Engineer

Apprentissage automatique

Créer un logiciel qui apprend une tâche à partir d'exemples

Ce que c'est

L'apprentissage automatique consiste à construire un logiciel qui apprend une tâche à partir d'exemples, plutôt que de règles écrites par un programmeur. Vous lui donnez de nombreux cas dont la réponse est connue ; il trouve une relation entre les entrées et les réponses et l'utilise pour prédire la réponse de chaque nouveau cas. Le nom est plus ancien que presque tous les logiciels qui l'emploient aujourd'hui : A. L. Samuel l'a placé dans le titre d'une étude du jeu de dames publiée en juillet 1959 par l'IBM Journal of Research and Development.

Le vocabulaire se précise facilement avec scikit-learn, bibliothèque Python open source dont cet article suit la documentation. Les exemples sont présentés dans un tableau. Chaque ligne est un échantillon, comme une facture, un client ou une mesure de capteur ; chaque colonne est une caractéristique, un nombre décrivant l'échantillon. La réponse que vous voulez faire apprendre est la cible. Pour scikit-learn, ce qui apprend est un estimateur. Tout estimateur possède une méthode `fit` qui apprend à partir des lignes présentées. Les estimateurs prédictifs utilisent `predict` pour fournir des réponses pour de nouvelles lignes ; les transformateurs utilisent `transform` pour produire un tableau d'échantillons transformé.

Deux familles couvrent l'essentiel du travail. En apprentissage supervisé, chaque ligne d'entraînement porte sa cible. Si celle-ci est une catégorie, comme payé à temps ou en retard, la tâche est une classification ; si c'est une quantité, comme le nombre de jours jusqu'au paiement, c'est une régression. En apprentissage non supervisé, il n'y a aucune cible : le logiciel cherche lui-même une structure, par exemple en regroupant des clients aux comportements proches. La documentation scikit-learn consacre un chapitre à chaque famille, puis un à la sélection et à l'évaluation des modèles, où se situe l'essentiel de la difficulté pratique.

Du moins dans ce sens, l'apprentissage automatique ne désigne pas tout logiciel paraissant intelligent. Une règle signalant toute facture supérieure à un montant fixe a été écrite par quelqu'un et fait exactement ce qu'elle dit. Un modèle qui signale des factures a déduit sa règle des factures passées : il peut donc se tromper d'une manière que personne n'a programmée, et son comportement change quand changent les données dont il apprend.

Utilisez le concept pour prendre une décision

Utilisez « Qu'est-ce que l'apprentissage automatique ? » pour éclairer une décision réelle. Nous pouvons vous aider à relier ce concept au système que vous construisez ou achetez. Un agent connecté à vos systèmes, testé sur des cas rédigés par vos experts, supervisé par votre équipe et indépendant d'un modèle unique.

Découvrir nos prestations Contactez-nous →

Vérifiez les hypothèses derrière le terme

Appuyez-vous sur l'explication et les réalisations liées pour formuler les questions de votre projet. Nous pouvons examiner les hypothèses avec vous avant le choix d'un outil. Moteur de 268 contrôles pour le dépôt — 268 contrôles de commit et 146 tests des contrôles eux-mêmes

Moteur de 268 contrôles pour le dépôt Contactez-nous →

Pourquoi c'est important

Certaines tâches se montrent bien plus facilement par l'exemple qu'elles ne se décrivent. Personne ne sait écrire une règle complète indiquant quelles factures seront payées en retard, mais la plupart des entreprises possèdent des années de factures avec leur résultat. L'apprentissage automatique transforme cet historique en règle opérationnelle et peut en faire autant pour de nombreuses questions semblables à la fois : c'est pourquoi il est passé de la recherche aux logiciels métier ordinaires.

En contrepartie, les données font désormais le travail que le code faisait auparavant. Si l'historique contient des erreurs, doublons ou lacunes, le modèle les apprend aussi, sans que le code le montre. D. Sculley et neuf collègues de Google ont décrit les conséquences dans l'article *Dette technique cachée des systèmes d'apprentissage automatique*, paru dans les actes de la conférence NIPS de 2015. Leur première figure représente le code d'apprentissage comme une petite boîte noire au centre d'un système bien plus grand ; la légende qualifie l'infrastructure qui l'entoure de vaste et complexe. Dans un passage, ils estiment qu'un système mûr peut finir avec au plus 5 % de code d'apprentissage automatique et au moins 95 % de code de liaison faisant entrer et sortir les données de composants généralistes.

Ils soutiennent aussi qu'une dépendance aux données coûte plus cher qu'une dépendance au code. Les compilateurs et systèmes de construction suivent les dépendances entre les éléments du code ; les outils équivalents pour les données sont bien moins courants, si bien qu'une colonne peut changer de sens sans être détectée. Ils nomment un principe CACE, pour « Changing Anything Changes Everything », soit « tout changement change tout » : modifiez la distribution d'une entrée et la manière dont le modèle utilise toutes les autres peut changer avec elle.

Un modèle est donc peu coûteux à créer et coûteux à maintenir. Le travail qui détermine son utilité porte surtout sur les données : leur origine, la présence d'un client une ou trois fois, l'exactitude du résultat enregistré et la personne qui remarquera qu'un système en amont commence à envoyer autre chose.

À quoi cela ressemble en pratique

Prenez une entreprise voulant repérer, dès le jour de leur émission, les factures qui risquent d'être payées en retard, afin d'appeler ces clients avant l'échéance plutôt qu'après. Un export contient une ligne par facture passée, triée par date d'émission, avec quatre colonnes connues le jour de l'envoi : le montant, les jours de crédit accordés, le nombre de factures reçues par le client l'année précédente et le nombre de celles payées en retard. Une cinquième colonne consigne le résultat : 1 pour un paiement tardif, 0 pour un paiement à temps.

Le script ci-dessous a été écrit pour cet article et vérifié ligne par ligne à partir des documentations de scikit-learn 1.9.1 et NumPy 2.5 en octobre 2026. Il entraîne un modèle sur les anciennes factures et le teste sur le quart le plus récent :

```
# 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))
```

Quatre choix y comptent davantage que celui de l'algorithme.

Le quart le plus récent est réservé avant tout apprentissage. Avec `shuffle=False`, `train_test_split` garde les lignes dans l'ordre et place le dernier quart dans le jeu de test : le modèle apprend sur les anciennes factures et est évalué sur les nouvelles, comme lors de son utilisation. Avec la valeur par défaut, `shuffle=True`, le partage serait aléatoire : il serait en partie testé sur des factures antérieures à certaines factures d'entraînement.

La mise à l'échelle reste dans le pipeline. `StandardScaler` centre chaque caractéristique en soustrayant sa moyenne, puis la met à l'échelle pour obtenir une variance unitaire. Comme `make_pipeline` l'enchaîne au classificateur, moyenne et variance proviennent uniquement des lignes d'entraînement : aucune information du quart le plus récent ne s'infiltre dans le modèle.

Le score est l'exactitude équilibrée. Il fait la moyenne, sur les deux classes, de la proportion correctement reconnue dans chacune : détection des factures payées en retard et reconnaissance de celles payées à temps. Si une facture sur dix est payée en retard, l'exactitude ordinaire récompense un modèle répondant toujours « à temps ». L'exactitude équilibrée ne le fait pas.

Le modèle doit battre une référence. `DummyClassifier` avec `strategy="most_frequent"` ignore les caractéristiques et donne toujours la réponse la plus fréquente. Sur deux classes, son exactitude équilibrée est de 0,5 : il reconnaît toutes les factures payées à temps et se trompe sur toutes les tardives. Un modèle ne faisant pas clairement mieux n'a rien appris d'utile, quelle que soit son exactitude ordinaire apparente.

Rien de cela ne sauve un modèle de mauvaises lignes. Si un client existe sous trois enregistrements, le nombre de ses factures tardives est divisé en trois et le modèle apprend un historique qui n'a jamais existé. Notre page Préparation des données à l'IA traite de ce socle : un seul enregistrement par client, chaque personne n'accédant qu'à ce qu'elle doit voir, et toute réponse traçable jusqu'aux données qui la fondent. Pour un modèle, ce dernier point permet de remonter d'une prédiction étrange aux lignes qui l'ont produite.

Où les problèmes surviennent

Des informations futures entrent dans l'entraînement. Le guide des pièges courants de scikit-learn appelle cela une fuite de données : construire le modèle avec des informations indisponibles au moment de prédire. Dans l'exemple des factures, la colonne tentante est le nombre de relances, qui prédit admirablement le retard puisque les relances ne partent qu'après l'échéance. La même fuite est plus discrète lorsqu'un outil de mise à l'échelle ou d'imputation est ajusté sur toutes les lignes avant le partage ; la règle du guide est de ne jamais appeler `fit` sur les données de test. Pour chaque colonne, notez quand sa valeur devient connue, partagez d'abord les données et gardez toutes les étapes de prétraitement dans le pipeline pour les ajuster sur les seules lignes d'entraînement.

Le modèle est évalué sur ce qu'il a déjà vu. Un modèle ne répétant que ses cibles d'entraînement obtiendrait un score parfait sur ses propres lignes tout en étant inutile sur les nouvelles ; le guide de validation croisée appelle cela le surapprentissage. Une forme plus subtile consiste à essayer de nombreux réglages puis garder le meilleur score sur les lignes réservées : des informations sur ces lignes influencent alors le choix. Comparez les réglages par validation croisée sur la partie d'entraînement, divisée en plusieurs plis prenant chacun leur tour le rôle du jeu de test, puis ne consultez les lignes réservées qu'une fois, à la fin.

Une exactitude élevée qu'une constante égalerait. Lorsqu'une classe est rare, l'exactitude est dominée par la plus fréquente. Le guide avertit qu'avec des classes rares, comme en détection de fraude, un pli peut ne contenir aucun exemple de la classe rare. Indiquez l'exactitude équilibrée ou une autre mesure par classe, comparez avec un `DummyClassifier` et utilisez des plis stratifiés, qui gardent des proportions de classes à peu près identiques d'un pli à l'autre. `cross_val_score` les utilise par défaut pour un classificateur lorsque vous lui passez un nombre de plis.

Un partage aléatoire pour des données qui évoluent dans le temps. Le guide de validation croisée note que l'hypothèse d'échantillons indépendants et identiquement distribués tient rarement en pratique, et que des données ordonnées dans le temps doivent être testées sur des observations postérieures à celles de l'apprentissage. Un partage aléatoire laisse le modèle entrevoir l'avenir et annonce un score qu'il n'atteindra pas. Partagez par date, comme dans l'exemple, et utilisez en validation croisée un outil tenant compte du temps tel que `TimeSeriesSplit`, dont les plis doivent, selon le guide, couvrir chacun la même durée.

Le prétraitement diffère entre l'entraînement et la production. Si le modèle a appris sur des caractéristiques mises à l'échelle et que le système réel lui envoie les valeurs brutes, les prédictions sont fausses sans erreur signalée. Le guide exige que toute transformation de l'entraînement soit aussi appliquée aux données de test et de production. Livrez le pipeline ajusté comme un seul objet, plutôt qu'un modèle accompagné d'une note sur le traitement préalable des données.

Le monde change après le lancement. Un modèle entraîné sur les factures de l'an passé suppose que les clients se comportent encore de même. Sculley et ses coauteurs suggèrent de suivre le biais de prédiction : dans un système fonctionnant comme prévu, la proportion de chaque étiquette prédite devrait généralement correspondre à la proportion observée ; un écart qui se creuse est souvent le premier signe d'une évolution du contexte. Poursuivez cette comparaison en continu, réentraînez sur des données récentes et gardez une copie figée et versionnée de tout signal d'entrée appartenant à une autre équipe, pour qu'un changement bien intentionné en amont ne modifie pas silencieusement votre modèle.

Pour aller plus loin

- Guide d'utilisation — la référence scikit-learn sur l'apprentissage supervisé et non supervisé, la sélection et l'évaluation des modèles, chapitre par chapitre.
- Premiers pas — l'interface d'estimateur en quelques exemples : ajustement et prédiction, transformateurs, pipelines et validation croisée.
- Pièges courants et pratiques recommandées — prétraitement incohérent, fuites de données et aléatoire, avec les mauvaises et bonnes pratiques côte à côte dans le code.

- Validation croisée : évaluer la performance d'un estimateur — pourquoi tester sur l'entraînement trompe, et quel partage convient aux données groupées, déséquilibrées ou temporelles.
- Dette technique cachée des systèmes d'apprentissage automatique — Sculley et ses collègues de Google, NIPS 2015, sur le coût d'un modèle en production au-delà du modèle lui-même.

Glossaire

Choisissez selon vos contraintes réelles

Présentez le processus de travail, les interfaces existantes et les utilisateurs qui ont besoin du changement. Nous pouvons définir une livraison testable et une transmission permettant à votre équipe d'assurer la maintenance des éléments livrés.

Consulter les tarifs Contactez-nous →

Rendez la terminologie concrète

Demandez comment « Qu'est-ce que l'apprentissage automatique ? » s'applique à votre organisation. Un exemple concret tiré de votre système actuel suffit pour engager la discussion.

Découvrir nos prestations

welcome@engineer.company +45 50 10 61 84 Copenhague, Danemark

Emportez cette page avec vous

Fiche de contact

Créé avec soin et amour par D.

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

Droit d'auteur © 2025 – aujourd'hui · Engineer ApS · Tous droits réservés.

Imprimé depuis https://engineer.company/fr/glossary/machine-learning/