugewiesen wird).

2.8 Formale Sprachen

Formale Sprachen sind teilweise für Menschen etwas schwer verständlich, bieten aber den großen Vorteil, dass sie einer automatisierten Bearbeitung zugänglich sind. Unter einem **Alphabet** versteht man in der Informatik eine endliche Menge von Symbolen, z. B. A = {a,b,c}. Mit A* bezeichnet man die Menge aller Wörter, die mit dem Alphabet A gebildet werden können, wobei auch das leere Wort (geschrieben als **ε**) enthalten ist.

Mit einer **Grammatik** kann man formale Sprachen beschreiben. Diese besteht aus einer Menge von Terminalsymbolen (Tokens), Non-Terminalsymbolen (Variablen), einem Startsymbol und einer Menge von Ableitungsregeln. Alle Wörter aus Terminalsymbolen, die unter Anwendung der Ableitungsregeln aus dem Startsymbol gebildet werden können, umfassen die durch die Grammatik beschriebene Sprache.

Ableitungsregeln können in der sog. **BNF-Notation** (Backus-Naur-Form) dargestellt werden (Abb. 2.11).

Anhand der zulässigen Ableitungsregeln unterscheidet man verschiedene Klassen von Grammatiken. Bei kontextfreien Grammatiken befindet sich auf der linken Seite der Ableitungsregel immer genau eine Variable und kein Token. Bei kontextsensitiven Grammatiken gilt diese Einschränkung nicht.

Ein sog. **endlicher Automat** besteht aus einer Menge von Zuständen, einem Eingangsalphabet, einer Übergangsrelation, einem Startzustand und einer Menge von Endzuständen.

```
<ausdruck> ::= <zahl> | (<ausdruck>) | <ausdruck> [+|-|*|/] <ausdruck>
<zahl> ::= <bziffer> {ziffer}* | 0
<bziffer> ::= 1|2|3|4|5|6|7|8|9
<ziffer> ::= <bziffer> | 0
```

Abb. 2.11 Beispiel: BNF-Notation für arithmetische Ausdrücke

Die von diesem Automaten erkannte Sprache bezeichnet alle Wörter, die den Startzustand durch die Übergangsrelation in einen der Endzustände überführen. Als reguläre Sprachen bezeichnet man genau diejenigen Sprachen, die von endlichen Automaten akzeptiert werden. Eine derartige reguläre Sprache kann man auch durch einen **regulären Ausdruck** beschreiben.

Von den regulären Ausdrücken abgeleitet sind **Wildcards** (z. B. *,?) und **Boolesche Operatoren** (z. B. AND, OR, NOT), die für Suchfunktionen in Datenbanken verwendet werden: Der Ausdruck „M?ier“ steht für alle Begriffe, die mit „M“ beginnen und mit „ier“ enden, das Symbol „?“ steht für einen beliebigen Buchstaben; der Ausdruck „17 AND 23“ liefert als Ergebnis diejenigen Einträge, bei denen die Zahlen 17 und 23 in Kombination auftreten.

2.9 Programmiersprachen

Es gibt eine Fülle von Programmiersprachen, mit denen Algorithmen implementiert werden können. Man unterscheidet hierbei interpretierende Sprachen wie z. B. Perl, bei denen zur Laufzeit der Programmquelltext interpretiert wird, und compilierende Sprachen wie z. B. C++, bei denen Maschinencode vor dem Start des Programms erzeugt wird und daher die Ausführungsgeschwindigkeit höher ist. Bei **Java** [Java] wird aus dem Programmtext ein sog. Bytecode erzeugt, der zur Laufzeit in Maschinencode umgewandelt wird.

In der traditionellen Programmierung wird zwischen Code und Daten unterschieden. Unter Code versteht man Funktionen und Prozeduren, die auf den Daten arbeiten. In der **objektorientierten Programmierung** (OOP) – z. B. bei Java – werden Code und Daten zu Objekten integriert. Dabei ist das „Innenleben“ des Objektes für den Programmierer unbekannt (sog. Black Box). Klassen sind Prototypen von Objekten. Eine Klasse definiert, wie Objekte, die zu dieser Klasse gehören, sich verhalten sollen. Ein **Objekt** ist eine konkrete Instanz einer Klasse. Der Zugriff auf Objekte geschieht über Methoden. Die objektorientierte Programmierung wurde u.a. entwickelt, um der menschlichen Art des Denkens bei der Entwicklung von Programmen näher zu kommen. So wird eine komplexe Aufgabe in viele kleinere Teilaufgaben zerlegt. Komplexe Systeme werden dadurch deutlich überschaubarer.

2.10 Software Engineering

Bereits 1965 wurde der Begriff „Software-Krise“ geprägt: Durch leistungsfähigere Hardware waren kompliziertere Programme möglich, die jedoch unsystematisch erstellt wurden. Dies führte aufgrund der Komplexität der Softwaresysteme zu unüberschaubaren, fehlerhaften Programmen. Software Engineering ist das technische und planerische Vorgehen zur systematischen Herstellung und Pflege von Software, die zeitgerecht und unter Einhaltung der geschätzten Kosten entwickelt bzw. modifiziert wird.

Das **Wasserfallmodell** ist ein lineares, nicht iteratives Vorgehensmodell in der Softwareentwicklung. Es besteht aus den Schritten Ermittlung der Anforderungen (englisch **Requirement Management**) an die Software (als Lastenheft bzw. Pflichtenheft), Software-Entwurf, Implementation, Testen und Wartung der Software. Die Bezeichnung Wasserfall kommt daher, dass diese Phasen oft grafisch als Kaskade dargestellt werden. Das Wasserfallmodell kann dann gut angewendet werden, wenn die genauen Anforderungen an eine Software vorab für die Planung genau festgelegt werden können. In der Medizin ist dies leider häufig nicht ohne weiteres möglich, weshalb modernere Vorgehensmodelle eingesetzt werden. Es ist beim Requirement Management wesentlich, die Anforderungen zu priorisieren und nach dem Grad der Verbindlichkeit zwischen Muss-, Soll- und Kann-Anforderungen zu unterscheiden.

Eine Weiterentwicklung des Wasserfallmodells ist das **V-Modell** (Abb. 2.12), bei dem Entwicklungsphasen der Software und entsprechende Phasen der Qualitätssicherung definiert sind. Am Anfang werden die Anforderungen an das IT-System (sowohl funktionale wie nicht-funktionale Anforderungen) ermittelt und daraus eine System-Architektur und ein System-Entwurf abgeleitet. Dies wird umgesetzt in eine Software-Architektur und als Software-Entwurf implementiert. Implementierung als **Open Source** bedeutet, dass der Quellcode öffentlich zugänglich ist, was Vorteile bietet im Hinblick auf die Erweiterbarkeit und die Überprüfbarkeit des Quellcodes. Im rechten, nach oben führenden Ast werden die Testphasen dargestellt, die den jeweiligen Phasen auf der linken Seite entsprechen: Zunächst ein Test der einzelnen Softwarekomponenten (**Unit-Tests**) und dann gemeinsame Tests von mehreren Komponenten (**Integrations-Tests**). Beim V-Modell sind Iterationen möglich: Wenn sich in den Unit-Tests Fehler zeigen, kann bei Bedarf auch die

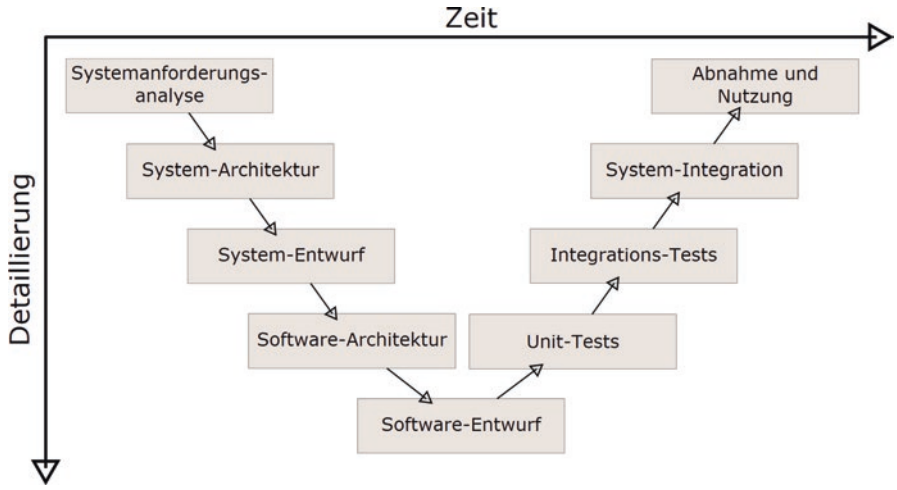


Abb. 2.12 V-Modell: Phasen der Softwareentwicklung von der Anforderungsanalyse über Software-Entwurf und Tests bis zur Nutzung der Software.
Quelle: Michael Pätzold, https://de.wikipedia.org/wiki/V-Modell

Software-Architektur nachgebessert werden. Treten in einer späteren Phase Probleme auf, kann in der korrespondierenden Phase der linken Seite nachjustiert werden, wobei dann die Folgeschritte erneut durchlaufen werden müssen. Sobald das IT-System integriert und abgenommen ist, kann es genutzt werden.

Noch mehr Flexibilität als das V-Modell bietet die **agile Softwareentwicklung**. Es handelt sich hierbei um iterative Verfahren der Softwareentwicklung, die versuchen, den bürokratischen Aufwand der Softwareentwicklung zu reduzieren („flexibler und schlanker“) und zugleich qualitativ hochwertige IT-Systeme zu ermöglichen. Ein typisches Beispiel für agile Softwareentwicklung ist **Scrum**. Die Grundüberlegung besteht hierbei darin, dass ein wesentlicher Teil der Anforderungen eines IT-Systems zu Beginn nicht klar sind und daher Zwischenergebnisse geschaffen werden, die es ermöglichen, die genauen Anforderungen zu klären. Ein Entwicklungszyklus von zwei bis vier Wochen, der Zwischenergebnisse in Form von Produktfunktionalitäten produziert, wird bei Scrum als **Sprint** bezeichnet. Die Liste der Anforderungen, die kontinuierlich verfeinert wird, heißt **Backlog**. Die Einträge im Backlog legt der **Product Owner** fest. Die Anforderungen an die Software können sich bei Scrum somit während der Entwicklung ändern, insbesondere im Rahmen eines sogenannten **Sprint Review** am Ende eines Sprints. Dies ermöglicht eine deutlich größere Flexibilität als beispielsweise beim Wasserfallmodell. Scrum wird meist in kleinen Teams von 3 bis 9 Personen durchgeführt, die sich täglich kurz treffen zum Informationsaustausch (**Daily Scrum**) und die von einem **Scrum Master** geleitet werden. Die bereits geleistete und noch verbleibende Arbeit kann in einem sogenannten **Burn-Down-Chart** (Abb 2.13) visualisiert werden.

Die fertige und veröffentlichte Version einer Software wird als **Release** bezeichnet. Unterschiedliche Versionen einer Software werden üblicherweise mit **Versionsnummern** gekennzeichnet (Abb. 2.14). Eine Versionsnummer setzt sich typischerweise zusammen

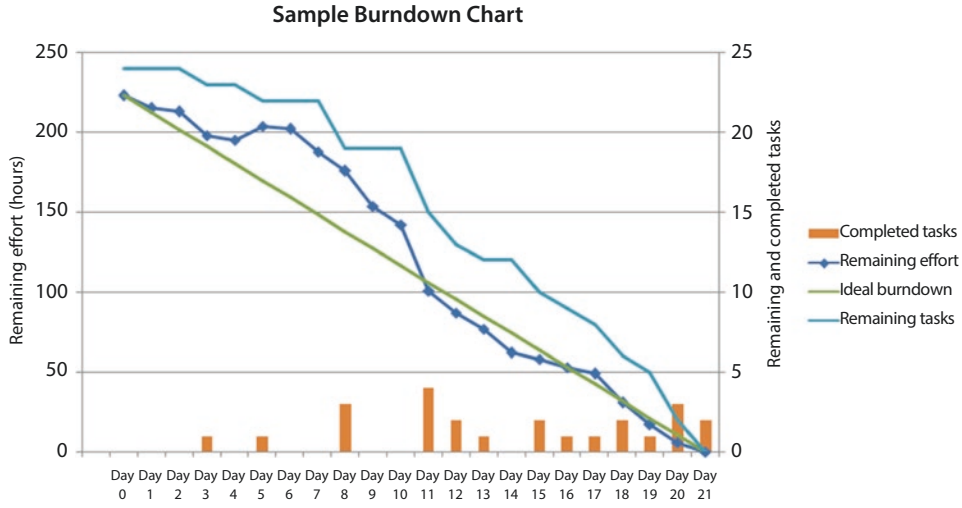


Abb. 2.13 Beispiel für Burn-Down-Chart bei Softwarentwicklung mit Scrum. Quelle: https://de.wikipedia.org/wiki/Burn-Down-Chart

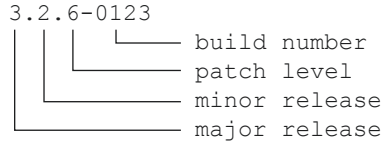


Abb 2.14 Beispiel für Versionsnummer einer Software

aus der Hauptversionsnummer (englisch: major release; große Änderung am Programm), der Nebenversionsnummer (englisch: minor release, kleine Erweiterung des Programms), der Revisionsnummer (englisch: patch level, Version zur Fehlerbehebung) und Buildnummer (englisch: build number, Einzelschritt der Entwicklungsarbeit).

2.11 Aufbau eines Computers und Betriebssysteme

Personalcomputer (**PC**s) haben einen modularen Aufbau. Das Mainboard enthält den Prozessor (englisch central processing unit, abgekürzt **CPU**), der Befehle in Maschinensprache ausführt. Die dafür erforderlichen Daten werden über den Datenbus transportiert. Die Geschwindigkeit der Befehlsausführung hängt von der Taktrate des Prozessors und der Geschwindigkeit des Datenbusses ab. Zur Beschleunigung wird typischerweise ein schneller Zwischenspeicher, der sog. Cache, eingesetzt. Allgemeine Systemeinstellungen, die vor allem beim Startvorgang des Rechners wichtig sind, werden in einem nichtflüchtigen Speicher abgelegt, der **BIOS** (Basic Input/Output System) bezeichnet wird. Beim Speicher unterscheidet man den schnellen **RAM**-Speicher (Random-Access-Memory), der gelöscht wird, sobald der Strom abgeschaltet wird, und den deutlich langsameren, nichtflüchtigen Speicher, meist als Festplatte. Peripheriegeräte werden über Schnittstellen (englisch Interfaces) angesprochen, zum Beispiel USB-Schnittstelle für Scanner und Drucker. Zu den weiteren Komponenten eines PCs gehören Grafikkarte mit Monitor, Soundkarte, Netzwerkkarte und Eingabegeräte wie Maus und Tastatur.

Tabletcomputer (Tablet-PC oder kurz Tablet) sind tragbare flache Computer mit einem Touchscreen. Anstelle einer mechanischen Tastatur gibt es eine Bildschirmtastatur. Die Programme auf Tablets werden **Apps** (von englisch Applications) genannt. Eine Verbindung zum Internet kann über **WLAN** (Wireless Local Area Network) oder über Mobilfunk (zum Beispiel LTE-Netz) erfolgen. Als Massenspeicher werden bei Tablets anstelle von Festplatten Flash-Speicher oder Solid-State-Drives (SSD) eingesetzt. Stromsparende CPUs und Displays, effizientes Betriebssystem und leistungsfähiger Akku sind bei Tablets von besonderer Bedeutung, um geringes Gewicht zu erreichen. Der Übergang von Tablets zu **Smartphones**, also Mobiltelefonen mit Computerfunktionalität, ist fließend, weil eine Reihe von Tablets mit SIM-Karten (Subscriber Identity Module, eine Chipkarte zur Nutzeridentifikation im Mobilfunknetz) ausgestattet werden können.

Nach DIN 44300 versteht man unter einem **Betriebssystem** „die Programme eines digitalen Rechensystems, die zusammen mit den Eigenschaften dieser Rechenanlage die Basis der möglichen Betriebsarten des digitalen Rechensystems bilden und die insbesondere die Abwicklung von Programmen steuern und überwachen“. Man unterscheidet Betriebssysteme nach folgenden Kriterien:

- Single-Tasking: Ein einziges Programm läuft zu einem bestimmten Zeitpunkt.
- Multi-Tasking: Mehrere Programme werden gleichzeitig bearbeitet.
- Single-User: Der Computer steht nur einem einzigen Benutzer zur Verfügung.
- Multi-User: Mehrere Benutzer teilen sich die Computerleistung.
- Single-Processor: Das Betriebssystem unterstützt nur einen Prozessor.
- Multi-Processor: Mehrere Prozessoren werden gleichzeitig eingesetzt.

Der Betriebssystem-Kern, der Basisdienste für alle anderen Teile des Betriebssystems erbringt und mit der Hardware interagiert, heißt **Kernel**. Die sog. Betriebssystem-Shell dient zur Interaktion mit dem Benutzer. Aufgaben, die vom Betriebssystem übernommen werden, sind z. B. Dateiorganisation und -verwaltung, Speicherverwaltung, Sicherheitsfunktionen, Prozessverwaltung und Kommunikation. Beispiele für Betriebssysteme sind Linux [Linux], Android [Android], iOS [iOS], MacOS und Microsoft Windows [Microsoft Windows].

2.12 Mustererkennung und maschinelles Lernen

Informatik-Verfahren werden in der Medizin häufig eingesetzt, um bestimmte Muster in größeren Datenmengen zu erkennen, zum Beispiel in der Bildverarbeitung oder Bioinformatik. Die Herausforderung besteht oft darin, das Muster von Störsignalen oder Rauschen möglichst zuverlässig abzugrenzen. Für diese Fragestellung gibt es eine sehr große Anzahl von verschiedenen Algorithmen, hier wird auf Sekundärliteratur verwiesen. Beim maschinellen Lernen kann man überwachtes Lernen (englisch **Supervised Learning**) und unüberwachtes Lernen (englisch **Unsupervised Learning**) unterscheiden. Supervised Learning bedeutet, dass der Algorithmus eine Funktion aus gegebenen Paaren von Ein- und Ausgaben lernt; die korrekten Ausgaben sind also vorgegeben. Beim Unsupervised Learning wird aus den Eingaben ein Modell erzeugt, das die Eingaben beschreibt und Vorhersagen machen kann.

2.12.1 Neuronale Netze

Neuronale Netze versuchen, die Fähigkeiten von biologischen Nervensystemen durch Nachbildung der biologischen Neuronenstrukturen nachzuahmen. Sie können eingesetzt werden für Klassifikations-, Vorhersage-, Interpolations- und Regelungsprobleme, wobei auch nichtlineare Zusammenhänge modelliert werden.

Ein **künstliches Neuron** erhält ein oder mehrere Eingangssignale, die von außen oder von der Ausgabe anderer Neuronen stammen. Jedem Eingang ist ein bestimmtes Gewicht zugeordnet. Die Differenz aus der gewichteten Summe der Eingangssignale und dem für jedes Neuron vorgegebenen **Schwellenwert** wird als Aktivierungssignal bzw. postsynaptisches Potenzial (**PSP**) bezeichnet. Wenn das PSP größer als Null ist, wird das Neuron aktiviert und aus dem PSP wird über eine sog. **Aktivierungsfunktion** (= Transferfunktion) das Ausgangssignal des Neurons ermittelt. Für die Transferfunktion wird häufig eine **Sigmoid-Funktion** eingesetzt; diese bildet einen beliebigen Wert auf das Intervall]0;1[ab und hat einen S-förmigen Verlauf. Bei **Feedforward-Netzen** (Abb. 2.15) fließen Signale von der Eingabeschicht (**Input Layer**) über eine oder mehrere Zwischenschichten (**Hidden Layers**) zur Ausgabeschicht (**Output-Layer**).

Bevor ein neuronales Netz für Vorhersage- oder Steuerungsprobleme eingesetzt werden kann, müssen seine Parameter (Schwellenwerte und Gewichte) an die jeweilige Aufgabe angepasst werden. Dies wird als **Training** oder Lernphase bezeichnet. Beim **Supervised Learning** wird eine Menge von Konfigurationen (= **Trainingsmenge**) verwendet, die aus Eingabesignalen und den dazugehörigen richtigen Ausgangssignalen besteht. Durch einen Trainingsalgorithmus werden die Gewichte und Schwellenwerte der Neurone schrittweise angepasst. Häufig eingesetzt wird das **Backpropagation**-Verfahren, ein Spezialfall eines allgemeinen Gradientenverfahrens in der Optimierung, basierend auf dem mittleren quadratischen Fehler.

Neuronale Netze werden unter anderem im Rahmen des **Deep Learning** verwendet. Es handelt sich hierbei um Verfahren des **Machine Learning**, die auf dem maschinellen Lernen von Repräsentationen von Daten basieren. Neuronale Netze werden zum Beispiel erfolgreich für die Bilderkennung oder für die Übersetzung von Texten eingesetzt.

Generalisierbarkeit bedeutet, dass korrekte Vorhersagen nicht nur für die Trainingsmenge, sondern auch für eine davon unabhängige Testmenge von Fällen möglich ist. Unter **Over-Fitting** versteht man, dass das Modell zwar gute Ergebnisse bei der Trainingsmenge,

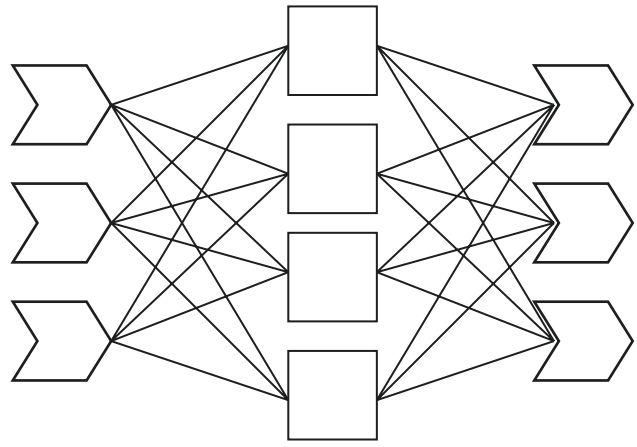


Abb. 2.15 Neuronales Feedforward-Netz mit drei Eingabeneuronen (links), Hidden Layer und drei Ausgabeneuronen (rechts)

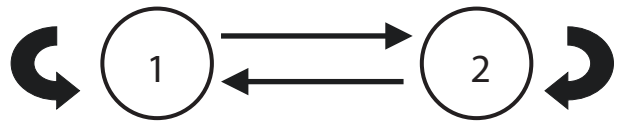
nicht jedoch in der unabhängigen Testmenge liefert. Over-Fitting tritt typischerweise auf, wenn die Anzahl der Trainingsfälle im Verhältnis zum Vorhersagemodell zu klein ist. Als „Fluch der Dimensionalität“ (englisch **Curse of Dimensionality**) bezeichnet man das Phänomen, dass bei nichtlinearen Zusammenhängen auch bei kleiner Anzahl von Variablen sehr viele Trainingsfälle benötigt werden.

Zur Verminderung von Over-Fitting wird die Technik der **Cross-Validierung** eingesetzt: Die vorhandenen Fälle werden aufgeteilt in k gleich große Teilmengen. k–1 dieser Teilmengen bilden die Trainingsmenge, die restliche Teilmenge dient als Testmenge. Das Rechenverfahren wird mit der Trainingsmenge trainiert und anschließend der Fehler in der Testmenge bestimmt. Dieses Verfahren wird k-mal wiederholt (k-fache Cross-Validierung).

2.12.2 Hidden-Markov-Modelle

Hidden-Markov-Modelle werden für die Mustererkennung bei medizinischen Daten eingesetzt. Eine Markov-Kette ist ein stochastischer Prozess, der aus Zuständen und Übergängen zwischen diesen (Transitionen) besteht. Für jede Transition s_x => s_y gibt es eine konstante Übergangswahrscheinlichkeit. Die Folge der auftretenden Zustände der Markov-Kette wird als Zustandsfolge bezeichnet. Die Übergangswahrscheinlichkeiten können in Form einer Matrix dargestellt werden (Abb 2.16).

Ein Hidden-Markov-Modell (**HMM**) ist eine Erweiterung der Markov-Kette, bei der die Zustandsfolge durch einen zweiten stochastischen Prozess in eine Folge von Beobachtungen übergeht. Die Zustandsfolge der Markov-Kette wird hierbei als „hidden state“-Sequenz bezeichnet (Abb. 2.17). Ein anschauliches Beispiel dazu: Man kann ohne Blick auf eine Ampel – nur durch die Beobachtung der Autos – mit einer bestimmten Wahrscheinlichkeit auf die Ampelschaltung schließen. Die „hidden state“-Sequenz ist in diesem Fall die Abfolge der Rot/Grün-Phasen der Ampel, die beobachtete Sequenz ist das Fahren bzw. Stehenbleiben der Fahrzeuge.



p(Übergang) von / nach	1	2
1	0,2	0,8
2	0,8	0,2

Zustandsfolge: 1 2 1 2 2 1

Abb. 2.16 Markov-Kette mit zwei Zuständen, Tabelle der Übergangswahrscheinlichkeiten sowie eine mögliche Zustandsfolge

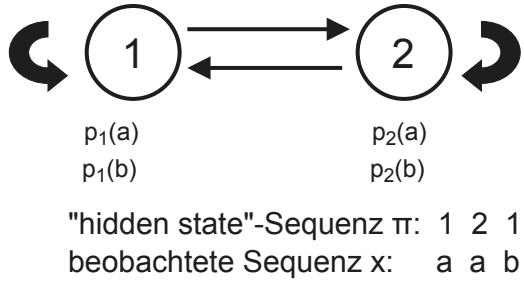


Abb. 2.17 Hidden-Markov-Modell mit zwei Zuständen, einem Beispiel für eine „hidden state"-Sequenz und einer möglichen beobachteten Sequenz

Hidden-Markov-Modelle können zur Erkennung von Sequenzmustern in der Bioinformatik aber auch für die Spracherkennung eingesetzt werden. Mit einem Trainingsset von bekannten Mustern werden die Parameter des HMM optimiert, anschließend kann das HMM zur Erkennung dieser Muster (Klassifikation) eingesetzt werden.

2.12.3 Hauptkomponentenanalyse

Im Rahmen der Mustererkennung können Verfahren der **Dimensionsreduktion** eingesetzt werden, um hochdimensionale Daten besser darstellen und analysieren zu können. Ein in der Medizin häufig eingesetztes Verfahren zur Dimensionsreduktion ist

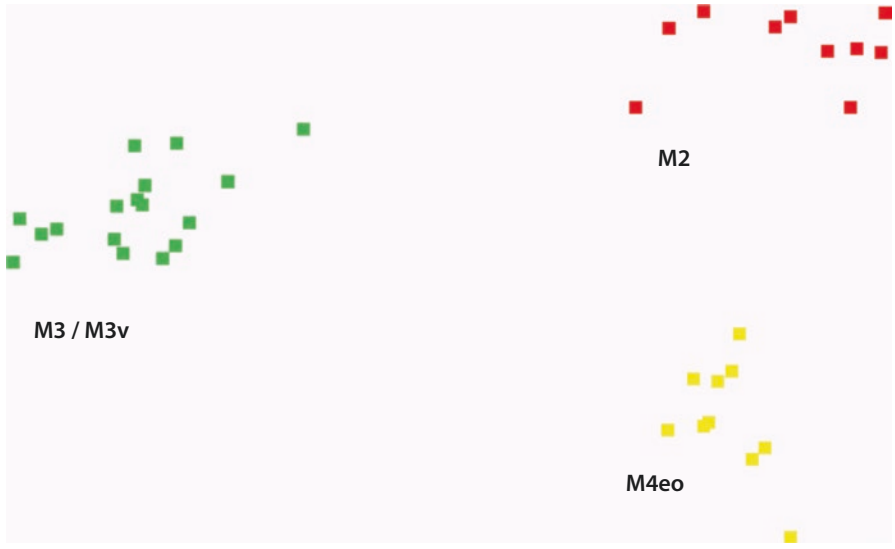


Abb. 2.18 Hauptkomponentenanalyse (PCA) von Genexpressionsdaten. Jeder Punkt entspricht einem Microarray-Experiment mit mehr als 10.000 Meßwerten. Es ist ersichtlich, dass mindestens drei Gruppen von Experimenten vorliegen, die in diesem Beispiel unterschiedlichen Leukämie-Formen entsprechen

die **Hauptkomponentenanalyse** (englisch Principal Components Analysis, abgekürzt **PCA**). Bei der PCA werden aus den hochdimensionalen Datenvektoren Hauptkomponenten ermittelt, die einen möglichst großen Anteil der Varianz der ursprünglichen Vektoren enthalten. Die folgende Abbildung zeigt wie durch PCA Genexpressionsmessungen dargestellt werden können, bei denen je Experiment mehr als 10.000 Meßpunkte erhoben werden (Abb. 2.18).

2.12.4 Clusteranalyse

Verschiedene Arten von Clusteranalysen können eingesetzt werden, um ähnliche Muster in großen Datenbeständen zu erkennen. Das Ziel einer Clusteranalyse ist es, neue Gruppen in den Daten zu identifizieren. Ein häufig genutztes Verfahren ist das **hierarchische**

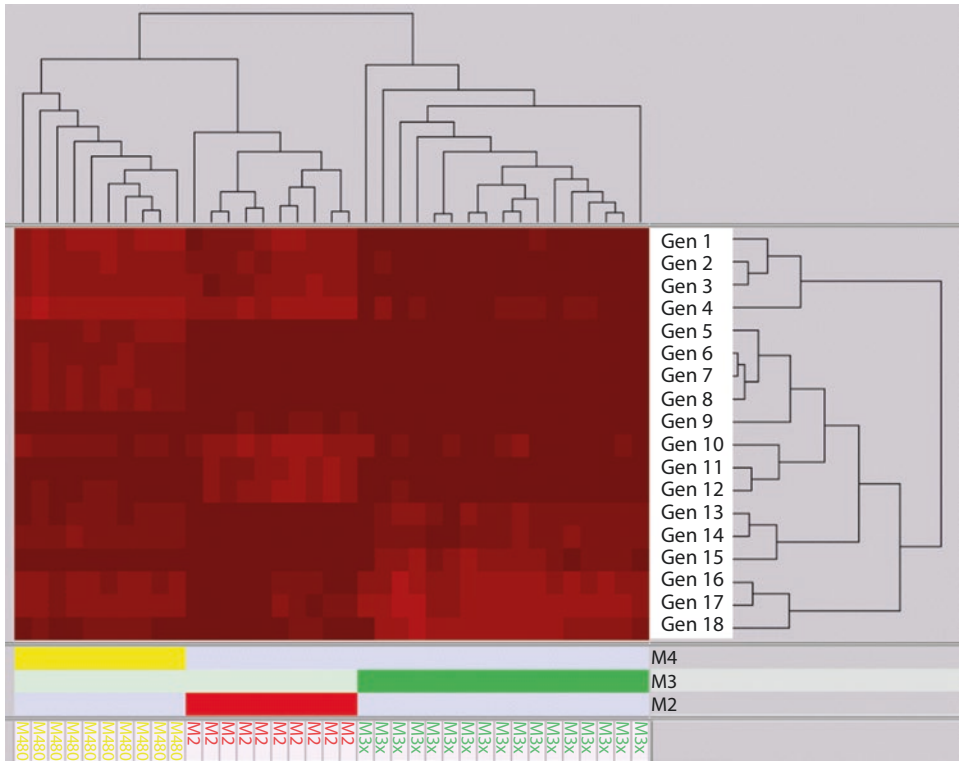


Abb. 2.19 Hierarchisches Clustering von Genexpressionsdaten: Auf der rechten Seite ist eine Liste von Genidentifikatoren angegeben. Die Farbintensitäten in jeder Zeile entsprechen den Expressionswerten des jeweiligen Gens (hellrot = maximale Expression). Jede Spalte entspricht einem Experiment. Die Dendrogramme visualisieren koregulierte Gene bzw. ähnliche Genexpressionsmuster der Experimente

Clustering. Zur Visualisierung der bei einem hierarchischen Clustering entstehenden Baumstruktur kann ein **Dendrogramm** genutzt werden. Das Dendrogramm ist ein Baum, der die hierarchische Zerlegung der Datenmenge in immer kleinere Teilmengen darstellt. Hierarchisches Clustering kann zum Beispiel eingesetzt werden, um Genexpressionsdaten zu analysieren. Dabei werden sowohl die Experimente als auch die Gene nach Ähnlichkeit geordnet und man kann Gruppen von ähnlichen Experimenten als auch von ähnlichen Genen identifizieren (Abb. 2.19). Diese Darstellungsform wird auch **Heatmap** genannt. Bei der Interpretation dieser Heatmaps ist zu beachten, dass die Anzahl der Messwerte je Individuum typischerweise deutlich größer ist als die Anzahl der Individuen und dass somit auch rein zufällige Assoziationen zwischen den Expressionswerten auftreten. Die Sicherheit von Aussagen basierend auf diesen Analysen sollte daher mit speziellen statistischen Verfahren (insbesondere **Permutationstest**; Bestimmung der False Discovery Rate, abgekürzt **FDR**) geprüft werden.

Literatur

[Android] Android, http://www.android.com/. Zugegriffen am 16.12.2016
[Breiman] Breiman L, Friedman J, Stone C, Olshen R (1984) Classification and Regression Trees. Chapman & Hall. ISBN 0412048418
[HTML] Hyper Text Markup Language, https://www.w3.org/html/. Zugegriffen am 16.12.2016
[i2b2] i2b2, https://www.i2b2.org/. Zugegriffen am 16.12.2016
[iOS] iOS, http://www.apple.com/de/ios/. Zugegriffen am 16.12.2016
[Java] Java, http://java.com/. Zugegriffen am 16.12.2016
[JavaScript] JavaScript, http://javascript.com/. Zugegriffen am 16.12.2016
[JSON] JSON, http://www.json.org/. Zugegriffen am 16.12.2016
[Linux] Linux, http://www.linux.org/. Zugegriffen am 16.12.2016
[Microsoft Windows] Microsoft Windows, http://www.microsoft.com/
[MySQL] MySQL Datenbank, http://www.mysql.com/. Zugegriffen am 16.12.2016
[OMOP] OMOP CDM, http://omop.org/CDM. Zugegriffen am 16.12.2016
[PCORnet] PCORnet CDM, http://www.pcornet.org/pcornet-common-data-model/. Zugegriffen am 16.12.2016
[PostgreSQL] PostgreSQL Datenbank, http://www.postgresql.org/. Zugegriffen am 16.12.2016
[Talend] Talend Open Studio, https://sourceforge.net/projects/talend-studio/. Zugegriffen am 16.12.2016
[XML] XML, http://www.w3c.org/XML/. Zugegriffen am 16.12.2016

http://www.springer.com/978-3-662-53327-7

Medizininformatik
Ein Kompendium für Studium und Praxis
Dugas, M.
2017, X, 259 S. 160 Abb., 113 Abb. in Farbe., Hardcover
ISBN: 978-3-662-53327-7