Heart Disease

Introduzione

Il dataset **Heart Disease** disponibile presso l'**UCI** Machine Learning Repository riguarda la diagnosi di malattie cardiache e contiene dati raccolti da quattro diverse istituzioni:

1. Cleveland Clinic Foundation - `src/hd/data/notProcessedDataset/cleveland.data`
2. Hungarian Institute of Cardiology, Budapest - `src/hd/data/notProcessedDataset/hungarian.data`
3. V.A. Medical Center, Long Beach, CA - `src/hd/data/notProcessedDataset/va.data`
4. University Hospital, Zurich, Switzerland - `src/hd/data/notProcessedDataset/switzerland.data`

Ogni database ha lo stesso formato per quanto riguarda le istanze. Sebbene i database contengano 76 attributi grezzi, solo 14 di essi sono effettivamente utilizzati in esperimenti passati. Per questo motivo, sono stati creati due copie di ogni database: una contenente tutti gli attributi e un'altra con solo i 14 attributi utilizzati.

Anche nel nostro caso abbiamo sfruttato le copie con 14 attributi di ciascun dataset, combinandole in un unico dataset che è stato utilizzato per i nostri esperimenti.

Descrizione delle feature del dataset

Feature di input

Questi sono gli attributi che rappresentano le variabili indipendenti. Sono usati come input per i modelli che sfrutteremo. Nel dataset delle malattie cardiache, le feature di input includono:

- **Age (età)**: L'età del paziente in anni - Variabile numerica continua.
- **Sex (sesso)**: Sesso del paziente (1 = maschio, 0 = femmina) - Variabile categorica binaria.
- **CP (tipologia di dolore toracico)**:
    - Valore 1: Angina tipica
    - Valore 2: Angina atipica
    - Valore 3: Dolore non-anginoso

    - Valore 4: Asintomatico

  Si tratta di una variabile categorica nominale.

- **Trestbps (pressione arteriosa a riposo)**: Pressione arteriosa a riposo in mm Hg all'ammissione in ospedale - Variabile numerica continua
- **Chol (colesterolo)**: Livello di colesterolo sierico in mg/dl - Variabile numerica continua.
- **FBS (glicemia a digiuno)**: Glicemia a digiuno > 120 mg/dl (1 = vero; 0 = falso) - Variabile categorica binaria.
- **Restecg (risultati dell'elettrocardiogramma a riposo)**:
    - Valore 0: Normale
    - Valore 1: Anomalia delle onde ST-T (inversione dell'onda T e/o sopraslivellamento o sottoslivellamento > 0.05 mV)
    - Valore 2: Ipertrofia ventricolare sinistra probabile o definitiva

  Si tratta di una variabile categorica nominale.

- **Thalach (frequenza cardiaca massima raggiunta)**: Frequenza cardiaca massima raggiunta dal paziente - Variabile numerica continua.
- **Exang (angina indotta dall'esercizio)**: Angina indotta dall'esercizio (1 = sì; 0 = no) - Variabile categorica binaria
- **Oldpeak (depressione del tratto ST)**: Depressione del tratto ST indotta dall'esercizio rispetto al riposo - Variabile numerica continua.
- **Slope (pendenza del tratto ST durante l'esercizio massimo)**:
    - Valore 1: Pendenza ascendente
    - Valore 2: Pendenza piatta
    - Valore 3: Pendenza discendente

  Si tratta di una variabile categorica ordinale.

- **CA (numero di vasi principali colorati)**: Numero di vasi principali (da 0 a 3) colorati dalla fluoroscopia - Variabile categorica discreta
- **Thal (talassemia)**:

- Valore 3: Normale
- Valore 6: Difetto fisso
- Valore 7: Difetto reversibile

Si tratta di una variabile categorica nominale.

Feature di output

Questo è l'attributo che il modello tenta di prevedere. Nel dataset, la feature di output è **Num** che appresenta la presenza o assenza di malattia cardiaca. Si tratta di una variabile categorica che può assumere valori da 0 a 4, dove 0 indica l'assenza di malattia e i valori da 1 a 4 indicano la presenza e la gravità della malattia.

Supponiamo di voler utilizzare i nostri modelli in un contesto clinico dove è fondamentale decidere rapidamente se un paziente necessita di ulteriori esami o trattamenti, in questo caso ci basta sapere solamente se il paziente sia sano o malato, piuttosto che quantificare lo stato della malattia se presente. Pertanto ci occuperemo di un task di **classificazione binaria**, dove la variabile **Num** è pari a 0, se assente la malattia cardiaca, altrimenti è pari a 1 (tutti i valori strettamente maggiori di 1 nel dataset di partenza saranno riassegnati a 1).

Preprocess del Dataset

Riferimento: `src/hd/heartDisease.py`

Nel processo di preparazione del dataset, abbiamo seguito una serie di fasi per garantire che i dati fossero combinati e utilizzati correttamente. L'obiettivo era quello di ottenere un dataset pronto per essere sfruttato dai modelli, che abbiamo utilizzato, e che i dati fossero coerenti e privi di errori, che potessero compromettere la validità delle analisi successive.

Abbiamo caricato i dati provenienti dalle quattro fonti sopracitate in singoli DataFrame, che sono stati uniti in uno solo, e, successivamente, abbiamo sviluppato due pipeline di preprocessing per gestire le feature numeriche e categoriche separatamente.

Per le feature numeriche, abbiamo deciso di sostituire i valori mancanti con la mediana perché non è influenzata dai valori anomali. Sapendo che la media è influenzata dagli outlier, aver preferito l'uso della mediana ci garantisce che i valori mancanti vengano sostituiti con un valore che rappresenta il centro della distribuzione dei dati, senza considerare le distorsioni dei valori estremi. Dopo l'imputazione queste feature sono state standardizzate.

Per quanto riguarda le feature categoriche, la scelta presa per la sostituzione dei valori mancanti è stata quella che sembrava più logica, ossia la moda, che rappresenta la categoria

più frequente per ciascuna feature. In seguito, le feature categoriche sono state codificate utilizzando la tecnica **One-Hot Encoding**, che trasforma ogni categoria in una variabile binaria indipendente, in maniera tale da evitare che un modello interpreti in maniera errata le relazioni tra categorie, dato che le variabili categoriche non hanno un ordine intrinseco.

Le due pipeline di preprocessing sono state quindi combinate usando `ColumnTransformer`, che ha applicato automaticamente le trasformazioni appropriate a ciascun gruppo di feature. A questo punto, il dataset era pronto per la suddivisione in set di addestramento e di test. Abbiamo suddiviso i dati in un set di addestramento che comprende l'80% del totale e un set di test con il restante 20%.

Dato che il dataset originale presentava un certo disequilibrio tra le classi di circa il 10% e con una predominanza della classe di *assenza di malattia*, abbiamo utilizzato la tecnica **SMOTE** (Synthetic Minority Over-sampling Technique) per bilanciare le classi all'interno del set di addestramento. SMOTE ha generato esempi sintetici della classe minoritaria, garantendo una distribuzione più equilibrata delle classi e migliorando così le performance dei modelli.

Alla fine di questo processo abbiamo ottenuto il dataset preprocessato, bilanciato e pronto per l'addestramento dei modelli predittivi.

Modelli

Riferimenti:

- `src/hd/model.py`
- `src/hd/ann.py`

Per affrontare il problema di classificazione binaria abbiamo utilizzato una serie di modelli: **Decision Tree**, **Logistic Regression**, **Random Forest**, **Gradient Boosting** e un'**ANN**.

Pipeline

La pipeline implementata per l'ottimizzazione, l'addestramento e la valutazione dei modelli è stata progettata per garantire di ottimizzare le prestazioni e fornire una valutazione di ciascun modello.

Il punto di partenza è l'ottimizzazione degli iperparametri, realizzato con `Optuna`, che utilizza un approccio di **ottimizzazione bayesiana**, per suggerire le combinazioni di parametri, valutando ciascuna combinazione attraverso la **cross validation**.

Abbiamo già affrontato l'ottimizzazione con Optuna nella sezione riguardante il problema `NSP`: l'unica cosa che cambia nel procedimento è appunto la funzione obiettivo da massimizzare, che viene calcolata attraverso la media dei punteggi di accuracy ottenuti con la cross validation.

Una volta identificati i migliori iperparametri, procediamo alla creazione e all'addestramento del modello ottimizzato. Il modello viene addestrato sul set di training completo utilizzando i parametri trovati con il processo di ottimizzazione. Successivamente, valutiamo le prestazioni del modello sul set di test. Questa valutazione include il calcolo di diverse metriche: accuracy, precision, recall e F1-score. È stata tracciata una curva di apprendimento per ciascun modello come ulteriore strumento di analisi. La curva di apprendimento mostra come la performance del modello variava al crescere della dimensione del set di addestramento. Inoltre vengono esaminati i punteggi di training e test, calcolando metriche come il gap tra le prestazioni di training e test, la varianza dei punteggi e la stabilità del modello.

Spostiamo la nostra attenzione su ciascun modello fornendone una descrizione e cercando di interpretarne i risultati ottenuti.

Decision Tree

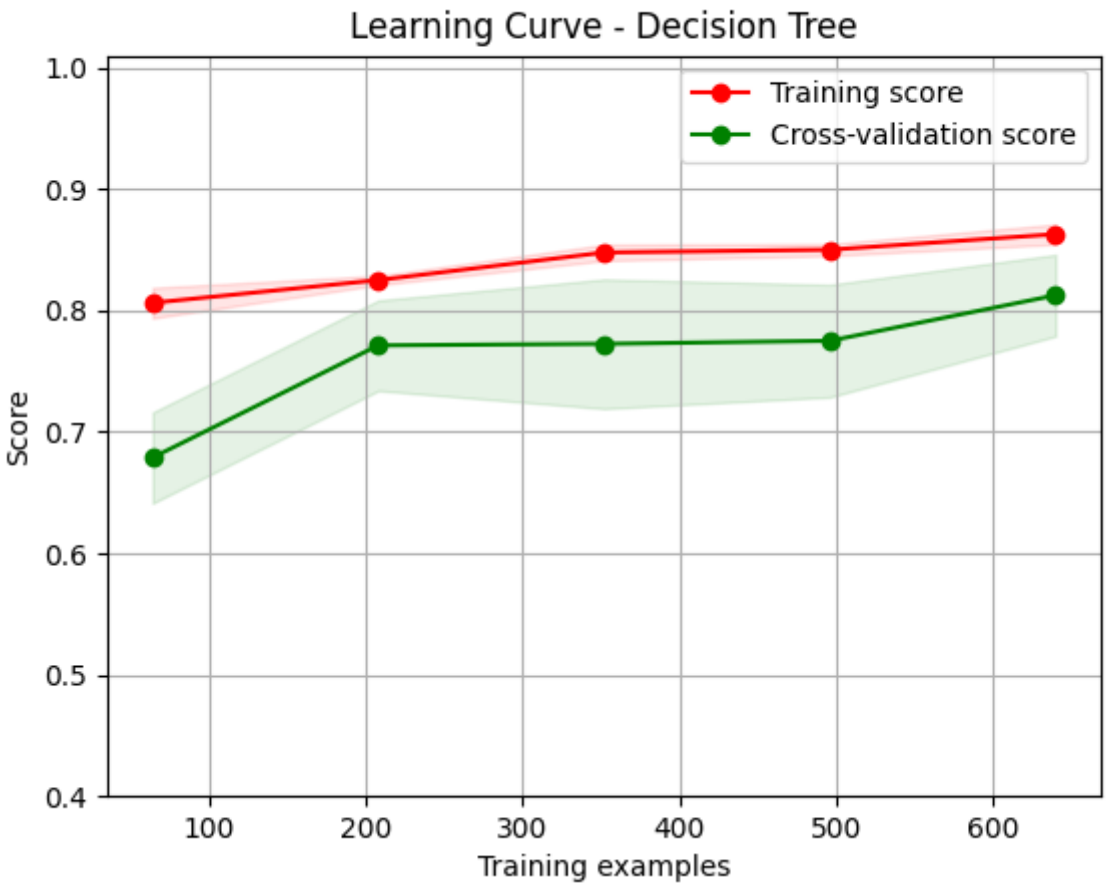
Il modello `DecisionTreeClassifier` è un algoritmo basato su alberi decisionali, che segmenta iterativamente il dataset in sottoinsiemi basati sui valori delle feature. Questo processo continua fino a quando le foglie dell'albero contengono dati puri o fino a raggiungere una profondità massima predefinita.

Nel nostro contesto possiamo immaginare un albero dove ogni nodo rappresenta una domanda su una caratteristica del paziente come: "la pressione è superiore a 120?". A seconda della risposta, ci si sposta su un ramo diverso dell'albero fino ad arrivare ad una foglia che rappresenta la classificazione finale.

I parametri chiave ottimizzati sono:

- **`max_depth=22`**: La profondità massima dell'albero impedisce che l'albero cresca troppo in profondità. Un valore troppo basso può causare underfitting, mentre un valore troppo alto può portare a overfitting.
- **`min_samples_split=20`**: Il numero minimo di campioni richiesti per dividere un nodo, così da evitare divisioni basate su troppi pochi dati. Aumentando questo valore, l'albero diventa più conservativo, riducendo il rischio di overfitting.
- **`min_samples_leaf=7`**: Il numero minimo di campioni che un nodo foglia deve contenere, per evitare previsioni basate su casi troppo specifici.

Curve di apprendimento



- **Training score**

  La curva rossa rappresenta il punteggio di addestramento e si mantiene costantemente alta, oscillando intorno a un valore di circa 0.81-0.86. Questa curva mostra un leggero ma costante miglioramento all'aumentare dei dati di training, indicando che il modello continua ad apprendere.

- **Cross-validation score**

  La curva verde rappresenta il punteggio di cross-validation e inizia a un valore relativamente più basso, intorno a 0.70, per poi stabilizzarsi e in seguito raggiungere 0.81 man mano che aumenta la dimensione del set di addestramento. Questo comportamento può suggerirci che il modello raggiunge presto un buon livello di generalizzazione.

Sebbene le curve di training e di validation non si sovrappongano completamente, il divario tra loro si riduce leggermente con l'aumentare dei dati di addestramento. Quindi con un numero maggiore di dati, il modello potrebbe ulteriormente migliorare la sua capacità di generalizzazione.

Inoltre, il gap tra la media dei punteggi di addestramento e di test è di circa 0.05, il che può segnalare una leggera tendenza all'overfitting. La varianza sia per il train score che per il test score è bassa, indicando che il modello è stabile e non presenta grande volatilità nei risultati.

Sebbene ci sia un leggero divario tra le curve di addestramento e test, non è sufficientemente ampio da indicare un forte overfitting. Il cross-validation score finale, seppur leggermente inferiore al punteggio di addestramento, suggerisce che il modello ha una buona capacità di generalizzare ai dati non visti.

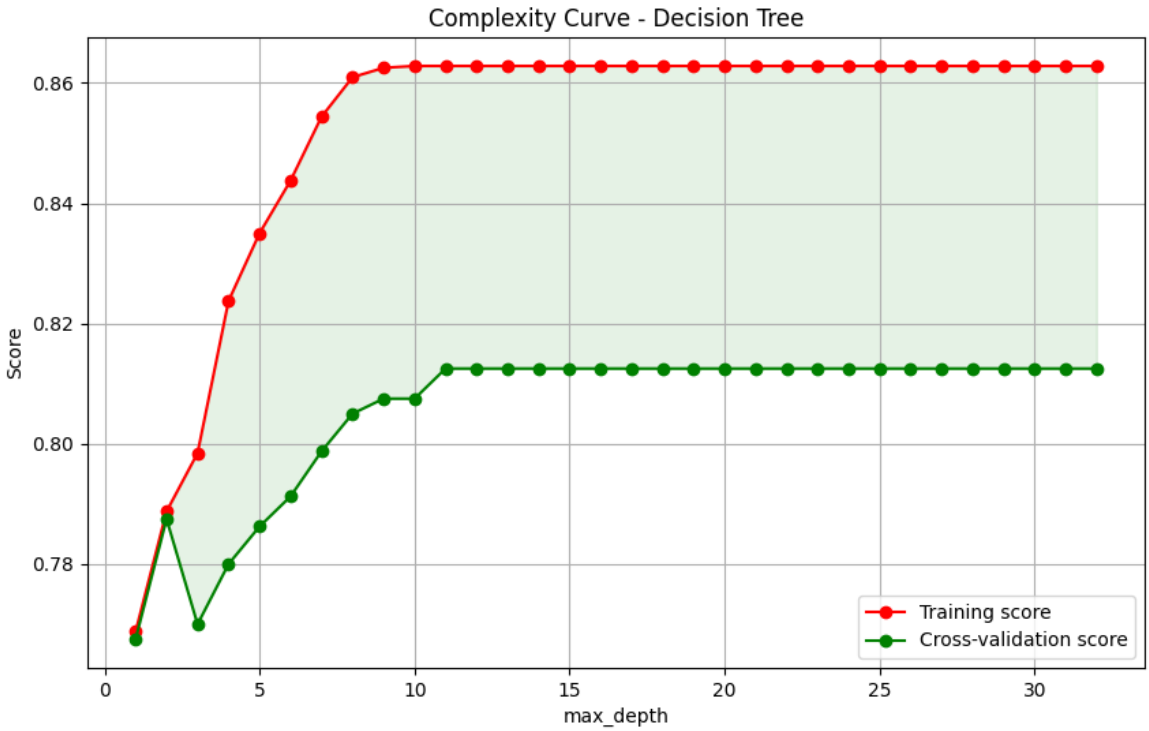
Risultati ottenuti con i parametri ottimizzati

	Media	Std	Var
Train	0.862813	0.008232	0.000068
Test	0.812500	0.033541	0.001125

best_accuracy	0.8125
final_accuracy	0.7663
precision	0.7753
recall	0.7663
f1_score	0.7680

Il leggero overfitting già evidente dalle curve di apprendimento viene così confermato anche dalla differenza tra **best_accuracy** (calcolata durante la fase di tuning) e **final_accuracy**.

Consideriamo il parametro `max_depth` come una misura di complessità, la cui variazione ci fornisce informazioni significative sul rischio di overfitting del modello.



Questo grafico mostra che aumentare eccessivamente `max_depth` porta a overfitting. Il modello diventa troppo complesso, adattandosi perfettamente ai dati di training ma perdendo capacità di generalizzazione. La profondità ottimale sembra essere intorno a 8-10, dove il cross-validation score raggiunge un suo massimo prima di stabilizzarsi, offrendo il miglior compromesso tra capacità predittiva e generalizzazione.

Impostando questo parametro in un range tra 9 e 11 abbiamo ottenuto gli stessi risultati che si hanno con `max_depth=22` dopo il tuning, quindi, vedendola in maniera positiva, abbiamo ridotto la complessità del modello per ottenere praticamente gli stessi risultati.

Invece impostando `max_depth=8`, otteniamo i seguenti risultati:

Risultati ottenuti con `max_depth=8`

	Media	Std	Var
Train	0.860938	0.007461	0.000056
Test	0.805000	0.032692	0.001069

accuracy	0.7608
precision	0.7711
recall	0.7608
f1_score	0.7627

Innanzitutto possiamo notare che il modello con profondità massima pari a 22 presenta prestazioni leggermente migliori in tutte le metriche rispetto al modello con profondità 8, tuttavia ha una varianza leggermente più alta sia per il train che per il test set, il che suggerisce una maggiore sensibilità ai dati di input.

Il modello "ottimizzato" offre un leggero miglioramento delle prestazioni a scapito di una maggiore complessità e di un potenziale aumento del rischio di overfitting.

Se vogliamo semplificare il modello mantenendo prestazioni comparabili, potremmo preferire il modello con `max_depth=8`. Se invece cerchiamo le migliori prestazioni possibili senza curarci della complessità, potremmo scegliere il primo modello, anche se non è tanto raccomandabile, essendo leggermente più esposto al rischio di overfitting.

Logistic Regression

Il modello di `LogisticRegression` è un algoritmo lineare utilizzato per la classificazione binaria. È particolarmente utile quando le relazioni tra le feature e la variabile target sono lineari. La regressione logistica calcola la probabilità che un campione appartenga a una determinata classe utilizzando una funzione sigmoide.

In altre parole, il modello nel nostro contesto calcola la probabilità che un paziente appartenga ad una determinata classe, cioè se è a rischio di malattia cardiaca o meno, basandosi su una combinazione lineare delle caratteristiche del paziente.

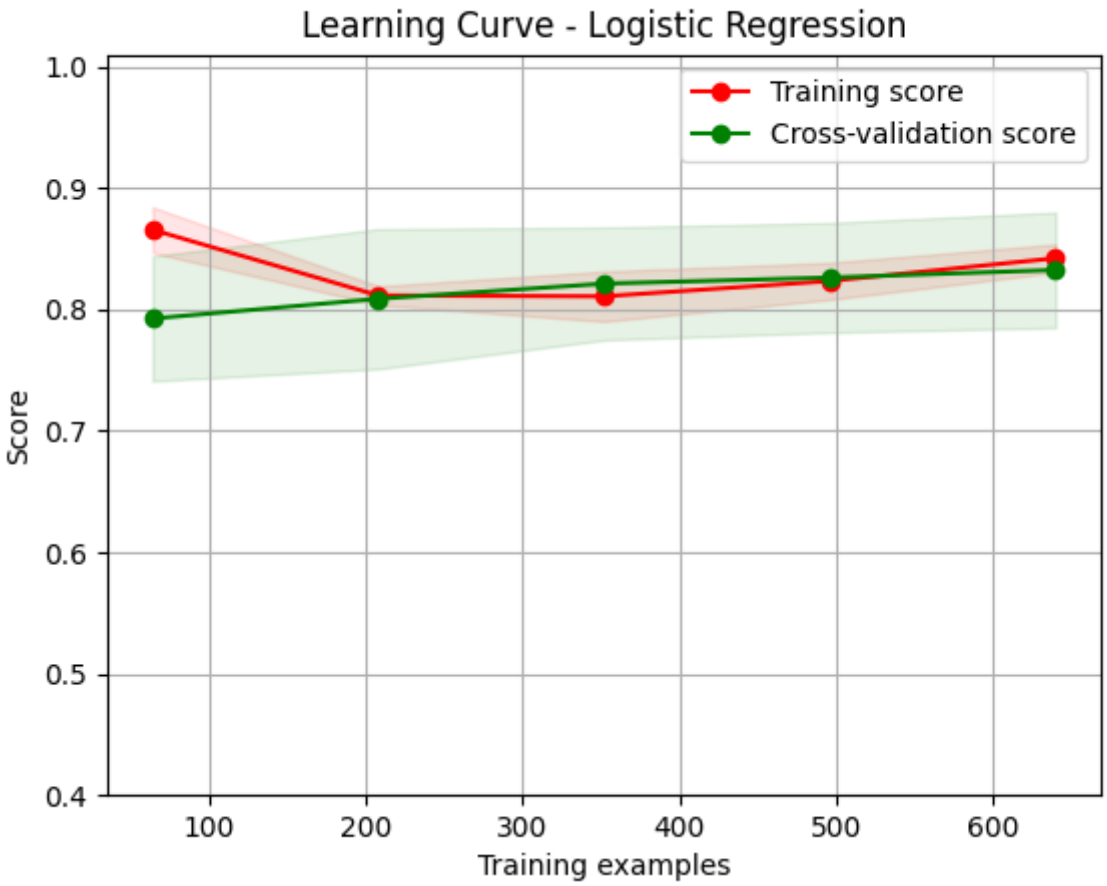
I parametri chiave ottimizzati sono:

- **`C=0.1536351925055615`**: Si tratta di un parametro di regolarizzazione. Un valore alto di `C` riduce la regolarizzazione, aumentando la complessità del modello, mentre un valore basso di `C` aumenta la regolarizzazione, prevenendo l'overfitting e riducendo la complessità del modello.
- **`penalty=l2`**: Il tipo di penalizzazione da applicare, che può essere `l1` (lasso) o `l2` (ridge). L1 favorisce la selezione delle feature più rilevanti eliminando quelle irrilevanti,

mentre L2 riduce il valore assoluto di tutti i coefficienti, che determinano quanto ciascuna feature contribuisce alla previsione del modello, senza eliminarli completamente.

- `max_iter=788` : Questo parametro definisce il numero massimo di iterazioni che l'algoritmo può eseguire durante l'ottimizzazione, ossia nel processo di aggiornamento dei coefficienti.

Curve di apprendimento



- **Training score**

  La curva di training inizia con un punteggio alto, che poi diminuisce leggermente per poi aumentare intorno a un valore di circa 0.84-0.85.

- **Cross-validation score**

  La curva di cross-validation parte da un punteggio più basso, sale e si avvicina a quella del training, stabilizzandosi anch'essa intorno a 0.82-0.83.

Le curve di apprendimento per il training set e il cross-validation set convergono a valori simili con l'aumentare dei dati di training. Questo è un buon segno, indicando che il modello non sta overfittando o underfittando in modo significativo.

Il piccolo gap (circa 0.01) tra la media degli score di training e di test conferma che il modello non soffre di overfitting.

La varianza del train score è molto bassa, indicando consistenza nelle prestazioni di training, mentre quella del test score è più alta, ma ancora relativamente bassa, seggerendo una buona generalizzazione con una leggera variabilità.

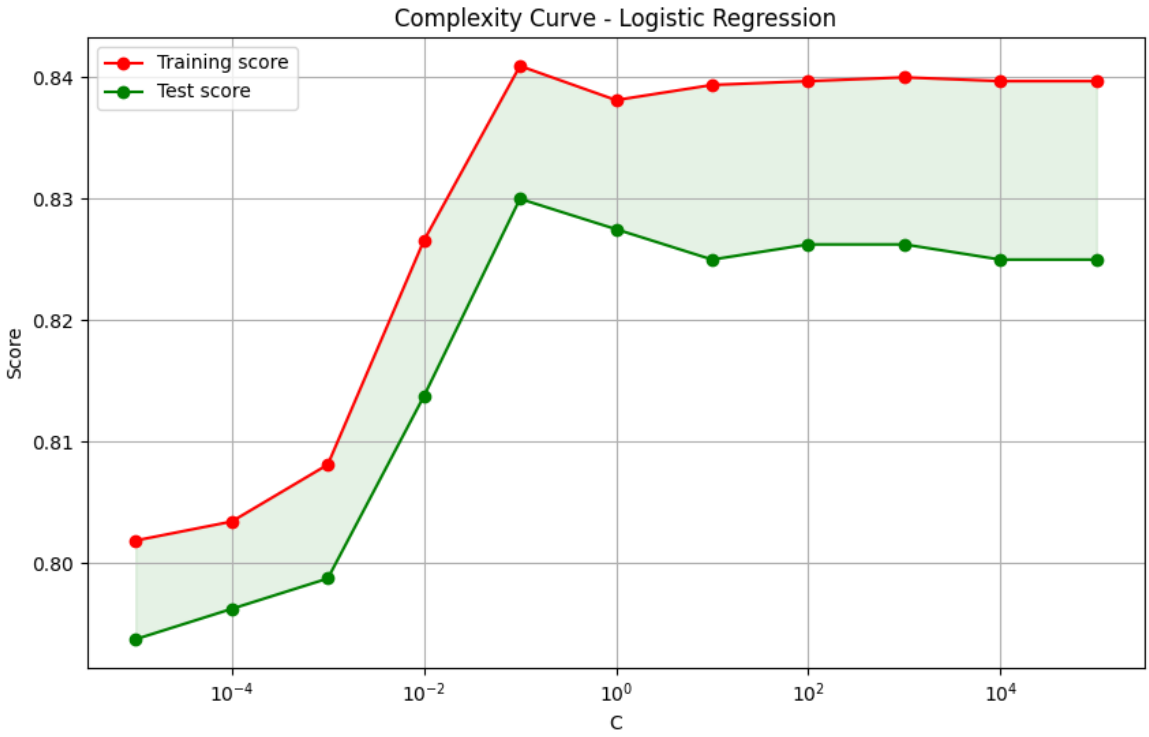
Risultati ottenuti con i parametri ottimizzati

	Media	Std	Var
Train	0.842187	0.011567	0.000134
Test	0.832500	0.047500	0.002256

best_accuracy	0.8324
final_accuracy	0.7880
precision	0.7946
recall	0.7880
f1_score	0.7894

La final accuracy è leggermente più bassa rispetto alla best accuracy, ma in questo caso non è allarmante. Le altre metriche sono tutte vicine tra loro, indicando un modello ben bilanciato.

Per avere ulteriori conferme sulla nostra analisi, consideriamo come misura di complessità il parametro `C`:



All'aumentare di C sia il training score che il test score tendono ad aumentare inizialmente, per poi stabilizzarsi. Il punto di massima performance per il test score sembra essere intorno a `C=0.1`. Questo suggerisce che una leggera regolarizzazione è benefica per le prestazioni del modello sui dati di test. Quindi il valore ottenuto dall'ottimizzazione `C=0.1536351925055615` risulta coerente con tutte le considerazioni fatte in precedenza.

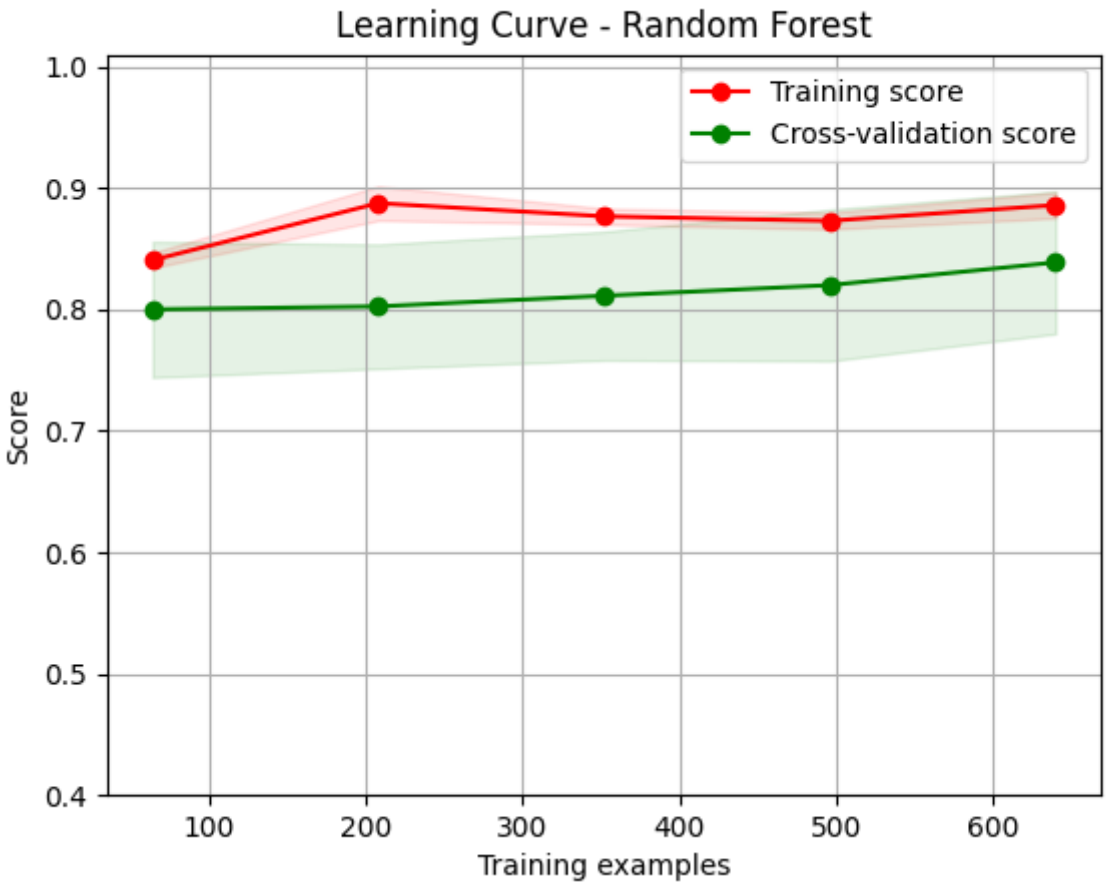
Random Forest

Il `RandomForestClassifier` è un **ensemble** di alberi decisionali, dove ciascun albero è addestrato su un sottoinsieme casuale di dati e feature. I risultati di tutti gli alberi vengono combinati per produrre la previsione finale. Questo approccio riduce la varianza rispetto a un singolo albero, migliorando la generalizzazione.

I parametri ottimizzati includono:

- **`n_estimators=93`**: Il numero di alberi nella foresta. Un numero maggiore di alberi generalmente migliora la performance, ma aumenta il costo computazionale.
- **`max_depth=9`**, **`min_samples_split=16`**, e **`min_samples_leaf=3`**: Gli stessi parametri dell'albero decisionale, applicati a ciascun albero della foresta.

Curve di apprendimento



- **Training score**

  Il modello dimostra ottime prestazioni, con lo score di training che raggiunge circa 0.89. La curva di training sale rapidamente all'inizio, raggiunge un picco intorno ai 200 esempi di training, e poi si stabilizza con un leggero aumento costante.

- **Cross-validation score**

  La curva verde del punteggio di validazione incrociata inizia più bassa, intorno a 0.80, ma cresce gradualmente fino a circa 0.83-0.84 man mano che vengono utilizzati più dati. Questa curva mostra un leggero miglioramento graduale e costante all'aumentare dei dati di training, indicando una buona capacità di generalizzazione.

Sebbene le curve non si sovrappongano, il divario tra il punteggio di addestramento e quello di validazione incrociata è moderato. Questo suggerisce un lieve overfitting, ma questa differenza non è eccessivamente preoccupante, specialmente considerando lo score finale di cross-validation.

Il gap tra i punteggi medi di addestramento e di test è di circa 0.05, indicativo di un modello che si adatta bene ai dati di addestramento ma che potrebbe non generalizzare perfettamente.

Tuttavia, la varianza del punteggio di test è relativamente più alta rispetto al train, il che indica una certa instabilità nelle performance sui dati non visti.

Risultati ottenuti con i parametri ottimizzati

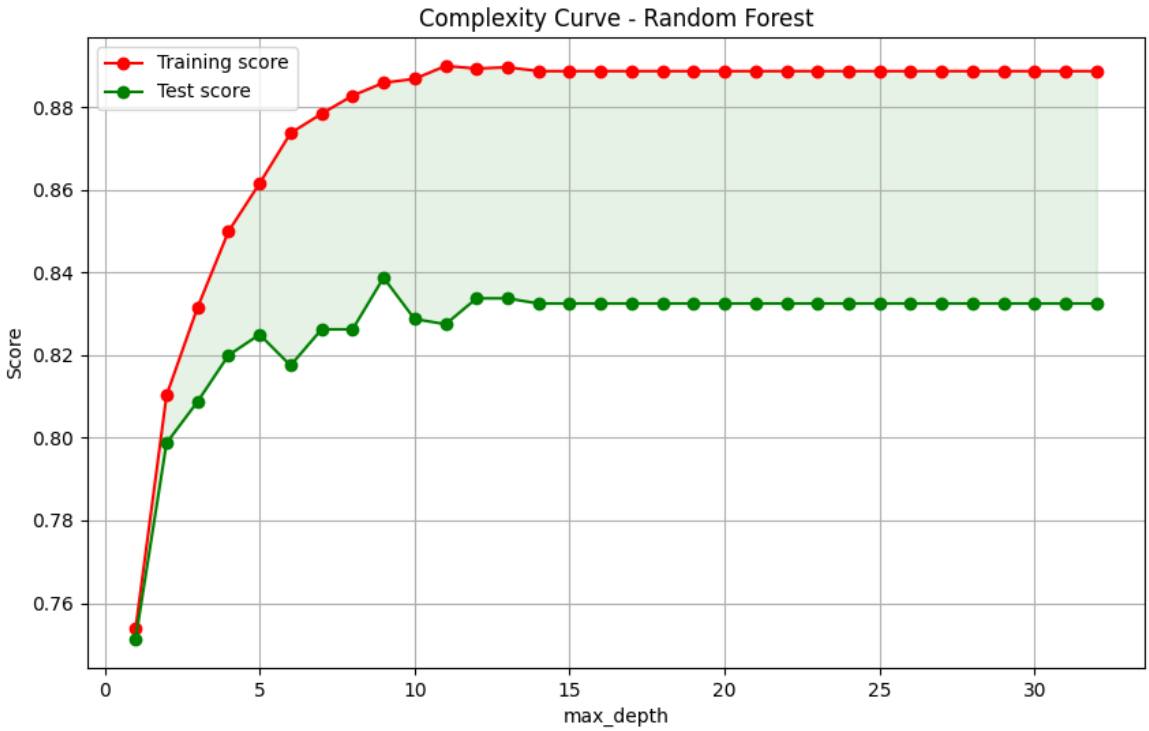
	Media	Std	Var
Train	0.885938	0.010597	0.000112
Test	0.838750	0.058683	0.003444

best_accuracy	0.8387
final_accuracy	0.8478
precision	0.8497
recall	0.8478
f1_score	0.8483

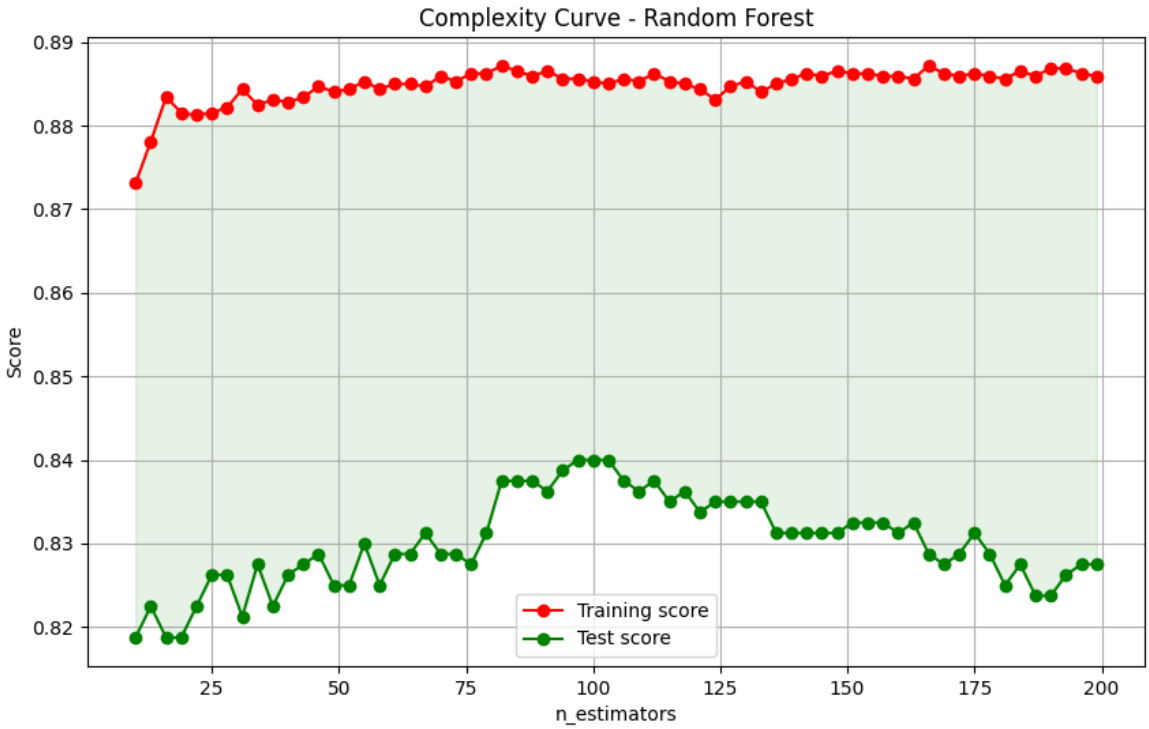
L'accuratezza finale è leggermente superiore alla best accuracy, indicando una buona generalizzazione sul set di test.

Le metriche di precision, recall e F1-score sono molto bilanciate, mostrando un'ottima performance complessiva.

Per analizzare meglio il rischio di overfitting nel modello consideriamo due misure di complessità: `n_estimators` e `max_depth`.



Il grafico mostra che il modello rischia di overfittare quando il parametro `max_depth` supera 9. Quindi mantenere `max_depth=9` come ottenuto dalla fase di ottimizzazione si è rivelata una scelta coerente.



Il grafico suggerisce che, oltre un certo numero di alberi (intorno a 100), il beneficio di aggiungere ulteriori alberi diminuisce, e il modello può iniziare a overfittare.

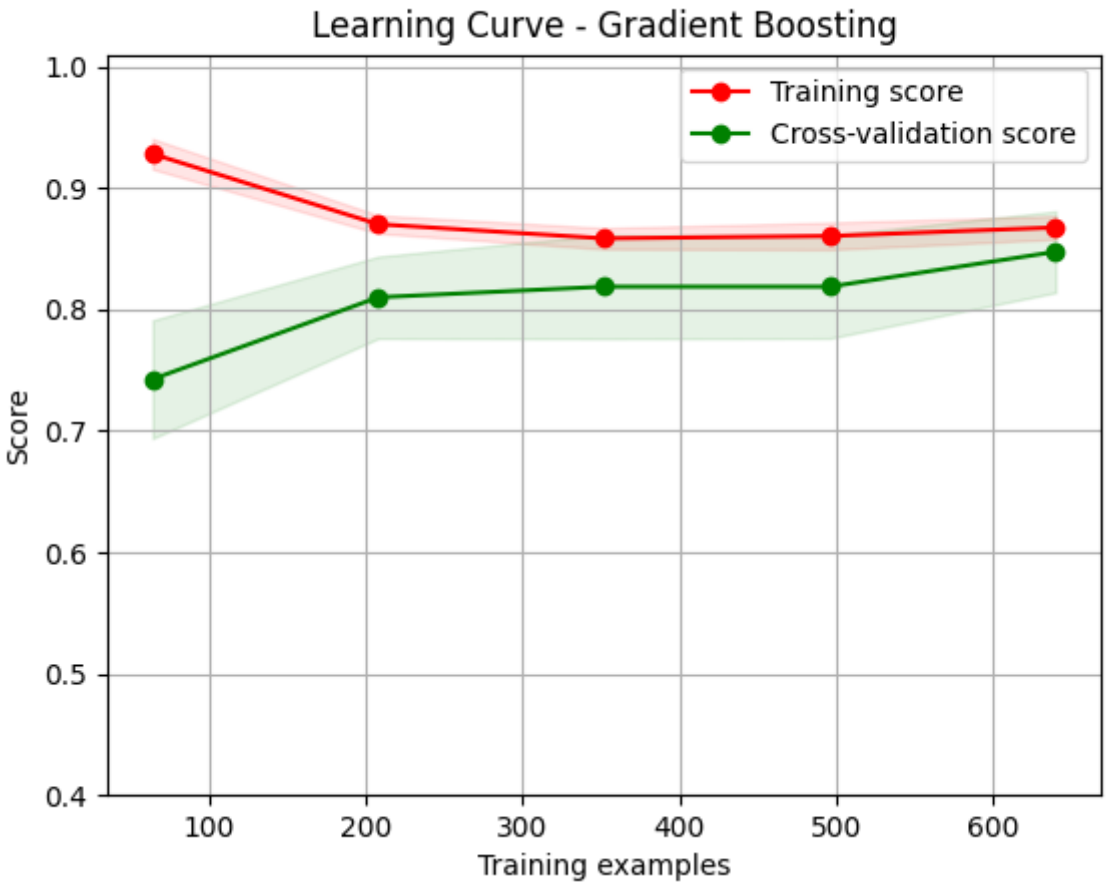
Tuttavia, pur impostando `n_estimators=97` o ad un altro valore vicino a 100, si ottengono sperimentalmente gli stessi risultati. Quindi anche in questo caso la scelta di `n_estimators=93` risulta coerente con la considerazione precedente e permette anche di ridurre la complessità del modello.

Gradient Boosting

Il modello `GradientBoostingClassifier` è un altro algoritmo di ensemble, ma a differenza della foresta casuale, costruisce gli alberi in modo sequenziale. Ogni nuovo albero cerca di correggere gli errori commessi dagli alberi precedenti, migliorando gradualmente la previsione. I parametri chiave ottimizzati sono:

- **`n_estimators=183`**: Il numero di alberi nella sequenza di boosting. Aumentando questo valore, si può migliorare la performance fino a un certo punto, oltre il quale si rischia l'overfitting.
- **`learning_rate=0.03526678108288866`**: La velocità di apprendimento che riduce l'impatto di ogni albero aggiunto, permettendo una costruzione più graduale del modello.
- **`max_depth=2`**, **`min_samples_split=10`**, e **`min_samples_leaf=16`**: Gli stessi parametri dei modelli basati su alberi, utilizzati per controllare la complessità di ciascun albero.

Curve di apprendimento



- **Training score**

  La curva rossa del punteggio di addestramento inizia molto alta, vicina a 0.93, e scende leggermente fino a stabilizzarsi intorno a 0.87. Questo comportamento indica che il modello inizialmente si adatta molto bene ai dati di addestramento, ma perde un po' di accuratezza man mano che vengono introdotti più dati.

- **Cross-validation score**

  La curva verde del punteggio di validazione incrociata inizia significativamente più bassa, intorno a 0.75, ma sale gradualmente fino a stabilizzarsi intorno a 0.83-0.85. La progressione positiva suggerisce che l'introduzione di più dati sta effettivamente aiutando il modello a migliorare la sua capacità di generalizzazione.

Sebbene le curve non si sovrappongano completamente, il divario tra il punteggio di addestramento e quello di test è moderato e si riduce gradualmente. Questo è un buon segnale: il modello non soffre di overfitting significativo e sta beneficiando dell'introduzione di nuovi dati.

Il gap tra i punteggi medi di addestramento e test è di circa 0.02, il che è relativamente piccolo. La varianza del punteggio di addestramento e del punteggio di test è molto bassa, indicando che il modello è stabile e non mostra instabilità significativa.

Risultati ottenuti con i parametri ottimizzati

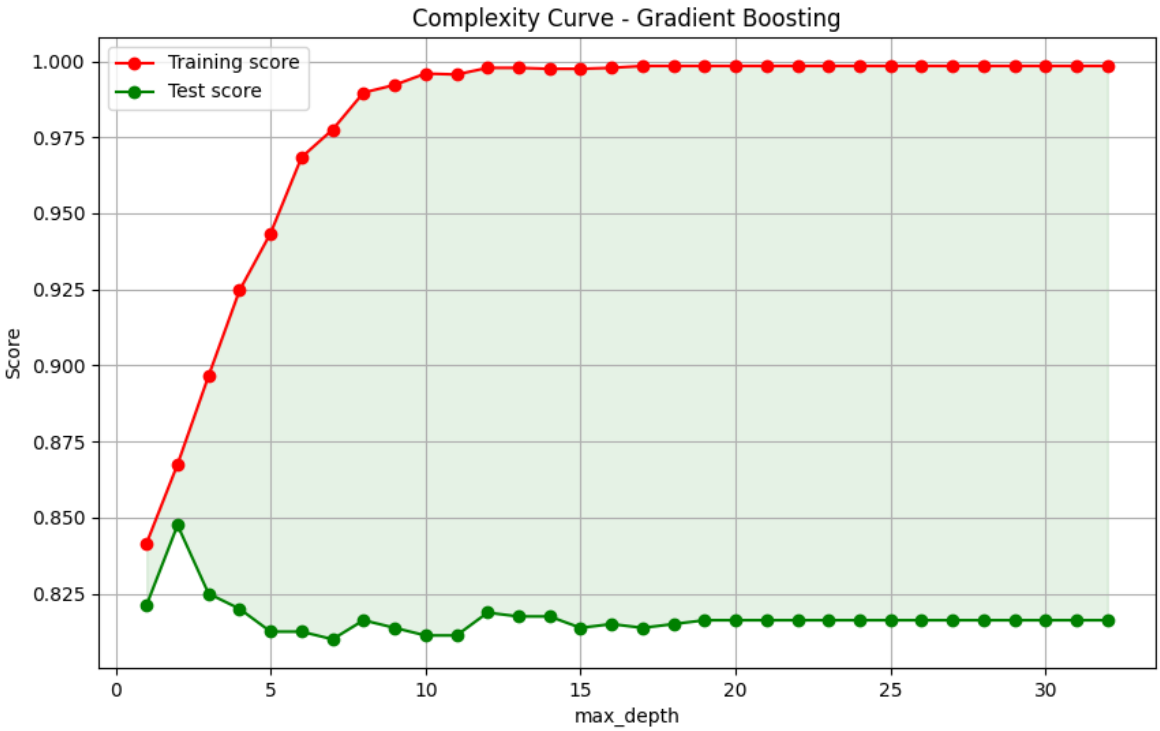
	Media	Std	Var
Train	0.867500	0.009550	0.000091
Test	0.847500	0.033448	0.001119

best_accuracy	0.8474
final_accuracy	0.8260
precision	0.8313
recall	0.8260
f1_score	0.8271

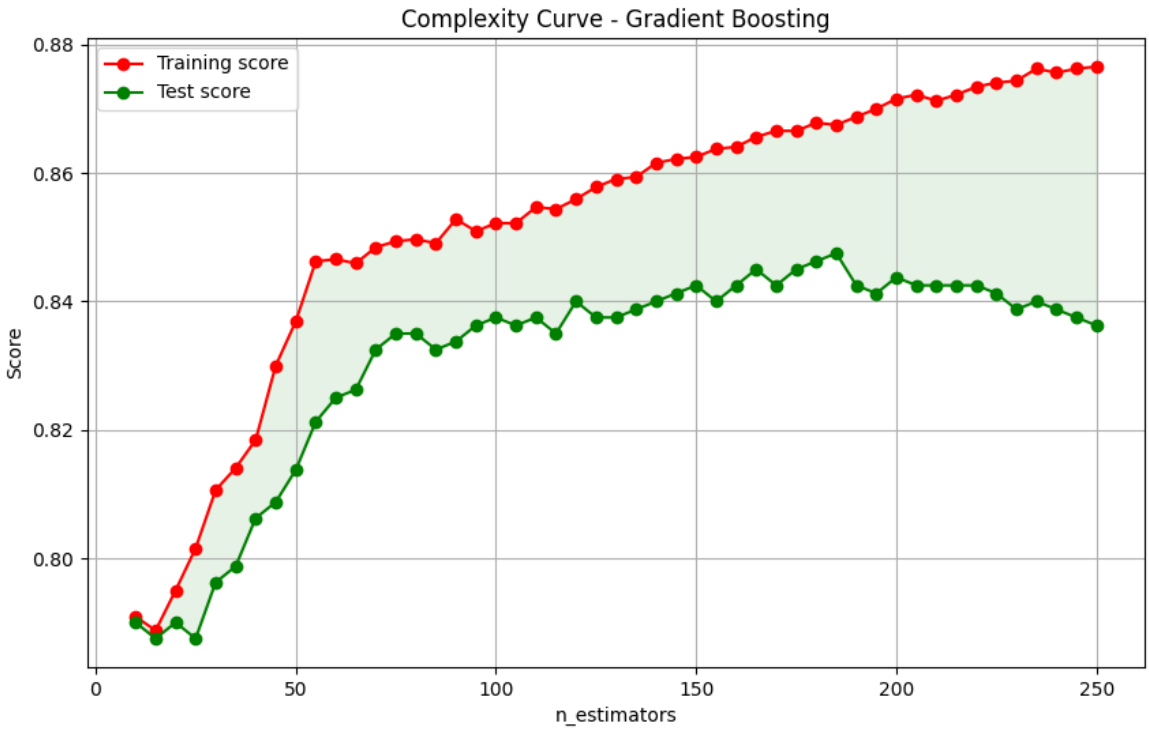
L'accuratezza finale è molto vicina alla best accuracy, indicando una buona generalizzazione.

Le metriche di precision, recall e F1-score sono tutte intorno a 0.83, suggerendo un buon equilibrio tra le misure di performance.

Anche in questo caso consideriamo come misure di complessità i parametri `n_estimators` e `max_depth`.



Il grafico mostra chiaramente che il modello di Gradient Boosting rischia di overfittare quando il parametro `max_depth` è troppo grande (oltre 3). Quindi il valore trovato con l'ottimizzazione `max_depth=2`, è prova del fatto che fornisca un miglior equilibrio tra complessità e capacità di generalizzazione, prevenendo l'overfitting e migliorando le prestazioni sui dati di test.



DA RICOPIARE L'IMMAGINE

Aumentare il numero di alberi inizialmente migliora le prestazioni del modello, tuttavia, oltre un certo punto (intorno a 180), lo score di cross-validation comincia a diminuire, mostrando una chiara divergenza tra entrambe le curve. Questo comportamento è indicatore di overfitting, quindi quando `n_estimators` è troppo grande il modello perde la capacità di generalizzare bene sui nuovi dati. Il valore ottenuto `n_estimators=183` dall'ottimizzazione è in linea con quest'ultima considerazione. Questa è una maggior conferma rispetto alle considerazioni fatte in precedenza sul fatto che il modello non soffra di overfitting.

Artificial Neural Network - (ANN)

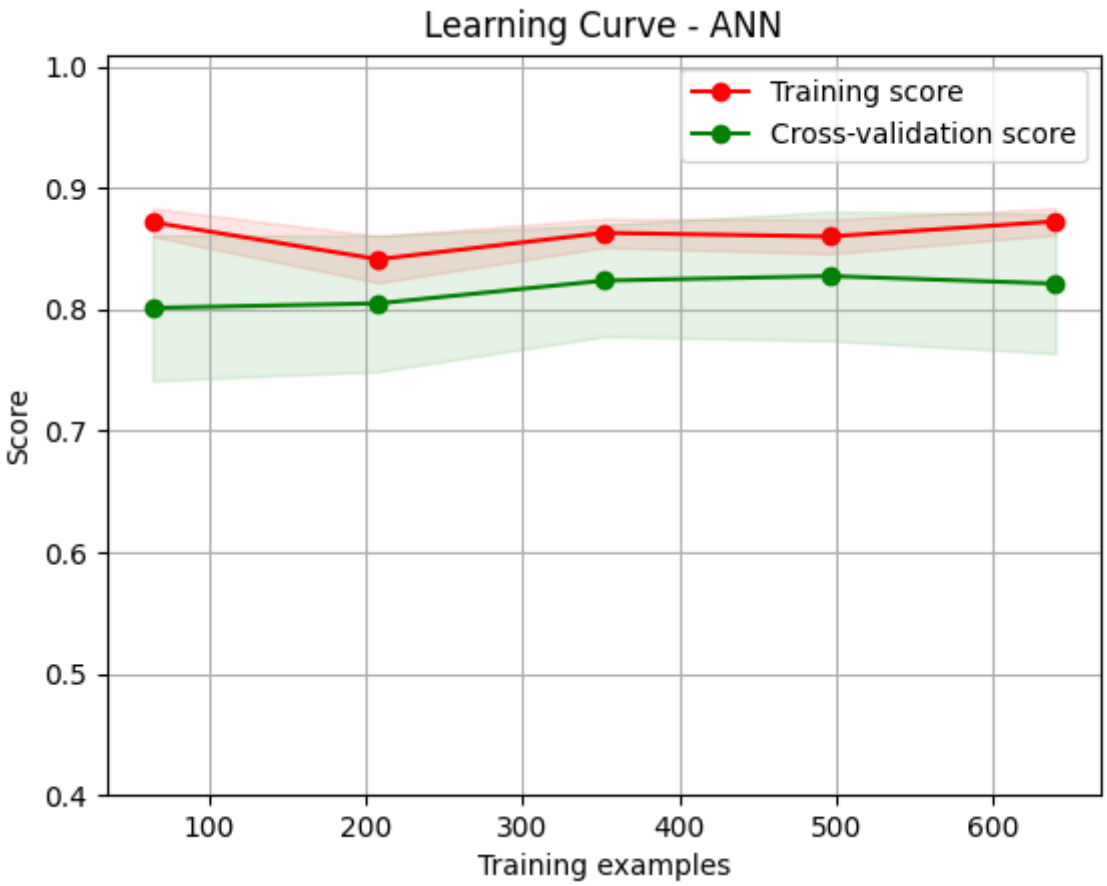
La rete neurale artificiale (ANN) è un modello potente ispirato al funzionamento del cervello umano, composto da strati di neuroni interconnessi. Ogni neurone applica una trasformazione ai dati e trasmette l'output al livello successivo.

L'architettura della rete neurale implementata è molto semplice: la rete ha un input layer, un hidden layer con ReLu come funzione di attivazione e dropout applicato dopo, e un output layer.

I parametri ottimizzati per l'ANN sono:

- **`module__hidden_size=64`** : Il numero di neuroni nello strato nascosto, che controlla la capacità della rete di apprendere rappresentazioni complesse.
- **`module__dropout_rate=0.6359806350639419`** : Il tasso di dropout applicato durante l'addestramento per prevenire l'overfitting, spegnendo casualmente alcuni neuroni.
- **`max_epochs=28`** : Il numero massimo di epoche di addestramento. Un numero maggiore di epoche permette alla rete di convergere, ma rischia di causare overfitting se non controllato.
- **`lr=0.0028864634848197596`** : Il tasso di apprendimento, che determina la dimensione dei passi presi dall'algoritmo di ottimizzazione durante l'aggiornamento dei pesi.
- **`batch_size=32`** : La dimensione dei lotti di dati su cui vengono calcolati gli aggiornamenti dei pesi in ciascuna iterazione.

Curve di apprendimento



- **Training score**

  La curva rossa dello score di training è relativamente stabile, oscillando intorno a un valore di circa 0.88. Pur essendoci alcune fluttuazioni, la curva si mantiene in un range abbastanza stabile (0.87-0.88), indicando una buon adattamento ai dati di addestramento.

- **Cross-validation score**

  La curva verde dello score di cross-validation inizia intorno a 0.80 e cresce leggermente fino a stabilizzarsi intorno a 0.82. Sebbene vi sia un miglioramento man mano che vengono utilizzati più dati, questa curva rimane al di sotto della curva di addestramento, suggerendo un possibile overfitting.

Le due curve non convergono, e c'è un leggero divario tra loro, che ci può suggerire presenza di overfitting.

Il gap tra i punteggi medi di addestramento e test è di circa 0.05, conferma il leggero overfitting osservato sulle curve di apprendimento. La varianza sul punteggio di test è più alta rispetto a quella sul punteggio di addestramento, indicando una certa instabilità nelle predizioni sui dati non visti.

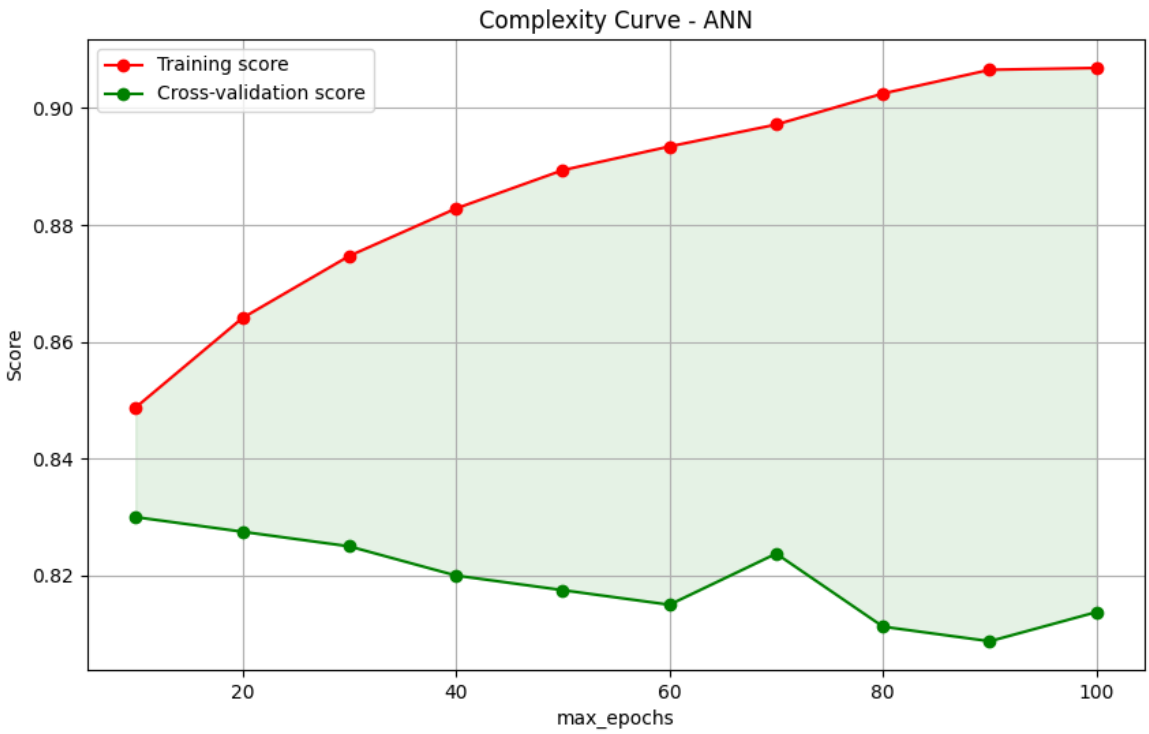
Risultati ottenuti con i parametri ottimizzati

	Media	Std	Var
Train	0.877500	0.013679	0.000187
Test	0.826250	0.048477	0.002350

best_accuracy	0.8387
final_accuracy	0.8206
precision	0.8250
recall	0.8206
f1_score	0.8216

La final accuracy è leggermente inferiore alla best accuracy. Il resto delle metriche indicano una performance complessiva buona e coerente.

Consideriamo `max_epochs` come misura di complessità:



Il grafico mostra chiaramente che la rete neurale inizia a overfittare all'aumentare del numero massimo di epoche. Il divario fra le due curve è particolarmente evidente oltre `max_epochs=30`, questa divergenza crescente è il campanello d'allarme per riconoscere il rischio di overfitting. Con l'ottimizzazione abbiamo ottenuto `max_epochs=28`, il che può risultare abbastanza coerente nell'ottenere un modello che non soffra di overfitting. Tuttavia dall'analisi svolta c'è una leggera tendenza all'overfitting, che potrebbe essere mitigata con una tecnica come l'**early stopping**.

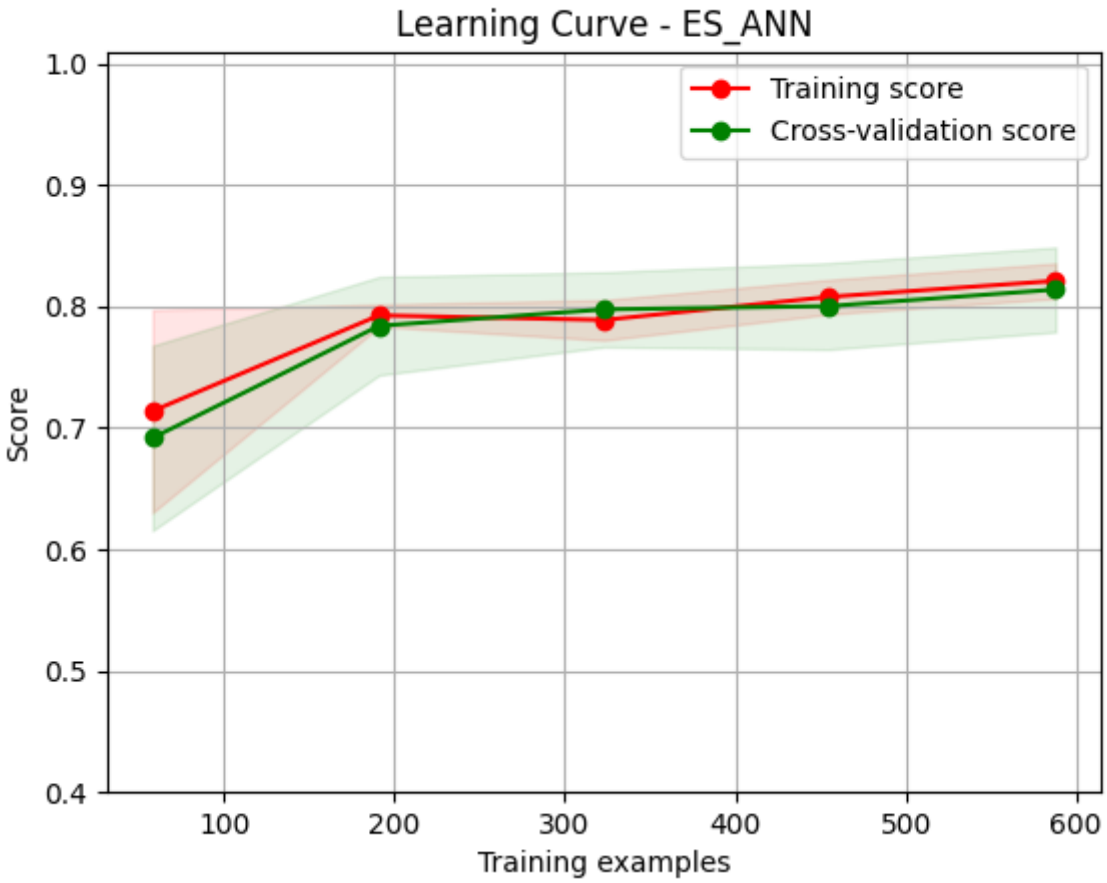
L'idea è stata quella di ripercorrere tutta la pipeline precedentemente discussa con un'unica differenza: includere il meccanismo di **early stopping** sia in fase di ottimizzazione sia in fase di esecuzione, in maniera tale da accelerare il processo di ottimizzazione e da garantire che il modello non overfitti e che sia addestrato solo fino al punto ottimale. Nel nostro caso l'**early stopping** funziona monitorando l'**accuracy sui dati di validazione** durante l'addestramento della rete neurale: se l'accuracy non migliora per 5 epoche consecutive, l'addestramento viene interrotto prematuramente.

I parametri ottenuti dopo questo procedimento sono i seguenti:


```
"best_params": {
        "module__hidden_size": 64,
        "module__dropout_rate": 0.6513712125759036,
        "max_epochs": 20,
        "lr": 0.000692804683095548,
        "batch_size": 32
    }
```


Con 20 come numero massimo di epoche ci aspettiamo che il modello non overfitti.

Analizziamo le **curve di apprendimento** e i risultati ottenuti per questo nuovo modello:



Risultati ottenuti

	Media	Std	Var
Train	0.826190	0.012188	0.000149
Test	0.811224	0.054613	0.002983

best_accuracy	0.8138
final_accuracy	0.8260
precision	0.8281
recall	0.8260
f1_score	0.8267

Notiamo diversi miglioramenti, infatti le curve del train score e del test score sono molto vicine, il che indica un modello bilanciato senza evidenti segni di overfitting. Le varianze sono relativamente basse e il gap tra train e test score è ridotto, suggerendo un modello con una

maggiore stabilità e capacità di generalizzazione. Per quanto riguarda il resto delle metriche è possibile notare un leggero miglioramento rispetto al modello precedente.

Quindi se l'obiettivo è massimizzare la capacità di generalizzazione e minimizzare il rischio di overfitting, quest'ultimo modello può essere la scelta preferibile.

Risultati complessivi

Di seguito i risultati ottenuti per ciascun modello discusso

Model	Accuracy	Precision	Recall	F1 Score
Decision Tree	0.766	0.775	0.766	0.768
Logistic Regression	0.788	0.795	0.788	0.789
Random Forest	0.848	0.85	0.848	0.848
Gradient Boosting	0.826	0.831	0.826	0.827
ANN	0.821	0.825	0.821	0.822
MD8 Decision Tree	0.761	0.771	0.761	0.763
ES_ANN	0.826	0.828	0.826	0.827

Per questo specifico problema, il Random Forest sembra essere il modello più performante, seguito da Gradient Boosting e reti neurali.