Linee Guida per il Progetto di Esame

Programming and Machine Learning for Biological Data (PMLBD)

Sapienza Università di Roma — LM Genetica e Biologia Molecolare

A chi è rivolto questo documento — Queste linee guida spiegano come realizzare il progetto di esame di PMLBD. L'obiettivo non è fornire una ricetta da seguire meccanicamente, ma aiutarti a **capire cosa stai facendo e perché**. Chi valuta il progetto vuole vedere non solo che il codice funziona, ma che tu abbia compreso le ragioni di ogni scelta metodologica, i rischi degli errori più comuni e le implicazioni dei risultati nel contesto del problema biologico o clinico scelto.

Le linee guida seguono l'ordine esatto del workflow che dovrai implementare. Tra una sezione e l'altra trovi riquadri colorati: ⚠️ **Attenzione** segnala errori comuni e rischi di data leakage; ✅ **Pratica raccomandata** indica la soluzione corretta; ℹ️ **Nota metodologica** approfondisce il ragionamento dietro una scelta.

Struttura del documento

1. Scelta e descrizione del dataset
2. Esplorazione iniziale (EDA)
3. Preprocessing prima dello split
4. Split del dataset
5. Preprocessing dopo lo split — la Pipeline
6. Selezione e confronto dei modelli
7. Ottimizzazione degli iperparametri (Hyperparameter Tuning)
8. Valutazione finale e verifica di overfitting/underfitting
9. Analisi dell'importanza delle feature (Feature Importance)
10. Conclusioni

Appendice A — Il Data Leakage: riepilogo visivo

Appendice B — Checklist finale prima della consegna

1. Scelta e descrizione del dataset

1.1 Criteri di scelta

Il dataset può essere scelto da Kaggle e deve prestarsi a un problema di **classificazione supervisionata binaria o multiclasse**. Prima di sceglierlo, verifica che:

- **Il target sia categorico** (binario o multiclasse) e che abbia un significato chiaro e interpretabile nel contesto del problema.

- **Il numero di osservazioni sia adeguato**: con meno di 200 righe i risultati statistici saranno molto instabili. Idealmente, usa almeno 500–1000 osservazioni.
- **Le feature abbiano senso**: ci devono essere variabili con una relazione plausibile con il target.
- **Il rapporto tra numero di data points e numero di features sia adeguato**: come regola generale, sono necessarie almeno 10-20 osservazioni per ogni feature, per ridurre il rischio di overfitting e ottenere stime statisticamente stabili.
- **Non ci siano feature che costituiscono data leakage strutturale**: variabili che codificano direttamente il target o disponibili solo dopo l'evento da predire (es. 'final_diagnosis' per predire una diagnosi, 'final_score' per predire il successo).

1.2 Descrizione del dataset nel notebook

Dedica la prima sezione del notebook alla presentazione del dataset. Questa sezione deve contenere tutti gli elementi seguenti — non sono opzionali:

1. **Contesto del problema:** Qual è il problema reale? Perché è rilevante? In ambito clinico o biologico, cosa significa predire correttamente il target? Cosa comporta un falso negativo o un falso positivo?
2. **Fonte:** Link al dataset su Kaggle. Numero di osservazioni e di feature.
3. **Descrizione del target:** Qual è la variabile target? Quante classi ha? Cosa rappresenta ciascuna classe? Come è codificata nel dataset?
4. **Tabella delle feature:** Per ogni variabile scrivi: nome, tipo (numerica continua / categorica nominale / categorica ordinale / binaria), range o valori possibili, e significato nel contesto del problema.

> ℹ️ **Nota metodologica**
>
> La distinzione tra tipo di variabile non è un formalismo: determina direttamente quale preprocessing applicare.
>
> **Variabile numerica continua** (es. Age, budget in USD) → standardizzazione con StandardScaler.
>
> **Variabile categorica nominale** (3+ categorie senza ordine, es. paese, genere cinematografico) → One-Hot Encoding nella Pipeline.
>
> **Variabile categorica ordinale** (con ordine, es. basso/medio/alto) → mapping numerico deterministico che rispetti l'ordine.
>
> **Variabile binaria** (già 0/1, o mappata a 0/1) → passthrough, nessuna trasformazione.
>
> Applicare lo stesso preprocessing a tutti i tipi di variabile è un **errore concettuale**.

1.3 Attenzione alla creazione del target

Quando il target non è già presente nel dataset e deve essere creato binarizzando una variabile continua (es. score ≥ 7 → film di successo), è necessario:

- **Mostrare la distribuzione della variabile sorgente** prima di scegliere la soglia (istogramma).

- **Motivare la soglia scelta** con argomenti quantitativi (es. mediana, percentile) o bibliografici.
- **Eliminare immediatamente la variabile sorgente** dalle feature dopo aver creato il target.
- **Verificare il bilanciamento delle classi risultante**.

> ⚠️ **Attenzione — Data Leakage**
> Se crei il target binarizzando una variabile continua (es. score_high da score) e NON rimuovi la variabile sorgente dalle feature, il modello userà quel valore per ricostruire banalmente la soglia.
>
> Le metriche saranno eccellenti (>95%) ma completamente prive di significato predittivo reale: stai addestrando il modello a ricordare una regola che hai scritto tu, non a trovare pattern nei dati.

2. Esplorazione iniziale dei dati (EDA)

L'EDA non è decorativa: serve a capire la struttura dei dati e a prendere decisioni informate sul preprocessing. **Ogni output deve essere commentato**: non basta mostrare un grafico o una tabella, devi spiegare cosa osservi e come questo influenza le scelte successive.

2.1 Ispezione iniziale

- Visualizza le prime righe con `df.head()` e verifica che i dati abbiano senso.
- Controlla `df.shape`, `df.dtypes` e `df.nunique()` per ogni colonna.
- **Identifica le variabili che sembrano numeriche ma sono categoriche** (es. Gender=1/2, Group=1/2). Questo è uno degli errori più frequenti: trattare queste variabili come numeri introduce ordini artificiali inesistenti.

2.2 Valori mancanti

Usa `df.isnull().sum()` per colonna. Per ogni colonna con NaN, decidi e motiva:

Situazione	Strategia raccomandata
< 5% di NaN	Imputa con media/mediana (numeriche) o moda (categoriche) — DENTRO la Pipeline, dopo lo split.
5–30% di NaN	Valuta caso per caso. Se la variabile è rilevante, imputa. Altrimenti considera la rimozione. Motiva la scelta.
> 30% di NaN	In genere conviene rimuovere la colonna. Troppi NaN rendono la variabile inaffidabile.
NaN concentrati in poche righe	Puoi rimuovere quelle righe prima dello split (non apprende informazione), ma verifica che non siano sistematicamente diverse dalle altre.

> ⚠️ **Attenzione — Data Leakage**
>
> L'imputazione (SimpleImputer) deve avvenire DOPO lo split e DENTRO la Pipeline, così le statistiche (media, mediana, moda) vengono calcolate solo sul training set. Se imputi prima, le statistiche del test set 'contaminano' il training: questo è data leakage.

2.3 Duplicati

Verifica con `df.duplicated().sum()`. I duplicati non rimossi possono far sì che la stessa riga finisca sia nel training che nel test set, rendendo le metriche artificialmente ottimistiche. Rimuovili con `df.drop_duplicates()` prima dello split e commenta quanti erano.

2.4 Distribuzione del target e bilanciamento delle classi

Prima di procedere, **verifica sempre la distribuzione del target** con `value_counts()` e un grafico. Commenta i valori numerici esatti delle proporzioni. Questo passo è fondamentale:

- **Classi bilanciate**: l'accuracy è affidabile. Non è necessario ricampionare.
- **Classi sbilanciate**: l'accuracy può essere fuorviante. Usa F1-score, precision e recall come metriche principali. Considera tecniche di resampling solo DOPO aver verificato le performance senza resampling.

> ℹ️ **Nota metodologica**
>
> Quando e come applicare il resampling:
>
> **Undersampling** (RandomUnderSampler): riduce la classe maggioritaria. Preferibile se il dataset è grande.
>
> **Oversampling sintetico** (SMOTE): genera esempi sintetici della classe minoritaria. Utile se il dataset è piccolo, ma richiede cautela: può causare overfitting se usato in modo scorretto.

2.5 Distribuzioni e correlazioni

- Visualizza le distribuzioni delle variabili numeriche principali (istogrammi, boxplot). Commenta outlier evidenti.
- Calcola la matrice di correlazione di Pearson **solo sulle variabili numeriche continue** (non su quelle categoriche codificate come numeri) e visualizzala come heatmap.
- Commenta le correlazioni più forti: tra feature (multicollinearità) e tra feature e target.
- Non usare la correlazione col target come unico criterio per eliminare feature prima del modello: alcune feature con bassa correlazione lineare possono essere rilevanti per modelli non lineari.

3. Preprocessing prima dello split

Regola fondamentale: prima dello split puoi fare solo operazioni che **non apprendono informazione dai dati**.

3.1 Cosa è ammesso prima dello split

✅ **Pratica raccomandata**

Operazioni corrette prima dello split (non apprendono dai dati):

- Rimozione di colonne inutili (identificatori come ID, nome del paziente, ecc.)
- Rimozione di righe duplicate
- Rimozione di colonne con troppi valori mancanti (> 30–50%)
- Mapping deterministico di variabili categoriche binarie (es. Gender: 1→1 Maschio, 2→0 Femmina)
- Creazione del target e rimozione della variabile sorgente (se il target è derivato da una variabile continua)
- Ricodifica del target in 0/1 (se i valori originali sono 1/2 o simili)

⚠️ **Attenzione — Data Leakage**

Operazioni NON ammesse prima dello split:

- Imputazione dei valori mancanti (SimpleImputer): impara media/mediana/moda dal dataset → data leakage
- StandardScaler o MinMaxScaler: imparano media e deviazione standard dal dataset → data leakage
- OneHotEncoder o pd.get_dummies su variabili con 3+ categorie: imparano quali categorie esistono nel dataset → data leakage
- Rimozione di outlier basata su statistiche calcolate sull'intero dataset (es. IQR, z-score)
- Selezione di feature basata su correlazioni o test statistici calcolati sull'intero dataset

3.2 Mapping deterministico delle variabili binarie

Le variabili binarie con codifica non 0/1 vanno rimappate prima dello split. Esempio:

```
df['Gender'] = df['Gender'].map({1: 1, 2: 0})  # Male→1, Female→0
```

Per ogni mapping, spiega nel markdown: quale valore corrisponde a quale categoria e perché hai scelto quale assegnare a 0 e quale a 1.

3.3 Classificazione delle variabili per il preprocessing

In questa sezione del notebook, elenca esplicitamente le variabili divise in tre gruppi, perché determineranno la struttura della Pipeline:

- **Variabili numeriche continue**: riceveranno imputazione + StandardScaler.
- **Variabili categoriche nominali a 3+ categorie**: riceveranno imputazione + OneHotEncoder.
- **Variabili binarie (0/1)**: riceveranno solo imputazione se necessario, poi passthrough.

4. Split del dataset

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=SEED, stratify=y)
```

Parametro	Perché è importante
test_size=0.2	20% per il test set (80% training). Con dataset piccoli puoi usare 0.25–0.30.
random_state=SEED	Garantisce riproducibilità. Definisci SEED=42 all'inizio del notebook e usalo ovunque.
stratify=y	Mantiene le stesse proporzioni del target in entrambi i set. SEMPRE obbligatorio per classificazione.

Dopo lo split, verifica: (1) le dimensioni di X_train e X_test; (2) la distribuzione del target in entrambi i set con `value_counts(normalize=True)`.

> ⚠️ **Attenzione — Data Leakage**
>
> Dal momento in cui esegui lo split, il test set è intoccabile fino alla valutazione finale.
>
> Non usarlo per scegliere iperparametri, non scalare su di esso, non 'sbirciare' le sue metriche durante lo sviluppo.
>
> Il test set viene usato UNA SOLA VOLTA, alla fine, per la valutazione del modello definitivo.

5. Preprocessing dopo lo split — la Pipeline

Tutto il preprocessing che 'apprende' dai dati (scaling, encoding, imputazione) deve avvenire dentro una **Pipeline sklearn**. Questo garantisce automaticamente che il fit avvenga solo sul training set e che lo stesso trasformatore fittato venga applicato al test set, senza possibilità di errori o leakage.

5.1 Struttura della Pipeline con ColumnTransformer

La struttura corretta è un `ColumnTransformer` che applica trasformatori diversi a gruppi diversi di colonne:

```
numeric_transformer = Pipeline(steps=[
```

```
        ("imputer", SimpleImputer(strategy="median")),
        ("scaler", StandardScaler())
])
categorical_transformer = Pipeline(steps=[
        ("imputer", SimpleImputer(strategy="most_frequent")),
        ("encoder", OneHotEncoder(drop="first", handle_unknown="ignore",
   sparse_output=False))
])
binary_transformer = Pipeline(steps=[
        ("imputer", SimpleImputer(strategy="most_frequent"))
])
preprocessor = ColumnTransformer(transformers=[
        ("num", numeric_transformer, numeric_features),
        ("cat", categorical_transformer, categorical_features),
        ("bin", binary_transformer, binary_features)
])
```

5.2 Perché non standardizzare le variabili binarie

Le variabili binarie (0/1) non vanno standardizzate. La standardizzazione risolve il problema di feature con range e varianze molto diverse. Le variabili binarie hanno già tutte la stessa varianza massima possibile per una variabile 0/1. Standardizzarle non porta benefici e può introdurre confusione.

5.3 OneHotEncoder vs pd.get_dummies

Usa `OneHotEncoder` di sklearn (dentro la Pipeline) invece di `pd.get_dummies`. La differenza fondamentale è:

- **pd.get_dummies** applicato prima dello split → data leakage (vede le categorie del test set). Anche applicato dopo lo split su X_train, non garantisce che X_test abbia le stesse colonne.
- **OneHotEncoder dentro la Pipeline**: viene fittato solo su X_train, poi applicato a X_test con le stesse categorie. Se nel test set compare una categoria mai vista nel training, `handle_unknown='ignore'` la gestisce senza crash.

5.4 Il modello finale nella Pipeline

Il modello classificatore viene aggiunto come ultimo step della Pipeline:

```
full_pipeline = Pipeline(steps=[('preprocessor', preprocessor), ('model',
RandomForestClassifier(random_state=SEED))])
```

In questo modo, chiamando `full_pipeline.fit(X_train, y_train)`, il preprocessor viene fittato su X_train e il modello viene addestrato sugli output del preprocessor. Chiamando `full_pipeline.predict(X_test)`, il preprocessor trasforma X_test con i parametri già fittati su X_train.

6. Selezione e confronto dei modelli

Prima di ottimizzare gli iperparametri, confronta almeno 3–4 modelli diversi usando la **cross-validation sul training set**. Questo ti permette di capire quale famiglia di modelli funziona meglio sui tuoi dati, senza toccare il test set.

6.1 Modelli consigliati da confrontare

Modello	Quando funziona bene
K-Nearest Neighbors (KNN)	Dataset piccoli/medi, feature su scala simile. Soffre in alta dimensionalità (curse of dimensionality).
Logistic Regression	Relazioni lineari tra feature e target. Veloce, interpretabile, buon baseline.
Decision Tree	Relazioni non lineari, dati misti. Rischio di overfitting se non regolato (max_depth).
Random Forest	Robusto, gestisce bene feature miste e outlier. Fornisce feature_importances_. Ottimo in molti contesti.
Gradient Boosting / XGBoost	Spesso il più preciso su dati tabulari. Più lento da ottimizzare. Buona alternativa al Random Forest.

6.2 Come confrontare i modelli correttamente

Usa `StratifiedKFold` con `cross_val_score` su X_train, per ogni modello (dentro la sua Pipeline):

```
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=SEED)
scores = cross_val_score(pipeline, X_train, y_train, cv=cv, scoring='f1')
```

Per ogni modello, riporta **media e deviazione standard** degli score di CV. La std è importante: un modello con F1 medio 0.78 ± 0.02 è più affidabile di uno con 0.80 ± 0.12.

6.3 Quale metrica usare per confrontare i modelli

Situazione	Metrica principale
Classi bilanciate	Accuracy o F1-score (macro)
Classi sbilanciate	F1-score (weighted o sulla classe minoritaria), ROC-AUC
Contesto in cui i falsi negativi costano molto (es. diagnosi medica)	Recall (sensibilità) della classe positiva

Contesto in cui i falsi positivi costano molto (es. spam filter aggressivo)	Precision della classe positiva

> ⚠️ **Attenzione — Data Leakage**
>
> Non ottimizzare gli iperparametri per ogni modello in questa fase: confronta i modelli con iperparametri di default (o con un valore ragionevole e uguale per tutti).
>
> Se ottimizzi alcuni modelli e non altri prima del confronto, il confronto non è equo.
>
> L'ottimizzazione degli iperparametri va fatta SOLO sul modello vincitore, nella fase successiva.

7. Ottimizzazione degli iperparametri

Dopo aver scelto il modello migliore, ottimizza i suoi iperparametri usando **GridSearchCV o RandomizedSearchCV** con cross-validation sul training set.

7.1 GridSearchCV nella Pipeline

```
grid_search = GridSearchCV(estimator=full_pipeline, param_grid=param_grid, cv=cv, scoring='f1', n_jobs=-1, refit=True)
grid_search.fit(X_train, y_train)
```

Parametri importanti di GridSearchCV:

Parametro	Significato
param_grid	Dizionario con gli iperparametri da esplorare. Per una Pipeline, il nome del parametro è 'step__parametro' (es. 'model__n_estimators').
cv=cv	Lo stesso StratifiedKFold usato per il confronto dei modelli.
scoring='f1'	La metrica da ottimizzare. Deve essere la stessa usata per confrontare i modelli.
refit=True	Dopo aver trovato i migliori iperparametri, ri-addestra automaticamente il modello su TUTTO il training set. Questo produce il modello finale pronto per la valutazione.
n_jobs=-1	Usa tutti i core disponibili per parallelizzare la ricerca. Velocizza molto il processo.

7.2 Quali iperparametri ottimizzare

Modello	Iperparametri principali da esplorare

KNN	n_neighbors (ampio range, es. 1–40), weights ('uniform'/'distance'), metric ('euclidean'/'manhattan')
Logistic Regression	C (forza di regolarizzazione: 0.01, 0.1, 1, 10, 100), solver ('liblinear'/'lbfgs')
Decision Tree	max_depth (None, 3, 5, 10), min_samples_split (2, 5, 10), min_samples_leaf (1, 2, 4)
Random Forest	n_estimators (50, 100, 200), max_depth (None, 5, 10), min_samples_split, min_samples_leaf

7.3 Il flusso corretto — schema

Il flusso che DEVE essere seguito:

Tutti i dati disponibili

▼

Split immediato: Test set (20%) messo da parte — MAI toccato

▼

Training set (80%) — tutto il lavoro avviene qui

▼

Preprocessing dentro Pipeline (imputer + scaler + encoder)

▼

Cross-validation per confronto modelli

▼

GridSearchCV per hyperparameter tuning del modello migliore

▼

Fit finale su TUTTO il training set (refit=True)

▼

Valutazione UNICA sul test set → metriche finali

> ⚠️ **Attenzione — Data Leakage**
> Errori comuni nel tuning:
>
> - Ottimizzare gli iperparametri valutando le performance sul test set invece che con CV sul training set: è data leakage — il modello vede indirettamente il test e le metriche finali non saranno affidabili
> - Dopo GridSearchCV, eseguire un'ulteriore CV sul `best_estimator_`: non ha senso. GridSearchCV con `refit=True` ha già fittato il modello ottimale sull'intero training set. La valutazione finale si fa una sola volta sul test set, non con un'altra CV

🎬 Usare una metrica diversa per la selezione del modello e per il tuning (es. accuracy per il confronto, f1 per GridSearchCV): i risultati non sono direttamente comparabili — usa la stessa metrica in entrambe le fasi
🎬 Non includere il preprocessor nella Pipeline di GridSearchCV: lo scaler e l'encoder vengono fittati sull'intero training set prima della CV interna, causando data leakage sui fold di validazione

8. Valutazione finale e verifica di overfitting/underfitting

Questa è la fase finale del workflow. Il modello ottimale (`grid_search.best_estimator_`) viene valutato sul test set per la prima e unica volta.

8.1 Metriche da riportare

Calcola e commenta tutte le metriche seguenti, sia sul training set che sul test set:

- **Accuracy**: percentuale di predizioni corrette. Affidabile solo con classi bilanciate.
- **Precision**: tra i campioni predetti come positivi, quanti lo sono davvero?
- **Recall (Sensibilità)**: tra i campioni realmente positivi, quanti vengono identificati?
- **F1-score**: media armonica di precision e recall. Metrica principale per classi sbilanciate.
- **Matrice di confusione**: mostra TP, TN, FP, FN. Commenta i falsi positivi e i falsi negativi nel contesto del problema reale.
- **Classification report**: riporta precision, recall e F1 per classe. Non limitarti a stamparlo: leggi i numeri e commentali.

8.2 Verifica di overfitting e underfitting

Questo passaggio è obbligatorio. Confronta le metriche sul training set con quelle sul test set:

Situazione	Interpretazione e cosa fare
Training score ≫ Test score (es. train F1=0.98, test F1=0.72)	Overfitting: il modello ha memorizzato il training set. Regola la complessità (max_depth, min_samples_leaf, C più piccolo).
Training score ≈ Test score (buon valore)	Situazione ideale: il modello generalizza bene.
Training score ≈ Test score (valore basso)	Underfitting: il modello è troppo semplice o le feature non sono sufficientemente informative.
Test score > Training score	Segnale di anomalia. Cause probabili: data leakage, test set troppo piccolo o 'fortunato'. Controlla il workflow dall'inizio.

> ⚠️ **Attenzione — Data Leakage**
>
> Un modello che performa meglio sul test set che sul training è quasi sempre un segnale che qualcosa non va nel workflow.
>
> Le cause più probabili sono: (1) data leakage — informazioni del test set sono trapelate nel training durante preprocessing o scaling; (2) test set troppo piccolo — con pochi campioni le metriche sono molto instabili; (3) split non stratificato — il test set contiene una proporzione favorevole delle classi.
>
> Se osservi questa situazione, rivedi il workflow dall'inizio, in particolare: encoding e scaling avvengono dentro la Pipeline? Il resampling è applicato solo al training set?

8.3 Curva ROC-AUC (facoltativa ma raccomandata)

La curva ROC e l'AUC (Area Under the Curve) forniscono una valutazione della capacità discriminativa del modello indipendente dalla soglia di classificazione. Un AUC di 0.5 equivale a un classificatore casuale, 1.0 a un classificatore perfetto. Sono particolarmente utili per confrontare più modelli in un unico grafico.

9. Analisi dell'importanza delle feature

L'analisi della feature importance è un passaggio **obbligatorio**. Serve a capire quali variabili il modello ha trovato più utili per la predizione, e a trarre conclusioni interpretabili nel contesto del problema biologico o clinico.

9.1 Come ottenerla in base al modello

Modello	Come ottenere la feature importance
Random Forest	rf.feature_importances_ — importanza basata sulla riduzione media dell'impurità (Gini). Direttamente accessibile.
Gradient Boosting / XGBoost	Stesso attributo feature_importances_. Simile a Random Forest.
Logistic Regression	Coefficienti (coef_) dopo standardizzazione. Un coefficiente alto (in valore assoluto) indica alta importanza.
Decision Tree	feature_importances_. Stesso approccio di Random Forest.
KNN	Non disponibile nativamente. Usa metodi model-agnostic come Permutation Importance (sklearn.inspection.permutation_importance).

9.2 Come visualizzarla e commentarla

Visualizza le feature importance come grafico a barre orizzontale, ordinando le feature dalla più alla meno importante. Mostra le prime 10–20 feature.

Nel markdown, **commenta il risultato nel contesto del problema**: quali variabili sono risultate più importanti? Ha senso biologicamente/clinicamente? Ci sono feature che ti aspettavi importanti e invece non lo sono? Ci sono variabili sorprendentemente rilevanti?

> ℹ️ **Nota metodologica**
> Attenzione all'interpretazione della feature importance per Random Forest:

9.3 Conclusioni sulla feature importance

Dopo l'analisi, scrivi un paragrafo che risponda a queste domande:

- Quali sono le 3–5 feature più importanti? Cosa hanno in comune?
- I risultati confermano conoscenze pregresse sul dominio del problema o le mettono in discussione?
- Ci sono feature che potresti raccogliere in futuro per migliorare il modello?

10. Conclusioni

Le conclusioni devono essere **esaustive e strutturate**. Non è sufficiente dire 'il modello funziona bene'. Chi legge deve capire cosa hai fatto, perché, e cosa significano i risultati.

10.1 Struttura raccomandata

5. **Sintesi del workflow:** Riepiloga brevemente ogni fase: dataset scelto, preprocessing applicato, modelli confrontati, modello selezionato, iperparametri ottimizzati.
6. **Problemi incontrati:** Hai trovato valori mancanti? Come li hai gestiti? Le classi erano sbilanciate? Hai applicato resampling? Hai incontrato overfitting? Come lo hai affrontato?
7. **Performance del modello finale:** Riporta le metriche principali (accuracy, F1, precision, recall) sul test set. Commenta la matrice di confusione nel contesto del problema: i falsi positivi o falsi negativi hanno implicazioni diverse?
8. **Interpretazione biologica/clinica:** Cosa dicono le feature importance? Quali variabili contano di più? Cosa significa questo nel contesto del problema reale?
9. **Limiti del modello:** Il dataset è piccolo? Alcune feature rilevanti non erano disponibili? I risultati potrebbero non generalizzare ad altre popolazioni? Sii onesto/a sui limiti.

10. **Possibili sviluppi futuri:** Cosa miglioreresti se avessi più dati, più tempo, o più risorse? Quali feature aggiuntive raccoglieresti?

Appendice A — Il Data Leakage: riepilogo

Il data leakage è l'errore metodologico più insidioso nel machine learning: le metriche sembrano ottime, ma il modello non funzionerà mai su dati nuovi. Si verifica quando informazioni del test set (o del futuro) 'trapelano' nel training.

A.1 Forme di data leakage più comuni

Forma di leakage	Come riconoscerlo	Come evitarlo
Preprocessing prima dello split	Scaling, encoding o imputation calcolati su tutto il dataset prima di train_test_split.	Sposta tutto dentro la Pipeline. Fai lo split prima di qualsiasi operazione che apprende dai dati.
Target leakage	Una feature è derivata dal target o disponibile solo dopo l'evento da predire.	Elimina le feature che 'vedono il futuro'. Analizza ogni variabile e chiediti: questa info è disponibile al momento della predizione?
Variabile sorgente del target nel dataset	Hai creato target da score≥7, ma score è ancora in X.	Elimina immediatamente la variabile sorgente dopo aver creato il target.
Scaler fuori dalla CV	Fitty StandardScaler su X_train, poi fai CV: i fold di validazione hanno già visto lo scaler.	Metti lo scaler DENTRO la Pipeline passata a cross_val_score o GridSearchCV.
Resampling prima dello split o fuori dalla Pipeline	Applichi SMOTE su tutto il dataset o su X_train prima della CV.	Applica resampling solo sul training set. Se usi SMOTE in CV, mettilo nella Pipeline (usa imblearn.pipeline).

Appendice B — Checklist finale prima della consegna

Prima di consegnare il progetto, verifica punto per punto:

Dataset e descrizione

- □ Il dataset è descritto con contesto, fonte, numero di osservazioni e feature.

- □ Ogni feature è classificata come numerica continua / categorica nominale / categorica ordinale / binaria.
- □ Il target è descritto (classi, significato, codifica).
- □ Se il target è stato creato da una variabile continua, la variabile sorgente è stata rimossa.

EDA

- □ I valori mancanti sono stati analizzati colonna per colonna.
- □ I duplicati sono stati verificati e rimossi se presenti.
- □ La distribuzione del target è stata visualizzata e commentata quantitativamente.
- □ Le distribuzioni delle variabili numeriche principali sono state visualizzate.
- □ La matrice di correlazione è stata calcolata solo sulle variabili numeriche continue.

Preprocessing e split

- □ Lo split è avvenuto PRIMA di scaling, encoding e imputation.
- □ Le variabili binarie sono state rimappatae a 0/1 prima dello split con mapping deterministico.
- □ Le variabili categoriche a 3+ categorie sono codificate con OneHotEncoder DENTRO la Pipeline.
- □ Lo StandardScaler è applicato solo alle variabili numeriche continue.
- □ Le variabili binarie non vengono standardizzate.
- □ train_test_split usa stratify=y.

Modelli e tuning

- □ Almeno 3 modelli diversi sono stati confrontati con CV sul training set.
- □ Il confronto usa la stessa metrica per tutti i modelli.
- □ L'ottimizzazione degli iperparametri (GridSearchCV) usa CV sul training set.
- □ GridSearchCV è configurato con refit=True.
- □ Nessuna valutazione sul test set è avvenuta durante il confronto o il tuning.

Valutazione finale

- □ Il test set è stato valutato UNA SOLA VOLTA con il modello finale.
- □ Le metriche training e test sono confrontate per verificare overfitting/underfitting.
- □ La matrice di confusione è commentata nel contesto del problema.
- □ Il classification report è letto e commentato, non solo stampato.

Feature importance e conclusioni

- □ La feature importance è calcolata e visualizzata.
- □ I risultati della feature importance sono interpretati nel contesto biologico/clinico.
- □ Le conclusioni includono: sintesi del workflow, problemi incontrati, performance finale, interpretazione, limiti, sviluppi futuri.

Markdown e commenti

- □ Ogni cella di codice è preceduta o seguita da una cella markdown che spiega COSA si fa e PERCHÉ.
- □ Ogni scelta metodologica è motivata (es. perché OHE invece di LabelEncoder? Perché random forest invece di KNN? Perché la soglia scelta per il target?).
- □ Ogni output (grafico, tabella, metriche) è commentato: non basta mostrarlo, devi dire cosa osservi.

In bocca al lupo!