Engineer

Maschinelles Lernen

Software Aufgaben anhand von Datenbeispielen lernen lassen

Was es ist

Maschinelles Lernen ist die Praxis, Software zu bauen, die eine Aufgabe aus Beispielen lernt statt aus Regeln, die ein Programmierer ausschreibt. Man gibt ihr viele Fälle mit bereits bekannter Antwort. Sie findet einen Zusammenhang zwischen Eingaben und Antworten und nutzt ihn, um die Antwort für jeden neuen Fall vorherzusagen. Der Name ist älter als fast sämtliche Software, die ihn heute verwendet: A. L. Samuel setzte ihn in den Titel einer Studie über das Damespiel, die das IBM Journal of Research and Development im Juli 1959 veröffentlichte.

Die Begriffe lassen sich am einfachsten anhand von scikit-learn festlegen, der quelloffenen Python-Bibliothek, deren Dokumentation dieser Artikel folgt. Die Beispiele stehen in einer Tabelle. Jede Zeile ist ein Datenpunkt, etwa eine Rechnung, ein Kunde oder ein Sensorwert, und jede Spalte ein Merkmal: eine Zahl, die den Datenpunkt beschreibt. Die Antwort, die die Software lernen soll, ist die Zielgröße. In scikit-learns Begriffen heißt das lernende Objekt Estimator. Jeder Estimator hat eine Methode `fit`, die aus den gezeigten Zeilen lernt. Vorhersagende Estimatoren liefern mit `predict` Antworten für neue Zeilen; Transformer erzeugen mit `transform` eine umgewandelte Tabelle von Datenpunkten.

Zwei Gruppen decken die meiste Arbeit ab. Beim überwachten Lernen hat jede Trainingszeile ihre Zielgröße. Ist diese eine Kategorie, etwa pünktlich oder verspätet bezahlt, heißt die Aufgabe Klassifikation. Ist sie eine numerische Größe, etwa die Zahl der Tage bis zur Zahlung, heißt sie Regression. Beim unüberwachten Lernen gibt es überhaupt keine Zielgröße. Die Software sucht selbst nach Strukturen, etwa indem sie Kunden mit ähnlichem Verhalten gruppiert. Die scikit-learn-Dokumentation widmet jeder Gruppe ein eigenes Kapitel. Danach folgt eines über Modellauswahl und -bewertung, wo die meisten praktischen Schwierigkeiten liegen.

Maschinelles Lernen ist, zumindest in diesem Sinn, kein Wort für jede klug wirkende Software. Eine Regel, die jede Rechnung über einem festen Betrag markiert, wurde von einem Menschen geschrieben und tut genau das, was darin steht. Ein Modell, das Rechnungen markiert, hat seine Regel aus früheren Rechnungen abgeleitet. Es kann deshalb auf Arten irren, die niemand aufgeschrieben hat, und sein Verhalten ändert sich, sobald sich die Daten ändern, aus denen es lernt.

Das Konzept für eine Entscheidung nutzen

Nutzen Sie „Was ist maschinelles Lernen?“ für eine konkrete Entscheidung. Wir können helfen, das Konzept mit dem System in Verbindung zu bringen, das Sie entwickeln oder einkaufen. Ein Agent wird an Ihre Systeme angebunden, an Fällen Ihrer eigenen Fachleute geprüft und von Ihren Mitarbeitenden beaufsichtigt — ohne Bindung an ein einziges Modell.

Unsere Leistungen entdecken Kontakt aufnehmen →

Die Annahmen hinter dem Begriff prüfen

Nutzen Sie die Erklärung und die verlinkten Arbeiten, um Fragen für Ihr Projekt zu formulieren. Wir können die Annahmen gemeinsam mit Ihnen prüfen, bevor ein Werkzeug ausgewählt wird. Repository-Prüfengine mit 268 Checks entwickelt — 268 Commit-Prüfungen und 146 Tests der Prüfungen selbst

Repository-Prüfengine mit 268 Checks entwickelt Kontakt aufnehmen →

Warum es wichtig ist

Manche Aufgaben lassen sich viel leichter anhand von Beispielen zeigen als beschreiben. Niemand kann eine vollständige Regel dafür schreiben, welche Rechnungen verspätet bezahlt werden. Die meisten Unternehmen haben aber Rechnungen aus mehreren Jahren mit dokumentiertem Ausgang. Maschinelles Lernen macht aus dieser Aufzeichnung eine funktionierende Regel und kann das für viele solche Fragen zugleich tun. Deshalb ist es aus der Forschung in gewöhnliche Geschäftssoftware gewandert.

Der Preis ist, dass nun Daten die Arbeit übernehmen, die zuvor der Code tat. Enthält die Historie Fehler, Duplikate oder Lücken, lernt das Modell auch sie, ohne dass im Code etwas davon zu sehen ist. D. Sculley und neun Kollegen bei Google beschrieben die Folgen in *Hidden Technical Debt in Machine Learning Systems*, einem Beitrag im Tagungsband der NIPS-Konferenz 2015. Ihre erste Abbildung zeigt den Lerncode als kleinen schwarzen Kasten in der Mitte eines viel größeren Systems. Die Bildunterschrift nennt die umgebende Infrastruktur gewaltig und komplex. An einer Stelle schätzen sie, dass ein ausgereiftes System am Ende aus höchstens 5 % Code für maschinelles Lernen und mindestens 95 % Verbindungscode bestehen kann, der Daten in allgemeine Programmpakete hinein und aus ihnen heraus bewegt.

Sie argumentieren außerdem, dass eine Datenabhängigkeit mehr kostet als eine Codeabhängigkeit. Compiler und Buildsysteme verfolgen, welcher Code wovon abhängt. Entsprechende Werkzeuge für Daten sind viel seltener. Eine Spalte kann deshalb unbemerkt ihre Bedeutung ändern. Und sie benennen das Prinzip CACE für „Changing Anything Changes Everything“: Wird irgendetwas geändert, ändert sich alles. Verschiebt sich die Verteilung einer Eingabe, kann sich zugleich ändern, wie das Modell jede andere Eingabe verwendet.

Ein Modell ist also günstig zu erstellen und teuer zu erhalten. Ob es hilft, entscheidet vor allem die Datenarbeit: Woher kommen die Daten, erscheint jeder Kunde einmal oder dreimal, ist das dokumentierte Ergebnis richtig, und wer bemerkt, wenn ein vorgelagertes System etwas anderes zu senden beginnt?

Wie es in der Praxis aussieht

Nehmen wir ein Unternehmen, das am Tag der Rechnungsstellung wissen möchte, welche Rechnungen wahrscheinlich verspätet bezahlt werden. So kann jemand die betreffenden Kunden vor statt nach dem Fälligkeitstag anrufen. Ein Export enthält für jede frühere Rechnung eine nach Ausstellungsdatum sortierte Zeile mit vier Spalten, deren Werte am Versandtag bereits bekannt waren: den Betrag, die eingeräumte Zahlungsfrist in Tagen, die Zahl der Rechnungen, die der Kunde in den vorangegangenen zwölf Monaten erhalten hatte, und wie viele davon verspätet bezahlt worden waren. Eine fünfte Spalte hält den Ausgang fest: 1 bei verspäteter und 0 bei pünktlicher Zahlung.

Das folgende Skript wurde für diesen Artikel geschrieben und im Oktober 2026 Zeile für Zeile anhand der Dokumentation von scikit-learn 1.9.1 und NumPy 2.5 geprüft. Es trainiert ein Modell auf den älteren Rechnungen und testet es auf dem neuesten Viertel:

```
# invoices.csv: one row per invoice, sorted by issue date, numbers only.
# The last column is 1 if the invoice was paid late, 0 if on time.
import numpy as np
from sklearn.dummy import DummyClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import balanced_accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler

data = np.loadtxt("invoices.csv", delimiter=",", skiprows=1)
X, y = data[:, :-1], data[:, -1].astype(int)
```

```
# Hold the newest quarter back: nothing learns from it.
X_old, X_new, y_old, y_new = train_test_split(
    X, y, test_size=0.25, shuffle=False
)

model = make_pipeline(StandardScaler(), LogisticRegression())
baseline = DummyClassifier(strategy="most_frequent")
for name, candidate in [("model", model), ("baseline", baseline)]:
    candidate.fit(X_old, y_old)
    score = balanced_accuracy_score(y_new, candidate.predict(X_new))
    print(name, round(score, 3))
```

Vier Entscheidungen darin sind wichtiger als die Wahl des Algorithmus.

Das neueste Viertel wird zurückgehalten, bevor irgendetwas lernt. Mit `shuffle=False` lässt `train_test_split` die Zeilen in ihrer Reihenfolge und legt das letzte Viertel in die Testmenge. Das Modell lernt so aus älteren Rechnungen und wird an neueren beurteilt, genau wie im späteren Einsatz. Mit dem Standard `shuffle=True` wäre die Aufteilung zufällig. Ein Teil der Testrechnungen wäre dann älter als manche Rechnungen, aus denen es gelernt hat.

Die Skalierung liegt innerhalb der Pipeline. `StandardScaler` zieht von jedem Merkmal seinen Mittelwert ab und skaliert es auf Varianz eins. Weil `make_pipeline` ihn mit dem Klassifikator verkettet, stammen Mittelwert und Varianz nur aus den Trainingszeilen. Nichts über das neueste Viertel sickert in das Modell ein.

Der Bewertungswert ist Balanced Accuracy. Er mittelt über die beiden Klassen den Anteil der jeweils richtig erkannten Fälle: die erfassten verspäteten und die erkannten pünktlichen Rechnungen. Wird eine von zehn Rechnungen verspätet bezahlt, belohnt die gewöhnliche Accuracy ein Modell, das jedes Mal „pünktlich“ antwortet. Balanced Accuracy tut das nicht.

Das Modell muss einen Vergleichsmaßstab übertreffen. `DummyClassifier` mit `strategy="most_frequent"` ignoriert die Merkmale und gibt immer die häufigste Antwort. Bei zwei Klassen erreicht er eine Balanced Accuracy von 0,5, weil er jede pünktliche Rechnung richtig und jede verspätete falsch einordnet. Ein Modell, das ihn nicht klar übertrifft, hat nichts Nützliches gelernt, wie gut seine gewöhnliche Accuracy auch aussieht.

Nichts davon rettet ein Modell vor fehlerhaften Datenzeilen. Existiert ein Kunde als drei Datensätze, wird die Zahl seiner verspäteten Rechnungen auf drei Datensätze verteilt. Das Modell lernt dann aus einer Geschichte, die nie stattgefunden hat. Unsere Seite zu KI-Datenbereitschaft behandelt diese Grundlage: ein einziger Datensatz je Kunde, jeder Mensch darf nur sehen, was er sehen soll, und jede Antwort lässt sich auf die zugrunde liegenden Daten zurückführen. Für ein Modell erlaubt Letzteres, von einer seltsamen Vorhersage zu den Zeilen zurückzugehen, aus denen sie entstand.

Wo es schiefgeht

Informationen aus der Zukunft gelangen ins Training. Der scikit-learn-Leitfaden zu häufigen Fallstricken nennt das Data Leakage: Das Modell wird mit Informationen gebaut, die beim Vorhersagen noch nicht existieren werden. Im Rechnungsbeispiel verlockt als zusätzliche Spalte die Zahl versandter Mahnungen. Sie sagt Verspätungen hervorragend voraus, weil Mahnungen erst nach Fälligkeit einer Rechnung versandt werden. Dasselbe Leck entsteht unauffälliger, wenn ein Skalierer oder ein Imputer vor der Aufteilung auf allen Zeilen angepasst wird. Die Regel des Leitfadens lautet, nie `fit` auf Testdaten aufzurufen. Halten Sie für jede Spalte fest, wann ihr Wert bekannt wird, teilen Sie zuerst auf und lassen Sie jeden Vorverarbeitungsschritt in der Pipeline, damit er nur an Trainingszeilen angepasst wird.

Das Modell wird an bereits gesehenen Daten beurteilt. Ein Modell, das nur seine erlernten Zielgrößen wiederholt, würde auf den eigenen Trainingszeilen perfekt abschneiden und bei allem Neuen nutzlos sein. Der Leitfaden zur Kreuzvalidierung nennt das Überanpassung. Subtiler ist es, viele Einstellungen auszuprobieren und die mit dem besten Ergebnis auf den zurückgehaltenen Zeilen zu wählen: So fließt Wissen über diese Zeilen in die Auswahl ein. Vergleichen Sie Einstellungen durch Kreuzvalidierung auf dem Trainingsanteil. Dabei werden die Zeilen in mehrere Teilmengen, sogenannte Folds, zerlegt, von denen jede abwechselnd als Testmenge dient. Sehen Sie sich die zurückgehaltenen Zeilen nur einmal an, am Ende.

Eine hohe Genauigkeit, die eine Konstante ebenso erreichen könnte. Ist eine Klasse selten, dominiert die häufige Klasse die Accuracy. Der Leitfaden warnt, dass bei seltenen Klassen, etwa in der Betrugserkennung, eine Teilmenge ganz ohne Beispiel der seltenen Klasse enden kann. Berichten Sie Balanced Accuracy oder ein anderes klassenbezogenes Maß, vergleichen Sie mit einem `DummyClassifier` und verwenden Sie stratifizierte Teilmengen. Sie halten die

Klassenanteile in jeder Teilmenge ungefähr gleich. `cross_val_score` verwendet sie für einen Klassifikator standardmäßig, wenn eine Zahl von Teilmengen übergeben wird.

Eine zufällige Aufteilung für Daten, die sich über die Zeit verändern. Der Leitfaden zur Kreuzvalidierung merkt an, dass die Annahme unabhängiger, identisch verteilter Stichproben in der Praxis selten zutrifft. Zeitlich geordnete Daten sollten an Beobachtungen getestet werden, die später liegen als die zum Lernen verwendeten. Eine zufällige Aufteilung lässt das Modell in die Zukunft sehen und meldet einen Wert, den es nicht erreichen wird. Teilen Sie wie im Beispiel nach Datum auf. Verwenden Sie bei der Kreuzvalidierung ein zeitabhängiges Aufteilungsverfahren wie `TimeSeriesSplit`, dessen Teilmengen dem Leitfaden zufolge jeweils einen gleich langen Zeitraum abdecken sollten.

Die Vorverarbeitung unterscheidet sich zwischen Training und Produktion. Hat das Modell aus skalierten Merkmalen gelernt und erhält im laufenden System Rohwerte, sind die Vorhersagen falsch, ohne dass ein Fehler gemeldet wird. Der Fallstrick-Leitfaden verlangt, jede im Training verwendete Transformation genauso auf Testdaten und Produktionsdaten anzuwenden. Liefern Sie die angepasste Pipeline als ein Objekt aus, statt ein Modell mit einer Notiz darüber, was vorher mit den Daten zu tun ist.

Die Welt ändert sich nach dem Start. Ein auf den Vorjahresrechnungen trainiertes Modell nimmt an, dass Kunden sich noch wie damals verhalten. Sculley und seine Mitautoren empfehlen, die Vorhersageverzerrung zu beobachten: In einem wie beabsichtigt funktionierenden System sollte der Anteil jeder vorhergesagten Klasse gewöhnlich dem tatsächlich beobachteten Anteil entsprechen. Eine sich öffnende Lücke ist oft das erste Zeichen, dass sich die Welt verändert hat. Führen Sie diesen Vergleich fort, trainieren Sie mit aktuellen Daten neu und halten Sie eine eingefrorene, versionierte Kopie jedes Eingabesignals vor, das einem anderen Team gehört. Eine gut gemeinte vorgelagerte Änderung verändert dann nicht unbemerkt Ihr Modell.

Weiterführende Lektüre

- Benutzerhandbuch — scikit-learns Referenz für überwachtes und unüberwachtes Lernen sowie Modellauswahl und -bewertung, Kapitel für Kapitel.
- Erste Schritte — die Estimator-Schnittstelle in kurzen Beispielen: Anpassen und Vorhersagen, Transformer, Pipelines und Kreuzvalidierung.

- Häufige Fallstricke und empfohlene Vorgehensweisen — abweichende Vorverarbeitung, Data Leakage und Zufälligkeit, mit falschem und richtigem Vorgehen nebeneinander im Code.
- Kreuzvalidierung: Estimator-Leistung bewerten — warum Tests auf Trainingsdaten täuschen und welche Aufteilung für gruppierte, unausgewogene oder zeitlich geordnete Daten passt.
- Verborgene technische Schulden in Systemen für maschinelles Lernen — Sculley und Kollegen bei Google, NIPS 2015, über die Betriebskosten eines Modells jenseits des Modells selbst.

Glossar

Für Ihre tatsächlichen Rahmenbedingungen auswählen

Beschreiben Sie den Arbeitsablauf, die vorhandenen Schnittstellen und die Nutzer, die die Veränderung benötigen. Wir können eine prüfbare Lieferung definieren und die Übergabe so gestalten, dass Ihr Team die gelieferten Bestandteile warten kann.

Preise ansehen Kontakt aufnehmen →

Die Begriffe praktisch nutzbar machen

Fragen Sie, wie „Was ist maschinelles Lernen?“ auf Ihre Organisation zutrifft. Ein konkretes Beispiel aus Ihrem bestehenden System reicht aus, um das Gespräch zu beginnen.

Unsere Leistungen entdecken

welcome@engineer.company +45 50 10 61 84 Kopenhagen, Dänemark

Nehmen Sie diese Seite mit

Kontaktkarte

Mit Sorgfalt und Liebe gestaltet von D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Copyright © 2025 – heute · Engineer ApS · Alle Rechte vorbehalten.

Gedruckt von https://engineer.company/de/glossary/machine-learning/