UNIVERSITÀ DEGLI STUDI DI BARI ALDO MORO
DIPARTIMENTO DI INFORMATICA

Documentazione Caso di Studio
Ingegneria della Conoscenza
A.A. 2024/25
Modello per la predizione del parkinson

Gruppo di lavoro

- Carbone Giuseppe Emanuele, 758282, g.carbone33@studenti.uniba.it
- Bellomo Nicolas, 762932, n.bellomo12@studenti.uniba.it
- Lastella Nicola, 776814, n.lastella2@studenti.uniba.it

https://github.com/nbellomo506/Icon-2425

A.A 2024-2025

Sommario

Introduzione

Il software per la predizione del Parkinson nasce con l'intento di sperimentare l'integrazione di tecniche di apprendimento automatico e basi di conoscenza all'interno di un unico sistema di supporto alla diagnosi del morbo di Parkinson. A differenza dei tradizionali modelli predittivi che si limitano a fornire una classificazione, il sistema sviluppato è in grado di generare un'analisi più ricca e strutturata, che unisce la componente statistica derivata dal machine learning con una componente simbolica derivata da regole e soglie clinicamente motivate.

Obiettivi del progetto

L'obiettivo principale è quello di costruire un sistema che, dato in input il profilo di un paziente (sotto forma di file JSON), sia in grado di stimare il rischio di Parkinson e di produrre un report completo. Per raggiungere questo scopo sono stati fissati alcuni sotto-obiettivi: realizzare un classificatore supervisionato addestrato su dati clinici e vocali, definire una base di conoscenza esplicita contenente soglie e regole, esportare la KB in RDF/Turtle per garantire trasparenza e integrare le due componenti (ML e KB) in un report HTML.

Strumenti adottati

Il sistema è stato interamente sviluppato in Python, utilizzando librerie come pandas e numpy per la gestione dei dati, scikit-learn per il machine learning, rdflib per la gestione della KB in RDF, joblib per la persistenza dei modelli e matplotlib/seaborn per la produzione dei grafici. L'insieme di questi strumenti ha consentito di creare una pipeline completa, dal preprocessamento fino al ragionamento simbolico.

Analisi dei dati

La **fase di ricerca** ha l'obiettivo di esplorare il dataset in modo sistematico, per comprendere meglio la natura dei dati raccolti e le relazioni tra le variabili.
In questo progetto, il dataset contiene parametri vocali estratti dalla voce di soggetti sani e di pazienti con malattia di Parkinson, insieme alla variabile target Parkinsons_Status che indica lo stato di salute (0 = sano, 1 = malato). È strutturata in diversi punti:

Analisi descrittiva iniziale

- Si carica il dataset e si osservano le sue dimensioni, i nomi delle colonne e il tipo di variabili.
- Si calcolano statistiche di base (media, deviazione standard, minimi e massimi) per ogni feature, utili per farsi un'idea dell'ordine di grandezza dei valori.

Controllo qualità dei dati

- Si verificano la presenza di valori mancanti o anomali.
- Si osservano eventuali outlier che potrebbero influenzare le soglie della Knowledge Base.

Analisi univariata (per singola variabile)

- Vengono tracciati **istogrammi** per studiare la distribuzione di feature come Jitter (%), Shimmer(dB) e HNR.
- Con i **boxplot** si individuano valori estremi e si capisce meglio la variabilità interna di ciascuna variabile.

Analisi bivariata (variabile vs target)

- Le feature vengono confrontate direttamente con lo stato di salute (Parkinsons_Status).
- Con **boxplot separati per gruppo** (sani vs malati) e **distribuzioni sovrapposte**, si osservano differenze significative: ad esempio, valori più alti di jitter e shimmer nei malati, e valori più bassi di HNR.
- Queste differenze aiutano a capire quali variabili sono più informative per distinguere tra i due gruppi.

Analisi multivariata

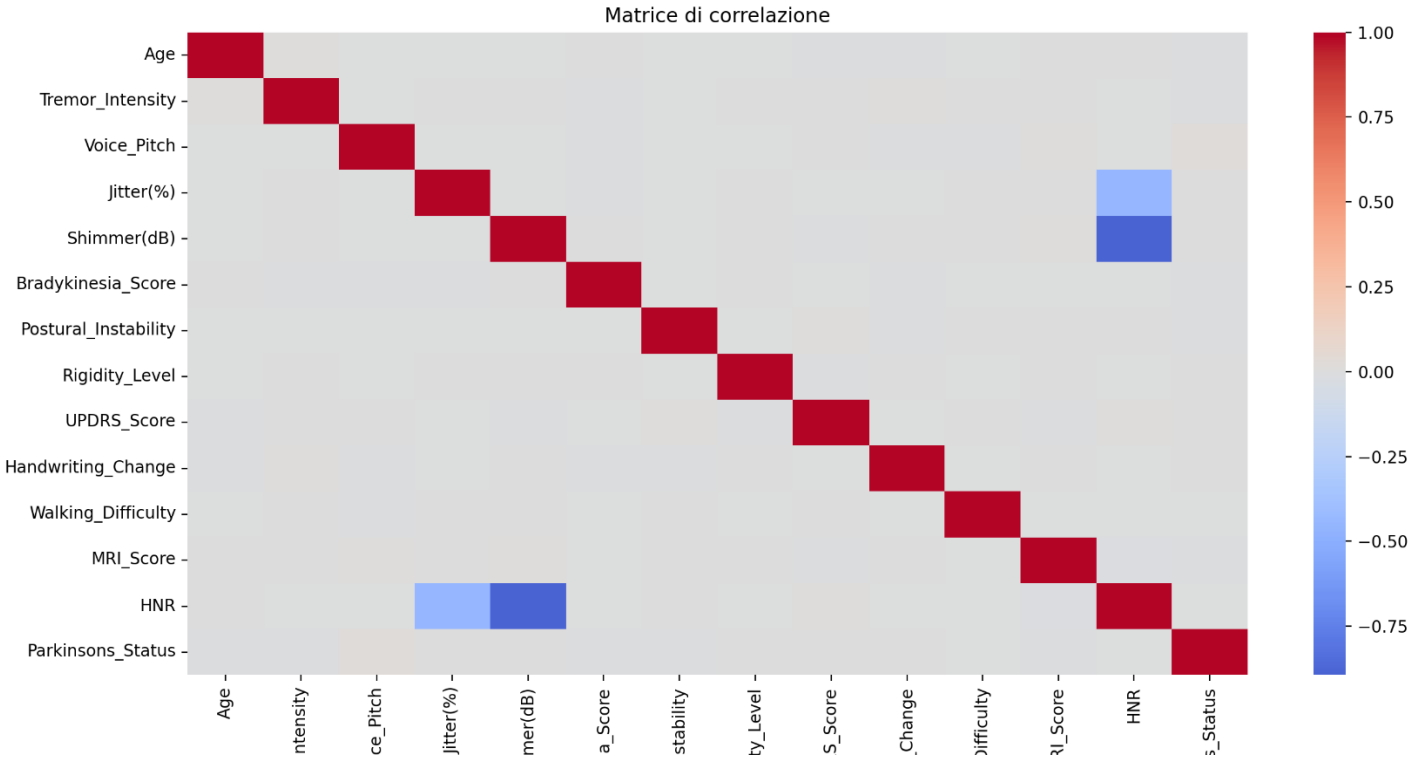
- Con la **matrice di correlazione** e la relativa heatmap si analizzano le relazioni tra le diverse feature. Questo consente di individuare variabili ridondanti (ad esempio jitter e shimmer che spesso sono correlati).
- I **pairplot** mostrano la separazione tra i gruppi su coppie di variabili, evidenziando quali combinazioni discriminano meglio sani e malati.

Individuazione di soglie candidate

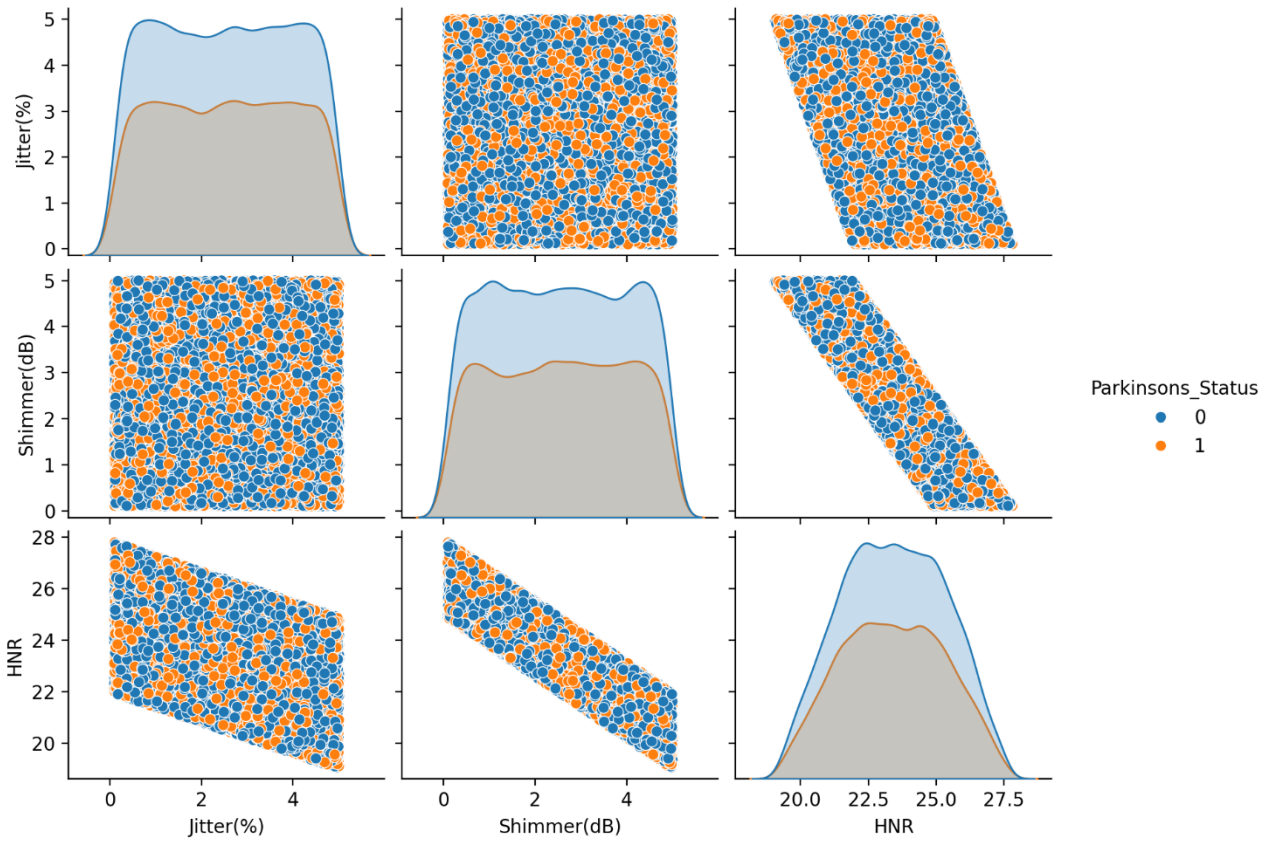
- L'EDA permette di stimare valori soglia direttamente dai dati, ad esempio usando i **quantili dei soggetti sani**:
    - 95° percentile per Jitter (%) e Shimmer(dB) (valori sopra a questa soglia indicano possibile anomalia).
    - 5° percentile per HNR (valori sotto a questa soglia indicano rischio).
- Questi valori diventano le **soglie di riferimento** che alimentano la Knowledge Base.

Per comprendere meglio la distribuzione del target è stato realizzato un grafico che mostra la proporzione tra pazienti sani e pazienti con diagnosi di Parkinson.

Per esplorare le relazioni tra le variabili numeriche è stata prodotta una matrice di correlazione rappresentata sotto forma di heatmap, che mette in evidenza eventuali correlazioni lineari.



Per esplorare le relazioni tra tutte le coppie di variabili, se ci sono pattern che distinguono i due gruppi, è stata prodotta una griglia di grafici che mostra sulla diagonale le distribuzioni univariate di ciascuna variabile mentre fuori dalla diagonale, lo scatter plot di tutte le coppie di variabili



Knowledge base (KB)

La **Knowledge Base (KB)** rappresenta il cuore logico del sistema, permettendo di derivare **informazioni cliniche interpretabili** da un insieme di misure acustiche estratte dalla voce dei pazienti.
L'obiettivo è duplice:

1. Fornire un **sistema basato su regole**, trasparente e facilmente ispezionabile.
2. Permettere l'integrazione con approcci **data-driven**, sfruttando i dati reali del dataset per affinare soglie e condizioni.

La KB combina quindi:

- **Soglie elementari** (feature → valore soglia → alert se superato).
- **Regole composte** (combinazioni logiche di più condizioni).
- **Ragionamento fuzzy** (score continuo di rischio).
- **Esportazione RDF** (per interoperabilità semantica).

Tutte queste componenti concorrono a creare un sistema che non solo "predice" o "classifica", ma soprattutto **spiega** il perché delle sue decisioni.

Strutture fondamentali

"SeverityLevel" è un'**enumerazione** che definisce tre gradi di severità per gli alert generati:

- **LOW** → segnalazione lieve, utile come indicatore iniziale.
- **MEDIUM** → alert di gravità intermedia.
- **HIGH** → rischio clinico rilevante.

Questa classificazione consente di stabilire rapidamente la priorità delle anomalie rilevate, e viene utilizzata sia per le regole singole sia per quelle composte.

“Threshold” invece è una dataclass che rappresenta la regola elementare che permette di generare alert in base a determinate condizioni.

Gli attributi principali della classe threshold sono: feature_canonical, ovvero il nome feature del dataset, value, che rappresenta il valore della soglia, source, rappresenta l’origine della soglia severity, il livello in base al l’ alert

```python
class SeverityLevel(Enum):
    LOW = 1
    MEDIUM = 2
    HIGH = 3


@dataclass
class Threshold:
    feature_canonical: str
    operator: str          # ">" oppure "<"
    value: float
    source: str            # es. "literature", "data_quantile_0.95"
    note: str = ""
    severity: SeverityLevel = SeverityLevel.MEDIUM
```

MiniKB – ExtendedKB

MiniKB è la classe base che consente di:

- Gestire il **mapping** tra colonne del dataset e nomi canonici (set_mapping).
- Inserire e recuperare soglie (insert_threshold, get_threshold).
- Esportare la KB in JSON (to_json).
- Interrogare le soglie con un'interfaccia comoda (query).

```
>>> from main import MiniKB, Threshold
>>>
>>> kb = MiniKB()
>>> kb.set_mapping({"jitter_pct": "Jitter(%)"})   # attento: set_mapping vuole un dict
>>> kb.insert_threshold(Threshold("jitter_pct", ">", 4.7, "data_quantile_0.95"))
```

Output:

	feature	operator	value	source	severity	note	dataset_column
0	jitter_pct	>	4.7	data_quantile_0.95	MEDIUM		Jitter(%)

Invece la classe ExtendedKB estende appunto MiniKB e introduce la possibilità di definire regole complesse che combinano più condizioni con logica AND.

Inoltre, contiene il metodo evaluate_patient che restituisce una lista di alert generati (soglie violate + regole composte soddisfatte) e il livello massimo di rischio osservato.

```
>>> from main import ExtendedKB, Threshold, SeverityLevel
>>> patient = {"Jitter(%)": 5.0, "Shimmer(dB)": 0.4, "HNR": 18.0}
>>> res = kb.evaluate_patient(patient)
>>> print(res)
{'alerts': [{'type': 'threshold', 'feature': 'jitter_pct', 'value': 5.0, 'threshold': 4.7, 'operator': '>',
'severity': 'MEDIUM', 'note': '', 'source': 'data_quantile_0.95'}, {'type': 'threshold', 'feature': 'shimmer
_db', 'value': 0.4, 'threshold': 0.35, 'operator': '>', 'severity': 'MEDIUM', 'note': '', 'source': 'data_qu
antile_0.95'}, {'type': 'composite', 'rule': 'Jitter e Shimmer elevati', 'severity': 'HIGH', 'note': 'Tutte
le condizioni soddisfatte'}], 'risk_level': 'High'}
```

Per rendere il sistema robusto a dataset con nomi potenzialmente diversi, la KB implementa una mappatura automatica con determinate regole di matching:

jitter_pct ↔ colonne contenenti "jitter" e %.

Shimmer_db ↔ colonne contenenti "shimmer" e "dB".

Hnr_db ↔ colonne contenenti "HNR" o "harmonic/noise ratio".

Tuttavia, i dati all'interno del dataset devono essere anche mappati in modo tale da creare delle categorie in base ai valori delle feature, infatti inizialmente i dati del dataset sono un po' esemplificativi, ma grazie alla funzione "refine_from_data()" si utilizza il dataset, scegliendo soggetti sani, per calcolare soglie più realistiche:

Feature	Operatore	Valore (≈)	Fonte
jitter_pct	>	4.7	data_quantile_0.95
shimmer_db	>	0.35	data_quantile_0.95
hnr_db	<	20.1	data_quantile_0.05

Fuzzy score

Oltre agli alert binari, la KB calcola un **fuzzy risk score** continuo ∈ [0,1].

- Per operatori ">": score = 0 sotto la soglia, cresce linearmente fino a 1 al 110% della soglia.
- Per operatori "<": score = 0 sopra la soglia, cresce linearmente fino a 1 al 90% della soglia.

Il fuzzy score è utile per:

- confrontare pazienti fra loro,
- monitorare l'evoluzione di un singolo paziente nel tempo,
- integrare i risultati in modelli ibridi ML + KB.

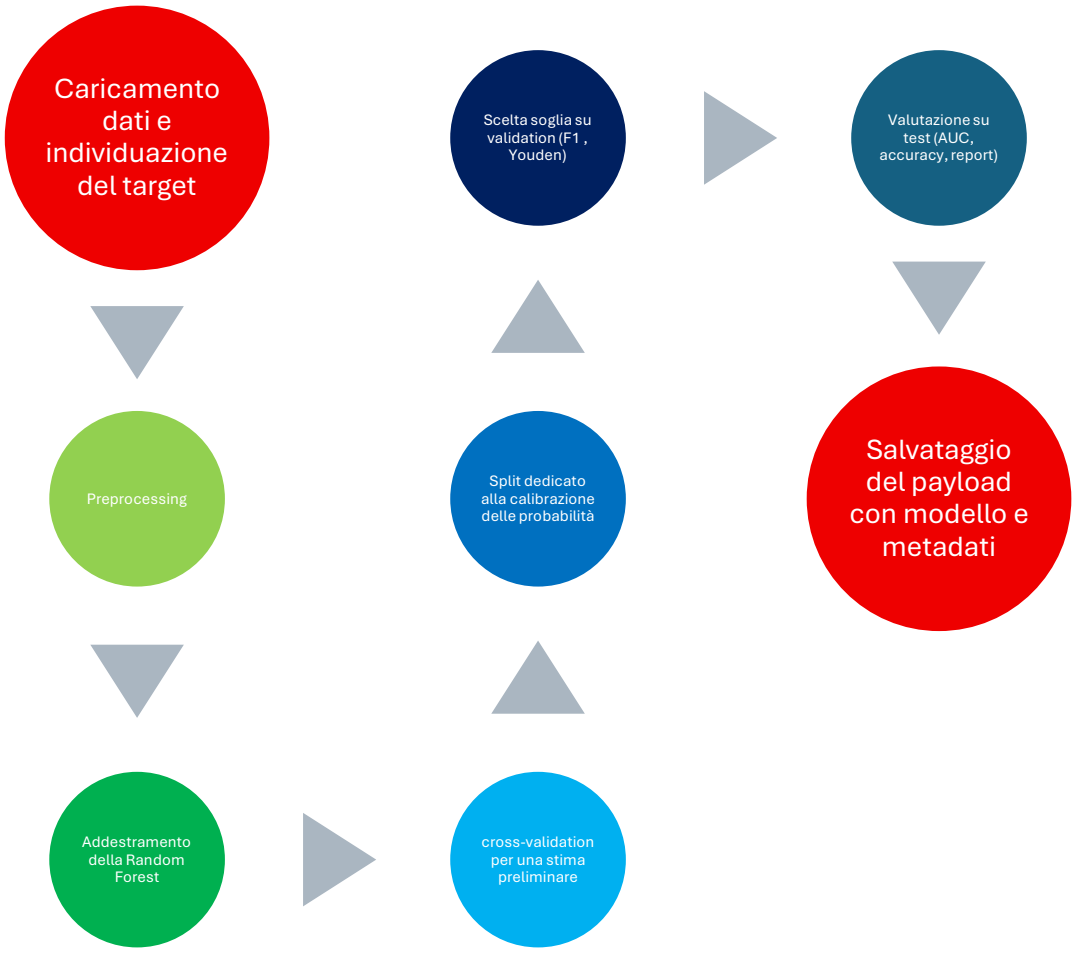
```
>>> fuzzy_score = fuzzy_risk_score(patient, kb)
>>> print(fuzzy_score)
0.8191489361702121
```

Training

Lo scopo della fase di apprendimento è quello di addestrare un classificatore con probabilità calibrate a partire dal dataset e produrre un artefatto riutilizzabile in predizione. Oltre al modello, vengono decise soglie operative (criteri F1 e Youden J), generate metriche di valutazione e salvati i metadati necessari per applicare il modello in modo coerente a nuovi casi. La fase di training prende in ingresso il dataset tabellare che contiene le feature numeriche come, ad esempio, quelle dei parametri vocali, oppure prende la colonna target binaria, oovero quella del Parkinsons_Status.

L'output atteso è un file con estensione. joblib che contiene:

- pipeline di preprocessing + classificatore,
- stima calibrata delle probabilità,
- soglie operative scelte (F1, Youden),
- lista ordinata delle feature attese,
- metadati di calibrazione.



Random Forest

È stato adottato come modello principale nella pipeline di apprendimento il Random Forest. La natura dei dati che vengono trattati è tipicamente tabellare, con grandezze acustiche eterogenee (come jitter, shimmer e HNR) e con possibili variabili aggiuntive di contesto. Queste misure non seguono relazioni semplicemente lineari: spesso interagiscono tra loro in modo non banale e presentano asimmetrie, code lunghe e outlier. In questo scenario, Random Forest risulta particolarmente adatto perché combina molti alberi di decisione—ognuno addestrato su campioni di dati leggermente diversi e su sottoinsiemi casuali di variabili—e ne media le previsioni. Questa strategia (bagging con sotto-campionamento di feature) riduce la varianza del singolo albero e fornisce un classificatore stabile, capace di catturare interazioni e non linearità senza bisogno di costruire manualmente termini polinomiali o trasformazioni complesse. Per T alberi con varianza σ^2 e correlazione ρ, la varianza dell'output medio è approssimativamente:

$$Var(f^{-})\approx\rho\sigma\text{^}2+((1-\rho)\ \sigma\text{^}2)/T$$

Un ulteriore vantaggio pratico è la robustezza del modello rispetto alla scala delle feature e alla presenza di valori anomali. Mentre nella nostra pipeline standardizziamo e imputiamo i dati per coerenza e per agevolare eventuali modelli alternativi, Random Forest funziona bene anche quando le variabili sono su scale diverse e quando alcuni punti sono rumorosi. Questo si traduce in una messa a terra più rapida: con pochi iperparametri ragionevoli—numero di alberi, profondità massima, dimensione minima delle foglie—si ottengono prestazioni solide e riproducibili. Inoltre, la possibilità di usare pesi di classe bilanciati aiuta a gestire eventuali squilibri tra sani e malati senza introdurre passaggi di risampling che complicherebbero la procedura.

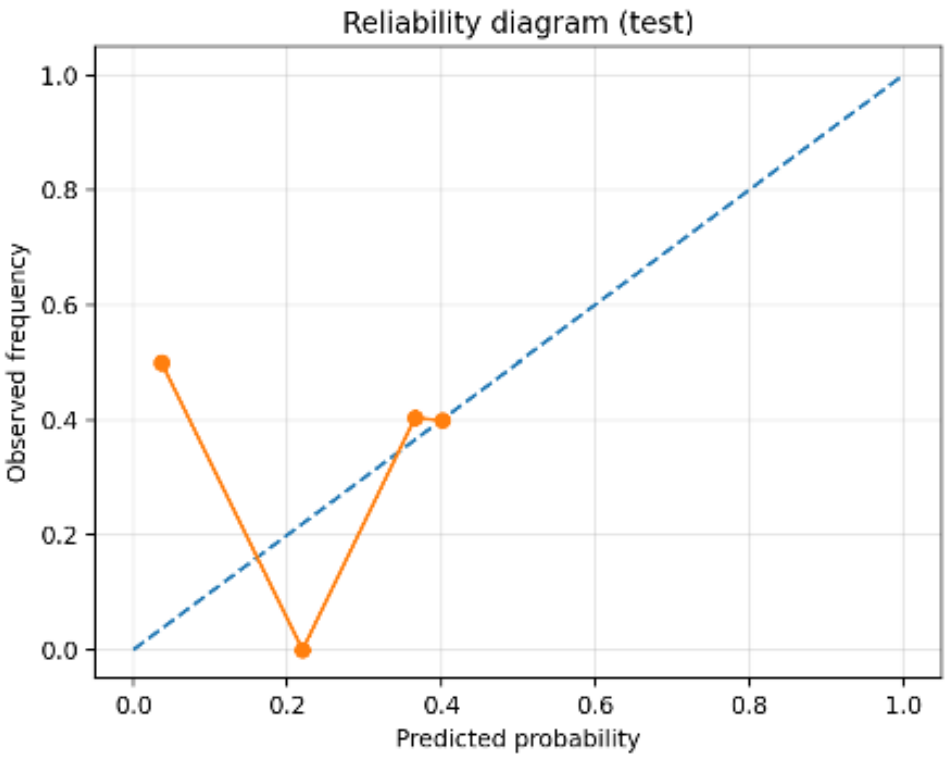
In termini di interpretabilità, pur non essendo trasparente come un singolo albero, Random Forest consente di ottenere indicazioni utili su "chi conta di più" attraverso le importanze di variabile e, quando necessario, tramite analisi come le partial dependence. Per il nostro contesto questo livello di interpretabilità è sufficiente: vogliamo un modello che performi bene, ma che al tempo stesso permetta di spiegare a grandi linee perché certe osservazioni vengono giudicate

più a rischio (ad esempio perché un aumento congiunto di jitter e shimmer, in presenza di HNR basso, sposta la probabilità verso la classe “malato”).

Un aspetto cruciale del nostro flusso è la **calibrazione delle probabilità** a valle del training. Le probabilità “native” prodotte da Random Forest possono risultare troppo ottimistiche o conservative; per questo, separiamo un set di calibrazione e applichiamo isotonic regression o Platt scaling a seconda della disponibilità di dati. In questo modo otteniamo probabilità affidabili, fondamentali per due motivi: primo, perché l’uso operativo del modello avviene tramite soglie decisionali (selezionate con criteri come F1 pesata o l’indice di Youden); secondo, perché queste probabilità si integrano in modo coerente con la Knowledge Base, che beneficia di stime credibili quando deve combinare regole e segnali di rischio.

Se confrontiamo Random Forest con alternative comuni, emergono ragioni pragmatiche. Una regressione logistica, pur elegante e interpretabile, presuppone un confine decisionale lineare a meno di un forte lavoro di feature engineering; spesso non basta per cogliere le interazioni che osserviamo nei dati vocali. Le SVM possono modellare frontiere complesse, ma richiedono un tuning più fine e le probabilità vanno comunque calibrate. I metodi boosting sono spesso allo stato dell’arte sui tabellari, ma sono più sensibili al tuning e al rischio di overfitting in set piccoli/medi; inoltre aggiungono complessità operativa senza garantire un vantaggio sostanziale nel nostro regime di dati. Le reti neurali, infine, tendono a dare il meglio con campioni più ampi e con una fase di sviluppo più onerosa; nel nostro caso l’aumento di complessità non è giustificato.

Dopo aver effettuato il train, vengono generati due grafici:



Il **Reliability diagram (test)**, è il grafico di calibrazione delle probabilità.

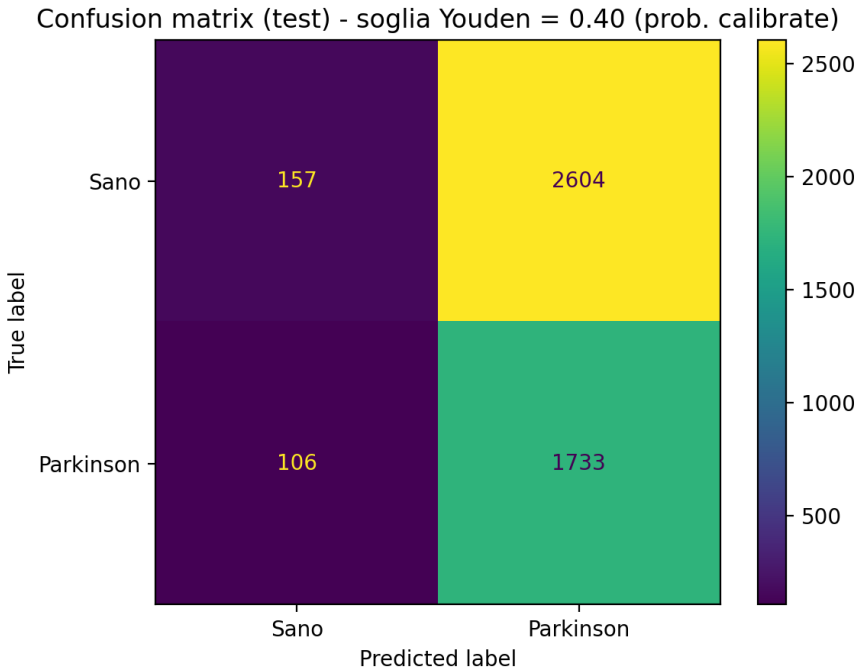
- Asse X = probabilità media predetta in ciascun “bin” (es. 0–0.1, 0.1–0.2, ...).
- Asse Y = frequenza osservata di positivi dentro quel bin.
- Linea tratteggiata (diagonale) = calibrazione perfetta: se il modello dice 0.30, allora circa il 30% dei casi in quel bin è davvero positivo.

Scelta della soglia: Youden vs F1-weighted

Quando trasformiamo una **probabilità** in un'**etichetta** (Parkinson/Sano) dobbiamo fissare una **soglia decisionale**. Nel progetto adottiamo due criteri automatici per scegliere tale soglia sul set di validazione:

Indice di Youden (J)

- **Definizione**: J=Sensibilità+Specificita`−1J = \text{Sensibility} + \text{Specificità} - 1J=Sensibilita`+Specificita`−1.
- **Obiettivo**: individuare il punto della ROC che massimizza **contemporaneamente** la capacità di trovare i positivi (sensibilità) e quella di escludere i negativi (specificità).
- **Caratteristica**: è **indipendente dalla prevalenza** (quanti malati ci sono nel dataset) e molto usato in ambito **clinico/diagnostico**.

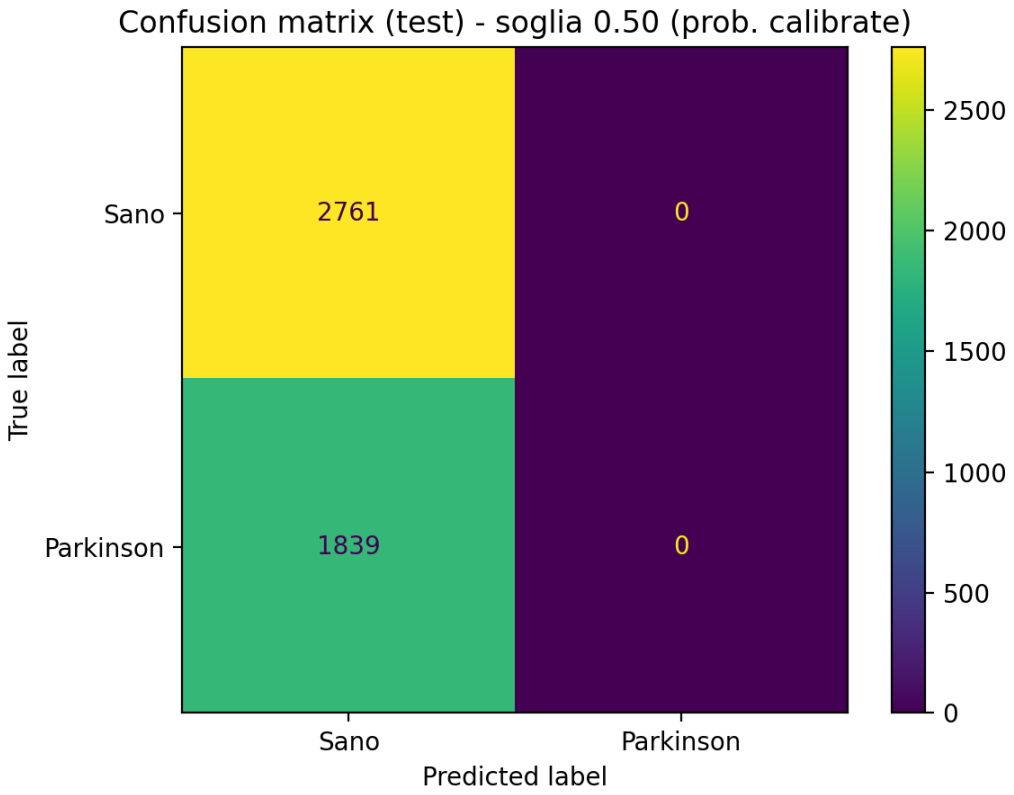


F1-weighted

- **Definizione**: media armonica tra **precision** e **recall**, pesata per la frequenza delle classi.
- **Obiettivo**: massimizzare un **compromesso globale** tra quante volte i positivi previsti sono corretti (precision) e quanti positivi reali vengono trovati (recall).
- **Caratteristica**: tiene conto dello **sbilanciamento** delle classi perché pesa le classi in base alla loro numerosità.

Differenza concettuale

- **Youden** cerca la soglia che "**separa meglio**" sani e malati in senso diagnostico (massimo distacco dalla diagonale ROC).
- **F1-weighted** cerca la soglia che **bilancia precision e recall** secondo la distribuzione reale delle classi.
  In pratica, **F1 tende a essere più conservativa** (meno positivi previsti) quando i falsi positivi costano molto; **Youden può spostarsi verso soglie più basse** se così aumenta molto la sensibilità, anche al prezzo di più falsi positivi.



Precision e Recall.

Alla soglia **Youden** il modello privilegia la sensibilità della classe *Parkinson* e produce un numero maggiore di positivi previsti: la **recall** della classe positiva aumenta, a fronte di una **precision** più bassa. Alla soglia **F1-weighted** il comportamento è più conservativo: i positivi previsti diminuiscono, la **precision** migliora e la **recall** scende leggermente. Nella nostra valutazione su test, questo si traduce in un diverso bilanciamento tra **veri positivi** e **falsi positivi**: con **Youden** emergono più TP (e più FP), con **F1** meno TP ma una quota di FP più contenuta. Per l'uso in **screening** riportiamo come valore "di record" Youden (massima separazione diagnostica); per contesti in cui i falsi allarmi sono più costosi, riportiamo anche i numeri alla soglia F1.

	precision	recall	f1-score	support
Sano	0.60	1.00	0.75	2761
Parkinson	0.00	0.00	0.00	1839
accuracy			0.60	4600
macro avg	0.30	0.50	0.37	4600
weighted avg	0.36	0.60	0.45	4600

PREDIZIONE

La predizione prende i dati di un paziente, carica il modello calibrato prodotto in addestramento e restituisce sia la probabilità di appartenenza alla classe *Parkinson* sia l'etichetta finale. L'idea è semplice: quello che in training è stato appreso (pipeline di preprocessing, classificatore, calibrazione delle probabilità e soglie operative) viene riutilizzato in modo identico e ripetibile, così che le decisioni in produzione siano coerenti con quelle ottenute in validazione.

Il training salva un unico artefatto, di solito un file `parkinson_model.joblib`. Dentro non c'è solo il modello: viene memorizzata l'intera pipeline già calibrata, l'elenco ordinato delle feature che il modello si aspetta di ricevere, e le soglie decise su validation (la soglia che massimizza l'F1 pesata e la soglia di Youden). In fase di inference, la funzione di predizione ricostruisce una riga a partire dal JSON del paziente, applica le stesse trasformazioni del training (imputazione dei mancanti, standardizzazione, one-hot delle categoriche, eventuali feature derivate), riallinea le colonne nell'ordine giusto in base alla lista salvata e calcola la probabilità calibrata. A quel punto la probabilità viene confrontata con una soglia: è questo confronto a trasformare un numero tra 0 e 1 nell'etichetta binaria "Parkinson" oppure "Sano".

L'input previsto è un semplice JSON con i nomi delle variabili così come compaiono nel dataset originale (per esempio `Jitter (%)`, `Shimmer(dB)` e `HNR`). Se qualche campo manca, l'imputatore lo sostituisce con valori plausibili appresi in training; se una categoria non è stata mai vista, l'encoder la ignora senza far fallire la predizione. Per chi preferisce un esempio pronto, il comando `random_patient` genera un JSON coerente con il dataset e quindi immediatamente utilizzabile.

L'uso è possibile sia da riga di comando sia via API. Da terminale il comando `python main.py predict --json paziente.json` calcola la probabilità e, in base alla modalità scelta, decide la soglia da applicare. È possibile chiedere esplicitamente la soglia di Youden (`--thr youden`), quella che massimizza l'F1 pesata (`--thr f1`) oppure imporre una soglia numerica fissa (`--thr fixed --thr_value 0.40`). Il risultato riporta l'etichetta finale, la probabilità calibrata e il valore della soglia che è stata effettivamente usata, così da rendere trasparente la decisione.

La scelta della soglia merita qualche parola in più. La soglia di Youden è quella che massimizza, sulla curva ROC, la somma tra sensibilità e specificità meno uno; in pratica seleziona il punto che separa meglio sani e malati in senso diagnostico. È spesso preferita in scenari di screening perché tende ad aumentare la sensibilità, cioè a ridurre i falsi negativi, anche a costo di avere più falsi positivi. La soglia che massimizza l'F1 pesata, invece, nasce per bilanciare precisione e richiamo tenendo conto della distribuzione reale delle classi: quando i falsi allarmi hanno un costo, questa scelta tende a essere più conservativa, quindi a produrre meno positivi previsti rispetto a Youden. Nelle tue prove si vede bene la differenza: con una soglia vicina a Youden (per esempio 0,40) la matrice di confusione mostra molti casi classificati come positivi e una sensibilità elevata; la stessa valutazione con una soglia F1 tipicamente riduce il numero di positivi previsti, migliorando la precisione a scapito di un po' di richiamo. È esattamente il comportamento atteso: Youden privilegia la capacità di "trovare" i malati, F1 privilegia un equilibrio più stringente tra trovare i malati e non generare allarmi inutili.

Conclusioni

Il percorso descritto parte da un'EDA solida a un modello addestrato e **calibrato**, fino a una predizione ripetibile e tracciabile. La scelta di Random Forest, unita a una pipeline unica di preprocessing e alla calibrazione, offre un compromesso efficace tra prestazioni, robustezza e manutenibilità. Il risultato non è solo un classificatore, ma un **sistema decisionale** che produce probabilità affidabili e le traduce in etichette tramite soglie documentate.

Le evidenze emerse chiariscono il punto chiave: la **soglia operativa** determina il comportamento in produzione. Con 0,50 il modello non intercetta positivi; adottando la soglia di **Youden** (~0,40) la sensibilità sulla classe Parkinson cresce fino a circa **0,94**, con una precisione intorno a **0,40** perché aumentano i falsi allarmi. È il trade-off atteso: Youden massimizza la separazione diagnostica e privilegia il recupero dei casi, mentre la soglia **F1-weighted** tende a essere più conservativa, riducendo i positivi previsti e migliorando la precisione a scapito di un po' di recall. La soglia, quindi, non è un dettaglio tecnico ma una **leva clinica**: per scenari di screening conviene privilegiare Youden; dove i falsi positivi hanno un costo alto, è preferibile la soglia F1 o una soglia fissa concordata.

Operativamente, l'inferenza replica esattamente il training (stesse trasformazioni, stesso ordine delle feature, stessa calibrazione) e registra sempre probabilità, soglia e modalità usata. Per mantenere affidabilità nel tempo, conviene monitorare periodicamente **calibrazione** e **derivazione dei dati**, pianificando ri-calibrazione o ri-addestramento quando cambiano popolazione o processo di misura. Con più dati e nuove feature sarà possibile valutare alternative (es. boosting) senza rinunciare al principio non negoziabile della calibrazione. In questo modo la soluzione rimane **spiegabile, adattabile e orientata all'uso clinico**, allineando evidenza statistica, regole della Knowledge Base e giudizio esperto.