

Engineer

Машинне навчання

побудова моделей, які вчаться на прикладах із даних



Що це таке

Машинне навчання (machine learning) — це створення програм, які вчаться виконувати завдання на прикладах, а не за правилами, заздалегідь записаними програмістом. Програмі показують багато випадків, для яких відповідь уже відома. Вона знаходить зв'язок між вхідними даними й відповідями, а потім за цим зв'язком передбачає відповідь для кожного нового випадку. Сам термін старший за майже все програмне забезпечення, яке його сьогодні використовує: А. Л. Семюел (A. L. Samuel) виніс його в назву дослідження гри в шашки, яке журнал IBM Journal of Research and Development надрукував у липні 1959 року.

Найпростіше розібратися з поняттями на прикладі scikit-learn, бібліотеки для Python з відкритим кодом; саме її документації дотримується ця стаття. Приклади зберігаються в таблиці. Кожен рядок — це одне спостереження (у scikit-learn його називають sample): рахунок, клієнт чи показ датчика. Кожен стовпець — ознака (feature), тобто число, що описує спостереження. Відповідь, яку програма має навчитися давати, називається цільовою змінною (target). Те, що навчається, у scikit-learn зветься оцінювачем (estimator). Кожен оцінювач має метод `fit`, який навчається на показаних йому рядках. Оцінювачі, що виконують прогнозування, використовують `predict`, щоб давати відповіді для нових рядків, а перетворювачі використовують `transform`, щоб отримати перетворену таблицю спостережень.

Більшість задач належить до однієї з двох груп. У навчанні з учителем кожен навчальний рядок має свою цільову змінну. Якщо це категорія, наприклад «оплачено вчасно» чи «оплачено із запізненням», задача називається класифікацією. Якщо це кількість, наприклад число днів до оплати, — регресією. У навчанні без учителя цільової змінної немає взагалі, і програма сама шукає структуру в даних, скажімо, об'єднує в групи клієнтів зі схожою

поведінкою. У документації scikit-learn кожна з двох груп має окремий розділ, а за ними йде розділ про вибір моделі та її оцінювання. Саме там і криється більшість практичних труднощів.

Водночас машинне навчання не є назвою для будь-якої програми, що здається розумною. Правило, яке позначає кожен рахунок понад певну суму, написала людина, і воно робить рівно те, що в ньому записано. Модель, яка позначає рахунки, вивела своє правило з минулих рахунків сама. Тому вона може помилятися так, як ніхто не передбачав, і змінює поведінку щоразу, коли змінюються дані, на яких вона навчається.

Застосуйте поняття до рішення

Застосуйте «Що таке машинне навчання?» до конкретного рішення. Ми можемо пов'язати поняття із системою, яку ви створюєте або купуєте. Агент, під'єднаний до систем, якими ви вже користуєтеся, перевірений на прикладах, які складають ваші експерти, під наглядом ваших людей і не прив'язаний до однієї моделі.

Переглянути наші послуги Зв'язатися з нами →

Перевірте припущення за терміном

Використайте пояснення та пов'язані роботи, щоб сформулювати запитання для проєкту. Ми можемо перевірити припущення разом до вибору інструмента. Побудувала механізм із 268 перевірок репозиторію — 268 перевірок коміту і 146 тестів самих перевірок

Побудувала механізм із 268 перевірок репозиторію Зв'язатися з нами →

Чому це важливо

Деякі завдання набагато легше показати, ніж описати. Ніхто не напише вичерпного правила, які рахунки оплатять із запізненням, але більшість компаній має роки рахунків, де зафіксовано, чим усе скінчилося. Машинне навчання перетворює цю історію на робоче правило і може зробити це для багатьох подібних питань одночасно. Тому воно й перейшло з лабораторій у звичайні бізнесові програми.

Зворотний бік у тому, що тепер роботу коду виконують дані. Якщо в історії є помилки, дублікати чи прогалини, модель вивчить і їх, а в коді цього не видно. Д. Скаллі (D. Sculley) і дев’ятеро його колег із Google описали наслідки в статті *Hidden Technical Debt in Machine Learning Systems*, опублікованій у збірнику конференції NIPS 2015 року; її тема — технічний борг. На першому рисунку статті сам код машинного навчання намальовано маленькою чорною скринькою посеред значно більшої системи, а підпис називає інфраструктуру навколо неї величезною і складною. В одному місці автори оцінюють, що зріла система може складатися щонайбільше на 5 відсотків з

коду машинного навчання і щонайменше на 95 відсотків з клейового коду (glue code), який лише переносить дані до програмних пакетів загального призначення і назад.

Автори також доводять, що залежність від даних обходиться дорожче, ніж залежність від коду. Компілятори та системи збирання відстежують, який код від якого залежить, а подібні інструменти для даних трапляються значно рідше. Тож стовпець може непомітно змінити своє значення, і цього ніхто не зауважить. Ще вони дають назву принципу CACE, «Changing Anything Changes Everything»: варто змінитися розподілу однієї вхідної ознаки, і може змінитися те, як модель використовує всі інші.

Отже, модель дешево створити й дорого підтримувати. Те, чи буде від неї користь, здебільшого залежить від даних: звідки вони беруться, чи кожен клієнт записаний один раз, а не тричі, чи правильно зафіксовано результат і хто помітить, коли система вище за ланцюжком почне надсилати щось інше.

Як це виглядає на практиці

Уявімо компанію, яка хоче вже в день виставлення рахунку знати, чи його, імовірно, оплатять із запізненням. Тоді хтось зможе зателефонувати клієнтові до настання строку оплати, а не після нього. У вивантаженні є по одному рядку на кожен минулий рахунок, відсортовані за датою виставлення, і чотири стовпці, відомі вже в день, коли рахунок надіслали: сума, кількість днів відстрочки, скільки рахунків клієнт отримав за попередній рік і скільки з них оплатив із запізненням. П’ятий стовпець показує, чим усе скінчилося: 1, якщо рахунок оплачено із запізненням, і 0, якщо вчасно.

Скрипт нижче написано для цієї статті й рядок за рядком звірено з документацією scikit-learn 1.9.1 та NumPy 2.5 у жовтні 2026 року. Він навчає модель на найстаріших трьох чвертях рахунків і перевіряє її на найновішій чверті:

```
# invoices.csv: рядок на рахунок, відсортовано за датою, лише числа.
# Останній стовпець: 1, якщо рахунок оплачено із запізненням, інакше 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)

# Відкладаємо найновішу чверть: на ній ніщо не навчається.
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))
```

Чотири рішення в ньому важать більше, ніж вибір алгоритму.

Найновішу чверть відкладено ще до того, як щось навчається. З `shuffle=False` функція `train_test_split` зберігає порядок рядків і кладе останні з них у тестову вибірку. Отже, модель вчиться на старіших рахунках, а оцінюють її на новіших, тобто так само, як її використовуватимуть насправді. Зі значенням за замовчуванням, `shuffle=True`, поділ був би випадковим, і модель частково перевіряли б на рахунках, старших за деякі з тих, на яких вона вчилася.

Масштабування вбудовано в конвеєр. `StandardScaler` віднімає від кожної ознаки її середнє значення і масштабує її до одиничної дисперсії. Оскільки `make_pipeline` з'єднує масштабування з класифікатором, середнє та дисперсію обчислюють лише за навчальними рядками, і ніщо з найновішої чверті не просочується в модель.

Оцінка — це збалансована точність (balanced accuracy). Для кожного з двох класів рахують частку прикладів, які модель визначила правильно, тобто спіймані прострочені рахунки і розпізнані вчасні, а потім беруть середнє. Якщо із запізненням оплачують кожен десятий рахунок, звичайна точність винагороджує модель, яка щоразу відповідає «вчасно». Збалансована точність цього не робить.

Модель має перевершити базову. `DummyClassifier` зі `strategy="most_frequent"` ігнорує всі ознаки й завжди відповідає найпоширенішим класом. За збалансованою точністю він отримує 0,5 для двох класів, бо вгадує кожен вчасний рахунок і пропускає кожен прострочений. Якщо модель не випереджає його помітно, вона не навчилася нічого корисного, хоч якою гарною виглядає її звичайна точність.

Усе це не врятує модель від поганих рядків. Якщо той самий клієнт записаний як три різні записи, кількість його прострочених рахунків розпадається на три частини, і модель вчиться на історії, якої ніколи не було. Наша сторінка Підготовка даних для ШІ присвячена саме цій підготовчій роботі: щоб кожен клієнт існував в одному записі, щоб кожен працівник мав доступ лише до того, що йому належить бачити, і щоб будь-яку відповідь можна було звести до даних, на яких вона ґрунтується. Для моделі саме останнє дає змогу пройти назад від дивного прогнозу до рядків, які його породили.

Де виникають проблеми

У навчання потрапляє інформація з майбутнього. Посібник scikit-learn про поширені пастки називає це витоком даних (data leakage): модель будують на інформації, якої не буде в момент прогнозу. У прикладі з рахунками спокусливий додатковий стовпець — кількість надісланих нагадувань про оплату. Він чудово передбачає запізнення, бо нагадування надсилають лише тоді, коли рахунок уже прострочено. Той самий витік стається непомітніше, коли масштабування чи заповнення пропущених значень налаштовують на всіх рядках ще до поділу, і правило посібника — ніколи не викликати `fit` на тестових даних. Рішення: записати для кожного стовпця, коли саме стає відоме його значення, спершу поділити дані й тримати всю попередню обробку всередині конвеєра, щоб її налаштовували лише на навчальних рядках.

Модель оцінюють на тому, що вона вже бачила. Модель, яка просто повторювала б відповіді, на яких її навчили, отримала б ідеальний результат на власних навчальних рядках і була б марною для всього нового. Посібник про перехресну перевірку (cross-validation) називає це перенавчанням (overfitting). Підступніший варіант — перепробувати багато налаштувань і залишити те, що найкраще працює на відкладених рядках, бо тоді знання саме про ці рядки просочується у вибір. Натомість порівнюйте налаштування

перехресною перевіркою на навчальній частині: рядки ділять на кілька блоків (folds), і кожен блок по черзі стає тестовим. На відкладені рядки дивіться один раз, наприкінці.

Висока точність, якої досягла б і константа. Коли один клас рідкісний, точність визначає переважно поширений клас. Посібник попереджає, що за рідкісних класів, як-от у виявленні шахрайства, у блоці може не опинитися жодного прикладу рідкісного класу. Наводьте збалансовану точність або інший показник для кожного класу окремо, порівнюйте з `DummyClassifier` і використовуйте стратифіковані блоки, які зберігають частки класів приблизно однаковими в кожному блоці. `cross_val_score` бере їх за замовчуванням для класифікатора, коли йому передають кількість блоків.

Випадковий поділ даних, що змінюються в часі. Посібник про перехресну перевірку зазначає, що припущення про незалежні й однаково розподілені спостереження на практиці виконується рідко, а дані, впорядковані в часі, слід перевіряти на спостереженнях, пізніших за ті, на яких модель навчалася. Випадковий поділ дозволяє моделі зазирнути в майбутнє і дає результат, якого вона в роботі не досягне. Діліть за датою, як у прикладі, а для перехресної перевірки беріть поділ, що враховує час, наприклад `TimeSeriesSplit`, у якому, за посібником, кожен блок має охоплювати однаковий проміжок часу.

Попередня обробка під час навчання і в роботі різна. Якщо модель вчилася на масштабованих ознаках, а робоча система надсилає сирі числа, прогнози будуть хибними, і жодної помилки ніхто не побачить. Посібник про пастки вимагає, щоб кожне перетворення, застосоване під час навчання, так само застосовувалося до тестових даних і до даних у промисловій експлуатації. Передавайте в роботу налаштований конвеєр як один об’єкт, а не модель із приміткою, що спершу зробити з даними.

Світ змінюється після запуску. Модель, навчена на торішніх рахунках, припускає, що клієнти поводяться так само, як тоді. Скаллі та співавтори радять стежити за зміщенням прогнозів (prediction bias): у системі, яка працює як задумано, розподіл передбачених міток зазвичай має збігатися з розподілом фактично спостережених, і розрив між ними часто стає першою ознакою того, що світ змінився. Постійно робіть це порівняння, перенавчайте модель на свіжих даних і тримайте заморожену версійовану копію кожного вхідного сигналу, яким володіє інша команда, щоб добра за намірами зміна вище за ланцюжком не змінила вашу модель непомітно.

Що почитати далі

- User Guide — довідник scikit-learn про навчання з учителем і без учителя, а також про вибір моделі та її оцінювання, розділ за розділом.
- Getting Started — спільний інтерфейс оцінювачів у коротких прикладах: навчання і прогноз, перетворювачі, ланцюжки кроків і перехресна перевірка.
- Common pitfalls and recommended practices — неузгоджена попередня обробка, витік даних і випадковість, з хибним і правильним підходом поруч у коді.
- Cross-validation: evaluating estimator performance — чому перевірка на навчальних даних вводить в оману і який поділ пасує для згрупованих, незбалансованих чи впорядкованих у часі даних.
- Hidden Technical Debt in Machine Learning Systems — Скаллі та колеги з Google, NIPS 2015, про те, чого коштує експлуатація моделі понад саму модель.

Словник

Обирайте за фактичними умовами

Опишіть робочий процес, наявні інтерфейси та користувачів, яким потрібна зміна. Ми можемо визначити перевірюваний результат і спосіб передачі, який дасть вашій команді змогу підтримувати його.

Дивитися, скільки це коштує Зв'язатися з нами →

Зробіть термінологію практичною

Запитайте, як «Що таке машинне навчання?» стосується вашої організації. Конкретного прикладу з поточної системи достатньо для початку розмови.

Переглянути наші послуги

welcome@engineer.company +45 50 10 61 84 Копенгаген, Данія

Візьміть цю сторінку із собою

Візитівка

Дбайливо зроблено з любов'ю — D.

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

Авторське право © 2025 – дотепер · Engineer ApS · Усі права захищено.

Надруковано з https://engineer.company/uk/glossary/machine-learning/