Kursheft Datenanalyse und Modellierung

Teil 1: Daten verstehen, prüfen und mit SQL auswerten

Daten wirken auf den ersten Blick oft nüchtern: Tabellen, Zahlen, Spaltennamen. Trotzdem steckt darin fast immer eine Geschichte. Wenn du Daten auswertest, suchst du nicht einfach nur irgendwelche Werte heraus, sondern du versuchst, aus vielen Einzelfällen ein Muster zu erkennen. Genau darum geht es in diesem Teil. Du arbeitest mit DataBlick und SQL, prüfst Datensätze, formulierst Fragen an Daten und lernst, wie aus Tabellen eine begründete Aussage wird.

Am Anfang steht eine einfache, aber wichtige Einsicht: Eine Zahl allein bedeutet fast nie viel. Erst durch Vergleiche wird sie interessant. Wenn in einem Datensatz eine Pinguinart im Durchschnitt schwerer ist als eine andere, ist das eine Aussage. Wenn an bestimmten Uhrzeiten mehr Fahrräder ausgeliehen werden als an anderen, ist das ebenfalls eine Aussage. Datenanalyse heißt also: vergleichen, gruppieren, zusammenfassen und dann deuten.

Dabei reicht es nicht, nur auf ein Ergebnis zu schauen und sofort eine Erklärung zu erfinden. Genau hier passieren die meisten Fehler. Ein sichtbares Muster ist noch keine Ursache. Wenn zum Beispiel am Freitag in einer Mensa weniger verkauft wird, heißt das nicht automatisch, dass das Freitagsangebot schlecht ist. Vielleicht wurden weniger Freitage erfasst, vielleicht fehlen Daten, vielleicht gibt es an diesem Tag früher Schulschluss oder Exkursionen. Daten zeigen also oft zuerst nur: Hier ist etwas auffällig. Warum das so ist, muss man vorsichtig prüfen.

Ein guter Start in jede Analyse ist deshalb nicht die große Deutung, sondern die Prüfung des Datensatzes. Wie viele Zeilen gibt es? Welche Spalten sind vorhanden? Fehlen Werte? Sind Kategorien codiert? Sind Zahlen wirklich als Zahlen gespeichert? Das klingt unspektakulär, ist aber entscheidend. Wenn du mit falschen Datentypen arbeitest, mit übersehenen Lücken oder missverstandenen Codes, wird auch die schönste Abfrage am Ende unsauber.

Ein Beispiel dafür ist der Pinguin-Datensatz palmerpenguins. Dort gibt es Arten, Inseln, Körpermaße, Geschlecht und Jahr. Auf den ersten Blick ist das übersichtlich. Spannend wird es aber bei den fehlenden Werten. In Zahlenspalten wie `bodymassg` wird ein fehlender Wert beim korrekten Import zu `NULL`. In Textspalten wie `sex` kann stattdessen der Text `NA` stehen. Beides sieht nach „fehlt“ aus, ist aber technisch nicht dasselbe. Wenn du das nicht beachtest, rechnest du leicht falsch oder findest Lücken nicht dort, wo sie tatsächlich sind.

Ein zweiter wichtiger Datensatz ist Bike Sharing. Hier kommen Zeitangaben, Wetterdaten und Ausleihzahlen zusammen. Das wirkt schon deutlich realistischer und unübersichtlicher als bei den Pinguinen. Genau deshalb ist der Datensatz gut: Er zeigt, dass Daten fast nie fertig erklärt vor dir liegen. Manche Spalten sind codiert, etwa `season` oder `workingday`. Diese Zahlen sind keine Messwerte, sondern Kategorien mit fester Bedeutung. `season = 4` bedeutet nicht „mehr Saison“ als `season = 1`, sondern einfach eine andere Jahreszeit. Wer so etwas falsch liest, deutet Unsinn in die Daten hinein.

SQL ist in diesem Teil dein Werkzeug, um Fragen präzise zu stellen. Statt eine Tabelle nur anzustarren, kannst du sagen: Zeig mir die durchschnittliche Körpermasse pro Art. Oder: Berechne die durchschnittlichen Ausleihen pro Stunde. Oder: Vergleiche Arbeitstage und Nicht-Arbeitstage. SQL hilft dir also, aus einer allgemeinen Frage eine genaue Abfrage zu machen. Besonders oft brauchst du dabei `SELECT`, `WHERE`, `GROUP BY`, `COUNT`, `AVG`, `SUM` und `ORDER BY`.

Wichtig ist aber: Eine SQL-Abfrage ist nicht das Endprodukt. Sie ist nur der Schritt, mit dem du dein Ergebnis berechnest. Die eigentliche Leistung besteht darin, das Ergebnis sinnvoll zu lesen. Wenn du am Ende nur schreibst „Gentoo 5076“, fehlt das Entscheidende. Erst ein vollständiger Satz macht aus einer Zahl eine Analyse, zum Beispiel: Im Datensatz haben Gentoo-Pinguine die höchste durchschnittliche Körpermasse. Diese Aussage stützt sich auf den Mittelwert der Spalte `bodymassg`. Daraus folgt aber nicht, dass jeder einzelne Gentoo schwerer ist als jeder Adelie oder Chinstrap.

Genau diese vorsichtige Sprache ist in der Datenanalyse wichtig. Du sollst nicht kleinlaut sein, aber sauber. Eine gute Aussage hat meist drei Teile: Erstens den Vergleich, zweitens die Datengrundlage und drittens eine Grenze der Aussage. Also nicht nur was auffällt, sondern auch, woraus du das ableitest und was du trotzdem nicht sicher behaupten kannst.

Diagramme helfen dir dabei, Ergebnisse schneller zu sehen. Ein Balkendiagramm eignet sich gut für Gruppenvergleiche, ein Liniendiagramm für Zeitverläufe und ein Punktdiagramm für Zusammenhänge zwischen zwei Zahlenwerten. Aber auch hier gilt: Das Diagramm ersetzt die Analyse nicht. Es zeigt ein Ergebnis sichtbar an, doch du musst immer noch erklären, was man daraus wirklich sagen kann und was nicht. Besonders wichtig ist dabei eine faire Achse. Eine abgeschnittene y-Achse kann Unterschiede dramatischer wirken lassen, als sie eigentlich sind.

In diesem Teil lernst du also vor allem, Daten nicht blind zu konsumieren. Du prüfst, ob der Datensatz brauchbar ist. Du formulierst eine sinnvolle Frage. Du schreibst eine passende SQL-Abfrage. Du liest das Ergebnis nicht nur ab, sondern deutest es. Und du begrenzt deine Aussage so, dass sie fachlich sauber bleibt. Das klingt vielleicht zunächst weniger spektakulär als KI, ist aber die Grundlage für alles Weitere. Ohne saubere Datenanalyse gibt es später auch keine saubere Modellierung.

Ein typischer Denkfehler wäre zum Beispiel, Durchschnitt und Summe zu verwechseln. Wenn im Bike-Sharing-Datensatz der Durchschnitt pro Stunde berechnet wurde, darfst du das Ergebnis nicht wie eine Monatssumme behandeln. Ein anderer häufiger Fehler ist, codierte Kategorien wie echte Zahlen zu lesen oder aus einem Zusammenhang direkt auf eine Ursache zu schließen. Genau deshalb ist Teil 1 so wichtig: Er trainiert nicht nur Technik, sondern vor allem sauberes Denken mit Daten.

Wenn du aus diesem Teil eine Grundhaltung mitnehmen sollst, dann diese: Prüfen vor Rechnen, vergleichen statt raten und Ergebnisse immer vorsichtig formulieren. Wer das beherrscht, kann mit Daten deutlich mehr anfangen als jemand, der nur schnell eine Zahl ausspuckt.

Teil 2: Dieselbe Analyse mit Python weiterführen

Im zweiten Teil kommt Python dazu. Das wirkt zuerst wie ein großer Sprung, ist aber eigentlich eher eine Erweiterung als ein Neustart. Du lässt bekannte SQL-Abfragen nicht mehr nur in DataBlick laufen, sondern führst sie auch in Python aus. Dadurch wird aus einer einzelnen Abfrage ein wiederholbarer Ablauf: Daten laden, Ergebnis prüfen, weiterverarbeiten, darstellen und speichern.

Die Brücke zwischen beidem baut DataBlick selbst. Zu einer SQL-Abfrage kann DataBlick im Python-Tab passenden Code erzeugen. Das ist für den Einstieg hilfreich, weil du nicht alles von Hand neu schreiben musst. Du erkennst im Python-Code dieselbe Abfrage wieder, die du vorher schon in DataBlick getestet hast. Damit ist sofort klar: Python ersetzt SQL hier nicht einfach, sondern nutzt die bekannte Abfrage als Ausgangspunkt.

Der erste wichtige Gedanke in diesem Teil ist der Datenfluss. Du solltest verstehen, woher die Daten kommen, welche Datenbank geöffnet wird, welche SQL-Abfrage ausgeführt wird und wo das Ergebnis landet. Du musst nicht jede technische Hilfszeile perfekt auswendig können. Entscheidend ist, dass du den Ablauf erklären kannst: Python öffnet die Datenbank, übergibt die SQL-Abfrage an SQLite, lädt das Ergebnis und verarbeitet es weiter.

Am Anfang tauchen dabei mehrere Grundlagen von Python auf: Variablen, Datentypen, Listen, Dictionaries, Schleifen, Bedingungen und Dateipfade. Diese Dinge werden hier nicht als trockene Theorie behandelt, sondern direkt im Zusammenhang mit der Datenanalyse. Eine Variable ist dann nicht nur irgendein Beispiel wie `x = 5`, sondern etwa der Pfad zur Datenbank oder die SQL-Abfrage als Text. Eine Schleife ist nicht bloß eine Wiederholung, sondern das Durchgehen von Ergebniszeilen. Dadurch wird klarer, wozu Programmieren in diesem Kurs überhaupt gebraucht wird.

Sehr wichtig ist in diesem Teil auch der Umgang mit Fehlermeldungen. Wenn Python mit `NameError`, `IndentationError`, `FileNotFoundError` oder `no such table` reagiert, heißt das nicht automatisch, dass du zu schlecht im Programmieren bist. Meist steckt ein konkreter, lösbarer Fehler dahinter: ein falsch geschriebener Name, eine kaputte Einrückung, ein falscher Dateipfad oder eine falsche Tabelle. Wer Fehlermeldungen lesen kann, verliert schnell die Angst vor dem Programmieren.

Sobald das SQL-Ergebnis in Python geladen ist, arbeitest du meist mit einem DataFrame aus pandas. Ein DataFrame ist im Grunde eine Tabelle in Python. Diese Tabelle kannst du anschauen, prüfen, sortieren, filtern und erweitern. Genau wie in Teil 1 gilt auch hier: Erst prüfen, dann deuten. Ein Diagramm wirkt schnell überzeugend, aber wenn die geladenen Daten nicht stimmen, nützt dir die schönste Darstellung nichts.

Deshalb gehören Prüfungen wie `df.head()`, `df.info()`, Datentypen und Fehlwertkontrollen zu den wichtigsten Routinen. Du schaust also nicht nur, dass Daten geladen wurden, sondern auch wie sie angekommen sind. Sind die erwarteten Spalten vorhanden? Sind Zahlen wirklich Zahlen? Gibt es fehlende Werte? Ist die Anzahl der Zeilen plausibel? Diese Fragen machen den Unterschied zwischen einer oberflächlichen und einer sauberen Analyse aus.

Ein großer Vorteil von Python ist die Weiterverarbeitung. Du kannst Spalten auswählen, fehlende Werte entfernen, nach Werten sortieren, neue Spalten berechnen und Ergebnisse als CSV speichern. Das klingt technisch, ist aber extrem nützlich. Spätestens in den späteren Modellierungsprojekten brauchst du genau diese Schritte, um aus einem Datensatz gezielt die Merkmale und die Zielvariable herauszulösen. Teil 2 ist also nicht nur etwas Python, sondern die Vorbereitung auf alles, was danach kommt.

Besonders sichtbar wird der Mehrwert von Python bei Diagrammen. Mit pandas und matplotlib kannst du Balken-, Linien- oder Punktdiagramme erzeugen und deutlich freier anpassen als in einem einfachen Datenbankwerkzeug. Du kannst Achsen beschriften, Raster einblenden, Marker setzen, mehrere Datenreihen vergleichen und Diagramme als Datei exportieren. Damit wird die Darstellung nicht nur schöner, sondern vor allem passender zur jeweiligen Frage.

Ein Beispiel ist wieder der Pinguin-Datensatz. Wenn du die durchschnittliche Körpermasse pro Art berechnet hast, kannst du das Ergebnis als Balkendiagramm darstellen. Das Diagramm macht den Vergleich sofort sichtbar. Beim Bike-Sharing-

Datensatz eignet sich dagegen oft ein Liniendiagramm, weil Zeitverläufe so besser erkennbar sind. Besonders deutlich wird das bei den durchschnittlichen Ausleihen pro Stunde: Hier sieht man typische Spitzenzeiten viel schneller in einer Linie als in einer bloßen Tabelle.

Python zeigt seinen echten Mehrwert aber vor allem dann, wenn die Darstellung etwas komplexer wird. Wenn du zum Beispiel Arbeitstage und Nicht-Arbeitstage im selben Diagramm vergleichen willst, musst du das Ergebnis zunächst mit `pivot` umformen. Dabei ändern sich nicht die Werte, sondern nur die Form der Tabelle. Aus einer Gruppenspalte werden mehrere Spalten, sodass zwei Linien in einem Diagramm möglich werden. Das ist ein guter Moment, um zu merken: Datenanalyse ist nicht nur Rechnen, sondern oft auch geschicktes Umformen von Daten.

Auch in Teil 2 bleibt die Deutung am Ende entscheidend. Ein Diagramm ist kein Selbstzweck. Aus einem Balken oder einer Linie wird erst dann eine Analyse, wenn du daraus einen verständlichen Satz machst. Wie schon in Teil 1 sollte eine gute Aussage Vergleich, Datengrundlage und Vorsichtshinweis verbinden. Du beschreibst also nicht nur, was sichtbar ist, sondern auch, worauf sich diese Aussage stützt und was man daraus trotzdem nicht sicher folgern darf.

Außerdem taucht in diesem Teil zum ersten Mal ausdrücklich auf, dass KI als Programmierhilfe genutzt werden darf. Das ist sinnvoll, gerade bei Fehlern oder kleineren Anpassungen. Wichtig bleibt aber: Die fachliche Kontrolle liegt bei dir. Wenn du dir Code erklären lässt oder eine Änderung vorschlagen lässt, musst du am Ende trotzdem sagen können, welche Daten geladen werden, welche Spalten verwendet werden und warum das Ergebnis zur Frage passt.

Wenn du Teil 2 auf einen Kern bringen willst, dann so: Du nimmst eine bekannte SQL-Analyse, bringst sie nach Python, lädst das Ergebnis als DataFrame, prüfst es, verarbeitest es weiter und stellst es mit einem passenden Diagramm dar. Dadurch wird aus einer einmaligen Abfrage ein nachvollziehbarer, wiederholbarer Analyseablauf. Genau dieser Übergang ist später wichtig, wenn aus Analyse Modellierung wird.

Teil 3: Von der Beschreibung zur Vorhersage

Bisher hast du Daten vor allem beschrieben. Du hast untersucht, welche Gruppen sich unterscheiden, wann bestimmte Werte hoch oder niedrig sind oder welche Muster in einem Datensatz auffallen. Im dritten Teil ändert sich die Blickrichtung. Jetzt geht es nicht mehr nur darum, vorhandene Daten zu verstehen, sondern darum, aus bekannten Daten eine Vorhersage für neue Fälle abzuleiten.

Der Unterschied zwischen Analyse und Prognose ist wichtiger, als er zunächst klingt. Eine Analyse sagt zum Beispiel: Im Datensatz sind Gentoo-Pinguine im Durchschnitt schwerer als Adelie-Pinguine. Eine Prognose sagt: Dieser neu gemessene Pinguin gehört wahrscheinlich zur Art Gentoo. Bei der Analyse blickst du zurück oder auf vorhandene Daten. Bei der Prognose versuchst du, aus den gelernten Mustern etwas über einen neuen, bisher unbekannten Fall zu sagen.

Dafür brauchst du zwei Grundbegriffe: Merkmale und Zielvariable. Merkmale sind die Eingaben des Modells, also zum Beispiel Schnabellänge, Flossenlänge, Körpermasse, Uhrzeit, Temperatur oder Wochentag. Die Zielvariable ist das, was vorhergesagt werden soll, etwa die Pinguinart oder eine Nachfrageklasse. Ein Modell lernt also vereinfacht gesagt: Aus diesen Merkmalen soll diese Zielvariable vorhergesagt werden.

Genau hier lauert aber eine der wichtigsten Fallen überhaupt: Datenleckage. Ein Modell darf nur Informationen verwenden, die zum Vorhersagezeitpunkt wirklich bekannt wären. Wenn du dem Modell eine Spalte gibst, die die Lösung schon direkt verrät oder erst

nachträglich entsteht, ist das Ergebnis wertlos. Beim Bike Sharing wäre es zum Beispiel unsinnig, `cnt` vorherzusagen und gleichzeitig `casual` und `registered` als Merkmale zu verwenden, weil `cnt` aus diesen Werten zusammengesetzt ist. Dann würde das Modell nicht wirklich lernen, sondern nur einen versteckten Spickzettel ausnutzen.

Außerdem musst du unterscheiden, welche Art von Vorhersage du überhaupt machen willst. Bei einer Klassifikation sagt das Modell eine Kategorie voraus, etwa Adelie, Chinstrap oder Gentoo, oder niedrig, mittel und hoch. Bei einer Regression sagt es einen Zahlenwert voraus, etwa die Anzahl ausgeliehener Fahrräder oder einen Preis. Derselbe Datensatz kann je nach Frage beides hergeben. Aus den Pinguindaten kannst du die Art vorhersagen, aber auch die Körpermasse schätzen. Nicht der Datensatz allein entscheidet also über die Modellart, sondern deine Fragestellung.

Damit ein Modell sinnvoll geprüft werden kann, müssen die Daten getrennt werden. Genau das ist eine der wichtigsten Ideen in Teil 3. Wenn du ein Modell mit allen Daten trainierst und anschließend mit denselben Daten bewertest, ist die Prüfung unfair. Das Modell könnte die bekannten Beispiele sehr gut auswendig gelernt haben, ohne bei neuen Fällen wirklich gut zu sein. Deshalb teilt man den Datensatz in Trainingsdaten und Testdaten, oft ungefähr im Verhältnis 80 zu 20 oder 75 zu 25.

Die Trainingsdaten sind zum Lernen da. Mit ihnen sucht das Modell nach Mustern. Die Testdaten bleiben zunächst unberührt und werden erst ganz am Ende benutzt. Sie sind so etwas wie die Klassenarbeit nach dem Üben. Wenn du beim Modellieren immer wieder auf die Testdaten schaust, um Einstellungen zu verbessern, werden auch diese Daten schleichend zu Übungsmaterial. Dann ist die Abschlussprüfung nicht mehr ehrlich.

Um Modelle fair zu beurteilen, reicht eine einzige Trefferquote außerdem nicht aus. Stell dir vor, eine Klasse kommt viel häufiger vor als alle anderen. Dann kann ein Modell schon recht gut aussehen, wenn es einfach immer die häufigste Klasse rät. Genau deshalb brauchst du die Baseline. Die Baseline ist die einfachste Vergleichsregel: Sage immer die häufigste Klasse voraus. Erst wenn dein Modell deutlich besser ist als diese dumme Regel, ist das Ergebnis wirklich überzeugend.

Neben der Trefferquote ist auch wichtig, welche Klassen verwechselt werden. Dafür gibt es die Verwechslungsmatrix, oft Confusion Matrix genannt. Sie zeigt nicht nur, wie viele Vorhersagen richtig waren, sondern auch, wo typische Fehler auftreten. Ein Modell kann insgesamt ordentlich wirken und trotzdem bei einer seltenen Klasse komplett versagen. Wer nur auf die Gesamtzahl schaut, übersieht solche Probleme schnell.

Ein weiterer zentraler Begriff in diesem Teil ist Overfitting. Damit ist gemeint, dass ein Modell sich zu stark an die Trainingsdaten anpasst. Es lernt dann nicht nur echte Muster, sondern auch Zufälligkeiten und Sonderfälle. Das Ergebnis sieht auf den Trainingsdaten großartig aus, bricht aber auf den Testdaten ein. Ein klassisches Warnzeichen ist deshalb: sehr gute Werte im Training, deutlich schlechtere Werte im Test.

Gerade bei Entscheidungsbäumen ist Overfitting anschaulich. Ein sehr tiefer Baum kann immer speziellere Regeln bilden, bis fast jeder Einzelfall seine eigene Sonderregel bekommt. Das wirkt auf bekannte Daten beeindruckend, ist aber für neue Fälle unzuverlässig. Deshalb ist nicht der Baum am besten, der auf dem Training am höchsten punktet, sondern der, der auf neuen Daten brauchbar bleibt.

Teil 3 stellt außerdem die wichtigsten Modellideen des Kurses vor. Ein Entscheidungsbaum trifft nacheinander Regeln und ist gut erklärbar. Ein Random Forest kombiniert viele Entscheidungsbäume und ist oft stabiler, aber weniger durchschaubar. Ein kleines neuronales Netz ist flexibler und anspruchsvoller, braucht aber meist mehr Vorbereitung und ist schwerer zu erklären. Für tabellarische Daten wie im Kurs sind Entscheidungsbaum und Random Forest oft der naheliegende Einstieg.

Ganz am Ende wird noch etwas deutlich, das man leicht vergisst: Nicht jede Vorhersage, die technisch möglich ist, ist auch sinnvoll oder fair. Eine hohe Trefferquote reicht nicht automatisch als Rechtfertigung. Man muss auch fragen, ob die Zielvariable sinnvoll ist, ob die Merkmale angemessen sind, ob Gruppen benachteiligt werden könnten und ob man die Entscheidung einer betroffenen Person überhaupt verständlich erklären könnte. Modellierung ist also nicht nur Technik, sondern auch Verantwortung.

Wenn du Teil 3 in einem Satz zusammenfassen willst, dann so: Aus Daten wird noch kein gutes Modell, nur weil ein Algorithmus darüberläuft. Zuerst musst du eine sinnvolle Prognosefrage formulieren, die richtigen Merkmale auswählen, Datenleckage vermeiden, Training und Test sauber trennen, die Baseline berücksichtigen und auf Overfitting achten. Erst dann ist der Weg frei für ein wirkliches Modellprojekt.

Teil 4: Ein eigenes Modell bauen, prüfen und begründen

Im vierten Teil wird aus der ganzen Vorbereitung ein eigenes Projekt. Jetzt reicht es nicht mehr, nur über Zielvariable, Merkmale oder Overfitting zu sprechen. Du setzt ein Modell tatsächlich um, prüfst seine Qualität und dokumentierst deine Entscheidungen. Im Mittelpunkt steht dabei meist ein Entscheidungsbaum, oft ergänzt durch einen Random Forest als Vergleich.

Der Startpunkt ist immer eine Prognosefrage. Sie muss klar genug sein, dass man daraus eine Zielvariable und passende Merkmale ableiten kann. Eine gute Frage wäre zum Beispiel: Zu welcher Art gehört ein neu vermessener Pinguin? Oder: Ist die Fahrradnachfrage eher niedrig, mittel, hoch oder sehr hoch? Eine schlechte Projektfrage wäre dagegen bloß: Was kann man mit dem Datensatz machen? Solche Fragen sind zu offen und führen fast immer zu unklaren Projekten.

Wenn die Frage steht, legst du die Zielvariable und die Merkmale fest. Dabei musst du begründen können, warum genau diese Spalten verwendet werden. Gleichzeitig musst du ausschließen, was unzulässig ist. Eine Zielspalte darf nie gleichzeitig als Merkmal im Modell auftauchen. Auch Spalten, die die Lösung indirekt verraten oder erst nach dem Ereignis bekannt werden, sind tabu. Diese Trennung ist keine Formalität, sondern entscheidet darüber, ob dein Projekt fachlich überhaupt ernst zu nehmen ist.

Danach bereitest du die Daten vor. Fehlende Werte müssen geprüft und behandelt werden. Wenn du `dropna()` benutzt und dadurch Zeilen entfernst, solltest du wissen, wie viele das sind und ob eine Klasse besonders stark betroffen ist. Gerade bei kleinen oder ungleich verteilten Datensätzen kann das die Aussagekraft des Modells verändern. Auch kategoriale Merkmale müssen gegebenenfalls in Zahlen umgewandelt werden, denn viele Modelle in scikit-learn arbeiten nur mit numerischen Eingaben.

Dann folgt der Train-Test-Split. Das Modell lernt nur auf den Trainingsdaten. Getestet wird es auf den Testdaten. In vielen Beispielen wird zusätzlich `stratify=y` verwendet, damit die Klassenverteilung in beiden Teilmengen ungefähr erhalten bleibt. Das ist besonders wichtig, wenn einzelne Klassen seltener vorkommen. Ein Modell, das eine seltene Klasse im Testdatensatz kaum oder gar nicht mehr vorfindet, lässt sich schlechter bewerten.

Das Grundmuster in scikit-learn ist dabei erstaunlich klar: Modell erstellen, mit `fit()` trainieren, mit `predict()` vorhersagen. Dahinter steckt trotzdem viel fachliche Entscheidung. Denn auch wenn der eigentliche Trainingsaufruf nur eine Zeile ist, hängt die Qualität des Ergebnisses davon ab, wie sauber die Daten vorbereitet und wie sinnvoll die Merkmale ausgewählt wurden.

Ein Entscheidungsbaum ist für den Einstieg besonders geeignet, weil seine Regeln sichtbar gemacht oder als Text ausgegeben werden können. Du kannst also nachvollziehen, welche Fragen das Modell nacheinander stellt. Das macht den Baum gut erklärbar. Gleichzeitig ist genau das auch seine Schwäche: Ein einzelner Baum kann leicht overfitten und auf kleine Änderungen im Datensatz empfindlich reagieren. Deshalb gehört zum Pflichtteil in diesem Projekt meist auch der Vergleich mehrerer Baumtiefen.

Genau an diesem Punkt wird die Baseline wieder wichtig. Eine Trefferquote allein klingt schnell beeindruckend, ist aber ohne Vergleich wenig wert. Wenn deine Baseline auf den Testdaten zum Beispiel schon 45 Prozent erreicht und dein Modell nur 50 Prozent, dann ist das zwar technisch besser, aber kaum überzeugend. Erst wenn dein Modell klar über der Baseline liegt und dabei nicht nur im Training glänzt, wird es wirklich interessant.

Zur Bewertung gehören deshalb mindestens die Trefferquote, oft auch ein classification report und eine Confusion Matrix. Die Trefferquote sagt, wie viel insgesamt richtig war. Die Confusion Matrix zeigt, welche Klassen verwechselt werden. Der classification report ergänzt Kennzahlen pro Klasse. Diese Auswertungen sind keine Deko, sondern die Grundlage dafür, dein Modell ehrlich zu beschreiben. Du sollst also nicht nur schreiben Accuracy 0,81, sondern erklären, welche Klassen gut erkannt werden und wo das Modell Schwächen hat.

Ein zusätzlicher Schritt ist der Vergleich mit einem Random Forest. Ein Random Forest besteht aus vielen Entscheidungsbäumen, die gemeinsam abstimmen. Oft ist er stabiler und auf Testdaten besser als ein einzelner Baum. Dafür verliert man aber einen Teil der einfachen Erklärbarkeit. Genau das macht den Vergleich spannend: Nicht immer lohnt sich ein komplizierteres Modell automatisch. Du sollst also nicht bloß die höhere Zahl feiern, sondern auch fragen, ob der zusätzliche Aufwand und die geringere Transparenz den Gewinn rechtfertigen.

Außerdem kannst du bei Baumverfahren die Feature Importance ausgeben. Sie zeigt, welche Merkmale für das Modell wichtig waren. Aber auch hier gilt dieselbe Vorsicht wie schon in Teil 1: Wichtigkeit ist keine Ursache. Wenn ein Merkmal im Modell stark gewichtet wird, heißt das nur, dass es für die Vorhersage nützlich war. Es heißt nicht, dass dieses Merkmal die Zielvariable verursacht.

Ein wesentlicher Teil des Projekts ist die Dokumentation. Dazu gehört ein Prozesstagebuch, in dem du festhältst, was du wann gemacht hast, welche Probleme aufgetreten sind, welche Entscheidungen du getroffen hast und wie du Ergebnisse geprüft hast. Das klingt erst einmal nach Zusatzarbeit, ist aber in Wirklichkeit ein Schutz vor Chaos. Gerade bei längeren Projekten vergisst man sonst schnell, warum man eine bestimmte Entscheidung getroffen oder eine bestimmte Version des Codes benutzt hat.

Ebenso verpflichtend ist die Dokumentation der KI-Nutzung. Wenn du KI zur Fehlersuche, zum Verstehen von Code, zur Verbesserung von Formulierungen oder als Ausgangspunkt für GUI-Code genutzt hast, musst du das präzise festhalten. Entscheidend ist nicht, ob KI benutzt wurde, sondern ob du am Ende trotzdem erklären kannst, was dein Modell tut, wie es geprüft wurde und wo seine Grenzen liegen.

Eine kleine GUI oder ein gespeichertes Modell können am Ende ein schöner Bonus sein. Beides kommt aber erst nach dem Pflichtteil. Ein gespeichertes Modell ist nur dann sinnvoll, wenn vorher Training, Test, Baseline, Modellgüte und Deutung sauber erledigt wurden. Eine GUI macht ein schwaches Modell nicht besser, sondern höchstens benutzbarer.

Insgesamt ist Teil 4 der Punkt, an dem aus Unterrichtsstoff ein eigenes kleines Datenprojekt wird. Du formulierst eine Prognosefrage, wählst Daten und Merkmale aus, trainierst ein Modell, vergleichst es mit einer Baseline, prüfst Overfitting, deutest die

Ergebnisse ehrlich und dokumentierst deine Arbeit nachvollziehbar. Genau darin liegt der eigentliche Anspruch: nicht irgendein Modell laufen zu lassen, sondern zu zeigen, dass du verstehst, was du da tust.