Engineer

Maskinlæring

at bygge software, der lærer en opgave af eksempler i data

Hvad det er

Maskinlæring betyder at bygge software, der lærer en opgave af eksempler i stedet for at følge regler, som en programmør har formuleret på forhånd. Man viser den mange tilfælde, hvor svaret allerede er kendt. Den finder en sammenhæng mellem det, der gik ind, og det svar, der kom ud, og bruger derefter sammenhængen til at forudsige svaret i hvert nyt tilfælde. Betegnelsen er ældre end næsten al den software, der bærer den i dag: A. L. Samuel brugte den i titlen på en undersøgelse af brætspillet dam, som IBM Journal of Research and Development udgav i juli 1959.

Begreberne er nemmest at holde fast i med scikit-learn, et open source-bibliotek til Python, og det er dets dokumentation, denne artikel bygger på. Eksemplerne ligger i en tabel. Hver række er en observation (scikit-learn siger sample), for eksempel en faktura, en kunde eller en måling fra en sensor. Hver kolonne er en feature, altså et tal, der beskriver observationen. Det svar, softwaren skal lære at give, er målvariablen. Det, der lærer, kalder scikit-learn en estimator. Alle estimatorer har en `fit`-metode, som lærer af de rækker, man viser den. Prædiktive estimatorer bruger `predict` til at give svar på nye rækker; transformere bruger `transform` til at danne en transformeret tabel med observationer.

Langt det meste arbejde falder i to grupper. Ved superviseret læring har hver træningsrække sin målvariabel med. Er målvariablen en kategori, for eksempel betalt til tiden eller betalt for sent, kaldes opgaven klassifikation. Er den en størrelse, for eksempel antal dage til betaling, hedder det regression. Ved ikke-superviseret læring findes der ingen målvariabel, og softwaren leder selv efter struktur i data, typisk ved at samle kunder, der opfører sig ens, i grupper. I scikit-learns dokumentation har hver af de to grupper sit eget kapitel, og bagefter kommer et kapitel om modelvalg og evaluering. Det er dér, de fleste praktiske vanskeligheder gemmer sig.

Maskinlæring er derimod ikke et samlebegreb for al software, der virker klog. Når en regel markerer hver faktura over et bestemt beløb, har et menneske skrevet den, og den gør hverken mere eller mindre, end der står. En model, der markerer fakturaer, har selv udledt sin regel af gamle fakturaer. Den kan derfor tage fejl på måder, som ingen har skrevet ned, og den skifter adfærd, så snart de data, den lærer af, skifter.

Brug begrebet i en beslutning

Brug »Hvad er maskinlæring?« i en konkret beslutning. Vi kan knytte begrebet til det system, I bygger eller køber. En agent koblet til de systemer, I bruger, testet på sager, jeres egne eksperter skriver, under opsyn af jeres folk og aldrig låst til én model.

Se vores ydelser Kontakt os →

Undersøg antagelserne bag ordet

Brug forklaringen og det tilknyttede arbejde til at formulere spørgsmål til jeres projekt. Vi kan undersøge antagelserne sammen, før et værktøj vælges. Har bygget en kontrolmotor med 268 tjek — 268 commit-tjek og 146 tests af selve tjekkene

Har bygget en kontrolmotor med 268 tjek Kontakt os →

Hvorfor det betyder noget

Visse opgaver er langt lettere at vise end at forklare. Ingen kan skrive en udtømmende regel for, hvilke fakturaer der ender med at blive betalt for sent, men de fleste virksomheder ligger inde med flere års fakturaer, hvor det faktiske udfald er registreret. Maskinlæring forvandler den historik til en regel, der kan bruges, og kan gøre det for mange lignende spørgsmål på én gang. Derfor er teknikken gået fra forskningslaboratoriet til helt almindelig forretningssoftware.

Bagsiden er, at data nu gør det arbejde, som koden før gjorde. Er der fejl, dubletter eller huller i historikken, lærer modellen dem også, og der er intet i koden, der røber det. D. Sculley og ni kolleger fra Google beskrev følgerne i *Hidden Technical Debt in Machine Learning Systems*, en artikel i proceedings fra NIPS-konferencen i 2015, og deres emne er teknisk gæld. Artiklens første figur tegner selve læringskoden som en lille sort boks midt i et meget større system, og ifølge billedteksten er infrastrukturen omkring den både enorm og kompleks. Et sted vurderer forfatterne, at et modent system kan ende med at bestå af højst 5 procent maskinlæringskode og mindst 95 procent limkode, hvis eneste opgave er at flytte data ind i og ud af generelle softwarepakker.

De påpeger også, at det er dyrere at afhænge af data end af kode. Compilere og byggesystemer kan analysere, hvilken kode der afhænger af hvilken, men tilsvarende værktøjer til data er langt sjældnere. En kolonne kan derfor få en ny betydning, uden at noget som helst opdager det. Forfatterne giver også et

princip et navn, CACE, som står for »Changing Anything Changes Everything«: ændres fordelingen af ét input, kan modellens måde at bruge alle de andre input på ændre sig med.

En model er med andre ord billig at bygge og dyr at holde kørende. Det, der afgør, om den gør nytte, handler for det meste om data: hvor de stammer fra, om hver kunde står der én eller tre gange, om det registrerede udfald overhovedet er korrekt, og hvem der opdager det, når et system længere oppe i kæden begynder at levere noget andet end før.

Sådan ser det ud i praksis

Forestil dig en virksomhed, der vil vide allerede samme dag, som en faktura bliver udstedt, om den sandsynligvis bliver betalt for sent. Så kan en medarbejder ringe til kunden før forfaldsdagen og ikke først bagefter. Et udtræk har én række pr. tidligere faktura, sorteret efter udstedelsesdato, og fire kolonner, som alle var kendt den dag, fakturaen blev sendt: beløbet, antallet af kreditdage, hvor mange fakturaer kunden fik i det foregående år, og hvor mange af dem der blev betalt for sent. En femte kolonne fortæller, hvordan det gik: 1, hvis fakturaen blev betalt for sent, og 0, hvis den blev betalt til tiden.

Scriptet herunder er skrevet til denne artikel og gennemgået linje for linje mod dokumentationen til scikit-learn 1.9.1 og NumPy 2.5 i oktober 2026. Det træner en model på de ældste tre fjerdedele af fakturaerne og tester den på den nyeste fjerdedel:

```
# invoices.csv: én række pr. faktura sorteret efter dato, kun tal.
# Sidste kolonne er 1, hvis fakturaen blev betalt for sent, ellers 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)

# Hold den nyeste fjerdedel tilbage: intet lærer af 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))
```

Fire beslutninger i scriptet betyder mere end valget af algoritme.

Den nyeste fjerdedel lægges til side, før noget lærer. Med `shuffle=False` bevarer `train_test_split` rækkefølgen og lægger de sidste rækker i testsættet. Modellen lærer altså af ældre fakturaer og bliver bedømt på nyere, præcis som den skal bruges i virkeligheden. Lod man standardværdien `shuffle=True` stå, ville opdelingen blive tilfældig, og modellen ville delvis blive testet på fakturaer, der er ældre end nogle af dem, den har lært af.

Skaleringen sidder inde i pipelinen. `StandardScaler` trækker hver features middelværdi fra og skalerer den til enhedsvarians. Fordi `make_pipeline` kæder skaleringen sammen med klassifikatoren, beregnes middelværdi og varians alene ud fra træningsrækkerne, og intet fra den nyeste fjerdedel siver ind i modellen.

Målet er balanceret nøjagtighed. For hver af de to klasser regnes den andel ud, som modellen ramte rigtigt, altså de sene fakturaer, den fangede, og de rettidige, den genkendte, og så tages gennemsnittet. Bliver hver tiende faktura betalt for sent, belønner almindelig nøjagtighed en model, der svarer »til tiden« hver eneste gang. Det gør balanceret nøjagtighed ikke.

Modellen skal slå en referencemodel. `DummyClassifier` med `strategy="most_frequent"` ser bort fra alle features og svarer altid med den hyppigste klasse. Målt på balanceret nøjagtighed får den 0,5, når der er to klasser, for den rammer hver rettidig faktura og misser hver sen. Slår modellen den ikke tydeligt, har den ikke lært noget brugbart, uanset hvor pæn dens almindelige nøjagtighed ser ud.

Intet af det redder en model fra dårlige rækker. Står den samme kunde som tre forskellige poster, bliver antallet af kundens sene fakturaer fordelt på tre, og modellen lærer af en historik, der aldrig har fundet sted. Vores side om Klargøring af data til AI handler om netop det forarbejde: at hver kunde kun findes én gang, at den enkelte medarbejder kun får adgang til det,

vedkommende skal se, og at ethvert svar kan følges tilbage til de data, det bygger på. For en model er det sidste punkt det, der gør det muligt at gå baglæns fra en underlig forudsigelse til de rækker, der gav den.

Hvor det går galt

Oplysninger fra fremtiden sniger sig ind i træningen. Vejledningen om almindelige faldgruber i scikit-learn kalder det datalækage (data leakage): modellen bygges med oplysninger, som ikke vil findes, når den skal forudsige. I fakturaeksemplet er den fristende ekstra kolonne antallet af udsendte rykkere. Den forudsiger forsinkelse fremragende, fordi rykkere først sendes, når en faktura allerede er forfalden. Samme læk opstår mere ubemærket, når en skalering eller en udfyldning af manglende værdier tilpasses på samtlige rækker før opdelingen, og vejledningens regel er, at man aldrig kalder `fit` på testdata. Løsningen er at skrive ned, hvornår hver kolonnes værdi bliver kendt, at dele data først og at lade al forbehandling ligge i pipelinen, så den kun tilpasses på træningsrækkerne.

Modellen bedømmes på noget, den allerede har set. En model, der bare gentog de svar, den var trænet på, ville få fuld score på sine egne træningsrækker og være værdiløs over for alt nyt. Vejledningen om krydsvalidering kalder det overtilpasning (overfitting). En mere lumsk variant er at afprøve mange indstillinger og beholde den, der scorer bedst på de tilbageholdte rækker, for så siver viden om netop de rækker ind i valget. Sammenlign i stedet indstillinger med krydsvalidering på træningsdelen, hvor rækkerne skæres i flere fold, og hver fold på skift er testsæt. De tilbageholdte rækker ser man på én gang, til allersidst.

Høj nøjagtighed, som en konstant kunne matche. Når den ene klasse er sjælden, bliver nøjagtigheden domineret af den almindelige. Vejledningen advarer om, at en fold ved sjældne klasser, som ved afsløring af svindel, kan ende uden et eneste eksempel på den sjældne klasse. Rapportér balanceret nøjagtighed eller et andet mål pr. klasse, sammenlign med en `DummyClassifier`, og brug stratificerede fold, der holder klassernes andele nogenlunde ens i hver fold. `cross_val_score` bruger dem som standard for en klassifikator, når man giver den et antal fold.

En tilfældig opdeling af data, der ændrer sig over tid. Vejledningen om krydsvalidering bemærker, at antagelsen om uafhængige og identisk fordelte observationer sjældent holder i praksis, og at tidsordnede data bør testes på

observationer, der ligger senere end dem, modellen har lært af. En tilfældig opdeling lader modellen kigge ind i fremtiden og giver en score, den aldrig når i drift. Del efter dato, som eksemplet gør, og brug ved krydsvalidering en opdeling, der tager hensyn til tiden, for eksempel `TimeSeriesSplit`, hvis fold ifølge vejledningen hver bør dække et lige så langt tidsrum.

Forbehandlingen er ikke den samme i træning og drift. Har modellen lært af skalerede features, og sender det kørende system rå tal, bliver forudsigelserne forkerte, uden at der kommer nogen fejlmeddelelse. Vejledningen om faldgruber kræver, at enhver transformation, der bruges under træningen, også anvendes på testdata og på data i produktion. Lever den tilpassede pipeline som ét samlet objekt frem for en model med en vejledning i, hvad der først skal gøres ved data.

Verden ændrer sig, efter modellen er sat i drift. En model, der er trænet på sidste års fakturaer, går ud fra, at kunderne stadig opfører sig som dengang. Sculley og hans medforfattere foreslår at holde øje med forudsigelsesbias (prediction bias): i et system, der fungerer efter hensigten, bør fordelingen af de forudsagte kategorier normalt svare til fordelingen af dem, der faktisk observeres, og et voksende gab mellem de to er ofte det første tegn på, at verden har flyttet sig. Lad den sammenligning køre fast, træn igen på nyere data, og hold en fastfrosset, versioneret kopi af hvert inputsignal, som et andet team ejer, så en velment ændring længere oppe i kæden ikke i al stilhed ændrer jeres model.

Læs mere

- User Guide — scikit-learns opslagsværk om superviseret og ikke-superviseret læring og om modelvalg og evaluering, kapitel for kapitel.
- Getting Started — estimatorernes fælles grænseflade vist i korte eksempler: tilpasning og forudsigelse, transformere, sammenkædede trin og krydsvalidering.
- Common pitfalls and recommended practices — inkonsekvent forbehandling, datalækage og tilfældighed, med den forkerte og den rigtige fremgangsmåde side om side i kode.
- Cross-validation: evaluating estimator performance — hvorfor en test på træningsdata vildleder, og hvilken opdeling der passer til grupperede, skævt fordelte eller tidsordnede data.
- Hidden Technical Debt in Machine Learning Systems — Sculley og kolleger fra Google, NIPS 2015, om hvad det koster at drive en model i produktion ud over selve modellen.

Ordliste

Vælg ud fra jeres faktiske vilkår

Beskriv arbejdsgangen, de eksisterende grænseflader og de brugere, der har brug for ændringen. Vi kan definere en testbar leverance og en overdragelse, der gør teamet i stand til at vedligeholde løsningen.

Se, hvad det koster Kontakt os →

Gør begreberne praktiske

Spørg, hvordan »Hvad er maskinlæring?« gælder i jeres organisation. Et konkret eksempel fra jeres nuværende system er nok til at begynde samtalen.

Se vores ydelser

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

Tag denne side med dig

Visitkort

Lavet med kærlighed af D.

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

Copyright © 2025 – nu · Engineer ApS · Alle rettigheder forbeholdes.

Udskrevet fra https://engineer.company/da/glossary/machine-learning/