IA moderna

Scritto da Febogamedev

Con IA moderna possiamo indicare l'insieme degli approcci nei quali l'apprendimento dai dati ha assunto un ruolo centrale.
Nell'IA classica, soprattutto simbolica, una parte importante delle conoscenze e delle regole necessarie per affrontare un problema viene rappresentata esplicitamente dal progettista.
Nell'IA moderna basata sul machine learning, invece, il sistema utilizza dati o esperienza per apprendere automaticamente alcune delle regolarità necessarie a svolgere il proprio compito (Goodfellow et al., 2016).

Il passaggio fondamentale può essere rappresentato in modo semplificato così:

- IA classica → conoscenze e regole rappresentate esplicitamente.
- IA moderna → regolarità apprese dai dati attraverso un modello.

Questo non significa che l'IA moderna abbia completamente sostituito quella classica. Molti sistemi contemporanei combinano infatti apprendimento automatico, ricerca, regole esplicite e altri strumenti (Russell & Norvig, 2021).

L'IA moderna comprende tecniche molto differenti, ma molti sistemi basati sul machine learning condividono alcune caratteristiche:

- Apprendimento dai dati: I dati vengono utilizzati durante l'addestramento per modificare i parametri di un modello e permettergli di individuare regolarità utili al compito.
- Modelli parametrizzati: Il comportamento del sistema dipende da un insieme di parametri che vengono determinati o modificati durante l'apprendimento.
- Generalizzazione: L'obiettivo non è soltanto ottenere buoni risultati sui dati utilizzati durante l'addestramento, ma riuscire a elaborare correttamente anche nuovi casi mai incontrati prima (James et al., 2021).
- Approccio statistico: Molti modelli lavorano attraverso relazioni statistiche, probabilità, distanze o funzioni matematiche, anziché attraverso un insieme completo di regole logiche definite manualmente.
- Scalabilità: In alcune famiglie di modelli, soprattutto nel deep learning, l'aumento della quantità di dati, della capacità del modello e delle risorse computazionali può migliorare significativamente le prestazioni (Goodfellow et al., 2016).
- Capacità generativa: Alcuni modelli non si limitano a classificare o prevedere, ma possono generare nuovi contenuti, come testi, immagini, audio, video o codice.

L'apprendimento dai dati permette di affrontare problemi difficili da formalizzare attraverso regole esplicite, ma introduce anche nuovi limiti.

- Uno dei principali riguarda la dipendenza dai dati: la qualità del modello dipende fortemente dalla qualità, dalla quantità e dalla rappresentatività dei dati utilizzati durante l'addestramento. Se i dati contengono errori, squilibri o distorsioni, il modello può apprendere regolarità altrettanto distorte (National Institute of Standards and Technology [NIST], 2024).

- Un altro problema riguarda l'interpretabilità: alcuni modelli, come una regressione lineare o un piccolo albero decisionale, possono essere relativamente semplici da analizzare. Modelli molto complessi, come grandi reti neurali, possono invece rendere difficile ricostruire in modo completo il processo che ha prodotto un determinato risultato (James et al., 2021).
- Esiste inoltre un problema di generalizzazione: un modello può ottenere ottimi risultati su dati simili a quelli utilizzati durante l'addestramento e comportarsi peggio quando incontra situazioni significativamente differenti (James et al., 2021).
- Infine, alcuni sistemi moderni, soprattutto quelli di grandi dimensioni, possono richiedere quantità elevate di dati, memoria, potenza di calcolo ed energia: questo costo, tuttavia, varia enormemente in base al modello (NIST, 2024).

Lo sviluppo dell'IA moderna non riguarda soltanto il modo in cui vengono costruiti i sistemi informatici, ma produce anche conseguenze sociali, economiche e culturali.

L'automazione può modificare attività professionali e processi produttivi, mentre contemporaneamente emergono nuove competenze e nuovi ruoli legati allo sviluppo e all'utilizzo dei sistemi di IA (Russell & Norvig, 2021).
Gli strumenti generativi rendono inoltre accessibili attività come produzione di immagini, testi, audio e codice, modificando il rapporto tra competenza tecnica, creatività e strumenti digitali.

Queste tecnologie sollevano però anche questioni relative a bias e discriminazioni, privacy, sorveglianza, disinformazione, diritto d'autore, trasformazione del lavoro e rapporto tra esseri umani e sistemi artificiali (NIST, 2024).
Un sistema può inoltre produrre testi, immagini o risposte con caratteristiche apparentemente umane senza che questo permetta, da solo, di concludere che possieda comprensione, emozioni o coscienza (Russell & Norvig, 2021).

L'IA moderna pone quindi non soltanto problemi tecnici, ma anche una domanda più generale: quali attività vogliamo delegare alle macchine, con quali criteri e sotto quale responsabilità umana?

Machine learning

Il machine learning (ML) è uno degli approcci centrali dell'IA moderna.
Comprende metodi statistici e algoritmici che permettono a un sistema di apprendere regolarità dai dati o dall'esperienza e di utilizzarle per effettuare previsioni, classificazioni, decisioni o altre elaborazioni (Goodfellow et al., 2016).

Il sistema non riceve necessariamente dal programmatore tutte le regole necessarie per svolgere il compito.
Attraverso un processo di addestramento, un algoritmo modifica i parametri di un modello sulla base dei dati disponibili, cercando di migliorarne progressivamente le prestazioni (Goodfellow et al., 2016).

Apprendere dai dati significa quindi individuare relazioni o strutture che possono essere utilizzate anche su nuovi casi.

Per esempio, immaginiamo di avere un insieme di dati relativi alla vendita di alcune abitazioni. Per ogni casa conosciamo caratteristiche come la superficie, il numero di stanze, la zona in cui si trova e il prezzo di vendita.

Osservando molti esempi potremmo individuare alcune regolarità: abitazioni più grandi tendono, a parità di altre condizioni, ad avere prezzi maggiori; la posizione può influenzare il valore; anche il numero di stanze può essere associato al prezzo.

Nell'apprendimento supervisionato possiamo rappresentare in modo semplificato questa relazione come:
$y = f(x_1, x_2, x_3)$
Dove:

- x_1 ,x_2, x_3 rappresentano alcune caratteristiche della casa.
- y rappresenta il prezzo.

L'algoritmo di apprendimento utilizza molti esempi nei quali sono noti sia gli input sia il prezzo reale per costruire un modello che approssimi questa relazione.

Una volta addestrato, il modello può ricevere i dati relativi a una nuova abitazione, mai incontrata durante l'addestramento, e produrre una stima del prezzo.

- Dati di addestramento → algoritmo di apprendimento → modello.
- Nuovo input → modello → previsione.

Un altro esempio è la classificazione delle email.
Supponiamo di disporre di molte email già indicate come spam o non spam.
Ogni email può essere descritta attraverso caratteristiche come le parole presenti, il numero di collegamenti o alcune informazioni sul mittente.

L'algoritmo utilizza questi esempi per costruire un modello capace di riconoscere regolarità associate alle due categorie.
Quando riceve una nuova email, il modello può quindi classificarla come spam o non spam oppure, a seconda del modello utilizzato, assegnarle una probabilità di appartenenza alle diverse classi.
Il modello non deve necessariamente possedere una comprensione del concetto di spam analoga a quella umana: utilizza le relazioni apprese durante l'addestramento per produrre una previsione.

Possiamo quindi definire il machine learning come l'insieme di metodi attraverso cui un algoritmo utilizza dati o esperienza per costruire o modificare un modello capace di individuare regolarità e utilizzarle per affrontare nuovi casi.

Il processo di sviluppo di un sistema di machine learning può essere rappresentato, in modo semplificato, attraverso alcune fasi.

- Inizialmente vengono raccolti i dati necessari al problema.

- Questi possono richiedere una fase di preprocessing, durante la quale vengono controllati, puliti, trasformati o normalizzati.
- Successivamente viene scelto il tipo di modello e il metodo di apprendimento più adatto al problema.
- Durante l'addestramento, l'algoritmo modifica i parametri del modello sulla base dei dati e di un criterio che permette di valutarne le prestazioni.
- Il modello viene poi sottoposto a una fase di validazione, utilizzata durante lo sviluppo per confrontare differenti configurazioni e scegliere gli **iperparametri**. Gli iperparametri sono valori impostati dall'esterno prima o durante l'addestramento e non vengono appresi direttamente dai dati. Determinano il modo in cui avviene l'apprendimento, per esempio la velocità di apprendimento, la profondità massima di un albero decisionale o il numero di cluster in k-means (James et al., 2021).
- Infine viene effettuato il test, utilizzando dati non impiegati durante l'addestramento o la scelta del modello, per verificare quanto il sistema sia capace di generalizzare, cioè di comportarsi correttamente su nuovi casi (James et al., 2021).

Uno dei problemi principali dell'apprendimento è trovare un equilibrio tra **underfitting** e **overfitting**.

- Nell'underfitting, il modello non riesce ad apprendere sufficientemente le regolarità presenti nei dati e presenta prestazioni scarse anche sugli esempi utilizzati durante l'addestramento.
- Nell'overfitting, invece, il modello si adatta eccessivamente ai dati di training: ottiene prestazioni molto buone su questi esempi, ma significativamente peggiori su dati nuovi (James et al., 2021).

L'obiettivo non è quindi semplicemente ridurre al minimo l'errore sui dati di addestramento, ma costruire un modello che sia capace di generalizzare.
Se le prestazioni non sono soddisfacenti, il processo può essere ripetuto modificando i dati, il modello, gli iperparametri o altre scelte progettuali.
Quando il risultato raggiunge un livello considerato adeguato, il modello può essere utilizzato in produzione e successivamente monitorato.

Il machine learning permette di affrontare problemi nei quali sarebbe difficile definire manualmente tutte le regole necessarie. Può essere applicato a domini molto diversi, come riconoscimento delle immagini, elaborazione del linguaggio, medicina, finanza, videogiochi e sistemi di raccomandazione (Russell & Norvig, 2021).

Presenta però anche alcuni limiti.
Le prestazioni dipendono fortemente dalla qualità e dalla rappresentatività dei dati; alcuni modelli possono richiedere notevoli risorse computazionali; modelli complessi possono essere difficili da interpretare; inoltre errori o distorsioni presenti nei dati possono riflettersi nei risultati prodotti dal sistema.

Il machine learning viene tradizionalmente suddiviso in tre grandi famiglie.

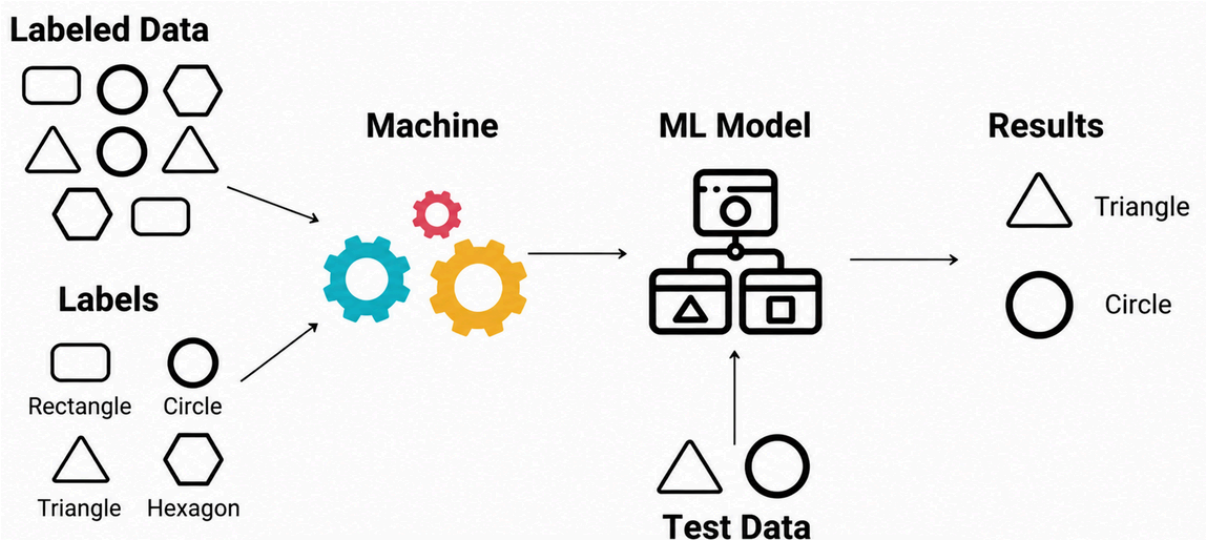
- **Nell'apprendimento supervisionato**, il sistema apprende da esempi nei quali a ogni input è associato un output di riferimento. Per esempio, può imparare a distinguere immagini di gatti e cani oppure a stimare il prezzo di una casa. Tra i metodi utilizzati troviamo regressione lineare, alberi decisionali e reti neurali.

- **Nell'apprendimento non supervisionato**, invece, i dati non possiedono output o etichette di riferimento. L'algoritmo cerca quindi strutture, relazioni o gruppi presenti nei dati. Applicazioni tipiche sono il clustering, la riduzione della dimensionalità e il rilevamento delle anomalie. Tra i metodi utilizzati troviamo k-means, PCA e autoencoder.
- **Nell'apprendimento per rinforzo**, infine, un agente interagisce con un ambiente, compie azioni e riceve ricompense, eventualmente anche negative. Attraverso l'esperienza cerca di apprendere una strategia che massimizzi la ricompensa cumulativa attesa nel tempo. Un esempio è un robot che impara a muoversi in un ambiente oppure un agente che apprende a giocare. Tra gli algoritmi classici troviamo il Q-learning, mentre sistemi più complessi possono utilizzare tecniche di deep reinforcement learning (Sutton & Barto, 2018).

Queste tre famiglie condividono l'idea generale dell'apprendimento, ma differiscono soprattutto per il tipo di informazione attraverso cui il sistema riceve indicazioni su ciò che deve apprendere: output conosciuti nel supervisionato, struttura dei dati nel non supervisionato e ricompense derivanti dall'interazione nel reinforcement learning.

Apprendimento supervisionato

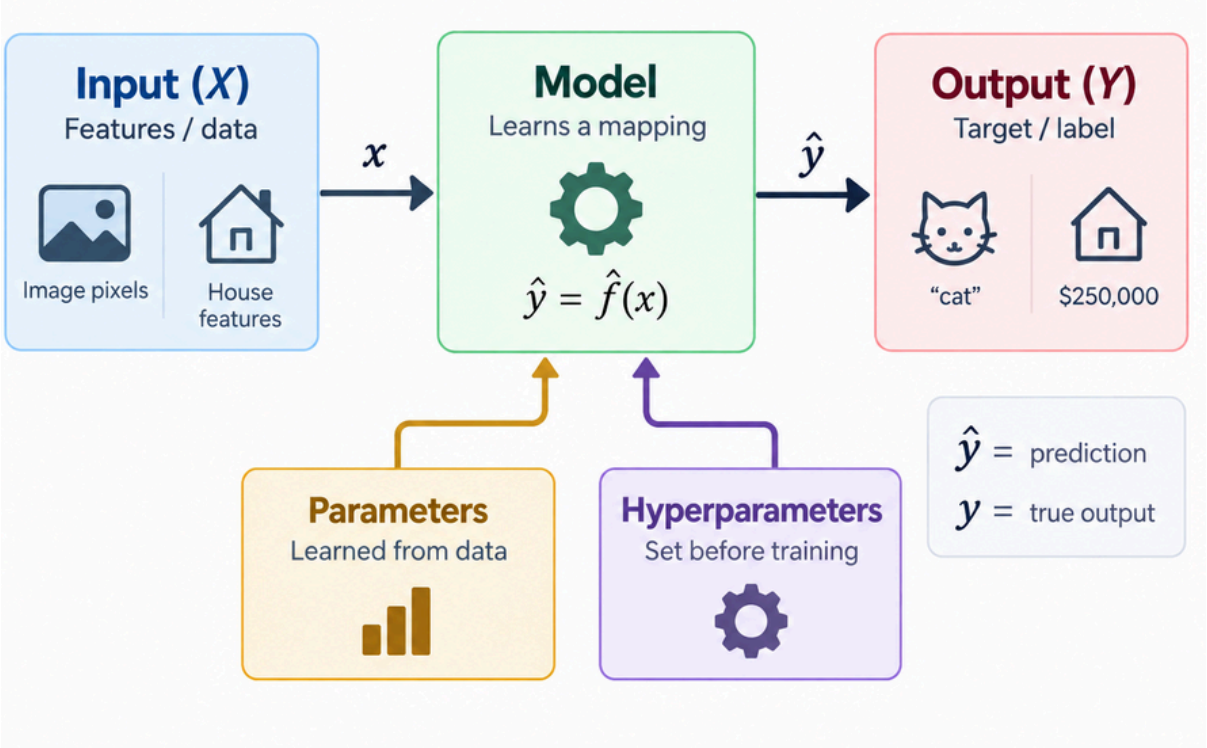
Nell'apprendimento supervisionato, un algoritmo impara a svolgere un compito a partire da un insieme di esempi etichettati, cioè dati di input associati al relativo output corretto o atteso (James et al., 2021).



L'obiettivo è costruire un modello parametrizzato capace di apprendere una relazione tra input e output e, soprattutto, di generalizzare, cioè di produrre risultati corretti anche quando riceve nuovi dati che non erano presenti durante l'addestramento.

In forma semplificata, un problema di apprendimento supervisionato comprende:

- Input (*X*): le caratteristiche che descrivono il fenomeno, per esempio i pixel di un'immagine, le caratteristiche di un'abitazione o le misure provenienti da un sensore.
- Output (*Y*): il valore o l'etichetta che il modello deve imparare a prevedere, per esempio "gatto" o "cane", il prezzo di una casa oppure la presenza o meno di spam.
- Modello (*f*): una funzione parametrizzata che cerca di rappresentare la relazione tra input e output. Dato un input *x*, il modello produce una previsione $\hat{y} = \hat{f}(x)$.
    - $\hat{y}$ è la previsione del modello.
    - y è l'output corretto. Ovviamente il modello fa una giusta previsione se $\hat{y} = y$.
    - $f(x)$ è la funzione ideale che associa correttamente *x* a *y*.
    - $\hat{f}(x)$ è la funzione calcolata dal modello che associa *x* a $\hat{y}$.
- Parametri: i valori interni del modello che vengono modificati automaticamente durante l'addestramento sulla base dei dati.
- Iperparametri: valori utilizzati per configurare il modello o il processo di apprendimento, ma che non vengono appresi direttamente dai dati. Possono essere, per esempio, il learning rate, la profondità massima di un albero decisionale o il numero di iterazioni (Goodfellow et al., 2016).



L'algoritmo non conosce in anticipo la relazione che lega gli input agli output.
Durante l'addestramento cerca quindi di trovare valori dei parametri che permettano al modello di produrre previsioni il più possibile vicine agli output osservati negli esempi.

Si può paragonare questo processo, con una certa semplificazione, a uno studente che riceve molti esercizi già risolti: osservando esempi e risposte cerca di individuare delle regolarità che gli permettano successivamente di affrontare esercizi nuovi.

Il dataset è la collezione di esempi utilizzata per sviluppare e valutare il modello. Nell'apprendimento supervisionato ogni esempio contiene almeno un input e il relativo output corretto di riferimento (James et al., 2021).

Per esempio, se vogliamo costruire un sistema capace di riconoscere animali nelle fotografie, il dataset potrebbe essere costituito da immagini associate a etichette come "gatto", "cane" o "uccello".

Una singola coppia input-output viene chiamata esempio, istanza o campione.

Esempio
Un'istanza potrebbe essere:
- Input: fotografia di un barboncino.
- Output: "cane".

Le caratteristiche utilizzate per rappresentare l'input vengono chiamate **features**.
Non vanno confuse con i parametri del modello:
- Le feature appartengono ai dati.
- I parametri sono valori interni del modello che vengono appresi durante l'addestramento (James et al., 2021).

Consideriamo un problema nel quale vogliamo prevedere il prezzo di un'abitazione:

Superficie (mq)	Numero stanze	Quartiere	Prezzo
80	3	Centro	250.000
120	4	Periferia	180.000
60	2	Centro	200.000

In questo caso superficie, numero di stanze e quartiere costituiscono le caratteristiche dell'input, mentre il prezzo costituisce l'output da prevedere.
Ogni riga rappresenta un esempio.

Per verificare che il modello non impari soltanto a riprodurre gli esempi che ha già visto, il dataset viene normalmente suddiviso in diversi sottoinsiemi.
- ***Training set***: viene utilizzato per addestrare il modello. L'algoritmo usa questi esempi per modificare i parametri interni. Il training set è quindi il materiale dal quale il modello apprende direttamente.
- **Validation set**: viene utilizzato durante lo sviluppo per confrontare configurazioni, scegliere gli iperparametri e controllare la capacità di generalizzazione. Il validation set non viene utilizzato per apprendere direttamente i parametri del modello. Serve invece a prendere decisioni sul processo di sviluppo, per esempio confrontare modelli differenti, scegliere un iperparametro o decidere quando interrompere l'addestramento.

- **Test set**: viene mantenuto separato fino alla fine e serve a valutare le prestazioni finali del modello su dati non utilizzati né per l'addestramento né per le scelte progettuali. Il test set rappresenta una verifica finale. Se venisse continuamente utilizzato per modificare il modello o scegliere gli iperparametri, non costituirebbe più una valutazione indipendente (James et al., 2021).

Esempio
Immaginiamo di avere 10.000 email già etichettate come "spam" o "non spam".
Prima di iniziare lo sviluppo dividiamo il dataset, per esempio, in:
- Training set: 7.000 email. Serve per apprendere i parametri del modello.
- Validation set: 1.500 email. Serve per confrontare configurazioni e scegliere gli iperparametri.
- Test set: 1.500 email. Viene tenuto da parte fino alla fine e serve per valutare il modello definitivo.

Il workflow potrebbe essere questo:
Dataset → divisione → training → validation → tuning → nuovo training → validation → modello scelto → test finale

Durante il training, il modello analizza le 7.000 email del training set. Per ogni esempio produce una previsione, la confronta con l'etichetta corretta e modifica i propri parametri per ridurre la funzione di perdita.
Supponiamo, per esempio, di utilizzare come modello un albero decisionale.
Dopo l'addestramento otteniamo:
- Accuratezza training: 96%
- Accuratezza validation: 78%

La grande differenza suggerisce che il modello potrebbe essere in overfitting.
A questo punto utilizziamo il validation set per provare un diverso iperparametro, per esempio riducendo la profondità massima dell'albero (un iperparametro del modello).

Addestriamo nuovamente il modello sul training set:
- Profondità massima = 5
- Accuratezza training: 90%
- Accuratezza validation: 88%

Proviamo anche:
- Profondità massima = 8
- Accuratezza training: 93%
- Accuratezza validation: 90%

e:
- Profondità massima = 15
- Accuratezza training: 98%
- Accuratezza validation: 84%

Il validation set ci suggerisce quindi che, tra queste configurazioni, profondità 8 generalizza meglio.

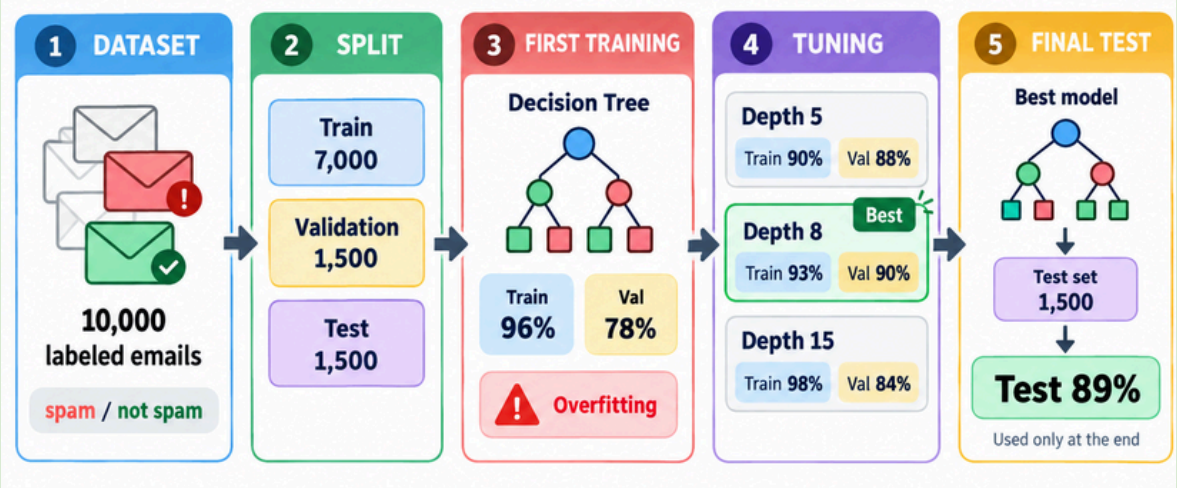
A questo punto scegliamo quella configurazione e consideriamo terminata la fase di sviluppo.

Solo ora utilizziamo il test set, che il modello non ha mai utilizzato né per apprendere i parametri né per scegliere gli iperparametri.
Supponiamo di ottenere: Accuratezza sul test set: 89%

Possiamo quindi stimare che il modello abbia una capacità di generalizzazione vicina a quella osservata sul validation set.

La cosa importante è che i tre insiemi svolgono ruoli diversi:
- Training set → "Come modifico i parametri del modello?"
- Validation set → "Quale configurazione del modello scelgo?"
- Test set → "Quanto funziona realmente il modello scelto su dati nuovi?"

C'è poi una regola importante: il test set non deve essere utilizzato per scegliere il modello.
Se guardassimo il risultato sul test e poi cambiassimo continuamente gli iperparametri per migliorarlo, il test diventerebbe di fatto un altro validation set e non sarebbe più una valutazione indipendente.



Durante l'addestramento è necessario stabilire quanto le previsioni prodotte dal modello siano lontane dai risultati attesi.
Per questo viene utilizzata una **funzione di perdita (loss function)**, che assegna un valore all'errore commesso dal modello (Goodfellow et al., 2016).
In forma generale:

$$L(\hat{y},\ y)$$

dove:
- $\hat{y}$ la previsione del modello.
- y rappresenta l'output corretto.

L'algoritmo di apprendimento modifica i parametri del modello cercando di ridurre la perdita sugli esempi del training set (Goodfellow et al., 2016).

Lo scopo dell'addestramento non è semplicemente ottenere risultati perfetti sugli esempi già osservati, ma costruire un modello capace di comportarsi correttamente anche su nuovi dati.

Questa capacità viene chiamata **generalizzazione**.
Un modello nella generalizzazione può incontrare due problemi opposti.

- Nell'underfitting, il modello è troppo semplice o non è stato addestrato sufficientemente e non riesce a rappresentare bene nemmeno le regolarità presenti nel training set. Le sue prestazioni risultano quindi scarse già sui dati di addestramento.
- Nell'overfitting, invece, il modello si adatta eccessivamente alle caratteristiche specifiche del training set. Può ottenere prestazioni molto elevate sui dati già osservati ma peggiorare significativamente quando riceve nuovi dati.

Per questo motivo un errore molto basso sul training set non dimostra, da solo, che il modello abbia imparato correttamente. Durante lo sviluppo si utilizza il validation set per controllare la generalizzazione, mentre il test set viene utilizzato alla fine per stimare le prestazioni del modello scelto (James et al., 2021).

Esempio
Supponiamo di voler costruire un modello capace di riconoscere automaticamente se in una fotografia è presente un gatto, un cane oppure un uccello.

Il dataset contiene molte immagini già etichettate con la categoria corretta. Ogni immagine costituisce un input, mentre la relativa etichetta costituisce l'output di riferimento.

Il dataset potrebbe, per esempio, essere suddiviso in:

- 70% training set, utilizzato per apprendere i parametri del modello.
- 15% validation set, utilizzato per confrontare configurazioni e scegliere gli iperparametri.
- 15% test set, utilizzato per la valutazione finale.

Queste percentuali sono soltanto un esempio: la suddivisione effettiva dipende dal problema e dalla quantità di dati disponibili.

Durante il processo di sviluppo e utilizzo del modello:

- Analizza le immagini del training set, produce una previsione e la confronta con l'etichetta corretta.
- La funzione di perdita misura quanto la previsione differisce dal risultato atteso e l'algoritmo modifica progressivamente i parametri del modello per ridurre l'errore.
- Durante la validazione vengono confrontate diverse configurazioni del modello e vengono scelti gli iperparametri che permettono di ottenere prestazioni migliori su dati non utilizzati direttamente per l'addestramento.
- Una volta terminata la fase di sviluppo, il modello viene valutato sul test set, che permette di stimare quanto sia capace di generalizzare su immagini mai utilizzate prima.
- Terminato il processo, il modello può ricevere una nuova fotografia e classificarla, per esempio, come "gatto", "cane" oppure "uccello".

Due delle principali tipologie di problemi affrontati attraverso l'apprendimento supervisionato sono classificazione e regressione (James et al., 2021).

Problema	**Output**	**Esempio**
Classificazione	Categoria o classe discreta	Stabilire se un'email è "spam" o "non spam"
Regressione	Valore numerico continuo	Stimare il prezzo di un appartamento

L'apprendimento supervisionato viene utilizzato in numerosi ambiti. Un sistema di riconoscimento vocale può essere addestrato attraverso registrazioni associate alle relative trascrizioni, mentre un sistema di analisi di immagini mediche può utilizzare immagini associate a valutazioni precedentemente verificate da specialisti.

Uno dei principali vantaggi dell'apprendimento supervisionato è che, quando dispone di dati adeguati e di una buona progettazione del modello, può raggiungere prestazioni elevate anche in problemi complessi.
La necessità di disporre di output di riferimento rappresenta però anche uno dei suoi principali limiti.

- In alcuni problemi la produzione di grandi quantità di dati correttamente etichettati può essere costosa e richiedere molto lavoro.
- Inoltre, se il dataset non è sufficientemente rappresentativo oppure il modello viene addestrato in modo inadeguato, possono verificarsi problemi di generalizzazione e overfitting.

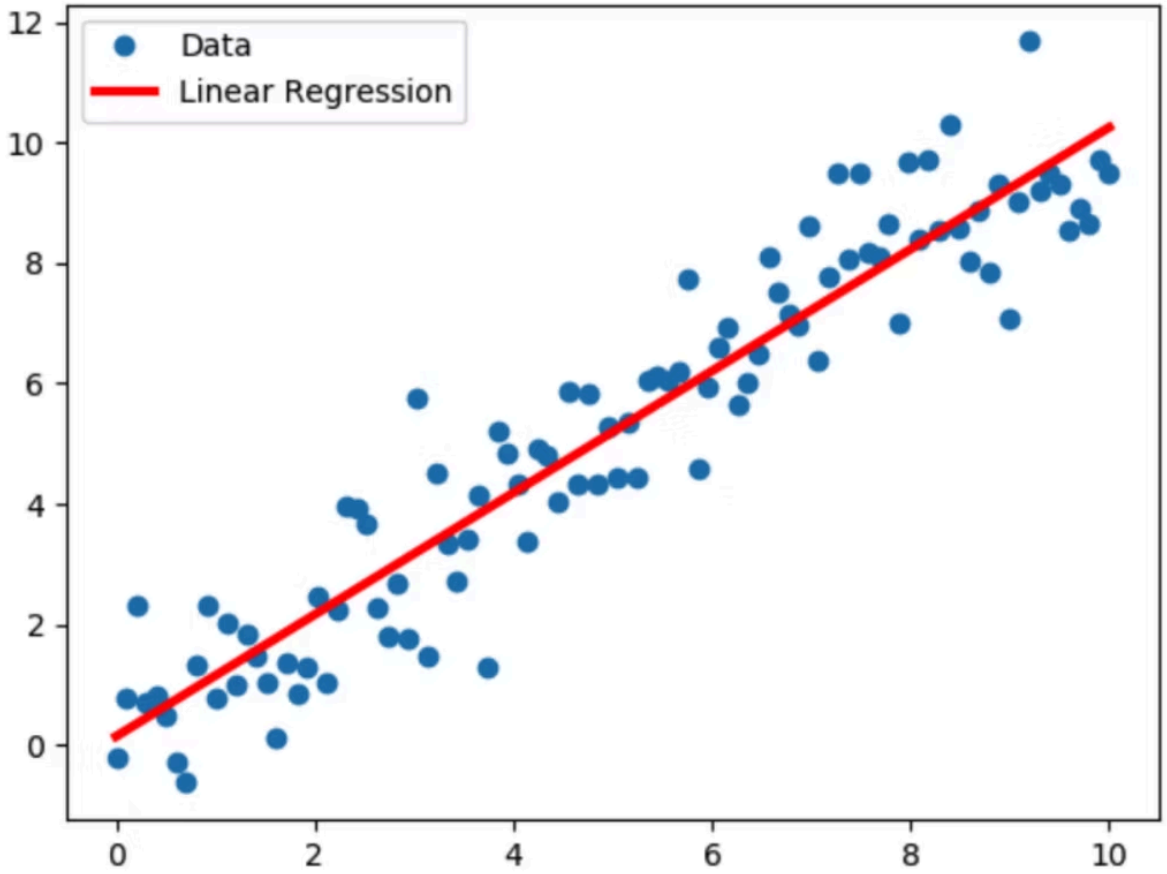
Il principio fondamentale dell'apprendimento supervisionato può quindi essere sintetizzato così:
Esempi input-output → apprendimento dei parametri → modello → generalizzazione su nuovi input.

Regressione

La regressione è un problema di apprendimento supervisionato in cui l'output è continuo.
In questo capitolo si esamina un algoritmo di regressione chiamato **metodo dei minimi quadrati (Ordinary Least Squares, *OLS*)** (James et al., 2021)**.**

Questo metodo è un modo semplice ed efficace per stimare la funzione che meglio approssima il rapporto tra input e output.
Di seguito viene descritto questo metodo che, per semplicità, verrà considerato nel caso più semplice: esempi con una sola caratteristica in input.

Il metodo assume una relazione lineare tra input e output, cioè tra una variabile dipendente y (l'output) e una variabile indipendente x (l'input).
A causa di questa linearità, l'algoritmo impiega una **linea di regressione**, nota anche come **linea di adattamento**.



Nell'immagine ogni punto rappresenta un esempio del dataset: sull'asse orizzontale è riportato il valore dell'input *x*, mentre sull'asse verticale è riportato il corrispondente output *y*.

L'obiettivo dell'algoritmo è trovare la retta che descrive nel modo migliore la relazione tra input e output.
Questa retta non deve necessariamente passare per tutti i punti, ma deve approssimarli complessivamente nel modo più efficace.

Una volta determinata la retta di regressione, è possibile fornire un nuovo valore di input *x* e utilizzare la retta per ottenere il corrispondente valore previsto $\hat{y}$.

Dato un dataset con "*n*" esempi, si definisce un esempio come (x_i, y_i) dove:

- x è l'input che per semplicità abbiamo detto possiede solo una caratteristica.
- i è l'indice dell'esempio.
- y è l'output dell'esempio.

La retta di regressione ha equazione $\hat{y} = a + bx$
dove:

- a indica dove la retta intercetta (dove la retta taglia l'asse verticale).
- b indica la pendenza (quanto cresce y se aumenta x).

Quindi dato l'esempio x_i possiamo calcolare il suoi output predetto dalla rettaretta come:

$$\widehat{y}_i = a + bx_i$$

Il valore osservato y_i non coincide necessariamente con il valore previsto dalla retta di regressione $\widehat{y}_i$.
La differenza tra questi due valori prende il nome di **errore o residuo dell'esempio** (James et al., 2021):

$$e_i = y_i - \widehat{y}_i = y_i - (a + bx_i)$$

Se si mette la retta a caso, alcuni errori saranno grandi, altri piccoli.
Non vogliamo la retta perfetta (non esiste), ma quella che minimizza gli errori, cioè la retta che minimizza la somma dei quadrati dei residui (per non far annullare errori positivi e negativi, si eleva al quadrato il residuo):

$$E = \sum_i^n (y_i - \widehat{y}_i)^2 = \sum_i^n (y_i - (a + bx_i))^2 =$$

$$= (y_1 - (a + bx_1))^2 + (y_2 - (a + bx_2))^2 + \ldots.$$

Questa è la "funzione errore", la somma di tutti i residui al quadrato, da rendere il più piccola possibile (James et al., 2021).

Per risolvere basta trovare a e b in modo che minimizzino la funzione E.
Risolvendo questo problema di minimizzazione si trova questa formula (dove $\bar{x}$ è la media dei valori di input e $\bar{y}$ la media dei valori di output) (James et al., 2021):

- $b = \frac{\sum_i^n (x_i - \bar{x})(y_i - \bar{y})}{\sum_i^n (x_i - \bar{x})^2}$
- $a = \bar{y} - b\bar{x}$

Quindi sostituendo i valori di a e b e semplificando, la retta dei minimi quadrati è questa:

$$\widehat{y} = a + bx = \bar{y} + \frac{\sum_i^n (x_i - \bar{x})(y_i - \bar{y})}{\sum_i^n (x_i - \bar{x})^2}(x - \bar{x})$$

Esempio
Dati gli esempi:

$(1,\ 2), (2,\ 3), (3,\ 2.5), (4,\ 4)$
Trovare la linea di regressione.

Si Calcolano le medie di *x* e *y*.

- $\bar{x} = (1 + 2 + 3 + 4)/4 = 2.5$
- $\bar{y} = (1 + 2 + 2.5 + 4)/4 = 2.875$

Si calcolano i parametri:

- $b = \frac{(1-2.5)(2-2.875)+(2-2.5)(3-2.875)+(3-2.5)(2.5-2.875)+(4-2.5)(4-2.875)}{(1-2.5)^2+(2-2.5)^2+(3-2.5)^2+(4-2.5)^2} = 0.55$
- $a = 2.875 - (0.55 * 2.5) = 1.5$

La retta di regressione stimata è:
$y = 1.5 + 0.55x$

Quando si hanno più caratteristiche in input, il ragionamento resta lo stesso: si vuole una funzione che riduca al minimo gli errori, ma invece di una retta su un piano 2D, si ha un piano o in generale un iperpiano in uno spazio con più dimensioni (James et al., 2021).

Quindi la retta: $y = a + bx$, con più parametri diventa:

$$y = a + b_1x_1 + b_2x_2 + \ldots + b_nx_n$$

Dove:

- $x_1, x_2, \ldots, x_n$ sono le variabili di un esempio con *n* caratteristiche.
- $b_1, b_2, \ldots, b_n$ sono i parametri che descrivono il contributo della variabile alla previsione del modello.

Classificazione

La classificazione è un problema di apprendimento supervisionato in cui l'output da prevedere non è un valore numerico continuo, ma una classe appartenente a un insieme finito di categorie (James et al., 2021).

Per esempio, un modello può essere addestrato per stabilire se un'email appartiene alla classe "spam" oppure "non spam", se una pianta è "commestibile" oppure "velenosa", oppure se un cliente appartiene alla classe "acquista" oppure "non acquista".

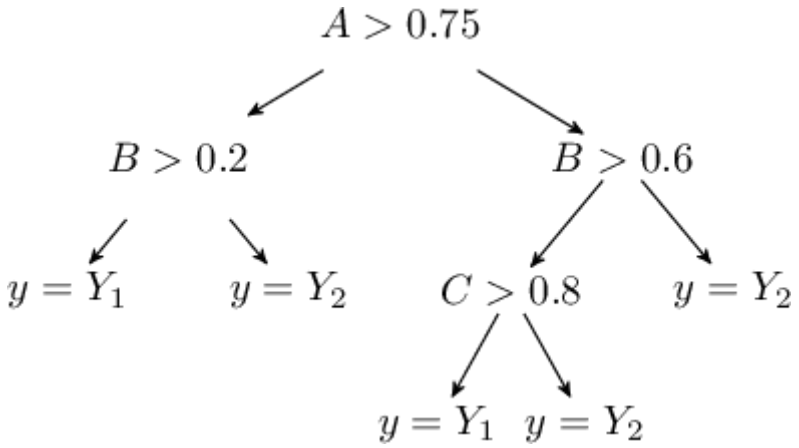
Nella classificazione, una classe è quindi una delle categorie che il modello può assegnare a un input.
In un sistema di riconoscimento delle immagini, per esempio, "gatto", "cane" e "uccello" possono rappresentare tre classi differenti.

Uno dei modelli utilizzati per affrontare problemi di classificazione è **l'albero decisionale** (decision tree).

Un albero decisionale classifica un input attraverso una sequenza di decisioni basate sui valori delle sue caratteristiche.
Durante l'addestramento, un algoritmo costruisce progressivamente l'albero individuando le caratteristiche e le regole che permettono di separare nel modo più efficace gli esempi appartenenti a classi differenti (Quinlan, 1986).

Il funzionamento può essere rappresentato come una successione di domande:
Caratteristiche dell'input → decisioni → classe prevista

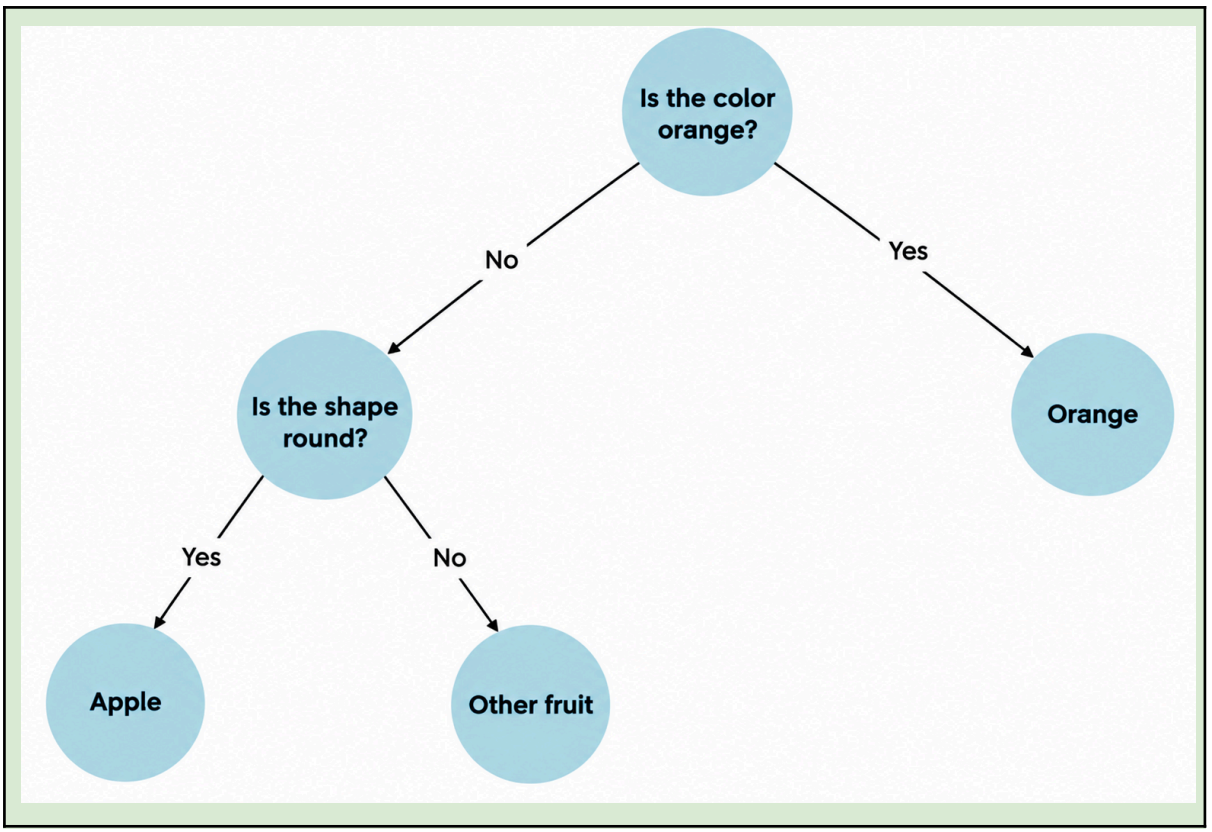
Ogni decisione viene effettuata in un nodo dell'albero. Il nodo verifica una condizione relativa a una caratteristica dell'input e, in base al risultato, indirizza l'esempio verso uno dei rami successivi.



Per esempio vediamo questa condizione "Età < 18?".
- Se la risposta è sì, l'esempio segue un ramo.
- Se la risposta è no, ne segue un altro.

Questa divisione in due rami viene chiamata **split**, cioè divisione del nodo.
Attraverso uno split, gli esempi presenti in un nodo vengono separati in due o più gruppi sulla base del valore di una caratteristica (Quinlan, 1986).

Esempio
Consideriamo, per esempio, un modello che deve classificare alcuni frutti come mela, arancia oppure altro frutto. Il dataset potrebbe contenere caratteristiche come colore, forma, peso e dimensione.
L'albero potrebbe iniziare verificando:
Il colore è arancione?
Se la risposta è sì, l'esempio potrebbe essere indirizzato verso una foglia associata alla classe "arancia". Se la risposta è no, potrebbe essere effettuata una seconda verifica:
La forma è rotonda?
Il processo continua fino a raggiungere una previsione finale.



Un albero decisionale è composto principalmente da:

- Nodo radice: È il primo nodo dell'albero e contiene la prima regola di decisione.
- Nodi interni: Contengono le regole utilizzate per dividere ulteriormente gli esempi.
- Rami: Rappresentano i possibili risultati di una decisione e collegano un nodo ai nodi successivi.
- Nodi foglia: Rappresentano l'uscita finale dell'albero e contengono la classe prevista o una distribuzione delle classi.
- Nodo puro: Contiene esempi appartenenti tutti alla stessa classe.
- Nodo impuro: Contiene esempi appartenenti a classi differenti.

Durante l'addestramento, l'algoritmo cerca di scegliere split che producano nodi sempre più omogenei, cioè composti prevalentemente da esempi appartenenti alla stessa classe (Quinlan, 1986).

La quantità di mescolanza tra classi presenti in un nodo viene chiamata impurità.
Per esempio:

- Un nodo contenente 50 esempi appartenenti tutti alla classe "mela" è puro.
- Un nodo contenente 25 mele e 25 arance è invece molto impuro.

L'algoritmo cerca quindi, a ogni passaggio, una caratteristica e una regola di split che permettano di ridurre l'impurità dei nodi risultanti.

Esistono diversi criteri per valutare la qualità di uno split.
Un possibile criterio è il guadagno informativo (information gain), che misura quanto una divisione riduce l'incertezza sulle classi (Quinlan, 1986).

Non tutti i nodi foglia devono necessariamente essere puri.
Una foglia potrebbe, per esempio, contenere il 70% di esempi appartenenti alla classe “mela” e il 30% alla classe “altro frutto”.
In questo caso il modello può assegnare la classe prevalente oppure utilizzare la distribuzione degli esempi per stimare una probabilità (James et al., 2021).

La crescita dell’albero deve però essere controllata. Se l’albero continua a creare nuove divisioni fino a separare quasi perfettamente tutti gli esempi del training set, può diventare eccessivamente complesso e adattarsi anche a caratteristiche casuali o poco significative dei dati.
In questo caso si verifica overfitting: il modello ottiene prestazioni molto elevate sui dati di addestramento, ma può generalizzare peggio su nuovi esempi.

Per limitare questo problema è possibile stabilire criteri di arresto, per esempio una profondità massima dell’albero, un numero minimo di esempi necessario per effettuare uno split oppure un miglioramento minimo richiesto per creare una nuova divisione.
Questi valori sono iperparametri del modello e possono essere scelti utilizzando il validation set (James et al., 2021).

Apprendimento non supervisionato

L’apprendimento non supervisionato è una famiglia di tecniche di machine learning in cui l’algoritmo analizza dati privi di etichette di riferimento con l’obiettivo di individuare strutture, regolarità o relazioni presenti nei dati (James et al., 2021).

A differenza dell’apprendimento supervisionato, nel quale a ogni esempio è associato un output corretto da prevedere, nell’apprendimento non supervisionato vengono forniti soltanto gli input. L’algoritmo non conosce quindi in anticipo le categorie o i risultati che dovrebbe ottenere, ma cerca di individuare autonomamente una possibile organizzazione interna dei dati.

L’apprendimento non supervisionato non mira quindi necessariamente a prevedere un valore noto, ma a rivelare strutture latenti nei dati, cioè configurazioni che non sono state specificate esplicitamente dal progettista.
Può essere utilizzato, per esempio, per raggruppare elementi simili, semplificare dataset con molte caratteristiche oppure individuare osservazioni anomale.

Tra le principali applicazioni troviamo:
- Clustering: raggruppa gli esempi in insiemi chiamati cluster, cercando di fare in modo che gli elementi appartenenti allo stesso gruppo siano più simili tra loro rispetto agli elementi appartenenti ad altri gruppi.
- Riduzione della dimensionalità: riduce il numero di caratteristiche utilizzate per rappresentare i dati, cercando di preservare quanto più possibile l’informazione rilevante.

- Rilevamento delle anomalie: può essere utilizzato per individuare esempi che presentano caratteristiche significativamente differenti rispetto alla maggior parte dei dati, come transazioni insolite, malfunzionamenti o comportamenti anomali.

Esempio
Supponiamo di avere un dataset contenente migliaia di clienti di un e-commerce. Per ogni cliente conosciamo alcune caratteristiche, come la spesa media, la frequenza degli acquisti e le categorie di prodotti acquistate, ma non disponiamo di categorie predefinite alle quali assegnarli.

- Un algoritmo di clustering potrebbe individuare gruppi di clienti con comportamenti simili. L'algoritmo non attribuisce necessariamente un significato ai gruppi trovati: sarà successivamente l'analista a interpretarli, per esempio identificandoli come clienti ad alta spesa, clienti occasionali o clienti particolarmente fedeli a determinate categorie.
- Un algoritmo di riduzione della dimensionalità potrebbe invece trasformare le numerose caratteristiche dei clienti in un numero minore di componenti, rendendo più semplice visualizzare o analizzare le principali relazioni presenti nei dati.
- Un metodo di rilevamento delle anomalie potrebbe infine individuare un account con un comportamento molto diverso da quello abituale, segnalando per esempio una possibile frode.

L'assenza di etichette rende però l'apprendimento non supervisionato più difficile da valutare. Nel supervisionato possiamo confrontare direttamente la previsione del modello con l'output corretto; nel non supervisionato, invece, spesso non esiste una ground truth esplicita con cui confrontare il risultato.

La qualità delle strutture individuate deve quindi essere valutata attraverso criteri specifici del problema, misure matematiche oppure attraverso l'interpretazione dei risultati (James et al., 2021).
Anche gli iperparametri possono influenzare significativamente il risultato. Nel clustering, per esempio, alcuni algoritmi richiedono di stabilire in anticipo quanti gruppi cercare oppure quale misura utilizzare per determinare la somiglianza tra gli esempi.
Inoltre, le strutture individuate dall'algoritmo non possiedono necessariamente un significato immediatamente comprensibile: un gruppo matematicamente ben definito può richiedere un'analisi successiva per essere interpretato nel contesto reale del problema (James et al., 2021).

La differenza fondamentale tra apprendimento supervisionato e non supervisionato può quindi essere sintetizzata così:

- Apprendimento supervisionato → conosciamo gli output di riferimento e insegniamo al modello a prevederli.
- Apprendimento non supervisionato → non disponiamo di output di riferimento e utilizziamo l'algoritmo per esplorare la struttura interna dei dati.

Clustering

Il clustering è una tecnica di apprendimento non supervisionato che raggruppa esempi simili in insiemi chiamati **cluster**.
L'obiettivo è ottenere gruppi nei quali gli esempi siano più simili tra loro rispetto agli esempi appartenenti ad altri gruppi, secondo una determinata misura di distanza o somiglianza (James et al., 2021).

- Input: sono i dati, senza etichette.
- Output: una partizione del dataset in gruppi.

Esempio
Di seguito sono riportati alcuni esempi pratici in cui il clustering può essere utilizzato:

- Marketing: segmentare i clienti in base a comportamenti d'acquisto.
- Biologia: raggruppare geni con profili di espressione simili.
- Visione artificiale: riconoscere oggetti simili in immagini senza etichette.

Uno degli algoritmi più semplici e famosi per realizzare il clustering si chiama **k-means**.
Il k-means è un algoritmo che prende dati senza etichette e li organizza in gruppi, basandosi sulla vicinanza (Arthur & Vassilvitskii, 2007).
K-means funziona con questa idea:

- Abbiamo dei punti (i nostri dati).
- Questi dati devono essere divisi in "*k*" gruppi. Il numero di cluster *k* è un iperparametro che deve essere stabilito prima dell'esecuzione dell'algoritmo.
- Ogni gruppo è rappresentato dal suo centro, chiamato **centroide**.
- Ogni punto viene assegnato al gruppo con il centro più vicino.

Di seguito è descritto lo svolgimento di k-means:

Premessa
Si ha un insieme di dati di numero "*n*" (*X* è l'insieme dei dati e *x* è un dato):
$X = \{x_1, x_2, \ldots, x_n\}$

Ogni dato ha un certo numero di caratteristiche di numero "*d*".
Poiché k-means utilizza le distanze tra i punti, quando le caratteristiche hanno scale molto differenti è spesso opportuno normalizzarle o standardizzarle (tutti i valori sono nella stessa scala numerica) (James et al., 2021).
$x_i = \{x_{i,1}, x_{i,2}, \ldots, x_{i,d}\}$

I cluster devono essere di numero "*k*". Quindi l'insieme dei cluster è:
$K = \{C_1, C_2, \ldots, C_k\}$

L'obiettivo di k-means è ottenere cluster compatti: i punti appartenenti a uno stesso cluster devono trovarsi il più vicino possibile al relativo centroide. Per questo l'algoritmo cerca di

minimizzare la somma delle distanze quadratiche tra ogni punto e il centroide del cluster a cui appartiene (Arthur & Vassilvitskii, 2007).
Per far questo K-means alterna due operazioni:

- Assegna ogni punto al centroide più vicino.
- Ricalcola ogni centroide come media dei punti assegnati.

Ripetendo queste operazioni, riduce progressivamente la distanza complessiva dei punti dai rispettivi centroidi fino alla convergenza (Arthur & Vassilvitskii, 2007).

Sotto vediamo i passaggi che consentono la realizzazione del K-means.

Inizializzazione dei centroidi
All'inizio i centroidi non esistono, quindi vanno scelti artificialmente.
Ci sono due modi principali per scegliere i *k* centroidi:

- Scelta casuale: vengono scelti a caso k punti a caso dal dataset. È il metodo più semplice, ma può dare risultati scarsi se i centroidi iniziali cadono troppo vicini o in posizioni "sbagliate".
- K-means++: è una strategia di inizializzazione molto utilizzata, progettata per scegliere centroidi iniziali ben distribuiti nello spazio dei dati. Si sceglie il primo centroide a caso tra i punti. Poi, ogni punto successivo viene scelto con probabilità proporzionale alla distanza quadrata dal centroide più vicino già scelto. Il risultato è che i centroidi iniziali partono sparsi bene nello spazio dei dati. Questa tecnica tende a favorire una convergenza più rapida e soluzioni migliori (Arthur & Vassilvitskii, 2007).

Alla fine si avrà un numero di centroidi pari al numero delle cluster scelto, cioè "*k*".
L'insieme dei centroidi è $\{\mu_1, \mu_2, \ldots, \mu_k\}$

Assegnazione
Ogni punto del dataset viene assegnato al centroide più vicino. In questo modo si formano i gruppi.
Dato un punto x_i e un insieme di centroidi $\{\mu_1, \mu_2, \ldots, \mu_k\}$, l'assegnazione del punto al cluster *j*-esimo avviene scegliendo l'indice *j* che minimizza la distanza tra tutti i centroidi, cioè:

$$c_i = argmin_{j \in \{1,\ldots,k\}} ||x_i - \mu_j|| = argmin_{j \in \{1,\ldots,k\}} \sqrt{\sum_{k=1}^{d} \left(x_{i,k} - \mu_k\right)^2}$$

Dove:

- c_i è il cluster assegnato al punto x_i
- $||x_i - \mu_j||$ è la distanza euclidea (norma 2) tra il punto di indice "*i*" e il centroide di indice "*j*"
- La funzione *argmin* significa "l'indice che minimizza". Più precisamente è l'insieme degli indici a cui l'argomento assume valore minimo (quindi equivale all'indice del centroide più vicino a x_i)
- $x_{i,k}$ è la caratteristica di indice "*k*" del punto di indice x_i.

- $\mu_{j,k}$ è la caratteristica di indice“*k*” del centroide μ_j.

Esempio
Se il *dato* ha due caratteristiche, la formula diventa:
$c_i = argmin_{j\in\{1,\dots,k\}} \sqrt{\left(x_{i,1} - \mu_{j,1}\right)^2 + \left(x_{i,2} - \mu_{j,2}\right)^2}$

Aggiornamento
Dopo aver assegnato tutti gli esempi al centroide più vicino, per ogni gruppo ricalcoliamo il centroide.

$$\mu_j = \frac{1}{|C_j|} \sum_{x_i \in C_j} x_i$$

Dove:
- μ_j è il centroide di indice *j*
- C_j sono gli esempi appartenenti al cluster di indice *j*
- $|C_j|$ è la cardinalità della classe K_j, cioè il numero degli esempi appartenenti a C_j
- $\sum_{x_i \in K_j}$ indica la somma di tutti gli esempi del cluster di indice *j*
- x_i è un esempio appartenente al cluster di indice *j*

Ripetizione
Ri-assegniamo i punti ai nuovi centroidi, poi ricalcoliamo i centroidi ancora. Si continua così finché i centroidi non cambiano più (o cambiano pochissimo) (Arthur & Vassilvitskii, 2007).

Esempio
Abbiamo 6 dati con una sola caratteristica:
$X = \{1, 2, 3, 10, 11, 12\}$
Si vuole dividere i punti in due cluster (K equivale a 2)

<u>Passo 1 – Inizializzazione dei centroidi</u>
Supponiamo di scegliere a caso i centroidi (scelta casuale):
- $\mu_1 = 2$
- $\mu_2 = 11$

<u>Passo 2 – Assegnazione</u>
Ogni punto va al centroide più vicino.
- I punti 1,2, 3 sono vicini a $\mu_1 = 2$

- I punti 10, 11, 12 sono vicini a $\mu_2 = 11$

Quindi i cluster sono:
- $S_1 = \{1, 2, 3\}$
- $S_2 = \{10, 11, 12\}$

Passo 3 – Aggiornamento dei centroidi
Calcoliamo la media di ogni cluster:
- $\mu_1 = \frac{(1+2+3)}{3} = 2$
- $\mu_2 = \frac{(10+11+12)}{3} = 11$

Passo 4 – Controllo di convergenza
In questo caso i centroidi non sono cambiati rispetto all'iterazione precedente, quindi l'algoritmo si ferma.

Risultato finale
- Cluster 1 con punti {1, 2, 3} con centroide 2
- Cluster 2 con punti {10,11, 12} con centroide 11

I punti sono stati divisi in due cluster ben separati secondo la distanza utilizzata: uno "vicino a 2" e uno "vicino a 11".

Punti di forza:
- È semplice da capire e da implementare.
- Funziona bene con dati che formano gruppi "rotondi" e di dimensioni simili.
- È veloce anche su dataset grandi.

Limiti
- Bisogna decidere il numero dei cluster in anticipo (non sempre sappiamo quanti cluster siano appropriati).
- Non funziona bene se i cluster hanno forme strane (es. a mezzaluna) o se hanno densità molto diverse (James et al., 2021).
- È sensibile agli outlier (punti molto lontani dalla massa dei dati).
- Centroidi iniziali differenti possono condurre a risultati differenti (Arthur & Vassilvitskii, 2007).
- Valori molto grandi o punti estremi possono modificare significativamente le distanze e la posizione dei centroidi (James et al., 2021).

Apprendimento per rinforzo

L'apprendimento per rinforzo (Reinforcement Learning, RL) è una delle principali famiglie del machine learning.

- A differenza dell'apprendimento supervisionato, l'agente non apprende da esempi associati a una risposta corretta.
- A differenza dell'apprendimento non supervisionato, inoltre, il suo obiettivo principale non è individuare strutture nascoste nei dati.

Nel reinforcement learning, un agente apprende attraverso l'interazione con un ambiente: osserva la situazione in cui si trova, compie delle azioni e riceve dall'ambiente delle ricompense (reward).
Attraverso l'esperienza cerca progressivamente di individuare una strategia che gli permetta di ottenere il massimo ritorno cumulativo atteso, cioè una quantità elevata di ricompense nel corso del tempo (Sutton & Barto, 2018).

Il reinforcement learning è quindi particolarmente utile nei problemi in cui conosciamo l'obiettivo da raggiungere, ma non sappiamo in anticipo quale sequenza di azioni permetta di raggiungerlo.

Il processo generale può essere rappresentato così:
Stato → Agente → Azione → Ambiente → Nuovo stato + Reward

L'agente osserva uno stato dell'ambiente, sceglie un'azione e, come conseguenza, l'ambiente può modificarsi.
L'agente osserva quindi un nuovo stato e riceve una ricompensa.

Esempio
Immaginiamo un robot che deve trovare l'uscita di un labirinto.
- Stati: Le diverse posizioni in cui il robot può trovarsi.
- Azioni: Muoversi verso l'alto, il basso, sinistra o destra.
- Reward: Per esempio +10 quando raggiunge l'uscita, −1 per ogni passo e −10 se compie un'azione particolarmente sfavorevole.
- Obiettivo: Imparare una strategia che permetta di raggiungere l'uscita ottenendo il maggior ritorno possibile.

All'inizio il robot non conosce necessariamente quali azioni siano convenienti. Attraverso molti tentativi può osservare le conseguenze delle proprie decisioni e modificare progressivamente il proprio comportamento.

Per comprendere il funzionamento del reinforcement learning è necessario distinguere alcuni elementi fondamentali.

Environment (Ambiente)
L'ambiente è il contesto, fisico o virtuale, nel quale l'agente opera. Riceve le azioni compiute dall'agente e determina le conseguenze di tali azioni, producendo un nuovo stato e una ricompensa (Sutton & Barto, 2018).

L'ambiente rappresenta quindi tutto ciò con cui l'agente interagisce durante il processo di apprendimento.

State (Stato)
Lo stato s rappresenta la situazione dell'ambiente in un determinato momento e contiene le informazioni rilevanti affinché l'agente possa prendere una decisione (Sutton & Barto, 2018).

Nel labirinto, per esempio, lo stato potrebbe essere rappresentato dalla posizione corrente del robot.

Action (Azione)
L'azione a è una delle possibili scelte che l'agente può effettuare in uno stato.
L'esecuzione di un'azione può modificare l'ambiente e portare l'agente in un nuovo stato.

Reward (Ricompensa)
La reward r è un valore numerico fornito dall'ambiente come conseguenza dell'interazione.
Può essere positiva, negativa oppure nulla.

La reward rappresenta il segnale attraverso il quale viene definito ciò che il sistema deve cercare di ottenere. Tuttavia, la singola reward rappresenta soltanto una conseguenza immediata: per valutare realmente una strategia è necessario considerare anche le ricompense che potranno essere ottenute successivamente.

Per questo l'obiettivo dell'agente non consiste semplicemente nel massimizzare la reward immediata, ma nel massimizzare il ritorno cumulativo atteso nel tempo (Sutton & Barto, 2018).

Policy (Politica o Strategia)
La policy π è la regola utilizzata dall'agente per scegliere quale azione compiere.
In forma semplice, una policy può associare a ogni stato una determinata azione:
$\pi(s) = a$

Una policy può però anche essere probabilistica, associando a uno stato una distribuzione di probabilità sulle diverse azioni disponibili (Sutton & Barto, 2018).

L'obiettivo del reinforcement learning è apprendere una policy che permetta all'agente di ottenere il massimo ritorno atteso possibile.

Value (Valore)
Il value stima quanto sia conveniente trovarsi in uno stato oppure compiere una determinata azione, considerando le ricompense che ci si aspetta di ottenere anche nel futuro (Sutton & Barto, 2018).

Non coincide quindi con la reward immediata.

Si distinguono principalmente due tipi di valore:
- Valore di stato $V(s)$: stima il ritorno futuro atteso a partire da uno stato.
- Valore stato-azione $Q(s, a)$: Stima il ritorno futuro atteso compiendo una determinata azione a nello stato s.

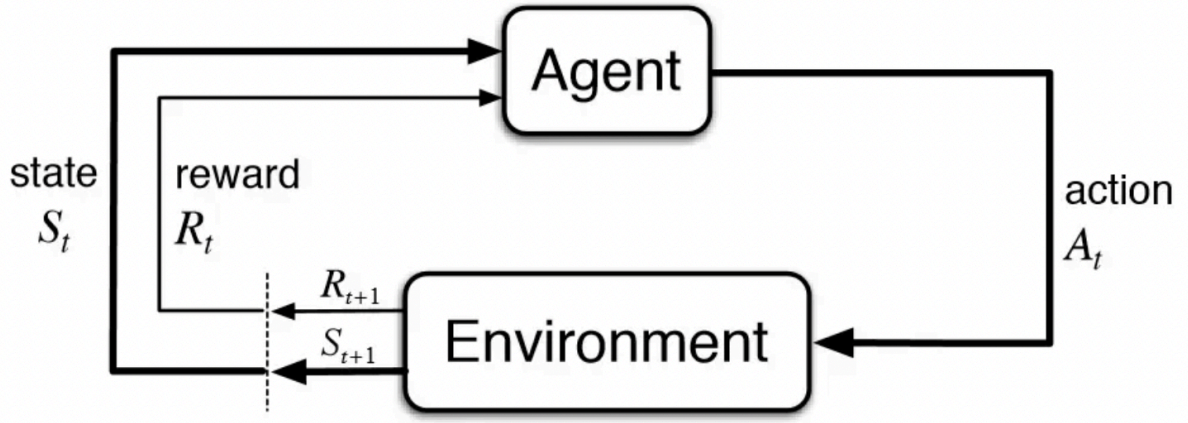
Esempio
Per esempio, nel labirinto uno stato può avere un valore elevato anche se non produce immediatamente una reward positiva, se da quella posizione esiste una buona probabilità di raggiungere successivamente l'uscita.

La policy e il value svolgono quindi funzioni differenti.

- La policy stabilisce quale azione compiere.
- Il value stima quanto sia conveniente uno stato o una coppia stato-azione.

In molti algoritmi di reinforcement learning le stime di value vengono utilizzate per migliorare progressivamente la policy (Sutton & Barto, 2018).

Esempio
Si immagini un agente che debba uscire da un labirinto. In uno stato dato, la policy può stabilire che l'azione migliore sia "muoversi a destra".
Il value, invece, non dice direttamente quale azione eseguire, ma misura quanto sia favorevole quella situazione oppure quanto sia promettente scegliere proprio quell'azione. Un passo che avvicina l'agente all'uscita può avere una reward immediata nulla o molto piccola, ma avere comunque un value alto, perché aumenta la probabilità di ottenere una ricompensa maggiore in futuro.



Uno schema che rappresenta un agente che, a partire da uno stato S_t, compie un'azione A_t; l'azione A_t modifica l'ambiente, che restituisce all'agente una ricompensa R_{t+1} e un nuovo stato S_{t+1}

Un problema fondamentale nasce però dal fatto che l'agente non conosce inizialmente tutte le conseguenze delle proprie azioni. Deve quindi trovare un equilibrio tra **esplorazione** ed **exploitation** (Sutton & Barto, 2018).

- L'esplorazione (exploration) consiste nel provare azioni alternative, comprese quelle che al momento non sembrano essere le migliori, per acquisire nuove informazioni sull'ambiente.

- Lo sfruttamento (exploitation) consiste invece nell'utilizzare le informazioni già apprese scegliendo l'azione che, sulla base delle conoscenze attuali, viene considerata più conveniente.

Una strategia che sceglie sempre l'azione con il valore stimato più alto viene chiamata **greedy** (Sutton & Barto, 2018).

Il compromesso tra esplorazione e sfruttamento è essenziale:
- Se l'agente esplora troppo, continua a sperimentare senza sfruttare sufficientemente ciò che ha già imparato.
- Se sfrutta troppo presto le conoscenze disponibili, può smettere di esplorare e non scoprire strategie migliori.

In molti problemi il processo di interazione viene organizzato in episodi.
Un episodio è una sequenza completa di interazioni che parte da uno stato iniziale e termina quando viene raggiunta una determinata condizione finale (Sutton & Barto, 2018).

Esempio
Nel labirinto, per esempio, un episodio potrebbe iniziare quando il robot viene posizionato all'ingresso e terminare quando raggiunge l'uscita oppure quando supera un numero massimo di passi.

Un episodio comprende quindi una sequenza del tipo:

$s_0 -> a_0 -> r_1 -> s_1 -> a_1 -> r_2 -> s_2 ->$…

Un episodio può terminare, per esempio, quando:
- Viene raggiunto uno stato terminale: il robot raggiunge l'uscita, il gioco termina oppure il braccio robotico completa il proprio compito.
- Si verifica una condizione di fallimento: il personaggio perde una vita oppure il sistema raggiunge una situazione dalla quale non può continuare.
- Viene raggiunto un limite: si supera un numero massimo di azioni o un determinato tempo.

Non tutti i problemi di reinforcement learning sono però episodici: esistono anche attività continue, nelle quali l'interazione tra agente e ambiente può proseguire senza una conclusione naturale (Sutton & Barto, 2018).
Nel caso dei problemi episodici, l'esperienza raccolta durante gli episodi può essere utilizzata dall'algoritmo per aggiornare le proprie stime e migliorare progressivamente il comportamento dell'agente.

Il reinforcement learning può quindi essere sintetizzato attraverso questo ciclo:
Osservare lo stato → scegliere un'azione → interagire con l'ambiente → ricevere nuovo stato e reward → apprendere dall'esperienza → migliorare la strategia.

Uno degli algoritmi più semplici per comprendere concretamente questo processo è il **Q-learning**, nel quale l'agente apprende progressivamente il valore delle diverse coppie stato-azione $Q(s, a)$.

Q-Learning

Il Q-learning è un algoritmo di reinforcement learning che permette a un agente di apprendere il valore delle diverse azioni disponibili in ciascuno stato, con l'obiettivo di massimizzare il ritorno cumulativo atteso nel tempo (Watkins & Dayan, 1992).

- L'agente esplora l'ambiente, riceve ricompense immediate e aggiorna progressivamente una tabella *Q* con i valori stato-azione corrispondenti.
- La tabella è composta da elementi $Q(s,\ a)$ che rappresentano la stima di quanto sia conveniente eseguire l'azione a nello stato s, considerando anche le ricompense future.
- Sotto determinate condizioni, il Q-learning può convergere verso i valori stato-azione ottimali. A partire da questi valori è possibile ricavare una policy ottimale scegliendo, in ogni stato, l'azione con il valore Q più alto (Watkins & Dayan, 1992).

La tabella, chiamata **Q-table**, è una tabella a due dimensioni organizzata in questo modo:

- Ogni riga corrisponde a uno stato.
- Ogni colonna corrisponde a un'azione.

Quindi, ogni casella della tabella corrisponde a:
Q(stato, azione) = la value di quell'azione in quello stato.

Esempio
La tabella Q-table per un robot in un labirinto può essere:

- Se il robot è davanti a un muro, la *Q* del movimento "contro il muro" sarà molto bassa.
- Se invece è vicino all'uscita, la *Q* per andare verso l'uscita sarà molto alta.

L'obiettivo del Q-learning è popolare la tabella in modo che l'agente abbia sempre a disposizione la migliore azione possibile per un particolare stato del problema.

Ma come funziona?
Per semplicità, possiamo inizializzare tutti i valori della Q-table a 0.
Ogni volta che un agente fa un'azione, succedono tre cose:

- Riceve una ricompensa (per esempio +10 se trova l'uscita o −1 se sbatte contro un muro).
- Si trova in un nuovo stato.
- Aggiorna il valore *Q* della coppia stato-azione relativa allo stato precedente e all'azione appena eseguita, tenendo conto della ricompensa ricevuta e del nuovo stato raggiunto.

La formula per aggiornare la value è (Watkins & Dayan, 1992):

$$Q(s,\ a)\ =\ (1\ -\ \alpha)*\ Q(s,\ a)\ +\ \alpha\left(r\ +\ \gamma max_{a'}Q\left(s',a'\ \right)\right)$$

Dove:

- *s* = stato attuale.
- *a* = azione eseguita.
- *r* = reward immediata ricevuta.
- s' = il nuovo stato raggiunto dopo l'azione.
- γ = "Discount factor", cioè quanto vengono considerate le ricompense future.
- α = "Learning rate", cioè quanto peso viene dato alla nuova esperienza.
- $max_{a'}Q\left(s',a'\ \right)$ = Miglior valore stato-azione attualmente stimato nel nuovo stato.

Esempio

Supponiamo che il robot sia in una casella e possa scegliere "destra" o "sinistra".

- Se va a sinistra: prende −1 e resta lì.
- Se va a destra: prende +5 e arriva vicino all'uscita.

All'inizio non sa nulla: $Q\ =\ 0$ ovunque.
Prova a destra → ottiene +5 → aggiorna la *Q* di "destra" con un valore positivo.
Col tempo, vedendo che "destra" porta quasi sempre all'uscita, la *Q* per quella mossa diventa sempre più alta.

In questo modo, però, l'agente sceglierebbe sempre l'azione ritenuta migliore; deve invece anche esplorare perché potrebbe sfuggirgli una soluzione ottimale.
L'agente deve provare per imparare.
Infatti se l'agente sceglie sempre l'azione con *Q* più alta, rischia di non scoprire alternative migliori.
Per questo si usa una strategia detta ε-greedy:

- Con probabilità ε fa una mossa a caso (esplora).
- Con probabilità *1−ε* sceglie la migliore (sfrutta) (Sutton & Barto, 2018).

All'inizio ε è alto (tanta esplorazione), poi diminuisce (più sfruttamento).

Quindi, per ricapitolare, l'algoritmo esegue questi passaggi ogni volta che deve compiere una nuova azione durante un episodio:

- Osserva lo stato attuale s in cui si trova.
- Seleziona un'azione a secondo la propria policy, per esempio greedy oppure ε-greedy.
- Esegue l'azione a nell'ambiente.
- Riceve una ricompensa immediata r.
- Osserva il nuovo stato s' raggiunto dopo l'azione.
- Valuta il miglior valore *Q* disponibile nel nuovo stato s'.
- Utilizza r, s' e i valori *Q* già appresi per aggiornare $Q(s, a)$

- Imposta s' come nuovo stato corrente.
- Ripete il processo finché viene raggiunto uno stato terminale o un altro criterio di fine episodio.

Materiale di supporto

Esercizi

Addestrare non significa memorizzare
Un'azienda vuole costruire un sistema che preveda se una macchina industriale avrà bisogno di manutenzione entro i successivi sette giorni.

Possiede 20.000 registrazioni storiche contenenti temperatura, vibrazioni, ore di funzionamento, consumo energetico e indicazione finale guasto / nessun guasto.

Dopo l'addestramento vengono ottenuti questi risultati:

Configurazione	Training	Validation
A	71%	69%
B	98%	76%
C	91%	89%

- Individua input e output del problema.
- Spiega perché si tratta di apprendimento supervisionato.
- Distingui parametri e iperparametri del modello.
- Spiega quale funzione svolge il training set.
- Spiega quale funzione svolge il validation set.
- Quale delle tre configurazioni sembra presentare underfitting? Motiva.
- Quale sembra presentare overfitting? Motiva.
- Quale sceglieresti tra le tre e perché?
- Dopo aver scelto la configurazione, quale insieme di dati dovrebbe essere utilizzato per la valutazione finale?
- Perché non sarebbe corretto osservare ripetutamente il test set e modificare il modello finché il risultato non migliora?
- Immagina che il modello raggiunga il 99% sul training set e il 60% su nuovi dati. Possiamo dire che ha "imparato molto bene"? Motiva.
- Spiega infine che cosa significa generalizzare in questo problema.

Classificazione o regressione?
Una piattaforma di produzione video vuole sviluppare diversi sistemi di machine learning. Deve:

- A. Prevedere quanti minuti saranno necessari per esportare un video.
- B. Stabilire se un file video contiene oppure no problemi di sincronizzazione audio.
- C. Stimare la quantità di spazio disco richiesta da un progetto.
- D. Classificare un'inquadratura come interno, esterno oppure mista.
- E. Prevedere il numero medio di visualizzazioni che un video potrebbe ottenere.

Per ciascun problema:

- Stabilisci se si tratta prevalentemente di classificazione o regressione.
- Individua il possibile output Y.
- Proponi almeno tre possibili feature di input X.
- Spiega perché feature e parametri del modello non sono la stessa cosa.
- Per uno dei problemi di classificazione, proponi almeno cinque esempi che potrebbero appartenere al dataset.
- Per uno dei problemi di regressione, spiega come potrebbe essere misurato l'errore tra previsione e valore reale.
- Un modello produce per un'esportazione la previsione di 14 minuti, mentre il valore reale è di 18 minuti. Individua il residuo.
- Quale classe assegneresti all'input a un modello di classificazione che restituisce invece:
    - interno: 0,20.
    - esterno: 0,70.
    - mista: 0,10.
- Spiega perché un modello di classificazione può produrre anche probabilità senza trasformare il problema in una regressione.
- Concludi indicando la differenza fondamentale tra i due tipi di problema.

Costruire e interpretare un modello di regressione
Un laboratorio vuole stimare il tempo necessario a renderizzare un'animazione in funzione del numero di fotogrammi.
Sono state osservate queste quattro produzioni:

Fotogrammi *x*	**Tempo di rendering *y* in minuti**
100	8
200	14
300	19
400	27

Si vuole costruire un semplice modello lineare: $\hat{y} = a + bx$
- Individua qual è la variabile di input.
- Individua qual è l'output.
- Spiega che cosa rappresenta $\hat{y}$.
- Spiega il significato dei parametri a e b.
- Disegna approssimativamente i quattro punti su un grafico cartesiano.
- Traccia una possibile retta che approssimi i dati.
- La retta deve attraversare necessariamente tutti i punti? Motiva.
- Supponi che per 300 fotogrammi il modello preveda 21 minuti. Calcola il residuo rispetto al valore osservato.
- Se un secondo modello prevedesse 19,5 minuti, quale dei due commetterebbe un errore minore su quell'esempio?
- Spiega perché il metodo dei minimi quadrati utilizza il quadrato dei residui.

- Una volta stimati a e b, come potrebbe essere utilizzato il modello per un'animazione di 500 fotogrammi?
- Questa previsione sarebbe automaticamente affidabile? Quali altre variabili potrebbero influenzare il tempo reale di rendering?
- Immagina di aggiungere come feature anche risoluzione, complessità delle scene e numero di effetti. Che cosa cambierebbe concettualmente rispetto alla regressione con un solo input?
- Spiega infine perché la retta rappresenta una regolarità appresa dai dati e non una regola universale sul rendering.

Scoprire gruppi senza conoscere le categorie
Uno studio di videogiochi raccoglie dati relativi a 5.000 giocatori.
Per ciascuno registra:
- Ore giocate alla settimana.
- Numero medio di sessioni.
- Durata media di una sessione.
- Percentuale di missioni completate.
- Numero di partite cooperative.
- Numero di oggetti acquistati nel gioco.

Non esistono categorie predefinite come "casual", "hardcore" o "social player".
Lo studio decide di utilizzare k-means.
- Spiega perché questo è un problema di apprendimento non supervisionato.
- Quali sono gli input del sistema?
- Qual è l'output prodotto dal clustering?
- Che cosa rappresenta k?
- Perché k è un iperparametro?
- Immagina di scegliere k = 3. Che cosa produrrà concettualmente l'algoritmo?
- Spiega che cos'è un centroide.
- Descrivi le due operazioni che k-means ripete fino alla convergenza.
- Perché potrebbe essere necessario normalizzare le feature prima di utilizzare le distanze?
- Immagina che gli acquisti siano espressi in euro da 0 a 2.000, mentre le ore settimanali variano da 0 a 20. Quale problema potrebbe emergere senza normalizzazione?
- Dopo l'esecuzione emergono tre cluster. Possiamo affermare automaticamente che rappresentano "principianti", "esperti" e "giocatori sociali"? Motiva.
- Chi deve interpretare il significato dei gruppi trovati?
- Che cosa potrebbe cambiare eseguendo nuovamente l'algoritmo con centroidi iniziali differenti?
- Indica almeno due situazioni nelle quali k-means potrebbe produrre risultati poco soddisfacenti.
- Concludi spiegando la differenza tra "Scoprire matematicamente una struttura nei dati." e "Attribuire a quella struttura un significato nel mondo reale".

Progettare un agente che apprende attraverso ricompense
In un videogioco un personaggio controllato dall'IA deve raccogliere risorse e tornare alla propria base.

Durante ogni partita può:

- Muoversi.
- Raccogliere una risorsa.
- Evitare un nemico.
- Attaccare.
- Tornare alla base.
- Esplorare zone sconosciute.

Vuoi progettare il problema come reinforcement learning.

- Definisci che cosa costituisce l'ambiente.
- Proponi quali informazioni dovrebbero far parte dello stato.
- Definisci le possibili azioni.
- Progetta un sistema di reward, assegnando valori positivi e negativi ad almeno cinque eventi.
- Spiega perché premiare soltanto la raccolta immediata di una risorsa potrebbe produrre un comportamento indesiderato.
- Distingui reward immediata e ritorno cumulativo.
- Spiega che cosa rappresenta la policy.
- Spiega invece che cosa rappresenta il value.
- Una posizione non produce alcuna reward immediata ma permette quasi certamente di raggiungere una zona con molte risorse. Potrebbe avere value elevato? Motiva.
- Definisci che cosa costituisce un episodio in questo gioco.
- Individua almeno tre possibili condizioni di fine episodio.
- Spiega il dilemma tra exploration ed exploitation.
- Che problema avrebbe una strategia completamente greedy fin dalle prime partite?
- Spiega il funzionamento generale di una strategia ε-greedy.
- Immagina che all'inizio ε = 0,8. Che tipo di comportamento ti aspetti?
- Immagina che dopo molto addestramento ε = 0,05. Che cosa cambia?
- Se utilizzi Q-learning, che cosa rappresenterebbe ogni valore Q(stato, azione)?
- Spiega perché il valore Q di un'azione può diventare elevato anche se quella specifica azione non produce immediatamente una grande ricompensa.
- Concludi ricostruendo il ciclo: Stato → scelta dell'azione → ambiente → nuovo stato + reward → aggiornamento → nuova scelta.

Bibliografia

Arthur, D., & Vassilvitskii, S. (2007). k-means++: The advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms* (pp. 1027–1035). SIAM. https://theory.stanford.edu/~sergei/papers/kMeansPP-soda.pdf

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press. https://www.deeplearningbook.org/

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2021). *An introduction to statistical learning: With applications in R* (2ª ed.). Springer. https://doi.org/10.1007/978-1-0716-1418-1

National Institute of Standards and Technology. (2024). *Artificial intelligence risk management framework: Generative artificial intelligence profile* (NIST AI 600-1). https://doi.org/10.6028/NIST.AI.600-1

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning, 1*(1), 81–106. https://doi.org/10.1007/BF00116251

Russell, S., & Norvig, P. (2021). *Artificial intelligence: A modern approach* (4ª ed.). Pearson. https://aima.cs.berkeley.edu/

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction* (2ª ed.). MIT Press. https://mitpress.mit.edu/9780262039246/reinforcement-learning/

Watkins, C. J. C. H., & Dayan, P. (1992). Q-learning. *Machine Learning, 8*(3–4), 279–292. https://doi.org/10.1007/BF00992698

Approfondimenti

- Stuart Russell e Peter Norvig, Intelligenza artificiale. Un approccio moderno, 4ª ed., Pearson, 2021-2022 — Rimane il manuale generale di riferimento. La quarta edizione dedica maggiore spazio al machine learning e tratta apprendimento supervisionato e non supervisionato, modelli probabilistici, deep learning e apprendimento per rinforzo, inserendoli nel quadro complessivo dell'IA. È particolarmente utile per comprendere come il machine learning si colleghi ad agenti, decisione, percezione e altre componenti dei sistemi intelligenti. L'edizione italiana è pubblicata in due volumi. .
- Sebastian Raschka e Vahid Mirjalili, Machine Learning con Python. Nuova edizione. Costruire algoritmi per generare conoscenza, Apogeo, 2020 — È probabilmente il testo più direttamente corrispondente alla parte tecnica della dispensa. Affronta preprocessing, classificazione, regressione, selezione e valutazione dei modelli, riduzione della dimensionalità, clustering, reti neurali e reinforcement learning. È particolarmente pertinente perché tratta esplicitamente regressione lineare OLS, overfitting, normalizzazione, cluster e generalizzazione, cioè molti degli stessi concetti sviluppati nel capitolo. .
- Roberto Marmo, Algoritmi per l'intelligenza artificiale. Progettazione, Machine Learning, Neural Network, Deep Learning, ChatGPT, Python, 2ª ed., Hoepli, 2024 — Offre una panoramica molto ampia ma operativa. L'indice comprende pre-elaborazione dei dati, valutazione del modello, regressione lineare e logistica, alberi decisionali, reti neurali e reinforcement learning. È quindi particolarmente utile per collegare i concetti matematici della dispensa alla progettazione concreta di algoritmi e modelli. .
- Luca Massaron e John Paul Mueller, Machine Learning For Dummies, Hoepli, 2019 — È la scelta più accessibile per uno studente che voglia approfondire il principio generale dell'apprendimento automatico senza partire subito da un manuale universitario. Affronta il ruolo dei dati, gli algoritmi, le basi matematiche e l'impiego pratico del machine learning con Python e R. Lo vedrei bene come ponte tra la tua dispensa e testi più tecnici come Raschka e Mirjalili. .
- Nello Cristianini, La scorciatoia. Come le macchine sono diventate intelligenti senza pensare in modo umano, il Mulino, 2023 — Non è un manuale di algoritmi, ma è un ottimo complemento concettuale. Spiega come una parte importante dell'IA

contemporanea sia arrivata a produrre comportamenti efficaci attraverso relazioni statistiche ricavate dai dati, senza dover necessariamente formalizzare esplicitamente conoscenze e regole come nell'IA simbolica. È particolarmente pertinente alla parte introduttiva della dispensa e alle riflessioni sui limiti, sull'opacità dei modelli e sull'impatto sociale dell'IA moderna. .

Nota editoriale

Per la realizzazione di questi materiali ho utilizzato strumenti di intelligenza artificiale generativa come supporto alla scrittura, in particolare per migliorare la forma del testo, riorganizzare i contenuti, correggere la formulazione e velocizzare alcune attività redazionali.

Lavorando autonomamente alla produzione di questi materiali, cerco infatti di automatizzare tutte le attività che possono esserlo, così da poter dedicare più tempo alla ricerca, alla progettazione e allo sviluppo dei contenuti.

L'intelligenza artificiale non determina però i contenuti dell'opera: la scelta degli argomenti, la struttura, le idee, le interpretazioni, gli esempi e l'impostazione didattica sono elaborati da me. L'IA viene quindi utilizzata principalmente come strumento di supporto alla produzione e alla revisione formale, mentre la responsabilità autoriale e progettuale dei contenuti rimane mia.

Salvo diversa indicazione, questo materiale è distribuito con licenza Creative Commons Attribuzione – Non commerciale – Condividi allo stesso modo 4.0 Internazionale (CC BY-NC-SA 4.0).

È quindi possibile condividere, ridistribuire, modificare e creare opere derivate a partire dal materiale, a condizione di attribuirne correttamente la paternità, di non utilizzarlo per scopi commerciali e di distribuire eventuali versioni modificate o derivate con la stessa licenza.

