

Engineer

Maskininlärning

att bygga program som lär sig en uppgift av exempel i data



Vad det är

Maskininlärning innebär att bygga program som lär sig en uppgift av exempel, i stället för att följa regler som en utvecklare har skrivit ner i förväg. Man matar in många fall där svaret redan är känt. Programmet hittar ett samband mellan det som gick in och svaret som kom ut, och använder sedan sambandet för att förutsäga svaret i varje nytt fall. Begreppet är äldre än nästan all mjukvara som använder det i dag: A. L. Samuel tog in det i titeln på en studie av brädspelet dam, som IBM Journal of Research and Development publicerade i juli 1959.

Orden blir enklast att reda ut med hjälp av scikit-learn, ett Python-bibliotek med öppen källkod, och det är dess dokumentation som den här artikeln följer. Exemplen ligger i en tabell. Varje rad är en observation (scikit-learn säger sample), till exempel en faktura, en kund eller ett mätvärde från en sensor. Varje kolumn är en feature, ett tal som beskriver observationen. Svaret som programmet ska lära sig att ge kallas målvariabel. Det som lär sig heter i scikit-learn en estimator. Varje estimator har en `fit`-metod som lär sig av raderna man visar den. Prediktiva estimatorer använder `predict` för att ge svar för nya rader; transformerare använder `transform` för att skapa en omvandlad tabell med observationer.

Det mesta arbetet hamnar i en av två grupper. Vid övervakad inlärning har varje träningsrad sin målvariabel med sig. När målvariabeln är en kategori, som betald i tid eller betald för sent, kallas uppgiften klassificering. När den är en storhet, som antal dagar till betalning, kallas den regression. Vid oövervakad inlärning finns ingen målvariabel alls, och programmet letar själv efter struktur, exempelvis genom att föra samman kunder som beter sig likadant i grupper. I dokumentationen för scikit-learn har båda grupperna ett eget kapitel, och efter dem kommer ett kapitel om modellval och utvärdering. Det är där de flesta praktiska svårigheterna finns.

Däremot är maskininlärning inte ett samlingsnamn för all mjukvara som verkar smart. En regel som flaggar varje faktura över ett visst belopp har skrivits av en människa och gör exakt det som står i den. En modell som flaggar fakturor har själv härlett sin regel ur gamla fakturor. Den kan därför ha fel på sätt som ingen har skrivit ner, och den beter sig annorlunda så fort de data den lär sig av förändras.

Använd begreppet i ett beslut

Använd ”Vad är maskininlärning?” i ett konkret beslut. Vi kan knyta begreppet till systemet ni bygger eller köper. En agent kopplad till de system du redan använder, testad på fall som dina egna experter skriver, övervakad av dina medarbetare och aldrig låst till en enda modell.

Se våra tjänster Hör av er →

Granska antagandena bakom ordet

Använd förklaringen och det länkade arbetet för att formulera frågor till projektet. Vi kan undersöka antagandena tillsammans innan ett verktyg väljs. Byggde repots kontrollmotor med 268 kontroller — 268 commit-kontroller, och 146 tester av själva kontrollerna

Byggde repots kontrollmotor med 268 kontroller Hör av er →

Varför det spelar roll

Vissa uppgifter är mycket lättare att visa än att beskriva. Ingen kan skriva en fullständig regel för vilka fakturor som kommer att betalas för sent, men de flesta företag har flera års fakturor där utfallet finns registrerat. Maskininlärning gör om den historiken till en regel som fungerar, och kan göra det för många liknande frågor på en gång. Därför har tekniken gått från forskningen till helt vanliga affärssystem.

Baksidan är att data nu gör det arbete som koden brukade göra. Innehåller historiken fel, dubbletter eller luckor lär sig modellen dem också, och ingenting i koden avslöjar det. D. Sculley och nio kolleger på Google beskrev följderna i *Hidden Technical Debt in Machine Learning Systems*, en artikel i konferensvolymen från NIPS 2015, och deras ämne är teknisk skuld. Artikelns första figur ritar själva inlärningskoden som en liten svart låda mitt i ett mycket större system, och enligt bildtexten är infrastrukturen runt den både omfattande och komplex. På ett ställe bedömer författarna att ett moget system kan sluta med högst 5 procent maskininlärningskod och minst 95 procent limkod, vars enda uppgift är att flytta data in i och ut ur generella programpaket.

De hävdar också att ett beroende av data kostar mer än ett beroende av kod. Kompilatorer och byggsystem kan analysera vilken kod som beror på vilken, men motsvarande verktyg för data är betydligt ovanligare. En kolumn kan alltså få en ny betydelse utan att något märker det. Författarna ger dessutom en princip ett

namn, CACE, för ”Changing Anything Changes Everything”: ändras fördelningen för en enda indata kan modellens sätt att använda alla andra indata ändras med den.

En modell är med andra ord billig att bygga och dyr att hålla vid liv. Det som avgör om den gör nytta handlar mest om data: var de kommer ifrån, om varje kund finns en eller tre gånger, om det registrerade utfallet ens stämmer och vem som märker det när ett system längre upp i kedjan börjar skicka något annat än förut.

Så ser det ut i praktiken

Tänk dig ett företag som vill veta redan samma dag som en faktura skickas ut om den troligen kommer att betalas för sent. Då kan någon ringa kunden före förfallodagen och inte först efteråt. En export har en rad per tidigare faktura, sorterad efter fakturadatum, och fyra kolumner som alla var kända den dag fakturan gick iväg: beloppet, antalet kreditdagar, hur många fakturor kunden fick året innan och hur många av dem som betalades för sent. En femte kolumn visar hur det gick: 1 om fakturan betalades för sent, 0 om den betalades i tid.

Skriptet nedan är skrivet för den här artikeln och kontrollerat rad för rad mot dokumentationen för scikit-learn 1.9.1 och NumPy 2.5 i oktober 2026. Det tränar en modell på de äldsta tre fjärdedelarna av fakturorna och testar den på den nyaste fjärdedelen:

```
# invoices.csv: en rad per faktura sorterad efter datum, bara tal.
# Sista kolumnen är 1 om fakturan betalades för sent, annars 0.
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)

# Håll den nyaste fjärdedelen utanför: inget lär sig av den.
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))
```

Fyra val i skriptet spelar större roll än valet av algoritm.

Den nyaste fjärdedelen läggs undan innan något lär sig. Med `shuffle=False` behåller `train_test_split` ordningen och lägger de sista raderna i testmängden. Modellen lär sig alltså av äldre fakturor och bedöms på nyare, precis så som den ska användas. Med standardvärdet `shuffle=True` skulle uppdelningen bli slumpmässig, och modellen skulle delvis testas på fakturor som är äldre än några av dem den lärde sig av.

Skalningen sitter inne i pipelinen. `StandardScaler` drar bort medelvärdet för varje feature och skalar den till enhetsvarians. Eftersom `make_pipeline` kopplar ihop skalningen med klassificeraren räknas medelvärde och varians enbart fram ur träningsraderna, och ingenting från den nyaste fjärdedelen läcker in i modellen.

Måttet är balanserad träffsäkerhet. För var och en av de två klasserna räknas andelen som modellen fick rätt, alltså de sena fakturor den fångade och de punktliga den kände igen, och sedan tas medelvärdet. Om var tionde faktura betalas för sent belönar vanlig träffsäkerhet en modell som svarar ”i tid” varenda gång. Det gör inte balanserad träffsäkerhet.

Modellen måste slå en referensmodell. `DummyClassifier` med `strategy="most_frequent"` bortser från alla features och svarar alltid med den vanligaste klassen. Mätt i balanserad träffsäkerhet får den 0,5 med två klasser, eftersom den prickar varje punktlig faktura och missar varje sen. En modell som inte tydligt slår den har inte lärt sig något användbart, hur fin dess vanliga träffsäkerhet än ser ut.

Inget av detta räddar en modell från dåliga rader. Finns samma kund som tre olika poster delas antalet sena fakturor för kunden upp på tre, och modellen lär sig av en historik som aldrig har ägt rum. Vår sida om Dataförberedelse för AI handlar om just det grundarbetet: att varje kund bara finns en gång, att varje medarbetare bara kommer åt det hen ska se och att man kan härleda ett svar

till de data det bygger på. För en modell är den sista punkten det som gör att man kan gå baklänges från en konstig förutsägelse till raderna som gav upphov till den.

Var det går fel

Information från framtiden kommer med i träningen. Guiden till vanliga fallgropar i scikit-learn kallar detta dataläckage (data leakage): att modellen byggs med information som inte kommer att finnas när den ska göra en förutsägelse. I fakturaexemplet är den lockande extra kolumnen antalet skickade betalningspåminnelser. Den förutsäger förseningar utmärkt, just för att påminnelser bara skickas när en faktura redan har förfallit. Samma läcka uppstår mer obemärkt när en skalning eller en ifyllnad av saknade värden anpassas på alla rader före uppdelningen, och guidens regel är att aldrig anropa `fit` på testdata. Lösningen är att skriva upp när värdet i varje kolumn blir känt, att dela upp data först och att lägga all förbehandling inne i pipelinen, så att den bara anpassas på träningsraderna.

Modellen bedöms på något den redan har sett. En modell som bara upprepade de svar den tränats på skulle få full poäng på sina egna träningsrader och vara oanvändbar på allt nytt. Guiden om korsvalidering kallar detta överanpassning (overfitting). En lömskare variant är att prova många inställningar och behålla den som får bäst resultat på de undanhållna raderna, för då läcker kunskap om just de raderna in i valet. Jämför i stället inställningar med korsvalidering på träningsdelen, där raderna delas in i flera veck och varje veck i tur och ordning får vara testmängd. De undanhållna raderna tittar man på en enda gång, allra sist.

Hög träffsäkerhet som en konstant skulle klara. När den ena klassen är ovanlig domineras träffsäkerheten av den vanliga. Guiden varnar för att ett veck vid ovanliga klasser, som vid upptäckt av bedrägerier, kan sakna ett enda exempel på den ovanliga klassen. Redovisa balanserad träffsäkerhet eller något annat mått per klass, jämför med en `DummyClassifier` och använd stratifierade veck, som håller klassernas andelar ungefär lika i varje veck. `cross_val_score` använder dem som standard för en klassificerare när man anger ett antal veck.

Slumpmässig uppdelning av data som förändras över tid. Guiden om korsvalidering påpekar att antagandet om oberoende, likafördelade observationer sällan håller i praktiken, och att tidsordnade data bör testas på observationer som ligger senare än dem modellen lärde sig av. En slumpmässig uppdelning låter modellen kika in i framtiden och ger ett resultat den aldrig når i

drift. Dela upp efter datum, som exemplet gör, och använd vid korsvalidering en uppdelning som tar hänsyn till tiden, till exempel `TimeSeriesSplit`, där varje veck enligt guiden bör täcka lika lång tid.

Förbehandlingen skiljer sig mellan träning och drift. Om modellen lärde sig av skalade features och det körande systemet skickar råa tal blir förutsägelserna fel, utan att något felmeddelande dyker upp. Guiden om fallgropar kräver att varje transformation som används vid träningen också tillämpas på testdata och på data i produktion. Leverera den anpassade pipelinen som ett enda objekt, inte som en modell plus en anvisning om vad som först ska göras med data.

Världen förändras efter att modellen har tagits i drift. En modell som tränats på förra årets fakturor utgår från att kunderna fortfarande beter sig som då. Sculley och hans medförfattare föreslår att man följer prediktionsbias (prediction bias): i ett system som fungerar som tänkt bör fördelningen av förutsagda kategorier i regel motsvara fördelningen av dem som faktiskt observeras, och ett glapp som växer fram mellan dem är ofta det första tecknet på att världen har förändrats. Håll den jämförelsen igång, träna om på färskare data och behåll en fryst, versionerad kopia av varje indatasignal som ett annat team äger, så att en välmenande ändring längre upp i kedjan inte i tysthet ändrar er modell.

Läs mer

- User Guide — referensverket för scikit-learn om övervakad och oövervakad inlärning och om modellval och utvärdering, kapitel för kapitel.
- Getting Started — estimatorernas gemensamma gränssnitt i korta exempel: anpassning och förutsägelse, transformatorer, sammankopplade steg och korsvalidering.
- Common pitfalls and recommended practices — inkonsekvent förbehandling, dataläckage och slump, med fel och rätt sätt sida vid sida i kod.
- Cross-validation: evaluating estimator performance — varför test på träningsdata vilseleder, och vilken uppdelning som passar grupperade, obalanserade eller tidsordnade data.
- Hidden Technical Debt in Machine Learning Systems — Sculley och kolleger på Google, NIPS 2015, om vad det kostar att driva en modell i produktion utöver själva modellen.

Ordlista

Välj utifrån era faktiska förutsättningar

Beskriv arbetsflödet, befintliga gränssnitt och användarna som behöver förändringen. Vi kan definiera en testbar leverans och en överlämning som gör att teamet kan underhålla lösningen.

Se vad det kostar Hör av er →

Gör begreppen praktiska

Fråga hur ”Vad är maskininlärning?” gäller i er organisation. Ett konkret exempel från ert nuvarande system räcker för att börja samtalet.

Se våra tjänster

welcome@engineer.company +45 50 10 61 84 Köpenhamn, Danmark

Ta med dig den här sidan

Visitkort

Omsorgsfullt skapad med kärlek av D.

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

Upphovsrätt © 2025 – nu · Engineer ApS · Alla rättigheter förbehållna.

Utskrivet från https://engineer.company/sv/glossary/machine-learning/