Profice

AI ACADEMY

Applicare l’Intelligenza Artificiale nello sviluppo software

AI ACADEMY

ML supervised
17/06/2025

INTRODUZIONE DELL'ISTRUTTORE

Tamas Szakacs

Formazione

- Laureato come programmatore matematico
- MBA in management

Principali esperienze di lavoro

- Amministratore di sistemi UNIX
- Oracle DBA
- Sviluppatore di Java, Python e di Oracle PL/SQL
- Architetto (solution, enterprise, security, data)
- Ricercatore tecnologico e interdisciplinare di IA

Dedicato alla formazione continua

- Teorie, modelli, framework IA
- Ricerche IA
- Strategie aziendali
- Trasformazione digitale
- Formazione professionale

email: tamas.szakacs@proficegroup.it

MOTIVI E RIASSUNTO DEL CORSO

L'**Intelligenza Artificiale (AI)** è oggi il motore dell'innovazione in ogni settore, grazie alla sua capacità di analizzare dati, automatizzare processi e generare nuove soluzioni. Questo corso offre una panoramica completa e pratica sullo sviluppo di applicazioni AI moderne, guidando i partecipanti dall'ideazione al rilascio in produzione.

Attraverso una **combinazione di teoria chiara ed esercitazioni pratiche**, saranno affrontate le tecniche e gli strumenti più attuali: **machine learning, deep learning, reti neurali, Large Language Models (LLM), Transformers, Retrieval Augmented Generation (RAG)** e progettazione di agenti AI.
Le competenze acquisite saranno applicate in progetti concreti, dallo sviluppo di chatbot all'integrazione di modelli generativi, fino al deploy di soluzioni AI in ambienti reali e collaborativi.

Il percorso è pensato per chi vuole imparare a progettare, valutare e integrare sistemi AI di nuova generazione, con particolare attenzione alle best practice di programmazione, collaborazione in team, sicurezza, valutazione delle performance ed etica dell'AI.

DURATA: 17 GIORNI

OBIETTIVI

Il percorso formativo è progettato per **giovani consulenti junior**, con una conoscenza base di programmazione, che stanno iniziando un percorso professionale nel settore AI.

L’obiettivo centrale è fornire una panoramica pratica, completa e operativa sull’intelligenza artificiale moderna, guidando ogni partecipante attraverso tutte le fasi fondamentali.

OBIETTIVI

- Allineare conoscenze AI, ML, DL di tutti i partecipanti
- Saper usare e orchestrare modelli LLM (closed e open-weight)
- Costruire pipeline RAG complete (retrieval-augmented generation)
- Progettare agenti AI semplici con strumenti moderni (LangChain, tool calling)
- Capire principi di valutazione, robustezza e sicurezza dei sistemi GenA
- Migliorare la produttività come sviluppatori usando tool GenAI-driven
- Padroneggiare best practice di sviluppo, versioning e deploy AI
- Introdurre i fondamenti di Graph Data Science e Knowledge Graph
- Ottenere capacità di valutazione dei modelli e metriche
- Comprensione dell'etica e dei bias nei modelli di intelligenza artificiale
- Approfondire le normative di riferimento: AI Act, compliance e governance AI

Il corso è **estremamente pratico** (circa il 40% del tempo in esercitazioni hands-on, notebook, challenge e hackathon), con l'utilizzo di Google Colab, GitHub, e tutti gli strumenti necessari per lavorare su progetti reali e simulati.

STRUTTURA DELLE GIORNATE – PROGRAMMA BREVE

Tutte le giornate sono di 8 ore (9:00-17:00), con 1 ora di pausa suddivisa (mezz'ora pranzo, due pause da 15 min durante la mattina e il pomeriggio).

La progettazione sintetica delle giornate:

Giorno	Tema	Breve descrizione
1	Git & Python clean-code	Collaborazione su progetti reali, versionamento, codice pulito e testato
2	Machine Learning Supervised	Modelli supervisionati per predizione e classificazione
3	Machine Learning Unsupervised	Clustering, riduzione dimensionale, scoperta di pattern
4	Prompt Engineering avanzato	Scrivere e valutare prompt efficaci per modelli generativi
5	LLM via API (multi-vendor)	Uso pratico di modelli LLM via API, autenticazione, deployment
6	Come costruire un RAG	Pipeline end-to-end per Retrieval-Augmented Generation
7	Tool-calling & Agent design	Progettare agenti AI che usano strumenti esterni
8	Hackathon: Agentic RAG	Challenge pratica: chatbot agentico RAG in team

STRUTTURA DELLE GIORNATE – PROGRAMMA BREVE

Tutte le giornate sono di 8 ore (9:00-17:00), con 1 ora di pausa suddivisa (mezz'ora pranzo, due pause da 15 min durante la mattina e il pomeriggio).

La progettazione sintetica delle giornate:

Giorno	Tema	Breve descrizione
9	Hackathon: Rapid Prototyping	Da prototipo a web-app con Streamlit e GitHub
10	AI Productivity Tools	Workflow con IDE AI-powered, automazione e refactoring assistito
11	Docker & HF Spaces Deploy	Deployment di app GenAI containerizzate o su HuggingFace Spaces
12	AI Act & ISO 42001 Compliance	Fondamenti di compliance e governance AI
13	Knowledge Base & Graph Data Science	Introduzione a Knowledge Graph e query con Neo4j
14	Model evaluation & osservabilità	Metriche avanzate, explainability, strumenti di valutazione
15	AI bias, fairness ed etica applicata	Analisi dei rischi, metriche e mitigazione dei bias
16-17	Project Work & Challenge finale	Lavoro a gruppi, POC/POD, presentazione e votazione progetti

METODOLOGIA DEL CORSO

1. Approccio introduttivo ma avanzato

Il corso è introduttivo nei concetti base dell'AI applicata allo sviluppo, ma affronta anche tecnologie, modelli e soluzioni avanzate per garantire un apprendimento completo.

2. Linguaggio adattato

Il linguaggio utilizzato è chiaro e adattato agli studenti, con spiegazioni dettagliate dei termini tecnici per favorirne la comprensione e l'apprendimento graduale.

3. Esercizi pratici

Gli esercizi pratici sono interamente svolti online tramite piattaforme come Google Colab o notebook Python, eliminando la necessità di installare software sul proprio computer.

4. Supporto interattivo

È possibile porre domande in qualsiasi momento durante le lezioni o successivamente via email per garantire una piena comprensione del materiale trattato.

NOTA

Il corso segue un **approccio laboratoriale**: ogni giornata combina sessioni teoriche chiare e concrete con molte attività pratiche supervisionate, per sviluppare *competenze reali* immediatamente applicabili.

I partecipanti lavoreranno spesso in gruppo, useranno notebook in Colab e versioneranno codice su GitHub, vivendo una vera simulazione del lavoro in azienda AI.

Nessun prerequisito avanzato richiesto: si partirà dagli strumenti e flussi fondamentali, con una crescita graduale verso le tecniche più attuali e richieste dal mercato.

ORARIO TIPICO DELLE GIORNATE

Orario	Attività	Dettaglio
09:00 – 09:30	Teoria introduttiva	Concetti chiave, schema della giornata
09:30 – 10:30	Live coding + esercizio guidato	Esempio pratico, notebook Colab
10:30 – 10:45	*Pausa breve*	
10:45 – 11:30	Approfondimento teorico	Tecniche, best practice
11:30 – 12:30	Esercizio hands-on individuale	Sviluppo o completamento di codice
12:30 – 13:00	Discussione soluzioni + Q&A	Condivisione e correzione
13:00 – 13:30	*Pausa pranzo*	
13:30 – 14:15	Teoria avanzata / nuovi tools	Nuovi strumenti, pattern, demo
14:15 – 15:30	Esercizio a gruppi / challenge	Lavoro di squadra su task reale
15:30 – 15:45	*Pausa breve*	
15:45 – 16:30	Sommario teorico e pratico	
16:30 – 17:00	Discussioni, feedback	Riepilogo, best practice, domande aperte

PANORAMICA DELLA GIORNATA E MODALITÀ DI LAVORO

Obiettivi della giornata:

- imparare a **collaborare su progetti Python** tramite Git e GitHub
- scrivere codice leggibile, testabile e facilmente migliorabile
- acquisire le basi di formattazione, refactoring e testing automatico
- sperimentare la revisione del codice in team
- **avere programmi e dati pronti nella propria repository**

Struttura delle attività:

- alternanza di **sessioni teoriche** (slide, spiegazioni, domande) e **sessioni pratiche** (esercizi su Colab, lavoro in gruppo)
- uso di una **code-base "sporca"** che verrà progressivamente migliorata
- lavoro continuo con strumenti reali: **Git, GitHub, Colab, Black, isort, pytest**

PANORAMICA DELLA GIORNATA E MODALITÀ DI LAVORO

Modalità di lavoro:

- ogni concetto sarà seguito da **esercitazioni pratiche supervisionate**
- collaborazione in team tramite branch e pull request su GitHub
- discussione aperta di errori, conflitti e soluzioni in aula
- domande e difficoltà affrontate insieme, con esempi reali

Output atteso a fine giornata:

- repository condiviso con codice pulito, test coperti, docstring inserite
- maggiore sicurezza nel gestire progetti Python in team
- comprensione pratica delle best practice di sviluppo moderno

Risorse utilizzate:

- esercizi guidati dai docenti
- cheat sheet di Git e PEP-8
- tutorial “flight rules” per risolvere problemi reali

DOMANDE?

Cominciamo!

OBIETTIVI DELLA GIORNATA E AGENDA

Obiettivi della giornata

- Comprendere il workflow supervisionato: dalla preparazione dati all'interpretazione dei risultati
- Imparare a costruire ed esplorare dataset realistici, anche simulati
- Applicare modelli di regressione e classificazione ai dati energetici
- Valutare le prestazioni dei modelli con le metriche corrette (MAE, R2, accuracy, F1…)
- Saper impostare un esperimento con train/test split e tuning degli iperparametri
- Leggere e interpretare confusion matrix, report di regressione, e feature importance
- Adottare le best practice per la collaborazione e la documentazione in progetti di Machine Learning

AGENDA DELLA GIORNATA

1. **Introduzione al workflow supervisionato**
   - Obiettivi e panoramica
2. **Analisi e preparazione dati**
   - Esplorazione del dataset energetico (Kaggle)
   - Feature engineering, simulazione dati
3. **Modelli di Machine Learning**
   - Regressione lineare e classificazione di base
   - **Random Forest**: principi e applicazione pratica
   - **XGBoost**: introduzione, vantaggi, esercizio guidato
4. **Gestione di classi sbilanciate**
   - Introduzione al sampling (oversampling, undersampling, SMOTE)
   - Applicazione pratica su classificazione “alto consumo”
5. **Suddivisione dati**
   - Train, validation, test split
   - K-fold cross-validation, ROC-AUC
6. **Tuning degli iperparametri**
   - GridSearchCV
   - Bayesian Optimization (Optuna): confronto e best practice
7. **Valutazione modelli**
   - Calcolo e interpretazione di MAE, R2, accuracy, precision, recall, F1, confusion matrix
   - Analisi degli errori
8. **Esercitazione pratica**
   - Implementazione e valutazione di diversi modelli
   - Discussione: quale approccio funziona meglio e perché
9. **Best practice e workflow**
   - Documentazione, collaborazione, riproducibilità
   - Domande frequenti, errori comuni
10. **Riepilogo e domande finali**
    - Cosa portiamo a casa
    - Prossimi passi e confronto

RIPASSO GIORNO 1 – CLEAN CODE & GIT

Clean Code:

- Codice leggibile, chiaro, pronto per essere capito e modificato da chiunque.
- Nomi descrittivi per variabili, funzioni e classi (es. calcola_media_temperatura invece di cmt).
- Funzioni brevi e con una sola responsabilità.
- Niente duplicazioni: ogni logica ripetuta si isola in una funzione.
- Commenti solo dove servono: il codice “parla da sé”.
- Struttura ordinata: spaziature, indentazione, nessun codice “appeso” o morto.

Vantaggi pratici:

- Sviluppo più veloce, meno bug.
- Più facile collaborare, aggiungere funzionalità, risolvere problemi.
- Essenziale nei progetti Data Science e ML dove script/notebook crescono rapidamente.

RIPASSO GIORNO 1 – FLUSSO DI LAVORO CON GIT

Git e GitHub:

- Gestione versioni tramite branch, fork, merge: ogni funzionalità su branch dedicato.
- Flusso:

  fork/clone → crea branch → sviluppa → commit semantic → push → pull request → code review → merge.

- Messaggi di commit chiari e standardizzati (semantic/conventional commit): es. feat: aggiunge funzione di validazione, fix: corregge bug su split dei dati.
- PR (pull request) sempre revisionata almeno da un collega, con feedback prima del merge.

Automazione e qualità:

- Workflow GitHub Actions per automatizzare lint (Black), test (pytest) e bloccare merge su errori.
- Pair-review: ogni modifica va discussa, approvata e integrata in modo trasparente.

CHE COS'È MACHINE LEARNING E DEEP LEARNING

Machine Learning (ML):

- Insieme di tecniche che permettono ai computer di apprendere dai dati, migliorando le proprie prestazioni senza essere programmati esplicitamente per ogni regola.
- Algoritmo costruisce un modello analizzando dati storici (pattern, relazioni) e lo applica su nuovi dati per fare previsioni o decisioni.
- Tipologie principali: supervised (con etichette), unsupervised (senza etichette), reinforcement (apprendimento tramite ricompensa).
- Esempi: raccomandazioni (Netflix), riconoscimento facciale, classificazione email.

Deep Learning (DL):

- Reti neurali artificiali, ispirati alla struttura del cervello umano.
- Utilizza reti neurali profonde (deep neural network) con molti strati (layer) per apprendere rappresentazioni complesse direttamente dai dati grezzi (es. immagini, audio, testo).
- Punti di forza: riconoscimento immagini, traduzione automatica, generazione testi, guida autonoma.
- Richiede molti dati e grande potenza di calcolo, ma offre risultati spesso superiori su compiti complessi.

Collegamento con oggi:

- Oggi lavoriamo con tecniche di machine learning supervisionato (supervised ML), base comune anche per molte soluzioni di deep learning più avanzate.

COS'È UN MODELLO SUPERVISIONATO E PERCHÉ USARLO

Definizione e logica:

- Un modello supervisionato apprende la relazione tra input (feature) e output (target) usando dati etichettati, cioè esempi in cui la risposta corretta è nota.
- L'algoritmo "vede" esempi storici, costruisce una regola generale e la applica a dati nuovi mai visti.

Perché è fondamentale:

- Permette automazione di previsioni, classificazioni e decisioni dove serve replicare l'esperienza "umana".
- Il supervisore ("label") garantisce addestramento controllato, migliore controllo sugli errori.

Quando scegliere il supervised learning:

- Problemi con tanti dati etichettati e output noti: prezzi storici, etichette di immagini, log con eventi passati.
- Obiettivi chiari: classificare, prevedere quantità, segmentare clienti, diagnosticare guasti, rilevare frodi.

Esempi concreti:

- Email: classifica spam vs. non-spam.
- Banca: valuta rischio credito (approvato/non approvato).
- Industria: prevede tempi di manutenzione o presenza guasto.
- Energia: stima consumo ora per ora (forecast).

PROBLEMI DI REGRESSIONE E CLASSIFICAZIONE

Regressione (output numerico):

- Obiettivo: prevedere un valore reale e continuo.
- Esempi:
  - Prevedere il prezzo di una casa in base a zona, mq, caratteristiche.
  - Stimare il consumo energetico orario da dati storici e temperatura.
  - Predire il valore di vendita di un prodotto.
- Applicazioni: energia, immobiliare, finanza, vendite.

Classificazione (output discreto/categoria):

- Obiettivo: assegnare ogni esempio a una classe predefinita.
- Esempi:
  - Diagnosi di malattia (malato/sano, più classi).
  - Classificare una transazione come “fraudolenta” o “regolare”.
  - Riconoscere un oggetto in una foto (gatto/cane/macchina).
- Applicazioni: medicina, sicurezza, marketing, customer care.

Differenze pratiche:

- Regressione → Risposta numerica, errori valutati con MAE/MSE.
- Classificazione → Risposta categoria, errori valutati con accuracy, precision, recall, F1.
- Alcuni problemi sono borderline: classificazione binaria (0/1) ≈ regressione logistica.

PIPELINE TIPICA DEL ML SUPERVISIONATO

1. **Definizione e analisi problema**
   - Chiarire cosa vogliamo prevedere o classificare, con quale utilità di business.
2. **Raccolta, pulizia e preparazione dati**
   - Raccolta dataset, gestione valori mancanti, feature engineering, scaling, encoding.
   - Attenzione a dati spuri, duplicati, errori.
3. **Suddivisione dati (split)**
   - Separare in training, validation, test per valutare le performance su dati mai visti.
4. **Scelta e addestramento modello**
   - Scegliere algoritmo (Decision Tree, Neural Network, Logistic Regression, Random Forest, XGBoost).
   - Allenare sui dati di training.
5. **Valutazione metrica**
   - Usare validation/test per calcolare performance con accuracy, precision, recall, F1, ROC-AUC.
   - Individuare se il modello generalizza o “impara a memoria”.
6. **Ottimizzazione e tuning**
   - Migliorare performance regolando iperparametri (GridSearch, Bayesian Opt.).
   - Cross-validation per robustezza.
7. **Deploy e monitoraggio**
   - Portare in produzione, monitorare errori e drift dei dati, retrain periodico.

Nota: Processo iterativo, sempre adattabile ai feedback reali.

CASI D'USO REALI NEL BUSINESS

Energia

- Previsione della domanda: modelli forecast per ottimizzare produzione e acquisti, ridurre sprechi.

Banche/Assicurazioni

- Fraud detection: classificazione transazioni sospette, blocco automatico su rischio.
- Approvazione prestiti: predizione probabilità di insolvenza.

Industria/Manifattura

- Manutenzione predittiva: anticipare guasti e pianificare interventi prima del fermo macchina.
- Controllo qualità: classificazione difetti da immagini o sensori.

Retail/Marketing

- Churn prediction: anticipare clienti a rischio abbandono, suggerire azioni mirate.
- Segmentazione: classificare clienti in gruppi per offerte personalizzate.

Sanità

- Diagnosi supportata da AI: analisi immagini mediche (radiografie, TAC) per evidenziare anomalie.
- Screening automatico: rilevare casi critici tra grandi volumi di dati.

Cybersecurity

- Rilevamento anomalie nei log di accesso e nei pattern di traffico di rete.

DOMANDE?

Facciamo gli esercizi!

ESERCIZIO – REGRESSIONE

Previsione del prezzo di una casa

Obiettivo

Sperimentare la pipeline completa di regressione supervisionata:

- generare un dataset simulato,
- salvarlo su file,
- addestrare un modello,
- valutare le predizioni,
- visualizzare i risultati.

Output atteso

- Due file CSV: uno con i dati di input, uno con predizioni.
- Un grafico che confronta prezzi reali e predetti.
- Comprensione concreta del ciclo completo “dati → modello → valutazione”.

ESERCIZIO – REGRESSIONE

Cosa si fa (step-by-step):

1.Generazione dati

- Simula 50 case, ciascuna con: metri quadri, numero stanze, distanza dal centro.
- Calcola un prezzo per ogni casa secondo una formula realistica.
- Visualizza le prime righe del dataset e salvalo su case_dataset.csv.

2.Analisi e modellazione

- Carica il CSV appena creato.
- Suddividi il dataset in train/test (ad esempio, 40/10).
- Allena un modello di regressione lineare (LinearRegression).
- Fai predizioni sul test set.

3.Valutazione e confronto

- Calcola e stampa MAE (Mean Absolute Error) e R^2 score.
- Salva i risultati (prezzi reali e predetti) su risultati_regressione.csv.
- Visualizza su grafico:
  - punti blu = dati reali (ideali),
  - punti verdi = predizioni del modello,
  - linea rossa tratteggiata = linea ideale (predizione perfetta).

ESERCIZIO – REGRESSIONE: GENERAZIONE DEI DATI

```python
import numpy as np
import pandas as pd

# Generazione dati
np.random.seed(42)
numero_dati = 50
mq = np.random.randint(40, 150, numero_dati)
stanze = np.random.randint(1, 6, numero_dati)
distanza = np.round(np.random.uniform(0.5, 10,
numero_dati), 2)

prezzo = mq * 1200 + stanze * 15000 - distanza * 2000
+ np.random.normal(0, 10000, n)

df = pd.DataFrame({
    "mq": mq,
    "stanze": stanze,
    "distanza_centro": distanza,
    "prezzo": prezzo.astype(int)
})
# ... ...
```

Dati randomizzati

- Numero delle case
- Area
- Numero stanze
- Distanza dal centro

- Prezzo calcolato per il catalogo

- Numpy dataframe per i dati

Si salvano i dati in CSV

ESERCIZIO – REGRESSIONE: PREDIZIONE

```
# Split train/test (ad esempio: primi 40 train,
ultimi 10 test)
train = df.iloc[:40]
test = df.iloc[40:]

X_train = train[["mq", "stanze", "distanza_centro"]]
y_train = train["prezzo"]
X_test = test[["mq", "stanze", "distanza_centro"]]
y_test = test["prezzo"]

# Modello e training
model = LinearRegression()
model.fit(X_train, y_train)

# Predizione sul test set
y_pred = model.predict(X_test)

# Valutazione e stampa delle metriche
print("MAE:", mean_absolute_error(y_test, y_pred))
print("R2 score:", r2_score(y_test, y_pred))
```

… Lettura dati salvati

Dati per l'addestramento e per il testing

Modello Linear Regression
Addestramento del modello con i dati

Predizione del modello usando i dati test

Valutazione del modello addestrato

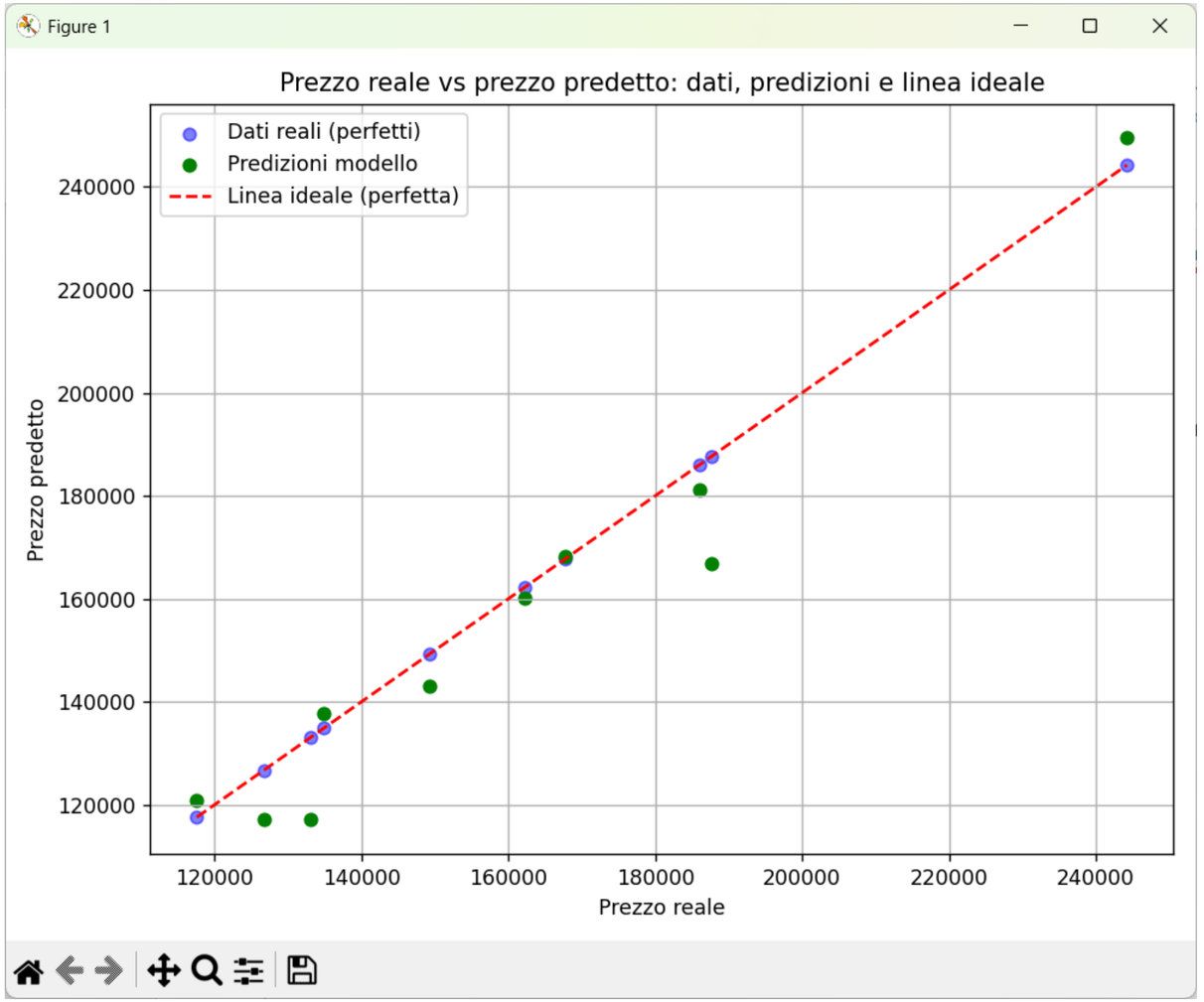
ESERCIZIO – REGRESSIONE: RISULTATI

Suggerimenti

- Guarda dove il modello funziona bene e dove no: osserva i punti verdi rispetto alla linea ideale.
- Controlla i CSV per analizzare gli errori più grandi.
- Modifica i parametri della generazione dati per sperimentare.
- Fai domande se qualche passaggio non ti è chiaro!

MAE (Mean Absolute Error)

- **Cos'è:** Media degli errori assoluti tra valore reale e valore predetto.
- **Uso:** Più è basso, migliore è la previsione (perfetto = 0).



ESERCIZIO – REGRESSIONE: ALTRI ESEMPI

Prevedere il fatturato mensile di un negozio

Obiettivo: Stimare l'ammontare di vendite totali (fatturato) che il negozio realizzerà in un mese.

Feature: numero clienti, spesa media, promozioni attive, giorni apertura.

Stima del tempo di risoluzione di ticket IT

Obiettivo: Prevedere in quante ore o minuti sarà risolto un ticket di assistenza informatica.

Feature: categoria problema, esperienza operatore, priorità, ora apertura.

3. Previsione delle vendite di un nuovo prodotto

Obiettivo: Stimare quante unità del nuovo prodotto saranno vendute in un periodo specifico.

Feature: prezzo, pubblicità investita, stagionalità, numero concorrenti.

4. Prevedere il costo totale di un progetto

Obiettivo: Stimare la spesa complessiva prevista per portare a termine un progetto aziendale.

Feature: numero dipendenti, durata prevista, livello di complessità, ore di straordinario.

5. Stima del tempo di consegna di un ordine e-commerce

Obiettivo: Prevedere quanti giorni (o ore) saranno necessari per la consegna di un ordine online.

Feature: distanza cliente, numero articoli, periodo dell'anno, metodo spedizione.

6. Previsione del punteggio di soddisfazione cliente

Obiettivo: Stimare il valore (scala 1–10) che un cliente assegnerà alla propria esperienza con l'azienda.

Feature: tempo risposta assistenza, numero contatti, tipologia richiesta, valore acquisti.

ESERCIZIO – CLASSIFICAZIONE

Previsione abbandono clienti (“churn prediction”)

Obiettivo

Simulare un caso reale di business: prevedere se un cliente abbandonerà il servizio (“churn”) o rimarrà fedele, utilizzando dati storici realistici.

Output atteso

- Un file con le predizioni (risultati_churn.csv), con clienti classificati sia come churn che come fedeli.
- Accuracy e confusion matrix stampate.
- Capacità di ragionare su chi viene previsto correttamente e chi no.

ESERCIZIO – CLASSIFICAZIONE

Cosa si fa (step-by-step):

1.Generazione e analisi dati

- Si lavora su un dataset simulato di 60 clienti: per ognuno sono disponibili numero reclami, mesi da ultimo acquisto, spesa mensile, cliente VIP (sì/no).
- Il file contiene già la “verità storica” (churn): sappiamo dal passato quali clienti hanno effettivamente abbandonato (label reale).
- La percentuale di abbandono è realistica (25–35%), quindi il modello può imparare davvero a riconoscere i clienti a rischio.
- Dataset pronto in clienti_churn.csv.

2.Compito pratico

- Carica i dati da clienti_churn.csv.
- Suddividi il dataset in train e test (ad es. primi 45 train, ultimi 15 test).
- Addestra un modello di classificazione (ad esempio LogisticRegression).
- Fai predizioni sulla colonna “churn” nel test set.

ESERCIZIO – CLASSIFICAZIONE

3.Valutazione risultati

- Calcola accuracy e confusion matrix.
- Salva le predizioni su risultati_churn.csv.
- Esamina: chi viene classificato correttamente? Dove il modello sbaglia?
- Verifica che il modello preveda sia clienti fedeli che clienti churn (almeno qualche “1” e qualche “0” nelle predizioni).

Condizioni di rischio per churn

Nel nostro esercizio, ogni cliente viene valutato su **tre possibili segnali di rischio** abbandono:

1. **Molti reclami:**
   Se un cliente ha avuto **più di 3 reclami** → segnale di insoddisfazione o problemi ricorrenti.
2. **Tempo lungo dall’ultimo acquisto:**
   Se sono passati **più di 12 mesi dall’ultimo acquisto** → il cliente è poco attivo, forse non interessato a continuare.
3. **Bassa spesa mensile:**
   Se la spesa media mensile è **inferiore a 50** (es. euro) → cliente poco coinvolto, valore basso per l’azienda.

ESERCIZIO – CLASSIFICAZIONE: GENERAZIONE DEI DATI

```
numero_dati = 60
# Genera feature
num_reclami = np.random.randint(0, 6, numero_dati)
mesi_ultimo_acquisto = np.random.randint(0, 25,
numero_dati)
spesa_mensile = np.random.randint(10, 300, numero_dati)
cliente_vip = np.random.binomial(1, 0.2, numero_dati)

# Churn: combinazione di regole + rumore
# Churn se almeno due condizioni di rischio vere,
oppure una ma con "forte rumore"
condizioni_rischio = (
    (num_reclami > 3).astype(int) +
    (mesi_ultimo_acquisto > 12).astype(int) +
    (spesa_mensile < 50).astype(int)
)

# Base: churn per chi ha almeno 2 condizioni
churn = np.where(condizioni_rischio >= 2, 1, 0)
```

Dati randomizzati

- Numero reclami del cliente
- Numero di mesi passati dall'ultimo acquisto
- Quanto il cliente spende mediamente ogni mese
- Se è VIP (il modello può imparare che essere VIP **riduce la probabilità di churn**)

Calcolo di churn reale

ESERCIZIO – CLASSIFICAZIONE: PREDIZIONE

```
df = pd.read_csv("clienti_churn.csv")
train = df.iloc[:45]
test = df.iloc[45:]

X_train = train[["num_reclami",
"mesi_ultimo_acquisto", "spesa_mensile",
"cliente_vip"]]
y_train = train["churn"]
X_test = test[["num_reclami", "mesi_ultimo_acquisto",
"spesa_mensile", "cliente_vip"]]
y_test = test["churn"]

model = LogisticRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

# Valutazione
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Confusion matrix:\n", confusion_matrix(y_test,
y_pred))
```

Lettura dati di addestramento e test

Separazione dei dati

Modello Logistic Regression
Addestramento del modello con i dati
Predizione del modello usando i dati test

Valutazione del modello addestrato

ESERCIZIO – CLASSIFICAZIONE: RISULTATI

num_reclami	mesi_ultimo_acquisto	spesa_mensile	cliente_vip	churn	churn_predetto
4	**15**	**195**	**0**	**1**	**0**
2	2	161	1	0	0
4	3	36	0	1	1
0	18	86	0	0	0
4	13	76	0	1	1
2	**14**	**242**	**0**	**1**	**0**
4	3	64	0	0	0
1	14	155	0	0	0
0	5	203	0	0	0
3	**22**	**195**	**0**	**1**	**0**
3	22	268	0	0	0
1	21	90	0	1	1
5	21	60	0	1	1
0	17	98	0	0	0
4	23	89	0	1	1

Accuracy (Accuratezza)

- **Cos'è:** Percentuale di risposte corrette sul totale delle predizioni fatte.
- **Uso:** Si usa nella classificazione.
  Se su 100 clienti ne predico correttamente 90, accuracy = 90%.
- **Nota:** Non è sempre affidabile con dati sbilanciati (es. 95 "no churn" e 5 "churn").

Suggerimenti

- Controlla che la colonna "churn_predetto" contenga sia 0 che 1 (il modello ha davvero appreso!).
- Apri risultati_churn.csv per vedere clienti "difficili" da prevedere.
- Confronta la performance con altri algoritmi (es: RandomForestClassifier).
- Analizza la confusion matrix: falsi positivi e falsi negativi sono normali in business reali.

ESERCIZIO – CLASSIFICAZIONE: ALTRI ESEMPI

Email phishing detection

Obiettivo: Classificare se una email è “phishing” oppure “legittima”.

Feature: presenza di allegati sospetti, mittente sconosciuto, numero link, parole strane

Previsione rischio credito

Obiettivo: Classificare una richiesta di prestito in “rischio alto”, “medio” o “basso”.

Feature: punteggio creditizio, debiti esistenti, età, storico pagamenti.

Selezione candidati in HR

Obiettivo: Classificare CV in “idoneo”/“non idoneo” per una posizione.

Feature: anni esperienza, titolo di studio, competenze tecniche, conoscenza lingue.

Identificazione di clienti promozionabili

Obiettivo: Classificare clienti in “adatto” o “non adatto” per una nuova promozione.

Feature: storico acquisti, risposta a campagne passate, preferenze prodotto.

Classificazione ticket di supporto

Obiettivo: Assegnare ticket a “priorità alta”, “media”, “bassa”.

Feature: parole chiave nella descrizione, tempo attesa, tipo problema.

Analisi sentiment recensioni clienti

Obiettivo: Classificare recensioni come “positive”, “negative” o “neutre”.

Feature: punteggio assegnato, numero parole positive/negative, presenza di emoji.

DECISION TREE – COS'È E COME FUNZIONA

Definizione:
Un Decision Tree (albero decisionale) è un modello di ML che divide i dati in base a domande ("split") successive, creando rami e foglie che portano a una previsione.

Come funziona:
- Ogni nodo rappresenta una domanda (es: "spesa_mensile < 50?")
- Si segue il ramo "sì" o "no" fino a una foglia (decisione finale)
- Ogni foglia predice una classe (classificazione) o un valore (regressione)

Vantaggi:
- Modello interpretabile ("spiega" il ragionamento)
- Gestisce dati misti (numerici/categorici)
- Non richiede normalizzazione delle feature

DECISION TREE – ESEMPIO PRATICO

Scenario:
Vuoi prevedere il churn clienti con un albero decisionale.

Esempio di regole apprese:

Se “num_reclami > 3” e “spesa_mensile < 50” → churn probabile
Se “cliente_vip = 1” → quasi sempre no churn

Visualizzazione:
Un albero che mostra “a colpo d’occhio” quali domande contano di più

Limiti:

Può “memorize” troppo i dati (overfitting)
Sensibile ai dati rumorosi (piccole variazioni possono cambiare l’albero)

NEURAL NETWORK – COS'È E COME FUNZIONA

Definizione:
Una Neural Network (rete neurale) è un modello ispirato al cervello umano, composto da "neuroni artificiali" organizzati in strati.

Come funziona:
- Ogni neurone riceve valori numerici, li combina e li trasmette agli strati successivi tramite funzioni matematiche
- La rete apprende "pesi" (importanza di ogni input) durante l'addestramento
- Può modellare relazioni molto complesse, anche non lineari

Vantaggi:
- Molto potente su dati complessi (immagini, testi, segnali)
- Può apprendere pattern nascosti difficili da vedere a occhio

Limiti:
- Poco interpretabile ("scatola nera")
- Richiede molti dati e calcolo
- Parametri e struttura da ottimizzare (non plug-and-play)

NEURAL NETWORK – ESEMPIO PRATICO

Obiettivo:
Usare una rete neurale per prevedere quali clienti abbandoneranno il servizio.
Come funziona nel caso churn:

- Gli input sono le stesse feature già viste:
  - numero reclami, mesi dall'ultimo acquisto, spesa mensile, status VIP.
- La rete trasforma queste variabili numeriche in un "punteggio di rischio" tramite molti piccoli calcoli nei vari strati.
- Durante l'addestramento, la rete "impara" pattern e combinazioni non evidenti (es. clienti con pochi reclami ma bassa spesa e non VIP).

Vantaggi rispetto ad altri modelli:

- Può cogliere interazioni complesse tra variabili (es. effetto combinato di più fattori).
- Può essere più precisa dove esistono pattern nascosti o dati non lineari.
- Si adatta bene a scenari con tante feature o dati storici ampi.

Limiti pratici:

- Risultato più difficile da interpretare per il business.
- Rischio di overfitting se i dati sono pochi.

DECISION TREE VS NEURAL NETWORK – CONFRONTO

Aspetto	Decision Tree	Neural Network
Interpretabilità	Alta (facile da spiegare)	Bassa (scatola nera)
Gestione dati	Ottimo per dati tabellari	Ottimo per dati complessi
Richiesta dati	Pochi dati sufficienti	Serve molto più training data
Overfitting	Rischio alto, ma gestibile	Rischio alto, richiede regole
Flessibilità	Limitata, regole rigide	Altissima, anche non lineare
Calcolo	Leggero	Più pesante (CPU/GPU)

- **Quando usare Decision Tree:** dati tabellari, vuoi spiegare le decisioni
- **Quando usare Neural Network:** dati complessi, molti pattern, meno interesse per la spiegazione

DOMANDE?

PAUSA

COS'È LA CLASSIFICAZIONE SBILANCIATA

Definizione:
Si parla di classificazione sbilanciata quando alcune classi sono molto più numerose di altre.

Esempio tipico:
- Churn prediction: il 90–95% dei clienti resta (classe "0"), solo 5–10% abbandona (classe "1").
- Frodi bancarie: solo 0.1–1% delle transazioni sono "frode", tutte le altre sono normali.

Problema:
Il modello può "imparare" a predire sempre la classe maggiore (es. sempre "non churn") e avere un'alta accuracy, ma **non è utile**!

Perché è un problema concreto
- **Accuracy ingannevole:** Se su 1000 clienti solo 20 abbandonano, predire sempre "resta" dà il 98% di accuracy… ma non trovi nessun churn!
- **Business impact:** In molti casi reali (churn, frodi, guasti, malattie rare), le classi minoritarie sono quelle più importanti da trovare.
- **Altro esempio reale:** Nel credito, trovare i pochi clienti che non pagheranno è molto più importante che indovinare tutti quelli affidabili.

COME SI AFFRONTA IL PROBLEMA: SAMPLING

Resampling dei dati:
Si modifica il dataset per “bilanciare” le classi e aiutare il modello a imparare.

- **Oversampling:**
  Si aumentano (duplicano o sintetizzano) gli esempi della classe minoritaria.
  Esempio: SMOTE, RandomOverSampler.
- **Undersampling:**
  Si riducono (rimuovono) gli esempi della classe maggioritaria.
  Esempio: RandomUnderSampler.
- **Class weight:**
  Si dà più peso agli errori sulle classi rare durante il training.

COME SI AFFRONTA IL PROBLEMA: SAMPLING

Esempio pratico: churn sbilanciato

Scenario:
Su 1000 clienti, solo 80 hanno abbandonato.

Cosa succede:

- Modello “grezzo”: predice sempre “no churn”, accuracy 92%, ma nessun churn trovato.

Soluzioni:

- **Oversampling:** duplica/sintetizza i 80 “churn” per arrivare, ad esempio, a 300 esempi.
- **Undersampling:** riduce i 920 “no churn” a 300 esempi.
- **Uso di class_weight in scikit-learn:** penalizza di più gli errori sulla classe minoritaria (es. LogisticRegression(class_weight='balanced')).

COME SI AFFRONTA IL PROBLEMA: SAMPLING

SMOTE sta per **Synthetic Minority Over-sampling Technique**.
È una tecnica di machine learning usata per **bilanciare dataset sbilanciati**:

- Crea nuovi esempi sintetici (artificiali) della classe minoritaria (ad es. “churn”) invece di limitarsi a duplicare quelli esistenti.
- Aiuta il modello a imparare meglio le caratteristiche della classe rara, migliorando la capacità di riconoscere i casi difficili.

Sintesi:
SMOTE = “Tecnica di sovracampionamento sintetico per la classe minoritaria”.
Usata spesso per problemi di classificazione sbilanciata.

1. Bilanciamento delle classi nel training

- **Prima di SMOTE:**
  Nel training set c’erano pochi esempi di churn (“1”), tanti di non-churn (“0”).
  Il modello tendeva a ignorare i churn e prevedere sempre la classe più grande (quasi solo “0”).
- **Dopo SMOTE:**
  Le due classi nel training sono equilibrate: il modello “vede” tanti esempi di abbandono quanti di permanenza.
  Questo obbliga il modello ad imparare davvero a riconoscere le caratteristiche tipiche di chi abbandona.

COME SI AFFRONTA IL PROBLEMA: SAMPLING

2. Migliore capacità di intercettare churn

- **Senza sampling:**
  Il modello rischia di **non individuare mai** o quasi mai i clienti churn.
  Potresti avere anche il 100% di accuracy, ma senza trovare alcun churn reale (molti “falsi negativi”).
- **Con sampling:**
  Il modello diventa più sensibile ai churn.
  Risultato:
  - Più casi di churn vengono correttamente riconosciuti (aumenta il recall per la classe 1)
  - Potresti avere qualche falso positivo in più, ma nel business è meglio “segnalare” troppo che perdere i veri churn.

3. Metriche più significative

Dopo il sampling:

- La **confusion matrix** mostrerà finalmente dei valori diversi da zero nella colonna dei churn previsti.
- **Recall** e **F1-score** per la classe churn (“1”) saranno sensibilmente più alti.
- L’**accuracy** può scendere un po’, ma il modello è più utile e concreto.

COME SI AFFRONTA IL PROBLEMA: SAMPLING

Cosa cambia nel codice con SMOTE

Senza SMOTE:

```
model.fit(X_train, y_train)
```

Il modello viene addestrato su dati originali, che spesso sono sbilanciati (molti “no churn”, pochi “churn”).
Rischio: il modello impara poco o nulla sulla classe minoritaria.

Con SMOTE:

```
from imblearn.over_sampling import SMOTE

sm = SMOTE(random_state=42)
X_train_bal, y_train_bal = sm.fit_resample(X_train, y_train)

model.fit(X_train_bal, y_train_bal)
```

Passaggio chiave: `sm.fit_resample()` genera nuovi esempi sintetici della classe minoritaria (“churn”).
Dopo SMOTE, il training set contiene tanti esempi “churn” quanti “no churn”.

Il modello viene addestrato su dati bilanciati, quindi è più capace di imparare a riconoscere i churn veri.

COME SI AFFRONTA IL PROBLEMA: SAMPLING

4. Tabella risultati

Il file `G2E1_risultati_churn_sampling.csv` ora permette di vedere:

- Quali clienti di test sono stati previsti correttamente come churn.
- Dove ci sono stati falsi positivi (clienti previsti a rischio, ma rimasti).
- Un confronto reale tra modello “cieco” e modello “sensibile”.

num_reclami	mesi_ultimo_acquisto	spesa_mensile	cliente_vip	churn	churn_predetto
4	**15**	**195**	**0**	**1**	**0**
2	2	161	1	0	0
4	3	36	0	1	1
0	18	86	0	0	0
4	13	76	0	1	1
2	**14**	**242**	**0**	**1**	**0**
4	**3**	**64**	**0**	**0**	**1**
1	14	155	0	0	0
0	5	203	0	0	0
3	**22**	**195**	**0**	**1**	**0**
3	22	268	0	0	0
1	21	90	0	1	1
5	21	60	0	1	1
0	17	98	0	0	0
4	23	89	0	1	1

RANDOM FOREST – COS'È E COME FUNZIONA

Definizione:
Una Random Forest è un insieme di tanti alberi decisionali "deboli" (decision tree), ciascuno addestrato su un diverso campione casuale dei dati.

Come funziona:
- Ogni albero dà la sua "votazione" su un caso (ad es. churn o no churn).
- Il risultato finale è la maggioranza dei voti (classificazione) o la media (regressione).

Punti di forza:
- Riduce l'overfitting dei singoli alberi.
- Gestisce bene dati rumorosi e variabili inutili.
- Fornisce una "importanza" delle feature (quali sono le più decisive).

Limiti:
- Più pesante da calcolare di un singolo albero.
- Più difficile da interpretare (anche se meglio delle neural network).

RANDOM FOREST – PREVISIONE CHURN

Esempio di uso:

- Prendi lo stesso dataset del churn: feature come reclami, spesa, mesi dall'ultimo acquisto, VIP.
- La Random Forest costruisce molti alberi diversi: alcuni potrebbero trovare pattern tra reclami e VIP, altri tra spesa e tempo.
- Il modello risulta robusto: trova sia pattern "semplici" che "combinati".

Quando preferirla:

- Hai dati tabellari e vuoi un modello più potente di un solo decision tree.
- Vuoi una stima della rilevanza di ciascuna variabile.

XGBOOST – COS'È E COME FUNZIONA

Definizione:
XGBoost è un algoritmo avanzato di boosting basato su alberi, che costruisce sequenze di alberi decisionali, ciascuno specializzato a "correggere" gli errori dei precedenti.

Come funziona:
- Gli alberi vengono creati uno dopo l'altro.
- Ogni nuovo albero cerca di migliorare dove i precedenti hanno sbagliato.
- Il risultato finale è una combinazione "pesata" di tutti gli alberi (molto preciso e flessibile).

Punti di forza:
- Spesso è lo stato dell'arte in competizioni e casi reali.
- Gestisce dati mancanti, feature numeriche e categoriche.
- Altissima accuratezza anche su dati tabellari di business.

Limiti:
- Più complesso da impostare (parametri da ottimizzare).
- Meno interpretabile rispetto a Random Forest e Decision Tree.
- Richiede più risorse computazionali.

RANDOM FOREST VS XGBOOST – CONFRONTO RAPIDO

Aspetto	Random Forest	XGBoost
Potenza	Alta, robusto, versatile	Massima, spesso “top”
Velocità	Più veloce da addestrare	Più lento, ma ottimizzato
Interpretabilità	Discreta (importanza feature)	Più bassa (“black box”)
Overfitting	Resistente	Ben controllato
Parametri	Pochi da regolare	Molti da ottimizzare
Quando usarlo	Modello solido “all purpose”	Quando vuoi massima accuratezza

Sintesi:
Random Forest e XGBoost sono oggi i modelli più forti su dati aziendali tabellari: spesso usati come “baseline avanzata” prima di applicare deep learning!

ESERCIZIO – PREVISIONE CHURN CON RANDOM FOREST

Obiettivo:
Utilizzare un modello Random Forest per prevedere quali clienti abbandoneranno (churn) sulla base degli stessi dati storici visti negli esercizi precedenti.

Cosa si fa (step-by-step):

1. **Carica i dati reali**
   - Usa il file **G2E1_clienti_churn.csv** (già preparato).
2. **Suddividi train/test**
   - Primi 45 clienti = training, ultimi 15 = test.
3. **Addestra una Random Forest**
   - Usa le feature: numero reclami, mesi dall'ultimo acquisto, spesa mensile, cliente VIP.
4. **Prevedi il churn nel test set**
5. **Valuta i risultati**
   - Calcola accuracy, confusion matrix, classification report.
   - Salva una tabella risultati con le predizioni: **G2E2_risultati_churn_rf.csv**.

ESERCIZIO – PREVISIONE CHURN CON RANDOM FOREST

Profice

DOMANDA

Quale parte del codice di Logistic Regression bisogna cambiare?

Quante righe di codice da cambiare?

ESERCIZIO – PREVISIONE CHURN CON RANDOM FOREST

```
df = pd.read_csv("clienti_churn.csv")
train = df.iloc[:45]
test = df.iloc[45:]

X_train = train[["num_reclami",
"mesi_ultimo_acquisto", "spesa_mensile",
"cliente_vip"]]
y_train = train["churn"]

X_test = test[["num_reclami",
"mesi_ultimo_acquisto", "spesa_mensile",
"cliente_vip"]]
y_test = test["churn"]

model = LogisticRegression()

model.fit(X_train, y_train)

y_pred = model.predict(X_test)
```

Seguono poi l’output e la valutazione…

ESERCIZIO – PREVISIONE CHURN CON RANDOM FOREST

```
df = pd.read_csv("clienti_churn.csv")
train = df.iloc[:45]
test = df.iloc[45:]

X_train = train[["num_reclami",
"mesi_ultimo_acquisto", "spesa_mensile",
"cliente_vip"]]
y_train = train["churn"]

X_test = test[["num_reclami",
"mesi_ultimo_acquisto", "spesa_mensile",
"cliente_vip"]]
y_test = test["churn"]

model = LogisticRegression()

model.fit(X_train, y_train)

y_pred = model.predict(X_test)
```

```
df = pd.read_csv("G2E1_clienti_churn.csv")
train = df.iloc[:45]
test = df.iloc[45:]

X_train = train[["num_reclami",
"mesi_ultimo_acquisto", "spesa_mensile",
"cliente_vip"]]
y_train = train["churn"]

X_test = test[["num_reclami",
"mesi_ultimo_acquisto", "spesa_mensile",
"cliente_vip"]]
y_test = test["churn"]

model_rf = RandomForestClassifier(n_estimators=100, random_state=42)

model_rf.fit(X_train, y_train)

y_pred = model_rf.predict(X_test)
```

Seguono poi l’output e la valutazione…

ESERCIZIO – PREVISIONE CHURN CON RANDOM FOREST

```
df = pd.read_csv("clienti_churn.csv")
train = df.iloc[:45]
test = df.iloc[45:]

X_train = train[["num_reclami",
"mesi_ultimo_acquisto", "spesa_mensile",
"cliente_vip"]]
y_train = train["churn"]

X_test = test[["num_reclami",
"mesi_ultimo_acquisto", "spesa_mensile",
"cliente_vip"]]
y_test = test["churn"]

model = LogisticRegression()

model.fit(X_train, y_train)

y_pred = model.predict(X_test)
```

```
df = pd.read_csv("G2E1_clienti_churn.csv")
train = df.iloc[:45]
test = df.iloc[45:]

X_train = train[["num_reclami",
"mesi_ultimo_acquisto", "spesa_mensile",
"cliente_vip"]]
y_train = train["churn"]

X_test = test[["num_reclami",
"mesi_ultimo_acquisto", "spesa_mensile",
"cliente_vip"]]
y_test = test["churn"]

model_rf = RandomForestClassifier(n_estimators=100, random_state=42)

model_rf.fit(X_train, y_train)

y_pred = model_rf.predict(X_test)
```

L’unica differenza è il modello usato e i suoi parametri!

ESERCIZIO – PREVISIONE CHURN CON RANDOM FOREST

Cosa significano `n_estimators=100`, `random_state=42` nella Random Forest

`n_estimators=100`

- Indica **quanti alberi decisionali** vengono creati nella foresta.
- **100** è il valore di default e rappresenta un buon compromesso tra accuratezza e velocità.
- **Più alberi = modello più stabile e preciso**, ma anche più lento da addestrare.
- Puoi aumentare questo numero per problemi più complessi, oppure diminuirlo per accelerare i test.

`random_state=42`

- Fissa il “seme” del generatore casuale (random seed).
- Serve per **rendere i risultati ripetibili**:
  - Eseguendo il codice più volte, ottieni sempre gli stessi alberi e le stesse predizioni.
  - 42 è un valore convenzionale scelto per caso (è un “inside joke” tra programmatori!), ma puoi usare qualsiasi numero.

ESERCIZIO – PREVISIONE CHURN CON RANDOM FOREST

Suggerimenti per studenti:

- Confronta le predizioni della Random Forest con quelle ottenute dalla Logistic Regression.
- Analizza la **confusion matrix**: la Random Forest trova più churn o meno falsi positivi?
- Prova a cambiare il parametro n_estimators (numero alberi): cosa succede?
- Verifica quali feature sono risultate più importanti (model_rf.feature_importances_).
- In caso di dataset sbilanciato, considera anche qui l'uso di tecniche di sampling (come già visto).

Output atteso:

- File **G2E2_risultati_churn_random_forest.csv** con le predizioni.
- Metriche di valutazione stampate.
- Possibilità di commentare pregi e limiti di Random Forest rispetto agli altri modelli.

ESERCIZIO – PREVISIONE CHURN CON RANDOM FOREST

Valutazione

Il modello trova bene i churn:

- Riconosce 5 su 7 clienti che abbandonano (recall buona!).
- Solo 2 churn non individuati (FN), 1 solo “falso allarme” (FP).
- Precisione alta: se dice “churn”, di solito ha ragione.

num_reclami	mesi_ultimo_acquisto	spesa_mensile	cliente_vip	churn	churn_predetto_rf
4	**15**	**195**	**0**	**1**	**0**
2	**2**	**161**	**1**	**0**	**1**
4	3	36	0	1	1
0	18	86	0	0	0
4	13	76	0	1	1
2	**14**	**242**	**0**	**1**	**0**
4	3	64	0	0	0
1	14	155	0	0	0
0	5	203	0	0	0
3	**22**	**195**	**0**	**1**	**0**
3	22	268	0	0	0
1	21	90	0	1	1
5	21	60	0	1	1
0	17	98	0	0	0
4	23	89	0	1	1

Rispetto a modelli base (es. senza sampling, o solo logistic regression) questa Random Forest:

- Trova più churn (utile per il business).
- Tiene bassa la percentuale di falsi positivi.

•Limite:

- Due clienti churn sfuggiti, si può migliorare ulteriormente (es. ottimizzando i parametri, aggiungendo sampling, provando XGBoost).

ESERCIZIO – PREVISIONE CHURN CLIENTI CON XGBOOST

Obiettivo:
Addestrare e valutare un modello XGBoost per prevedere il churn sugli stessi dati, confrontando i risultati con la Random Forest.

Cosa si fa (step-by-step)

1. **Carica i dati reali**
   - Usa il file **G2E1_clienti_churn.csv**.
2. **Suddividi in train/test**
   - Primi 45 clienti = training, ultimi 15 = test.
3. **Addestra un modello XGBoost**
   - Feature: numero reclami, mesi dall'ultimo acquisto, spesa mensile, cliente VIP.
4. **Prevedi il churn sul test set**
5. **Valuta i risultati**
   - Calcola accuracy, confusion matrix, classification report.
   - Salva una tabella risultati con le predizioni: **G2E3_risultati_churn_xgb.csv**.

ESERCIZIO – PREVISIONE CHURN CLIENTI CON XGBOOST

Esercizio personale:
Scrivere il codice completo modificando Random Forest.

Modello XGBoost e parametri da usare:

```
model_xgb = XGBClassifier(n_estimators=100, use_label_encoder=False,
eval_metric='logloss', random_state=42)
```

`n_estimators=100`

- Indica **quanti alberi decisionali** vengono creati nella foresta.

`random_state=42`

- Fissa il “seme” del generatore casuale (random seed). Serve per **rendere i risultati ripetibili.**

ESERCIZIO – PREVISIONE CHURN CLIENTI CON XGBOOST

`use_label_encoder=False`
Disattiva il vecchio “label encoder” di XGBoost, che trasformava le etichette in valori numerici in modo automatico (vecchio stile).

`eval_metric='logloss'`
Indica quale metrica XGBoost deve usare per valutare la performance durante l’addestramento.

Cosa significa “logloss”:

- È la **loss** più comune nei problemi di classificazione binaria.
- Misura quanto la probabilità prevista dal modello è vicina alla classe vera (più bassa è meglio).

Perché si usa:

- È lo standard per classificazione 0/1 (come nel churn).
- Esplicitandolo, si evitano warning e il comportamento è chiaro per tutti.

ESERCIZIO – PREVISIONE CHURN CLIENTI CON XGBOOST

Suggerimenti

- Confronta le predizioni di XGBoost con quelle di Random Forest e Logistic Regression.
- Analizza la **confusion matrix**: XGBoost trova più churn o genera più falsi positivi?
- Prova a cambiare il parametro n_estimators (più alberi: cambia la sensibilità?).
- Nota che XGBoost può essere più sensibile (riconoscere quasi tutti i churn), ma anche segnalare qualche cliente fedele di troppo.
- Con dati sbilanciati, valuta anche tecniche di sampling o aggiustamento dei pesi.

Output atteso

- File **G2E3_risultati_churn_xgboost.csv** con le predizioni di XGBoost.
- Metriche stampate: accuracy, confusion matrix, classification report.
- Possibilità di commentare punti di forza e debolezza rispetto agli altri modelli testati.

ESERCIZIO – PREVISIONE CHURN CLIENTI CON XGBOOST

num_reclami	mesi_ultimo_acquisto	spesa_mensile	cliente_vip	churn	churn_predetto_xgb
4	**15**	**195**	**0**	**1**	**0**
2	2	161	1	0	0
4	3	36	0	1	1
0	18	86	0	0	0
4	**13**	**76**	**0**	**1**	**0**
2	**14**	**242**	**0**	**1**	**0**
4	3	64	0	0	0
1	14	155	0	0	0
0	5	203	0	0	0
3	**22**	**195**	**0**	**1**	**0**
3	22	268	0	0	0
1	21	90	0	1	1
5	21	60	0	1	1
0	17	98	0	0	0
4	23	89	0	1	1

Il cliente ha abbandonato davvero (churn=1), ma XGBoost NON lo ha riconosciuto (ha previsto 0).

ESERCIZIO – PREVISIONE CHURN CLIENTI CON XGBOOST

Perché può succedere?

- **Il modello non è perfetto:** Anche i migliori algoritmi possono sbagliare alcuni casi, specie se sono “al limite” o rari nel dataset.
- **Dati poco distintivi:** I valori di questa riga potrebbero non essere “chiari” per il modello: magari tanti clienti con caratteristiche simili non hanno abbandonato, quindi XGBoost si “fida” di più della classe 0.
- **Dataset piccolo:** Su pochi esempi, ogni caso pesa molto: se ci sono pochi churn, il modello può avere difficoltà a generalizzare.
- **Parametri non ottimizzati:** Il modello è stato addestrato con parametri standard. A volte, servono più alberi, un learning rate più basso, o tecniche di bilanciamento per cogliere meglio casi “difficili”.

Come si chiama questo errore?

Falso negativo (“False Negative”):
Il modello NON individua un churn vero.

- **Effetto business:** rischi di perdere un cliente senza preavviso!

ESERCIZIO – PREVISIONE CHURN CON RANDOM FOREST

Profice

DOMANDA

Cosa si può fare per migliorare il modello?

Quante righe di codice da cambiare?

ESERCIZIO – PREVISIONE CHURN CLIENTI CON XGBOOST

Cosa si può fare per migliorare:

Prima di tutto:

- **Ottimizzare i parametri** di XGBoost (più alberi, learning rate, profondità).

Altre tecniche:

- Usare **sampling** per bilanciare meglio la classe churn nel training.
- Analizzare le feature più importanti: magari ne manca qualcuna che aiuta a distinguere meglio i casi critici.
- Unire più modelli (ensemble, stacking) per migliorare la robustezza.

ESERCIZIO – PREVISIONE CHURN CLIENTI CON XGBOOST

Learning rate (learning_rate)
Indica di quanto viene “corretto” il modello dopo ogni nuovo albero.

Valori tipici:
Da 0.01 a 0.3 (il default spesso è 0.3)

Effetto pratico:

- **Valori bassi**: l’apprendimento è più lento, ma il modello può diventare più preciso e meno soggetto ad overfitting (a patto di aumentare il numero di alberi!).
- **Valori alti**: il modello impara più in fretta, ma rischia di “saltare” soluzioni migliori e overfittare.

Esempio uso:

```
XGBClassifier(learning_rate=0.1, ...)
```

ESERCIZIO – PREVISIONE CHURN CLIENTI CON XGBOOST

Profondità massima degli alberi (max_depth)
Imposta **quanti livelli** può avere ciascun albero (quante domande consecutive può fare).

Valori tipici:
Da 3 a 10 (in pratica si va spesso da 3 a 6)

Effetto pratico:

- **Valori bassi**: gli alberi sono più semplici, rischio di underfitting.
- **Valori alti**: gli alberi sono più complessi, rischio di overfitting.

Esempio uso:

```
XGBClassifier(max_depth=4, ...)
```

ESERCIZIO – PREVISIONE CHURN CLIENTI CON XGBOOST

Come cambiare learning rate e profondità in XGBoost

Nel codice originale:

```
model_xgb = XGBClassifier(
    n_estimators=100,
    use_label_encoder=False,
    eval_metric='logloss',
    random_state=42
)
```

Per abbassare la learning rate (esempio: da 0.3 a 0.1):

```
model_xgb = XGBClassifier(
    n_estimators=100,
    learning_rate=0.1,        # <--- learning rate più bassa!
    use_label_encoder=False,
    eval_metric='logloss',
    random_state=42
)
```

ESERCIZIO – PREVISIONE CHURN CLIENTI CON XGBOOST

Come cambiare learning rate e profondità in XGBoost

Per abbassare/aumentare la profondità massima degli alberi (esempio: da default 6 a 4):

```
model_xgb = XGBClassifier(
    n_estimators=100,
    learning_rate=0.1,
    max_depth=4,              # <--- profondità massima per ogni albero!
    use_label_encoder=False,
    eval_metric='logloss',
    random_state=42
)
```

Altri parametri importanti:

- `subsample`: percentuale di dati usati per ogni albero (aiuta a regolarizzare)
- `colsample_bytree`: percentuale di feature usate per ogni albero

ESERCIZIO – PREVISIONE CHURN CLIENTI CON XGBOOST

num_reclami	mesi_ultimo_acquisto	spesa_mensile	cliente_vip	churn	churn_predetto_xgb
4	**15**	**195**	**0**	**1**	**0**
2	2	161	1	0	0
4	**3**	**36**	**0**	**1**	**1**
0	18	86	0	0	0
4	13	76	0	1	**0**
2	**14**	**242**	**0**	**1**	**0**
4	3	64	0	0	0
1	14	155	0	0	0
0	5	203	0	0	0
3	**22**	**195**	**0**	**1**	**0**
3	22	268	0	0	0
1	**21**	**90**	**0**	**1**	**1**
5	21	60	0	1	1
0	17	98	0	0	0
4	23	89	0	1	1

Compito personale:

Migliorare la predizione del modello come sulla tabella sopra, modificando i parametri `max_depth e learning_rate`.

ESERCIZIO – PREVISIONE CHURN CLIENTI CON XGBOOST

Soluzione

```
model_xgb = XGBClassifier(
    n_estimators=100,
    max_depth=8,
    learning_rate=0.01,
    use_label_encoder=False,
    eval_metric='logloss',
    random_state=42
)
```

Cosa è cambiato?

- Elevato la profondità degli alberi a 8
- Diminuito il learning rate a 0.01

RACCOMANDAZIONI PER MODIFICARE I PARAMETRI

Per ottimizzare XGBoost, inizia sempre dai valori di default (es: learning_rate=0.3, max_depth=6).

Per modelli più robusti, abbassa la learning rate:

- Prova learning_rate=0.1 o learning_rate=0.05
- **ATTENZIONE:** se abbassi la learning rate, **aumenta il numero di alberi** (n_estimators) per compensare (es: 200 o 300).

Per ridurre l'overfitting, abbassa la profondità:

- Prova max_depth=3 o max_depth=4

Fai esperimenti cambiando un parametro alla volta

- Tieni traccia delle metriche (accuracy, recall, F1-score)
- Usa sempre lo stesso random_state per confrontare i risultati

Non usare valori troppo estremi:

- Un modello troppo profondo (max_depth alto) rischia di imparare a memoria (overfitting)
- Una learning rate troppo bassa può rendere l'addestramento lento o inefficace

SPLIT TRAIN, VALIDATION E TEST

I dati di addestramento devono essere divisi in tre

Train set

- Serve per **addestrare** il modello: qui il modello “impara” dai dati storici.

Validation set

- Serve per **tuning** e scelta dei parametri (es: profondità, learning rate).
- Non viene mai usato per l’addestramento diretto: permette di capire se il modello sta “imparando troppo a memoria” (overfitting).
- Si può anche usare la **cross-validation** (più validation diverse a rotazione).

Test set

- Usato **solo alla fine**, per valutare le prestazioni reali del modello su dati “nuovi”, mai visti prima.
- È la simulazione più fedele di cosa accadrà in produzione.

SPLIT TRAIN, VALIDATION E TEST

Perché non basta solo train/test?

- Se scegli parametri usando il test set, rischi di ottimizzare il modello **sul test** (overfitting mascherato).
- Serve un validation set separato per tuning onesto.

Esempio pratico

Split classico:

- **Train:** 60-70%
- **Validation:** 15-20%
- **Test:** 15-20%

Nel nostro esercizio: primi 45 (train), ultimi 15 (test), ma per progetti reali aggiungi anche validation!

SPLIT TRAIN, VALIDATION E TEST

```
# Split: primi 35 = train, successivi 10 = validation, ultimi 15 = test
train = df.iloc[:35]
val = df.iloc[35:45]
test = df.iloc[45:]

X_train = train[["num_reclami", "mesi_ultimo_acquisto", "spesa_mensile", "cliente_vip"]]
y_train = train["churn"]
X_val = val[["num_reclami", "mesi_ultimo_acquisto", "spesa_mensile", "cliente_vip"]]
y_val = val["churn"]
X_test = test[["num_reclami", "mesi_ultimo_acquisto", "spesa_mensile", "cliente_vip"]]
y_test = test["churn"]

# Addestra modello XGBoost con tuning
model_xgb = XGBClassifier(n_estimators=100, max_depth=8, learning_rate=0.01,
   use_label_encoder=False, eval_metric='logloss', random_state=42
)
model_xgb.fit(X_train, y_train, eval_set=[(X_val, y_val)], verbose=True)

# Predizioni sul test set
y_pred = model_xgb.predict(X_test)
```

SPLIT TRAIN, VALIDATION E TEST

Esercizio personale:
Aumentate il numero dei dati da generare, riutilizzando il programma G2E1_Classificazione_Dati.py.
Osservate la tabella dei risultati, cosa possiamo notare?

Aiuto:

Aumenta il numero dei dati:

```
numero_dati = 60
```

Cambia nome del file output:

```
df_churn.to_csv("G2E2_clienti_churn_molti_dati.csv", index=False)
print("Dataset clienti salvato in 'G2E2_clienti_churn_molti_dati.csv'")
```

Salva il programma con un altro nome:

`G2E2_Classificazione_Molti_Dati.py`

Riavvia il programma `G2E2_XGBoost_Validation.py`
utilizzando `G2E2_clienti_churn_molti_dati.csv`

DISCUSSIONE APERTA – DOMANDE E RISPOSTE

Domande utili :

- Ti senti sicuro su come suddividere i dati in train, validation e test?
- Vorresti riprovare da solo la creazione di un modello e la valutazione delle sue metriche?
- Hai compreso quando preferire modelli semplici (decision tree, logistic regression) rispetto a quelli avanzati (Random Forest, XGBoost, Neural Network)?
- Ti è chiaro cosa significa overfitting e come evitarlo?
- Hai domande su tecniche come SMOTE, tuning dei parametri o interpretazione della confusion matrix?
- Vuoi vedere esempi pratici di tuning dei parametri o visualizzazione dei risultati?

Invito attivo:

- Nessuna domanda è “banale”: spesso i dubbi aiutano tutta la classe a chiarire i concetti fondamentali.
- Prenditi un attimo per rivedere gli esercizi o gli appunti, e chiedi anche sulle piccole incertezze!

DOMANDE?

PAUSA PRANZO

Metriche: accuracy, precision, recall, F1-score, support

Introduzione alle metriche di valutazione

Quando costruiamo un modello di Machine Learning, **non basta sapere quanti casi azzecca in totale**: dobbiamo capire *come* e *dove* il modello sbaglia.
Nel business, spesso è più importante individuare certi errori rispetto ad altri (es: “perdere un cliente” vs. “disturbare chi non lo è”).
Per questo usiamo diverse **metriche di valutazione**:

- Ognuna risponde a una domanda diversa sulla qualità delle predizioni.
- Conoscerle bene ti aiuta a scegliere il modello giusto per ogni problema.

In questo modulo vediamo le principali:

- Accuracy
- Precision
- Recall
- F1-score
- Support

MERTICHE DEI MODELLI

Utilizziamo la tabella e le **definizioni** dei quattro elementi fondamentali nelle metriche di classificazione: **True/False Positives/Negatives** (veri/falsi positivi/negativi).

Sigla	Nome italiano	Definizione
TP – True positive	**Vero positivo**	Il modello prevede “sì” e la realtà è “sì”. (Esempio: prevede churn e c’è churn)
FP – False positive	**Falso positivo**	Il modello prevede “sì” ma la realtà è “no”. (Esempio: prevede churn, ma non c’è)
TN – True negative	**Vero negativo**	Il modello prevede “no” e la realtà è “no”. (Prevede no churn e davvero non c’è)
FN – False negative	**Falso negativo**	Il modello prevede “no” ma la realtà è “sì”. (Prevede no churn, ma c’è churn)

ACCURACY, PRECISION, RECALL, F1-SCORE, SUPPORT

Accuracy

Definizione:
Percentuale di predizioni corrette sul totale dei casi.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \quad \text{cioè} \quad \text{Accuracy} = \frac{\text{veri positivi} + \text{veri negativi}}{\text{totale dei casi}}$$

Quando usarla:

- Se le classi sono ben bilanciate (es: churn sì/no quasi 50/50).
- Quando vuoi una visione d'insieme della correttezza.

Limite:

- Se una classe è molto più frequente, può essere fuorviante!

Esempio:

- Su 100 clienti, 90 non churn, 10 sì.
  Il modello prevede 92 giusti su 100 → accuracy = 92%

ACCURACY, PRECISION, RECALL, F1-SCORE, SUPPORT

Precision

Definizione:
Percentuale di predetti positivi che sono veramente positivi.

$$\text{Precision} = \frac{TP}{TP + FP} \qquad \text{cioè} \qquad \text{Precision} = \frac{\text{veri positivi}}{\text{tutti i predetti positivi}}$$

Quando usarla:

- Quando è importante ridurre i “falsi allarmi” (falsi positivi).
- Es: chiamare clienti davvero a rischio churn, non disturbare i fedeli.

Esempio:

- Il modello segnala 8 clienti a rischio churn, 6 lo sono davvero → precision = 6/8 = 75%

ACCURACY, PRECISION, RECALL, F1-SCORE, SUPPORT

Recall

Definizione:
Percentuale dei veri positivi individuati rispetto a tutti i positivi reali.

$$\text{Recall} = \frac{TP}{TP + FN} \qquad \text{cioè} \qquad \text{Recall} = \frac{\text{veri positivi}}{\text{veri positivi} + \text{falsi negativi}}$$

Quando usarla:

- Se è fondamentale non “perdere” nessun caso importante.
- Es: meglio contattare tutti i clienti churn, anche se qualcuno non lo è.

Esempio:

- Ci sono 10 clienti che fanno churn, il modello ne individua 7 → recall = 7/10 = 70%

ACCURACY, PRECISION, RECALL, F1-SCORE, SUPPORT

F1-score

Definizione:
Media armonica tra precision e recall.

$$F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Quando usarla:

- Se vuoi un equilibrio tra ridurre i falsi positivi e non perdere i veri positivi.

Esempio:

- Precision = 80%, Recall = 60% → F1 = 2×(0.8×0.6)/(0.8+0.6) = 0.69 (69%)

Cosa significa F1-score?

- **Un valore alto di F1** indica che **sia la precision che il recall sono alti**:
  - il modello trova molti casi positivi veri (alto recall)
  - e tra quelli che segnala come positivi, pochi sono falsi (alta precisione)
- **Un valore basso** significa che o la precision o il recall (o entrambe) sono basse.

Come si usa la F1?

- **Quando ci serve un bilanciamento tra evitare i falsi negativi** (non perdere positivi veri) **e i falsi positivi** (non segnalare troppi casi sbagliati).
- È **particolarmente utile** quando le classi sono sbilanciate, o servono sia la qualità delle segnalazioni che la copertura dei casi veri.

ACCURACY, PRECISION, RECALL, F1-SCORE, SUPPORT

Support

Definizione:
Support è semplicemente **il numero reale di esempi presenti nei dati per ciascuna classe**.

In pratica:
Mostra **quanti casi di ogni classe** ci sono nei dati (es: quanti “churn” e quanti “non churn”).
Esempio:
Su 100 clienti, 15 hanno fatto churn, 85 no:
- Support classe “churn” = 15
- Support classe “non churn” = 85

Aiuta a capire quanto sono affidabili le metriche:
- Se il support di una classe è molto basso (es: solo 3 churn su 100), precision e recall su quella classe possono essere instabili e poco rappresentative.

Indica la presenza di classi sbilanciate:
- Fa subito vedere se si sta lavorando con un dataset in cui una classe è molto più frequente dell’altra (caso tipico del churn).

METRICHE

Cos’è la confusion matrix?

Definizione:
La **confusion matrix** è una tabella che mostra, per ogni classe, **quante predizioni del modello sono corrette e quante sono sbagliate**.

Come funziona:

Ogni riga rappresenta la **classe reale** (la verità).
Ogni colonna rappresenta la **classe predetta dal modello**.
Si usa soprattutto per classificazione binaria (ma anche multi-classe).

Perché è utile?

- Ti permette di vedere **esattamente dove il modello sbaglia** (es: trova pochi churn, fa troppi falsi allarmi…)
- Da questa matrice si ricavano tutte le metriche principali (accuracy, precision, recall, F1)

Esempio (binario, “churn” sì/no):

	Predetto: No	**Predetto: Sì**
Reale: No	TN	FP
Reale: Sì	FN	TP

METRICHE

Confusion matrix binaria sbilanciata

```
Confusion matrix:
 [[80   10]
 [  3    7]]
```

	Predetto: No	Predetto: Sì
Reale: No	80	10
Reale: Sì	3	7

Interpretazione:

- Il modello trova 7 churn su 10 veri (TP), ma fa anche 10 falsi allarmi (FP).
- Ci sono 3 churn non individuati (FN).

Confusion matrix multi-classe (3 classi: “basso rischio”, “medio”, “alto”)

	Predetto: Basso	Predetto: Medio	Predetto: Alto
Reale: Basso	20	5	0
Reale: Medio	4	16	5
Reale: Alto	1	2	12

Interpretazione:

- La diagonale (20, 16, 12) sono le predizioni giuste.
- Tutti gli altri sono errori di classificazione, ad es. 5 clienti “medi” scambiati per “alti”, 4 “medi” scambiati per “basso”.

ESERCIZI PRATICI PER CALCOLARE LE METRICHE

Churn binario (20 clienti)

	Predetto: No	Predetto: Sì
Reale: No	10	2
Reale: Sì	1	7

Domanda:
Calcola accuracy, precision, recall, F1 per la classe “churn”.

ESERCIZI PRATICI PER CALCOLARE LE METRICHE

Churn binario (20 clienti)

	Predetto: No	Predetto: Sì
Reale: No	10	2
Reale: Sì	1	7

Domanda:
Calcola accuracy, precision, recall, F1 per la classe “churn”.

Soluzione:

- TP = 7, FP = 2, TN = 10, FN = 1
- **Accuracy:** (7+10)/20 = 17/20 = 85%
- **Precision:** 7/(7+2) = 7/9 ≈ 78%
- **Recall:** 7/(7+1) = 7/8 = 87.5%
- **F1:** 2 × (0.78×0.875)/(0.78+0.875) ≈ 0.824 (82%)

ESERCIZI PRATICI PER CALCOLARE LE METRICHE

Dati molto sbilanciati (100 clienti):

	Predetto: No	Predetto: Sì
Reale: No	88	2
Reale: Sì	7	3

Domanda:
Calcola accuracy, precision, recall, F1 per la classe “churn”.

Soluzione:

- TP = 3, FP = 2, TN = 88, FN = 7
- **Accuracy:** (3+88)/100 = 91%
- **Precision:** 3/(3+2) = 60%
- **Recall:** 3/(3+7) = 30%
- **F1:** 2 × (0.6×0.3)/(0.6+0.3) = 0.4 (40%)

COME LEGGERE LE METRICHE NEI REPORT?

Nei report di sklearn (classification_report), le metriche (accuracy, precision, recall, F1, support) sono mostrate **per ogni classe**.

Importante:

- Non limitarti all'accuracy globale!
- Controlla sempre le metriche della **classe minoritaria** (es: churn), spesso sono le più rilevanti per il business.

Esempio di output:

```
Classification report:
              precision    recall  f1-score   support

           0       0.80      0.96      0.87       105
           1       0.86      0.48      0.62        50

    accuracy                           0.81       155
   macro avg       0.83      0.72      0.74       155
weighted avg       0.82      0.81      0.79       155
```

Suggerimento:

La media "macro" e "weighted" danno una panoramica, ma *il support avverte se la classe è poco rappresentata*.

COME LEGGERE LE METRICHE NEI REPORT?

Cosa chiedersi leggendo i risultati

- La classe importante ha recall e precision accettabili?
- C'è un forte sbilanciamento tra le metriche delle due classi?
- Il support è sufficiente per fidarsi delle metriche?
- Quale tipo di errore è più "costoso" per il business?
- Meglio qualche falso positivo o meglio non perdere nessun vero positivo?
- Serve migliorare il modello o serve più dato?

Best practice per presentare i risultati

- **Mostra sempre la confusion matrix** insieme alle metriche: rende subito evidenti gli errori.
- **Evidenzia la classe di interesse** (spesso la minoritaria/churn):
  - Metti in risalto recall e F1 di quella classe.
- **Spiega i limiti:**
  - "Con pochi dati la precisione/recall può oscillare"
  - "Il modello trova solo 3 churn su 10: serve migliorare recall"
- **Racconta l'impatto business:**
  - "Quanti clienti churn si rischiano di perdere?"
  - "Vale la pena disturbare 10 clienti per trovarne 1 davvero a rischio?"
- **Usa grafici semplici** (es: barre di precision/recall, confusion matrix visuale).

K-FOLD CROSS-VALIDATION STRATIFICATO

Cos'è:
Tecnica per valutare il modello **spezzando i dati in "K" parti (fold)**.
Il modello viene addestrato su K-1 parti e testato sulla parte restante, a rotazione.

Perché stratificato?
- Assicura che **ogni fold abbia la stessa proporzione di classi** (es: churn sì/no) del dataset originale.
- Fondamentale nei casi sbilanciati, per evitare che qualche fold resti senza casi positivi.

Vantaggi:
- Dà una stima più affidabile delle performance su dati nuovi.
- Riduce la dipendenza da un singolo split fortunato/sfortunato.

Esempio pratico:
5-fold stratificato su 100 clienti → 5 test diversi da 20 clienti ciascuno, sempre con la stessa percentuale di churn.

COS'È LA CURVA ROC E L'AUC

ROC (Receiver Operating Characteristic):

- È una curva che mostra come variano **veri positivi** e **falsi positivi** al cambiare della soglia di decisione.
- Sull'asse X: **tasso di falsi positivi** (FPR)
- Sull'asse Y: **tasso di veri positivi** (TPR = recall)

AUC (Area Under the Curve):

- L'area sotto la curva ROC.
- Varia tra 0.5 (modello casuale) e 1 (modello perfetto).
- Più l'AUC è vicina a 1, meglio il modello distingue le classi.

Quando è utile:

- Nei casi sbilanciati, AUC è più informativo dell'accuracy.
- Aiuta a scegliere la soglia ottimale secondo le esigenze di business.

```
y_prob = model.predict_proba(X_test)[:, 1]

# Calcola ROC e AUC
fpr, tpr, soglie = roc_curve(y_test, y_prob)
auc = roc_auc_score(y_test, y_prob)
```

LETTURA E USO DI ROC-AUC NELLA PRATICA

Come si legge:

- Una ROC “alta e a sinistra” indica un modello capace di distinguere bene tra classi.
- Una curva vicino alla diagonale significa modello poco utile.

AUC in sintesi:

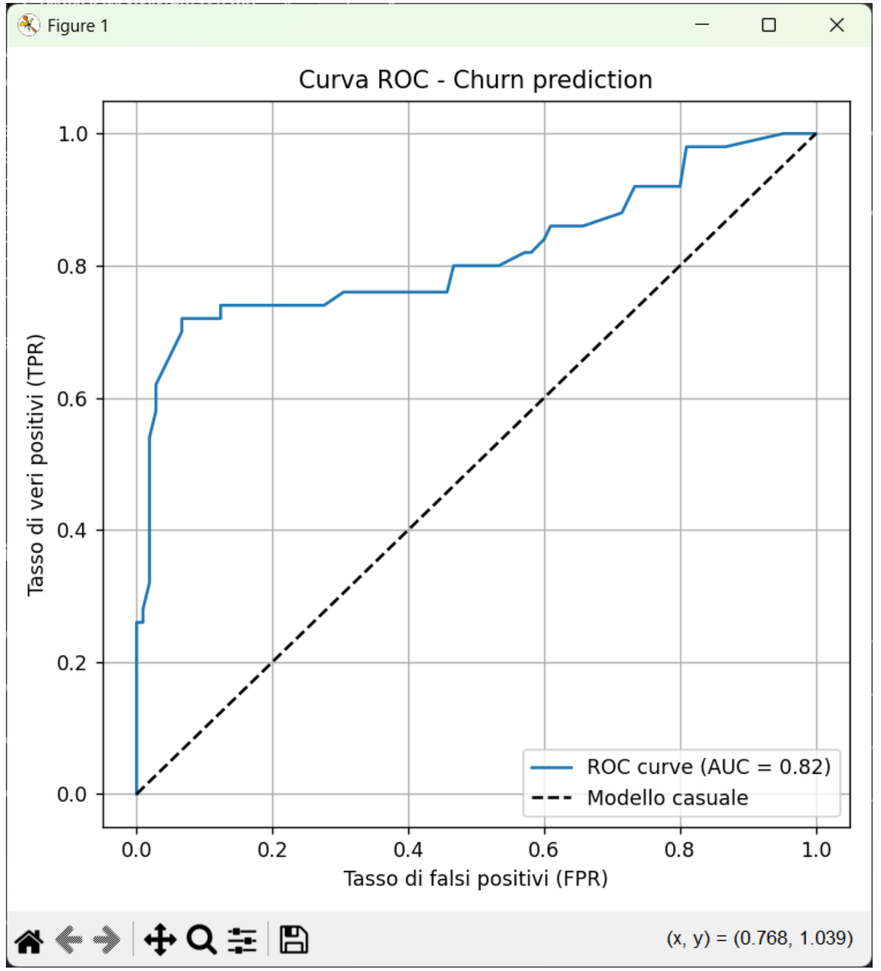
- **AUC > 0.9:** eccellente
- **AUC 0.8–0.9:** buono
- **AUC 0.7–0.8:** discreto
- **AUC < 0.7:** debole

Come usarlo:

- Confronta più modelli tramite il loro AUC.
- Usalo insieme alle altre metriche (precision, recall) per una valutazione completa.

Esempio visuale:

- Mostra una curva ROC con AUC evidenziato.



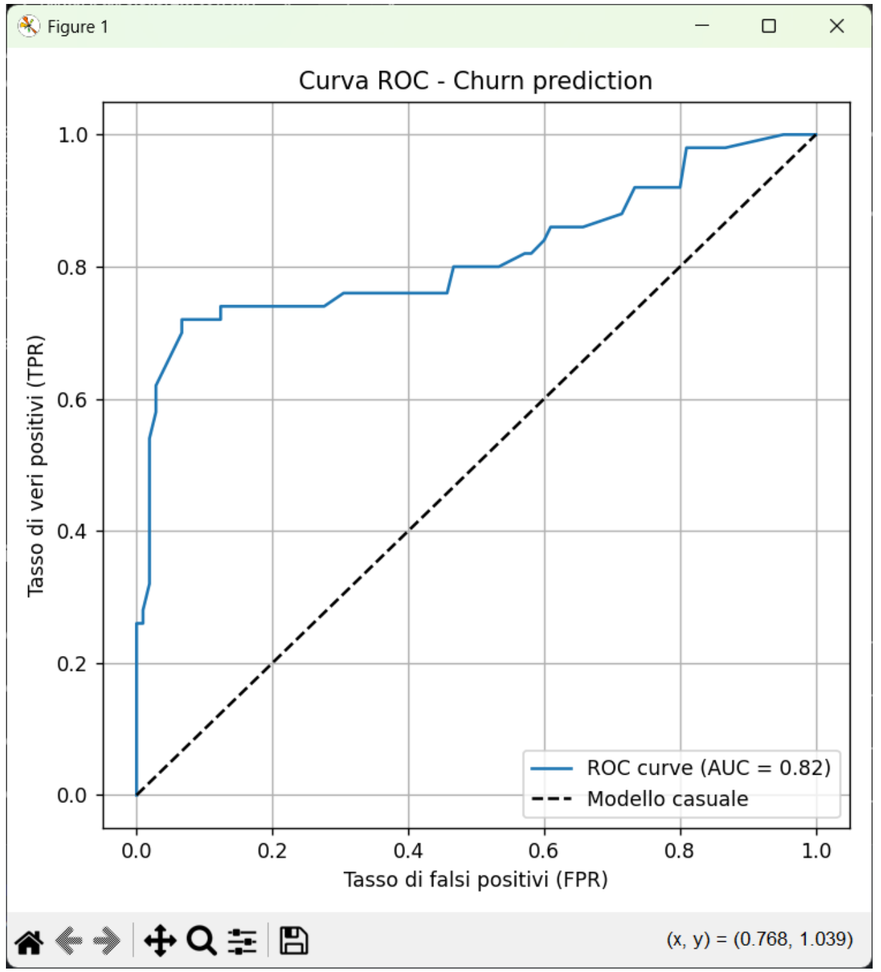
ESERCIZI DI K-FOLD E AUC-ROC

Esercizio personale:
Eseguire gli esercizi e spiegare i risultati.

```
python G2E4_K-fold.py
Accuracy media sui 5 fold: 0.88
AUC media sui 5 fold: 0.86
```

Cosa fa questo codice:

- Divide i dati in **5 partizioni stratificate**, mantenendo la proporzione tra classi.
- Addestra e testa il modello 5 volte, ogni volta su una porzione diversa.
- Calcola **accuracy** e **AUC** per ogni fold e stampa le medie finali.




```
python G2E4_AUC_ROC.py
```

SOLUZIONE ESERCIZI DI K-FOLD E AUC-ROC

Accuracy media sui 5 fold

Cos'è:
È la **media dell'accuracy** (percentuale di predizioni corrette) calcolata su ciascun fold della cross-validation.

Significato pratico:
Indica, in media, quante volte il modello "indovina" la classe giusta, su tutte le suddivisioni dei dati.
Più l'accuracy è alta (vicina a 1 o 100%), più il modello funziona bene nel distinguere tra clienti churn e non-churn.

Vantaggio:
Non è una stima "casuale": deriva dalla performance su **più test diversi**, quindi è **più affidabile** rispetto a un solo train/test split.

SOLUZIONE ESERCIZI DI K-FOLD E AUC-ROC

AUC media sui 5 fold

Cos'è:
È la **media dell'AUC** (Area Under the Curve) della curva ROC calcolata su ciascun fold.

Significato pratico:
L'AUC misura la **capacità del modello di distinguere tra classi** (ad esempio, clienti churn vs non churn), su tutte le soglie possibili.
Un valore vicino a 1 indica un modello ottimo; 0.5 è un modello casuale.

Cosa indica:
Valori alti di AUC (es: >0.8) significano che il modello è bravo a separare i casi positivi da quelli negativi.
Anche qui, la media sui fold rende la valutazione più solida.

In sintesi:
- **La curva** ROC è **il grafico** di tutte le combinazioni FPR/TPR (tasso di falsi positivi / tasso di veri positivi).
- **L'AUC** è **il valore numerico** dell'area sotto quella curva.
- **Entrambi** vengono calcolati usando le probabilità di output del modello (non solo le predizioni 0/1).

DOMANDE?

PAUSA

HYPERPARAMETER TUNING: PERCHÉ SERVE?

Cos'è:
Scegliere i "parametri di controllo" di un modello (non appresi dai dati) per ottimizzare le sue performance.

Esempi di iperparametri:
- Profondità degli alberi (max_depth),
- Numero di alberi (n_estimators),
- Learning rate (per XGBoost),
- Numero dei neuroni nei strati (per neural network).

```
model_xgb = XGBClassifier(
    n_estimators=100,
    max_depth=8,
    learning_rate=0.01,
    use_label_encoder=False,
    eval_metric='logloss',
    random_state=42
)
```

Perché è fondamentale:
- Parametri sbagliati = rischio overfitting o underfitting
- Il tuning può aumentare di molto la qualità delle previsioni!

Obiettivo:
- Trovare la combinazione che dà le migliori metriche su validation/test.

GRID SEARCH: RICERCA ESAUSTIVA DEGLI IPERPARAMETRI

Cos'è:
Prova **tutte le combinazioni possibili** di un insieme predefinito di valori per ciascun iperparametro.

Come funziona:

- Esempio:
  - max_depth = [3, 5, 7]
  - n_estimators = [100, 200]
  - Learning rate = [0.01, 0.1]
- Vengono provate tutte le combinazioni (es: 3×2×2 = 12 modelli).

Pro:

- Semplice da implementare, trova sicuramente il massimo tra le combinazioni scelte.

Contro:

- Diventa molto lento con tanti parametri e valori ("esplosione combinatoria").

Quando usarla:

- Set di parametri piccoli o medi.

BAYESIAN OPTIMIZATION: TUNING INTELLIGENTE

Cos'è:

- Tecnica di **ottimizzazione intelligente** che, invece di provare tutte le combinazioni, "impara" quali zone degli iperparametri è più promettente.
- Usa la probabilità e modelli statistici per **scegliere la prossima combinazione da testare**.

Pro:

- Molto più veloce su spazi grandi di iperparametri.
- Trova combinazioni ottime con meno tentativi.

Contro:

- Più complessa da impostare rispetto a GridSearch.

Quando usarla:

- Quando hai molti iperparametri e vuoi ottimizzare tempo/calcolo.

Esempi di librerie:

- Più complessa da impostare rispetto a GridSearch.
- optuna, scikit-optimize, hyperopt, bayesian-optimization.

OTTIMIZZAZIONE E TUNING NEI MODELLI

Profice

Confronto tra i due metodi

Metodo	Pro	Contro	Quando usarlo
Grid Search	Semplice, esaustivo	Lento con tanti parametri	Parametri pochi
Bayesian Opt.	Efficiente, “intelligente”	Più difficile da settare	Parametri molti

Nel Machine Learning, costruire un buon modello non si limita alla scelta dell’algoritmo giusto: è fondamentale anche **ottimizzare i parametri che lo controllano**, detti iperparametri.
Questa fase, chiamata *hyperparameter tuning*, permette di trovare il miglior equilibrio tra accuratezza, robustezza e capacità di generalizzare su nuovi dati.
Per ottimizzare i modelli esistono diverse strategie: dalla ricerca esaustiva (GridSearch), semplice ma lenta, fino agli approcci più intelligenti e rapidi come la Bayesian Optimization, che imparano progressivamente dove “conviene cercare”.
Questi metodi, combinati con una valutazione rigorosa tramite tecniche come la cross-validation e metriche adeguate (accuracy, recall, ROC-AUC), aiutano a ottenere modelli affidabili e adatti agli obiettivi reali del business.

ESERCIZIO: OTTIMIZZAZIONE E TUNING NEI MODELLI

Profice

Modello e parametri originali di XGBoost

```
model_xgb = XGBClassifier(
   n_estimators=100,
   max_depth=8,
   learning_rate=0.01,
   use_label_encoder=False,
   eval_metric='logloss’,
   random_state=42
)
```

Utilizziamo parametri dinamici

```
param_grid = {
    "n_estimators": [100, 150],
    "max_depth": [3, 5],
    "learning_rate": [0.01, 0.1],
    "use_label_encoder": [False],
    "eval_metric": ["logloss"],
    "random_state": [42]
}
```

Usiamo GridSearch per XGBoost

```
xgb = XGBClassifier()
grid = GridSearchCV(
    estimator=xgb,
    param_grid=param_grid,
    scoring="roc_auc",
    cv=2,
    n_jobs=-1,
    verbose=1
)
```

Addestramento e predizione del miglior modelli con parametri ottimali

```
grid.fit(X_train, y_train)

print("Migliori parametri trovati:",
grid.best_params_)

best_model = grid.best_estimator_
```

ESERCIZIO: OTTIMIZZAZIONE E TUNING NEI MODELLI

Risultati:

```
Migliori parametri trovati: {'eval_metric': 'logloss',
'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 150,
'random_state': 42, 'use_label_encoder': False}
AUC finale sul test set: 0.81
Accuracy finale sul test set: 0.75
Confusion matrix:
 [[21  7]
 [ 3  9]]
```

- Il modello finale XGBoost è stato addestrato con questi iperparametri, che sono risultati **ottimali** tra quelli testati durante la GridSearch.
- In particolare:
- **max_depth: 5** → Ogni albero può avere fino a 5 livelli.
- **n_estimators: 150** → Sono stati usati 150 alberi “weak learners” per formare il modello finale.
- **learning_rate: 0.1** → Quanto “pesa” ogni albero aggiunto nella sequenza.
- Gli altri sono parametri fissi per compatibilità e riproducibilità.

ESERCIZIO: OTTIMIZZAZIONE E TUNING NEI MODELLI

Risultati:

```
Migliori parametri trovati: {'eval_metric': 'logloss',
'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 150,
'random_state': 42, 'use_label_encoder': False}
AUC finale sul test set: 0.81
Accuracy finale sul test set: 0.75
Confusion matrix:
 [[21  7]
 [ 3  9]]
```

AUC finale sul test set: 0.81

- **AUC = 0.81** significa che il modello distingue bene tra clienti churn e non churn.
  Un valore superiore a 0.8 è generalmente considerato buono.
- **AUC** misura quanto il modello è “discriminante” su tutte le soglie di probabilità (non solo su una soglia fissa come 0.5).

ESERCIZIO: OTTIMIZZAZIONE E TUNING NEI MODELLI

Risultati:

```
Migliori parametri trovati: {'eval_metric': 'logloss',
'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 150,
'random_state': 42, 'use_label_encoder': False}
AUC finale sul test set: 0.81
Accuracy finale sul test set: 0.75
Confusion matrix:
 [[21  7]
 [ 3  9]]
```

Accuracy finale sul test set: 0.75

- L'**accuracy** del modello è del 75%.
  Significa che il modello **prevede correttamente 3 clienti su 4** nel test set.
- Da sola l'accuracy non dice tutto, specialmente se le classi sono sbilanciate (ma qui AUC è buono, quindi ok).

ESERCIZIO: OTTIMIZZAZIONE E TUNING NEI MODELLI

Risultati:

```
Migliori parametri trovati: {'eval_metric': 'logloss',
'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 150,
'random_state': 42, 'use_label_encoder': False}
AUC finale sul test set: 0.81
Accuracy finale sul test set: 0.75
Confusion matrix:
 [[21  7]
 [ 3  9]]
```

Confusion matrix

- **21** = veri negativi (**clienti NON churn previsti correttamente**)
- **7** = falsi positivi (**clienti NON churn previsti churn per errore**)
- **3** = falsi negativi (**clienti churn non individuati**)
- **9** = veri positivi (**clienti churn previsti correttamente**)

ESERCIZIO: OTTIMIZZAZIONE E TUNING NEI MODELLI

Cosa ha trovato il modello?

- Riconosce bene i clienti che non faranno churn (**21 su 28**, ≈75%).
- Trova **9 clienti churn su 12** (≈75% recall sulla classe “churn”).
- Fa alcuni errori (3 falsi negativi, 7 falsi positivi), ma nel complesso è **molto bilanciato**.
- L’AUC alto indica che la probabilità assegnata dal modello separa bene i due gruppi.

In sintesi

- **Modello ben ottimizzato, adatto per predire churn.**
- La qualità delle previsioni è **buona sia per accuracy che per AUC**.
- La confusion matrix mostra una **buona capacità di riconoscere sia chi farà che chi non farà churn**, con pochi errori.

ESERCIZIO: OTTIMIZZAZIONE E TUNING NEI MODELLI

Esercizio personale:
Modifica il programma `G2E5_XGBoost_GridSearch.py` che calcoli queste metriche:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} = \frac{9 + 21}{9 + 21 + 7 + 3} = \frac{30}{40} = 0.75\ (75\%)$$

$$\text{Precision} = \frac{TP}{TP + FP} = \frac{9}{9 + 7} = \frac{9}{16} \approx 0.5625\ (56\%)$$

$$\text{Recall} = \frac{TP}{TP + FN} = \frac{9}{9 + 3} = \frac{9}{12} = 0.75\ (75\%)$$

$$\text{F1} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} = 2 \times \frac{0.5625 \times 0.75}{0.5625 + 0.75} = 2 \times \frac{0.4219}{1.3125} = 2 \times 0.3215 = 0.643\ (64\%)$$

ESERCIZIO: OTTIMIZZAZIONE E TUNING NEI MODELLI

Profice

Soluzione:

Aggiungi queste righe:

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
classification_report

# Già ottenuti: y_test (veri) e y_pred (predetti)

# Calcola metriche per la classe "churn"
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, pos_label=1)
recall = recall_score(y_test, y_pred, pos_label=1)
f1 = f1_score(y_test, y_pred, pos_label=1)

print(f"Accuracy finale sul test set: {accuracy:.2f}")
print(f"Precision (churn): {precision:.2f}")
print(f"Recall (churn): {recall:.2f}")
print(f"F1-score (churn): {f1:.2f}")

# Report completo (per entrambe le classi)
print("\nClassification report:\n", classification_report(y_test, y_pred))
```

BEST PRACTICE PER WORKFLOW SUPERVISIONATO

Best practice 1 – Comprensione del problema e degli obiettivi

- **Definisci chiaramente la domanda:**
  Cosa vuoi prevedere? È una regressione o una classificazione?
- **Conosci i dati:**
  Analizza il significato delle colonne e il contesto di business.
- **Stabilisci le metriche di successo:**
  (es: MAE per regressione, F1-score per classificazione)

Best practice 2 – Pulizia e preparazione dei dati

- **Gestisci valori mancanti e outlier:**
  Scegli se eliminarli, imputarli o correggerli.
- **Trasforma le variabili:**
  Encoding per le categoriche, scaling per i numerici se necessario.
- **Feature engineering:**
  Crea nuove variabili utili (es: ora, giorno della settimana, rolling mean).

BEST PRACTICE PER WORKFLOW SUPERVISIONATO

Best practice 3 – Suddivisione dati: train, validation, test

- **Non testare mai sul test fin dall'inizio!**
  Suddividi i dati in modo stratificato (per classi) o temporale (se serie storica).
- **Usa la validation per tuning e scelta modello.**
- **Test finale solo a tuning concluso**
  per valutare la vera generalizzazione.

Best practice 4 – Modellazione e tuning

- **Scegli modelli adatti al problema e alla quantità di dati.**
- **Esegui hyperparameter tuning**
  (GridSearch o Bayesian Optimization).
- **Evita l'overfitting:**
  Usa cross-validation, regularizzazione e fermati appena la performance peggiora in validation.

BEST PRACTICE PER WORKFLOW SUPERVISIONATO

Best practice 5 – Valutazione e interpretazione

- **Valuta più metriche, non solo una.**
- **Analizza la confusion matrix (classificazione)**
  o il residuo (regressione).
- **Interpreta il modello:**
  Importanza delle feature, grafici di errore, ROC curve.

Best practice 6 – Documentazione e riproducibilità

- **Tieni traccia di tutti i passaggi, modelli e parametri.**
- **Salva codice, pipeline e versioni dei dati.**
- **Scrivi note su scelte fatte e motivazioni.**

Best practice 7 – Deployment e monitoraggio

- **Testa il modello su dati reali prima del rilascio.**
- **Implementa monitoraggio:**
  verifica che la qualità resti stabile nel tempo.
- **Aggiorna e riaddestra periodicamente**
  se cambiano i dati o il contesto.

RIASSUNTO DELLA GIORNATA

Cosa abbiamo imparato oggi:

- Differenza tra **classificazione** e **regressione** supervisionata
- Analisi e preparazione dei dati reali (valori mancanti, feature engineering)
- Come impostare correttamente **train, validation, test split**
- Uso pratico di modelli: **Logistic Regression, Random Forest, XGBoost, Neural Network (overview)**
- Valutazione con metriche chiave: **accuracy, precision, recall, F1-score, ROC-AUC, confusion matrix**
- Tecniche di **hyperparameter tuning** (GridSearch, Bayesian Optimization)
- Best practice di workflow:
  - interpretazione delle metriche
  - documentazione
  - riproducibilità
  - attenzione all'overfitting e validazione
- Esercizi con dati generati

PROSSIMI PASSI E MOTIVAZIONE

Cosa portiamo a casa:

- Le basi per affrontare un progetto di **Machine Learning supervisionato** con metodo e consapevolezza
- La sicurezza di poter scegliere, valutare e confrontare modelli diversi grazie alle **metriche giuste** e alle best practice
- Una **metodologia concreta** per preparare i dati, evitare errori comuni, e ottimizzare le prestazioni in modo rigoroso
- **Avere programmi e dati pronti nella propria repository**

Come continuare:

- Ripassa i materiali della giornata e prova a **replicare gli esercizi** (regressione, classificazione, tuning…) su nuovi dati
- Fai attenzione a **train/validation/test split** e alle metriche, non solo all'accuracy
- Sperimenta diverse tecniche di **tuning** e interpretazione dei risultati
- Tieni sempre traccia delle scelte, dei parametri, e confrontati con altri:
  il modello migliore nasce sempre dal confronto

Domande finali? Feedback?

- Lo spazio è aperto:
  fai tutte le domande che hai, suggerisci temi o strumenti da approfondire nelle prossime giornate!

GRAZIE PER L'ATTENZIONE