RELAZIONE PROGETTO DI INGEGNERIA DELLA CONOSCENZA 2022-2023

SISTEMA DI DIAGNOSTICA PREVENZIONE ATTACCHI CARDIACI

Studenti:

- **Dibenedetto Ruggiero 724561 r.dibenedetto21@studenti.uniba.it**
- **Digioia Marco 724562 m.digioia31@studenti.uniba.it**

INDICE

1) Introduzione
2) Sistema esperto
3) CSP
4) Ontologia
5) Apprendimento non supervisionato
6) Apprendimento supervisionato

INTRODUZIONE

L’ attacco cardiaco è una condizione medico grave che causa danni al muscolo cardiaco. La prevenzione degli attacchi cardiaci è fondamentale per mantenere un cuore sano e prevenire conseguenze gravi, come insufficienza cardiaca o ictus.

Un sistema di diagnostica per la prevenzione degli attacchi cardiaci è un insieme di procedure e strumenti utilizzati per effettuare una diagnosi precoce ed aiutano a identificare precocemente il rischio di un attacco cardiaco e prevenire la sua insorgenza.

Tramite l’aiuto di alcuni pazienti presi in esame il nostro programma previene e verifica un possibile attacco cardiaco, prendendo in esame i vari sintomi che si possono presentare.

Il progetto si basa su diversi argomenti tra cui:

- Un sistema esperto basato sulle risposte ottenute dall’utente per poter prevenire un attacco cardiaco
- L’utilizzo di algoritmi di apprendimento non supervisionato per la previsione di attacchi cardiaci
- L’utilizzo di algoritmi di apprendimento supervisionato per la previsione di attacchi cardiaci

SISTEMA ESPERTO

Un sistema esperto è un tipo di software che utilizza la conoscenza di un esperto umano in un determinato dominio offrendo soluzioni e consigli a problemi specifici.

Tale sistema si basa su un insieme di **regole** e **fatti** che l'intelligenza artificiale utilizza per prendere delle decisioni.

Per implementare il sistema esperto abbiamo deciso di utilizzare la libreria di nome **Experta** che viene messa a disposizione da python.

Fatti

Un fact(fatto) in un sistema esperto è una specifica informazione o conoscenza che il sistema ha acquisito e memorizzato nella sua base di conoscenza. Rappresentano le informazioni di base su cui si basa il sistema esperto per raggiungere una soluzione o una risposta.

```
class AttacchiDiCuore_expert(KnowledgeEngine):
    @DefFacts()
```

@DefFacts è un costrutto attraverso il quale è possibile definire un insieme di fatti di base da utilizzare nel sistema esperto. Nel nostro caso viene utilizzato per definire un insieme di fatti basate sulle diverse analisi che un paziente può effettuare (colesterolo, pressione sanguigna , frequenza cardiaca).

REGOLE

In **Experta** una regola è rappresentata da un insieme di condizioni logiche che vengono valutate per determinare se un'azione deve essere eseguita o meno. Una regola è formata da due parti **lhs** e **rhs.**

Lhs si riferisce alla parte sinistra e rappresenta la condizione che deve essere soddisfatta affinché sia considerata vera.

Rhs si riferisce alla parte destra che contiene l'azione che deve essere effettuata se la regola è considerata vera.

```python
@Rule(Fact(chiedi_sintomi="si"))
def rule_5(self):
    r1 = self._prototype_ask_symptom("Provi un fastidio al torace(pressione/dolore)? [si/no] ", Fact(molto_dolore="si"))
    r2 = self._prototype_ask_symptom("Mancanza di fiato? [si/no] ", Fact(mancanza_fiato="si"))
    r3 = self._prototype_ask_symptom("Disagio nella parte superiore del corpo(Braccia,Spalle,Collo,Schiena)? [si/no] ", Fact(sente_disagio="si"))
    r4 = self._prototype_ask_symptom("Provi nausea,vertigi,aumento della sudorazione? [si/no] ", Fact(sente_sudorazione="si"))
    r5 = self._prototype_ask_symptom("Provi tanta debolezza? [si/no] ", Fact(sente_debolezza="si"))

    if r1 == "no" and r2 == "no" and r3 == "no" and r4 == "no" and r5 == "no":
        self.flag_no_symptoms = 1
```

L'utente sarà sottoposto ad una serie di domande alle quali dovrà rispondere attentamente. Questa regola prevede che, se viene attivata la condizione chiedi_sintomi (**lhs**) con valore "si", allora verrà eseguito un insieme di domande (**rhs**), ciascuna volta per verificare se il paziente ha un determinato sintomo. Se il paziente risponde "no" a tutte le domande, viene impostato il valore "1", indicando che il paziente non presenta i sintomi oggetto delle domande.

GESTIONE DEI FATTI

All'interno di un sistema esperto ci sono quattro operazioni fondamentali che posso essere eseguite su un fatto: dichiarare, modificare, ritrattare e duplicare.

Dichiarare: aggiungere una nuova informazione all'interno del sistema esperto.

```python
@Rule(OR(Fact(molto_dolore="si"), Fact(mancanza_fiato="si")))
def exam_2(self):
    self.declare(Fact(chiedi_esami_frequenzaCardiaca="si"))
```

Nel codice riportato dichiariamo i fatti molto_dolore, mancanza_di fiato.

Modificare: aggiornare le informazioni di un fatto già esistente.

Ritrattare: rimuovere un fatto dalla memoria.

```python
@Rule(Fact(chiedi_esami_pressione="si"))
def ask_pressure_exam(self):
    print("Hai fatto l'esame della pressione sanguigna?")
    response = str(input())

    while valid_response(response) == False:
        print("Hai fatto l'esame della pressione sanguigna?")
        response = str(input())

    if response == "si":
        self.declare(Fact(esame_pressione_eseguito="si"))
    else:
        self.declare(Fact(prescrizione_esame_pressione="no"))
```

I fatti vengono modificati e ritrattati all'interno della regola ask_pressure_exam. Nel caso in cui l'utente risponda si viene dichiarato il fatto esame_pressione_eseguito, nel caso in cui l'utente risponda no viene dichiarato il fatto prescrizione_esame_pressione col fine di far prenotare un esame all'utente.

DUPLICARE: creare una copia di un fatto esistente.

CSP

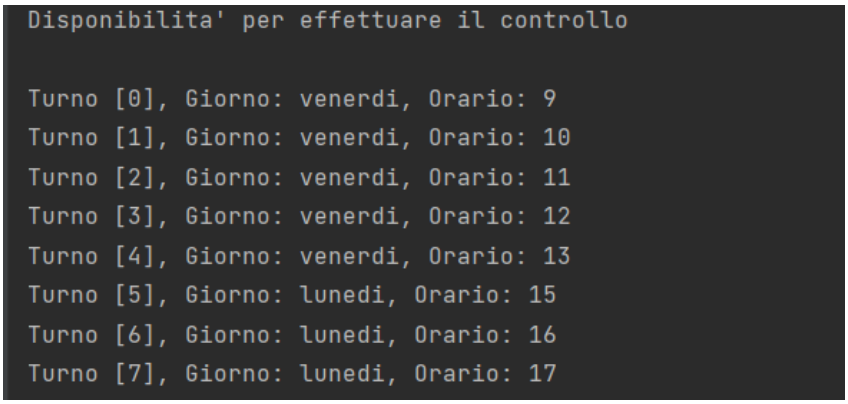
CSP sta per Constraint Satisfaction Problem(Problema di soddisfacimento di Vincoli) ed è un tipo di problema che consiste nel trovare una soluzione che soddisfi un insieme di vincoli. Per risolvere un problema di CSP nell'ambito dell'ingegneria della conoscenza è necessario definire il dominio (*D1, D2…Dn*) delle variabili (*X1,X2,..Xn*) e trovare una soluzione che soddisfi i vari vincoli denominati in inglese constraint(*C1,C2…Cn*).I vincoli rappresentano una condizione o un requisito applicati alle variabili.

Un problema di soddisfacimento di vincoli presuppone un assegnamento iniziale, ovvero un insieme di variabili già vincolate. L'assegnamento iniziale può anche essere vuoto. La risoluzione del problema prosegue estendendo l'assegnamento iniziale, ovvero assegnando via via valori alle variabili ancora libere.

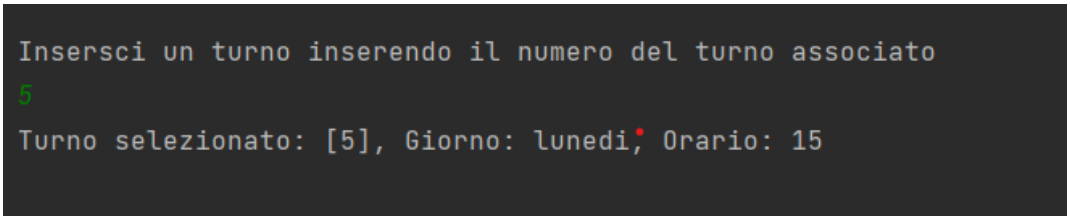
Per realizzare un semplice CSP abbiamo adottato l'utilizzo della libreria **constraint** che fornisce uno strumento per risolvere problemi di ottimizzazione soggetti a vincoli.

Abbiamo definito una classe **Centro Controlli** che estende la classe Problem (definita in constraint). Tale classe serve a risolvere un problema di pianificazione per un centro di controllo.

La classe rappresenta un insieme di variabili che indicano il giorno e l'ora in cui è possibile effettuare un controllo. L'utente, in base alle proprie esigenze, utilizzerà il metodo "get_disponibilità" per pianificare il proprio appuntamento, tenendo conto dei vincoli stabiliti dalle variabili. Questo metodo risolverà il problema di pianificazione in modo efficiente e preciso.



Successivamente saranno stampati a video i turni con annesso giorno ed orario in cui sarà possibile effettuare la prenotazione.



ONTOLOGIA

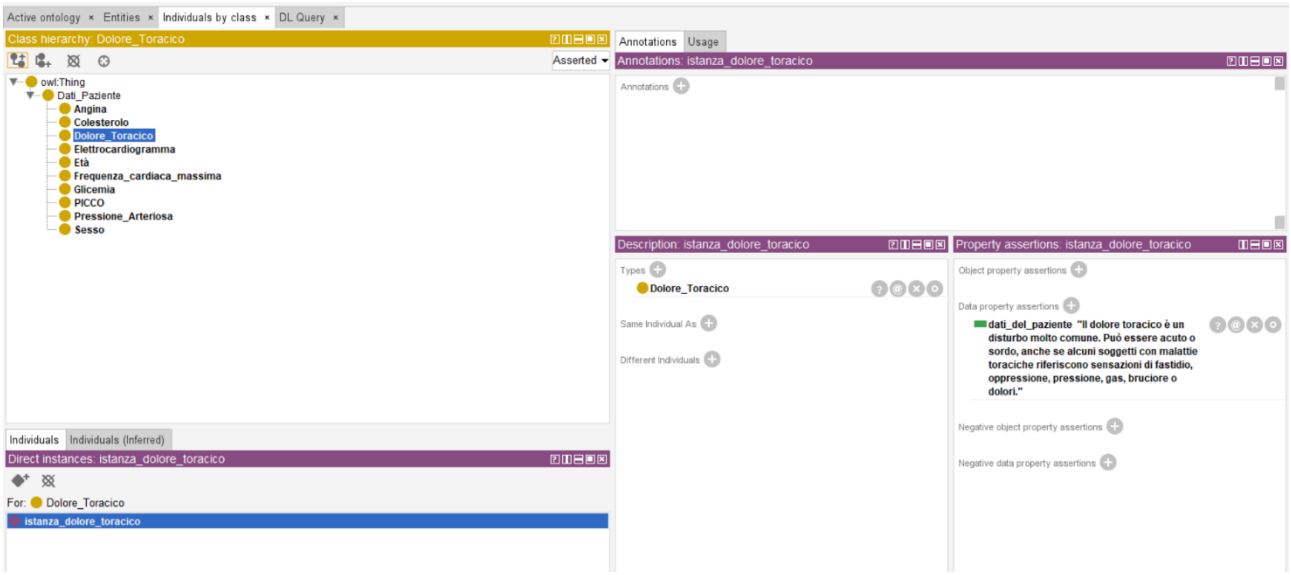
Un'ontologia è una descrizione formale e strutturata di un dominio di conoscenze che rappresenta i concetti le relazioni e le proprietà che caratterizzano quel dominio. È un modello di conoscenza che consente di rappresentare in modo univoco le informazioni relative ad un determinato dominio. Spesso sono utilizzate in ambito informatico per la costruzione di sistemi intelligenti.

La definizione di un'ontologia può essere eseguita mediante un processo di analisi del dominio, in cui si indentificano i concetti principali, le relazioni e le proprietà che caratterizzano il dominio. Successivamente questi concetti, relazioni e proprietà possono essere formalizzate mediante un linguaggio di rappresentazione formale come ad esempio OWL (Ontology Web Language).

OWL è formato da un insieme di costrutti, tra cui classi, proprietà, relazioni e restrizioni. Le classi definiscono i concetti di un determinato dominio e le relazioni tra di essi. Le proprietà definiscono le relazioni tra le classi e le restrizioni sono utilizzate per specificare determinate condizioni che devono essere soddisfatte.

Realizzazione Ontologia

L’ontologia è stata create mediante il software **Protege** che offre un’interfaccia grafica intuitiva.



Grazie all'utilizzo del software **Protege**, sono state create diverse entità corrispondenti ai dati del paziente che emergono durante la visita medica. Ad ogni entità è stata associata un'istanza e una spiegazione dettagliata di esso. In questo modo, è possibile rappresentare in modo accurato e completo le informazioni relative alla salute del paziente, consentendo una gestione efficace e sicura dei dati medici.

L’ontologia è stata create mediante l’utilizzo della libreria **Owlready2** in grado di manipolare e gestire ontologie OWL.

```python
from owlready2 import *
import os


class AttaccoDiCuoreOntology:
    def __init__(self):
        self.ontology = get_ontology(os.path.basename("AttaccoDiCuore.owl")).load()
        self.dati_paziente = {}
```

È stata creata la classe "AttacchiDiCuoreOntology" che ha ricevuto come input il file "AttaccoDiCuore.owl" precedentemente creato. La classe "AttacchiDiCuoreOntology" consente di gestire e manipolare tale ontologia attraverso una serie di metodi e funzioni

GESTIONE DATASET

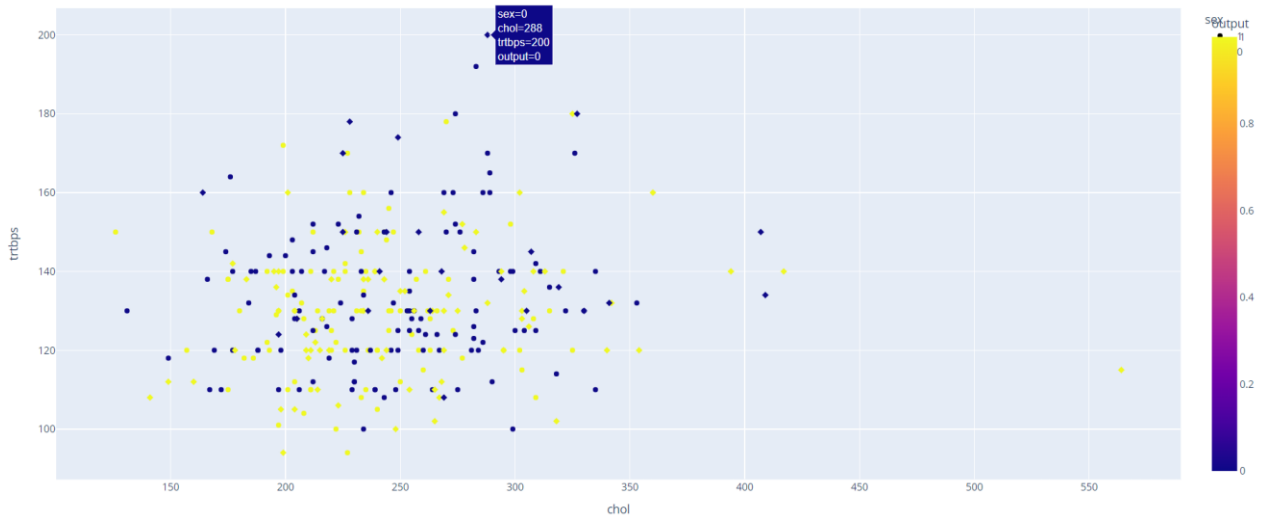
Abbiamo deciso di utilizzare il seguente dataset: https://www.kaggle.com/datasets/rashikrahmanpritom/heart-attack-analysis-prediction-dataset. Tuttavia, abbiamo effettuato una pulizia del dataset eliminando alcune colonne che non erano rilevanti ai fini della nostra predizione.

APPRENDIMENTO NON SUPERVISIONATO

L'apprendimento non supervisionato si occupa di analizzare i dati senza la presenza di un supervisore o di un insegnante esterno. L'obbiettivo è quello di estrarre informazioni utili e significative senza una guida esterna, ciò può includere l'identificazione di cluster di dati.

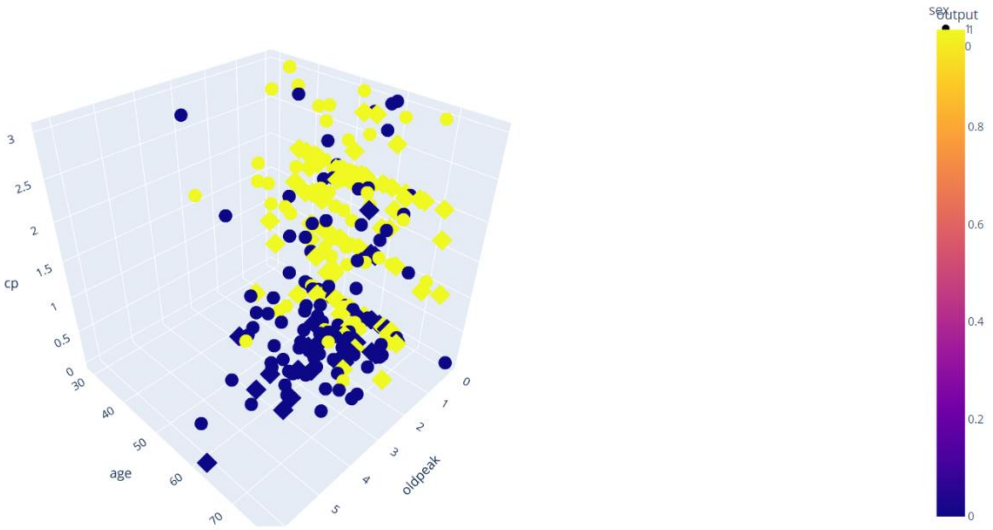
Il clustering è una tecnica di apprendimento automatico non supervisionato che consiste nell'organizzare un insieme di oggetti o dati in gruppi o cluster omogenei.

Abbiamo utilizzato la libreria **Plotly** per la visualizzazione dei dati e la libreria **scikit-learn** per l'implementazione di algoritmi di clustering.



Nel diagramma l'asse delle ascisse è denotato da chol(colesterolo) e l'asse delle ordinate è denotato da trtbps (pressione sanguigna) .L'obbiettivo di tale diagramma

è evidenziare il rapporto tra i due assi mostrando se il paziente ha avuto un attacco cardiaco denotato o meno.



In base alla ripartizione dei dati in cluster, ogni punto viene colorato con un colore differente a seconda dell'appartenenza al cluster corrispondente. In questo modo, il grafico 3D consente di visualizzare e confrontare facilmente i cluster creati, individuando eventuali sovrapposizioni o differenze tra di essi.

K-MEANS

K-means è un algoritmo di clustering che viene utilizzato per raggruppare un insieme di punti in cluster in base alla loro similarità. L'obbiettivo è di trovare k cluster, in cui K è un parametro specificato dall'utente. Tale algoritmo si basa su due passaggi fondamentali:

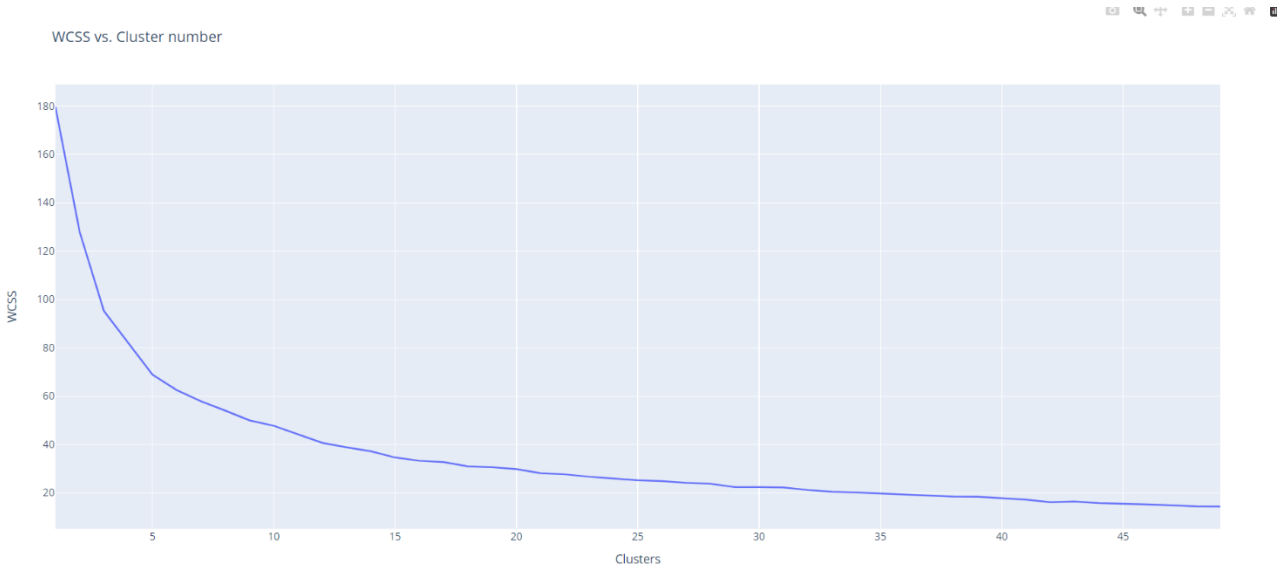
- La fase di assegnazione, l'algoritmo assegna ogni punto al cluster il cui centroide è più vicino
- La fase di aggiornamento dei centroidi, l'algoritmo calcola i nuovi centroidi per ogni cluster

Per realizzare tale algoritmo abbiamo utilizzato il metodo **Elbow**, tale tecnica è utilizzata per determinare un numero ottimale di cluster da utilizzare, ovvero k cluster. Utilizziamo il **WCSS** (Within-Cluster Sum of Square) che permette di misurare la somma dei quadrati delle distanze tra ogni punto del cluster e il centroide del cluster stesso. Esso mostra i valori WCSS(y) corrispondenti ai diversi valori k(x). Se

nel grafico è presente una forma a gomito scegliamo il valore k in cui viene creato il gomito.

```
wcss = []
clusters = 50
for i in range(1, clusters):
    kmeans = KMeans(n_clusters=i, init="k-means++", max_iter=500, n_init=10, random_state=123)
    kmeans.fit(X_train.values)
    wcss.append(kmeans.inertia_)
```

L'inerzia misura quanto bene un set di dati è stato raggruppato in cluster da K-Means. Un buon modello deve avere un'inerzia bassa ed anche un basso numero di cluster. Per trovare il k ottimale utilizziamo il metodo Elbow che trova il punto in cui inizia la diminuzione dell'inerzia.

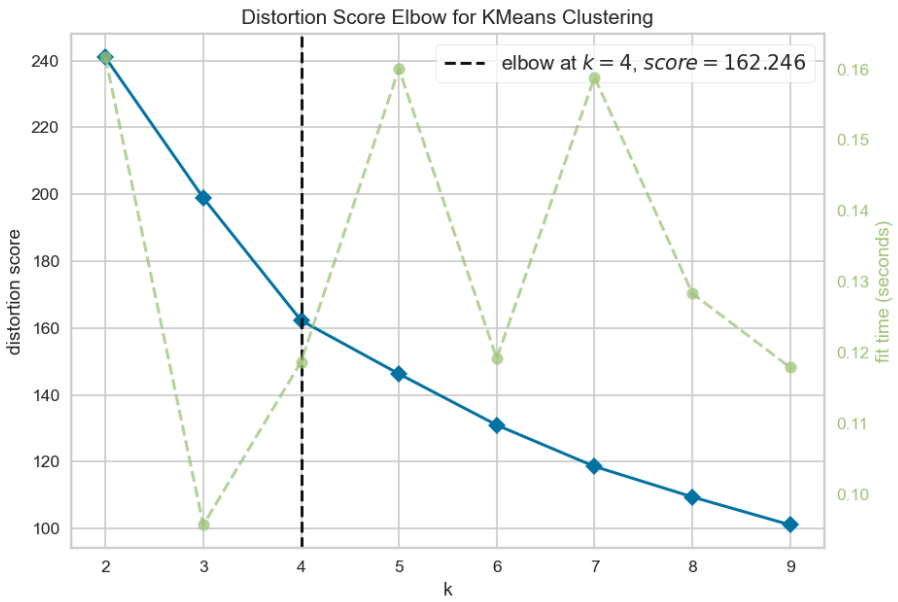


Il grafico permette di individuare il punto del gomito (elbow point), ovvero il numero di cluster ottimale in cui la diminuzione della WCSS inizia a diventare meno significativa.

```
from yellowbrick.cluster import KElbowVisualizer
```

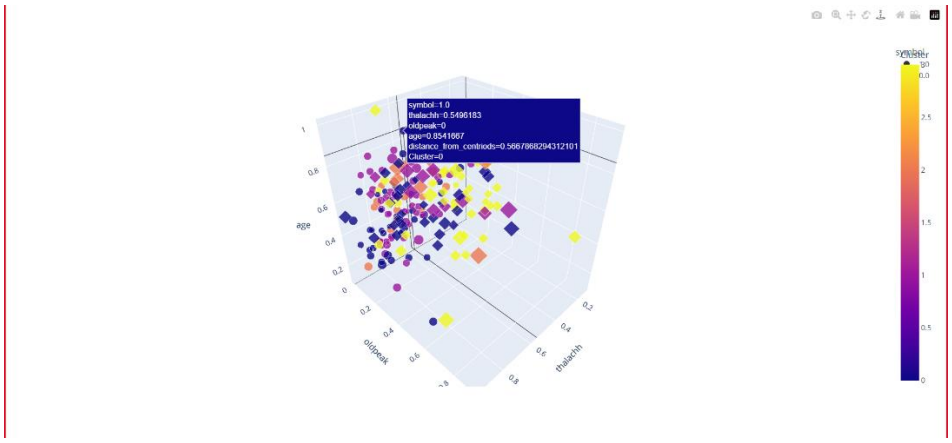
La classe "**KelbowVisualizer**" della libreria "**yellowbrick**" è utilizzata per generare il grafico del gomito, il quale aiuta a identificare il numero ottimale di cluster da utilizzare in un'analisi di clustering

Il grafico ottenuto per la ricerca del k ottimale è il seguente:



Il punto in cui la curva del grafico si piega a formare un gomito indica il numero di cluster ottimale per i dati in analisi. In questo caso il numero ottimale di cluster è pari a 4.

Grafico 3D:



In base alla ripartizione dei dati in cluster, ogni punto viene colorato con un colore differente a seconda dell'appartenenza al cluster corrispondente. In questo modo, il grafico 3D consente di visualizzare e confrontare facilmente i cluster creati, individuando eventuali sovrapposizioni o differenze tra di essi.

```python
for i in range(1,3):
    knn1 = KMeans(i)
    knn1.fit(X_train,y_train)
    pred1 = knn1.predict(X_test)
    print(f"For Knn-{i}: \n")
    print(classification_report(y_test,pred1))
    print('==================================')
```

Il codice esegue un'analisi di classificazione utilizzando l'algoritmo di clustering K-Means, specificando un numero di cluster compreso tra 1 e 3. Successivamente, viene generato un rapporto di classificazione che include le principali metriche di valutazione del modello, come precision, recall e f1-score, per ogni classe di destinazione.

Questo rapporto di classificazione è fondamentale per valutare le prestazioni del modello K-Means nella classificazione dei dati.

In questo modo, è possibile valutare le prestazioni del modello e individuare eventuali aree di miglioramento, ottimizzando i parametri dell'algoritmo di clustering e migliorando l'accuratezza della classificazione.

```
For Knn-1:

              precision    recall  f1-score   support

         0.0       0.42      1.00      0.59        42
         1.0       0.00      0.00      0.00        58

    accuracy                           0.42       100
   macro avg       0.21      0.50      0.30       100
weighted avg       0.18      0.42      0.25       100
```

APPRENDIMENTO SUPERVISIONATO

L’apprendimento supervisionato si occupa di risolvere problemi di previsione e classificazione, in particolare l’apprendimento supervisionato utilizza dati di input e di output etichettati per addestrare il modello a fare previsioni su nuovi dati di input.

Nel nostro progetto abbiamo utilizzato algoritmi di classificazione quali**: K-Nearest Neighbour** e **Logistic Regression.**

In questa parte di codice le librerie utilizzate sono **Numpy** per la computazione scientifica, **Pandas** per la manipolazione e l'analisi dei dati, **MaplotLib e Seaborn** per la creazione e la visualizzazione dei grafici statistici, **Sklearn** per problemi di clussificazione, regressione e clustering.

KNN

KNN è un algoritmo di apprendimento supervisionato utilizzato per la classificazione e per la regressione, classifica un nuovo punto di dati sulla base della sua vicinanza ai punti di dati già classificati. "K" rappresenta il numero di punti di dati più vicini che vengono utilizzati per classificare un nuovo punto di dati.

Senza scalare i dati

L'algoritmo KNN può avere dei problemi nel trovare la distanza tra i punti di dati poiché i dati non scalati possono avere valori molto diversi tra loro .Ciò può portare ad un accuratezza inferiore del modello.

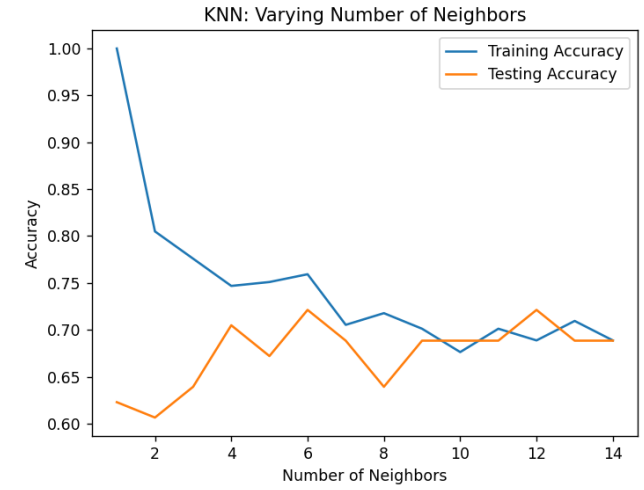
```
# Models
from sklearn.neighbors import KNeighborsClassifier
#pipeline
```

Importiamo la libreria "**scikit-learn**" dalla quale importiamo la classe "**KNeighborsClassifier**" per la ricerca del KNN.

```python
# Crea la variabile dei vicini da assegnare al modello da inserire come iperparametro
neighbors = np.arange(1, 15)

# crea dizionari per memorizzare i risultati
train_accuracies = {}
test_accuracies = {}

for neighbor in neighbors:
    # Set up a KNN Classifier
    knn = KNeighborsClassifier(n_neighbors=neighbor)

    # Fit the model
    knn.fit(X_train, y_train)

    # Compute accuracy
    train_accuracies[neighbor] = knn.score(X_train, y_train)
    test_accuracies[neighbor] = knn.score(X_test, y_test)
```

In questo codice, viene addestrato un modello di machine learning utilizzando i dati di addestramento (X_train, y_train). Successivamente, il modello viene valutato utilizzando le metriche di precisione (accuracy) sui dati di addestramento e sui dati di test (X_test, y_test) per ciascun valore di neighbors.

L'addestramento del modello con diversi valori di neighbors consente di individuare il valore ottimale per questo parametro dell'algoritmo, ottimizzando l'accuratezza del modello e migliorando la sua capacità di generalizzazione.

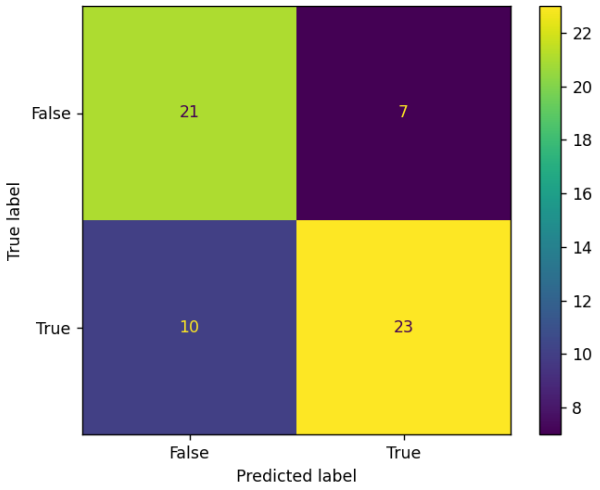


In questo grafico viene rappresentata l'accuratezza del modello di machine learning. Il colore blu rappresenta l'accuratezza del modello sui dati di addestramento (X_train, y_train), mentre il colore arancione rappresenta l'accuratezza del modello sui dati di test (X_test, y_test).

L'accuratezza sui dati di addestramento fornisce un'indicazione delle prestazioni del modello sui dati utilizzati per l'addestramento. Il grafico consente di confrontare le prestazioni del modello sui dati di addestramento e sui dati di test, evidenziando eventuali differenze tra i due set di dati.

Matrice di confusione

Una matrice di confusione è una matrice 2x2 che riassume le prestazioni di un modello di classificazione, mette in evidenza gli errori del modello, le istanze in cui risponde meglio e quelle in cui risponde peggio.



In questo caso abbiamo 21 casi true-negative ovvero non hanno avuto un attacco cardiaco ,7 casi false-positive,23 casi true-positive ovvero hanno avuto un attacco cardiaco ed infine 10 casi false-negative ovvero non sono stati predetti correttamente.

In generale la matrice di confusione fornisce informazioni importanti sulla precisione ed il richiamo del modello.

```
0.7213114754098361
              precision    recall  f1-score   support

           0       0.68      0.75      0.71        28
           1       0.77      0.70      0.73        33

    accuracy                           0.72        61
   macro avg       0.72      0.72      0.72        61
weighted avg       0.73      0.72      0.72        61
```

Precision: è il numero di veri positivi diviso la somma di tutte le predizioni positive. Nel nostro caso 0=0.68 ed 1=0.77.

Recall: è il numero di veri positivi diviso la somma dei veri positivi ed i falsi negativi. Nel nostro caso 0=0.75 ed 1=0.70.

F1-score: è la media armonica di precision e recall, assegna un peso uguale a precisione e richiamo tenendo conto sia del numero di errori commessi nel modello sia del tipo di errori. Nel nostro caso 0=0.71 ed 1=0.73

Scalare i dati

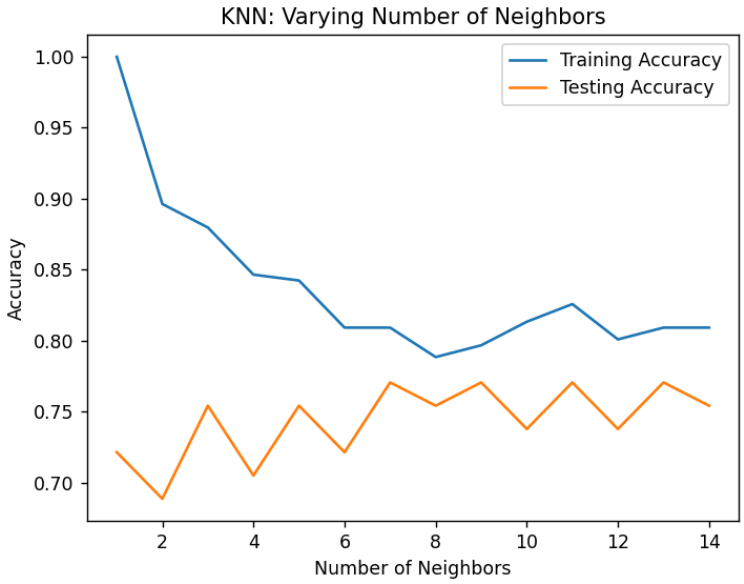
La scalatura dei dati è utile per migliorare le prestazioni dell'algoritmo KNN perché garantisce che tutte le variabili abbiano lo stesso peso nella determinazione delle distanze tra le istanze di dati.

```
for neighbor in neighbors:
    # Set up a KNN Classifier in pipeline
    knn_std_mdl = make_pipeline(StandardScaler(), KNeighborsClassifier(n_neighbors=neighbor))
```

Utilizziamo il trasformatore **StandarsScaler()** che fa parte della libreria **scikit-learn** per standardizzare i dati di input. Questo passo è importante perché l'algoritmo KNN calcola le distanze tra i punti di dati e , se le scale delle variabili sono diverse, alcune variabili potrebbero avere un peso maggiore .

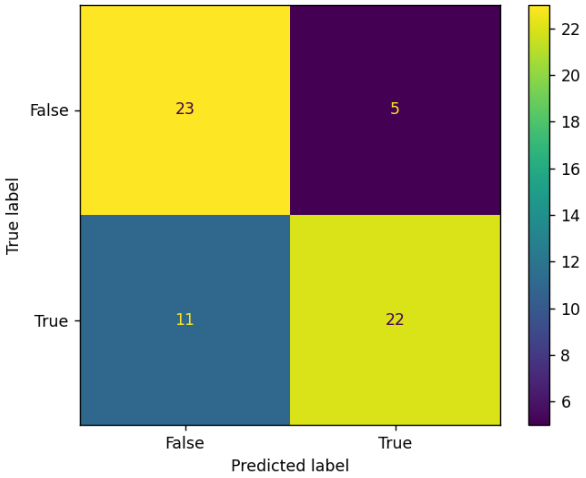
```
# Display the plot
plt.show()
print(max(test_accuracies, key=test_accuracies.get))

knn_std_mdl = make_pipeline(StandardScaler(), KNeighborsClassifier(n_neighbors=12))

# Fit the classifier to the training data
knn_std_mdl.fit(X_train, y_train)

# Print the accuracy
print(knn_std_mdl.score(X_test, y_test))
```

Questo codice definisce un modello di classificazione KNN standardizzato con un numero di vicini più prossimi pari a 12,utilizzando i dati di addestramento (X_train,y_train) ed utilizza i dati di test(X_test,y_test) ed infine visualizza un grafico delle accuratezze del modello.



In questo grafico viene rappresentata l'accuratezza del modello di machine learning. Il colore blu rappresenta l'accuratezza del modello sui dati di addestramento (X_train, y_train), mentre il colore arancione rappresenta l'accuratezza del modello sui dati di test (X_test, y_test).

Matrice di confusione



In questo caso abbiamo 23 casi true-negative ovvero non hanno avuto un attacco cardiaco ,5 casi false-positive,22 casi true-positive ovvero hanno avuto un attacco cardiaco ed infine 11 casi false-negative ovvero non sono stati predetti correttamente.

In generale la matrice di confusione fornisce informazioni importanti sulla precisione ed il richiamo del modello.

```
0.7377049180327869
              precision    recall  f1-score   support

           0       0.68      0.82      0.74        28
           1       0.81      0.67      0.73        33

    accuracy                           0.74        61
   macro avg       0.75      0.74      0.74        61
weighted avg       0.75      0.74      0.74        61
```

Precision: è il numero di veri positivi diviso la somma di tutte le predizioni positive. Nel nostro caso 0=0.68 ed 1=0.81.

Recall: è il numero di veri positivi diviso la somma dei veri positivi ed i falsi negativi. Nel nostro caso 0=0.82 ed 1=0.67.

F1-score: è la media armonica di precision e recall, assegna un peso uguale a precisione e richiamo tenendo conto sia del numero di errori commessi nel modello sia del tipo di errori. Nel nostro caso 0=0.74 ed 1=0.73

LOGISTIC REGRESSION

La **regressione logistica** è un algoritmo di apprendimento utilizzato per modellare la relazione tra una variabile dipendente binaria (assume valore 0 o 1) ed una o più variabili indipendenti continue o categoriche. L'obbiettivo della regressione logistica è quello di stimare i coefficienti di regressione in modo tale da poter predire la probabilità che la variabile dipendente assuma un valore binario in funzione dei valori delle variabili indipendenti.

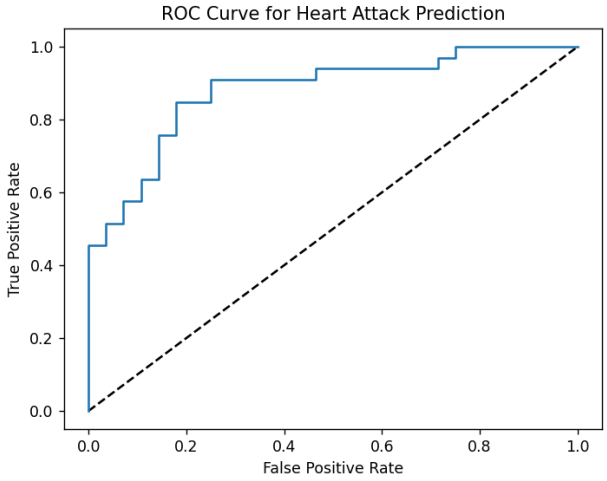
```
##REGRESSIONE LOGISTICA

# Import LogisticRegression
from sklearn.linear_model import LogisticRegression
```

Nel codice la classe "**LogisticRegression**" importata dalla libreria **sklearn** viene utilizzata per addestrare modelli di regressione logistica sui dati.

```
# Import LogisticRegression
from sklearn.linear_model import LogisticRegression

# Instantiate the model
logreg = LogisticRegression()

# Fit the model
logreg.fit(X_train, y_train)

# Predict probabilities
y_pred_probs = logreg.predict_proba(X_test)[:, 1]

# Import roc_curve
from sklearn.metrics import roc_curve

# Genera i valori della curva ROC: fpr, tpr, soglie
fpr, tpr, thresholds = roc_curve(y_test, y_pred_probs)

plt.plot([0, 1], [0, 1], 'k--')

# Plot tpr against fpr
plt.plot(fpr, tpr)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for Heart Attack Prediction')
plt.show()
```

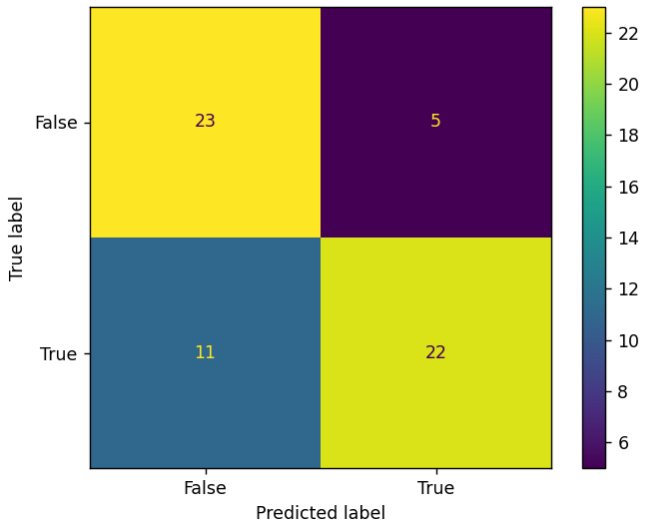
Il seguente codice ha lo scopo di valutare le prestazioni del modello di regressione logistica addestrato attraverso la curva **ROC** (Receiver Operating Characteristic). La curva ROC è una rappresentazione grafica che mostra la capacità del modello di distinguere tra le classi positive e negative in un problema di classificazione binaria.

Per generare i valori della curva ROC, viene importato il modulo "**roc_curve**" dalla libreria "**sklearn.metrics**".



Successivamente, viene generata la curva ROC utilizzando i valori di tasso di falsi positivi (False Positive Rate, FPR) sull'asse x e di tasso di veri positivi (True Positive Rate, TPR) sull'asse y. Viene anche tracciata la retta diagonale (linea tratteggiata nera) che rappresenta la linea di base che indica le prestazioni casuali.

Matrice di confusione



In questo caso abbiamo 23 casi true-negative ovvero non hanno avuto un attacco cardiaco ,5 casi false-positive,22 casi true-positive ovvero hanno avuto un attacco cardiaco ed infine 11 casi false-negative ovvero non sono stati predetti correttamente.

Come possiamo notare abbiamo ottenuto gli stessi risultati della matrice di confusione per KNN con valori scalari. Se il dataset di input e le condizioni sperimentali sono gli stessi, la regressione logistica e il KNN possono produrre matrici di confusione simili o identiche.

```
roc_auc_score: 0.8798701298701299
accuracy: 0.819672131147541
```

Questi sono i risultati ottenuti per quanto riguarda il ROC score e l'accuratezza. In questo caso i valori di **precision**, **recall** e **f1-score** saranno uguali alla matrice di confusione per KNN con valori scalari come definito precedentemente.

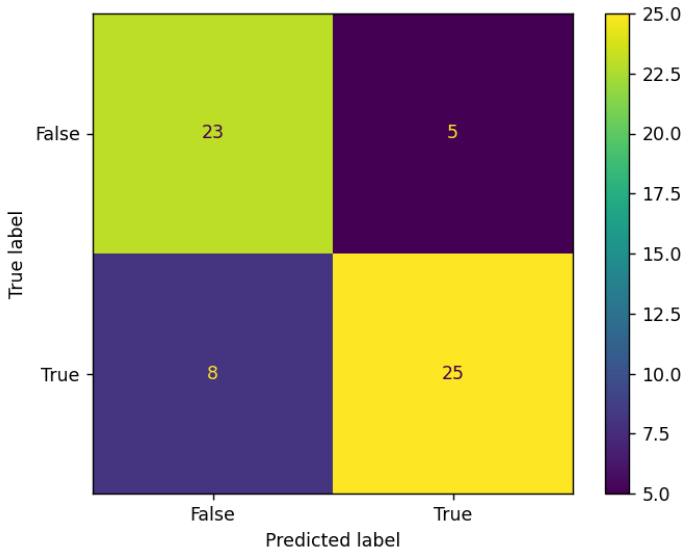
Standardizzazione

Standardizzare le variabili di input nella regressione logistica è importante per evitare problemi di scale poiché le variabili di input possono avere scale molto diverse tra di loro ed è importante poiché migliora la precisione del modello di regressione logistica.

```
logreg_std = make_pipeline(StandardScaler(), LogisticRegression())
```

Nel codice la standardizzazione avviene tramite "**StandardScaler()"** metodo di preprocissing nella libreria **scikit-learn**.

Matrice di confusione



In questo caso abbiamo 23 casi true-negative ovvero non hanno avuto un attacco cardiaco ,5 casi false-positive,25 casi true-positive ovvero hanno avuto un attacco cardiaco ed infine 8 casi false-negative ovvero non sono stati predetti correttamente.

```
roc_auc_score: 0.8798701298701299
accuracy: 0.7868852459016393
              precision    recall  f1-score   support

           0       0.74      0.82      0.78        28
           1       0.83      0.76      0.79        33

    accuracy                           0.79        61
   macro avg       0.79      0.79      0.79        61
weighted avg       0.79      0.79      0.79        61
```

Questi sono i risultati ottenuti per quanto riguarda il ROC score e l'accuratezza.

Precision: è il numero di veri positivi diviso la somma di tutte le predizioni positive. Nel nostro caso 0=0.74 ed 1=0.83.

Recall: è il numero di veri positivi diviso la somma dei veri positivi ed i falsi negativi. Nel nostro caso 0=0.82 ed 1=0.76.

F1-score: è la media armonica di precision e recall, assegna un peso uguale a precisione e richiamo tenendo conto sia del numero di errori commessi nel modello sia del tipo di errori. Nel nostro caso 0=0.78 ed 1=0.79