Engineer

Maskinlæring

å bygge programvare som lærer en oppgave av eksempler i data

Hva det er

Maskinlæring vil si å lage programvare som lærer en oppgave av eksempler, i stedet for å følge regler en utvikler har skrevet på forhånd. Du gir den mange tilfeller der svaret allerede er kjent. Den finner en sammenheng mellom det som gikk inn og svaret som kom ut, og bruker så denne sammenhengen til å forutsi svaret i hvert nytt tilfelle. Uttrykket er eldre enn nesten all programvaren som bruker det i dag: A. L. Samuel satte det i tittelen på en studie av damspill, som IBM Journal of Research and Development trykket i juli 1959.

Det er lettest å få begrepene på plass med scikit-learn, et Python-bibliotek med åpen kildekode, og det er dokumentasjonen til scikit-learn denne artikkelen følger. Eksemplene står i en tabell. Hver rad er én observasjon (scikit-learn kaller det en sample), for eksempel en faktura, en kunde eller en måling fra en sensor. Hver kolonne er en feature, et tall som beskriver observasjonen. Svaret programvaren skal lære å gi, er målvariabelen. Det som lærer, heter en estimator hos scikit-learn. Alle estimatorer har en `fit` -metode som lærer av radene du viser den. Prediktive estimatorer bruker `predict` til å gi svar for nye rader; transformatorer bruker `transform` til å lage en transformert tabell med observasjoner.

Det aller meste av arbeidet faller i to kategorier. Ved veiledet læring har hver treningsrad målvariabelen sin med seg. Når målvariabelen er en kategori, som betalt i tide eller betalt for sent, er oppgaven klassifisering. Når den er en mengde, som antall dager til betaling, er det regresjon. Ved ikke-veiledet læring finnes det ingen målvariabel, og programvaren leter selv etter struktur, for eksempel ved å samle kunder som oppfører seg likt, i grupper. Dokumentasjonen til scikit-learn gir hver av de to kategoriene et eget kapittel og lar dem etterfølges av et kapittel om modellvalg og evaluering. Det er der de fleste praktiske problemene ligger.

Maskinlæring er derimot ikke et navn på all programvare som virker smart. En regel som flagger hver faktura over et fast beløp, er skrevet av et menneske og gjør nøyaktig det som står i den. En modell som flagger fakturaer, har selv utledet regelen sin fra gamle fakturaer. Den kan derfor ta feil på måter ingen har skrevet ned, og den endrer oppførsel hver gang dataene den lærer av, endrer seg.

Bruk begrepet i en beslutning

Bruk «Hva er maskinlæring?» i en konkret beslutning. Vi kan knytte begrepet til systemet dere bygger eller kjøper. En agent koblet til systemene du allerede bruker, testet på saker dine egne fagfolk skriver, under oppsyn av folkene dine og aldri låst til én modell.

Se tjenestene våre Ta kontakt →

Undersøk antakelsene bak ordet

Bruk forklaringen og det tilknyttede arbeidet til å formulere spørsmål for prosjektet. Vi kan undersøke antakelsene sammen før et verktøy velges. Bygde repoets kontrollmotor med 268 kontroller — 268 commit-kontroller, og 146 tester av selve kontrollene

Bygde repoets kontrollmotor med 268 kontroller Ta kontakt →

Hvorfor det er viktig

Noen oppgaver er mye lettere å vise enn å forklare. Ingen klarer å skrive en fullstendig regel for hvilke fakturaer som kommer til å bli betalt for sent, men de fleste bedrifter sitter på flere år med fakturaer der utfallet er registrert. Maskinlæring gjør den historikken om til en regel som virker, og kan gjøre det for mange slike spørsmål samtidig. Derfor har teknikken flyttet seg fra forskningsmiljøene og inn i helt vanlig forretningsprogramvare.

Baksiden er at dataene nå gjør jobben koden pleide å gjøre. Har historikken feil, duplikater eller hull, lærer modellen det også, og ingenting i koden avslører det. D. Sculley og ni kolleger hos Google beskrev konsekvensene i *Hidden Technical Debt in Machine Learning Systems*, en artikkel i proceedings fra NIPS-konferansen i 2015. Temaet deres er teknisk gjeld. Den første figuren i artikkelen viser selve læringskoden som en liten svart boks midt i et mye større system, og ifølge bildeteksten er infrastrukturen rundt både enorm og sammensatt. Et sted anslår forfatterne at et modent system kan ende opp med høyst 5 prosent maskinlæringskode og minst 95 prosent limkode som bare flytter data inn i og ut av generelle programvarepakker.

De peker også på at det koster mer å være avhengig av data enn av kode. Kompilatorer og byggesystemer kan analysere hvilken kode som avhenger av hvilken, mens tilsvarende verktøy for data er langt sjeldnere. Dermed kan en kolonne få en ny betydning uten at noe som helst merker det. Forfatterne gir

også et prinsipp et navn, CACE, for «Changing Anything Changes Everything»: forskyves fordelingen av én inndata, kan måten modellen bruker alle de andre på, forskyve seg samtidig.

En modell er altså billig å lage og dyr å holde i live. Det som avgjør om den er til nytte, handler stort sett om data: hvor de kommer fra, om hver kunde står der én eller tre ganger, om det registrerte utfallet faktisk stemmer, og hvem som oppdager det når et system lenger opp i kjeden begynner å levere noe annet enn før.

Slik ser det ut i praksis

Tenk deg en bedrift som vil vite allerede samme dag som en faktura sendes ut, om den sannsynligvis blir betalt for sent. Da kan noen ringe kunden før forfallsdatoen, ikke etter. Et uttrekk har én rad per tidligere faktura, sortert etter fakturadato, og fire kolonner som alle var kjent den dagen fakturaen gikk ut: beløpet, antall kredittdager, hvor mange fakturaer kunden fikk året før, og hvor mange av dem som ble betalt for sent. En femte kolonne viser hvordan det gikk: 1 hvis fakturaen ble betalt for sent, 0 hvis den ble betalt i tide.

Skriptet nedenfor er skrevet for denne artikkelen og sjekket linje for linje mot dokumentasjonen til scikit-learn 1.9.1 og NumPy 2.5 i oktober 2026. Det trener en modell på de eldste tre fjerdedelene av fakturaene og tester den på den nyeste fjerdedelen:

```
# invoices.csv: én rad per faktura sortert etter dato, bare tall.
# Siste kolonne er 1 hvis fakturaen ble 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 fjerdedelen tilbake: ingenting lærer 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))
```

Fire valg i skriptet betyr mer enn hvilken algoritme som brukes.

Den nyeste fjerdedelen settes til side før noe lærer. Med `shuffle=False` beholder `train_test_split` rekkefølgen og legger de siste radene i testsettet. Modellen lærer dermed av eldre fakturaer og blir vurdert på nyere, akkurat slik den skal brukes. Med standardverdien `shuffle=True` ville delingen blitt tilfeldig, og en del av testen ville bestått av fakturaer som er eldre enn noen av dem modellen lærte av.

Skaleringen ligger inne i pipelinen. `StandardScaler` trekker fra gjennomsnittet for hver feature og skalerer den til enhetsvarians. Fordi `make_pipeline` kobler skaleringen sammen med klassifikatoren, beregnes gjennomsnitt og varians bare fra treningsradene, og ingenting fra den nyeste fjerdedelen lekker inn i modellen.

Målet er balansert nøyaktighet. For hver av de to klassene regnes andelen modellen traff riktig, altså de sene fakturaene den fanget og de punktlige den kjente igjen, og så tas gjennomsnittet. Blir hver tiende faktura betalt for sent, belønner vanlig nøyaktighet en modell som svarer «i tide» hver eneste gang. Det gjør ikke balansert nøyaktighet.

Modellen må slå en referansemodell. `DummyClassifier` med `strategy="most_frequent"` overser alle featurene og svarer alltid med den vanligste klassen. Målt i balansert nøyaktighet får den 0,5 med to klasser, fordi den treffer hver punktlige faktura og bommer på hver sene. Slår ikke modellen den klart, har den ikke lært noe nyttig, samme hvor fin den vanlige nøyaktigheten ser ut.

Ingenting av dette redder en modell fra dårlige rader. Ligger samme kunde inne som tre ulike poster, blir antallet sene fakturaer for kunden fordelt på tre, og modellen lærer av en historikk som aldri har skjedd. Siden vår om Klargjøring av data for AI handler om nettopp dette grunnarbeidet: at hver kunde finnes bare én gang, at hver ansatt bare får tilgang til det vedkommende skal se, og at

ethvert svar kan følges tilbake til dataene det bygger på. For en modell er det siste punktet det som gjør at du kan gå baklengs fra en merkelig prediksjon til radene som ga den.

Hvor det går galt

Informasjon fra fremtiden kommer med i treningen. Veiledningen om vanlige fallgruver i scikit-learn kaller dette datalekkasje (data leakage): modellen bygges med informasjon som ikke vil finnes når den skal gjøre en prediksjon. I fakturaeksemplet er den fristende ekstra kolonnen antall purringer som er sendt. Den forutsier forsinkelse svært godt, nettopp fordi purringer først sendes når fakturaen allerede er forfalt. Samme lekkasje skjer mer i det stille når en skalering eller en utfylling av manglende verdier tilpasses på alle rader før delingen, og regelen i veiledningen er at man aldri kaller `fit` på testdata. Løsningen er å skrive ned når verdien i hver kolonne blir kjent, å dele dataene først og å holde all forbehandling inne i pipelinen, slik at den bare tilpasses på treningsradene.

Modellen vurderes på noe den allerede har sett. En modell som bare gjentok svarene den var trent på, ville fått full score på sine egne treningsrader og vært ubrukelig på alt nytt. Veiledningen om kryssvalidering kaller dette overtilpasning (overfitting). En mer lumsk variant er å prøve mange innstillinger og beholde den som scorer best på de tilbakeholdte radene, for da lekker kunnskap om akkurat de radene inn i valget. Sammenlign heller innstillingene med kryssvalidering på treningsdelen, der radene deles i flere fold og hver fold etter tur er testsett. De tilbakeholdte radene ser du på én gang, helt til slutt.

Høy nøyaktighet som en konstant kunne matchet. Når den ene klassen er sjelden, blir nøyaktigheten dominert av den vanlige. Veiledningen advarer om at en fold med sjeldne klasser, som ved avsløring av svindel, kan ende uten et eneste eksempel på den sjeldne klassen. Rapporter balansert nøyaktighet eller et annet mål per klasse, sammenlign med en `DummyClassifier`, og bruk stratifiserte fold, som holder andelen av hver klasse omtrent lik i alle fold. `cross_val_score` bruker dem som standard for en klassifikator når du gir den et antall fold.

Tilfeldig deling av data som endrer seg over tid. Veiledningen om kryssvalidering påpeker at antakelsen om uavhengige, likt fordelte observasjoner sjelden holder i praksis, og at tidsordnede data bør testes på observasjoner som kommer senere enn dem modellen lærte av. En tilfeldig deling lar modellen kikke

inn i fremtiden og gir en score den aldri når i drift. Del etter dato, slik eksemplet gjør, og bruk ved kryssvalidering en deling som tar hensyn til tid, for eksempel `TimeSeriesSplit`, der hver fold ifølge veiledningen bør dekke like lang tid.

Forbehandlingen er ulik i trening og drift. Har modellen lært av skalerte features og det kjørende systemet sender rå tall, blir prediksjonene gale uten at det kommer noen feilmelding. Veiledningen om fallgruver krever at enhver transformasjon som brukes under treningen, også brukes på testdata og på data i produksjon. Lever den tilpassede pipelinen som ett samlet objekt, ikke som en modell pluss en beskrivelse av hva som må gjøres med dataene først.

Verden endrer seg etter at modellen er satt i drift. En modell trent på fjorårets fakturaer går ut fra at kundene fortsatt oppfører seg som da. Sculley og medforfatterne hans foreslår å følge med på prediksjonsskjevhet (prediction bias): i et system som virker som tenkt, bør fordelingen av de predikerte kategoriene vanligvis være lik fordelingen av dem som faktisk observeres, og et gap som åpner seg mellom de to, er ofte det første tegnet på at verden har flyttet seg. Hold den sammenligningen gående, tren på nytt med ferske data, og ta vare på en fryst, versjonert kopi av hvert inndatasignal som et annet team eier, slik at en velment endring lenger opp i kjeden ikke stille endrer modellen din.

Les mer

- User Guide — oppslagsverket til scikit-learn om veiledet og ikke-veiledet læring og om modellvalg og evaluering, kapittel for kapittel.
- Getting Started — det felles grensesnittet for estimatorer i korte eksempler: tilpasning og prediksjon, transformatorer, sammenkoblede trinn og kryssvalidering.
- Common pitfalls and recommended practices — inkonsekvent forbehandling, datalekkasje og tilfeldighet, med feil og riktig fremgangsmåte side om side i kode.
- Cross-validation: evaluating estimator performance — hvorfor testing på treningsdata villeder, og hvilken deling som passer for grupperte, skjevt fordelte eller tidsordnede data.
- Hidden Technical Debt in Machine Learning Systems — Sculley og kolleger hos Google, NIPS 2015, om hva det koster å drive en modell i produksjon utover selve modellen.

Ordliste

Velg ut fra deres faktiske rammer

Beskriv arbeidsflyten, eksisterende grensesnitt og brukerne som trenger endringen. Vi kan definere en testbar leveranse og en overlevering som gjør teamet i stand til å vedlikeholde løsningen.

Se hva det koster Ta kontakt →

Gjør begrepene praktiske

Spør hvordan «Hva er maskinlæring?» gjelder i organisasjonen deres. Et konkret eksempel fra dagens system er nok til å starte samtalen.

Se tjenestene våre

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

Ta med deg denne siden

Visittkort

Laget med kjærlighet av D.

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

Opphavsrett © 2025 – nå · Engineer ApS · Alle rettigheter forbeholdt.

Skrevet ut fra https://engineer.company/nb/glossary/machine-learning/