Documentazione della Pipeline Tabellare
`xai_tab`

June 27, 2025

1. Contesto

Per analizzare il *Disagreement Problem*, che consiste nel comprendere perché due modelli simili possano fornire spiegazioni differenti pur ottenendo risultati comparabili, è stato scelto il dataset Breast–Cancer Wisconsin. Questo dataset contiene informazioni mediche tabellari relative a 569 pazienti, descritte tramite 30 caratteristiche numeriche. Sono stati addestrati due modelli identici nella struttura (reti neurali MLP con gli stessi strati e neuroni) ma inizializzati con seed diversi, generando quindi minime variazioni nelle loro configurazioni iniziali. L'obiettivo finale è quantificare e comprendere quanto le attribuzioni delle feature, ovvero le spiegazioni di come ogni caratteristica influenzi le predizioni, divergano tra i due modelli.

2. Tecnologie e librerie

La pipeline è stata implementata in Python 3.10 utilizzando un ambiente virtuale `conda` chiamato `xai`. Le librerie scelte sono state selezionate per i seguenti motivi:

- **PyTorch**: per creare e addestrare reti neurali efficienti e flessibili.
- **scikit-learn**: per il preprocessing robusto e testato dei dati.
- **Captum**: per tecniche di Explainable AI (Integrated Gradients).
- **NumPy**: per calcoli numerici e operazioni su vettori e matrici.
- **Matplotlib**: per visualizzazioni intuitive e chiare dei risultati.

3. Flusso di lavoro

Il processo si articola in tre fasi: training dei modelli, generazione delle spiegazioni e calcolo delle metriche di disaccordo.

3.1 Training dei modelli (`train.py`)

Il processo di training inizia caricando il dataset Breast–Cancer tramite la funzione `load_breast_cancer()`, che restituisce una matrice con le caratteristiche dei pazienti e un vettore con le etichette delle classi (benigno o maligno). Per garantire che tutte le caratteristiche siano considerate in modo equo dal modello, i dati vengono standardizzati: ogni caratteristica avrà media pari a zero e deviazione standard pari a uno.

Successivamente, il dataset viene diviso in due parti: una di training (80%) e una di test (20%). La divisione viene fatta con stratificazione per assicurare che la proporzione tra le classi sia mantenuta uguale in entrambe le parti, così da avere risultati più affidabili.

I modelli utilizzati sono semplici reti neurali artificiali (MLP) con due strati nascosti, ciascuno di 16 neuroni. Questa configurazione bilancia la capacità del modello di apprendere caratteristiche complesse dei dati senza esagerare nella complessità, evitando così fenomeni di sovra-adattamento (overfitting).

Per osservare come piccoli cambiamenti possano influenzare le predizioni, vengono addestrati due modelli con la stessa struttura ma inizializzati con semi casuali diversi (seed 0 e seed 1). Infine, i modelli addestrati vengono salvati per essere riutilizzati in seguito (`models/mlp_seed0.pt` e `models/mlp_seed1.pt`).

3.2 Generazione delle spiegazioni (`compare_tabular.py`)

In questa fase, si parte dai dati di test precedentemente preparati, trasformandoli in una struttura specifica chiamata *tensore*, utilizzata da PyTorch per eseguire rapidamente calcoli complessi.

I due modelli addestrati vengono caricati e posti in modalità di valutazione (`eval()`), che disattiva comportamenti specifici del training come il dropout, rendendo le predizioni stabili e coerenti.

Per generare le spiegazioni, si utilizza la tecnica *Integrated Gradients*, che misura l'importanza delle caratteristiche confrontando la predizione del modello con un punto di riferimento detto *baseline*, scelto in questo caso come un vettore di zeri, rappresentante una situazione neutrale.

L'integrazione avviene in 50 passi (`n_steps = 50`), un compromesso ideale tra accuratezza della spiegazione e costo computazionale.

Le attribuzioni ottenute, ovvero le spiegazioni che indicano quanto ciascuna caratteristica influenzi la previsione finale, vengono trasformate in semplici vettori numerici NumPy per facilitare le analisi successive..

Passi di integrazione (`n_steps`) Integrated Gradients misura l'importanza delle caratteristiche considerando il cambiamento della predizione del modello da un punto di partenza neutro (baseline) fino al campione reale analizzato. Questo calcolo avviene tramite una approssimazione matematica chiamata *somma di Riemann*, che divide il percorso tra baseline e campione reale in un certo numero di intervalli più piccoli (chiamati *passi di integrazione*). Più passi si utilizzano, più precisa e stabile risulta la spiegazione, anche se aumenta il tempo necessario per il calcolo. In questa analisi, sono stati scelti 50 passi.

Calcolo delle attribuzioni Le spiegazioni vengono calcolate creando un oggetto `IntegratedGradients` per ciascun modello e invocando il metodo `attribute`. Questo metodo restituisce un tensore (struttura dati) contenente i valori numerici delle attribuzioni per ciascun campione e caratteristica. Questi tensori vengono poi convertiti in vettori NumPy, più semplici da utilizzare nelle analisi successive.

3.3 Calcolo delle metriche di disaccordo (`metrics.py`)

Questa fase si occupa di calcolare quanto le spiegazioni differiscono tra i due modelli. Le metriche implementate sono:

- **Feature Disagreement**: misura quante delle 8 caratteristiche più importanti differiscono tra i due modelli.
$$1 - \frac{|\mathrm{Top}_k(\vec{a}) \cap \mathrm{Top}_k(\vec{b})|}{k}.$$
- **Sign Disagreement**: oltre a valutare le caratteristiche selezionate, considera se il segno (positivo o negativo) associato all'importanza delle caratteristiche differisce tra i due modelli. Questo è importante per capire non solo *quali* caratteristiche sono influenti, ma anche *come* influenzano la predizione.

- **Euclidean**: misura la distanza complessiva tra le spiegazioni, considerando sia intensità che segno.

$$\| \frac{\vec{a}}{\|\vec{a}\|} - \frac{\vec{b}}{\|\vec{b}\|} \|_2.$$

- **Euclidean-abs**: misura la distanza considerando solo l'intensità, ignorando il segno.

Ogni metrica è calcolata riga-per-riga su tutti i campioni di test e quindi sintetizzata in media ± deviazione standard.

4. Risultati globali

Eseguendo:

```
python src/train.py --seeds 0 1
python src/compare_tabular.py
```

si ottengono:

$$\begin{aligned} \text{FeatureDisagreement} &= 0.294 \pm 0.126, \\ \text{SignDisagreement} &= 0.297 \pm 0.125, \\ \text{Euclidean} &= 0.432 \pm 0.127, \\ \text{Euclidean-abs} &= 0.374 \pm 0.089. \end{aligned}$$

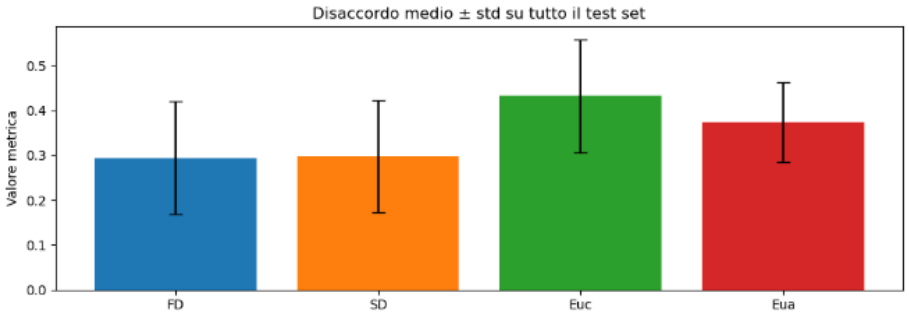

Figure 1: Bar-plot delle metriche di disaccordo medie ± std sul test set.

Il bar-plot sintetizza le quattro metriche di disaccordo su tutto il test set:

- **Feature Disagreement (FD)**: con media ≈ 0.29 e deviazione standard ≈ 0.13, indica che in media il 71% delle top-8 feature è condiviso tra i due modelli, con alcuni casi in cui il disaccordo supera il 40%.
- **Sign Disagreement (SD)**: media ≈ 0.30, segnala che non solo cambiano le feature selezionate, ma in quasi un terzo dei casi si inverte anche il segno dell'attribuzione, ossia il "verso" dell'influenza.
- **Euclidean**: distanza L_2 media ≈ 0.43, con std ≈ 0.13, mostra una divergenza complessiva moderata nei vettori normalizzati di attributi.
- **Euclidean-abs**: media ≈ 0.37, dimostra che buona parte della differenza è dovuta all'intensità delle importanze, ma che il segno amplifica ulteriormente il disaccordo complessivo.

Questa sintesi conferma che, sebbene i due modelli abbiano prestazioni quasi identiche sul test set, le spiegazioni possono variare in modo consistente.

5. Case study su campione 0

Nella seconda parte della figura **??**, vengono mostrate le explanation per il campione 0. I valori delle metriche per questo singolo esempio sono:

$$\mathrm{FD}_0 = 0.25, \quad \mathrm{SD}_0 = 0.25, \quad \mathrm{Euc}_0 = 0.39, \quad \mathrm{Eua}_0 = 0.39.$$

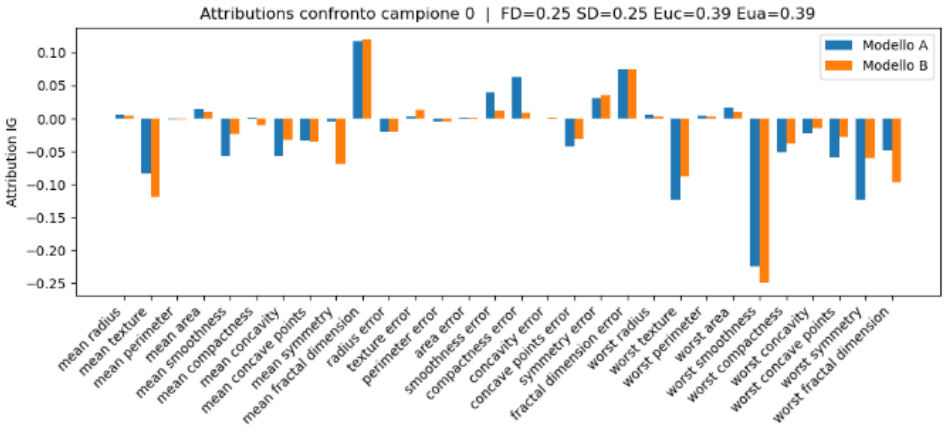

Figure 2: Confronto delle attribuzioni IG per il campione 0.

Cosa mostrano le barre

Ogni coppia di barre (blu + arancio) corrisponde a una feature del dataset Breast–Cancer (sull'asse orizzontale). L'altezza di ciascuna barra è il valore di attribuzione:

- positivo → feature che "spinge" la rete verso la classe positiva (maligno);
- negativo → feature che "spinge" verso la classe negativa (benigno).

Confrontando Modello A vs. Modello B si evidenzia dove e quanto i due modelli differiscono nell'attribuire importanza.

1 Glossario

- **Disagreement Problem**: Fenomeno per cui modelli simili possono produrre spiegazioni differenti nonostante risultati simili.
- **MLP (Multilayer Perceptron)**: Rete neurale artificiale costituita da più strati di neuroni.
- **Seed**: Valore iniziale utilizzato per inizializzare casualmente i parametri del modello.
- **Baseline**: Punto di riferimento neutrale utilizzato nella tecnica Integrated Gradients.
- **Tensore**: Struttura dati multidimensionale utilizzata per calcoli efficienti con PyTorch.
- **Integrated Gradients**: Metodo di Explainable AI che misura l'importanza delle feature rispetto a un punto di riferimento (baseline).
- **Standardizzazione**: Processo di trasformazione dei dati per avere media 0 e deviazione standard 1.

- **Stratificazione**: Divisione dei dati mantenendo le proporzioni originali delle classi.
- **Feature Disagreement**: Metrica che misura le differenze nelle caratteristiche selezionate come più importanti.
- **Sign Disagreement**: Metrica che valuta le differenze nel segno attribuito all'importanza delle caratteristiche.
- **Euclidean e Euclidean-abs**: Metriche basate sulla distanza matematica tra vettori di attribuzione.