KAPITEL 4

Data Mining

Data Mining wird häufig als Oberbegriff verwendet, um alle Schritte auf dem Weg von Daten (z. B. Messwerten) zu Wissen (d. h. der kontextualisierten Interpretation dieser Daten) zu beschreiben. Diesen Weg mit seinen Schritten hatten wir bereits im letzten Kapitel genauer untersucht. Streng genommen bezeichnet Data Mining allerdings nur den letzten Schritt dieses Wegs. Die Methoden des *Data and Knowledge Engineering*, die verwendet werden, um einen Workflow zu erstellen, der Daten verarbeitet und den *Data Lifecycle* abbildet, wurden mit den zugrunde liegenden Techniken in Kapitel 3 bereits eingeführt: Datenbanken, Datenformate und Methoden.

Nachdem ein solcher Workflow entwickelt wurde, können die Daten mit entsprechenden Methoden ausgewertet werden. Klassischerweise sind das im Data Mining vor allem Methoden zur Klassifizierung und zum Clustering. Im weiteren Sinne zählen alle statistischen Auswertungen dazu, wie zum Beispiel *Text-Mining* oder *Web-Mining*. In den Lebenswissenschaften geht es um Klassifizierung von Bildern (Was wird abgebildet? Ist auf dem CT-Scan ein bösartiger Tumor zu erkennen?), Texten (Gib mir alle wissenschaftlichen Texte, die Thema X behandeln. Welcher Arztbrief verordnet Medikament Y?) oder anderen Daten (Welche Messreihen sind atypisch? Ist das gegebene EKG pathologisch?). Clustering wird vor allem eingesetzt, um einen kompletten Datensatz mit vordefinierten Ähnlichkeiten in Gruppen (sogenannte Cluster) zu unterteilen. Daher ist es eine Art Generalisierung der Klassifikation.

In diesem Kapitel werden vor allem die Grundlagen dieser Methoden erklärt. Wir werden viele Praxisbeispiele in den folgenden Kapiteln sehen, so etwa zur Bilddatenklassifikation im Abschnitt »Objektklassifizierung« auf Seite 191.

Bevor wir uns mit den beiden Themenkomplexen Klassifizierung und Clustering beschäftigen, möchten wir Sie noch einmal auf den *Data Lifecycle* hinweisen, der im Abschnitt »Der Data Lifecycle« auf Seite 67 beschrieben wurde. Er gibt einen guten Überblick über die Herausforderungen, die Ihnen im Data Mining begegnen können.

Klassifizierung

Die Klassifizierung von Daten ist eines der Hauptprobleme im Data Mining. Es geht im Wesentlichen um automatisierte Entscheidungen der Form »Ist dieser Datensatz 0 oder 1?«. Dabei entspricht die Ausprägung 0 einer Aussage und die Ausprägung 1 ihrem Gegenteil. Es handelt sich also um eine einfache Entscheidung. Zum Beispiel: Ist diese Aussage in diesem Satz zu finden? Ist diese Mail Spam oder nicht? Hat der Benutzer die Werbung auf der Webseite gesehen? Wird ein Patient sterben, wenn ich ihm dieses Medikament gebe?

Klassifizierung kann aber auch die Frage beantworten: »Ist dieser Datensatz gleich i für $i \in \{0, ..., n\}$?« Dies führt zu n Klassen oder Kategorien. Es handelt sich also um eine komplexere Entscheidung mit verschiedenen Kategorien. Zum Beispiel: Ist diese E-Mail privat, beruflich, von Freunden oder von der Familie? Hier ist $n = 4$. Ist dieses Gen von einem Menschen, einer Maus ...?

Es muss des Weiteren zwischen beschreibender (*deskriptiver*) und voraussagender (*prädiktiver*) Klassifikation unterschieden werden. *Beschreibende Klassifizierung* behandelt die Aufgabe, ein bestehendes Datenset zu kategorisieren. Das kann man ohne einen vollständigen Workflow optimieren, da die Daten schon vollumfänglich existieren und die Menge abgeschlossen ist. *Voraussagende Klassifizierung* hingehen hat eine Menge von Datensets (ein Trainingsset, einen Gold-Standard ...) und sollte idealerweise auf beliebigen Eingabemengen gut funktionieren. Deswegen muss man in diesem Fall schon ein funktionierendes und vollständiges Datenmodell sowie einen kompletten Workflow vorliegen haben.

Normalerweise liefern Klassifikationen nicht nur eine Zuweisung in Kategorien, sondern auch eine bestimmte Wahrscheinlichkeit. Dies gilt insbesondere für statistische Modelle, aber auch für diskrete Heuristiken (z.B. das Zählen von Häufigkeiten). In aller Regel definiert man eine sogenannte *Cutoff-Wahrscheinlichkeit* als Schwellenwert. Ab dieser gilt eine Zugehörigkeit zu einer Kategorie, darunter nicht.

Zunächst behandeln wir zwei nicht statistische Verfahren: Binning und Hashing. Danach werden einige statistische Verfahren diskutiert.

Binning

Binning ist kurz gefasst der Prozess, stetige Variablen in Kategorien einzuteilen. Es wird auch als *Diskretisierung* von stetigen Variablen bezeichnet. Bevor Sie diese Methode verwenden, müssen Sie die zugrunde liegende Struktur der Eingabedaten analysieren. Einige Daten sind geradezu trivial, trotzdem hat die gewählte Struktur eine immense Auswirkung.

Als Eingabe hat Binning n *Records* $r(i) = \{d(i)_1, ..., d(i)_n\} \in \mathbb{R}$ mit m Dateneinträgen. Eine Binning-Funktion ist eine Funktion $bin : \mathbb{R} \rightarrow \mathbb{C}$ aus der Menge der Records $\mathbb{R}$ in eine Menge von Klassen $\mathbb{C}$. Obacht: Trotz der hier verwendeten Schreibweise sind bei diesen Mengen weder die reellen noch die komplexen Zahlen gemeint.

Zum Beispiel können wir bei Personendaten in der Variablen d_1 den Namen und in d_2 das Alter speichern. Man könnte zwei Kategorien C_0 mit Menschen, die höchstens 50 Jahre alt sind, und C_1 mit Menschen, die älter sind, wählen.

Dann ist die Binning-Funktion gegeben durch: $b(d) = \begin{cases} 0 & d_2 \leq 50 \\ 1 & d_2 > 50 \end{cases}$

Dies können Sie sehr einfach für jedes beliebige Java-Objekt implementieren. Es muss nur eine Funktion für den spezifischen Datentyp geschrieben werden. Wir wollen jetzt eine Klasse `Patient` implementieren und die dazugehörige Binning-Funktion `bin`:

```
public class Patient {
    private String Name;
    private int age;

    public String getName() {
        return Name;
    }
    public void setName(String name) {
        Name = name;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
}

public int bin(Patient input) {
    if (input.getAge() <= 50) {
        return 0;
    } else {
        return 1;
    }
}
```

Durch den Parameter `Patient input` wird ein Objekt vom Typ `Patient` übergeben und durch eine einfache `if-else`-Abfrage entweder 0 oder 1 als Integer zurückgegeben, was dann symbolisch für die Klasse der unter bzw. über 50-Jährigen steht.

Deutlich zu erkennen ist auch der Zusammenhang zwischen Binning und Histogrammen, wie sie in der Statistik häufig verwendet werden. Hier werden Häufigkeitsverteilungen oder stetige Variablen in mehrere Gruppen unterteilt.

Binning-Funktionen können aber auch in Abhängigkeit zu mehr als einem Wert in dem Daten-Record stehen. Zum Beispiel könnte eine Klassifizierung von Teilnehmern einer Studie nötig sein, die sich nicht nur in Abhängigkeit vom Alter ($20 \leq age \leq 35$), sondern auch von einem besonderen Medikament (z. B. Ibuprofen) ergibt. Eine solche Funktion könnte wie folgt aussehen:

```
public int bin(Patient input) {
    if ( (input.getAge() <= 35) &&
         (input.getAge() >= 20) &&
         (input.getDrugs().contains("ibuprofen"))) {
        return 0;
    } else {
        return 1;
    }
}
```

Dazu wurde die Klasse `Patient` durch die Funktion `getDrugs()` ersetzt, die eine Liste mit verabreichten Medikamenten zurückgibt. Hier wird also die Klasse 0 nur dann gesetzt, wenn der Patient zwischen 20 und 35 Jahre alt ist und den Wirkstoff Ibuprofen verabreicht bekommen hat.

Für einige Anwendungsgebiete können solche Funktionen aber nicht präzise angegeben werden, oder es gibt nur Wahrscheinlichkeitswerte. In solchen Situationen müssen Sie einen Schwellenwert (engl. *Cutoff*) definieren.

Das folgende Beispiel illustriert sehr einfaches Text-Mining. Gegeben seien wissenschaftliche Texte als `Document`-Objekte. Ihre Aufgabe ist es nun, zu entscheiden, ob Dokumente eine wissenschaftliche Studie beschreiben. Dabei wird ganz simpel aufgrund von zwei String-Werten ein Score berechnet:

```
public class Document {
    private String text;
    private String authors;
    private String title;
    private Date publishingDate;
    private String journal;
    public String getText() {
        return text;
    }
    public void setText(String text) {
        this.text = text;
    }
    public String getAuthors() {
        return authors;
    }
    public void setAuthors(String authors) {
        this.authors = authors;
    }
    public String getTitle() {
        return title;
    }
    public void setTitle(String title) {
        this.title = title;
    }
    public Date getPublishingDate() {
        return publishingDate;
    }
    public void setPublishingDate(Date publishingDate) {
        this.publishingDate = publishingDate;
    }
    public String getJournal() {
```

```
        return journal;
    }
    public void setJournal(String journal) {
        this.journal = journal;
    }

}

public int bin(Document input, int cutoff) {
    int score = 0;
    // Text contains "This study" -- small indication
    if ( input.getText().contains("This study") ) {
        score++;
    }
    // Title contains "Study on" -- indication
    if ( input.getText().contains("Study on") ) {
        score+=5;
    }
    if (score > cutoff) {
        return 1;
    } else {
        return 0;
    }
}
```

Natürlich kann dieses Beispiel beliebig erweitert werden. Komplexere Ansätze normalisieren einen Score (z.B. das Auftreten von Wörtern durch die Anzahl aller Wörter). Grundsätzlich benötigen Sie eine gute Übersicht über die Daten, um diese in passende Kategorien einzuordnen.

Wenn Ihnen beispielsweise die Verteilung der Daten nicht bekannt ist, können Sie höchstwahrscheinlich keine gleichmäßige Klassifizierung über Binning generieren. Ein einfaches Beispiel ist die Kategorisierung von Patienten in Altersgruppen wie 0 bis 10, 11 bis 20, 21 bis 30 etc. Hier wird eine annähernde Gleichverteilung angenommen. Wenn allerdings die meisten Patienten über 60 sind, wie beispielsweise in einer Demenzstudie, wären andere Klassen (z.B. 0 bis 50, 51 bis 70, 71 bis 75, 76 bis 80 etc.) eine günstigere Wahl.

Der Hauptvorteil von Binning ist seine Einfachheit und die Möglichkeit, es auf allen diskreten Daten anzuwenden. Es eignet sich für binäre Entscheidungen (ja/nein) genauso wie für komplexere Entscheidungen auf Texten und anderen komplexeren Datenobjekten.

Praxisbeispiel 1: Klassifizierung von Dokumenten

Eine Klassifizierungsfunktion soll Dokumente aus der Dokumentendatenbank PubMed[1] in die Kategorien »Clinical Study«, »Review« und »Other research« einteilen. Dazu wird eine Liste von *PubMed-Identifier* (sogenannten PMID) per Kommando-

1 Siehe *https://www.ncbi.nlm.nih.gov/pubmed/*.

zeilenparameter eingelesen, diese wird von der PubMed-API eingelesen, und das Ergebnis wird als CSV-Datei wieder ausgegeben.

PubMed bietet mit »Article Type« schon eine Kategorisierung, die dazu genutzt werden kann, die eigene Funktion zu testen.

Als Beispiel für die Binning-Funktion bietet sich folgender Auszug an. Hierbei wird von jedem Dokument der Titel oder Text in Kleinbuchstaben überführt (`toLowerCase()`) und anschließend mittels `contains()` auf bestimmte Stichwörter oder Formulierungen untersucht. Ausgehend davon werden die Scores für bestimmte Klassen erhöht:

```
public static String bin(Document doc) {
    int clinicalScore = 0;
    int reviewScore = 0;
    int otherScore = 0;

    // Text contains "clinical study" -- indication
    if (doc.getText().toLowerCase().contains("clinical trial")) {
        clinicalScore += 5;
    }

    // Text contains "trial registration" -- indication
    if (doc.getText().toLowerCase().contains("trial registration")) {
        clinicalScore += 5;
    }

    // Text contains "trial registration" -- indication
    if (doc.getText().toLowerCase().contains("this study")) {
        clinicalScore += 5;
    }

    // Text contains "We show" -- small indication
    if (doc.getText().toLowerCase().contains("we show")) {
        otherScore++;
        reviewScore--;
    }

    // Title contains "review" -- indication
    if (doc.getTitle().toLowerCase().contains("review")) {
        reviewScore += 5;
    }

    // Title contains "systematic review" -- indication
    if (doc.getTitle().toLowerCase().contains("systematic review")) {
        reviewScore += 5;
    }

    // Title contains "evidence review" -- indication
    if (doc.getTitle().toLowerCase().contains("evidence review")) {
        reviewScore += 5;
    }

    // Title contains "screening" -- small indication
    if (doc.getTitle().toLowerCase().contains("screening")) {
```

```
            reviewScore++;
        }

        ...

        return highestCategory;
    }
```

Hier wird also eine bestimmte Anzahl von Schlüsselbegriffen im Titel der Publikationen analysiert. Gleiches kann man mithilfe des Texts oder des Abstracts erreichen.

Praxisbeispiel 2: Klassifizierung von Farbwerten

Wenn Farbwerte mittels hexadezimaler RGB-Codierung (z.B. FF0044) angegeben werden, kann man diese in die Klassen Rot, Grün, Blau, Schwarz und Weiß einteilen. Dabei sollte ein Farbwert per Kommandozeilenparameter eingelesen und die Farbklasse sowie eine Komplementärfarbe sollten auf dem Bildschirm ausgegeben werden.

Sie können die Implementierung unter *http://www.camick.com/java/source/HSLColor.java* verwenden, um den RGB-Wert in HSL umzucodieren. HSL (*Hue, Saturation, Luminance*) gibt als Hue-Wert den Farbwert (Ton) zurück. Wird dieser als Array mit drei Einträgen übergeben, kann eine mögliche Klassifizierung wie folgt geschehen:

```
public static String HSLToString(float[] color) {
    float h = color[0];
    float s = color[1];
    float l = color[2];

    if (l < 20) return "black";
    if (l > 80) return "white";

    if (s < 25) return "grey";

    if (h < 60) return "red";
    if (h < 180) return "green";
    if (h < 300) return "blue";
    return "red";
}
```

Die Komplementärfarbe können Sie mit einer einfachen Verschiebung der Farbwerte generieren:

```
color[0] += 180;
if (color[0] > 360) {
    color[0] -= 360;
}
color[2] = 100 - color[2];
```

Diese einfache Kategorisierung zeigt, wie viel Kenntnis der darunterliegenden Daten Sie haben sollten. Selten finden sich sehr schnell einfache und gleichzeitig zu-

treffende Binning-Funktionen. Wenn Sie auf unbekannten Datenmengen arbeiten, verwendet man häufig statt des Binnings das Hashing, das wir uns nun anschauen.

Hashing

Hashing ist eine Technik, die in verschiedensten Anwendungen genutzt wird. In Datenbanksystemen wird sie zum effizienten Ablegen und Abrufen von Daten verwendet, sie kann für Dictionary-Datenstrukturen eingesetzt werden und ebenso in der Kryptografie, um ähnliche oder doppelte Strukturen und Daten zu finden. Gleiches gilt für DNA-Strukturen.

Wie beim Binning haben wir beim Hashing eine Eingabe von n Daten-Records $r(i) = \{d(i)_1, ..., d(i)_m\} \in \mathbb{R}$ (wobei $r(i) = \{d(i)_1, ..., d(i)_m\} \in \mathbb{R}$ die Menge der Daten-Records beschreibt) mit m Einträgen. Daneben existiert eine festgelegte Anzahl von Klassen $\mathbb{C}$. Dann ist eine *Hashfunktion* eine deterministische Abbildung $h : \mathbb{R} \rightarrow \mathbb{C}$.

Damit ist Hashing dem Binning sehr ähnlich, beide liefern eine Klassifizierung. Allerdings sind die Grundlagen ganz unterschiedlich: Hashing wird in der Regel verwendet, wenn eine Datenmenge gar nicht oder nur unvollständig bekannt ist. Dazu versucht Hashing immer eine möglichst gleichmäßige Verteilung in Klassen zu finden. Zum Beispiel werden Patientendaten in Klassen sortiert, die dem ersten Buchstaben ihres Nachnamens entsprechen. Man kann hier an die Hängemappen denken, die sich in manchen Arztpraxen zur Aktenverwaltung bis heute finden. Diese Hashfunktion kann zwar in konstanter Zeit berechnet werden, aber es gibt auch viele *Kollisionen*, d. h., Daten werden auf denselben Punkt abgebildet. Kollisionen liegen also dann vor, wenn mehrere Einträge in dieselbe Klasse abgebildet werden: Gibt es beispielsweise viele Patienten, deren Name mit »A« beginnt, würde man eine Listenstruktur benötigen, die diese Menge an Patienten »auffängt«. Und dann müsste man in einer möglicherweise ziemlich großen Liste nach dem gewünschten Patienten suchen. Dies kann sehr langsam werden und ist der Grund dafür, dass Datenmengen möglichst gleichmäßig auf die Klassen gehasht werden sollten.

Diese Klassen bzw. Orte der Datenablage werden Hashtabelle (engl. *Hash Table*) genannt. Wenn die Datenmenge größer als die Hashtabelle ist – und das ist normalerweise der Fall -, gibt es notwendigerweise Kollisionen. In aller Regel wählt man eine Primzahl als Größe der Tabelle.

Java bietet eine Implementation von Hashtabellen in `java.util.Hashtable` oder `java.util.HashMap`. Der Hauptunterschied zwischen beiden Klassen ist nur für Anwendungen, die Threading nutzen, relevant, da `Hashtable` nicht synchronisiert ist. Beide Klassen benötigen allerdings eine sauber implementierte Hashfunktion.

Eine Hashfunktion muss *deterministisch* sein, das heißt, jeder Aufruf der Hashfunktion auf derselben Eingabe muss denselben Hashwert zurückgeben. Sie muss gleichmäßig sein, was bedeutet, dass die Hashfunktion die Daten so gleichmäßig wie möglich auf die Hashtabelle verteilen muss. Wenn die Anzahl der Kategorien,

also die Kardinalität von $\mathbb{C}$, nicht fest ist, spricht man von einer dynamischen Hashfunktion (engl. *Dynamic Hash Function*).

Die Anzahl der Einträge in der Hashtabelle wird mit $|\mathbb{C}| = M$ notiert. Dabei gibt es M Slots (oder Kategorien) von 0 bis $M - 1$. Jede Abbildung eines Werts $r \in \mathbb{R}$ durch die Hashfunktion h sollte also zu Werten $h(r) = j$ mit $0 \leq j \leq M - 1$ führen.

In aller Regel ist es keine gute Idee, *triviale Hashfunktionen* zu nutzen. Ein Beispiel für eine solche Funktion wäre, 100 Daten-Records mit eindeutigen IDs, die zwischen 0 und 100 liegen, in 100 Einträge in eine Hashtabelle abzubilden. Das würde durch die Funktion $h(r) = id(r)$ passieren. Für kleine Zeichenfolgen wäre es möglich, die Summe aller ASCII- oder UNICODE-Werte jedes Zeichens als triviale Hashfunktion zu nutzen. Damit ist es zwar ziemlich einfach möglich, eine *perfekte Hashfunktion* (d.h. eine injektive Hashfunktion) zu finden, diese perfekten Hashfunktionen werden allerdings bei größeren Datenmengen sehr schnell ineffizient und führen zu riesigen Hashtabellen.

Angenommen, $M = 800$ und $\mathbb{R} = \{1, ..., 100\}$. Mit der Funktion $h(r) = r$ bekommt man eine perfekte Hashfunktion, jeder Eintrag in der Hashtabelle hat null oder einen Eintrag. Dabei verteilt diese Funktion die Daten aber sehr schlecht. Besser wäre beispielsweise eine Abbildung der Form $h(r) = \frac{r}{|\mathbb{R}|}$.

Eine Hashfunktion für natürliche, real verteilte Daten zu finden, ist nicht trivial. So ist beispielsweise die Altersverteilung in der Bevölkerung nicht linear, Gleiches gilt für viele andere Messwerte. Außerdem verfügen Sie vielleicht am Anfang Ihrer Analyse nur über einen Auszug der Daten, was zu weiteren Verzerrungen führen kann. In einem solchen Fall wissen Sie wenig oder vielleicht sogar gar nichts über die Verteilung der Daten.

Deshalb sind häufige Ansätze für Hashfunktionen folgende:

- Modulo für ganze Zahlen: Zum Beispiel benötigt die Funktion $h(r) = r \bmod M$ mit $M \in \mathbb{N}$ nur eine Hashtabelle mit M Einträgen.
- Strings können als Summe von Zeichen mit ganzzahligen Einträgen gesehen werden.

Wir wollen nun Beispiele dazu betrachten, wie man dies konkret in Java umsetzen kann.

Eine neue `HashMap`, die `String` mittels `Integer`-Werten als Index speichert, können Sie wie folgt erzeugen:

```
HashMap<String, Integer> map = new HashMap<>();
```

Mit der `put`-Funktion fügen Sie Werte hinzu:

```
map.put("Bonn", 150);
```

Dies bildet das Wertpaar `"Bonn"` und `150` ab. Dazu berechnet die HashMap intern einen Eintrag in der Hashtabelle, der Index und der interne Hashwert sind also

nicht identisch. Natürlich können Sie auch eine eigene Hashfunktion benutzen. Betrachten wir dazu erneut die Klasse `Patient` und eine Funktion `h`, die den Modulo des Alters berechnet:

```
public class Patient {
    private String Name;
    private int age;

    public String getName() {
        return Name;
    }
    public void setName(String name) {
        Name = name;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
}
public static int h(Patient input) {
    return input.getAge() % 10;
}
```

Damit können Sie eine neue `HashMap` erzeugen und `Patient` mit `Integer`-Werten in die Hashtabelle mappen:

```
HashMap<Patient, Integer> map = new HashMap<>();
```

Um ein neues Objekt `pat` vom Typ `Patient` mit der Hashfunktion `h` hinzuzufügen, nutzen Sie die folgende Zeile:

```
map.put(pat, h(pat));
```

Die meisten Operationen auf `HashMap` haben nur eine konstante Laufzeit. Auf Elemente kann als Schlüssel-Wert-Paare zugegriffen werden.

Neben Binning und Hashing werden vor allem statistische Modelle zur Klassifizierung verwendet. Für dieses Thema wollen wir im folgenden Abschnitt die Grundlagen legen und Beispiele diskutieren.

Statistische Modelle

Um statistische Modelle zu nutzen, bietet es sich an, die *Apache Commons Mathematics Library*[2] zu verwenden. Diese Abhängigkeit können Sie der *pom.xml*-Datei mit den folgenden Zeilen hinzufügen:

```
<dependency>
    <groupId>org.apache.commons</groupId>
    <artifactId>commons-math3</artifactId>
```

2 Siehe *commons.apache.org/proper/commons-math*.

```
    <version>3.6</version>
</dependency>
```

Dieses Paket beinhaltet nicht nur einfache mathematische Funktionen, sondern auch Methoden der Numerik, Verteilung, Statistik, Kurvenanpassung, der linearen Algebra sowie des maschinellen Lernens. Zunächst beschäftigen wir uns mit deskriptiver Statistik.

Deskriptive Statistik

Das Ziel von deskriptiver Statistik im Data Mining ist die systematische Visualisierung von Daten, Häufigkeiten und statistischen Werten. Ohne Grundlagen der deskriptiven Statistik ist es nahezu unmöglich, einen guten Überblick über Daten und ihre Beziehung zueinander zu bekommen.

Das Paket `org.apache.commons.math3.stat.descriptive` hat zwei Klassen, `DescriptiveStatistics` und `SummaryStatistics`.

Bei der Beschreibung der folgenden Inhalte kann es zu einiger babylonischer Sprachverwirrung kommen. Die Fachbegriffe existieren nicht nur in englischer und deutscher Sprache, sondern in beiden auch noch in verschiedenen Ausprägungen. Dazu geben wir hier eine Übersicht: Daten werden normalerweise als Liste von Datensätzen angegeben. Diese können als Tabellen (engl. *Spreadsheets*, *Data Frames*) interpretiert werden. Eine Zeile entspricht dann einem Datensatz (oder: Fall, Beispiel, Instanz, Sample), eine Spalte einem Feature (oder: Attribut, Eingabe, Variable, Messwert). Wir verwenden hier in der Regel die erstgenannten, geläufigeren Begriffe.

In Tabelle 4-1 finden sich Beispieldatensätze, um diese Begriffe zu illustrieren. Der Datensatz bildet eine Tabelle. Hier werden Angestellte eines Krankenhauses dargestellt. Alle Personen haben eine Kategorie (Beruf), einen Namen und ein Alter. Dies sind die Features bzw. Variablen. Drei Zeilen bilden drei einzelne Datensätze. Dies ist nur ein Auszug mit drei Datensätzen aus einem größeren Datensatz.

Tabelle 4-1: Beispieldatensatz, der Angestellte in einem Krankenhaus darstellt.

Kategorie	Name	Alter
Arzt	Hans Müller	28
Arzt	Marco Darms	45
Krankenhausleitung	Marc McMoney	35

Skalenniveaus

Zunächst müssen Skalenniveaus eingeführt werden, da diese zeigen, wie viel »wirkliches Wissen« in einer Variablen gespeichert werden kann. Gleichzeitig muss zwischen Informatik- und Statistiksicht unterschieden werden. Ein Informatiker sieht *Strings, Integer, Arrays* etc. Der Statistiker sieht Skalenniveaus. Jede Variable stellt ein Feature dar, das interpretiert werden muss. Einige Interpretationen können sinnig sein, andere auch völlig falsch.

Angenommen, Sie würden Patientendaten betrachten. Ist dort die Größe der Personen gespeichert, können Sie Aussagen wie »Person A ist größer als Person B« treffen. Ist der Name gespeichert, ergibt eine Aussage der Form »Person A ist mehr Sebastian als Person B« gar keinen Sinn. Deswegen soll hier die Klassifikation nach Stevens[3] eingeführt werden.

Nominalskalierte Variablen

Eine *nominalskalierte Variable* gibt nur Information darüber wieder, ob zwei Einträge gleich sind oder nicht. Beispiele sind die taxonomischen Ränge von Pflanzen, politische Parteien oder die Herkunft von Menschen. Dabei ist die Repräsentation sehr offen. Als Beispiel sei das biologische Geschlecht angeführt. Identifiziert man weiblich mit 1, männlich mit 2 und weitere Ausprägungen entsprechend fortlaufend, kann diese Variable nicht sortiert werden, sondern führt schlicht zu disjunkten, d.h. nicht überlappenden Mengen von Klassen. Nur Gleichheit oder Ungleichheit kann berechnet werden – und das, obwohl die Daten numerisch erscheinen (aber codiert sind).

Ordinalskalierte Variablen

Eine *ordinalskalierte Variable* kann sortiert werden, d.h., sie verfügt über eine bestimmte Ordnung, man kann also eine Reihenfolge auf den Variablen festlegen. Dabei kann keine Aussage darüber getroffen werden, wie der Abstand zwischen zwei Variablen ist. So können Gewinner in einem Fahrradrennen in der Reihenfolge ihres Eintreffens im Ziel sortiert werden. Diese Sortierung sagt aber nichts über den zeitlichen Abstand ihres Eintreffens aus, sondern nur darüber, wer der Gewinner ist, wer der Zweitplatzierte ist etc. Es kann also neben Gleichheit auch eine Reihenfolge berechnet werden.

Intervallskalierte Variablen

Nimmt man Temperaturen in Celsius als Werte, sind diese durch gleichmäßig verteilte Punkte zwischen Gefrier- und Siedepunkt definiert. Der Abstand ist also immer gleich. Die Temperatur in Celsius ist daher eine *intervallskalierte Variable*. Die meisten psychologischen Werte (z.B. IQ-Werte) sind intervallskaliert. So ist der Abstand zwischen zwei Personen mit IQ 50 und 60 der gleiche wie zwischen zwei Personen mit IQ 120 und 130. Aber eine Person mit IQ 120 ist nicht doppelt so intelligent wie die Person mit IQ 60. Es können also Gleichheit, Reihenfolge und Differenz berechnet werden.

Verhältnisskalierte Variablen

Bei *verhältnisskalierten Variablen* kann zusätzlich das Verhältnis (*Ratio*) angegeben werden. Dazu zählen physikalische Messwerte wie Länge, Dauer oder Masse. Hier können Aussagen getroffen werden wie »A ist doppelt so schwer/groß/... wie B«, und es können Gleichheit, Reihenfolge, Differenz sowie ein Verhältnis berechnet werden.

3 Siehe S. S. Stevens. *On the Theory of Scales of Measurement*. 1946.

Diese Skalen ähneln, wie bereits erwähnt, den Variablentypen in Java, allerdings wird häufig nur eine Repräsentation verwendet. Zum Beispiel ist es üblich, statt Strings Integer als Kategorien abzulegen. So kann eine Herkunftsliste erstellt werden, in der 0 der Stadt Bonn entspricht, 1 Köln, 2 Berlin etc. Dabei ist dies auch oft schon ein Indiz für die Interpretation der Daten, wie sie z. B. berechnet oder visualisiert werden.

Zur Illustration soll das kleine Beispiel aus Tabelle 4-2 betrachtet werden. Schüler einer Musikschule wurden per Fragebogen befragt. Angegeben wurde das Alter in Jahren, die Anfahrtszeit zur Musikschule in Minuten, das Transportmittel (0 = zu Fuß, 1 = mit dem Bus, 2 = mit dem Auto) und ob sie Musik mögen (0 = gar nicht, 5 = sehr).

Tabelle 4-2: Auswertung eines Fragebogens, den Schüler einer Musikschule ausgefüllt haben.

Alter	Anfahrtszeit (min)	Transportmittel	mag Musik
8	6	0	5
8	3	0	5
9	16	1	4
8	25	1	5
7	5	0	5
10	15	2	2
8	18	0	2
14	4	1	0
12	1	1	1
7	40	0	4
8	16	0	5
9	15	2	5
9	10	2	4
12	5	1	3
14	10	0	4

Die Skalenniveaus der einzelnen Variablen können relativ einfach angegeben werden: Zeit ist eine physikalische Einheit, also liegt bei Alter und Anfahrtszeit eine Verhältnisskala vor. Das Transportmittel ist eine Liste ohne Reihenfolge und Ordnung – man kann nicht sagen »mit dem Bus ist besser als zu Fuß«. Deshalb ist diese Variable nominalskaliert. Die Liebe zur Musik kann zwar nach Grad sortiert werden, aber die Unterschiede müssen nicht gleich sein. Deshalb ist diese Variable auf Ordinalskalenniveau.

Interessant ist nun die Darstellung in Java: Für alle Variablen können Sie numerische Datentypen einsetzen, in unserem Beispiel sogar durchgängig einfaches `int`, da lediglich ganze Zahlen vorliegen. Allerdings sind bei einem solchen Ansatz der

vollständigen Integer-Codierung Informationen über jene oben beschriebenen Skalen vollständig implizit. Bei `string` (Bus, zu Fuß, Pkw ...) könnte hingegen grundsätzlich eine Nominalskala angenommen werden, gegebenenfalls eine Ordinalskala (Kleinwagen, Mittelklasse, Oberklasse) mit Zusatzinformationen. Diese *Metainformationen*, die letztlich die Daten beschreiben, sind gerade im Kontext von Studiendaten immens wichtig für die korrekte Handhabung der Datensätze, da sie die »legale« Anwendung von statistischen Methoden einschränken. Das Beispiel oben zeigt anhand der Spalte »mag Musik« schließlich, dass die durchaus übliche Codierung einer Ordinalskala als Integer fälschlicherweise dazu verleiten kann, Verhältnisberechnungen anzustellen, die aber nur unter einer Ratioskala sinnvoll sind. Eine Metrik kann aber bei der subjektiven Einordnung einer persönlichen Vorliebe nur annährend angenommen werden.

Häufigkeiten und statistische Kennwerte

Häufigkeitsverteilungen sind die Grundlage für die meisten statistischen Kennwerte. Sie können dazu verwendet werden, Daten zu untersuchen und Lagemaße zu bestimmen, d. h. die Stellen, an denen die meisten Datenpunkte zu finden sind. So kann man mithilfe des *Lagemaßes* zum Beispiel diese Frage beantworten: »Was ist der typische Wert für diese Daten?« Das *Streuungsmaß* hingegen beantwortet die Frage: »Wie breit oder lang ist eine Verteilung?«, oder anders: »Wie scharf umrissen kann eine Klasse sein?«

Die *absolute Häufigkeit* ist die Anzahl eines bestimmten Attributs (die Merkmalsausprägung). Sei X ein Merkmal mit Werten $a_1, ..., a_j, ..., a_k$, dann gibt $n_j = n(a_j)$ die absolute Häufigkeit eines Attributs a_j an.

Die *relative Häufigkeit* eines Attributs a_j ist gegeben durch $h_j = h(a_j) = \frac{n_j}{n}$, wobei n die Kardinalität des Merkmals bzw. die Gesamthäufigkeit oder der Umfang der Stichprobe ist.

Wenn Sie noch einmal einen Blick auf die Beispieldaten in Tabelle 4-1 werfen, können Sie verschiedene Häufigkeiten berechnen. Die absolute Häufigkeit für Ärzte ist $n(doctor) = 2$ und für Leiter $n(manager) = 1$. Die relativen Häufigkeiten hingegen sind gegeben durch $h(doctor) = \frac{2}{3}$ und $h(manager) = \frac{1}{3}$.

Diese Art von Statistik wird univariate bzw. deskriptive Statistik genannt, da sie stets nur eine Merkmalsausprägung bzw. Variable untersucht. Dabei erbt die Klasse `DescriptiveStatistics` von `StatisticalSummary`. Ein mögliches Beispiel sieht wie folgt aus:

```
DescriptiveStatistics statistics = new DescriptiveStatistics();
statistics.addValue(new Float(0.5));
statistics.addValue(new Float(3.5));
statistics.addValue(new Float(5));

System.out.println(statistics.getMean());
```

Damit wird das arithmetische Mittel der drei `Float`-Werte errechnet, die dem Objekt `statistics` hinzugefügt wurden. Um einen guten Überblick über die Daten zu bekommen, ist es in aller Regel sehr hilfreich, ein *Histogramm* zu berechnen. Dies repräsentiert die Verteilung der Daten. Für n Klassen wird die Anzahl der Attribute gegeben, die in dieser Klasse zu finden sind. Damit ist dieses Verfahren sehr nah an den Klassifizierungen, die bereits ein paar Abschnitte vorher eingeführt wurden.

Das folgende Beispiel berechnet ein solches Histogramm:

```
final int classes = 2;
long[] histogram = new long[classes];
EmpiricalDistribution distribution = new EmpiricalDistribution(classes);
distribution.load(statistics.getValues());
int k = 0;
for(SummaryStatistics stats: distribution.getBinStats())
{
    histogram[k++] = stats.getN();
}

for (int i=0; i<histogram.length; i++) {
    System.out.println(i+" - "+histogram[i]);
}
```

Ein Histogramm gibt vielfältige Informationen über die Verteilung. Häufige Fragen sind: »Wo liegen die meisten Werte?« und: »Folgen die Werte einer bestimmten Verteilung?« Ein Histogramm ist eine (auch grafische) Hilfestellung, um Verteilungen, die in stark vereinfachenden Kennwerten wie dem arithmetischen Mittel gleich sind, differenzierter betrachten zu können. Ein recht extremes Beispiel wäre hier eine klassische unimodale Gauß'sche Normalverteilung im Vergleich zu einer bimodalen Verteilung mit dem gleichen Mittelwert. Dieser ist im ersten Fall aussagekräftig, im zweiten aber überhaupt nicht repräsentativ für die Gegebenheiten.

Wir hatten im letzten Abschnitt schon den Zusammenhang zwischen einem Histogramm und einer möglichen Klassifizierung beschrieben und auch diskutiert, welche Anzahl von Klassen und welche Grenzen sinnvoll sind. Für die reine Fragestellung, ob ein Wert eher niedrig oder eher hoch ist, würde der Vergleich zum arithmetischen Mittel im obigen Beispiel ausreichen. Ginge es allerdings um eine Quantifizierung, *wie typisch* ein Wert für seine zugeteilte Klasse ist, wäre der Klassifikator im Fall der bimodalen Verteilung schlecht gewählt; hier würde besser die Abweichung zum näher liegenden lokalen Maximum herangezogen werden. Diese Entscheidung lässt sich im Vorfeld unter Zuhilfenahme des Histogramms deutlich gezielter fällen.

Neben dem oft genutzten Mittelwert können auch andere Kennwerte sehr einfach berechnet werden; sie geben entsprechende Auskunft für mögliche Klassen. Nicht alle Kennwerte sind für alle genannten Skalenniveaus verfügbar.

Ab **Nominalskala** kann der *Modalwert* berechnet werden. Dies ist der häufigste Messwert einer Verteilung. Er wird mit *mod* oder x_{mod} bezeichnet.

Ab **Ordinalskala** kann der *Median* berechnet werden. Das ist der Wert, für den gilt, dass mindestens 50 % aller Messwerte kleiner oder gleich diesem Wert sind; damit sind 50 % auch größer oder gleich. Eine *Quantile* ist eine Verallgemeinerung dieses Konzepts. Ein Quantil $q_{(p)}$ ist der Wert, für den gilt, dass ein bestimmter Anteil, angegeben durch p, an Werten kleiner oder gleich ist. Für eine 2-Quantile gibt es also zwei Gruppen Q_1 und Q_2 mit 50 % der höchsten und der niedrigsten Werte. Das ist der Median.

Andere Spezialfälle sind die *Quartile* (4-Quantile), die *Dezile* (10-Quantile) und die *Perzentile* (100-Quantile). Der Grad der *Quantelung* bestimmt also die Anzahl der Klassen, die für eine Klassifizierung verwendet werden können. Gerade die Perzentile ist ein gut vermittelbares Maß, da sich Messwerte direkt als Prozentwert ausdrücken lassen: p_{80} im Kontext einer Körpergewichtsmessung beschreibt, dass 80 % einer Vergleichskohorte (z. B. gleichaltrige Kinder) leichter und 20 % schwerer sind als der Proband. Ein weiteres wichtiges Maß ist der Interquartilabstand IQA (oder IQR, R für *Range*): Dies ist die Breite des Bereichs vom zweiten bis dritten Quartil – in ihm liegen die mittleren 50 % aller Werte (25 % bis 75 %). Er ist damit ein Streuungsmaß; je breiter der IQR, desto gestreuter sind die Werte.

Ab **Intervallskala** kann das *arithmetische Mittel* (oft auch Durchschnitt oder Mittelwert genannt) berechnet werden. Es wird mit

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$$

angegeben.

Wichtig ab Intervallskalenniveau ist auch die *Varianz*, die mittlere quadratische Abweichung der Messwerte vom Mittelwert. Sie wird als Schätzer für Werte benutzt, wobei sie die Differenz zwischen der Schätzung (z. B. dem Mittelwert) und den eigentlichen Werten betrachtet. Damit werden die Werte nicht null, da die Differenzen quadriert werden:

$$s^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2$$

Die *Standardabweichung* ist die Wurzel $s = \sqrt{s^2}$ der Varianz. Damit haben die Standardabweichung und die Daten dasselbe Skalenniveau.

Wir kommen noch einmal auf das Beispiel in Tabelle 4-2 zurück. Sie können das arithmetische Mittel, die Varianz und die Standardabweichung für alle Variablen mit metrischem Skalenniveau, d. h. mindestens Intervallskala, berechnen.

$$\bar{age} = \frac{1}{15}\sum_{i=1}^{15} age_i = 9.35$$

$$\bar{time} = \frac{1}{15}\sum_{i=1}^{15} time_i = 12.6$$

Für die Varianz ergeben sich die folgenden beiden Werte:

$$s_{age}^2 = \frac{1}{15}\sum_{i=1}^{15}(age_i - \bar{age})^2 = 5.552$$

$$s_{time}^2 = \frac{1}{15}\sum_{i=1}^{15}(time_i - \bar{time})^2 = 102.971$$

Und entsprechend kann die Standardabweichung als Wurzel berechnet werden:

$$s_{age} = \sqrt{s_{age}^2} = 2.356$$

$$s_{time} = \sqrt{s_{time}^2} = 10.147$$

Für die beiden anderen Variablen können Sie nur den Modalwert berechnen:

$mod(transport) = 0$

$mod(music) = 5$

Des Weiteren können absolute und relative Häufigkeiten für beide Variablen berechnet werden. Für die Werte von »mag Musik« würde eine Auswertung wie folgt aussehen:

Wert	Absolute Häufigkeit	Relative Häufigkeit
0	1	6,7
1	1	6,7
2	2	13,3
3	2	13,3
4	3	20
5	6	40

Für jedes Objekt des Typs `DescriptiveStatistics` steht die Funktion `toString()` zur Verfügung. Damit kann ein vollständiger Überblick über die deskriptive Statistik erzeugt werden. Mittels `println` wird zum Beispiel der folgende Output erzeugt:

```
DescriptiveStatistics:
n: 3
min: 0.5
max: 5.0
mean: 3.0
std dev: 2.29128784747792
median: 3.5
skewness: -0.9352195295828245
kurtosis: NaN
```

Für diese Kennwerte – und noch einige mehr – existieren entsprechende Methoden. Die JavaDocs stehen unter *http://commons.apache.org/proper/commons-math/*

javadocs/api-3.3/org/apache/commons/math3/stat/descriptive/DescriptiveStatistics.html zur Verfügung und geben einen vollständigen Überblick.

Ergänzend können Sie `Frequency`-Objekte benutzen. Diese können ähnlich wie `DescriptiveStatistics` mit der Funktion `addValue()` gefüllt werden. Dieses Objekt stellt verschiedene Funktionen wie `getCumFreq()` für kumulative Häufigkeiten zur Verfügung. Die JavaDocs finden Sie unter *http://commons.apache.org/proper/commons-math/javadocs/api-3.3/org/apache/commons/math3/stat/Frequency.html*, sie bieten weiterführende Informationen. Erneut liefert `toString()` einen aussagekräftigen Output in Form einer Häufigkeitstabelle:

```
Value           Freq.           Pct.            Cum Pct.
0.5             1               25%             25%
3.5             1               25%             50%
5.0             2               50%             100%
```

Hier wurden alle Werte gezählt. Wenn Intervalle statt Werten genutzt werden, erzeugt dies ein Histogramm.

Diese statistischen Grundlagen können Sie zur Analyse von Datenmengen, aber auch direkt zur dynamischen Klassifizierung verwenden.

Clustering

Ein Clustering-Verfahren verfolgt das Ziel, eine Menge von Objekten in Teilmengen, sogenannte Cluster, zu zerlegen. Objekte innerhalb eines Clusters sollen zueinander mehr Ähnlichkeit haben als zu solchen, die in anderen Clustern liegen. Die Teilmengen sind vorher nicht bekannt. Es findet also *keine* Klassifizierung in *vorher* festgelegte Klassen statt.

Je nach Anwendungsfall können die Objekte zu einem eindeutigen Cluster oder zu mehreren gehören, eventuell sogar nur mit einer gegebenen Wahrscheinlichkeit. Bei der Zuordnung von Objekten zu einem eindeutigen Cluster spricht man von *Hard Clustering*, ansonsten von *Soft Clustering*. Hier soll zunächst angenommen werden, dass die Cluster disjunkt sind, Objekte also einem eindeutigen Cluster zugeordnet werden können.

Klassifizierung versus Clustering

Da es häufig zu Verwirrung zwischen den Begriffen Clustering und Klassifizierung kommt, noch einmal eine Klarstellung: Bei der Klassifizierung stehen die Kategorien und ihre Anzahl *vorher* fest. Beim Clustering werden die Kategorien erst nach der Erstellung der Mengen gebildet, das heißt, dass ein Clustering gegebenenfalls die Anzahl der Cluster festlegen kann, aber niemals deren Definition.

Graph-basiertes Clustering

Graphen werden noch wesentlich ausführlicher in Kapitel 5, *Netzwerkanalyse: Graphen mit Java*, dargestellt. Deshalb geben wir an dieser Stelle nur eine kurze

Einführung in das spezielle Clustering. Graphen bieten sich zur Darstellung von sehr vielen Problemen an. Im folgenden Kapitel wird auf das Auffinden von Cliquen und stabilen Mengen eingegangen. Bereits das kann schon als Clustering angesehen werden. Es gibt allerdings eine Fülle weiterer Clustering-Algorithmen. Es kann grundsätzlich unterschieden werden zwischen dem Clustering von gewichteten und gerichteten Graphen. Ungerichtete und ungewichtete Graphen werden in aller Regel in Cliquen oder stark zusammenhängende Subgraphen zerlegt.

Nicht alle Graphen oder Arten von Graphen zerfallen in natürliche Cluster. Trotzdem gibt ein Clustering-Algorithmus immer Cluster aus – ob diese sinnvoll sind oder nicht. So ist es nötig, die Eingabe zu betrachten, da im schlimmsten Fall eine zufällige Partition in Cluster generiert wird. Wird im ersten Schritt eine dem Problem angemessene Ähnlichkeitsfunktion gewählt, wird auch der Graph, der diese Daten repräsentiert, in nicht zufällige Cluster zerfallen. So gibt es auch keine allgemeingültige Definition, um Graphen in Cluster zu zerlegen – die Definition hängt immer von den Eingabedaten ab.

Auf einige Problemstellungen wird in Kapitel 5, *Netzwerkanalyse: Graphen mit Java*, eingegangen: Wie findet man stabile Mengen oder Cliquen? Wir schauen uns jetzt noch das Clustering anhand von verschiedenen Messverfahren an. Hierzu wird jedem Knoten v im Graphen G ein Label $lab : V \rightarrow \mathbb{N}$ gegeben, das den Cluster bezeichnet.

Dazu bietet sich das Package `org.jgrapht.alg.scoring` an, das folgende Messmethoden beinhaltet:

- Alpha Centrality
- Betweenness Centrality
- Closeness Centrality
- Clustering Coefficient
- Coreness
- Harmonic Centrality
- PageRank

Diese Methoden sind unterschiedlich und liefern dementsprechend auch völlig andere Ergebnisse, d.h., es muss für jeden Anwendungsfall getestet werden, welche Methode funktioniert. Denn ob ein Knoten wirklich »wichtig« ist, hängt von der Anwendung ab, und daraufhin sollte auch die Messmethode gewählt werden. Die Betweenness Centrality misst beispielsweise auch, wie oft ein Knoten auf einem kürzesten Pfad zwischen anderen Knoten liegt. Damit misst es, ob ein Knoten eine Brücke zwischen verschiedenen Knoten des Netzwerks darstellt. Die *Eigen Centrality* (auch: *Eigenvector Centrality*) bemisst zwar ebenfalls den Einfluss eines Knotens aufgrund der Anzahl von Verbindungen zu anderen Knoten im Netzwerk, nutzt aber auch die Information darüber, wie viele Verbindungen der Knoten selbst hat, und iteriert diesen Wert über alle Nachbarn.

Für einen Graphen `G` können Sie die Messmethoden wie folgt anwenden:

```
VertexScoringAlgorithm<String, Double> pr = new PageRank<>(G, 0.85, 100, 0.0001);
```

Hier wird beispielsweise ein *PageRank* verwendet. Die verschiedenen Methoden haben unterschiedliche Parameter, abhängig von der Art der Berechnung. PageRank verfügt über die folgenden Parameter:

```
double dampingFactor, int maxIterations, double tolerance
```

Für Harmonic Centrality stehen dagegen folgende Parameter zur Verfügung:

```
boolean incoming, boolean normalize
```

Die Dokumentation der einzelnen Methoden finden Sie unter *https://jgrapht.org/javadoc/org/jgrapht/alg/scoring/package-summary.html*.

Auf die einzelnen Werte können Sie mit der Methode `getVertexScore` zugreifen:

```
pr.getVertexScore("a")
```

Mit einer entsprechenden Iteration über alle Knoten kann das Clustering erfolgen.

K-Means

K-Means ist einer der bekanntesten und am meisten verwendeten Algorithmen zur Berechnung eines Clusterings von Daten. Der Algorithmus wurde schon 1967 eingeführt.

Als Eingabe benötigt K-Means Daten mit einzelnen Datenwerten und eine positive Zahl $k \in \mathbb{N}^+$, die die Anzahl der gesuchten Cluster angibt.

Es bietet sich an, die freie Bibliothek WEKA zu verwenden (siehe *https://www.cs.waikato.ac.nz/ml/weka/*). Sie können WEKA durch folgenden Eintrag in der *pom.xml* zu Maven hinzufügen:

```
<dependency>
    <groupId>nz.ac.waikato.cms.weka</groupId>
    <artifactId>weka-stable</artifactId>
    <version>3.8.0</version>
</dependency>
```

Die einzelnen Datenpunkte können Sie einer Instanz wie folgt hinzufügen:

```
Attribute a1 = new Attribute("Value");
ArrayList<Attribute> attrList = new ArrayList<Attribute>();
attrList.add(a1);

Instances daten = new Instances("Daten", attrList, 0);
```

K-Means kann nun folgendermaßen verwendet werden, wobei einer Integer-Variablen `k` die Anzahl der gewünschten Cluster übergeben wird:

```
int k = 4;
SimpleKMeans kMeans = new SimpleKMeans();
```

```
kMeans.setNumClusters(k);
kMeans.buildClusterer(daten);
```

Die einzelnen Werte können dann mit `get()` abgerufen werden:

```
cluster=kMeans.clusterInstance(daten.get(j));
```

Durch eine entsprechende Iteration über alle Datenpunkte ergibt sich das Clustering.

Der Standard für die Abstandsfunktion zwischen den einzelnen Punkten in der Datenmenge ist die euklidische Distanz. Eine Distanzfunktion kann wie folgt erstellt und dem Clustering übergeben werden:

```
EuclideanDistance df = new EuclideanDistance();
kMeans.setDistanceFunction(df);
```

Andere Distanzfunktionen finden Sie im Package `weka.core`.

Übungsaufgaben

Die Lösungen für diese Übungsaufgaben finden Sie unter *https://github.com/jd-s/Java-fuer-die-Life-Sciences-Loesungen*.

Übungsaufgabe 4.1 – Data Mining und Klassifikation mit Binning

Erstellen Sie eine Klassifizierungsfunktion mit einer Bewertung, die Dokumente in der Datei in die Kategorien »Klinische Studie«, »Review« und »Sonstige Forschung« einteilt. Ihre Anwendung soll eine Liste von PubMed-Identifiern[4] (gegeben in einer Datei oder direkt per Kommandozeilenargument) lesen, die Daten aus PubMed über deren API[5] holen und eine Exportdatei (definiert über ein Kommandozeilenargument, CSV-Datei) mit der Klassifizierung schreiben.

Stellen Sie eine ausreichende Dokumentation über Kommandozeilenargumente und Kategorien zur Verfügung. Sie dürfen den von PubMed bereitgestellten Datentyp `Article Type` nicht verwenden – aber ein Blick darauf könnte helfen, die Binning-Funktion zu verbessern und Ideen zu entwickeln.

Wie präzise ist Ihr Ansatz? Welche Schwierigkeiten treten auf?

Übungsaufgabe 4.2 – Binning II

Implementieren Sie eine Klassifizierungsfunktion, die hexadezimale RGB-Farben (z. B. `FF0044`) als Rot, Grün, Blau, Schwarz oder Weiß klassifiziert. Die Anwendung

4 Siehe *https://www.ncbi.nlm.nih.gov/pubmed/*.

5 Siehe *https://www.ncbi.nlm.nih.gov/home/develop/api/*.

soll einen Kommandozeilenparameter (`--color`) entgegennehmen und die Farbklasse sowie die Komplementärfarbe auf dem Bildschirm ausgeben.

Übungsaufgabe 4.3 – Hashing

Erstellen Sie eine `ArrayList` mit mindestens 30 zufälligen ganzen Zahlen. Erstellen Sie zwei Instanzen von `HashMap<Integer, Integer>`. Implementieren Sie ein Hashing der Zufallswerte mit einer Modulo-Hashfunktion und einer Mid-Square-Methode. Was funktioniert besser? Erklären Sie die Ergebnisse.

Übungsaufgabe 4.4 – Binning und Hashing

Nennen Sie zwei Beispiele für Binning und Hashing. Warum ist Binning Hashing, aber nicht jedes Hashing ein Binning? Erklären Sie die Unterschiede und Vorteile der einzelnen Methoden.

Übungsaufgabe 4.5 – Kombination verschiedener Datenquellen

Wir wollen eine neue Kombination aus zwei Proteindatenbanken, *https://www.ncbi.nlm.nih.gov/protein* und *https://www.uniprot.org*, entwickeln. Werfen Sie einen Blick auf die APIs der einzelnen Datenbanken und diskutieren Sie die möglichen Ausgaben. Es sollen alle relevanten Metadaten gespeichert werden. Machen Sie einen Vorschlag für ein gemeinsames Datenschema und diskutieren Sie, welche Daten auf das Schema abgebildet werden können und welche nicht.

Übungsaufgabe 4.6 – Grundlegende Statistiken

Wir wollen versuchen, einige Auswertungen des Artikels »Two replications of an investigation on empathy and utilitarian judgement across socioeconomic status« (*https://www.nature.com/articles/sdata2016129*) zu reproduzieren. Alle Daten sind unter *https://osf.io/sgjpy/* verfügbar. Wichtig: Sie sollten die Daten vorher sorgfältig analysieren.

- Die Anwendung soll einen Kommandozeilenparameter entgegennehmen, der eine CSV-Datei angibt. Die Daten sollen genau so wie in dem oben genannten Repository formatiert sein. Die beiden im Artikel erwähnten Replikationen können durch die Spalte *sample* identifiziert werden.
- Beschreiben Sie die Variablen und erklären Sie, welche Messgröße sie haben.
- Wir wollen beide Gruppen vergleichen und benötigen die folgende Ausgabe:
  - Ein aussagekräftiges Histogramm zum Vergleich des Alters.
  - Häufigkeiten und statistische Werte von *Age, Gender, Ethnicity, Religiousity, Political Aff, Income, Education* und *Current Country*.

Übungsaufgabe 4.7 – Klassifizierung und Statistik

Wir werfen einen Blick auf einen unter *https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/28204&version=1.0* frei verfügbaren Datensatz. Das ist eine häufige Situation: Die Daten sind nicht in einem offenen oder einfachen Format verfügbar.

- Beschreiben Sie, wie diese Daten in ein offenes Format (z. B. ODT, Text, CSV o. Ä.) umgewandelt werden können. Dieses Format soll leicht von einer Java-Anwendung zu lesen sein.
- Die Anwendung soll alle Dateien einlesen, die per Kommandozeile übergeben werden. Geben Sie dazu Beispielparameter an.
- Es sollen alle gängigen deskriptiven Kennzahlen der Teilnehmer und ein aussagekräftiges Histogramm für das Bildungsniveau und das Alter ausgegeben werden.
- Schreiben Sie einen wortbasierten Klassifikator, der die Daten in zwei Klassen einteilt nach den Kriterien, ob ein Text mehr oder weniger formal ist und ob ein Text Hochschulwortschatz verwendet oder nicht. Diese Klassifizierung muss nicht perfekt sein, aber Sie sollten zumindest drei Indikatoren für jede Klasse angeben.
- Welche Skalenniveaus werden für Alter, Geschlecht und Bildungsniveau verwendet? Welche für die beiden neuen Variablen? Welcher Korrelationskoeffizient kann errechnet werden, um die Korrelation zwischen all diesen Variablen zu identifizieren? Berechnen Sie alle möglichen Korrelationen und diskutieren Sie, ob es eine Korrelation gibt oder nicht.