Analisi del Dataset BCWD
link repo GitHub:
https://github.com/SettecentoTrenta/ICON

Francesco Spada [778544]
Anno Accademico: 2024/2025

August 27, 2025

Contents

Chapter 1

Analisi del Dataset BCWD

1.1 Introduzione

Il seguente progetto si focalizza sulla classificazione di masse tumorali al seno, distinguendo tra diagnosi benigne e maligne. A tale scopo, vengono impiegate tecniche di machine learning applicate a un dataset pubblico contenente feature estratte da immagini digitalizzate utilizzando la tecnica dell'agobiopsia.

Il dataset utilizzato è il *Breast Cancer Wisconsin (Diagnostic) Data Set (BCWD)*, disponibile presso l'UCI Machine Learning Repository al seguente link: `https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)`.

1.1.1 Strumenti Utilizzati

Il prgoetto è sviluppato in Python, utilizzando le seguenti librerie:

- **Pandas**: per la manipolazione e l'analisi dei dati.
- **NumPy**: per il calcolo numerico.
- **Matplotlib** e **Seaborn**: per la visualizzazione dei dati.
- **pgmpy**: per la costruzione e l'inferenza di reti bayesiane.
- **NetworkX**: per la visualizzazione delle reti bayesiane
- **Scikit-learn**: per le tecniche di machine learning.

1.1.2 Argomenti Trattati

Possiamo suddividere il progetto nelle seguenti fasi principali:

1. Esplorazione del dataset
2. Apprendimento supervisionato
3. Confronto tra i modelli
4. Apprendimento della struttura del dataset
5. Conclusioni

Chapter 2

Esplorazione del Dataset

2.1 Analisi Della Struttura del Dataset

Sono prendi un totale di 569 campioni e 32 features (inclusa la colonna 'id' e 'diagnosis').

Le possiamo riassumere in:

Table 2.1: Descrizione delle 10 features di base

Feature	**Significato**
radius	Media delle distanze dal centro ai punti del perimetro
texture	Deviazione standard dei valori in scala di grigi
perimeter	Perimetro della massa tumorale
area	Area della massa tumorale
smoothness	Variazione locale nelle lunghezze dei raggi
compactness	Perimetro^2 / area - 1.0
concavity	Gravità delle porzioni concave del contorno
concave points	Numero di porzioni concave del contorno
symmetry	Simmetria della massa tumorale
fractal dimension	"Approssimazione della linea di costa" - 1

Per ogni features sono state calcolate 3 statistiche: media, errore standard e "worst" (valore più alto), quindi le features complete sono le seguenti:

```
'id', 'diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
'fractal_dimension_se', 'radius_worst', 'texture_worst', 'perimeter_worst',
'area_worst', 'smoothness_worst', 'compactness_worst', 'concavity_worst',
'concave points_worst', 'symmetry_worst', 'fractal_dimension_worst'
```

Listing 2.1: Colonne del dataset

2.1.1 Tipi di Dato

La features 'id' è di tipo intero, 'diagnosis' è di tipo categorico (M = maligno, B = benigno), mentre tutte le altre sono di tipo float.

Non sono presenti valori mancanti nel dataset.

2.2 Analisi della distribuzione delle variabili

La colonna 'diagnosis' è la nostra variabile target, che indica se un tumore è Maligno ('M') o Benigno ('B'). Per visualizzare la distribuzione di queste due classi, sono stati utilizzati:

- **Count Plot**: Un grafico a barre che mostra il numero di campioni per ogni diagnosi.

- **Grafico a Torta**: Fornisce una rappresentazione percentuale della distribuzione delle diagnosi.

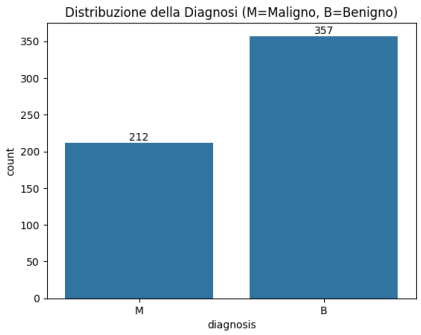


Figure 2.1: Distribuzione delle diagnosi nel dataset.

In percentuali, invece:

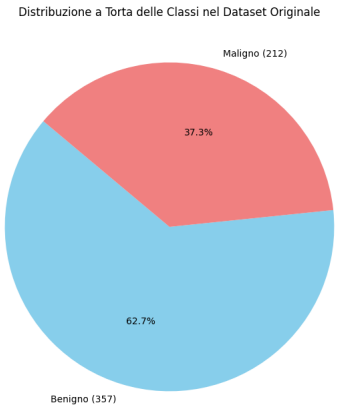


Figure 2.2: Distribuzione delle diagnosi nel dataset.

Possiamo notare che il dataset è sbilanciato, con una maggioranza di diagnosi benigne (357) rispetto a quelle maligne (212).

2.2.1 SMOTE per il Bilanciamento delle Classi

Per affrontare il problema dello sbilanciamento delle classi, è stata applicata la tecnica SMOTE (Synthetic Minority Over-sampling Technique). SMOTE genera nuovi campioni sintetici della classe minoritaria (in questo caso, le diagnosi maligne) per bilanciare il dataset e migliorare le performance dei modelli di machine learning.

Così facendo, il numero di campioni per entrambe le classi diventa uguale.

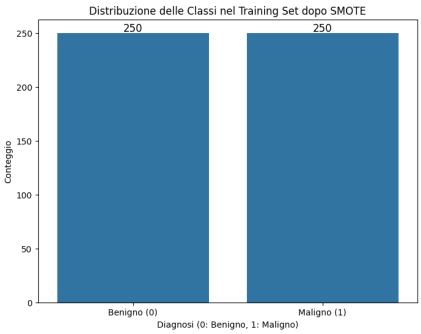


Figure 2.3: Distribuzione delle diagnosi nel dataset (dopo SMOTE).

Con un grafico a torta:

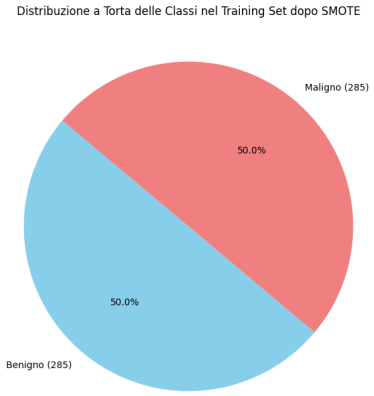


Figure 2.4: Distribuzione delle diagnosi nel dataset (dopo SMOTE).

Distribuzione delle Features

Possiamo notare la distribuzione delle features prima e dopo l'applicazione di SMOTE.

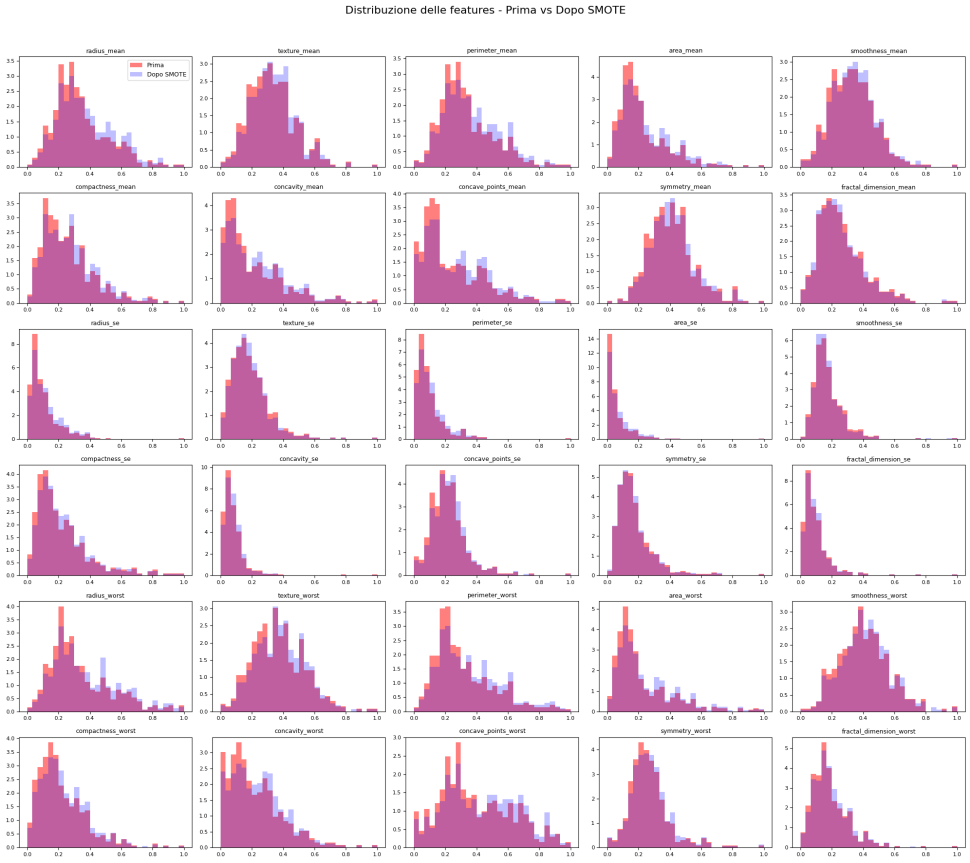


Figure 2.5: Confronto distribuzione delle features prima e dopo SMOTE.

2.3 Matrice di Correlazione

Per comprendere le relazioni lineari tra le diverse features numeriche, è stata calcolata e visualizzata una matrice di correlazione.

- La colonna `id` è stata esclusa in quanto non è una feature rilevante per l'analisi.
- Una **heatmap** è stata utilizzata per visualizzare la matrice di correlazione.

I colori più caldi indicano una correlazione positiva forte, mentre i colori più freddi indicano una correlazione negativa forte.

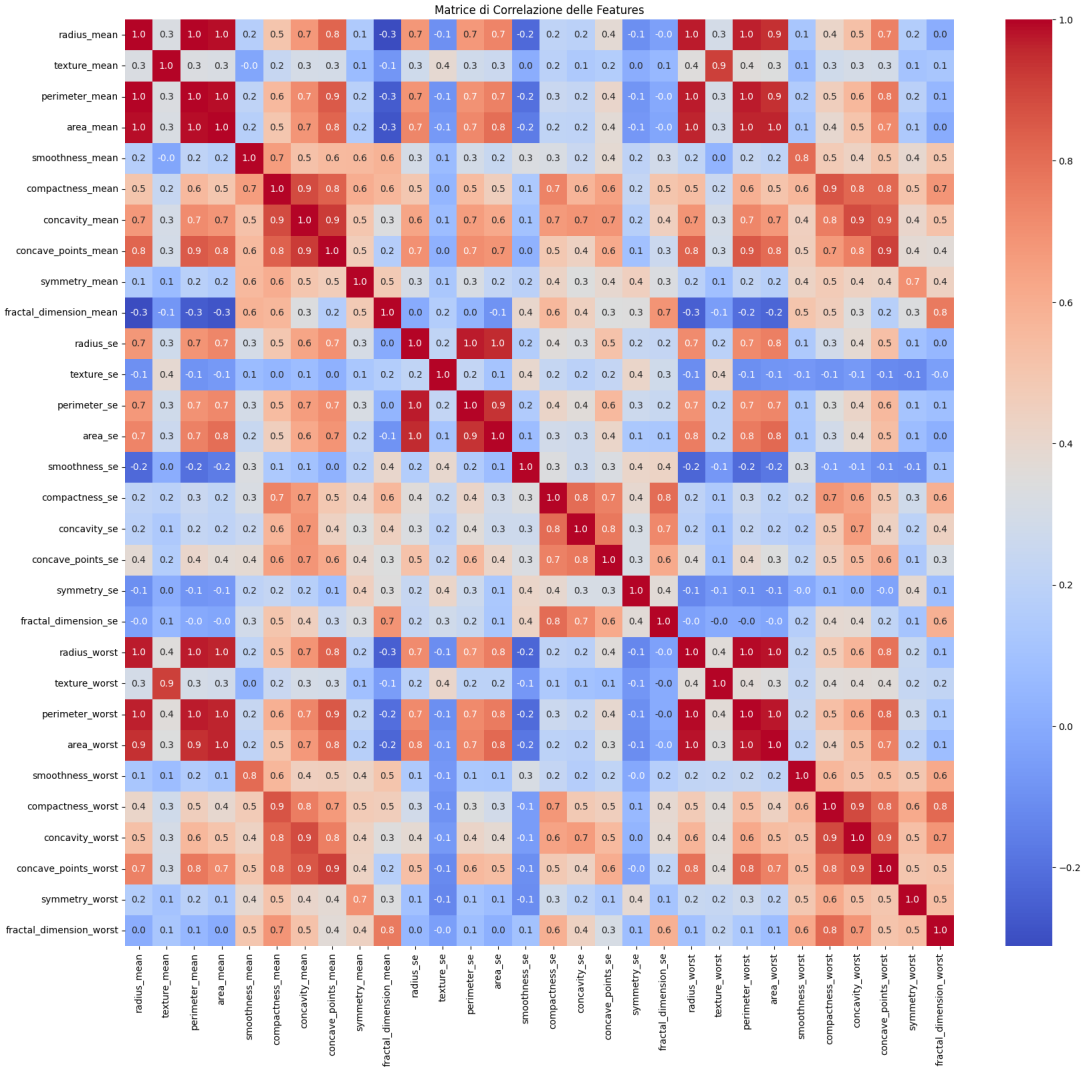


Figure 2.6: Matrice di correlazione delle features del dataset.

Osservazioni

Dall'analisi della heatmap della matrice di correlazione, emergono diverse osservazioni interessanti:

- **Forte correlazione tra Features**: Molte features mostrano una correlazione molto alta tra loro (valori vicini a +1).
  - Ad esempio, le features `radius_mean`, `perimeter_mean`, e `area_mean` sono fortemente correlate. Questo è prevedibile, poiché sono tutte misure legate alla dimensione del nucleo cellulare. Un raggio maggiore implica un perimetro e un'area maggiori.
  - Similmente, features come `compactness_mean`, `concavity_mean`, e `concave_points_mean` mostrano una forte correlazione positiva tra loro.

- **Ridondanza delle Features**: Molte features hanno una forte correlazione tra loro, questo suggerisce che alcune features possono essere ridondanti. Ad esempio, `perimeter_mean` fornisce informazioni molto simili a `radius_mean`.
- **Features meno Correlate**: Features come `fractal_dimension_mean` e `smoothness_mean` mostrano correlazioni più deboli con le altre features.

Chapter 3

Apprendimento Supervisionato

3.1 Cos'è l'apprendimento Supervisionato?

L'apprendimento supervisionato è una tecnica di machine learning in cui un modello viene addestrato su un dataset etichettato, ovvero un insieme di dati in cui ogni esempio di input è associato a un'etichetta.

L'obiettivo è quello di far sì che il modello impari a mappare gli input alle etichette corrette, in modo da poter fare previsioni accurate su nuovi dati non visti durante l'addestramento.

3.2 Suddivisione del Dataset

Prima di addestrare qualsiasi modello di apprendimento supervisionato, è una pratica fondamentale suddividere i dati a disposizione in almeno due insiemi separati: un set di addestramento (training set) e un set di test (test set). Questa separazione è cruciale per valutare correttamente le performance del modello.

- **Training Set**: Questa porzione di dati viene utilizzata per addestrare il modello, permettendogli di apprendere le relazioni e i pattern presenti nei dati.
- **Test Set**: Questo insieme, che il modello non ha mai visto durante la fase di addestramento, viene utilizzato per valutare quanto bene il modello generalizza su nuovi dati non visti.

Questa metodologia consente di ottenere una misura della capacità predittiva del modello e di verificare che non abbia semplicemente "imparato a memoria" i dati di addestramento (overfitting).

Note

- La colonna `id` è stata rimossa in quanto non fornisce informazioni utili per la classificazione.
- La colonna `diagnosis` è stata convertita in valori numerici: 'M' = 1, 'B' = 0. Inoltre, è stata separata dalle altre features per essere utilizzata come variabile target.

3.3 Preparazione dei Dati

In questa fase, prepariamo il dataset per l'addestramento dei modelli di classificazione. I passaggi eseguiti sono i seguenti:

- **Separazione delle Features e della Variabile Target**:
  - Il dataset viene diviso in 'X', che contiene tutte le colonne predittive (le features), e 'y', che contiene la colonna target ('diagnosis'), ovvero l'etichetta che vogliamo predire (0 per benigno, 1 per maligno).
- **Suddivisione in Set di Addestramento e di Test**:

– I dati vengono suddivisi in un set di addestramento ('X$_train$', 'y_train')$eunsetditest('X_test', 'y_test')utilizzandolafun$

- **Bilanciamento del Set di Addestramento con SMOTE**:
  - Per affrontare il problema dello sbilanciamento delle classi nel set di addestramento, viene applicata la tecnica **SMOTE** (Synthetic Minority Over-sampling Technique).
  - SMOTE genera nuovi campioni sintetici per la classe di minoranza, portando il numero di campioni di entrambe le classi allo stesso livello.
- **Scalatura delle Features**:
  - Le features numeriche vengono scalate utilizzando 'MinMaxScaler'. Questo trasforma ogni feature in modo che i suoi valori siano compresi nell'intervallo [0, 1].
  - La scalatura è importante perché molti algoritmi di machine learning sono sensibili alla scala delle variabili di input.

Alla fine di questo processo, otteniamo i dati pronti per essere utilizzati per addestrare e valutare i nostri modelli.

3.4 Random Forest (RF)

3.4.1 Descrizione del Modello

Il Random Forest è un modello di apprendimento supervisionato di tipo *ensemble*, che combina le previsioni di più alberi decisionali per ottenere una previsione più accurata e stabile.

Per la classificazione, ogni albero nel Random Forest "vota" per una classe, e la classe che riceve il maggior numero di voti viene scelta come previsione finale.

3.4.2 Implementazione

Il modello di Random Forest è stato implementato utilizzando la libreria `scikit-learn` in Python. Sono stati utilizzati i seguenti parametri:

Table 3.1: Iperparametri del modello Random Forest

Iperparametro	**Spiegazione**
`n_estimators`	Numero di alberi decisionali nella foresta.
`max_depth`	Profondità massima di ogni albero.
`min_samples_split`	Numero minimo di campioni richiesti per suddividere un nodo.
`min_samples_leaf`	Numero minimo di campioni richiesti in un nodo foglia.
`criterion`	Funzione per misurare la qualità di una suddivisione (es. "gini").

3.5 Gradient Boosting (GB)

3.5.1 Descrizione del Modello

Il Gradient Boosting è un algoritmo di tipo ensemble che costruisce un modello predittivo in modo iterativo e sequenziale. L'idea fondamentale è quella di combinare più modelli deboli, tipicamente alberi decisionali, per creare un unico modello robusto e accurato.

A differenza di altri metodi ensemble come il Random Forest, dove gli alberi vengono costruiti in parallelo e in modo indipendente, nel Gradient Boosting ogni nuovo albero viene addestrato per correggere gli errori residui commessi dalla combinazione degli alberi precedenti. Questo processo di correzione progressiva consente al modello di focalizzarsi sui campioni più difficili da classificare, migliorando gradualmente le performance complessive.

3.5.2 Implementazione

Anche il modello di Gradient Boosting è stato implementato utilizzando la libreria `scikit-learn`. I principali iperparametri utilizzati sono:

Table 3.2: Iperparametri del modello Gradient Boosting

Iperparametro	Spiegazione
`learning_rate`	Riduce il contributo di ogni albero; c'è un trade-off con `n_estimators`.
`n_estimators`	Il numero di stadi di boosting da eseguire (numero di alberi).
`subsample`	Frazione di campioni da utilizzare per l'addestramento di ogni albero.
`max_depth`	Profondità massima dei singoli stimatori di regressione (alberi).

3.6 K-Nearest Neighbors (KNN)

3.6.1 Descrizione del Modello

Il K-Nearest Neighbors (KNN) è un algoritmo di apprendimento supervisionato. È un metodo semplice e intuitivo. L'idea principale alla base del KNN è che i punti simili tendono a essere vicini nello spazio delle feature.

Per classificare un nuovo punto, l'algoritmo calcola la distanza tra questo punto e tutti i punti nel dataset di addestramento. Successivamente, seleziona i 'k' punti più vicini (i "vicini") e assegna al nuovo punto la classe più comune tra questi 'k' vicini.

3.6.2 Implementazione

Il modello KNN è stato implementato utilizzando la libreria `scikit-learn`. I principali iperparametri considerati sono:

Table 3.3: Iperparametri del modello KNN

Iperparametro	Spiegazione
`n_neighbors`	Numero di vicini da utilizzare per la classificazione.
`weights`	Funzione di peso. 'uniform' o 'distance'.
`metric`	Metrica di distanza da utilizzare (es. 'euclidean').

3.7 Linear Regression (LR)

3.7.1 Descrizione del Modello

La Regressione Logistica (LR) è un modello di apprendimento supervisionato utilizzato per problemi di classificazione, in particolare per la classificazione binaria. Nonostante il nome, è un modello di classificazione e non di regressione.

Il modello stima la probabilità che un campione appartenga a una determinata classe utilizzando la funzione logistica (o sigmoide). L'output della funzione sigmoide è un valore compreso tra 0 e 1, che può essere interpretato come la probabilità della classe positiva. Una soglia (solitamente 0.5) viene quindi utilizzata per assegnare il campione a una delle due classi.

3.7.2 Implementazione

Il modello di Regressione Logistica è stato implementato utilizzando la libreria `scikit-learn`. I principali iperparametri considerati sono:

Table 3.4: Iperparametri del modello di Regressione Logistica

Iperparametro	Spiegazione
`penalty`	Specifica la norma utilizzata nella penalizzazione (es. 'l1', 'l2').
`C`	Inverso della forza di regolarizzazione; valori più piccoli indicano una regolarizzazione più forte.
`solver`	Algoritmo da utilizzare nel problema di ottimizzazione.
`max_iter`	Numero massimo di iterazioni per la convergenza del solver.

3.8 Tuning degli Iperparametri

Per ottimizzare le performance di ciascun modello, è stata eseguita una fase di *tuning* degli iperparametri. Questo processo consiste nel trovare la combinazione di iperparametri che massimizza le prestazioni del modello.

3.8.1 Randomized Search

Per questa operazione, è stata utilizzata la tecnica di **Randomized Search**, implementata dalla funzione `RandomizedSearchCV` di `scikit-learn`. A differenza di una ricerca esaustiva su una griglia di parametri (`GridSearchCV`), la Randomized Search campiona un numero fisso di combinazioni di parametri da distribuzioni specificate.

Questo approccio offre diversi vantaggi e svantaggi:

Table 3.5: Vantaggi e Svantaggi di `RandomizedSearchCV`

Vantaggi	Svantaggi
• **Efficienza computazionale**: È molto più veloce della GridSearchCV, specialmente quando lo spazio dei parametri è grande. Non esplora tutte le combinazioni, risparmiando tempo e risorse. • **Flessibilità**: Permette di esplorare distribuzioni di parametri continue, rendendola utile per ottimizzare iperparametri come le regolarizzazioni o i learning rate.	• **Non garantisce la combinazione migliore**: Non c'è la garanzia di trovare la combinazione di parametri globalmente ottimale, poiché la ricerca è casuale e non esaustiva. • **Sensibilità a n_iter**: La qualità della soluzione dipende dal numero di iterazioni (`n_iter`); un numero troppo basso potrebbe non trovare una buona soluzione, mentre uno troppo alto ne riduce i vantaggi di velocità.

3.8.2 Funzionamento di RandomizedSearchCV

Ecco come funziona:

- **Campionamento Casuale**: A differenza di GridSearchCV che prova tutte le combinazioni possibili, RandomizedSearchCV seleziona un numero fisso di combinazioni di iperparametri in modo casuale. Questo lo rende molto più veloce e spesso altrettanto efficace.
- **Convalida Incrociata (Cross-Validation)**: Per ogni combinazione di iperparametri selezionata, esegue una convalida incrociata (in questo caso `cv=10`). Questo significa che divide i dati in 10 parti, addestra il modello su 9 e lo valuta sulla parte rimanente, ripetendo il processo 10 volte.
- **Selezione del Migliore**: Dopo aver testato tutte le combinazioni, identifica quella che ha prodotto il punteggio migliore (in questo caso, l'accuracy) e la presenta come la configurazione ottimale. Il modello finale viene quindi ri-addestrato su tutti i dati con questi iperparametri.

Alcuni dei parametri chiave utilizzati in `RandomizedSearchCV` sono:

Table 3.6: Parametri chiave di `RandomizedSearchCV`

Parametro	**Spiegazione**
`n_iter=10`	Numero di combinazioni di iperparametri da provare. `RandomizedSearchCV` selezionerà e valuterà 10 combinazioni casuali.
`cv=10`	Specifica il numero di "fold" per la convalida incrociata. Per ogni combinazione di iperparametri, i dati di addestramento vengono divisi in 10 parti; il modello viene addestrato 10 volte su 9 parti e testato sulla decima rimanente.
`scoring='accuracy'`	La metrica utilizzata per valutare e confrontare le diverse combinazioni di iperparametri. In questo caso, si usa l'accuracy.

3.8.3 Motivazione della Scelta

È stato scelto di utilizzare `RandomizedSearchCV` in quanto GridSearchCV, che esplora tutte le combinazioni possibili di iperparametri, richiede più risorse computazionali, e questo è un problema quando lo spazio degli iperparametri è ampio. RandomizedSearchCV, selezionando un numero fisso di combinazioni casuali, offre un compromesso tra esplorazione dello spazio degli iperparametri ed efficienza computazionale.

Con i parametri testati in questo progetto, RandomizedSearchCV viene eseguito molto più velocemente rispetto a GridSearchCV, pur trovando combinazioni di iperparametri altrettanto efficaci.

Chapter 4

Confronto tra i Modelli

4.1 Metriche di Valutazione

Prima di descrivere le metriche, è utile definire i termini fondamentali derivati dalla matrice di confusione:

- **TP (True Positive)**: Vero Positivo. Il modello ha predetto correttamente la classe positiva.
- **TN (True Negative)**: Vero Negativo. Il modello ha predetto correttamente la classe negativa.
- **FP (False Positive)**: Falso Positivo. Il modello ha predetto la classe positiva quando in realtà era negativa.
- **FN (False Negative)**: Falso Negativo. Il modello ha predetto la classe negativa quando in realtà era positiva.

Per valutare le performance dei modelli di classificazione, sono state utilizzate le seguenti metriche:

Table 4.1: Metriche di Valutazione per la Classificazione

Metrica	**Spiegazione**	**Formula**
Accuracy	Percentuale di previsioni corrette sul totale delle previsioni.	$\frac{TP+TN}{TP+TN+FP+FN}$
Precision	Percentuale di veri positivi tra tutte le previsioni positive.	$\frac{TP}{TP+FP}$
Recall	Percentuale di veri positivi tra tutti i casi effettivamente positivi.	$\frac{TP}{TP+FN}$
F1-Score	Media armonica tra precision e recall.	$2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$

4.2 Risultati dei Modelli

Dopo aver addestrato e testato i modelli di Random Forest, Gradient Boosting, K-Nearest Neighbors e Regressione Logistica, sono stati ottenuti i seguenti risultati:

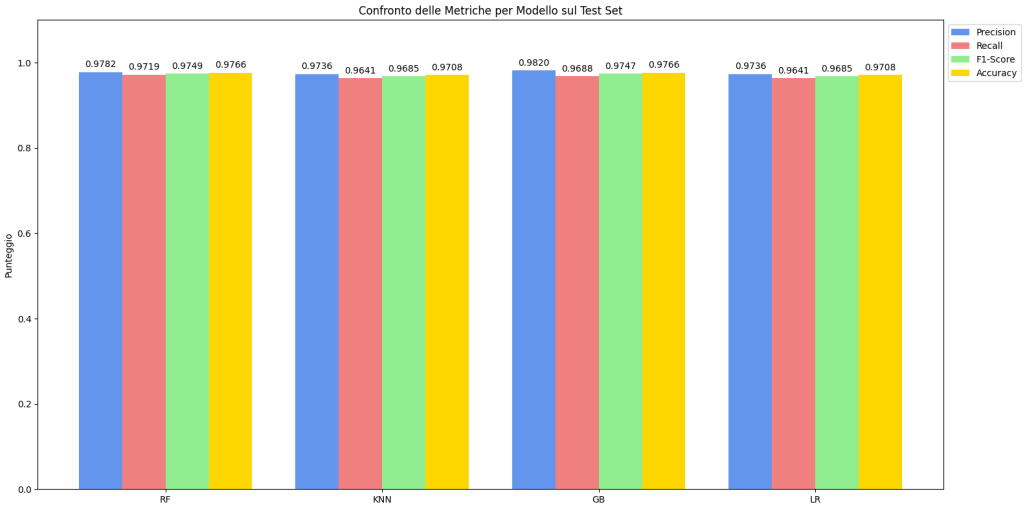
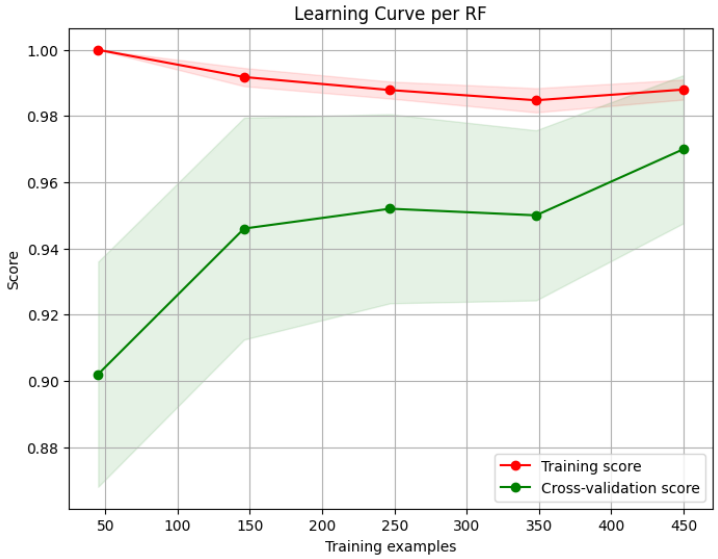
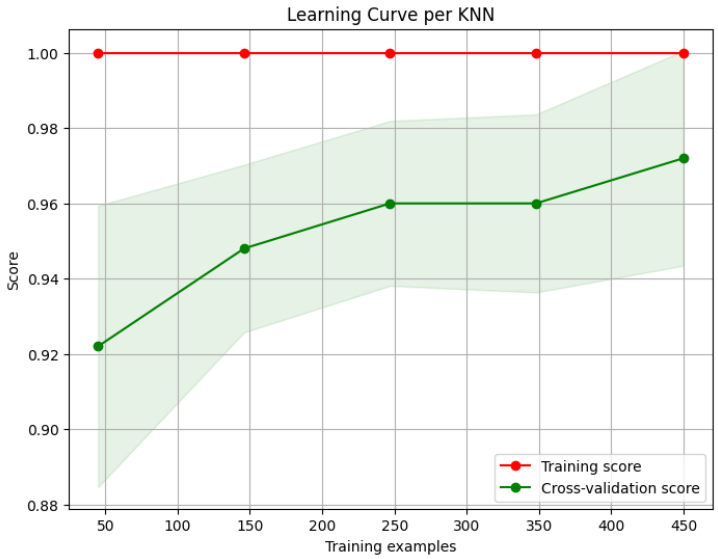
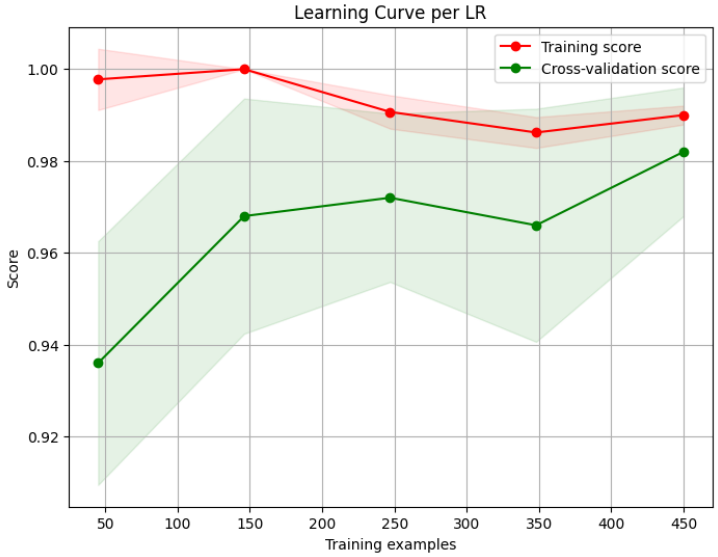


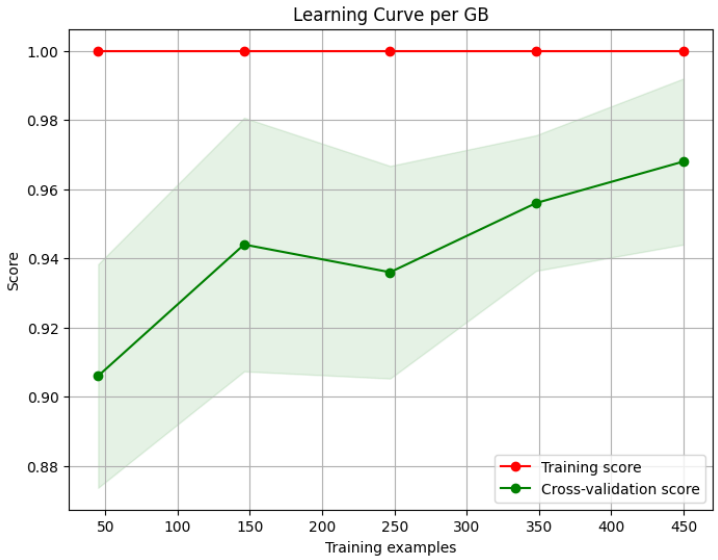
Figure 4.1: Metriche dei modelli.

Inoltre, per ogni modello sono state generate le rispettive learning curve:









4.3 Conclusioni sui Risultati

Dall'analisi delle learning curve possiamo notare che:

- La varianza nel modello Logistic Regression è bassa, soprattutto verso la fine del training.
- Random Forest ha una varianza leggermente più alta, ma comunque contenuta.
- Gradient Boosting e KNN mostrano una varianza più elevata, con una differenza maggiore tra le curve di training e test.

Gradient Boosting e KNN sembrano convergere più lentamente rispetto a Logistic Regression e Random Forest.

Chapter 5

Apprendimento della Struttura del Dataset

L'apprendimento della struttura è un processo utilizzato nelle reti bayesiane per determinare la struttura del grafo che rappresenta le relazioni di dipendenza tra le variabili. Invece di definire manualmente le connessioni tra le variabili, l'apprendimento della struttura utilizza algoritmi per scoprire automaticamente queste relazioni a partire dai dati.

5.1 Local Search

La ricerca locale è una tecnica di ottimizzazione utilizzata per trovare una soluzione approssimativa a problemi complessi. Invece di esplorare l'intero spazio delle soluzioni, la ricerca locale inizia con una soluzione iniziale e iterativamente esplora le soluzioni vicine, cercando di migliorare la soluzione corrente.

Alcuni algoritmi di ricerca locale sono i seguenti:

Table 5.1: Algoritmi di Ricerca Locale

Algoritmo	**Spiegazione**
Hill Climbing	Inizia con una soluzione e cerca iterativamente una soluzione vicina migliore. Si ferma quando raggiunge un "picco" (ottimo locale).
Tabu Search	Una variante di Hill Climbing che utilizza una lista per tenere traccia delle soluzioni visitate di recente, evitando di rimanere bloccato in cicli o ottimi locali.
Genetic Algorithms	Un approccio basato sulla popolazione che si ispira all'evoluzione. Mantiene un insieme di soluzioni candidate e le combina (crossover) e modifica (mutazione) per generare nuove soluzioni.

5.2 Hill Climbing

L'algoritmo di Hill Climbing è una tecnica di ottimizzazione locale utilizzata per l'apprendimento della struttura delle reti bayesiane. L'algoritmo inizia con una struttura iniziale (che può essere vuota o casuale) e iterativamente apporta modifiche alla struttura per migliorare una funzione obiettivo, nel nostro caso, il K2 score.

5.2.1 perché Hill Climbing?

L'algoritmo di Hill Climbing è stato scelto per l'apprendimento della struttura delle reti bayesiane in quanto, rispetto ad altri algoritmi di ricerca, offre un buon compromesso tra efficienza computazionale e qualità della soluzione trovata.

5.3 Risultati

Dopo aver applicato l'algoritmo di Hill Climbing per l'apprendimento della struttura della rete bayesiana, è stata ottenuta la seguente struttura:

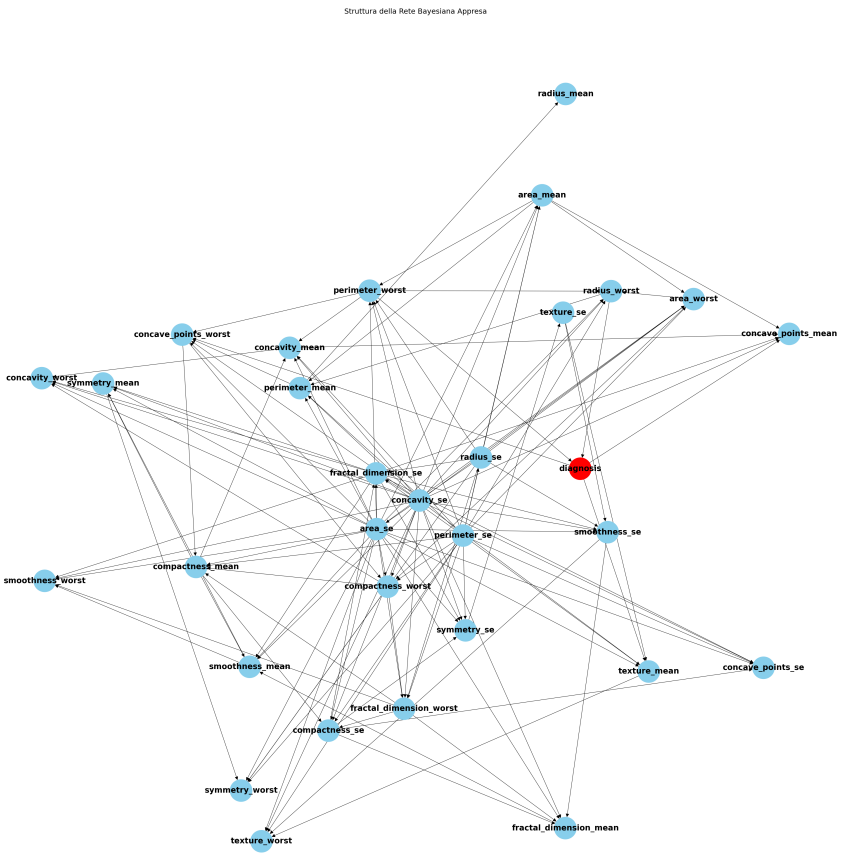


Figure 5.1: Rete Bayesiana appresa con Hill Climbing.

5.3.1 Analisi della Struttura Appresa

Dall'analisi della struttura appresa della rete bayesiana, emergono alcune osservazioni interessanti:

- **Relazioni tra Variabili**: La rete mostra diverse connessioni tra le variabili, indicando che alcune features sono direttamente influenzate da altre.
- **Assenza di Connessioni**: Alcune variabili non sono connesse ad altre, il che potrebbe indicare che esse non hanno un'influenza significativa sulla diagnosi o che la loro relazione non è stata catturata dai dati.

Limitazioni

È importante notare che l'algoritmo di Hill Climbing può rimanere bloccato in ottimi locali, e la struttura appresa dipende fortemente dalla configurazione iniziale e dai dati disponibili. Pertanto, la rete appresa potrebbe non rappresentare completamente tutte le relazioni esistenti tra le variabili.

Nota Ricordiamo che la rete persenta un numero di features molto elevato (30), e questo può rendere difficile l'interpretazione della struttura appresa, per questo motivo, è possibile ridurre il numero di features utilizzando tecniche come Markov Blanket.

5.4 Markov Blanket

Il Markov Blanket di una variabile in una rete bayesiana è l'insieme di nodi che rende la variabile indipendente da tutti gli altri nodi nella rete. Questo permette quindi di ridurre il numero di variabili da considerare quando si fa inferenza su una particolare variabile.

5.4.1 Risultati con Markov Blanket

Dopo aver applicato la tecnica di Markov Blanket per ridurre il numero di features, è stata ottenuta la seguente struttura della rete bayesiana:

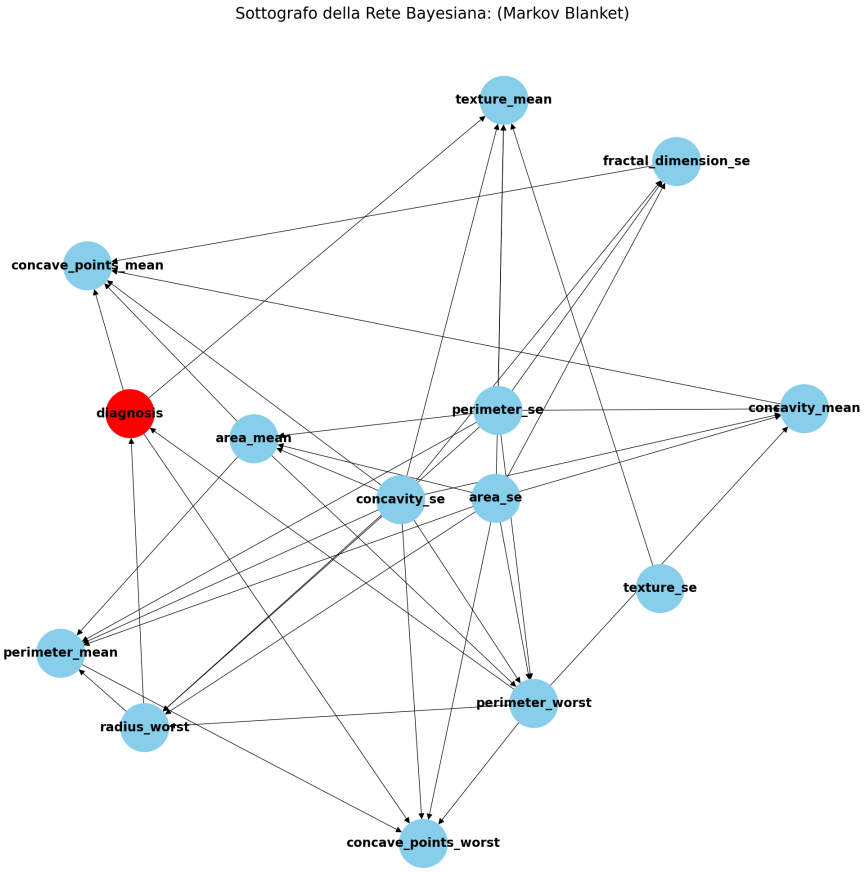


Figure 5.2: Rete Bayesiana appresa con Hill Climbing e Markov Blanket.

5.5 Conclusioni sulla Rete Bayesiana

5.5.1 Discretizzazione delle Variabili

Per poter utilizzare le reti bayesiane, è stato necessario discretizzare le variabili continue del dataset. Il dataset è stato discretizzato utilizzando `KBinsDiscretizer`. Ogni variabile continua è stata suddivisa in 3 intervalli di uguale ampiezza e a ciascun intervallo è stato assegnato un valore intero (0, 1, o 2).

Motivazione della Discretizzazione

La discretizzazione delle variabili continue è un passaggio fondamentale quando si lavora con reti bayesiane che utilizzano tabelle di probabilità condizionale (CPD). Gli algoritmi di apprendimento della struttura, come l'Hill Climbing con lo score K2, e gli algoritmi di inferenza sono progettati per operare su variabili discrete.

- **Calcolare le Probabilità**: Stimare le probabilità condizionali necessarie per costruire le CPD della rete.
- **Semplificare il Modello**: Ridurre la complessità del modello, rendendo l'apprendimento della struttura e l'inferenza computazionalmente più trattabili.

5.6 Risultati ottenuti

Sono stati generati un numero casuale di 400 campioni dalla rete bayesiana appresa. Questi campioni sono stati poi confrontati con il dataset originale per valutare la qualità della rete appresa. I risultati sono i seguenti:

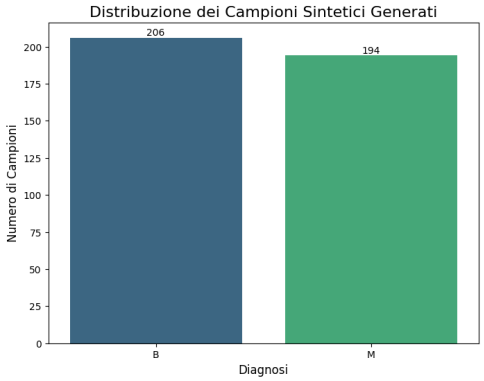


Figure 5.3: Risultati della rete bayesiana.

Oppure, vista in percentuale:

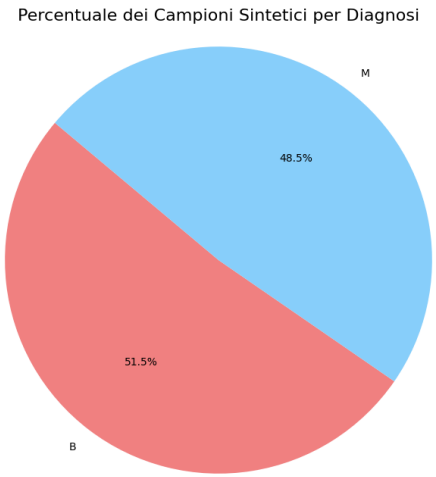


Figure 5.4: Risultati della rete bayesiana (in percentuale).

I risultati sono quasi bilanciati, in linea con il dataset originale (dopo l'applicazione di SMOTE).

Chapter 6

Conclusioni

In questo progetto, abbiamo esplorato il dataset BCWD per la classificazione delle masse tumorali al seno, utilizzando diverse tecniche di machine learning e reti bayesiane. I modelli di Random Forest e Gradient Boosting hanno mostrato le migliori performance, con un'accuracy molto alta. L'apprendimento della struttura con l'algoritmo di Hill Climbing ha permesso di scoprire relazioni interessanti tra le variabili, anche se la complessità della rete rende difficile l'interpretazione completa. L'uso di Markov Blanket ha aiutato a ridurre il numero di features, facilitando l'analisi della rete.

6.1 Lavori Futuri

Per migliorare ulteriormente questo progetto, si potrebbero considerare i seguenti lavori futuri:

- **Ottimizzazione degli Iperparametri**: Utilizzare tecniche più avanzate di ottimizzazione degli iperparametri, come Bayesian Optimization.
- **Altri Algoritmi di Apprendimento della Struttura**: Testare altri algoritmi di apprendimento della struttura, come Tabu Search o Genetic Algorithms.
- **Aggiornamento del Dataset**: Utilizzare dataset più recenti o più ampi per migliorare la generalizzazione dei modelli.
- **Feature Engineering**: Esplorare tecniche di feature engineering per creare nuove features che possano migliorare le performance dei modelli.
- **Utilizzo di Altri Modelli**: Testare altri modelli di machine learning, come Support Vector Machines o reti neurali profonde.