Predizione del cancro

Sara Gravante 886191

Alessio Tosato 886081

Luglio 2025

Progetto di Machine-Learning

Università degli studi di Milano-Bicocca
Corso di laurea Magistrale – Informatica

Indice

1 Introduzione

1.1 Informazioni sul dataset

Il dataset usato è disponibile al seguente link.

Le *features* (caratteristiche) sono calcolate da un'immagine digitalizzata di un ago aspirato (FNA) di una massa mammaria e descrivono le caratteristiche dei nuclei cellulari presenti nell'immagine.

Lo spazio tridimensionale è quello descritto in: [K. P. Bennett e O. L. Mangasarian: "Robust Linear Programming Discrimination of Two Linearly Inseparable Sets", Optimization Methods and Software 1, 1992, 23-34].

1.2 Descrizione del dataset

Dieci caratteristiche (*features*) a valore reale sono calcolate per ogni nucleo cellulare:

1. raggio (media delle distanze dal centro ai punti sul perimetro)
2. texture (deviazione standard dei valori in scala di grigi)
3. perimetro
4. area
5. levigatezza (variazione locale delle lunghezze del raggio)
6. compattezza ($perimetro^2/area - 1.0$)
7. concavità (gravità delle porzioni concave del contorno)
8. punti concavi (numero di porzioni concave del contorno)
9. simmetria
10. dimensione frattale ("approssimazione del litorale" - 1)

Per ogni immagine sono stati calcolati la *media*, l'*errore standard* e il valore "*peggiore*" o più grande (media dei tre valori più grandi) di queste caratteristiche, risultando in 30 caratteristiche. Ad esempio, il campo 2 è Raggio Medio, il campo 12 è Errore Standard del Raggio, il campo 22 è Raggio Peggiore.

Quindi, gli attributi risultanti saranno:

0. id
1. diagnosis
2. radius_mean
3. texture_mean
4. perimeter_mean

5. area_mean
6. smoothness_mean
7. compactness_mean
8. concavity_mean
9. concave points_mean
10. symmetry_mean
11. fractal_dimension_mean
12. radius_se
13. texture_se
14. perimeter_se
15. area_se
16. smoothness_se
17. compactness_se
18. concavity_se
19. concave points_se
20. symmetry_se
21. fractal_dimension_se
22. radius_worst
23. texture_worst
24. perimeter_worst
25. area_worst
26. smoothness_worst
27. compactness_worst
28. concavity_worst
29. concave points_worst
30. symmetry_worst
31. fractal_dimension_worst
32. Unnamed: 32

Tutti i valori delle caratteristiche sono ricodificati con quattro cifre significative.
Valori degli attributi mancanti: nessuno.
Distribuzione delle classi: 357 benigni, 212 maligni.

2 Analisi esplorativa dei dati

In questa sezione vengono esaminate le caratteristiche del dataset, al fine di comprenderne la struttura e le relazioni tra le variabili. L'obiettivo è identificare eventuali anomalie (outlier, valori mancanti), verificare lo sbilanciamento delle classi, osservare le distribuzioni delle feature e sondare correlazioni utili.

I risultati ottenuti guideranno le scelte di Preprocessing.

Il dataset presenta la seguente struttura:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean		concavity_worst	concave points_worst	symmetry_worst	fractal_dimension_worst	Unnamed: 32
0	842302	M	17.99	10.38	122.80	1001.0		0.7119	0.2654	0.4601	0.11890	NaN
1	842517	M	20.57	17.77	132.90	1326.0		0.2416	0.1860	0.2750	0.08902	NaN
2	84300903	M	19.69	21.25	130.00	1203.0		0.4504	0.2430	0.3613	0.08758	NaN
3	84348301	M	11.42	20.38	77.58	386.1		0.6869	0.2575	0.6638	0.17300	NaN
4	84358402	M	20.29	14.34	135.10	1297.0	***	0.4000	0.1625	0.2364	0.07678	NaN

Figura 1: Prime cinque righe del dataset visualizzate tramite `df.head()`

2.1 Valori nulli

Tramite le funzioni `.info()` e `.describe().T`, notiamo che il 32esimo attributo "`Unnamed: 32`" contiene solo valori nulli `NaN`.

Questo è confermato dalla funzione `.isnull().sum()`.

```
26  smoothness_worst         569 non-null    float64
27  compactness_worst        569 non-null    float64
28  concavity_worst          569 non-null    float64
29  concave points_worst     569 non-null    float64
30  symmetry_worst           569 non-null    float64
31  fractal_dimension_worst  569 non-null    float64
32  Unnamed: 32              0 non-null      float64
```

Figura 2: Output dell'esecuzione del comando `.info()`

smoothness_worst	0
compactness_worst	0
concavity_worst	0
concave points_worst	0
symmetry_worst	0
fractal_dimension_worst	0
Unnamed: 32	569

Figura 3: Output dell'esecuzione del comando `.isnull().sum()`

2.2 Bilanciamento target

Verifichiamo ora se il dataset sia bilanciato, controllando la distribuzione della variabile target "`diagnosis`":

- "B" si riferisce alla diagnosi di tumore benigno
- "M" si riferisce alla diagnosi di tumore maligno.

`diagnosis`	proportion
B	0.627417
M	0.372583

Tabella 1: Proporzioni delle classi nella variabile `diagnosis`

Dato che la variabile target è un po' sbilanciata, andranno bilanciate le classi durante il training dei modelli, ad esempio aggiungendo il parametro `class_weight='balanced'` alle definizioni dei classificatori.

2.3 Matrice di Correlazione

La **matrice di correlazione** è uno strumento fondamentale nell'analisi esplorativa dei dati, poiché consente di identificare relazioni lineari tra variabili. Ogni elemento della matrice rappresenta il *coefficiente di correlazione* tra due variabili, indicando la forza e la direzione della loro relazione.

Nella visualizzazione della matrice in Figura 4 sono stati esclusi gli attributi: `id`, `diagnosis` e `Unnamed:32`.

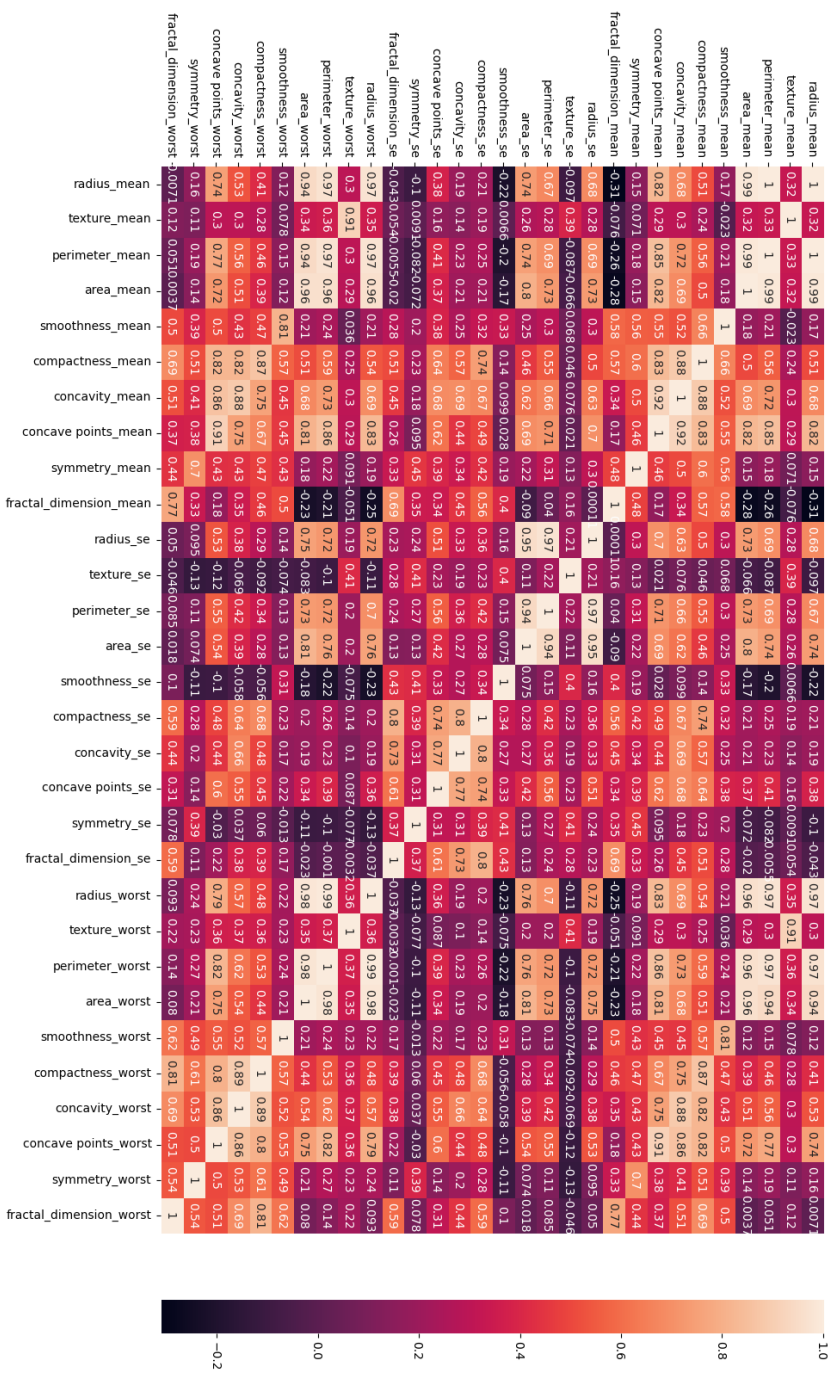

Figura 4: Matrice di correlazione: più è chiara la tonalità, più è alto il grado di correlazione

2.4 Distribuzione delle caratteristiche

Vengono ora mostrati istogrammi con curve di densità: questa visualizzazione aiuta a comprendere la distribuzione delle variabili e a individuare eventuali asimmetrie o anomalie nei dati.

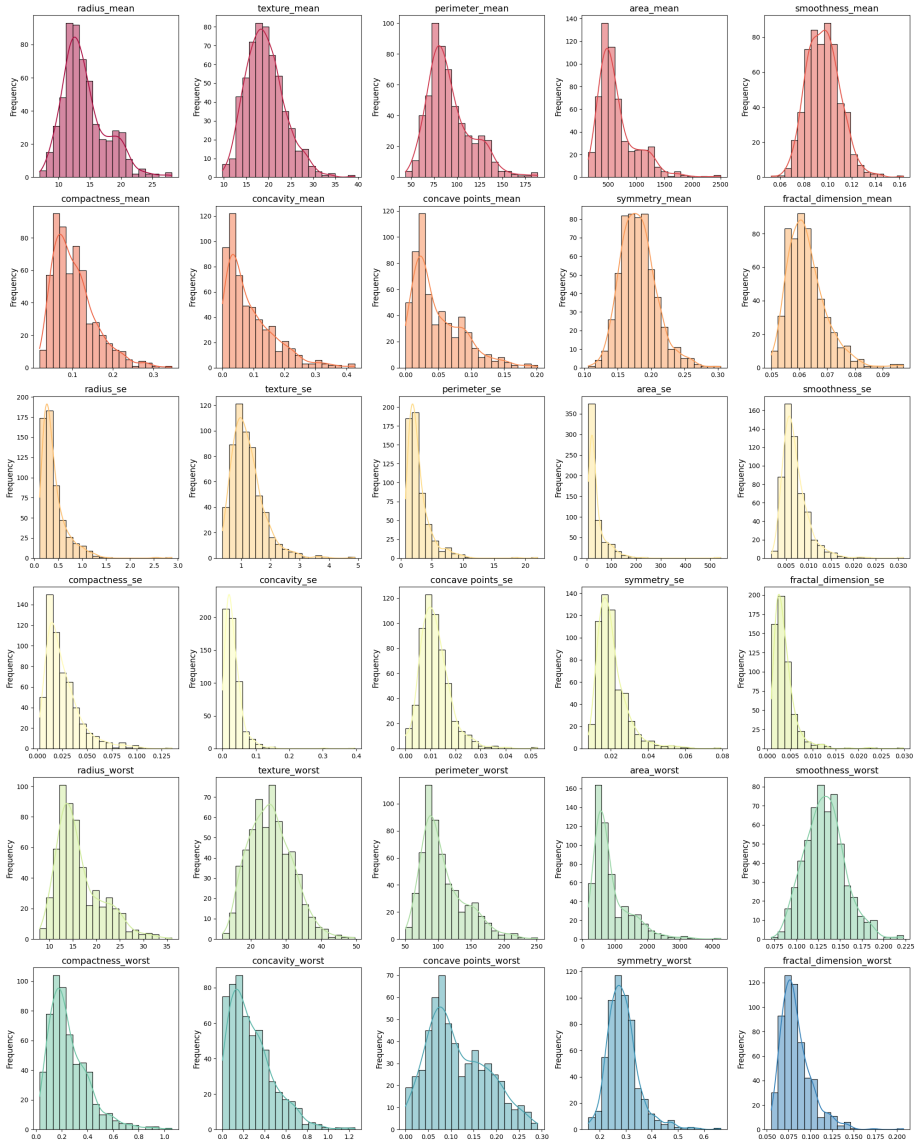

Figura 5: Istogrammi delle caratteristiche, eccetto `id`, `diagnosis` e `Unnamed:32`

2.5 Boxplot delle caratteristiche

Sono stati generati i boxplot, grafici che aiutano a individuare **outlier**, distribuzioni asimmetriche e la variabilità delle caratteristiche del dataset.

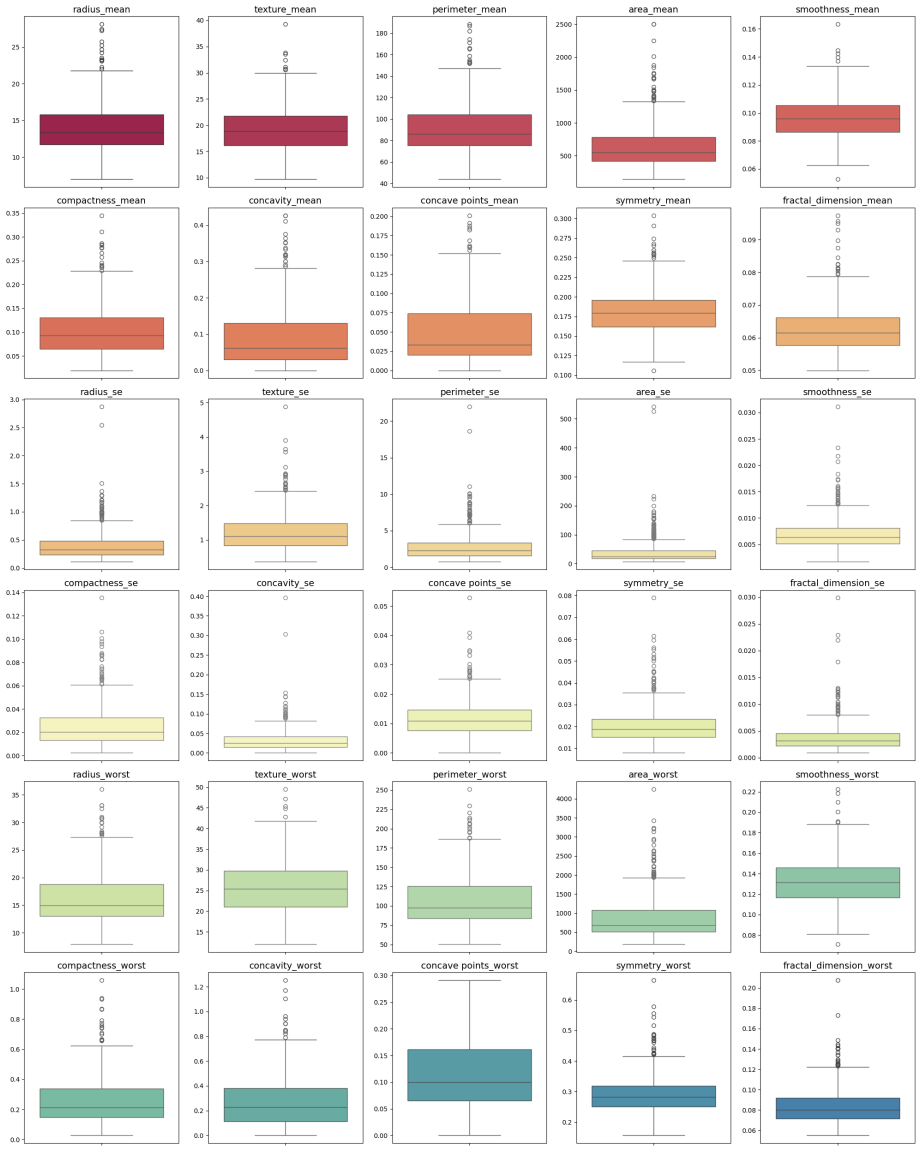

Figura 6: Boxplot delle caratteristiche, eccetto `id`, `diagnosis` e `Unnamed:32`

3 Preprocessing

3.1 Eliminazione valori nulli

Come osservato durante l'Analisi Esplorativa dei dati, è presente una colonna composta da solo valori nulli (`NaN`): la si elimina tramite il comando `.drop()`.

```
df = df.drop(['Unnamed: 32'], axis=1)
```

3.2 Codifica dati categorici

Gli algoritmi di machine learning possono leggere solo valori numerici, quindi è essenziale codificare le caratteristiche categoriche in valori numerici.

Attraverso l'utilizzo della funzione `LabelEncoder()`, "maligno" viene codificato come 1 e "benigno" come 0.

```
LEncoder = LabelEncoder()
df['diagnosis'] = LEncoder.fit_transform(df['diagnosis'])
```

3.3 Suddivisione train/set

Viene selezionato un insieme di dati di addestramento da fornire all'algoritmo di Machine Learning, in modo da garantire che l'addestramento dell'algoritmo di classificazione possa essere generalizzato efficacemente a nuovi dati.

Con la funzione `train_test_split` dal modulo `sklearn.model_selection` si divide il dataset negli insiemi di **training** e **test**: il 20% dei dati viene usato per il test, mentre il restante 80% è usato per l'addestramento.

```
y = df['diagnosis']
X = df.drop(columns=['id', 'diagnosis'])

X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.2, random_state=42)
```

3.4 Standardizzazione dei dati

Scalare le feature è una fase del preprocessing dei dati applicata alle variabili indipendenti e serve a normalizzare i dati entro un certo intervallo.

Al dataset viene applicata la **standardizzazione** tramite `StandardScaler`, una tecnica che trasforma le variabili in modo da avere media pari a 0 e deviazione standard pari a 1. Questo tipo di trasformazione è particolarmente indicato per algoritmi sensibili alla scala dei dati.

Viene applicato il metodo `.fit_transform()` sul training set e `.transform()` sul test set per evitare **data leakage**.

```
sc = StandardScaler()

X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Tale procedura è essenziale per garantire prestazioni ottimali in modelli che assumono una distribuzione standardizzata delle feature, come SVM.

4 Modelli

I modelli di classificazione supervisionata utilizzati in questo progetto sono **SVC** (Support Vector Classifier), appartenente alla famiglia degli **SVM** (Support Vector Machines), e l'albero decisionale (**Decision Tree**).

Viene eseguita una grid search (`GridSearchCV`) per ciascun modello, al fine di trovare la combinazione di parametri ottimale, con la quale viene effettuata anche una **cross-validation** (in questo caso, a 5 fold).

Inoltre, viene utilizzato il parametro `class_weight='balanced'` per bilanciare i pesi associati alle classi, utile in casi come questo in cui il dataset presenta uno sbilanciamento, come evidenziato nell'Analisi Esplorativa dei dati.

4.1 SVC

L'**SVC** è particolarmente efficace in problemi di classificazione binaria e riesce a trovare l'iperpiano ottimale che separa le classi massimizzando il margine tra esse. È adatto a dataset piccoli e ben separabili, anche complessi, e può essere potenziato con il kernel trick per gestire dati non linearmente separabili.

```
svc = SVC(probability=True, class_weight='balanced')
```

Parametri del Support Vector Classifier:

- `C` – Parametro di regolarizzazione: controlla il compromesso tra margine ampio e classificazione corretta dei punti di addestramento.

  Valori usati: `0.1, 1, 10, 100`.

- `kernel` – Tipo di kernel.

  Valori usati: `'linear'`, `'poly'`, `'rbf'`.

- `gamma` – Coefficiente del kernel: influenza quanto ogni punto singolo impatta sulla curva di decisione.

  Valori usati: `'scale'`, `'auto'`, `1`, `0.1`, `0.01`.

- `n_jobs=-1`: Indica al processo di utilizzare tutti i core disponibili per velocizzare.

```
param_grid_svc = {
    'C': [0.1, 1, 10, 100],
    'kernel': ['linear', 'rbf', 'poly'],
    'gamma': ['scale', 'auto', 1, 0.1, 0.01],
}

grid_search_svc = GridSearchCV(svc, param_grid_svc, cv=5,
    n_jobs=-1)
grid_search_svc.fit(X_train, y_train)
```

Output dell'esecuzione del codice di training del modello `SVC()` tramite `GridSearchCV()`:

```
Parametri migliori: {'C': 1, 'gamma': 'auto', 'kernel': 'rbf'}
Media punteggi CV=5: 0.9802197802197803}

SVC Train Accuracy: 0.9846
SVC Test Accuracy: 0.9649
```

4.2 Decision Tree

Il **Decision Tree** costruisce un modello sotto forma di struttura ad albero, in cui ogni nodo rappresenta una decisione basata su una feature del dataset. Questo modello è facilmente interpretabile, ma può essere soggetto a overfitting se non regolato correttamente.

```
dt = DecisionTreeClassifier(class_weight='balanced')
```

Iperparametri del Decision Tree Classifier:

- `criterion` – Criterio di divisione: `'gini'` oppure `'entropy'`.
- `max_depth` – Massima profondità dell'albero: controlla quanto l'albero può crescere.

  Se troppo grande, può portare overfitting; se troppo piccolo, underfitting.

  Valori usati: `2, 3, 4, 5, 6`.
- `min_samples_split` – Minimo numero di campioni per dividere un nodo: evita che l'albero cresca su rumore.

  Valori usati: `5, 10, 15`.
- `min_samples_leaf` – Minimo numero di campioni in una foglia.

  Valori usati: `2, 3, 4, 5, 6`.
- `max_features`: Numero di features che il decision tree considera a ogni split. Il valore giusto aiuta a prevenire overfitting e migliorare la generalizzazione.

  Valori usati: `'sqrt'`, `'log2'`.

```
param_grid_dt = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [2, 3, 4, 5, 6],
    'min_samples_leaf': [2, 3, 4, 5, 6],
    'min_samples_split': [5, 10, 15],
    'max_features': ['sqrt', 'log2']
}

grid_search_dt = GridSearchCV(dt, param_grid_dt, cv=5)
grid_search_dt.fit(X_train, y_train)
```

Durante le prime esecuzioni di `GridSearchCV()` applicate al Classificatore Decision Tree, si è riscontrata instabilità nella selezione dei parametri ottimali.

Questa variabilità era amplificata dalla natura flessibile del modello e dalle modeste dimensioni del dataset (569 istanze). Inoltre, l'assenza di un controllo sulla casualità interna contribuiva a generare alberi differenti ad ogni esecuzione.

Per aumentare la coerenza e la ripetibilità dei risultati, si è scelto di fissare manualmente il parametro `random_state` direttamente nella definizione del classificatore, così da stabilizzare il processo di training e rendere più affidabili le valutazioni ottenute tramite cross-validation.

Quindi la nuova definizione del `DecisionTreeClassifier()` diventa:

```
dt = DecisionTreeClassifier(class_weight='balanced',
    random_state=42)
```

Output dell'esecuzione del codice di training del modello `DecisionTreeClassifier()` tramite `GridSearchCV()`:

```
Parametri migliori: {'criterion': 'gini', 'max_depth': 4,
    'max_features': 'log2', 'min_samples_leaf': 6,
    'min_samples_split': 2}
Media punteggi CV=5: 0.9516483516483516

DT Train Accuracy: 0.967
DT Test Accuracy: 0.9737
```

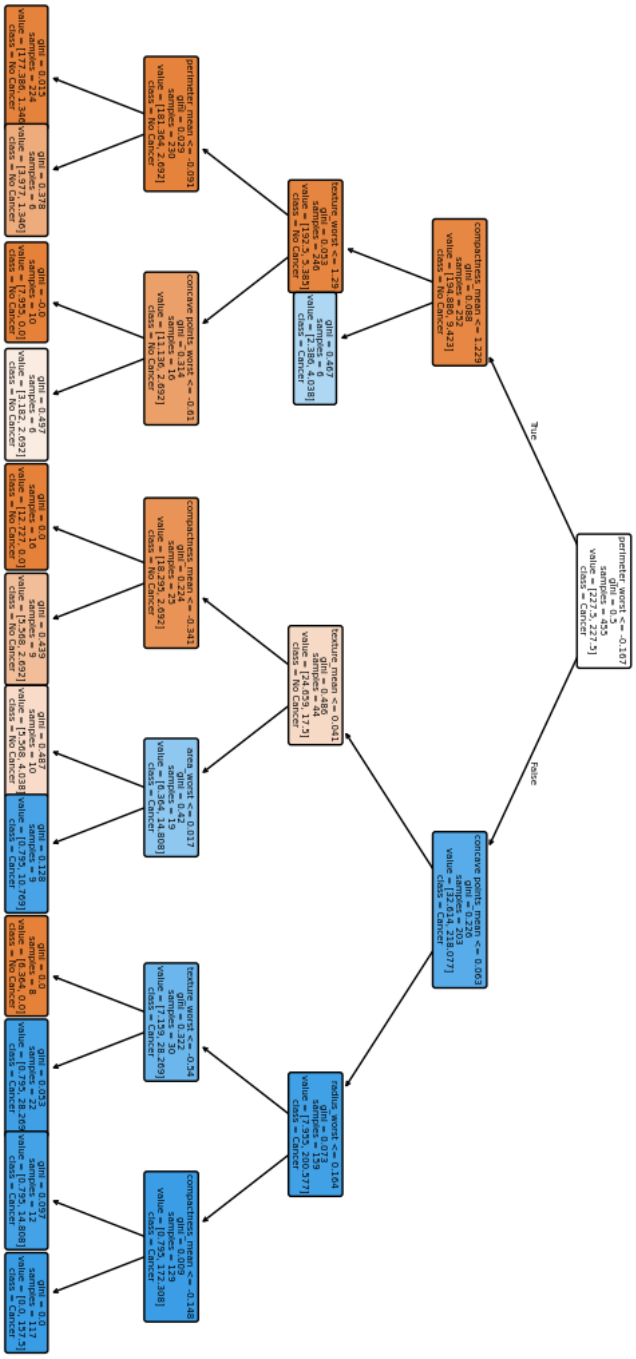

Figura 7: Rappresentazione grafica della struttura ad albero risultante dall'esecuzione del `DecisionTreeClassifier()` sul dataset

5 Valutazione modelli

Esistono numerose metriche che ci aiutano a valutare le prestazioni dei modelli di machine learning:

- Matrice di confusione
- Report di classificazione:
  - Accuratezza
  - Precisione
  - Recall
  - F1-score
- Punteggio ROC AUC
- Area sotto la curva (AUC)

Ora, vediamo un confronto delle prestazioni di ciascun classificatore.

5.1 Matrice di confusione

Una **matrice di confusione** è una tabella nella quale ogni riga rappresenta le istanze della classe *reale*, mentre ogni colonna rappresenta le istanze della classe *predetta*.

Nel caso di un classificatore binario, la classe "vera" è solitamente etichettata con 1, mentre la classe "falsa" è etichettata con 0.

- **Vero Positivo** (**TP – True Positive**): un'osservazione della classe positiva (1) è correttamente classificata come positiva (1) dal modello.
- **Falso Positivo** (**FP – False Positive**): un'osservazione della classe negativa (0) è erroneamente classificata come positiva (1).
- **Vero Negativo** (**TN – True Negative**): un'osservazione della classe negativa (0) è correttamente classificata come negativa (0).
- **Falso Negativo** (**FN – False Negative**): un'osservazione della classe positiva (1) è erroneamente classificata come negativa (0).

In questo caso, ogni volta che il modello predice **Yes** (0), indica l'*assenza* (classe negativa) di cellule tumorali (**Healthy**), mentre quando predice **No** (1), indica la *presenza* (classe positiva) di cellule tumorali (**Cancer**).

Visualizziamo ora la matrice di confusione per osservare quanto sono accurati i risultati ottenuti.

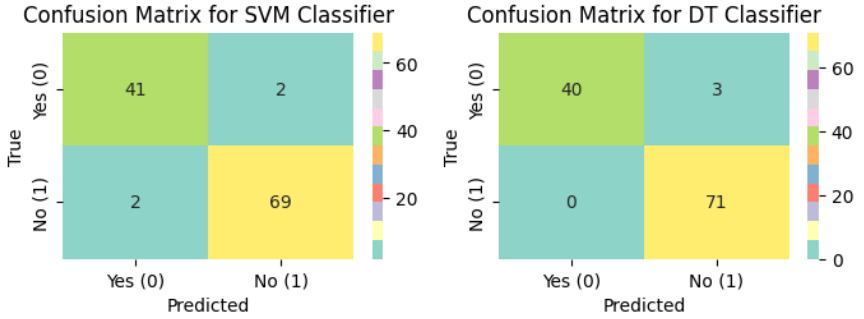

Figura 8: Matrice di confusione

Come possiamo vedere dalla Figura 8:

- **Vero Positivo** (TP): valori che il modello ha predetto come Yes (Healthy) e che sono effettivamente Yes (Healthy).
- **Vero Negativo** (TN): valori che il modello ha predetto come No (Cancer) e che sono effettivamente No (Cancer).
- **Falso Positivo** (FP): valori che il modello ha predetto come Yes (Healthy) ma che in realtà sono No (Cancer).
- **Falso Negativo** (FN): valori che il modello ha predetto come No (Cancer) ma che in realtà sono Yes (Healthy).

5.2 Metriche a confronto

Ora verrà analizzata ciascuna metrica per un confronto più dettagliato.

Accuracy è la misura di performance più intuitiva e rappresenta semplicemente il rapporto tra le osservazioni predette correttamente e il totale delle osservazioni.

$$Accuracy = \frac{TruePositives + TrueNegatives}{TP + TN + FP + FN}$$

Precision è il rapporto tra le osservazioni positive correttamente predette e il totale delle osservazioni predette come positive.

$$Precision = \frac{TruePositives}{TruePositives + FalsePositives}$$

Recall (Sensitivity) è il rapporto tra le istanze positive correttamente rilevate dal classificatore e tutte le osservazioni appartenenti alla classe positiva reale.

$$Recall = \frac{TruePositives}{TruePositives + FalseNegatives}$$

F1-Score è la media armonica tra precision e recall.

$$F1 - score = \frac{Precision \cdot Recall}{Precision + Recall}$$

5.2.1 Classification Report

Il **classification report** permette di visualizzare una panoramica delle metriche di ciascun modello.

	precision	**recall**	**f1-score**	**support**
0	0.97	0.97	0.97	71
1	0.95	0.95	0.95	43
accuracy			0.96	114
macro avg	0.96	0.96	0.96	114
weighted avg	0.96	0.96	0.96	114

Tabella 2: Classification Report del modello SVC

	precision	**recall**	**f1-score**	**support**
0	0.96	1.00	0.98	71
1	1.00	0.93	0.96	43
accuracy			0.97	114
macro avg	0.98	0.97	0.97	114
weighted avg	0.97	0.97	0.97	114

Tabella 3: Classification Report del modello DecisionTreeClassifier

Si può osservare che il modello **DecisionTreeClassifier** mostra $precision = 1$ e $recall = 0.93$, risultati apparentemente eccellenti: l'alta *precisione* suggerisce l'assenza di falsi positivi, mentre il *recall* dimostra una buona capacità di individuare i veri positivi.

Tuttavia, considerando le **ridotte dimensioni** del dataset, è importante interpretare questi numeri con cautela: il rischio di **overfitting** è elevato, e piccole variazioni nei dati possono influire significativamente sulla struttura dell'albero.

5.3 Riepilogo grafico

Di seguito vengono mostrati dei bar plot che riepilogano il confronto tra le performance dei due modelli.

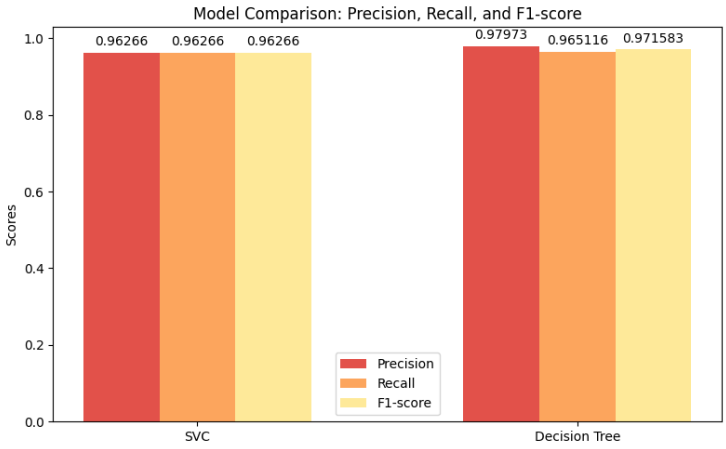

Figura 9: Confronto delle metriche `precision`, `recall` e `F1-score`

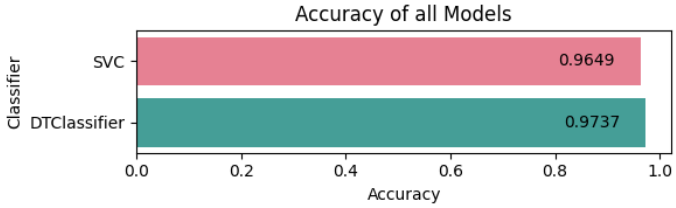

Figura 10: Confronto della metrica `accuracy`

Dai valori riportati in tabella si osserva che il Decision Tree Classifier (*accuracy* = 0.9737) **supera leggermente** l'SVC (*accuracy* = 0.9649), risultando il modello con le prestazioni complessivamente migliori sul dataset considerato.

5.4 ROC Curve

La **curva ROC** mostra il compromesso tra sensibilità (TPR) e specificità (1 – FPR). Questi due valori rappresentano:

True Positive Rate / Recall / Sensibilità Con quale frequenza il modello predice "Yes" (Healthy) quando in realtà è "Yes" (Healthy)?

$$\textbf{TPR} = \frac{TP}{TP+FP} = \frac{71}{71+2} = 0,97$$

False Positive Rate Con quale frequenza il modello predice "Yes" (Healthy) quando in realtà è "No" (Cancer)?

$$\textbf{FPR} = \frac{FP}{FP+TN} = \frac{2}{2+4} = 0,04$$

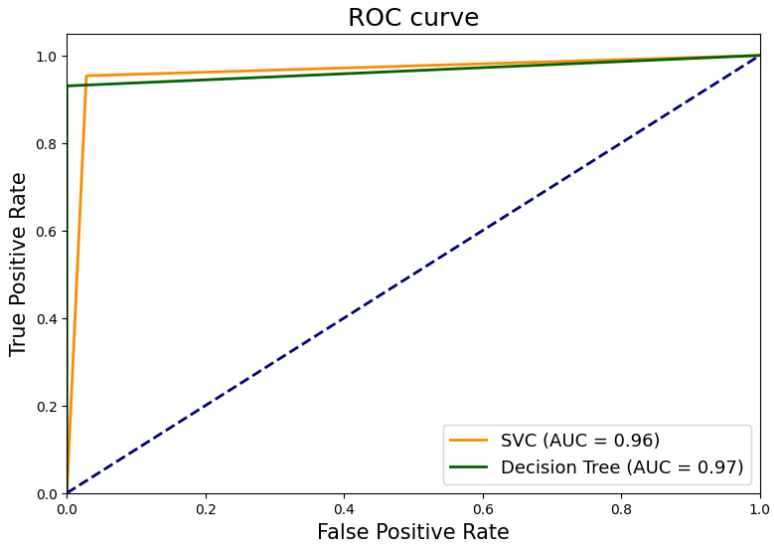

Figura 11: Confronto dei modelli tramite Curva ROC

Il classificatore Decision Tree ($AUC = 0.9651$) presenta un'area sotto la curva ROC **leggermente superiore** rispetto a quella dell'SVC ($AUC = 0.9627$), suggerendo prestazioni complessivamente migliori, seppur di misura contenuta.

6 Conclusioni

L'analisi condotta ha mostrato che entrambi i modelli, Support Vector Classifier (SVC) e Decision Tree Classifier (DT), raggiungono **performance molto elevate** sul dataset utilizzato, con accuratezze rispettivamente pari a 96.49% e 97.37% dopo l'ottimizzazione tramite GridSearch. Anche in termini di AUC, entrambi i modelli mostrano valori eccellenti, con il DT leggermente superiore (AUC = 0.9651) rispetto all'SVC (AUC = 0.9627). In particolare, il modello **Decision Tree** ha evidenziato una leggera superiorità nella classificazione dei casi di tumore maligno, mostrando una recall perfetta sulla classe "cancer", rendendolo potenzialmente più adatto in contesti clinici dove è cruciale ridurre al minimo i falsi negativi.

Tuttavia, questi risultati molto elevati sollevano un possibile campanello d'allarme: la probabilità di **overfitting**. Il dataset contiene infatti solo 569 istanze, un numero relativamente contenuto per addestrare modelli complessi. Il rischio è che i modelli si adattino troppo bene ai dati di training, perdendo la **capacità di generalizzare** correttamente su dati nuovi.

Per questo motivo, sarebbe opportuno in futuro valutare le performance su un dataset più ampio e diversificato, oppure condurre un'analisi di apprendimento più approfondita, includendo ad esempio una curva di apprendimento o una validazione incrociata più robusta.