Künstliche Intelligenz verstehen

Eine spielerische Einführung

DIE LESEPROBE

Kapitel 8
K-nächste-Nachbarn

Eine Tafel Schokolade hat mehr gemeinsame Eigenschaften mit einem Stück Torte als mit einer Karotte. Eine Tomate wiederum hat viel mehr Gemeinsamkeiten mit der Karotte als mit Schokolade. In diesem Kapitel zeigen wir, wie Sie solche Ähnlichkeiten und Nachbarschaftsverhältnisse in Zahlen fassen und diese nutzen können, um Ordnung in große Datenbestände zu bringen und Zusammengehörigkeiten darin zu erkennen.

Worum es in diesem Kapitel geht

- Zahlenförmige Daten lassen sich durch *Datenpunkte* darstellen.
- Distanzen zwischen Datenpunkten können wir mit einem einfachen geometrischen Verfahren berechnen.
- Der *k-nächste-Nachbarn-Algorithmus* nutzt solche Distanzmessungen, um Datenpunkte zu klassifizieren.

Angenommen, ein Datenbestand versammelt Nährwertangaben, die jeweils zwei Werte enthalten, einen für den Zucker- und einen für den Fettgehalt. Dann kann es sehr hilfreich sein, jedes einzelne dieser Wertepaare als einen *Datenpunkt* auf eine Fläche zu zeichnen. Dabei können wir den Zuckergehalt als Position des Punkts in der Waagerechten und den Fettgehalt als Position des Punkts in der Senkrechten nehmen.

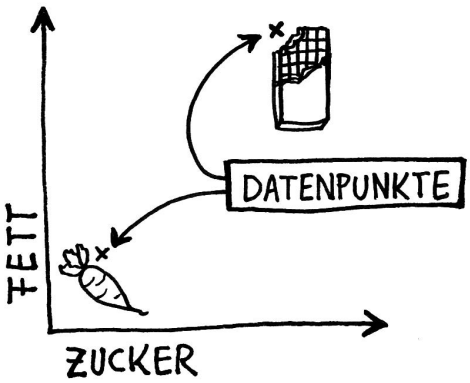


Abbildung 8.1 Messwerte lassen sich als Punkte darstellen.

Ein riesiger Vorteil dieser Darstellung: *Ähnlichkeit* zwischen Datenpunkten lässt sich jetzt *als räumliche Nähe* ablesen und berechnen: Je näher sich zwei Punkte auf der Fläche sind, umso so größer ist die Ähnlichkeit:

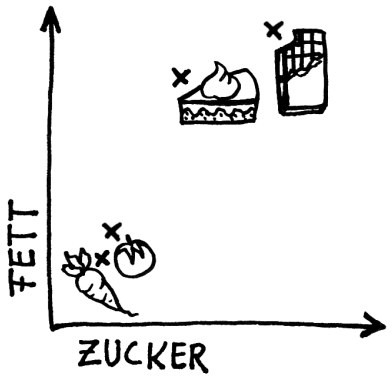


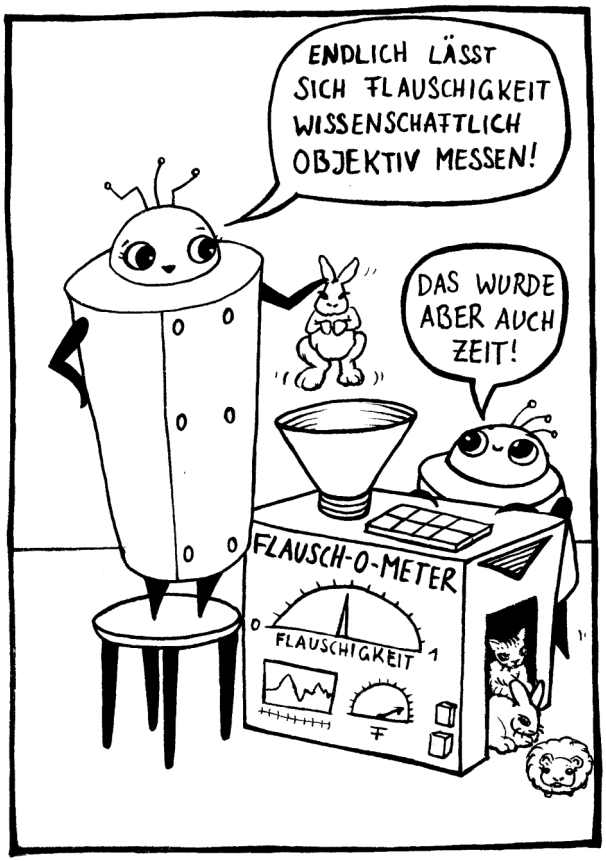
Abbildung 8.2 Räumlich nahegelegene Punkte haben ähnliche Eigenschaften.

Für dieses Kapitel haben wir uns ein alltagsnahes Anwendungsbeispiel ausgedacht, das insbesondere Tierfreundinnen und -freunden anschaulich sein wird: Es geht um die Identifizierung von Tierarten anhand der Eigenschaften *Niedlichkeit* und *Flauschigkeit*.

8.1 Häschen, Igel, Vogelspinne oder Hai?

Das Beispiel geht von folgenden Annahmen aus:

- Sie haben zwei Messgeräte konstruiert, mit denen Sie die Eigenschaften *Flauschigkeit* und *Niedlichkeit* jeweils auf einer Skala von 0 bis 1 präzise messen können. Das Messverfahren ist selbstverständlich ganz und gar harmlos, es kitzelt höchstens ein wenig.
- Mittels dieser Geräte haben Sie jeweils fünf Tiere der Spezies Vogelspinne, Häschen, Hai und Igel vermessen und entsprechende Datenpunkte erzeugt: `{niedlichkeit: 0.90, flauschigkeit: 0.90, spezies: "haeschen"}`. Sie besitzen also einen Datenbestand mit 20 Datenpunkten.
- Es gibt Tiere, von denen Sie nicht wissen, zu welcher Spezies sie gehören. Anhand der Werte für *Flauschigkeit* und *Niedlichkeit* wollen Sie diese einer der vier Ihnen bekannten Spezies zuordnen. Für diese Zuordnung wollen Sie die räumliche Nähe zu bereits klassifizierten Datenpunkten nutzen.



8.2 Das Beispielprogramm Tiere erkennen

Das Programm »Tiere erkennen« finden Sie wie alle anderen hier besprochenen Beispielprogramme auf der Webseite zum Buch: *https://maschinennah.de/ki-buch*. Wenn Sie das Programm starten, sehen Sie die vorliegenden 20 Datenpunkte und deren Klassifizierung, eingetragen im Koordinatensystem. Die waagerechte Achse steht für *Niedlichkeit*, die senkrechte Achse für *Flauschigkeit*. Sie können an dieser grafischen Darstellung ihres Datenbestands Folgendes ablesen:

- Vogelspinnen sind flauschig, aber nicht besonders niedlich.
- Häschen zeichnen sich durch hohe Flauschigkeit und Niedlichkeit aus.

- Haie sind weder besonders flauschig noch allzu niedlich.
- Igel wiederum sind außerordentlich niedlich, eine ausgeprägte Flauschigkeit hingegen gehört nicht zu ihren hervorstechenden Eigenschaften.

Wir müssen hier den Einwand gelten lassen, dass in unserer Darstellung auch die Haie recht niedlich geraten sind. Zudem gibt es nachweislich Personen, die sogar Vogelspinnen süß finden.

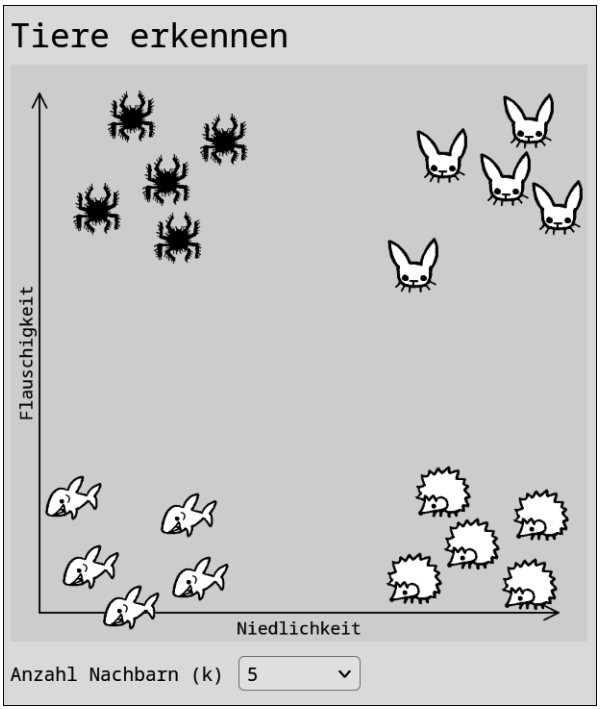


Abbildung 8.3 Das Beispielprogramm »Tiere erkennen«: Die 20 bekannten Datenpunkte sind hier abhängig von den Parametern »Niedlichkeit« und »Flauschigkeit« im Koordinatensystem angeordnet.

Mit dem Drop-down-Menü ANZAHL NACHBARN legen Sie fest, wie viele nächste Nachbarn zur Einordnung eines nicht klassifizierten Datenpunkts genutzt werden sollen.

Die Erzeugung eines neuen Datenpunkts ist ganz einfach: Klicken Sie mit der Maus auf eine beliebige Stelle im Koordinatensystem. Das Programm ordnet den neuen Datenpunkt sogleich einer Spezies zu. *Die Zuordnung passiert abhängig davon, welcher Spezies die meisten der benachbarten Datenpunkte angehören.* Abbildung 8.4, Abbildung 8.5 und Abbildung 8.6 zeigen Beispiele für solche Zuordnungen.

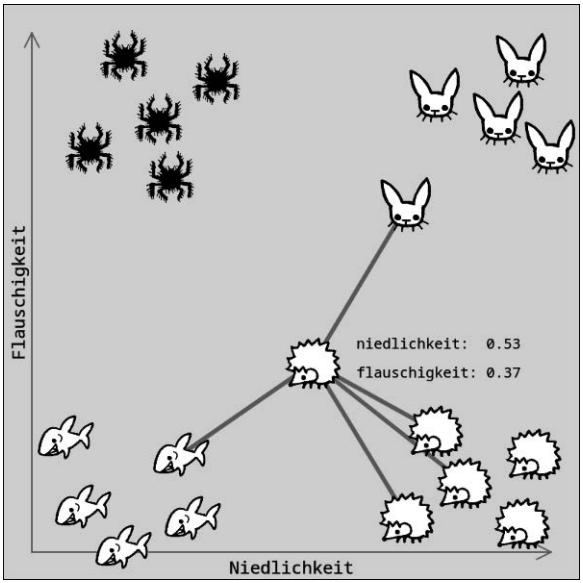


Abbildung 8.4 Das Programm ordnet unbekannte Datenpunkte den vier Spezies Vogelspinne, Häschen, Hai und Igel zu – hier wurde ein Igel erkannt, weil drei der fünf nächsten Nachbarn Igel sind.

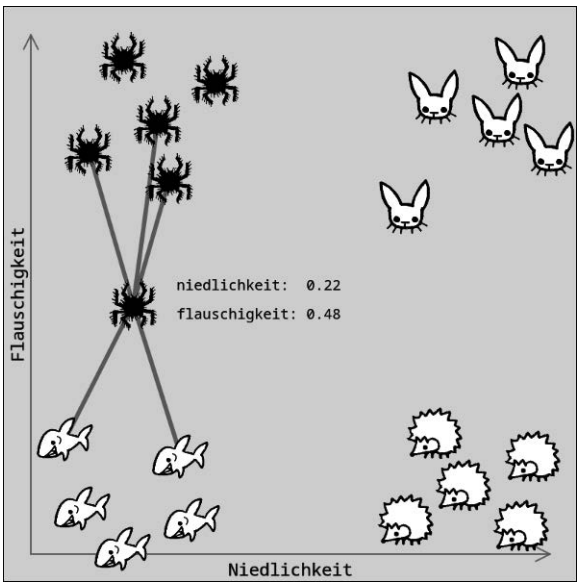


Abbildung 8.5 In diesem Fall hat das Programm eine Vogelspinne identifiziert, weil drei der fünf nächsten Nachbarn zu dieser Spezies gehören.

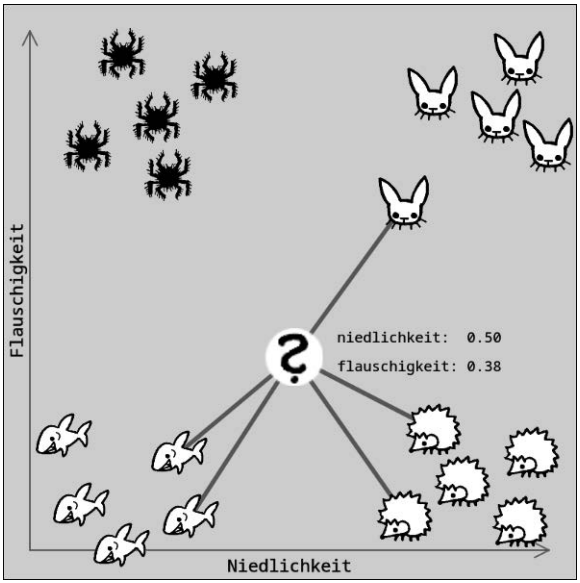


Abbildung 8.6 Weil es keinen eindeutigen »Sieger« gibt, hält sich das Programm mit einer Beurteilung zurück.

8.3 Entfernungen bestimmen mit Pythagoras

Jetzt kennen Sie das Prinzip des k-nächste-Nachbarn-Algorithmus. Es bleibt zu klären, *wie* wir die Entfernung zwischen zwei Datenpunkten bestimmen. Hier hilft uns ein alter Bekannter aus dem Matheunterricht weiter – der *Satz des Pythagoras*. Diese Gleichung macht eine einfache und nützliche Aussage über die Längenverhältnisse der Seiten eines rechtwinkligen Dreiecks. Dabei ist c die längste Seite, a und b die beiden kürzeren Seiten:

$$a^2 + b^2 = c^2$$

Abbildung 8.7 zeigt, wie Sie durch eine einfache Umformung dieser Gleichung die Länge von c berechnen können, wenn Ihnen die Längen von a und b bekannt sind.

In der Abbildung haben wir das Dreieck abweichend von der üblichen Darstellung des Pythagoras-Satzes so gedreht, *dass der rechte Winkel des Dreiecks unten links steht*. Damit wollen wir deutlich machen, dass wir die Formel zur Berechnung der Länge von c auch

nutzen können, um die Entfernung zwischen zwei Punkten P1 und P2 im Koordinatensystem zu ermitteln: Wir benennen die Seiten *a*, *b* und *c* schlichtweg um:

- Aus *a* wird die Distanz der beiden Punkte auf der x-Achse: Δx.
- Aus *b* wird die Distanz der beiden Punkte auf der y-Achse: Δy.
- Aus *c* wird die Distanz zwischen den beiden Punkten P1 und P2.

Das Zeichen Δ ist der griechische Buchstabe Delta, der in Formeln und Gleichungen häufig für Differenzen steht.

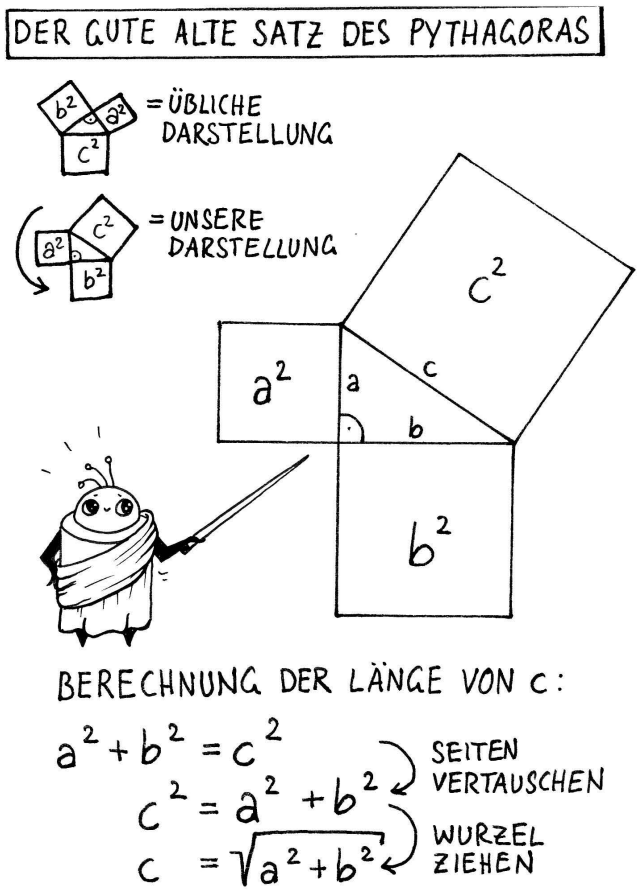


Abbildung 8.7 Mit dem Satz des Pythagoras können wir die Länge der Seite c berechnen ...

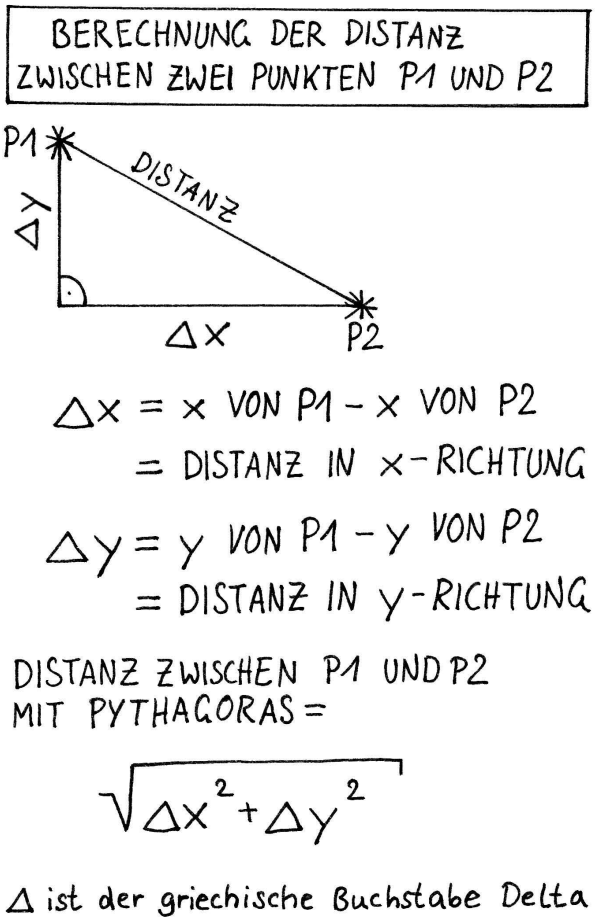


Abbildung 8.8 ... und dies für die Ermittlung der Distanz zwischen zwei Punkten nutzen.

Hintergrund: Euklidische und nicht-euklidische Geometrien

Die hier beschriebene, mit dem Satz des Pythagoras berechnete Entfernung zwischen zwei Punkten wird auch *euklidische Distanz* genannt. Dieser Name bezieht sich auf die vom griechischen Mathematiker Euklid begründete *euklidische Geometrie*. Sie beschreibt einen Raum, in dem Aussagen gelten wie: »Die Summe der Winkel in einem Dreieck beträgt immer 180 Grad« oder »Zwei parallele Linien berühren sich in keinem Punkt«.

Ab dem 19. Jahrhundert haben Mathematiker wie Felix Klein und Henri Poincaré begonnen, alternative, also *nicht-euklidische Geometrien* zu untersuchen. Ein einfaches Beispiel ist die Geometrie auf der Kugeloberfläche. Hier haben die eben genannten Aussagen zur Winkelsumme im Dreieck und den Parallelen keine Gültigkeit.

Jenseits solcher anschaulichen Beispiele untersuchen Mathematikerinnen und Mathematiker bis heute, unabhängig von jeder Anschauung oder Erfahrung, *wie überhaupt Räume beschaffen* und *Dinge zueinander angeordnet sein können*. Gibt es unendlich viele Möglichkeiten, unendlich viele Geometrien und Topologien? Lassen sich diese sinnvoll in Gruppen unterteilen?

Das sind höchst spannende Fragen, die sehr viel mehr mit den Thema dieses Buches zu tun haben, als auf den ersten Blick offensichtlich sein mag. Wir sprechen im Kontext von KI nämlich unentwegt über Räume, Dimensionen, Anordnungen von Objekten, Zustandsübergänge, Nachbarschaften, Entfernungen und deren Berechnung.

8.4 Der Code im Detail

Im Code unseres Beispielprogramms sind Objekte der Klasse `KNN` für die Umsetzung des **k**-**n**ächste-**N**achbarn-Algorithmus zuständig. Sie besitzt eine Eigenschaft und vier Funktionen, die in Tabelle 8.1 aufgelistet sind.

Name	Aufgabe

`datenpunkte`	Ein Array mit den bereits klassifizierten Datenpunkten in diesem Format:
	`{niedlichkeit: 0.90,`
	`flauschigkeit: 0.90,`
	`spezies: "haeschen"}`

`klassifiziere(datenpunktA, k)`	Klassifiziert einen Datenpunkt und liefert diesen zurück. `k` bestimmt die Anzahl der Nachbarn, die zur Klassifizierung genutzt werden. Ruft die Funktionen `nachbarn()` und `zaehlung()` auf.
`nachbarn(datenpunktA, k)`	Liefert ein Array mit den `k` nächsten Nachbarn von `datenpunktA`. Ruft die Funktion `distanz()` auf.

Tabelle 8.1 Eine Eigenschaft und vier Funktionen der Klasse KNN

Name	Aufgabe
`distanz(datenpunktA, datenpunktB)`	Ermittelt die Distanz zwischen `datenpunktA` und `datenpunktB`.

`zaehlung(nachbarn)`	Zählt die Häufigkeit der einzelnen Spezies in dem Datenpunkt-Array `nachbarn`.
	Rückgabewert ist eine Rangfolge im folgenden Format:
	`[["igel", 3],`
	`["hai", 1],`
	`["haeschen", 1]]`

Tabelle 8.1 Eine Eigenschaft und vier Funktionen der Klasse KNN (Forts.)

Listing 8.1 zeigt die Funktion zur Berechnung der Distanz zwischen zwei Datenpunkten, so wie wir sie in Abbildung 8.8 dargestellt haben. `deltaX` und `deltaY` sind die Distanzen zwischen den beiden Punkten in x- und y-Richtung. Die beiden Deltas werden anschließend quadriert und summiert. Die Quadratwurzel des Ergebnisses ist die Distanz:

```
distanz(datenpunktA, datenpunktB) {
  const deltaX = datenpunktA.niedlichkeit - datenpunktB.niedlichkeit;
  const deltaY = datenpunktA.flauschigkeit - datenpunktB.flauschigkeit;
  return sqrt(deltaX * deltaX + deltaY * deltaY);
}
```

Listing 8.1 Die Funktion knn.distanz() berechnet die Entfernung zwischen zwei Datenpunkten.

Die Funktion `nachbarn()` liefert die `k` nächsten Nachbarn zu einem gegebenen `datenpunktA` – im Fall unserer Anwendung ist `datenpunktA` ein noch nicht klassifizierter Punkt. Um die Nachbarn zu finden, sortiert `sort()` die `datenpunkte` abhängig von der Distanz zu `datenpunktA`. Anschließend wählt `slice()` die ersten `k` Datenpunkte aus:

```
nachbarn(datenpunktA, k) {
  this.datenpunkte.sort((L, R) =>
    this.distanz(datenpunktA, L) - this.distanz(datenpunktA, R));
  return datenpunkte.slice(0, k);
}
```

Listing 8.2 Die Funktion knn.nachbarn() liefert benachbarte Datenpunkte.

Jetzt fehlt noch die Auswertung des Rückgabewerts von `nachbarn()`: Sind die meisten der benachbarten Datenpunkte Vogelspinnen, Häschen, Haie oder Igel? Diese Auswertung liefert die Funktion `zaehlung()`.

Eine `for`-Schleife durchläuft alle im Argument übergebenen Datenpunkte und zieht jeweils die `spezies` heraus. Ist diese `spezies` im Objekt `zaehlung` noch nicht als Schlüssel vorhanden, wird der Schlüssel angelegt und mit dem Wert 1 initialisiert. Andernfalls wird der zu diesem Schlüssel eingetragene Wert um 1 erhöht. Dies ähnelt übrigens dem Vorgehen, das wir in Abschnitt 2.2, »Der Code des Nonsense-Texters unter der Lupe«, angewendet haben, als es um die Zählung von Worthäufigkeiten im Rahmen eines Markow-Prozesses ging. Abschließend wird das Objekt `zaehlung` per `Object.entries()` in ein Array umgewandelt. Die Funktion `sort()` sortiert das Array nach Anzahl. Das sortierte Array ist der Rückgabewert.

```
zaehlung(nachbarn) {
  const zaehlung = {};

  for (let datenpunkt of nachbarn) {
    const spezies = datenpunkt.spezies;
    if (!zaehlung[spezies]) {
      zaehlung[spezies] = 1;
    } else {
      zaehlung[spezies] += 1;
    }
  }

  const alsArray = Object.entries(zaehlung);
  alsArray.sort((L, R) => R[1] - L[1]);
  return alsArray;
}
```

Listing 8.3 Die Funktion knn.zaehlung() liefert eine Rangfolge der Spezies der in der Nachbarschaft gefundenen Datenpunkte.

In der Funktion `klassifiziere()` greifen alle vorgestellten Komponenten ineinander. Die Funktion erwartet als erstes Argument einen nicht klassifizierten `datenpunktA`. Das zweite Argument `k` bestimmt die Anzahl der Nachbarn, die für die Klassifikation relevant sind.

Die eben vorgestellten Funktionen liefern die entsprechenden nächsten Nachbarn und deren Auswertung nach Häufigkeit. Dann bleibt nur noch eine Sache zu tun, nämlich, die Auswertung in eine erkannte Spezies zu übersetzen – oder eben nicht:

- Die Bedingung `zaehlung.length == 1` ist genau dann erfüllt, wenn alle Nachbarn derselben Spezies angehören. In diesem Fall ist die Zuordnung eindeutig.
- Ein wenig komplizierter wird es, wenn die Auswertung mehrere Einträge enthält. Ist die Häufigkeit des ersten Eintrags größer als die des zweiten, dann ist die Zuordnung ebenfalls klar: Die Spezies des vorne stehenden Eintrags wurde identifiziert.
- Wenn keine dieser beiden Bedingungen erfüllt ist, gibt es keinen klaren Spitzenreiter: Die Spezies von `datenpunktA` erhält den Wert `"unbekannt"`.

```
klassifiziere(datenpunktA, k) {
  const nachbarn = this.nachbarn(datenpunktA, k);
  const zaehlung = this.zaehlung(nachbarn);

  if (zaehlung.length == 1 || zaehlung[0][1] > zaehlung[1][1]) {
    datenpunktA.spezies = zaehlung[0][0];
  } else {
    datenpunktA.spezies = "unbekannt";
  }
  return datenpunktA;
}
```

Listing 8.4 Die Funktion knn.klassifiziere() ordnet den unbekannten datenpunktA einer Spezies zu.

8.5 Ideen zum Weitermachen

Der k-nächste-Nachbarn-Algorithmus ist nicht besonders anspruchsvoll. Es sollte nicht schwer sein, unser Beispielprogramm zu modifizieren.

Sie könnten etwa eine weitere Tierart in das Programm aufnehmen. Hierfür sind lediglich zwei Ergänzungen notwendig:

- Zeichnen Sie das Tier und speichern Sie die Zeichnung im Ordner `daten` unter dem entsprechenden Namen. Ideal ist hier das PNG-Format mit einem Alpha-Kanal, also einem durchsichtigen Hintergrund.
- Schreiben Sie in die Datei *daten/datenpunkte.js* einige Datenpunkte, die Vertreter der neu hinzugefügten Tierart repräsentieren.

Alternativ könnten Sie Tiere auch nach anderen Eigenschaften ordnen: Wie wäre es, wenn sie Tiere nach Gewicht und Zutraulichkeit charakterisierten? Sie könnten auch ein ganz anderes Thema bearbeiten, etwa indem Sie Nahrungsmittel nach ihrem Fett- und

Zuckergehalt klassifizieren. Überlegen Sie dabei, welche Eigenschaften sich überhaupt sinnvoll auf einer Skala eintragen lassen.

8.6 Zusammenfassung und Ausblick

Der k-nächste-Nachbarn-Algorithmus ist ein einfaches Verfahren, um zahlenförmige Daten zu klassifizieren. Der Cartoon mit der falsch klassifizierten Katze zeigt die Grenzen auf: Es können nur Klassen erkannt werden, die schon vorher bekannt waren. Vielen ist die Erfahrung geläufig, von einem Computerprogramm oder einem formalen Prozess in ein Schema hineingepresst zu werden, das offensichtlich nicht passt.

Die in Abschnitt 8.3, »Entfernungen bestimmen mit Pythagoras«, vorgestellte Formel eignet sich ebenfalls für Daten mit mehr als zwei Dimensionen. Auch wenn wir uns das bildlich nur schwer vorstellen können: Wenn die Nährwertangaben eines Fertiggerichts jeweils Werte für Kohlenhydrate, Fett, Eiweiß, Ballaststoffe und Salz enthalten, können wir jedes Fertiggericht als Datenpunkt in einem fünfdimensionalen Raum behandeln und damit rechnen. Die Formel zur Berechnung der Entfernung wird nach demselben Prinzip aufgebaut. Wird die Anzahl der Dimensionen allerdings sehr groß, eignet sich dieses Berechnungsverfahren nicht mehr. Das liegt am sogenannten Fluch der Dimensionalität, den wir in Abschnitt 14.5.6 erklären. Dort stellen wir auch ein alternatives Verfahren zur Entfernungsmessung vor, die *Kosinus-Ähnlichkeit*.