

BREAST-CANCER

Progetto per “Ingegneria della Conoscenza”

AA 2024/2025

LINK REPOSITORY GITHUB:

<https://github.com/rafflog01/ICON>

REALIZZATO DA:

LOGLISCI RAFFAELE, 757060 , r.loglisci6@studenti.uniba.it

INDICE

#0 INTRODUZIONE	3
#1 DATASET	4
#1.1 DESCRIZIONE DATASET (DOMINIO)	4
#1.2 OSSERVAZIONE GRAFICA DEI DATI	6
#2 ONTOLOGIA	7
#2.1 ANALISI DOMINIO	7
#2.2 SOFTWARE PER LA REALIZZAZIONE DELL'ONTOLOGIA	7
#2.2.1 CLASSI	8
#2.2.2 OBJECT PROPERTY	9
#2.2.3 DATA PROPERTY	9
#2.2.4 INDIVIDUALS	10
#3 QUERY	10
#3.1 DL Query	10
#3.1 OwlReady2	11
#4 APPRENDIMENTO SUPERVISIONATO	12
#4.1 DECISIONI DI PROGETTO	12
#4.2 METRICHE	13
#4.2.1 DI VALIDAZIONE	13
#4.2.2 DI VALUTAZIONE	13
#4.3 SELEZIONE DELLE FEATURE	14
#4.4 PREPROCESSING DEI DATI	14
#4.4.1 DUMMIFICATION & LABEL ENCODING	14
#4.4.2 SMOTE	15
#4.4.3 STANDARDIZZAZIONE	15
#4.5 DIVISIONE DEI DATI	16
#4.6 K-NEAREST NEIGHBORS (K-NN)	17
#4.6.1 DECISIONI DI PROGETTO	17
#4.6.2 OTTIMIZZAZIONE DEL VALORE 'K'	17
#4.6.3 ADDESTRAMENTO	18
#4.6.4 PREDIZIONE	19
#4.6.5 VALUTAZIONE FINALE	19
#4.7 RANDOM FOREST	23
#4.7.1 DECISIONI DI PROGETTO	23
#4.7.2 OTTIMIZZAZIONE PARAMETRI DEL CLASSIFICATORE	23
#4.7.3 VALUTAZIONE FINALE	25
#4.8 SUPPORT VECTOR MACHINES (SVM)	27
#4.8.1. DECISIONI DI PROGETTO	27
#4.8.2 OTTIMIZZAZIONE PARAMETRO GAMMA	27
#4.8.3 VALUTAZIONE FINALE	29
#4.9 NEURAL NETWORK	31
#4.9.1 DECISIONI DI PROGETTO	31
#4.9.2 CREAZIONE MODELLO E ADDESTRAMENTO	31
#4.9.3 VALUTAZIONE FINALE	33
#4.10 CONCLUSIONI	34
#5 APPRENDIMENTO NON SUPERVISIONATO: CLUSTERING	36
#5.1 DECISIONI PROGETTUALI	36
#5.2 K-MEANS	37
#5.3 VALUTAZIONE FINALE DEL MODELLO	38
#6 CONCLUSIONE	40
#6.1 POSSIBILI SVILUPPI	40

#0 INTRODUZIONE

Il presente lavoro ha l'obiettivo di predire la diagnosi, di un cancro al seno, maligno o benigno sfruttando un DataSet disponibile online.

A tal proposito:

- È stata modellata un'[Ontologia](#) di riferimento che offre una rappresentazione formale e concettualizzata della realtà presa in esame, affinché possa venire interrogata tramite [Query](#).
- Sono state utilizzate tecniche di [Apprendimento Supervisionato](#) e [Non Supervisionato](#).

PROBLEMA IN QUESTIONE:

Il cancro al seno è il tumore più comune tra le donne al mondo. Rappresenta il 25% di tutti i casi di cancro e ha colpito oltre 2,1 milioni di persone solo nel 2015. Inizia quando le cellule del seno iniziano a crescere in modo incontrollato. Queste cellule di solito formano tumori visibili ai raggi X o palpabili come noduli nella zona del seno. La sfida principale per la sua individuazione è come classificare i tumori in maligni (cancerosi) e benigni (non cancerosi). Un tale DataSet può essere utilizzato per addestrare o utilizzare modelli e algoritmi per effettuare delle diagnosi.

MATERIALE UTILIZZATO:

LINGUAGGIO DI PROGRAMMAZIONE

- Python: <https://www.python.org/downloads/>

PROVIDER DEL DATASET

- Kaggle: <https://www.kaggle.com/datasets/>

APP PER REALIZZAZIONE DELL'ONTOLOGIA

- Protégé: <https://protege.stanford.edu/>

LIBRERIE PYTHON

- Scikit-learn: <https://scikit-learn.org/>
- Pandas: <https://pandas.pydata.org/>
- Seaborn: <https://seaborn.pydata.org/>
- OwlReady2: <https://owlready2.readthedocs.io/en/v0.48/index.html>

#1 DATASET

Propongo l'utilizzo di un dataset relativo allo screening delle diagnosi di tumore al seno, reperito al seguente indirizzo: <https://www.kaggle.com/datasets/yasserh/breast-cancer-dataset?resource=download>.

#1.1 DESCRIZIONE DATASET (DOMINIO)

FEATURE: 32

CAMPIONE: 570 istanze

FEATURE	FEATURE ROLE	DOMAIN TYPE	DESCRIPTION
id	Feature	Integer (0-800000)	Identificativo unico per riconoscere i pazienti.
Raggio Medio	Feature	Continuous (0-3000)	Media del raggio delle cellule tumorali.
Trama Media	Feature	Continuous (0-3000)	Media della variazione dell'intensità della texture nelle cellule.
Perimetro Medio	Feature	Continuous (0-3000)	Media della misura del perimetro delle cellule.
Area Media	Feature	Continuous (0-3000)	Media dell'area delle cellule tumorali.
Levigatezza Media	Feature	Continuous (0-3000)	Media della variazione locale dei contorni delle cellule.
Compattezza Media	Feature	Continuous (0-3000)	Media del rapporto tra perimetro ² e area delle cellule.
Concavità Media	Feature	Continuous (0-3000)	Media del grado di concavità nei contorni delle cellule.
Punti Concavi Medi	Feature	Continuous (0-3000)	Media del numero di punti concavi nei contorni delle cellule.
Simmetria Media	Feature	Continuous (0-3000)	Media della simmetria della forma delle cellule.
Dimensione Frattale Media	Feature	Continuous (0-3000)	Media della complessità dei contorni delle cellule.
Raggio Errore Standard	Feature	Continuous (0-3000)	Variazione del raggio delle cellule (errore standard).
Trama Errore Standard	Feature	Continuous (0-3000)	Variazione della texture delle cellule (errore standard).
Perimetro Errore Standard	Feature	Continuous (0-3000)	Variazione del perimetro delle cellule (errore standard).

Area Errore Standard	Feature	Continuous (0-3000)	Variazione dell'area delle cellule (errore standard).
Levigatezza Errore Standard	Feature	Continuous (0-3000)	Variazione della levigatezza delle cellule (errore standard).
Compattezza Errore Standard	Feature	Continuous (0-3000)	Variazione della compattezza delle cellule (errore standard).
Concavità Errore Standard	Feature	Continuous (0-3000)	Variazione della concavità delle cellule (errore standard).
Punti Concavi Errore Standard	Feature	Continuous (0-3000)	Variazione dei punti concavi delle cellule (errore standard).
Simmetria Errore Standard	Feature	Continuous (0-3000)	Variazione della simmetria delle cellule (errore standard).
Dimensione Frattale Errore Standard	Feature	Continuous (0-3000)	Variazione della dimensione frattale delle cellule (errore standard).
Raggio Peggior	Feature	Continuous (0-3000)	Peggior valore osservato del raggio delle cellule.
Trama Peggior	Feature	Continuous (0-3000)	Peggior valore osservato della trama delle cellule.
Perimetro Peggior	Feature	Continuous (0-3000)	Peggior valore osservato del perimetro delle cellule.
Area Peggior	Feature	Continuous (0-3000)	Peggior valore osservato dell'area delle cellule.
Levigatezza Peggior	Feature	Continuous (0-3000)	Peggior valore osservato della levigatezza delle cellule.
Compattezza Peggior	Feature	Continuous (0-3000)	Peggior valore osservato della compattezza delle cellule.
Concavità Peggior	Feature	Continuous (0-3000)	Peggior valore osservato della concavità delle cellule.
Punti Concavi Peggiori	Feature	Continuous (0-3000)	Peggior valore osservato dei punti concavi delle cellule.
Simmetria Peggior	Feature	Continuous (0-3000)	Peggior valore osservato della simmetria delle cellule.
Dimensione Frattale Peggior	Feature	Continuous (0-3000)	Peggior valore osservato della dimensione frattale delle cellule.
Diagnosi	Target	Binary (0 / 1)	Tipo di diagnosi del tumore. (0 = Benigno / 1 = Maligno)

#1.2 OSSERVAZIONE GRAFICA DEI DATI

Ho effettuato un'analisi visiva dei dati con l'**obiettivo** di individuare eventuali **correlazioni** tra le **variabili** e comprendere in che misura **influenzano** la diagnosi.

Ho dovuto mappare la feature **diagnosis** in numerico così da permetterne la visualizzazione.

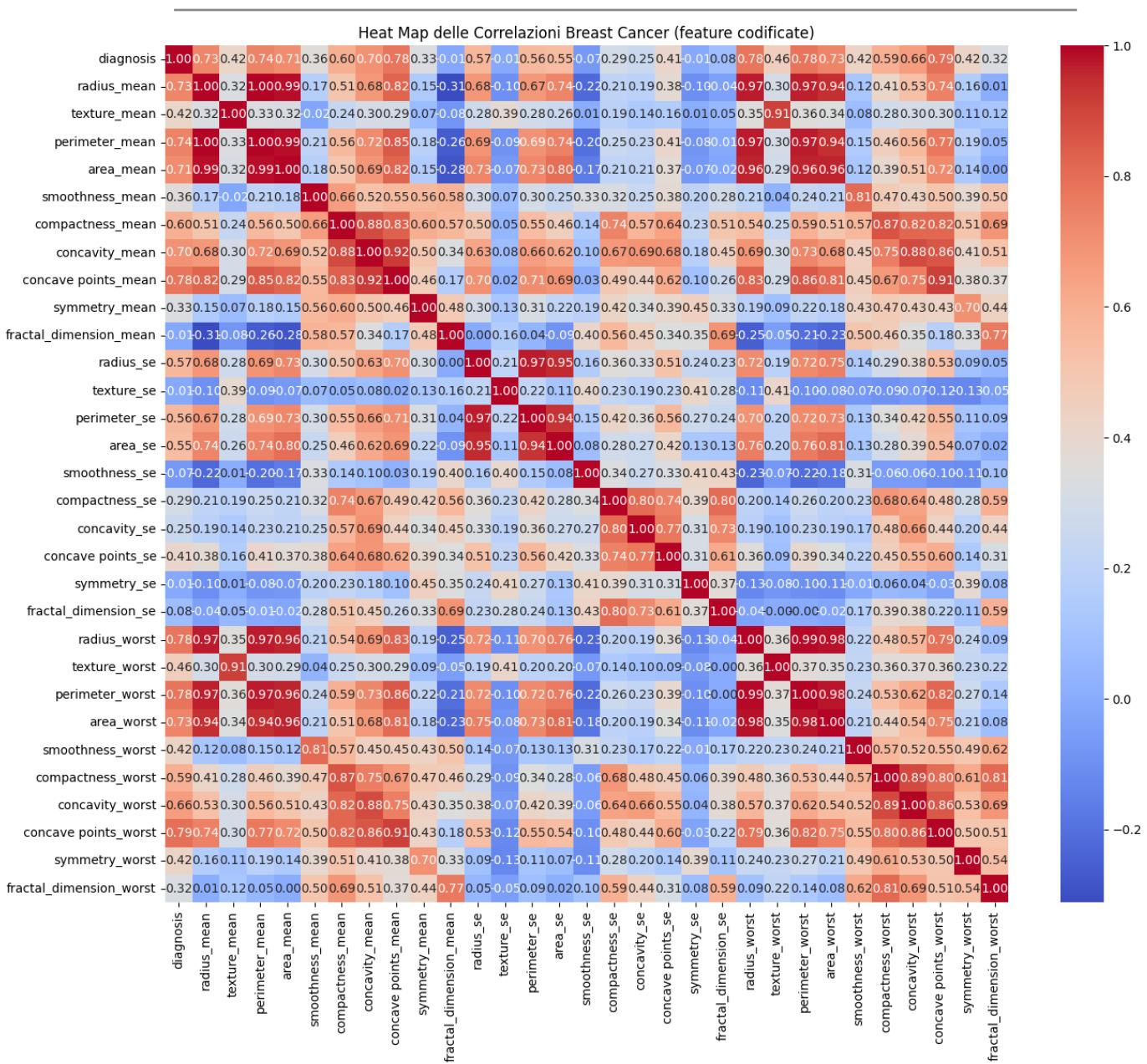
I **numeri** visibili su ogni cella della heatMap sono i **valori di correlazione** con un valore vicino ad **1** abbiamo una **forte correlazione**, al crescere di una variabile cresce anche l'altra, vicino allo **0 nessuna correlazione** mentre vicina a **-1** abbiamo una **forte correlazione negativa**, al crescere di una variabile l'altra diminuisce.

Di seguito una heatmap generata tramite Python a supporto di questa analisi:

MATRICE DI CORRELAZIONE

Una **matrice di correlazione** è un insieme di valori numerici che esprimono **relazioni** tra le **feature** del DataSet.

La visualizzazione di questa matrice viene comunemente realizzata tramite una **heatmap**, facilmente generabile con la libreria **Seaborn** in Python, che consente di interpretare visivamente le correlazioni attraverso una scala di colori.



#2 ONTOLOGIA

Un'Ontologia rappresenta in modo strutturato la **conoscenza** relativa a un determinato dominio. In termini semplici, si tratta di un sistema per **organizzare** e definire **concetti** e **relazioni** tra essi in maniera chiara e formale.

L'analisi del dominio consente di individuare i **concetti fondamentali**, le relazioni che li legano e le proprietà che li descrivono.

Una volta identificate, queste informazioni possono essere **formalizzate** utilizzando un linguaggio apposito, come **OWL (Ontology Web Language)**.

#2.1 ANALISI DOMINIO

Ho deciso di dividere gli attributi del DataSet in:

- “**ciò che deve essere rappresentato**”:
Gli attributi fondamentali e cruciali che devono essere rappresentati per catturare le informazioni essenziali dal DataSet.
 - Nel nostro contesto si parla di Pazienti, Diagnosi e Cancro.
- “**ciò che caratterizza ciò che deve essere rappresentato**”:
Le proprietà delle entità rappresentate.
 - Nel caso del Cancro, ad esempio, possono essere area, perimetro, etc.

#2.2 SOFTWARE PER LA REALIZZAZIONE DELL'ONTOLOGIA

L'ontologia è stata sviluppata utilizzando **Protégé**, un software open-source ampiamente utilizzato per la **creazione e gestione di ontologie**.

Protégé consente di definire concetti, classi e relazioni in modo strutturato, supportando standard come **OWL** (Web Ontology Language) per la rappresentazione formale della conoscenza.

#2.2.1 CLASSI

Ho deciso di rappresentare come **Classi/Entity** dell'ontologia:

- **Patient:**

Rappresenta l'insieme dei pazienti sottoposti al test.

A questa classe ho associato un'unica proprietà:

- id: identificativo unico per il riconoscimento

- **Diagnosis:**

Sottoclasse di Patient.

Rappresenta la diagnosi del paziente riguardante il cancro al seno che ha come proprietà:

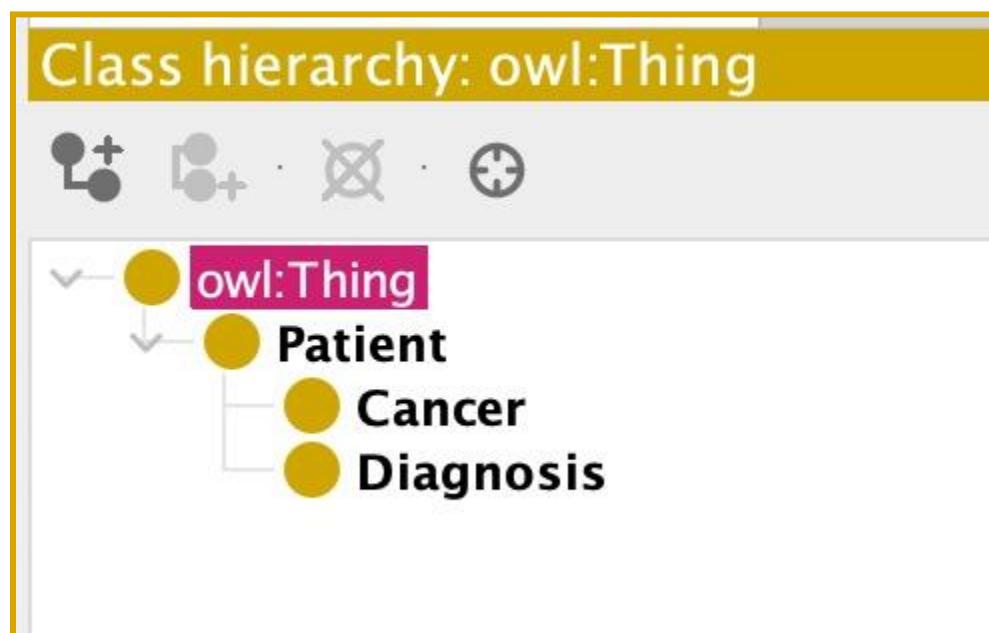
- Diagnosis_value: può avere come valori benigno (B) o maligno (M) per verificare lo stato del tumore.

- **Cancer:**

Sottoclasse di Patient.

Rappresenta il cancro al seno diagnosticato ai vari pazienti a cui vengono associati i valori visivi:

- radius_mean
- texture_mean
- perimeter_mean
- ...

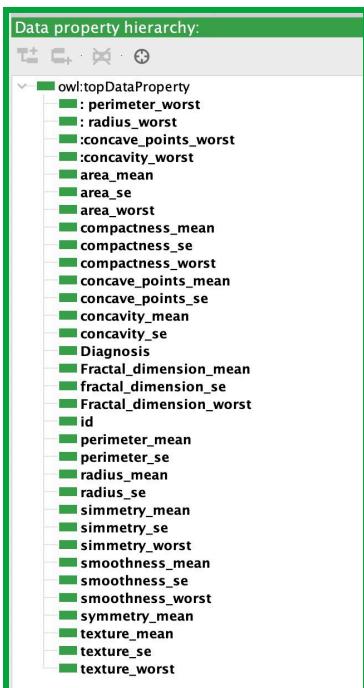
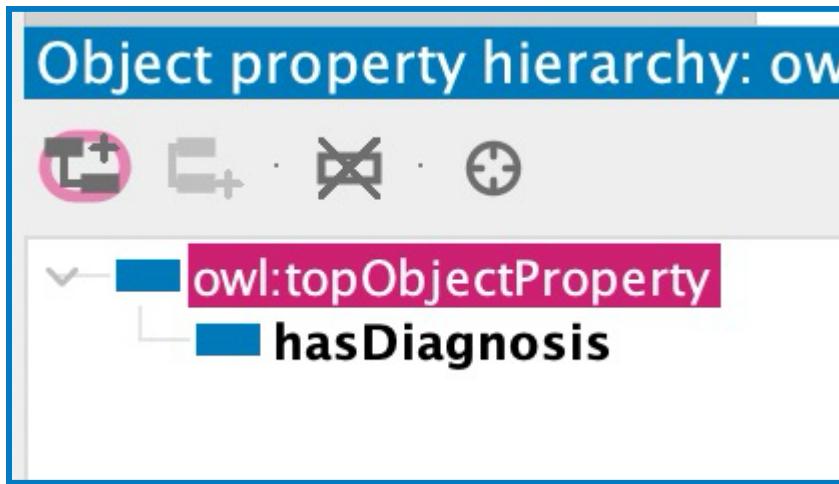


#2.2.2 OBJECT PROPERTY

Una **object property** consente di stabilire una **relazione tra due individui**, che possono appartenere sia a classi differenti sia alla stessa classe. Queste proprietà descrivono **legami strutturali** all'interno dell'ontologia, contribuendo a definire la semantica delle interazioni tra entità.

Tra le classi ho definito due relazioni che rappresentano come queste interagiscono tra di loro. Le relazioni in questione sono:

- *hasDiagnosis(Patient,Cancer) -> Diagnosis*



#2.2.3 DATA PROPERTY

Una **data property** definisce attributi o informazioni specifiche associate a un'istanza, mette in relazione un individuo con un valore primitivo.

- Le data property dei pazienti sono:
l'id per identificarlo e i valori delle sue sottoclassi.
- Le data property di Cancer sono:
i vari valori visivi (area, perimeter, radius, ...)
- Le data property di Diagnosis è:
diagnosis_value.

#2.2.4 INDIVIDUALS

Per alcune entità ho inserito delle **istanze (individuals)**, alcune ad esempio per individuare una tipologia di esame come quella istologica, altre per individuare istanze di persone a cui attribuire un cancro, ed effettuare prove di query DL

The image shows two side-by-side screenshots from the Protege ontology editor.

Left Screenshot: The title is "Individuals: Patient_000". It lists several individuals: Diagnosis_B, Diagnosis_M, Patient_000, Patient_001, Patient_002, and Patient_003. The individual "Patient_000" is highlighted with a pink background, indicating it is the current selected entity.

Right Screenshot: The title is "Property assertions: Patient_000". It shows object property assertions under the heading "hasDiagnosis Diagnosis_B". It also shows data property assertions under the heading "+". The data properties listed are: ': radius_worst' 10.23, area_mean 273.9, id 0, perimeter_mean 60.34, and radius_mean 9.504.

#3 QUERY

#3.1 DL Query

Interrogo l'ontologia tramite **DL - Descriptions Logics**, che e' un linguaggio formale utilizzato principalmente per la modellazione concettuale e l'ontologia nelle discipline dell'intelligenza artificiale e della rappresentazione della conoscenza.

QUERY REALIZZATE:

This screenshot shows a DL query interface with the following details:

DL query: Query (class expression)
Patient **and** (hasDiagnosis **value** Diagnosis_B)

Buttons: Execute, Add to ontology

Query results: Instances (2 of 2)
Patient_000, Patient_001

This screenshot shows a DL query interface with the following details:

DL query: Query (class expression)
Patient **and** (hasDiagnosis **value** Diagnosis_M)

Buttons: Execute, Add to ontology

Query results: Instances (2 of 2)
Patient_002, Patient_003

#3.1 OwlReady2

È stato poi creato il file “ontologyQuery.py” con il quale è possibile consultare l’ontologia direttamente in Python grazie alla libreria **OwlReady2** per la manipolazione di ontologie e il ragionamento.

Con il seguente codice è quindi possibile estrarre dall’ontologia la lista di classi, object property e data property.

```
print("ONTOLOGIA\n")

# Ottieni il percorso assoluto del file
current_dir = os.path.dirname(os.path.abspath(__file__))
owl_file = os.path.join(current_dir, "breast_ontology.owl")

onto = get_ontology(owl_file).load()

# Stampa di tutte le classi dell'ontologia
print("#####")
print("LISTA DELLE CLASSI DELL'ONTOLOGIA\n")
classes = list(onto.classes())
for cls in classes:
    print(f"- CLASSE: {cls.name}")
print()

# Stampa di tutte le object properties dell'ontologia
print("#####")
print("LISTA DELLE OBJECT PROPERTIES DELL'ONTOLOGIA\n")
object_properties = list(onto.object_properties())
for prop in object_properties:
    print(f"- OBJECT PROPERTY: {prop.name}")
print()

# Stampa di tutte le data properties dell'ontologia
print("#####")
print("LISTA DELLE DATA PROPERTIES DELL'ONTOLOGIA\n")
data_properties = list(onto.data_properties())
for prop in data_properties:
    print(f"- PROPERTY: {prop.name}")
print()
```

```
ONTOLOGIA

#####
LISTA DELLE CLASSI DELL'ONTOLOGIA

• CLASSE: Cancer
• CLASSE: Diagnosis
• CLASSE: Patient

#####
LISTA DELLE OBJECT PROPERTIES DELL'ONTOLOGIA

• OBJECT PROPERTY: hasDiagnosis

#####
LISTA DELLE DATA PROPERTIES DELL'ONTOLOGIA

• PROPERTY: area_mean
• PROPERTY: area_se
• PROPERTY: area_worst
• PROPERTY: compactness_mean
• PROPERTY: compactness_se
• PROPERTY: compactness_worst
• PROPERTY: concave_points_mean
• PROPERTY: concave_points_se
• PROPERTY: concave_points_worst
• PROPERTY: concavity_mean
• PROPERTY: concavity_se
• PROPERTY: concavity_worst
• PROPERTY: diagnosis
• PROPERTY: fractal_dimension_mean
• PROPERTY: fractal_dimension_se
• PROPERTY: fractal_dimension_worst
• PROPERTY: id
• PROPERTY: perimeter_mean
• PROPERTY: perimeter_se
• PROPERTY: perimeter_worst
• PROPERTY: radius_mean
• PROPERTY: radius_se
• PROPERTY: radius_worst
• PROPERTY: simmetry_mean
• PROPERTY: simmetry_se
• PROPERTY: simmetry_worst
• PROPERTY: smoothness_mean
• PROPERTY: smoothness_se
• PROPERTY: smoothness_worst
• PROPERTY: texture_mean
• PROPERTY: texture_se
• PROPERTY: texture_worst
```

Attraverso il seguente codice è stato invece possibile interrogare l'ontologia tramite [query](#).

```
# Query per ottenere pazienti con diagnosi "B" (Benigni)
diagnosis_B = onto.search_one(type=onto.Diagnosis, diagnosis="B")
query_result = list(onto.search(type=onto.Patient, hasDiagnosis=diagnosis_B))

print("\nPAZIENTI CON DIAGNOSI BENIGNA:")
for patient in query_result:
    print(f"• PAZIENTI: {patient.name}")

# Query per ottenere pazienti con diagnosi "M" (Maligni)
diagnosis_M = onto.search_one(type=onto.Diagnosis, diagnosis="M")
query_result = list(onto.search(type=onto.Patient, hasDiagnosis=diagnosis_M))

print("\nPAZIENTI CON DIAGNOSI MALIGNA:")
for patient in query_result:
    print(f"• PAZIENTI: {patient.name}")
```

```
#####
PAZIENTI CON DIAGNOSI BENIGNA:
• PAZIENTI: Patient_000, Patient_001

PAZIENTI CON DIAGNOSI MALIGNA:
• PAZIENTI: Patient_002, Patient_003
```

#4 APPRENDIMENTO SUPERVISIONATO

L'apprendimento supervisionato è una forma di apprendimento automatico in cui un modello viene addestrato su un insieme di dati etichettati, cioè contenenti input accompagnati dai rispettivi output corretti.

Lo scopo è permettere al modello di apprendere le relazioni tra input e output, così da poter effettuare previsioni accurate su dati mai visti prima.

In base alla natura del problema, l'apprendimento supervisionato può essere utilizzato per:
Classificazione, quando gli output appartengono a un insieme discreto e finito di categorie;
Regressione, quando gli output sono valori continui.

Durante la fase di addestramento, il modello cerca di riconoscere pattern e regolarità nei dati. In seguito, viene testato su dati non utilizzati in fase di training per valutare la sua capacità di generalizzare, distinguendo così un apprendimento reale da una semplice memorizzazione.

#4.1 DECISIONI DI PROGETTO

In questo progetto, l'obiettivo affidato all'apprendimento supervisionato è risolvere un **problema di classificazione**.

Ho utilizzato la colonna "*Diagnosis*" del DataSet come **variabile target** da predire basandosi sulle altre caratteristiche dei pazienti.

La prima cosa che ho fatto è stata decidere quali modelli di apprendimento supervisionato utilizzare.

Data l'elevata sensibilità del tema, quale lo stato di un tumore, era essenziale adottare algoritmi che garantissero alte prestazioni e precisione nelle previsioni, perciò ho scelto il **KNN** visto che è un algoritmo non parametrico facilmente interpretabile, fondamentale in ambito medico dove la trasparenza delle decisioni è cruciale per l'accettazione clinica e l'**SVM** che è particolarmente efficace quando il numero di feature è elevato rispetto al numero di campioni, situazione comune nei dataset medici. Tuttavia ho deciso di provare questi dati anche con altri due modelli quali **Random Forest** e **Neural Network**; il primo principalmente perché fornisce automaticamente un ranking dell'importanza delle variabili, cruciale per identificare i fattori prognostici più rilevanti per la sopravvivenza nel cancro al seno e il secondo per la scalabilità ovvero che se in futuro il dataset dovesse espandersi con più variabili (es. dati genomici, imaging), le neural network possono facilmente adattarsi.

- RECAP MODELLI REALIZZATI:
1. [**K NN - K Nearest Neighbors**](#)
 2. [**Random Forest**](#)
 3. [**SVM - Support Vector Machine**](#)
 4. [**Neural Network**](#)

#4.2 METRICHE

Andiamo adesso ad analizzare le metriche con cui sono stati valutati i modelli ed i risultati.

#4.2.1 DI VALIDAZIONE

La **cross-validation** è una tecnica comunemente utilizzata per valutare le **prestazioni** dei modelli in modo più **accurato** e **affidabile**, riducendo il rischio di risultati distorti dovuti alla suddivisione casuale dei dati.

In questo lavoro, ho scelto di applicare una **5-fold cross-validation**, suddividendo il dataset in cinque sottoinsiemi di pari dimensione. Il modello viene quindi addestrato e testato cinque volte, utilizzando ogni volta una fold diversa come set di test e le rimanenti come set di addestramento.

La **valutazione del modello** sarà basata sui risultati ottenuti durante la **cross-validation**: alcune metriche verranno calcolate sulla **migliore** delle cinque iterazioni, altre, invece, rifletteranno le **prestazioni medie** complessive su tutte le iterazioni

#4.2.2 DI VALUTAZIONE

Per ogni algoritmo implementato sono stati prodotti 2 tipologie di grafici differenti:

- **Bar Chart di Varianza e Deviazione Standard:**
Questo tipo di grafico a barre rappresenta la **Varianza** e la **Deviazione Standard** dei punteggi di **cross-validation**.
Aiutano a comprendere quanto i risultati siano consistenti o variabili su diverse iterazioni della cross-validation.
- **Stampa finale di alcune metriche:**
Classification Report e dei risultati stampati nel terminale:
esplorazione del DataSet, cv_scores_mean, cv_scores_variance, accuracy score, cv_score_devstandard, AUC e f1_score.

#4.3 SELEZIONE DELLE FEATURE

FEATURES SELEZIONATE:

Ho ritenuto che nessuna delle features del DataSet fosse irrilevante, in quanto erano tutte impattanti sulla diagnosi.

FEATURES SCARTATE:

/

#4.4 PREPROCESSING DEI DATI

L'addestramento di ogni modello inizia con l'esecuzione del Preprocessing dei dati.

#4.4.1 DUMMIFICATION & LABEL ENCODING

La **dummification** è una fase del preprocessing che consiste nella trasformazione di feature categoriche in **feature numeriche**.

In particolare, ogni valore possibile di una variabile categorica (cioè ogni categoria) viene convertito in una nuova feature binaria, che assume valore 1 se la categoria è presente e 0 altrimenti.

Nel mio caso, tuttavia, non è stata applicata una vera e propria dummification, bensì un'operazione di label encoding: ho trasformato una variabile categorica binaria con valori {'M', 'B'} in una variabile numerica binaria {1, 0}.

```
dataset['diagnosis'] = dataset['diagnosis'].map({'M': 1, 'B': 0})
```

#4.4.2 SMOTE

La funzione **SMOTE()**, implementabile grazie all'uso della libreria **imblearn**, ha lo scopo di ridimensionare la classe di esempi quando si lavora con DataSet sbilanciati, dove una classe è rappresentata in modo significativamente minore rispetto all'altra classe, la conseguenza e' che i modelli di machine learning possono venire "ingannati" dalla classe più numerosa e imparano a predire quasi sempre quella, ignorando la minoritaria.

Questo passo non è stato necessario in quanto il nostro DataSet è stato PREBILANCIATO dagli autori.

#4.4.3 STANDARDIZZAZIONE

Ho utilizzato la funzione **StandardScaler()** della libreria **scikit-learn** per standardizzare le feature, ovvero portarle tutte sulla **stessa scala**.

Questo passaggio è importante per evitare che variabili con scale diverse influenzino in modo sproporzionato l'addestramento dei modelli, specialmente quelli sensibili alle distanze tra le osservazioni.

In particolare:

La standardizzazione è stata fondamentale per modelli come K-NN, Neural Network, SVM e K-Means, che si basano direttamente sul calcolo di distanze o prodotti scalari. Al contrario, non è risultata necessaria per modelli come la Random Forest, che non si basano su distanze ma su suddivisioni basate su soglie, e quindi non sono influenzati dalla scala delle variabili.

```
# Standardizzazione dei dati
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

#4.5 DIVISIONE DEI DATI

SUDDIVISIONE DATASET: **80% TrainSet – 20% TestSet.**

Questa suddivisione è sembrata un **buon compromesso** tra l'esigenza di disporre di un numero sufficiente di dati per l'addestramento e quella di valutare in modo affidabile le prestazioni del modello.

- **Con una proporzione maggiore per il Test Set:** il modello avrebbe avuto meno dati su cui allenarsi, riducendo la sua capacità di apprendere le caratteristiche più complesse del dataset. Questo avrebbe potuto compromettere la sua capacità di generalizzare in modo efficace.
- **Con una proporzione maggiore per il Train Set:** si sarebbe corso il rischio di **overfitting**, ovvero che il modello imparasse anche il rumore e le particolarità dei dati di addestramento, con conseguente riduzione della capacità di adattarsi a nuovi dati.

```
# Divisione dataset (80% training, 20% test)
X_train, X_test, y_train, y_test = train_test_split(*arrays: X, y, test_size=0.20, random_state=42, shuffle=True, stratify=y)
```

#4.6 K-NEAREST NEIGHBORS (K-NN)

L'algoritmo alla base di questo metodo è il **Nearest Neighbor (NN)**, un approccio di classificazione che assegna a un dato in input la classe dell'esempio **più simile** presente nel dataset.

La similarità tra i dati viene misurata utilizzando una metrica, come ad esempio la **distanza euclidea**.

Il **K-Nearest Neighbors (K-NN)** rappresenta un'estensione del NN: invece di considerare un solo vicino, prende in esame i **K esempi più vicini** e assegna al dato in input la classe più frequente tra essi.

Questo consente di ottenere una classificazione più stabile e robusta, specialmente in presenza di rumore o dati ambigui.

#4.6.1 DECISIONI DI PROGETTO

Ho scelto di implementare inizialmente il modello **K-NN** perché risulta semplice da realizzare, flessibile e adatto a dataset di diverse dimensioni. Inoltre, si è dimostrato particolarmente **robusto** in presenza di rumore o valori anomali, poiché la classificazione finale si basa sui dati circostanti: ciò limita l'influenza di eventuali outlier sul risultato complessivo.

#4.6.2 OTTIMIZZAZIONE DEL VALORE 'K'

In genere:

- Valori di K troppo piccoli portano a problemi di **overfitting**.
- Valori di K troppo grandi portano a problemi di **underfitting**.

Per trovare il **valore ottimale** di K (numero di vicini) si itera per k da 1 a \sqrt{N} (N = numero di osservazioni nel DataSet).

Per ogni k si:

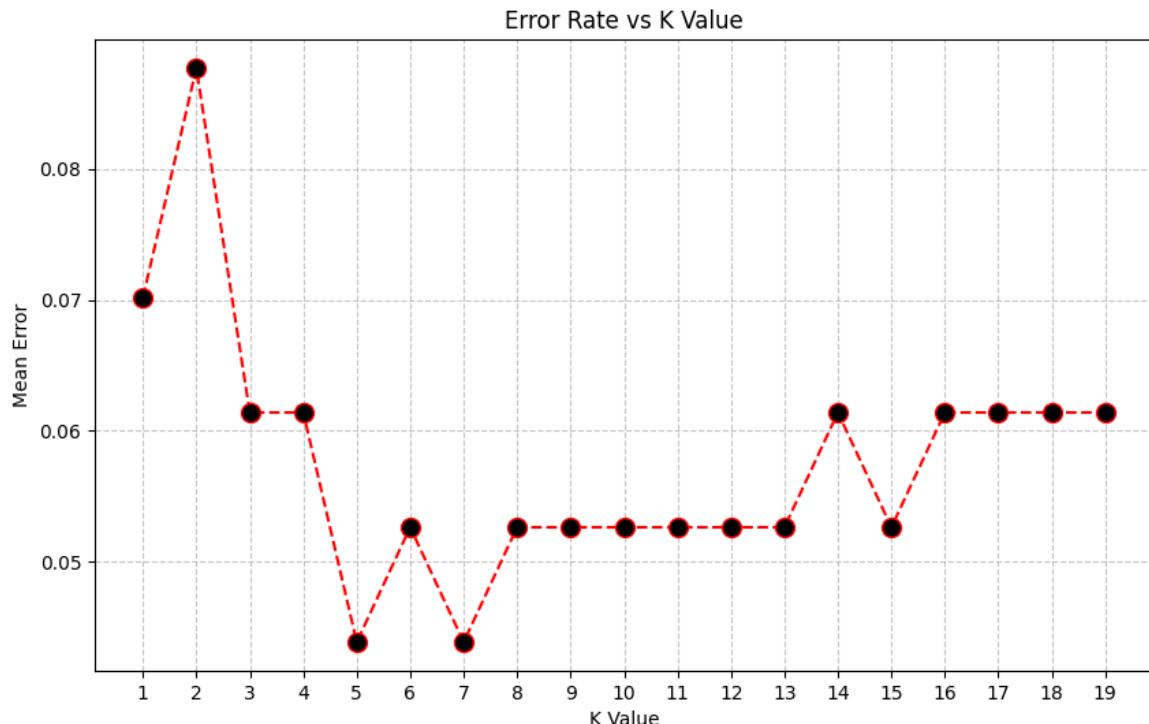
- Crea un classificatore K-NN con k vicini.
- Per ogni valore di k costruisce una pipeline (scaling, KNN)
- Addestra il modello sui dati di training (X_{train}, y_{train}).
- Effettua previsioni su X_{test} .
- Calcola l'errore medio di classificazione e lo salva in error.

```

# Trovo K ottimale eseguendo scaling e knn
error = []
for i in range(1, 20):
    pipeline_tmp = [
        ('scaler', StandardScaler()),
        ('knn', KNeighborsClassifier(n_neighbors=i))
    ]
    from sklearn.pipeline import Pipeline

    p = Pipeline(steps=pipeline_tmp)
    p.fit(X_train, y_train)
    pred_i = p.predict(X_test)
    error.append(np.mean(pred_i != y_test))

```



Consultando il grafico ottenuto constatiamo che esistono 2 valori ottimali per il parametro k.

Ho deciso di scegliere tra questi il valore minore in modo tale da ottenere un modello più sensibile e reattivo ovvero **k = 5**.

#4.6.3 ADDESTRAMENTO

Creiamo il nostro classificatore con **k** vicini.

Usiamo il metodo **fit()** in **scikit-learn** per addestrare un modello sui dati forniti.

A differenza di altri algoritmi di machine learning il K-NN non apprende un modello matematico basato sui dati di training ma memorizza semplicemente i dati di training.

```
pipeline.fit(X_train, y_train)
```

#4.6.4 PREDIZIONE

Uso il metodo **predict()** in **scikit-learn** per fare **previsioni** su ogni dato del TestSet basandosi sulle informazioni apprese dal modello addestrato in precedenza.

Vado poi a calcolare l'**accuratezza** facendo il rapporto tra il numero di previsioni corrette e il numero totale di previsioni effettuate.

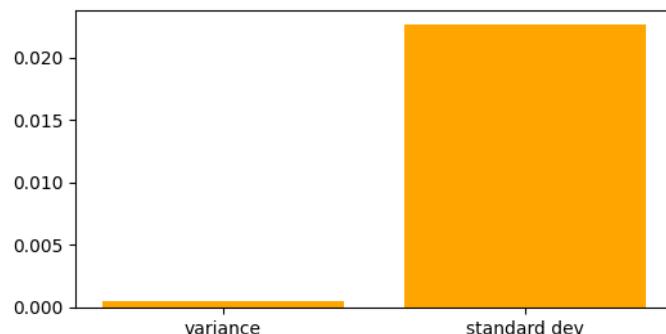
Usiamo quindi il metodo **accuracy_score()** in **scikit-learn** che restituisce la percentuale di previsioni corrette sul TestSet.

```
prediction = pipeline.predict(X_test)
accuracy = accuracy_score(y_test, prediction)
print(f"\nAccuracy Score: {accuracy:.4f}")
```

#4.6.5 VALUTAZIONE FINALE

BAR CHART DI VARIANZA E DEVIAZIONE STANDARD

Il grafico indica una bassa dispersione nei risultati del modello. Ciò suggerisce che il modello ha performance stabili e costanti attraverso le diverse esecuzioni, con fluttuazioni minime nelle sue predizioni.



```
0   id                      569 non-null    int64
1   diagnosis                569 non-null    object
2   radius_mean               569 non-null    float64
3   texture_mean              569 non-null    float64
4   perimeter_mean            569 non-null    float64
5   area_mean                 569 non-null    float64
6   smoothness_mean           569 non-null    float64
7   compactness_mean          569 non-null    float64
8   concavity_mean            569 non-null    float64
9   concave_points_mean       569 non-null    float64
10  symmetry_mean             569 non-null    float64
11  fractal_dimension_mean    569 non-null    float64
12  radius_se                 569 non-null    float64
13  texture_se                569 non-null    float64
14  perimeter_se              569 non-null    float64
15  area_se                   569 non-null    float64
16  smoothness_se              569 non-null    float64
17  compactness_se             569 non-null    float64
18  concavity_se              569 non-null    float64
19  concave_points_se         569 non-null    float64
20  symmetry_se                569 non-null    float64
21  fractal_dimension_se       569 non-null    float64
22  radius_worst               569 non-null    float64
23  texture_worst              569 non-null    float64
24  perimeter_worst            569 non-null    float64
25  area_worst                 569 non-null    float64
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
dtypes: float64(30), int64(1), object(1)
```

```
Cross-validation results:
```

```
Mean accuracy: 0.9626
```

```
Standard deviation: 0.0226
```

```
cv_score variance:0.0005120154570704022
```

```
Accuracy Score: 0.9561
```

```
Classification Report:
```

	precision	recall	f1-score	support
0	0.95	0.99	0.97	72
1	0.97	0.90	0.94	42
accuracy			0.96	114
macro avg	0.96	0.95	0.95	114
weighted avg	0.96	0.96	0.96	114

```
Confusion matrix:
```

```
[[71  1]
 [ 4 38]]
```

```
AUC: 0.982
```

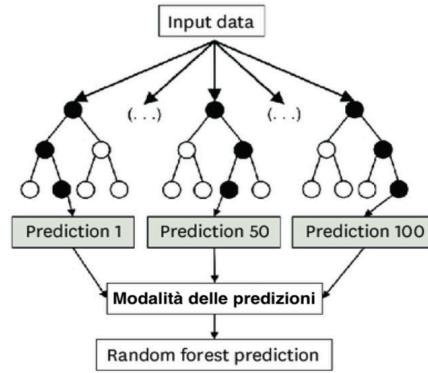
```
F1 Score: 0.9383
```

#4.7 RANDOM FOREST

Si tratta di un algoritmo che costruisce una "foresta" di alberi decisionali, ciascuno dei quali viene addestrato su un sottoinsieme casuale del dataset e delle sue caratteristiche (tecnica nota come bagging).

Ogni albero produce una propria previsione, e il risultato finale viene ottenuto tramite:

- **Media** nel caso di **Regressione**.
 - **Modalità** nel caso di **Classificazione**.
- delle previsioni fatte da tutti gli alberi.



#4.7.1 DECISIONI DI PROGETTO

I vantaggi del Random Forest includono una solida capacità di **generalizzazione** e una buona resistenza all'**overfitting**, grazie alla combinazione di più alberi decisionali.

È in grado di gestire efficacemente sia **feature numeriche che categoriali**, e si adatta bene anche a situazioni con **classi sbilanciate**.

Il modello tende a offrire **buona precisione e stabilità** nei compiti di classificazione.

Inoltre, la possibilità di **addestrare gli alberi in parallelo** ne migliora le prestazioni in termini di velocità, rendendolo adatto anche in ambienti con risorse computazionali limitate.

#4.7.2 OTTIMIZZAZIONE PARAMETRI DEL CLASSIFICATORE

Per costruire il classificatore Random Forest usiamo il metodo

RandomForestClassifier() presente nella libreria **scikit-learn** che prende 3 parametri:

- **max_depth**:
La profondità massima dell'albero decisionale.
 - Valore Alto: alberi più **complessi**, ma con rischio di **overfitting**.
 - Valore Bassa: alberi più **semplici**, ma con rischio di **underfitting**.
- **random_state**:
Controlla la generazione dei numeri casuali all'interno del modello. Fornendo un valore specifico ci si assicura che l'addestramento e la previsione del modello siano riproducibili.
- **n_estimators**:
Numero di alberi decisionali.
 - Valore Alto: modello più **complesso e performante**, ma con rischio di **overfitting**.
 - Valore Bassa: modello più **veloce**, ma con **performance inferiori**.

STEP #1

Per ottimizzare i parametri del modello Random Forest andiamo per prima cosa a definire un intervallo di valori da testare per ciascun parametro.

STEP #2

Utilizziamo cicli annidati per testare tutte le possibili combinazioni dei valori definiti per i parametri.

Per ogni combinazione di parametri:

- Viene creato un modello Random Forest con i parametri specificati.
- Viene eseguita una cross-validation a 5 fold tramite `cross_val_score`. La cross-validation suddivide il DataSet in 5 parti (fold), addestra il modello su 4 parti e testa la sua performance sulla parte rimanente. Questo processo viene ripetuto 5 volte, con ogni parte utilizzata una volta per il test. La funzione `cv_scores.mean()` calcola la media del punteggio di cross-validation.

STEP #3

Trovata la combinazione ottimale viene creato il modello finale.

```
# intervallo di valori da testare per max_depth
max_depth_values = [2, 4, 6, 8, 10, 12, 14, 16, 18, 20]

# intervallo di valori per random_state
random_state_values = [0, 4, 16, 64, 256, 1024, 4096]

# Memorizza i punteggi medi di cross-validation per ogni combinazione di max_depth e
scores = []
for max_depth in max_depth_values:
    for random_state in random_state_values:
        RFC = RandomForestClassifier(max_depth=max_depth, random_state=random_state)
        cv_scores = cross_val_score(RFC, X, y, cv=5)
        scores.append((max_depth, random_state, cv_scores.mean()))
```

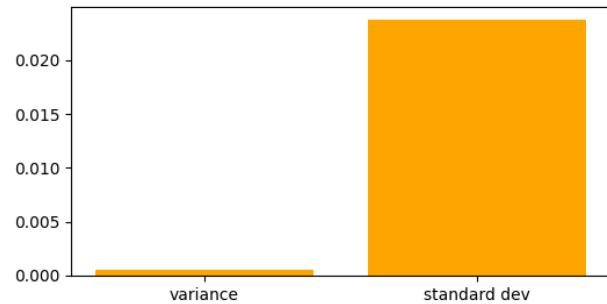
Valore migliore `max_depth`: 8.0

Valore migliore `random_state`: 4.0

#4.7.3 VALUTAZIONE FINALE

BAR CHART DI VARIANZA E DEVIAZIONE STANDARD

Il grafico mostra che i risultati del modello sono molto stabili, con una bassa dispersione tra le diverse esecuzioni. Questo suggerisce che le prestazioni del modello sono costanti e affidabili.



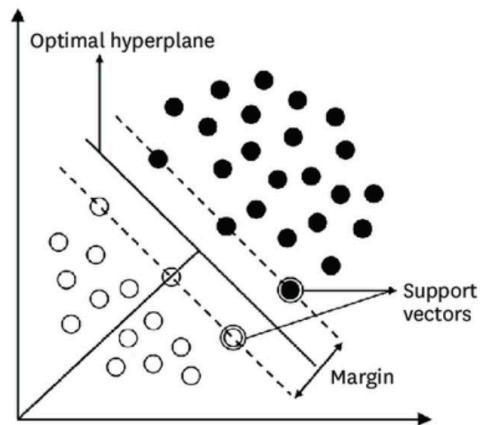
```
Classification report:  
precision    recall   f1-score   support  
  
          0       0.96      1.00      0.98      72  
          1       1.00      0.93      0.96      42  
  
accuracy           0.97      0.97      0.97     114  
macro avg       0.98      0.96      0.97     114  
weighted avg     0.97      0.97      0.97     114  
  
  
Confusion matrix:  
[[72  0]  
 [ 3 39]]  
  
cv_scores mean:0.9666511411271541  
  
cv_score variance:0.0005655664070423323  
  
cv_score dev standard:0.02378164012515395  
  
  
AUC: 0.993  
  
f1 score: 0.9629629629629629
```

#4.8 SUPPORT VECTOR MACHINES (SVM)

L'obiettivo principale di una **Support Vector Machine (SVM)** è individuare un **iperpiano ottimale** che separi le classi (nella classificazione) o che predica i valori target (nella regressione).

Le SVM puntano a **massimizzare il margine** tra l'iperpiano e i support vectors, ovvero i punti dati più vicini al confine decisionale.

Questa strategia contribuisce a rendere il modello **più robusto** e meno soggetto all'**overfitting**, migliorando così la capacità di generalizzazione su dati nuovi.



#4.8.1. DECISIONI DI PROGETTO

Questo modello è stato scelto non solo perché raccomandato dagli autori del dataset, ma anche per la sua efficacia nella **classificazione binaria**, soprattutto quando le classi risultano ben separate.

Le **SVM** si dimostrano particolarmente adatte a problemi con un **numero moderato di feature** e possono gestire **relazioni non lineari** grazie all'impiego dei **kernel**.

In particolare, è stato scelto il **kernel RBF (Radial Basis Function)**, poiché consente al modello di catturare **strutture complesse e non lineari** presenti nei dati. Questo approccio migliora la capacità della SVM di produrre una classificazione **accurata e ben generalizzabile**.

#4.8.2 OTTIMIZZAZIONE PARAMETRO GAMMA

Il parametro **gamma** è un iperparametro cruciale quando si usa il kernel RBF in un classificatore SVM.

Controlla quanto l'influenza di un singolo esempio di addestramento si estenda.

- Valori più alti di gamma rendono le decisioni più locali.
- Valori più bassi danno una maggiore influenza alle istanze circostanti.

Per trovare il miglior valore gamma ho utilizzato la tecnica della **grid search** insieme alla cross-validation.

STEP #1

Si inizia definendo una griglia di valori di gamma.

STEP #2

Dopo aver creato un modello SVM, si esegue la ricerca a griglia che addestrerà e valuterà il modello su diverse combinazioni di parametri gamma utilizzando la cross-validation con 5 fold.

STEP #3

Con il metodo fit, si otterrà il modello SVM ottimizzato con i migliori parametri gamma individuati dalla grid search.

```
# Grid Search per gamma
param_grid = {'gamma': [1e-4, 1e-3, 0.01, 0.1, 1, 10, 100]}
svm_model = svm.SVC(kernel='rbf', probability=True)
grid_search = GridSearchCV(svm_model, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train_scaled, y_train)

# Stampa risultati Grid Search
print("Risultati Grid Search:")
for gamma, mean_score in zip(grid_search.cv_results_['param_gamma'],
                               grid_search.cv_results_['mean_test_score']):
    print(f"Gamma: {gamma:.10f} Mean Score: {mean_score:.4f}")
print(f"\nMiglior valore di gamma: {grid_search.best_params_['gamma']}")
```

```
Risultati Grid Search:
Gamma: 0.0001      Mean Score: 0.7454
Gamma: 0.001       Mean Score: 0.9462
Gamma: 0.01        Mean Score: 0.9689
Gamma: 0.1         Mean Score: 0.9503
Gamma: 1.0         Mean Score: 0.6314
Gamma: 10.0        Mean Score: 0.6273
Gamma: 100.0        Mean Score: 0.6273

Miglior valore di gamma: 0.01
```

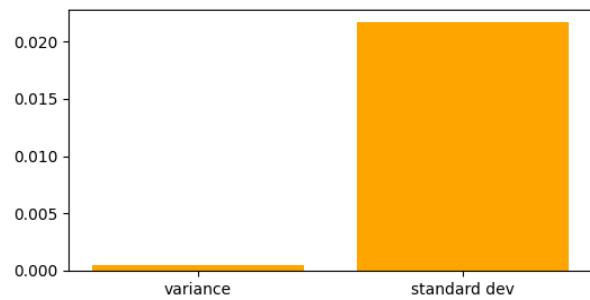
Questi risultati ci fanno capire che il modello sarà ottimizzato per il valore gamma **0.01**.

```
# Training con il miglior parametro
clf = svm.SVC(kernel='rbf', gamma=grid_search.best_params_['gamma'], probability=True)
clf.fit(X_train_scaled, y_train)
```

#4.8.3 VALUTAZIONE FINALE

BAR CHART DI VARIANZA E DEVIAZIONE STANDARD

Il grafico indica che i risultati del modello sono molto stabili, con una bassa dispersione tra le diverse esecuzioni. Questo suggerisce che il modello ha prestazioni consistenti e affidabili.



```
Miglior score: 0.9689
```

```
Accuracy Score: 0.9767
```

```
Classification Report:
```

	precision	recall	f1-score	support
0	0.96	1.00	0.98	54
1	1.00	0.94	0.97	32
accuracy			0.98	86
macro avg	0.98	0.97	0.97	86
weighted avg	0.98	0.98	0.98	86

```
Confusion matrix:
```

```
[[54  0]
 [ 2 30]]
```

```
Risultati Cross-validation:
```

```
Media accuracy: 0.9689
```

```
Deviazione standard: 0.0217
```

```
Varianza: 0.0005
```

```
AUC: 0.999
```

```
F1 Score: 0.9677
```

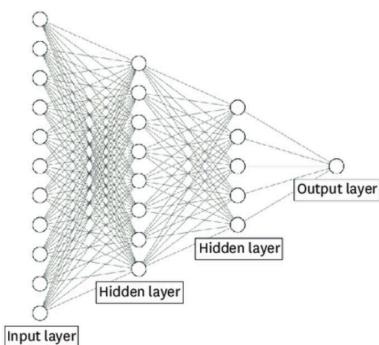
#4.9 NEURAL NETWORK

Le **reti neurali** sono modelli ispirati al funzionamento del cervello umano, composti da neuroni artificiali interconnessi.

Ogni **neurone** riceve degli **input**, li **pesa** in base alla loro **importanza**, li **somma** e confronta il risultato con una **soglia di attivazione**.

Se il valore supera tale soglia, il neurone si attiva e trasmette l'informazione agli altri neuroni.

In questo modo, la rete è in grado di elaborare stimoli complessi e apprendere relazioni non lineari tra i dati.



#4.9.1 DECISIONI DI PROGETTO

Il modello è stato scelto per la sua capacità di affrontare **problemi di apprendimento profondi e complessi** e per la sua capacità di raggiungere **prestazioni di livello superiore** in termini di accuratezza e precisione.

Ho scelto di definire una rete neurale sequenziale con due livelli:

- **1° LIVELLO:**
64 neuroni nascosti, utilizza la funzione di attivazione **ReLU**;
- **2° LIVELLO:**
32 neuroni nascosti, utilizza la funzione di attivazione **ReLU**;
- **3° LIVELLO:**
1 singolo neurone con una funzione di attivazione **sigmoidale**, che produce un valore compreso tra 0 e 1 per la classificazione binaria.

#4.9.2 CREAZIONE MODELLO E ADDESTRAMENTO

STEP #1

Definiamo la funzione **create_model()** che costruisce e restituisce un modello di rete neurale creato con **Keras**:

1. Usiamo il modello sequenziale, che permette di impilare i livelli uno dopo l'altro.
2. Aggiungiamo 2 **fully connected layer** rispettivamente di 64 e di 32 unità nascoste che usano la **ReLU - Rectified Linear Unit** come funzione di attivazione dei neuroni.
3. Aggiungiamo un **fully connected layer** di una sola unità che usa la **sigmoid** come funzione di attivazione dei neuroni per comprimere l'output in un intervallo tra 0 e 1.
4. La funzione termina con la compilazione del modello, utilizzando la funzione **binary_crossentropy**, che è appropriata per problemi di classificazione binaria e l'ottimizzatore **adam** che viene utilizzato per addestrare la rete, ed è noto per la sua efficacia in una varietà di scenari.

```

# Costruisco il modello della rete neurale
def create_model(): 2 usages & rafflog01
    model = keras.Sequential([
        keras.layers.Input(shape=(X_train_scaled.shape[1],)),
        keras.layers.Dense(units=64, activation='relu'),
        keras.layers.Dense(units=32, activation='relu'),
        keras.layers.Dense(units=1, activation='sigmoid')
    ])

    # Compilazione del modello
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

    return model

```

STEP #2

Si crea quindi un'istanza del modello e inizia l'addestramento sui dati.

I parametri in input sono:

- **epochs:**
Specifica quante volte l'intero TrainingSet verrà utilizzato per aggiornare i pesi del modello.
 - Ho scelto di addestrare il modello per 30 epochs, dato che maggiori epochs potrebbero consentire al modello di adattarsi meglio ai dati di addestramento, ma potrebbero anche portare all'**overfitting**.
- **batch_size:**
Definisce la dimensione del batch, ovvero quante istanze di addestramento vengono utilizzate prima di aggiornare i pesi del modello.
 - Ho scelto di utilizzare un batch di dimensione 64, perché rappresenta un **compromesso** tra l'accuratezza del modello e l'efficienza di addestramento. Potrebbe fornire un'adeguata quantità di dati per apprendere modelli complessi senza richiedere troppo tempo computazionale.

```

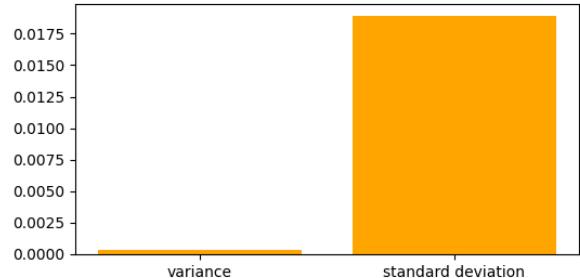
# Creazione istanza del modello e addestramento
model1 = create_model()
model1.fit(X_train_scaled, y_train, epochs=30, batch_size=64)

```

#4.9.3 VALUTAZIONE FINALE

BAR CHART DI VARIANZA E DEVIAZIONE STANDARD

Il Bar Chart di Varianza e Deviazione Standard mostra una varianza di circa 0.0004 e una deviazione standard di 0.02, indicando che le prestazioni del modello sono costanti e affidabili.



```
Classification report:  
precision    recall    f1-score   support  
  
          0       0.99      1.00      0.99      72  
          1       1.00      0.98      0.99      42  
  
accuracy                           0.99      114  
macro avg       0.99      0.99      0.99      114  
weighted avg     0.99      0.99      0.99      114  
  
  
Confusion matrix:  
[[72  0]  
 [ 1 41]]
```

```
cv_scores mean:0.9648351669311523  
  
cv_score variance:0.00035744471043841486  
|  
cv_score standard deviation:0.01890620825121777
```

```
AUC: 0.997
```

```
f1 score: 0.9879518072289156
```

#4.10 CONCLUSIONI

Nella seguente tabella sono elencate le metriche usate per confrontare i modelli.

	ACCURATEZZA	VARIANZA	DEV. STANDARD	F1-SCORE	AVERAGE-PRECISION	AUC
K-NN	0.96	0.0005	0.0226	0.9383	0.96	0.982
RANDOM FOREST	0.97	0.0006	0.0237	0.9629	0.98	0.993
SVM	0.97	0.0005	0.0217	0.9677	0.98	0.999
NEURAL NETWORK	0.99	0.00035	0.0189	0.9879	0.99	0.997

CONCLUSIONI DALLA TABELLA

- **Neural Network è il miglior modello**
 - Ha la **maggiore accuratezza (0.99)** e il **miglior F1-score (0.9879)**, il che indica un'eccellente capacità di classificazione.
 - Ha anche la **varianza più bassa (0.00035)** e la **deviazione standard più piccola (0.0189)**, quindi è più stabile rispetto agli altri modelli.
- **SVM e Random Forest sono molto validi**
 - Entrambi hanno un'**accuratezza di 0.97**, con SVM leggermente migliore nell'**F1-score (0.9677 vs 0.9629)**.
 - SVM ha anche il miglior **AUC (0.999)**, indicando un'eccellente separabilità tra le classi.
- **K-NN è il meno performante**
 - Ha la **minor accuratezza (0.96)** e il **minore F1-score (0.9383)**.

CONCLUSIONE GENERALE

Considerando l'importanza di una diagnosi accurata nel contesto medico, ho quindi deciso di adottare il modello di **Neural Network** per la classificazione dei tumori, garantendo così un'analisi più precisa e affidabile per supportare il processo diagnostico.

#5 APPRENDIMENTO NON SUPERVISIONATO: CLUSTERING

Questa tecnica permette di **individuare raggruppamenti naturali** all'interno dei dati anche in **assenza di etichette di classe**.

Ciò può rivelare **relazioni nascoste** tra le risposte al questionario, offrendo una prospettiva alternativa sulla struttura e sulla distribuzione del dataset.

Il clustering può essere di due tipi:

- **Hard Clustering:**
Prevede l'assegnazione statica di ciascun esempio a una classe di appartenenza.
- **Soft Clustering:**
Utilizza distribuzioni di probabilità per assegnare le classi associate a ciascun esempio.

#5.1 DECISIONI PROGETTUALI

Questa tecnica è stata scelta per raggruppare i pazienti in categorie basate sulle caratteristiche presenti nel file 'breast-cancer.csv'.

L'obiettivo è individuare dei **cluster**, ovvero gruppi di esempi simili a **centroidi** calcolati in modo automatico.

L'utilizzo di questa tecnica nel progetto mira a creare una nuova colonna da aggiungere alle features relative ai dati di ciascun paziente, in un nuovo file chiamato 'breast-cancer_clusters.csv'.

Questo potrebbe aprire nuove prospettive sulla varietà delle risposte e potenzialmente portare a nuove intuizioni sul cancro al seno e contribuire a una diagnosi più accurata e comprensiva.

Nel nostro caso, ho scelto di utilizzare una tecnica di **Hard Clustering**, in particolare il **k-means**, implementato tramite la libreria **Scikit-learn**.

#5.2 K-MEANS

K-Means è un algoritmo di **clustering** che suddivide un insieme di dati in K cluster, dove K è un valore predefinito. L'obiettivo è **raggruppare i dati** in modo tale che gli elementi all'interno di uno stesso cluster siano quanto più **simili** tra loro, mentre risultino distinti rispetto agli elementi appartenenti ad altri cluster.

L'algoritmo cerca di **minimizzare la distanza complessiva** tra ciascun punto e il centroide del proprio cluster, ottimizzando così la compattezza interna dei gruppi formati.

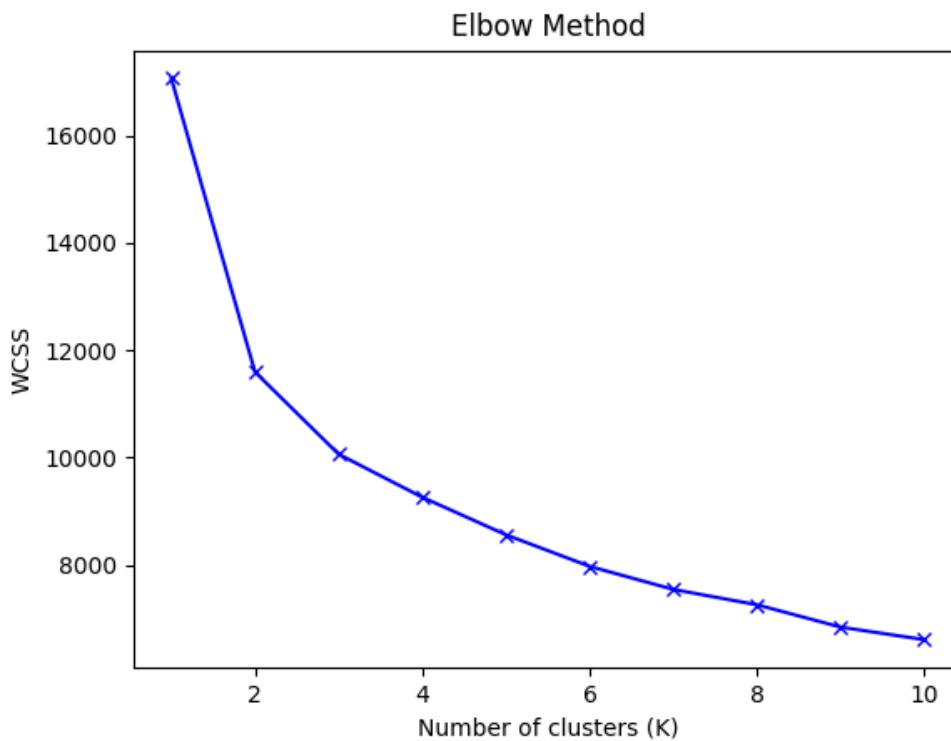
STEP #1

Ho dovuto, per prima cosa, determinare il numero ottimale di cluster. A tal scopo, ho utilizzato il **Metodo del Gomito** (Elbow Method):

- Sull'asse **Y** rappresentiamo il **WCSS - Within-Cluster Sum of Squares**, che misura la compattezza dei cluster.
- Sull'asse **X** rappresentiamo il numero di cluster **K**.
 - Il punto ottimale viene scelto osservando dove il WCSS **smette di diminuire significativamente**, formando un angolo simile a un "gomito".

```
# Calcolo WCSS per diversi valori di K
wcss = []
silhouette_scores = []
k_range = range(1, 11)

for k in k_range:
    kmeans = KMeans(n_clusters=k, n_init=10, random_state=42)
    kmeans.fit(X_scaled)
    wcss.append(kmeans.inertia_)
    # Il silhouette score non può essere calcolato per k=1
    if k > 1:
        silhouette_scores.append(silhouette_score(X_scaled, kmeans.labels_))
    else:
        silhouette_scores.append(0) # Per k=1 impostiamo il silhouette score a 0
```



STEP #2

Dall'analisi precedente, il valore di **K ottimale è risultato essere**

2. Addestro quindi il modello sui 2 cluster.

```
# Addestramento del modello finale con K=2
kmeans_final = KMeans(n_clusters=2, n_init=10, random_state=42)
kmeans_final.fit(X_scaled)
```

I parametri scelti per la configurazione del modello indicano:

- **n_clusters:**

Il numero dei cluster in cui dividere i dati del DataSet.

Nel mio caso avrà valore pari a 2.

- **n_init:**

Specifica il numero di volte che l'algoritmo sarà eseguito con diverse inizializzazioni casuali dei centroidi.

- Un valore maggiore aumenta le probabilità di ottenere una soluzione di migliore qualità a scapito di un maggior tempo di calcolo.

Ho scelto un valore non troppo alto, ma sufficiente per massimizzare la qualità dell'output, quindi 10.

- **random_state:**

Questo parametro controlla la riproducibilità dei dati.

- Fissandolo ad un valore alto, il modello fornirà gli stessi risultati quando viene addestrato con gli stessi dati.

Ho fissato il suo valore a 42.

#5.3 VALUTAZIONE FINALE DEL MODELLO

Sono state valutate le prestazioni del modello utilizzando due metriche:

- **WCSS (Within-Cluster Sum of Squares):**

Calcola la somma dei quadrati delle distanze di ogni punto rispetto al proprio centroide.

- Un valore più basso di WCSS indica una maggiore coesione all'interno dei cluster.

- **Silhouette Score:**

Valuta quanto un punto sia vicino al proprio cluster rispetto ai cluster vicini.

- **Valori vicini a 1:** Cluster ben separati.
- **Valori vicini a 0:** Cluster che si sovrappongono.
- **Valori negativi:** Assegnazioni errate dei punti ai cluster.

-WCSS: 11595.53
-Silhouette Score: 0.3434

	WITHIN-CLUSTER SUM OF SQUARES	SILHOUETTE SCORE
K-MEANS	11595.53	0.3434

CONCLUSIONE GENERALE

Con uno Silhouette Score pari a 0.3434 è possibile affermare che il clustering è **accettabile**, ma non perfettamente separato.

Il risultato ottimale sarebbe stato ottenere un valore superiore a 0.5, che avrebbe indicato una separazione più netta. Da ciò ho dedotto che potrebbero esserci sovrapposizioni tra i cluster e quindi deduco che il dataset **non è naturalmente divisibile in due gruppi ben distinti**. Questa sovrapposizione può derivare da dati rumorosi e ciò potrebbe aver creato difficoltà a creare un confine di decisione chiaro tra i gruppi.

#6 CONCLUSIONE

In questo progetto ho esplorato il tema delicato del cancro al seno da diverse prospettive, applicando tecniche di intelligenza artificiale per analizzare e classificare i dati relativi a questa patologia.

L'obiettivo prefissato era quello di sviluppare uno strumento di supporto alla diagnosi, in grado di aiutare a distinguere tra tumori benigni e maligni attraverso l'utilizzo dell'AI. I risultati ottenuti confermano la validità di questo approccio, evidenziando come l'intelligenza artificiale possa rappresentare un valido ausilio nel processo diagnostico.

#6.1 POSSIBILI SVILUPPI

Questo progetto potrebbe avere una serie di sviluppi futuri per migliorare la sua efficacia e l'applicazione pratica delle scoperte.

Alcune possibili direzioni per lo sviluppo futuro possono essere:

- **L'espansione del DataSet:**

Banalmente raccogliere dati aggiuntivi e aumentare le dimensioni del DataSet potrebbe migliorare ulteriormente le prestazioni dei modelli.

Un DataSet più grande può fornire maggiore variabilità nei dati e consentire una migliore generalizzazione.

- **L'integrazione con l'interpretazione medica:**

Ovvero collegare le scoperte dei modelli di apprendimento supervisionato con specialisti in ambito medico che potrebbero aiutarci ad indirizzare meglio il nostro lavoro.