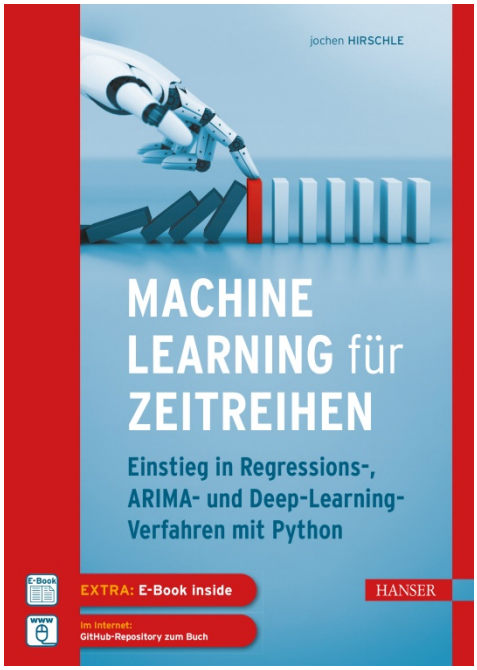
HANSER



Leseprobe

zu

Machine Learning für Zeitreihen

von Jochen Hirschle

Print-ISBN: 978-3-446-46726-2
E-Book-ISBN: 978-3-446-46814-6
E-Pub-ISBN: 978-3-446-46830-6

Weitere Informationen und Bestellungen unter
http://www.hanser-fachbuch.de/978-3-446-46726-2
sowie im Buchhandel

Inhalt

Vorwort

Zeitreihendaten entstehen heute in einer Vielzahl von Geschäftsfeldern – ob im Einzelhandel, in der Finanzbranche oder in der industriellen Produktion. Überall werden Messdaten aufgezeichnet und mit Zeitstempeln versehen abgelegt.

Solche Daten bilden nicht nur isolierte Zustände ab, sie sind wie Filme, die den Verlauf eines Vorgangs mit einer Serie von Momentaufnahmen nachzeichnen. Die zeitliche Dimension, die dadurch darstellbar wird, ist in einer datengetriebenen Ökonomie mehr als nur eine Spielart der Data Sciences. Der analytische Mehrwert von Zeitreihen wird bereits heute in einer Vielzahl von Geschäftsfeldern gewinnbringend eingesetzt und in einer noch größeren Zahl von Geschäftsfeldern wird das in der Zukunft geschehen.

Um diesen analytischen Mehrwert nutzen zu können, braucht man die richtigen Instrumente. Man braucht Konzepte und Techniken, muss verstehen, welche Muster in Zeitreihen typischerweise auftreten, welche Möglichkeiten es gibt, sie zu analysieren, und wie man sie zur Erzeugung von Prognosen einsetzen kann.

Genau das sind die Themen dieses Einführungsbuchs. Es bietet einen Einstieg in die Grundlagen und in die Praxis der Zeitreihenanalyse. Es zeigt anhand einer Vielzahl von Anwendungsbeispielen, wie sich Zeitreihen mit Python aufbereiten und analysieren lassen.

Dieses Buch ist aber kein Handbuch. Erwarten Sie also nicht, dass alle Verfahrensweisen, die es im Bereich der Zeitreihenanalyse gibt, besprochen werden. Es ist deshalb insbesondere für LeserInnen empfehlenswert, die neu in diesem Bereich sind und die sowohl eine Einführung in die wichtigsten Konzepte als auch in die zentralen Verfahren der Zeitreihenanalyse suchen. Wenn das Ihre Ziele sind und Sie darüber hinaus die nötigen Skills entwickeln möchten, um selbstständig mit Ihren eigenen Daten arbeiten zu können, bietet Ihnen dieses Buch einen soliden Einstieg.

Jetzt wünsche ich Ihnen eine anregende Lektüre. Ich hoffe, dass Sie währenddessen auch die Ehrfurcht vor der Materie verlieren, und dass das Buch Sie dazu inspiriert, eigene kreative Machine-Learning-Lösungen zu entwickeln!

Jochen Hirschle

Braunschweig im Oktober 2020

3 Grundprinzipien maschinellen Lernens

Maschinelles Lernen bezeichnet die Fähigkeit eines Algorithmus, selbstständig aus Daten zu lernen. In diesem Buch geht es vorwiegend um überwachte Lernverfahren (Supervised Learning). Dabei lernt ein Algorithmus eine Beziehung zwischen einer Anzahl von *x*-Variablen (den Merkmalen oder Features) und einer *y*-Variable (der Zielvariable oder Target).

Ziel ist es, den Algorithmus so zu trainieren, dass er für gegebene Werte auf den *x*-Variablen den Wert auf der Zielvariable vorhersagen kann. Zum Beispiel soll der Algorithmus auf Grundlage von Merkmalen wie Alter, Essgewohnheiten und Vorerkrankungen mögliche Folgeerkrankungen einer Person vorhersagen können.

Damit das funktioniert, wird der Algorithmus während des Trainings sowohl mit *x*- als auch mit y-Daten gefüttert. Wir müssen ihm also Gelegenheit geben, Erfahrungen zu sammeln. Dazu brauchen wir eine hinreichend große Menge von Daten, in denen solche Erfahrungen gespeichert sind. In den Daten müssen also für eine gewisse Anzahl von Untersuchungsobjekten (z. B. Personen) die x-Daten als auch die y-Daten enthalten sein.

Wie umfangreich diese Daten sein sollten, ist schwer zu sagen. Je mehr, desto besser. Allerdings brauchen wir keine vollständigen Daten, keine Grundgesamtheit, um einen Algorithmus zu trainieren. Es müssen ja nicht schon alle möglichen Erfahrungen in allen möglichen Variationen gemacht und gesammelt worden sein, bevor man etwas lernen kann. Es reicht ein Auszug, eine Stichprobe möglicher Erfahrungen, die hinreichend repräsentativ für die Grundgesamtheit ist. Schwierig wird es nur dann, wenn die Daten verzerrt sind. Zum Beispiel, wenn nur Untersuchungsobjekte einer bestimmten Art enthalten sind – nur Personen mit Diabetes oder Personen, die unter Tage arbeiten. Solche Personen entwickeln vermutlich mit höherer Wahrscheinlichkeit spezielle Krankheiten.

Während des Trainings lernt der Algorithmus dann aus der Stichprobe die Beziehungen zwischen den x- und y-Daten. Und da maschinelle Lernverfahren nicht auf Hellseherei beruhen, funktioniert das nur, wenn eine erlernbare Beziehung gegeben ist. Die x-Daten müssen Informationsanteile beinhalten, die auch die y-Daten beinhalten.

In manchen Fällen ist diese Beziehung sehr einfach. Zum Beispiel eine lineare Beziehung zwischen dem Alter und den Arztkosten in einer Stichprobe von PatientInnen. Häufig sind Beziehungen zwischen Variablen aber verdeckter. Denken Sie zum Beispiel an eine Entweder-oder-Beziehung: ein Pilz, der nur dann giftig ist, wenn der Hut entweder sehr hell oder sehr dunkel ist. Solche Beziehungen sind schwerer zu analysieren und die meisten einfacheren Algorithmen scheitern daran.

In diesem einführenden Kapitel schauen wir uns einige der wichtigsten Lernalgorithmen und ihre Funktionsweise an. Darunter die lineare Regression, die zur Schätzung stetiger Zielvariablen eingesetzt wird, die logistische und die Softmax-Regression, die zur Klassifi-

zierung eingesetzt werden. Danach kümmern wir uns um das unangenehme aber wichtige Thema der Vorverarbeitung der Daten (Preprocessing). Diese Arbeit wird gerne unterschätzt, ist aber entscheidend dafür, ein Modell dazu zu bringen, etwas aus Daten zu lernen. Zum Schluss nutzen wir dann das Gelernte, um einige einfache Analysen im Bereich der Zeitreihenanalyse durchzuführen.

3.1 Lineare Regression

Auch wenn die verschiedenen Algorithmen maschinellen Lernens in ihrer Anlage und Komplexität variieren, teilen die meisten eine Gemeinsamkeit: Sie basieren auf einem Grundprinzip des Lernens, das sich anhand des einfachsten dieser Verfahren am besten veranschaulichen lässt: an der linearen Regression. Deshalb wollen wir uns zu Beginn mit diesem Verfahren auseinandersetzen.

3.1.1 Grundlagen

Nehmen wir an, Sie führen ein kleines Café. In diesem Café haben sie über einige Tage jeweils die Anzahl der Gäste pro Stunde gezählt. Außerdem haben sie ihren Nettoverdienst in diesem Zeitraum ermittelt. Insgesamt liegen ihnen Informationen über 50 Zeitpunkte in einer Datenmatrix vor.

Nun möchten Sie aus diesen Daten ableiten, wie viel Gewinn sie in einer Stunde abhängig von der Anzahl der Gäste im Durchschnitt erzielen. Das könnte zum Beispiel interessant sein, um herauszufinden, wie viele Gäste Sie mindestens benötigen, um Gewinn zu erzielen und um auf dieser Grundlage die Öffnungszeiten anzupassen, früher oder später zu schließen, weil ab einer bestimmten Uhrzeit nur noch wenige Kunden kommen.

Wie können Sie vorgehen? Um sich mit dem Zusammenhang zwischen den beiden Variablen vertraut zu machen, visualisieren wir zunächst diesen Zusammenhang. Dazu verwenden wir einen Scatterplot, in dem wir die Anzahl der Gäste und den Nettoverdienst jeweils als Punkte in ein Koordinatensystem eintragen(Bild 3.1, links).

Jeder Punkt entspricht einer Datenzeile. Wählen wir als Beispiel den Punkt ganz links mit den Koordinaten $x = 0$, $y = -70$. Er bezieht sich auf eine Betriebseinheit in ihrem Café (eine Stunde), in der Sie keinen einzigen Gast hatten. Das heißt, Sie haben Verlust gemacht. Ziemlich genau 70 Euro. Schließlich mussten sie während dieser Stunde weiterhin ihre Aushilfskraft bezahlen und Sie hatten die üblichen laufenden Kosten: Strom, Wasser usw. Diese Situation trat aber nicht besonders häufig auf. In den meisten Fällen waren zehn oder mehr Gäste im Café, sodass Sie Gewinn erwirtschaftet haben.

Anhand der Punktewolke erkennt man nun ganz gut die Form dieses Zusammenhangs, der auch intuitiv offensichtlich ist: je mehr Gäste, umso mehr Verdienst. Die Gerade, die über die Punktewolke gelegt ist, bildet diesen Zusammenhang modellhaft ab. Genau genommen ist diese Gerade unser Schätzmodell. Darum kümmern wir uns jetzt etwas eingehender.

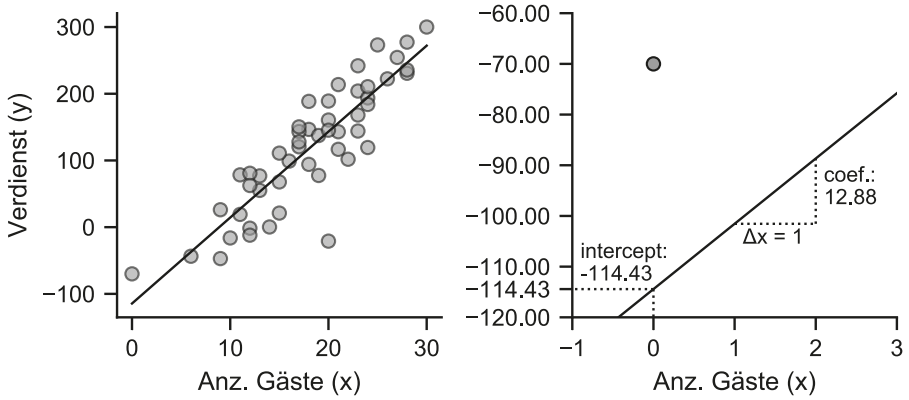


Bild 3.1 Modellierung des Zusammenhangs zweier Variablen mithilfe einer linearen Regression

Die Frage, wie wir zu dieser Gerade kommen, verlegen wir auf später. Schauen wir uns zunächst an, wie uns diese Gerade – das Modell – bei unserer Aufgabenstellung hilft. Wir wollen ja aus bekannten Informationen (x) auf unbekannte Informationen (y) schließen – von der Anzahl der Gäste auf den Verdienst. Genau das können wir jetzt.

Nehmen wir an, wir möchten wissen, wie viel wir verdienen, wenn wir in einer Stunde nur fünf Gäste bewirten. Dafür liegen uns keine Daten aus der Stichprobe vor. Aber nachdem wir das Modell erzeugt haben, brauchen wir die Datengrundlage ja nicht mehr. Stattdessen verwenden wir das Modell, um eine Schätzung durchzuführen. Dazu bewegen wir uns auf der x-Achse von links nach rechts, bis wir die 5 erreicht haben. Den Punkt merken wir uns, denn von dort aus gehen wir im rechten Winkel nach oben, bis wir die Gerade schneiden. Dort angekommen, drehen wir uns um neunzig Grad nach links, wandern von dort aus zur y-Achse und lesen den Wert des Achsenabschnitts ab, auf dem wir uns befinden. Das wären ziemlich genau −50 Euro.

Wenn wir also in einer Stunde nur genau fünf Gäste bewirten, rechnen wir mit 50 Euro Verlust. Genauso können wir es mit beliebigen anderen Gästeanzahlen machen, für die wir den Gewinn ermitteln wollen. Bei 25 Gästen verdienen wir, wenn unser Modell stimmt, etwas mehr als 200 Euro, bei 15 Gästen um die 80 Euro. Bei 9 Gästen liegt ungefähr die Schwelle zwischen Verlust und Gewinn.

Lineare Schätzgleichung

Natürlich würden wir in der Praxis nicht auf diese Weise vorgehen. Die Gerade selbst haben wir ja auch nicht einfach nach Augenmaß in die Punktewolke gelegt. Wir haben dafür ein mathematisches Verfahren benutzt, das wir uns gleich ansehen. Jedenfalls ist die Gerade selbst nur visuelle Repräsentation einer Schätzformel, die so aussieht:

$$\begin{aligned}\hat{y}_{\text{verdienst}} &= \alpha + \beta_{\text{gäste}} \cdot x_{\text{gäste}} \\ &= -114.43 + 12.88 \cdot x_{\text{gäste}}\end{aligned}$$

Die Formel beschreibt, wie wir den Verdienst für unterschiedliche Gästeanzahlen schätzen können. In der ersten Zeile ist die allgemeine Gleichung, in der zweiten die konkrete, für unseren Zweck mit Koeffizienten gefüllte, Gleichung abgedruckt.

Was bedeuten diese Koeffizienten? Sehen wir uns dazu die rechte Grafik in Bild 3.1 an. Sie zeigt einen vergrößerten Ausschnitt des gesamten Koordinatensystems. Der Wert für α entspricht dem *Intercept*. Er beschreibt die vertikale Lage der Kurve. Er bestimmt den Wert der *y*-Variable (Verdienst), wenn die *x*-Variable, die Gästeanzahl, 0 ist. Das kann man anhand der Formel leicht ableiten. Wenn wir $x_{\text{gäste}} = 0$ setzen, multiplizieren wir β mit 0 und erhalten entsprechend 0 zurück, ganz egal, welcher Wert für β festgelegt ist; übrig bleibt nur α. −114.43 ist also der Schätzwert für den Verdienst, den wir bei einer Gästezahl von 0 erzielen.

β beschreibt demgegenüber die Steigung der Geraden (*Slope*). Das ist der zweite Parameter, den wir benötigen, um eine Gerade in ein Koordinatensystem zu setzen. Wenn wir statt einem Gast zwei Gäste bewirten oder statt zehn Gästen elf Gäste, erhöht sich unser Verdienst um durchschnittlich 12 Euro und 88 Cent. β bezeichnet also die Veränderung des Verdienstes (y) bei einer Veränderung der Gästeanzahl (x) um genau eine Einheit.

Wenn wir α und β kennen, können wir den Verdienst für beliebige Gästezahlen schätzen, z. B. für 10 Gäste, 11 Gäste oder 30 Gäste:

$$14.39 = -114.43 + 12.88 \cdot 10$$
$$27.28 = -114.43 + 12.88 \cdot 11$$
$$310.71 = -114.43 + 12.88 \cdot 33$$

Natürlich können wir jetzt auch ganz und gar unsinnige Schätzungen durchführen. Zum Beispiel für −10 Gäste oder auch für 1000 Gäste, für die unser Café gar nicht ausgelegt ist. Die Schätzgerade kennt keinen Anfang und kein Ende. Wenn das aber keinen Sinn macht, müssen wir solche Schätzungen auch nicht durchführen.

Auf der anderen Seite könnten wir mit dieser Formel aber immerhin austesten, wie viel mehr Gewinn wir erzielen würden, wenn wir ein paar Sitzplätze mehr einrichten würden, z. B. indem wir die Tische näher zusammenrücken, um zu den Stoßzeiten mehr Gäste bewirten zu können. Nehmen wir also an, Sie hätten im Moment 33 Sitzplätze, aber es wäre ihnen möglich, wodurch auch immer, auf 40 Sitzplätze aufzustocken. Wenn Sie diese Gästeanzahl mit ihrer Aushilfskraft noch bewältigen können und es kommen tatsächlich so viele Gäste, würde sich ihr Gewinn in dieser Stunde von 310 auf über 400 Euro erhöhen.

Das sind die Optionen, die solche Modelle bieten. Sie können Szenarien austesten, Öffnungs- und Schließzeiten optimieren und darüber nachdenken, ob es sich lohnen könnte, das Café zu erweitern oder herausfinden, was sie verlieren, wenn sie renovieren müssen und in dieser Zeit einige Sitzplätze wegfallen.

Geschätzte vs. tatsächliche Werte: die Residuen

Jetzt wissen wir immerhin, was wir mit einem Modell anstellen können, wenn es angelernt ist. Die wichtigste Frage haben wir aber noch nicht geklärt: Woher beziehen wir die Koeffizienten, um die Formel zu füllen? Mit welchem mathematischen Verfahren lassen sich Lage und Steigung der Geraden bestimmen?

Offensichtlich müssen diese Parameter aus den Daten abgeleitet werden, auf deren Grundlage das Modell angelernt wurde. Um zu verstehen, wie das funktioniert, werfen wir noch einmal einen Blick in die Punktewolke. Diesmal sehen wir uns aber die Abstände der Punkte von der Geraden an, die *Residuen* (Bild 3.2). Die Residuen können wir aus zwei Gründen gebrauchen: Zum einen können wir aus ihnen die Qualität eines fertigen Schätzmodells ableiten. Zum anderen können wir uns an den Residuen bei der Optimierung der Gewichte orientieren.

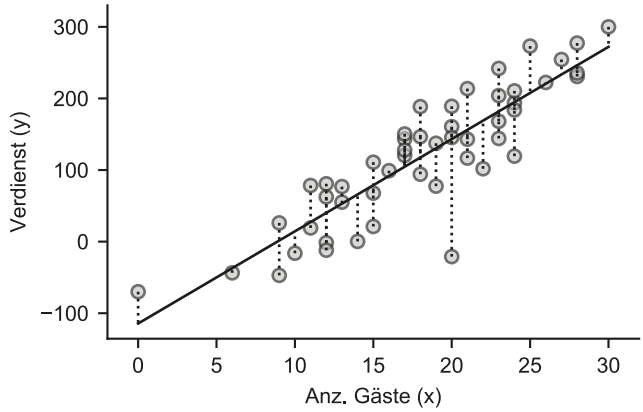


Bild 3.2 Lineare Regression, Residuen

Kümmern wir uns zuerst um die Qualität des Modells. Wir haben zwar bei der Geraden, die wir für die Schätzungen verwenden, ein ganz gutes Gefühl. Aber wir wissen auch, dass unsere Schätzungen nicht perfekt sind. Wir können also nicht erwarten, dass wir aus der Anzahl von Gästen ganz genau prognostizieren können, wie viel wir verdienen werden. Das liegt daran, dass die Assoziation zwischen den beiden Variablen nicht perfekt ist. Nicht alle Gäste konsumieren schließlich die gleiche Menge an Getränken oder Speisen. Zum Beispiel liegen uns für den Verdienst bei neun Gästen insgesamt zwei Messwerte vor. Das eine Mal haben wir ca. 47 Euro Verlust gemacht, das andere Mal 26 Euro Gewinn. Noch stärker sind die Unterschiede bei 20 Gästen; dafür liegen uns insgesamt 4 Messwerte vor; drei davon liegen relativ nahe an der Schätzgeraden, mit ihnen haben wir zwischen 145 Euro und 190 Euro Gewinn erzielt. Der vierte Datenpunkt ist ein Ausreißer – was auch immer in dieser Stunde passiert ist, vielleicht haben mehrere Kunden die Zeche geprellt – jedenfalls haben wir in diesem Fall 21 Euro Verlust gemacht.

Generell kann man sagen, dass ein Modell umso besser ist, je geringer die Residuen sind, je näher die Datenwolke an der Geraden liegt. Bei einem perfekten Modell würden alle Punkte wie an einer Perlenschnur an der Gerade aufgereiht sein. Bei einem sehr schlechten Modell sehen wir dagegen keinerlei Annäherung zwischen der Wolke und der Linie.

Wir wollen uns aber bei der Evaluation nicht von visuellen Eindrücken leiten lassen. Wir wollen mithilfe eines belastbaren Indikators die Qualität unseres Modells benennen. Dazu können wir zum Beispiel die Abstände jedes Datenpunkts von der Geraden messen, den Absolutwert über die Differenzen bilden[1] und dann über die Einzelwerte den Mittelwert berechnen. Je kleiner dieser Wert, umso besser das Modell. In unserem Fall liegt er (*mean absolute error*) bei 33,9. Das heißt, durchschnittlich liegen wir um 33 Euro und 90 Cent daneben, wenn wir einen Schätzwert mithilfe der Gästeanzahl erzeugen. Ob wir diesen Fehler akzeptieren oder nicht akzeptieren, kommt natürlich auf den Kontext an und darauf, was wir uns von einem Modell versprechen. Besser können wir mit einer linearen Regression und auf Grundlage der Gästezahl aber nicht schätzen.

[1] Damit sich positive und negative Abweichungen bei der Summierung nicht aufheben.

Verlustfunktion und Einstellung der Gewichte

Aber woher wissen wir das eigentlich? Woher wissen wir, dass wir das bestmögliche Modell gefunden haben?

Weil wir uns bei der Festlegung der Koeffizienten für die Regressionsgerade an den Residuen orientiert haben. Wir haben jene Koeffizienten gewählt, bei der die Summe der Residuen zwischen Gerade und den Datenpunkten den kleinstmöglichen Wert annehmen.

Das Verfahren, das wir dafür benutzt haben, nennt sich *Ordinary Least Square*. Es orientiert sich zwar nicht am Mittelwert über die absoluten Abweichungen, sondern an einer anderen Kenngröße, die nur wenig verschieden davon ist: am Mittelwert bzw. an der Summe über die quadratischen Abweichungen. Statt die Beträge über die Residuen zu bilden, werden die Residuen also quadriert und anstatt den Mittelwert zu bilden, wird die Summe über die einzelnen Differenzen berechnet. Die Quadratur bewirkt lediglich, dass starke Abweichungen höher gewichtet werden und damit mehr zur Einstellung der Koeffizienten beitragen, der Einfluss geringerer Abweichungen wird dagegen abgeschwächt.

Mit diesem Werkzeug können wir die Regressionsgerade ermitteln. Vergessen wir einmal, um es leichter zu machen, das Intercept und konzentrieren uns nur auf die Ermittlung der Steigung der Geraden.[2] Und nehmen wir außerdem an, wir würden bei der Ermittlung iterativ vorgehen. Das ist nämlich das, was die meisten Verfahren tun. Da wir zu Beginn nicht wissen, welchen Wert der Koeffizient annehmen muss, um unsere Daten bestmöglich abzubilden, setzen wir einfach einen Zufallswert ein. Wir beginnen zum Beispiel mit einem β von −1.7 – warum auch immer, zum Beispiel, weil wir einen Zufallsgenerator eingesetzt haben, um den Startwert zu bestimmen. Danach produzieren wir auf dieser Grundlage Schätzergebnisse. Die Formel würde zu Beginn (bei gegebenem Intercept) also so aussehen:

$$\hat{y}_{\text{verdienst}} = -114.43 + \left(-1.7\right) \cdot x_{\text{gäste}}$$

Mit dieser Formel können wir für alle verfügbaren x-Daten Schätzwerte für die *y*-Variable erzeugen, und dann die mittleren quadratischen Abweichungen von den realen y-Werten berechnen. Wir erhalten jetzt einen Wert, der ungefähr bei 80770 liegt.

Gradientenabstieg

Es ist natürlich sehr unwahrscheinlich, dass dieser Zufallswert schon nahe am Optimum für den Koeffizienten liegt und das tut er auch nicht, wie wir gleichen sehen werden. Aber jetzt haben wir immerhin einen Richtwert, an dem wir uns orientieren können. Und da wir uns für ein iteratives Verfahren entschieden haben, können wir auf dieser Grundlage schrittweise optimieren. Das heißt, von unserem derzeitigen Standort aus die richtige Richtung einschlagen. Im nächsten Schritt ermitteln wir, ob wir unseren Koeffizienten erhöhen oder reduzieren müssen, um einen geringen Wert für die mittleren quadratischen Abweichungen zu erzielen.

Wie geht das? Schauen wir, um das zu verstehen, die Abhängigkeit des mittleren quadratischen Fehlers vom Regressionskoeffizienten in einer Grafik an. Dazu setzen wir in unserer Schätzformel eine Bandbreite von Werten für β ein. Wir fangen zum Beispiel bei −5 an und gehen dann in kleinen Schritten (0,1) weiter, bis wir einen selbst gesetzten Maximalwert von 25 erreichen.

[2] Genau genommen können wir das *Intercept* wie einen Steigungskoeffizienten behandeln und anlernen. Der einzige Unterschied besteht darin, dass wir den Steigungskoeffizienten für jede Zeile in unserem Datensatz mit einem vorgegebenen x-Wert von 1 anlernen.

Für jeden dieser β-Werte erzeugen wir dann Schätzwerte über unsere gesamten Daten und für jedes dieser Ergebnisse berechnen wir jeweils die quadratischen Abweichungen.

Resultat ist die bohlenförmige Kurve, die in Bild 3.3 dargestellt ist. Auf der x-Achse sind die verschiedenen β-Koeffizienten abgetragen, auf der y-Achse die mittleren quadratischen Fehler, die mit diesen Koeffizienten auf Grundlage unserer Daten entstehen.

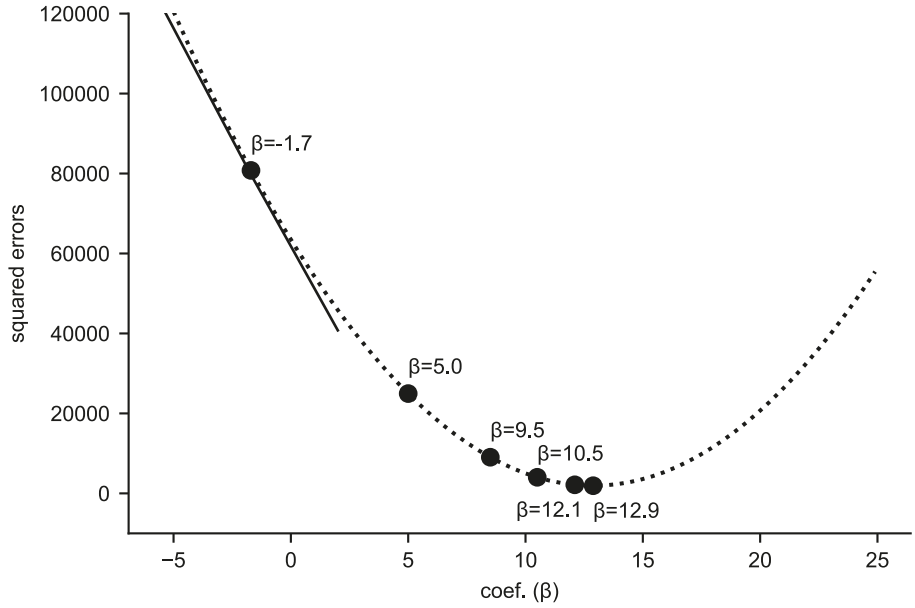


Bild 3.3 Verlustfunktion: Mittlere quadratische Abweichungen für verschiedene Koeffizienten.

Anhand dieser Kurve – der sogenannten *Verlustfunktion* (*Loss*) – sehen wir, wo wir uns im Moment befinden, und wir sehen, wohin wir uns bewegen müssen: ins Tal der Kurve. Dort finden wir den kleinstmöglichen Wert für den mittleren quadratischen Fehler. Er liegt bei 1.916.

Das Problem ist nur, dass wir den Verlauf dieser Kurve in diesem Moment nicht kennen. Wir sind blind dafür. Im Moment sind wir das natürlich nicht. Aber bei einem komplexen Problem, in dem wir nicht nur eine *x*-Variable, sondern eine Vielzahl von Features einsetzen, ist das anders. In solchen Fällen können wir nicht mehr jede Kombination aus Gewichten für jede Variable ausprobieren und dann den tiefsten Punkt auf der Verlustfunktion ablesen.

In der Literatur finden Sie dazu häufig eine Geschichte, die in etwa so geht:

Nehmen Sie an, Sie befinden sich in der Lage eines Bruchpiloten. Er hat in der Nacht die Kontrolle über sein Flugzeug verloren und sich mit dem Fallschirm gerettet. Dummerweise ist er irgendwo im Gebirge gelandet und muss nun versuchen, so schnell wie möglich ins Tal zu gelangen. Da er die Landschaft in der Dunkelheit nicht überblicken kann, weiß er nicht, wohin er gehen muss. Seine einzige Möglichkeit sich zu orientieren besteht darin, einen Schritt vor den anderen Schritt zu setzen und sich dorthin zu bewegen, wo es nach unten geht. Genau das Gleiche tun wir auch. Wir bewegen uns nach unten.

Aber wie finden wir ohne Tastsinn heraus, ob wir nach links (Reduktion des Koeffizienten) oder nach rechts (Erhöhung des Koeffizienten) gehen müssen, um ins Tal der Verlustfunktion zu gelangen?

Wir bedienen uns des Verfahrens der Differenzialrechnung. Wir legen, wie in Bild 3.3 angedeutet, eine Tangente an, an deren Neigung wir erkennen, in welche Richtung es ins Tal geht. Dadurch wissen wir, dass wir den Koeffizienten erhöhen müssen, um nach unten zu kommen. Das ist eine wichtige Erkenntnis. Das heißt aber noch lange nicht, dass wir dadurch schon wüssten, um welchen Betrag wir den Koeffizienten nach oben schrauben müssen, um in der Talsohle zu landen. Vielmehr können wir immer nur um einen festgelegten Betrag, Schritt für Schritt, wie der Bruchpilot, nach unten schreiten.

Das soll uns aber im Moment nicht stören. Nehmen wir weiter an, wir hätten die Möglichkeit, diesen Betrag, die *Lernrate*, dynamisch anzupassen, sodass sie bei jedem Schritt kleiner wird. Beim ersten Durchlauf beträgt die Lernrate 6,7. Das ist zwar ziemlich unrealistisch, aber für das Beispiel ganz geschickt. Wir inkrementieren den Koeffizienten β also um den Betrag von 6,7 und erhalten dann einen neuen Wert für β von 5,0.

Diesen Wert können wir in die Schätzgleichung einsetzen, damit wieder Schätzungen über alle Trainingsdaten produzieren und dann aus dem Vergleich mit den wahren y-Daten wieder die quadratischen Fehler der Residuen ermitteln. Wir kommen jetzt auf einen mittleren quadratischen Fehler von 24.961. Das ist schon deutlich besser als der Anfangswert. Aber es ist noch nicht der beste Wert.

Aber jetzt wissen wir ja, wie wir uns weiter vorarbeiten können. Wir legen also wieder die Tangente an und schauen, ob wir noch weiter nach rechts oder nach links gehen müssen (es könnte ja auch sein, dass wir durch unsere Schrittweite die Talsohle versehentlich übersprungen haben). Dann ändern wir den Koeffizienten wieder um einen festgelegten Betrag und wiederholen den Vorgang: Schätzwerte mit dem neuen β berechnen, quadratische Abweichungen ermitteln, Tangente anlegen usw. Wenn wir nach einigen weiteren Iterationsschritten – abhängig von der Lernrate – im Tal angelangt sind, sind wir fertig. Dann haben wir das bestmögliche Modell gefunden.

In Bild 3.4 ist der iterative Prozess der Anpassung des Steigungskoeffizienten über die Residuen und das Differenzialverfahren aus der Perspektive der Regressionsgeraden dargestellt. Man sieht, dass der Koeffizient, den wir zu Beginn für das β per Zufall festlegen, eine völlig unpassende Gerade legt, mit der sich die Assoziation zwischen x und y nicht einmal annäherungsweise abbilden lässt. Das Verfahren der inkrementellen Anpassung tastet sich langsam an eine bessere Lösung heran, Schritt für Schritt wird der Koeffizient in die richtige Richtung gedreht, bis das Optimum erreicht ist.

Das ist das Grundprinzip des Modelllernens. Sie werden es so oder so ähnlich in einer Reihe von Anwendungen des maschinellen Lernens finden. Bei linearen Regressionen, deren Koeffizienten in der Regel nach dem Prinzip der kleinsten Quadrate geschätzt werden, der sogenannten Ordinary Least Square Regression (OLS), wird allerdings ein anderes Verfahren eingesetzt. Da die Verlustfunktion in diesen Modellen nicht besonders komplex ist, ist es ohne Weiteres möglich, in einem einzigen Rechenschritt das Minimum zu ermitteln. Damit spart man sich die iterative Anpassung.

In einem neuronalen Netz müssen dagegen eine Vielzahl von Gewichten in unterschiedlichen Schichten trainiert werden. Dort kommt das oben beschriebene Verfahren, das sogenannte Gradientenabstiegsverfahren, zum Einsatz. Ähnliches gilt für die logistische Regression, die

allerdings eine andere Verlustfunktion (Maximum Likelihood) zum Anlernen verwendet. Auch dieses Verfahren lernt iterativ mithilfe eines Optimierers, da die Lösung nur in Ausnahmefällen in einem einzigen Rechenschritt ermittelt werden kann [Green2003, S. 468 ff.].

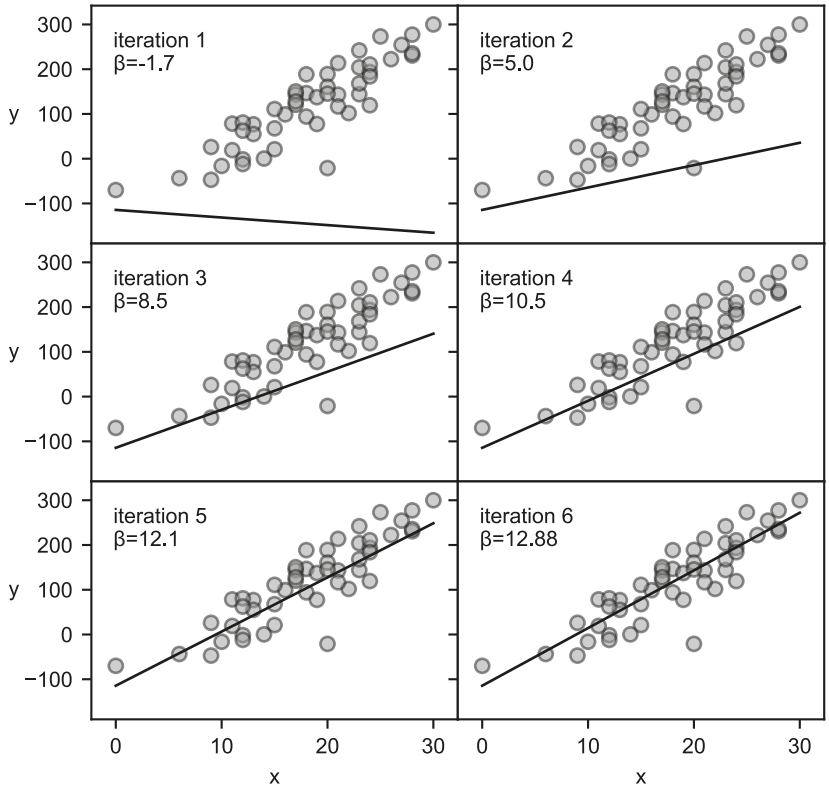


Bild 3.4 Iterative Anpassung des Steigungskoeffizienten (β)

3.1.2 Umsetzung mit Scikit-learn

Die Python-Bibliothek Scikit-learn verfügt nicht nur über eine leicht verständliche API, sie beinhaltet auch alle relevanten Klassen, die wir für maschinelle Lernaufgaben benötigen. Wir nutzen also die Gelegenheit, uns anhand einer einfachen Beispielanwendung mit der praktischen Umsetzung der linearen Regression und zugleich mit dem Umgang mit Scikit-learn vertraut zu machen.

Nehmen wir, zur Veranschaulichung, einen simulierten Datensatz, der Kunden einer Krankenversicherung beinhaltet. Um das Beispiel etwas anschaulicher zu machen, gehen wir davon aus, dass es sich um eine private Krankenversicherung handelt. Außerdem versetzen wir uns in die Rolle eines Versicherungsmaklers, der Neukunden gewinnen möchte. Um einem Neukunden ein angemessenes Angebot für eine Krankenversicherung zu unterbreiten, ist es für den Makler vorteilhaft, wenn er bereits einen Richtwert über die Kosten ermitteln kann, die ein Kunde vermutlich verursacht.

Natürlich könnte man dafür einfach den Mittelwert oder Median der jährlichen Arztkosten über alle Bestandskunden berechnen. Allerdings variieren diese Kosten vermutlich sehr stark von Kunde zu Kunde, wobei die Schwankungen nicht einfach zufällig auftreten, sondern zum Beispiel nach Alter, Vorerkrankungen, Lebenswandel etc. variieren. Was liegt also näher, als die Daten der Bestandskunden, die neben den Arztkosten eine Reihe weiterer Informationen beinhalten, auszunutzen, um ein Prognosemodell anzulernen. Wenn das gelingt, kann der Makler mit einigen Hintergrundinformationen über den potenziellen Kunden das System füttern, und darüber womöglich einen weitaus präziseren Schätzwert für die jährlichen Kosten gewinnen.

Das ist also unsere Aufgabe. Dabei beschränken wir uns zuerst auf einige wenige Features, um den Anlernprozess zu demonstrieren. Wir laden zunächst die Daten, wie gewohnt, als Pandas Data-Frame (*df*). Bei den Features stehen uns Alter (*age*), Body Mass Index (*bmi*) und Anzahl Kinder (*children*) zur Verfügung. Die Zielvariable bilden die jährlichen Arztkosten (*charges*), gemessen in US$. Insgesamt liegen uns 1138 Fälle vor. Sehen wir uns die ersten fünf Zeilen des Datensatzes an:

```
   age  bmi     children  charges
0  19   27.900  0         16884.924
1  18   33.770  1         1725.552
2  28   33.000  3         4449.462
3  33   22.705  0         21984.471
4  32   28.880  0         3866.855
```

Beginnen wir damit, die Daten besser kennenzulernen. Zum Beispiel haben wir die Vermutung, dass das Alter für die Erklärung der Arztkosten relevant sein könnte. Also stellen wir den Zusammenhang mithilfe eines Scatterplots dar, in der Hoffnung, dass sich diese These schon rein visuell bestätigt.

In Bild 3.5 (links) sehen wir das Ergebnis – auf der x-Achse ist das Alter, auf der y-Achse sind die Arztkosten abgetragen. Abgesehen von den drei vertikalen Lagen, die offensichtlich auf andere Faktoren als das Alter zurückgehen, zeigt sich der Zusammenhang wie erwartet: je älter eine Person, umso höher die Arztkosten.

Versuchen wir also, diese einfache Assoziation mithilfe einer linearen Regression zu modellieren.

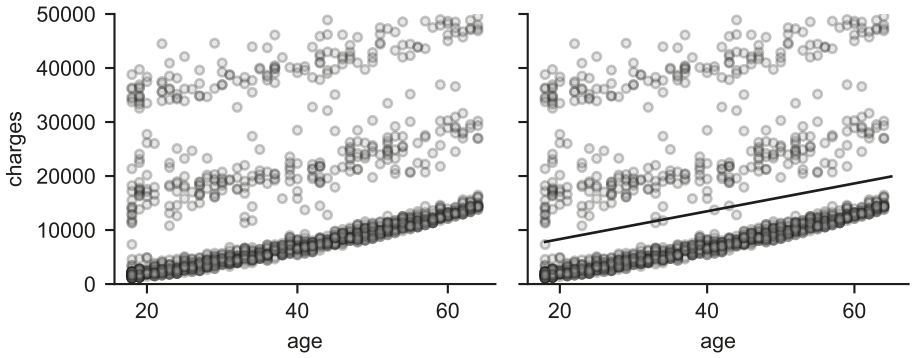


Bild 3.5 Zusammenhang zwischen Alter und Arztkosten (Beispieldaten)

Im ersten Schritt müssen wir die Features (X) und die Zielvariable (y) in separaten Arrays abspeichern. Dazu greifen wir einfach in den Data-Frame und lösen die Variablen heraus. Auch wenn das im Moment kein Problem ist, weil wir die beiden Zeilen aus einem Data-Frame ziehen, sollten wir uns im Klaren sein, dass Scikit-learn erwartet, dass sich korrespondierende x- und y-Daten anhand der Indexposition der beiden Arrays zuordnen lassen. Die *charges* an der Indexposition 0 des y-Arrays beziehen sich also auf den Datensatz mit dem Alter an der Indexposition 0 innerhalb des x-Arrays.

Außerdem müssen wir Scikit-learn die x-Daten zum Anlernen als zweidimensionales Array übergeben. Das gilt auch dann, wenn wir, wie in unserem Fall, nur einen einzigen Prädiktor verwenden. Jede Einheit, die einen Fall bezeichnet, muss in ein inneres Array verschachtelt sein. Bei den y-Daten ist das anders. Da in den meisten Fällen nur eine Zielvariable existiert, übergeben wir ein eindimensionales Array, in dem jeder Fall als einzelner Zahlenwert vorliegt.

```
1.  X = df[['age']].values
2.  y = df['charges'].values
3.  print(X.shape, y.shape)
4.  print(X[:5])
```

Ausgabe:

```
(1338, 1), (1338,)
[[19], [18], [28], [33], [32]]
```

Code-Hinweise: In *Zeile 1* lösen wir die x-Daten (das Alter) als NumPy-Array (*values*) heraus. Beachten Sie dabei die doppelten eckigen Klammern, mit denen wir den Data-Frame dazu veranlassen, ein zweidimensionales Array zurückzugeben. In *Zeile 2* verwenden wir dagegen nur eine einzige eckige Klammer, weil wir ein eindimensionales Array brauchen. In *Zeile 3* schauen wir uns noch die Struktur der Daten unter Aufruf des Attributs *shape* an, und in *Zeile 4* geben wir die ersten fünf Datensätze zur Ansicht aus. Darunter sind die Outputs dargestellt. X und y-Daten beinhalten jeweils 1338 Zeilen (Fälle). In den x-Daten sind die Altersangaben jeweils in ein extra Array (1338,1) eingeschlossen (zweidimensional), die y-Daten liegen als eindimensionales Array mit der Shape 1338 vor (eindimensional).

Nachdem wir die Daten vorbereitet haben, können wir das Modell instanziieren und anlernen. Dazu brauchen wir eine Instanz der Klasse *LinearRegression* aus Scikit-learn. Das Anlernen geschieht dann unter Aufruf der *fit*-Methode und unter Übergabe der x- und y-Daten. Das ist alles.

Die Klasse führt im Hintergrund den Anlernprozess durch und speichert das Intercept und die Gewichtung(en) für die Variable(n) als Attribute. Wenn wir möchten, können wir Intercept und Gewichte abfragen. In jedem Fall können wir mit der angelernten Instanz jetzt Schätzungen der Arztkosten für beliebige Altersangaben durchführen. Dabei müssen wir nur beachten, dass wir uns an das Format halten, das schon die *fit*-Methode verlangt: Die zu schätzenden Daten werden in einem zweidimensionalen Array übergeben:

```
1.  from sklearn.linear_model import LinearRegression
2.  model = LinearRegression()
3.  model.fit(X, y)
4.  intercept = model.intercept_
5.  coef = model.coef_
6.  y_pred = model.predict([[18], [65]])
7.  print('intercept {:.3f}, coef. age {}'.format(intercept, coef))
8.  print('predictions:', y_pred)
```

Ausgabe:

```
intercept: 3165.885, coef. age: [257.723]
predictions: [ 7804.892, 19917.855]
```

Code-Hinweise: Wenn Sie Scikit-learn installiert haben (entweder über *pip* oder *conda*), können Sie die Bibliothek und ihre Klassen importieren (*Zeile 1*). In *Zeile 2* instanziieren wir ein Objekt vom Typ *LinearRegression*. Um das Modell anzulernen, rufen wir die *fit*-Methode auf (*Zeile 3*) und übergeben zuerst die x- und dann die y-Daten. Danach hat das Modell die Koeffizienten berechnet und wir können damit arbeiten. In *Zeile 4* und *5* rufen wir aus dem angelernten Modell das Intercept und den Steigungskoeffizienten des einzigen Prädiktors (Alter) ab. Unter Verwendung dieser Koeffizienten produziert das Modell Schätzwerte, sobald wir die *predict*-Methode aufrufen (*Zeile 6*). Nehmen wir an, wir möchten Schätzwerte für eine Person, die 18 Jahre alt ist, und für eine Person, die 65 Jahre alt ist, also übergeben wir ein zweidimensionales Array mit den entsprechenden Werten. Die Rückgabe fangen wir in *y_pred* ab. Danach leiten wir nur noch die Ausgabe der Koeffizienten und die Ergebnisse der Prognose ein (*Zeile 7* und *8*).

Das Intercept und den Steigungskoeffizienten des angelernten Modells können wir wieder in die zugrunde liegende Schätzgleichung der Regression eintragen, wenn wir das möchten:

$$\begin{aligned}\hat{y}_{\text{charges}} &= \alpha + \beta_{\text{age}} \cdot x_{\text{age}} \\ &= 3165.885 + 257.723 \cdot x_{\text{age}}\end{aligned}$$

Das ist die Formel, die auch die *predict*-Methode im Hintergrund verwendet, um Schätzwerte zu erzeugen. Zum Beispiel, wenn wir die Arztkosten pro Jahr für zwei Personen mit 18 und 65 Jahren herausfinden möchten:

$$\begin{aligned}\hat{y}_{\text{charges (18 years)}} &= 3165.885 + 257.723 \cdot 18 \\ &= 7804.892\end{aligned}$$

$$\begin{aligned}\hat{y}_{\text{charges (65 years)}} &= 3165.885 + 257.723 \cdot 65 \\ &= 19917.855\end{aligned}$$

Das Modell können wir auch graphisch darstellen, indem wir einfach die Regressionsgerade in die Punktewolke legen (Bild 3.5, rechts). Die vertikale Lage der Geraden wird durch das Intercept eingestellt. Eine Person, die 0 Jahre alt ist, weist in unserem Modell also Arztkosten in Höhe von 3165 Euro auf. Die Steigung ergibt sich aus dem β-Koeffizienten. Mit jedem Jahr, das eine Person älter wird, steigen die Arztkosten um ungefähr 257 Euro an.

Jetzt müssen wir das Modell noch evaluieren. Wir müssen herausfinden, wie gut es arbeitet, damit wir wissen, was wir bei einer Schätzung erwarten können. Dazu berechnen wir den mittleren absoluten Fehler und das R-Quadrat. Den mittleren absoluten Fehler kennen wir schon, das R-Quadrat sehen wir uns gleich noch etwas genauer an.

Bei der Berechnung hilft uns Scikit-learn. Im Package *metrics* finden sich zur Berechnung beider Kenngrößen Funktionen, die wir verwenden:

```
1.  from sklearn.metrics import mean_absolute_error, r2_score
2.  y_pred = model.predict(X)
3.  mae = mean_absolute_error(y, y_pred)
4.  r2 = r2_score(y, y_pred)
5.  print('mean absolute error: {:.3f}, r2: {:.3f}'.format(mae, r2))
```

Ausgabe:

```
▶ mean absolute error: 9055.150, r2: 0.089
```

Code-Hinweise: Nachdem wir die Funktionen importiert haben (*Zeile 1*), erzeugen wir zuerst Schätzwerte. Da wir im Moment Trainings- und Testdaten noch nicht unterscheiden, übergeben wir dazu die kompletten x-Daten, mit dem wir das Modell angelernt haben (*Zeile 2*). In *Zeile 3* und *4* verwenden wir dann die Funktionen, um den mittleren absoluten Fehler bzw. das R-Quadrat zu berechnen. Dabei übergeben wir zuerst die wahren Arztkosten aus unseren Trainingsdaten (*y*) und dann die von unserem Modell geschätzten Arztkosten (*y_pred*). Aus den Differenzen werden dann die jeweils gewünschten Kenngrößen berechnet. In *Zeile 5* geben wir die berechneten Werte auf der Konsole aus.

Die Ausgabe des mittleren absoluten Fehlers zeigt, dass wir bei einer Schätzung im Durchschnitt um 9055 Euro im Jahr danebenliegen. Das ist nicht besonders gut. Aber denken Sie daran, dass wir nur eine einzige Variable, das Alter, zum Anlernen des Modells verwendet haben. Dafür ist das Ergebnis immer noch deutlich besser, als wenn wir einfach den Mittelwert über alle Daten als Grundlage für Prognosen verwendet hätten. Dann würden wir uns im Durchschnitt nämlich um 13231 Euro verschätzen.

Das zweite Gütemaß, das R-Quadrat, berechnet sich aus dem Vergleich der durch die Prädiktoren erklärten Varianz der *y*-Variable in Relation zur Gesamtvarianz. Der Vorteil des R-Quadrats ist, dass es (zumindest theoretisch) nur Werte zwischen 0 und 1 annehmen kann. In jedem Fall können wir damit unterschiedliche Modelle, die auf unterschiedlichen Daten angelernt wurden, miteinander vergleichen.

Außerdem lässt sich das R-Quadrat relativ leicht interpretieren: Ein Wert von 1 bedeutet, dass wir alle Unterschiede der *y*-Variable mithilfe der Prädiktoren erklären können. Ein Wert von 0 bedeutet, dass wir überhaupt keine Unterschiede erklären können. Je näher der Wert also bei eins liegt, umso besser ist unser Modell. Der Nachteil des R-Quadrats gegenüber dem mittleren absoluten Fehler ist, dass es relativ abstrakt ist. Es sagt wenig darüber aus, wie gut unser Modell im Hinblick auf eine spezifische Aufgabe funktioniert.

Bezogen auf unsere Aufgabe liegt das R-Quadrat bei 0,089. Wir können also ungefähr 9 Prozent der Varianz der Arztkosten durch das Alter erklären.

Was haben wir jetzt für Möglichkeiten, das Modell zu verbessern? Die wichtigste Option besteht im Moment darin, weitere Prädiktoren aufzunehmen. Dazu stehen im Moment der Body Mass Index und die Anzahl der Kinder zur Verfügung. Also nehmen wir diese Variablen einfach in die Reihe der erklärenden Variablen auf. Aus der bivariaten wird eine multivariate Regression, die für jeden Prädiktor einen separaten Koeffizienten anlernt:

$$\hat{y}_{\text{charges}} = \alpha + \beta_{\text{age}} \cdot x_{\text{age}} + \beta_{\text{bmi}} \cdot x_{\text{bmi}} + \beta_{\text{children}} \cdot x_{\text{children}}$$

Die Regressionsformel wird also einfach um die zusätzlichen Prädiktoren erweitert. Für jeden Prädiktor wird ein separater β-Koeffizient angelernt, dessen Wert auf das Schätzergebnis aufgeschlagen wird.

Um Scikit-learn dazu zu bringen, diese Funktion anzulernen, müssen wir nicht mehr tun, als der *fit*-Methode unserer Linear-Regression alle *x*-Variablen, mit denen wir das Modell ausrüsten wollen, zu übergeben. Im Hintergrund wird dann für jede Spalte ein separater Koeffizient angelernt:

```
6.  X = df[['age', 'bmi', 'children']].values
7.  y = df['charges'].values
8.  model = LinearRegression()
9.  model.fit(X, y)
10. intercept = model.intercept_
11. coef = model.coef_
12. y_pred = model.predict(X)
13. mae = mean_absolute_error(y, y_pred)
14. r2 = r2_score(y, y_pred)
15. print('intercept: {:.3f}, coefs.: {}'.format(intercept, coef))
16. print('mean absolute error: {:.3f}, r2: {:.3f}'.format(mae, r2))
```

Ausgabe:

```
intercept: -6916.243, coefs.: [239.994, 332.083, 542.865]
mean absolute error: 9015.442, r2: 0.120
```

Statt eines einzelnen Koeffizienten erhalten wir jetzt drei Koeffizienten. Sie werden bei Abfrage des Attributs *coef_* in der gleichen Reihenfolge, in der wir sie der *fit*-Methode übergeben haben, zurückgeliefert. An der Indexposition 0 finden wir den Wert für das Alter, an der Indexposition 1 den für den Body Mass Index und an der Indexposition 2 den Wert für die Anzahl der Kinder. Daraus ergibt sich folgende Schätzgleichung:

$$\hat{y}_{\text{charges}} = -6916.243 + 239.99 \cdot x_{\text{age}} + 332.08 \cdot x_{\text{bmi}} + 542.87 \cdot x_{\text{children}}$$

Und natürlich müssen wir uns jetzt, wenn wir die *predict*-Methode aufrufen, an diese Reihenfolge bei der Übergabe von x-Daten, für die wir Schätzwerte erhalten möchten, halten. Um die Arztkosten für eine 40-jährige Person mit einem Body Mass Index von 20,1 und einem Kind zu erhalten, rufen wir die Methode mit folgendem Array im Gepäck auf:

```
model.predict([[40., 20.1, 1.]])
```

Ausgabe:

```
[9901.276]
```

Im Hintergrund werden dann die Werte in die mit Schätzkoeffizienten gefüllte Regressionsgleichung eingesetzt und das Ergebnis zurückgeliefert:

$$\begin{aligned}\hat{y}_{\text{charges}} &= -6916.243 + 239.99 \cdot 40 + 332.08 \cdot 20.1 + 542.87 \cdot 1 \\ &= 9901.28\end{aligned}$$

Die Evaluationsergebnisse des neuen Modells sind allerdings etwas enttäuschend. Durch Hinzunahme der beiden zusätzlichen Variablen verbessert sich das R-Quadrat gerade mal um drei Prozentpunkte von 9 % auf 12 %. Und der durchschnittliche mittlere Schätzfehler reduziert sich nur von 9055 Euro auf 9015 Euro.

Das ist aber kein Problem unseres Modells, sondern ein Problem unserer Daten. Weder der Body Mass Index noch die Anzahl der Kinder sind offensichtlich gute Prädiktoren für die Arztkosten – oder die relevanten Informationen, die sie enthalten, sind bereits über das Alter in die Regression eingeflossen. Wenn wir andere Informationen hinzunehmen würden, z. B. bekannte Vorerkrankungen, würde sich das bestimmt ändern – das R-Quadrat würde steigen, der mittlere quadratische Fehler sinken.

3.1.3 Trainings- und Testdaten separieren

Bevor wir uns mit anderen Verfahren des überwachten Lernens auseinandersetzen, den Klassifizierern, nutzen wir die Gelegenheit und machen uns mit der typischen Vorgehensweise beim Training und der Evaluation von Lernalgorithmen vertraut.

Was wir im Abschnitt zuvor zu Demonstrationszwecken getan haben, würden wir in der Praxis eher vermeiden. Wir haben unser Modell über die Daten, mit denen wir es angelernt haben, evaluiert. Das Problem dieser Vorgehensweise besteht darin, dass alle Lernalgorithmen – manche mehr, andere weniger – zur Überanpassung neigen. Sie stellen die Koeffizienten auf das Datenmaterial ein, an dem wir sie anlernen. Das sollen sie auch tun. Allerdings sollen sie aus diesen Daten Modelle ableiten. Sie sollen die Trainingsdaten nicht „auswendig lernen".

Im Moment ist das zwar noch kein großes Problem. Schon allein, weil wir mit wenigen Variablen und einfachen Algorithmen arbeiten, die über zu wenig Gewichte verfügen, um sich allzu sehr an die Besonderheiten der Trainingsdaten anzupassen. Trotzdem sollten wir überprüfen, ob das, was ein Algorithmus aus den Trainingsdaten gelernt hat, auch noch funktioniert, wenn wir mit Daten arbeiten, die der Algorithmus nicht kennt. Schließlich wollen wir ja genau das, wenn wir später ein Modell in der Produktion einsetzen: Schätzungen unbekannter Daten durchführen.

Bevor man einen Algorithmus anlernt, sollte man deshalb immer einen Teil der Daten für Testzwecke separieren. Wir unterteilen die Gesamtdaten also in eine Trainingspartition, die wir zum Anlernen verwenden, und in eine Testpartition, an der wir das angelernte Modell evaluieren.

Unser Workflow sieht jetzt also folgendermaßen aus:

a) *Extraktion der x- und y-Daten aus dem Data-Frame (df):*

```
1.  X = df[['age', 'bmi', 'children']].values
2.  y = df['charges'].values
```

b) *Unterteilung der Daten in Trainings- und Testpartitionen im Verhältnis 80:20:*

```
3.  from sklearn.model_selection import train_test_split
4.  X_train, X_test, y_train, y_test = train_test_split(X, y,
5.                                      test_size=0.2, random_state=11)
6.  X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

Ausgabe:

```
(1070, 3), (268, 3), (1070,), (268,)
```

c) *Instanziierung und Anlernen des Modells über die Trainingsdaten:*

```
7.  model = LinearRegression()
8.  model.fit(X_train, y_train)
```

d) *Evaluation des Modells über die Testdaten:*

```
9.  y_test_pred = model.predict(X_test)
10. mae = mean_absolute_error(y_test, y_test_pred)
11. r2 = r2_score(y_test, y_test_pred)
```

Ergebnisse:

```
mean absolute error (test): 9057.371
R-Square (test): 0.083
```

Code-Hinweise: Das meiste kennen Sie jetzt schon. Neu ist nur der Arbeitsschritt *b)*. Wir importieren und verwenden die *train_test_split*-Funktion aus Scikit-learn (*Zeile 3*), um die Daten in Trainings- und Testdaten zu unterteilen. Der Funktion übergeben wir die bereits in x- und y-Arrays getrennten Daten (*Zeile 4* und *5*). Mit dem Parameter *test_size* legen wir dann noch fest, wie viele Fälle für das Training behalten und wie viele wir für die Evaluation separieren möchten. In diesem Fall legen wir 20 % der Daten für Testzwecke zur Seite, entsprechend fließen 80 % in die Trainingsdaten ein. Insgesamt verfügen wir dann noch über 1070 Fälle zum Anlernen, an den restlichen 268 Fällen können wir unser Modell evaluieren. Mit dem Parameter *random_state* können wir den Zufallsgenerator ansteuern, mit dem die Fälle den Trainings- und Testpartitionen zugewiesen werden. Die *train_test_split*-Funktion zieht nämlich nicht einfach die ersten 1070 Zeilen und weist sie den Trainingsdaten zu. Sie verwendet einen Zufallsgenerator. Wenn wir die Funktion allerdings mehrfach aufrufen, ohne *random_state* zu setzen, würden wir bei jedem Aufruf eine neue zufällige Einteilung in Trainings- und Testdaten erhalten.

> Das ist von Nachteil, zum Beispiel, weil eine günstige oder weniger günstige Zusammensetzung der Test- oder Trainingsdaten unsere Modellevaluation beeinflusst. Wenn wir die Leistung verschiedener Modelle, solche mit mehr oder weniger Features, untereinander vergleichen möchten, sollten wir sicherstellen, dass diese Modelle jeweils auf den gleichen Trainingsdaten angelernt und auf Basis der gleichen Testdaten evaluiert wurden. Dazu setzen wir den Parameter *random_state* auf einen beliebigen Integer-Wert. Die Zufallsziehung wird jetzt nur beim ersten Aufruf durchgeführt. Wird die Funktion danach mit dem gleichen Integer-Wert wieder und wieder aufgerufen, erhalten wir immer die gleiche Stichprobenauswahl wie beim ersten Aufruf. ■

Wir belassen es an dieser Stelle bei dieser einfachen Unterteilung in Trainings- und Testdaten. In der Praxis werden aber manchmal ausgefeiltere Methoden eingesetzt, um Überanpassung an bestimmte Datenauszüge zu verhindern.

Zum Beispiel kann es sinnvoll sein, statt eines einzigen Testauszugs zwei Auszüge zu erzeugen: Validierungs- und Testdaten. Dahinter steckt der Gedanke, dass wir unsere Modelle an den Testdaten abstimmen und damit implizit ein an die Testdaten adaptiertes Modell züchten. Wenn wir die Abstimmung stattdessen an den Evaluationsdaten erledigen, haben wir später immer noch die Möglichkeit, unser fertiges Modell an einem finalen Datensatz, den Testdaten, zu überprüfen. Erst wenn es auf diesen Daten vergleichbare Ergebnisse erzielt, können wir uns sicher sein, dass es effektiv arbeitet.

Gebräuchlich ist auch die sogenannte Kreuzvalidierungsmethode (*Cross-Validation*). Dabei werden Modelle mehrfach an unterschiedlichen Zufallspartitionen der Daten angelernt und jeweils an einer anderen Testpartition überprüft [Provost2013, S. 126 ff.]. Auf diese Weise können verschiedene Modelle mit unterschiedlichen Variablen und Hyperparametern ausprobiert werden. Am Ende wählt man das Modell aus, das über alle Partitionen hinweg die besten Evaluationsergebnisse erzielt. Vorteil ist, dass sich auf diese Weise ausschließen lässt, dass das gewählte Modell nur über eine bestimmte Train-/Testkonstellation funktioniert.

Die Kreuzvalidierung eignet sich aber nur für bestimmte Aufgabenstellungen. Sobald wir mit Algorithmen arbeiten, die lange anlernen, wird das Verfahren schwerfällig. Außerdem können wir aus Zeitreihendaten oft keine Zufallsstichproben ziehen. Zum Beispiel, wenn die Daten in einer bestimmten Ordnung stehen (Zeitintervalle), die beim Anlernen nicht zerstört werden darf. In solchen Fällen haben wir keine andere Wahl, als die Trainingsdaten als zusammenhängenden Datensatz aus dem ersten Teil der Gesamtdaten zu ziehen. Der verbleibende Rest fließt dann in die Testpartition.

Stichwortverzeichnis