Sistema diagnostica diabete

Davide Cellura, Matricola:721817

Email: d.cellura@studenti.uniba.it

Anno Accademico 2023-2024

Repository Github: https://github.com/DavideCellura/Progetto_ICON_2024

Indice

1:Introduzione

Il diabete è una malattia cronica caratterizzata da livelli elevati di glucosio (zucchero) nel sangue. Questo squilibrio è dovuto a un'alterazione nella produzione o nell'azione dell'insulina, un ormone prodotto dal pancreas che regola la quantità di glucosio nel sangue. Esistono principalmente due tipi di diabete:

1. **Diabete di tipo 1**: È una malattia autoimmune in cui il sistema immunitario attacca e distrugge le cellule beta del pancreas che producono insulina. Di conseguenza, l'organismo non è in grado di produrre insulina, rendendo necessaria la somministrazione di insulina dall'esterno.
2. **Diabete di tipo 2**: È la forma più comune e si verifica quando il corpo non utilizza efficacemente l'insulina (insulino-resistenza) o non ne produce abbastanza. Questo tipo è spesso associato a fattori di rischio come obesità, inattività fisica e dieta inadeguata.

Il diabete può portare a complicazioni gravi, come malattie cardiovascolari, danni ai reni, neuropatie e problemi alla vista, se non viene adeguatamente gestito.

Utilizzerò due moduli per la diagnostica del diabete:

- Un agente intelligente (rule-based) che tiene conto delle risposte dell’ utente per la diagnostica del diabete.
- L’ utilizzo di algoritmi di apprendimento supervisionato per la diagnostica del diabete.

2: Knowledge base

Per la diagnostica del diabete ho deciso di creare un sistema esperto, per fare ciò ho utilizzato la libreria python “Experta”.

Un sistema esperto è un programma informatico progettato per simulare il processo decisionale di un esperto umano in un determinato campo. Utilizza una base di conoscenze, che raccoglie informazioni ed esperienze specifiche, e un motore di inferenza, che applica regole logiche per risolvere problemi complessi o fornire consigli, proprio come farebbe un esperto umano.

In sintesi, un sistema esperto è uno strumento che "pensa" come un esperto in un particolare dominio, aiutando gli utenti a prendere decisioni informate.

2.1 Funzionamento Experta

“Experta” si basa su **fatti** e **regole**.

- **Fatti (Facts):** Le unità di informazione su cui vengono applicate le regole. Ogni fatto può essere visto come un insieme di attributi che descrivono una certa situazione.
- **Regole (Rules):** Dichiarazioni condizionali che descrivono un'azione da intraprendere quando certe condizioni sui fatti sono soddisfatte. Ogni regola è composta da:

  - LHS (sinistro): descrive le condizioni base alle quali la regola dovrebbe essere eseguita.

  - RHS (destro): è l’insieme delle azioni da eseguire quando la regola viene eseguita.

Ogni regola, affinchè possa essere utilizzata, deve essere un metodo di una sottoclasse KnowledgeEngine.

2.2 Gestione dei fatti

Un **fatto** può essere:

- Dichiarato: Aggiunta di un nuovo fatto all’elenco dei fatti.

- Ritrattato: Rimozione di un fatto esistente dall’elenco dei fatti.

- Modificato: Ritiro di un fatto e aggiunta del medesimo fatto modificato.

- Duplicato: Aggiunta di un nuovo fatto all’elenco utilizzando un fatto esistente come modello inserendo alcune modifiche.

2.3 Funzionamento sistema esperto

Durante l'esecuzione del sistema esperto l'utente è sottoposto ad una serie di domande attraverso il quale il sistema dichiara dei facts. Basandosi sulle regole il sistema sarà capace di fare una diagnosi del diabete.

2.4 Dataset

Per realizare il sistema ho utilizzato il seguente dataset:

https://www.kaggle.com/datasets/iammustafatz/diabetes-prediction-dataset

Il dataset contiene diverese informazioni come:

- ***gender***: in questa colonna è indicato il sesso dei pazienti (maschio o femmina)
- ***age***: in questa colonna è riportata l'età del paziente
- ***hypertension***: colonna in cui è riportato se il paziente ha avuto problemi di ipertensione (0=no , 1=si)
- ***heart_disease***: colonna in cui è riportato se il paziente ha avuto malattie al cuore (0=no , 1=si)
- ***smoking_history***: questa colonna sta ad indicare se il paziente ha precedenti con il fumare.
- ***bmi***: Il BMI (Indice di Massa Corporea) è un indicatore che misura la relazione tra il peso e l'altezza di una persona. Si calcola dividendo il peso in chilogrammi per il quadrato dell'altezza in metri. Viene usato per stimare se una persona è sottopeso, normopeso, sovrappeso o obesa.
- ***HbA1c_level***: Il livello di HbA1c indica la quantità di emoglobina glicata nel sangue, riflettendo la media della glicemia negli ultimi 2-3 mesi. È usato per monitorare il controllo a lungo termine del diabete.
- ***blood_glucose_level*****:** Il livello di glucosio nel sangue si riferisce alla quantità di glucosio nel flusso sanguigno in un dato momento. Livelli elevati di glucosio nel sangue sono un indicatore chiave del diabete.
- ***diabetes:*** è la variabile target da prevedere, con valori pari a 1 che indica la presenza di diabete, 0 che ne indica l' assenza.

2.5 Ontologia

In informatica, un'ontologia è una rappresentazione formale, condivisa ed esplicita di una concettualizzazione di un dominio di interesse. Il termine ontologia formale è entrato in uso nel campo dell'intelligenza artificiale e della rappresentazione della conoscenza, per descrivere il modo in cui diversi schemi vengono combinati in una struttura dati contenente tutte le entità rilevanti e le loro relazioni in un dominio.

Le ontologie distribuite sulla rete vengono descritte attraverso il linguaggio OWL (che sta per Web Ontology Language) che permette di descrivere il mondo in termini di:

- **Individui**: entità del mondo

- **Classi**: insieme di individui, accomunati da delle caratteristiche comuni

- **Proprietà**: relazioni che associano agli elementi del proprio dominio (individui), un elemento del codominio (valori che può assumere).

In particolare, si dividono in;

- **Datatype property**: se il codominio contiene solo tipi primitivi (interi, stringhe, …)

- **Object property**: quando il codominio contiene altre classi

La strutturazione dell'ontologia è avvenuta tramite il software **protège.**

Con **protège** ho creato la ontologia sul diabete definendo le classi e individui corrispondenti a:

- **Sintomo_diabete**: classe avente come sottoclassi tutti i sintomi del diabete più comuni, i quali contengono l'istanza dell'individuo collegata al sintomo, con una asserzione di proprietà di dati sulla descrizione del sintomo.
- **Malattie_correlate** : classe avente come sottoclasse tutte le malattie correlate al diabete, nei quali ci sono l'istanza dell'individuo collegata alla malattia correlata, con asserzione di proprietà di dati sulla descrizione della malattia.
- **Trattamenti malattie correlate**: sono diverse sottoclassi per ogni malattia correlata, le quali hanno l'istanza dell'individuo collegata al trattamento della malattia correlata, con asserzione di proprietà di dati sul possibile trattamento della malattia correlata.

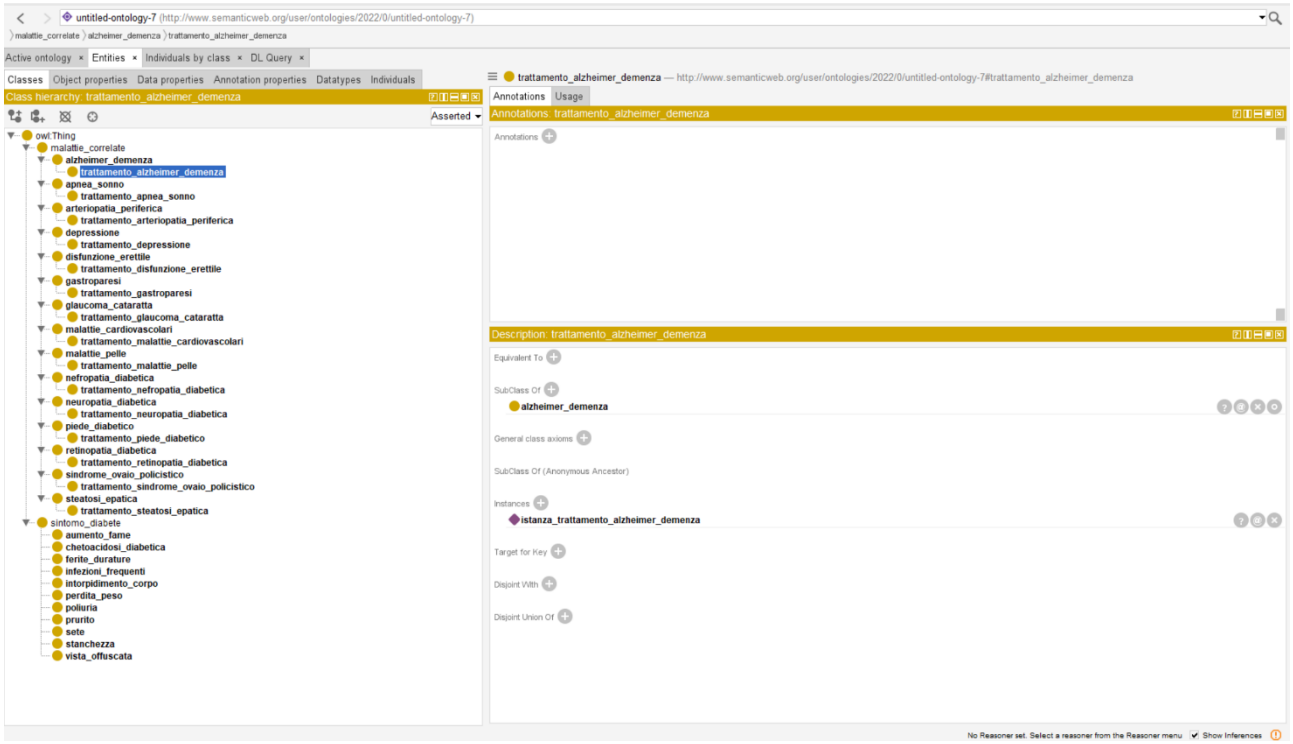


Figura 2.5.1: classi ontologia (Protege)

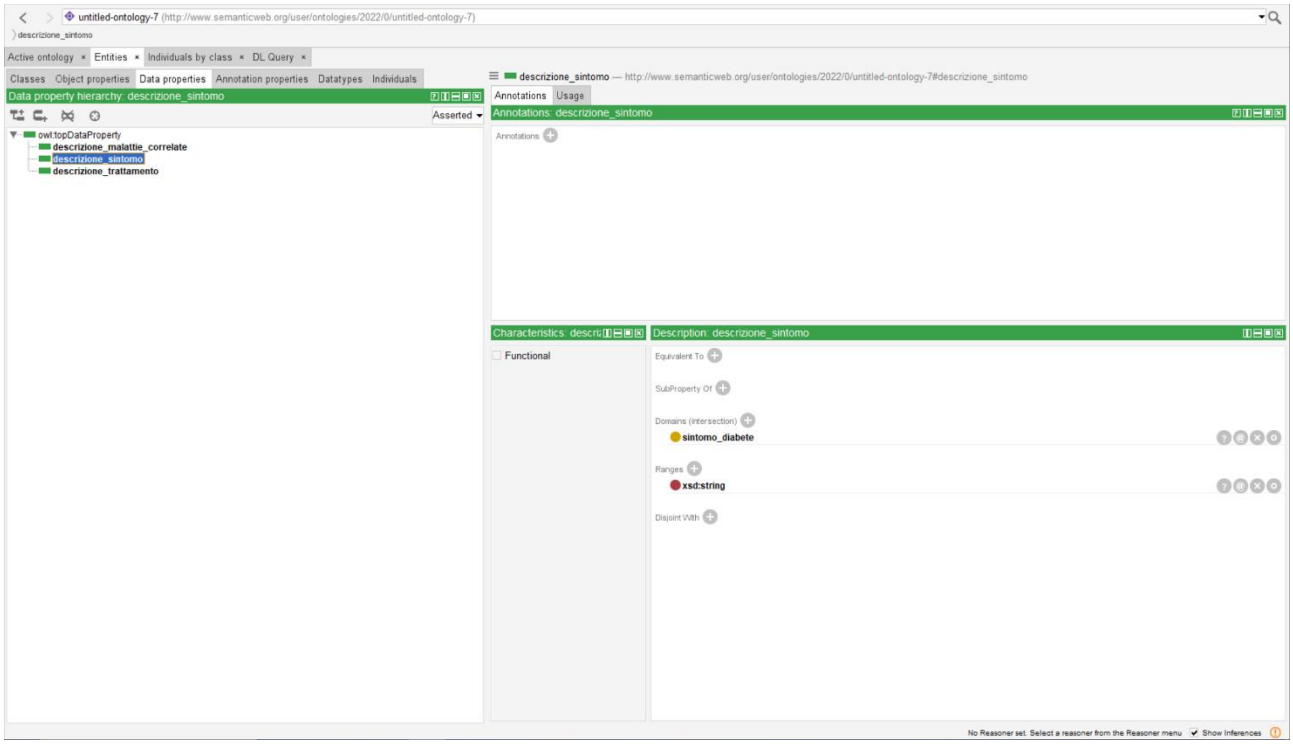


Figura 2.5.2: Proprietà dei dati (Protege)

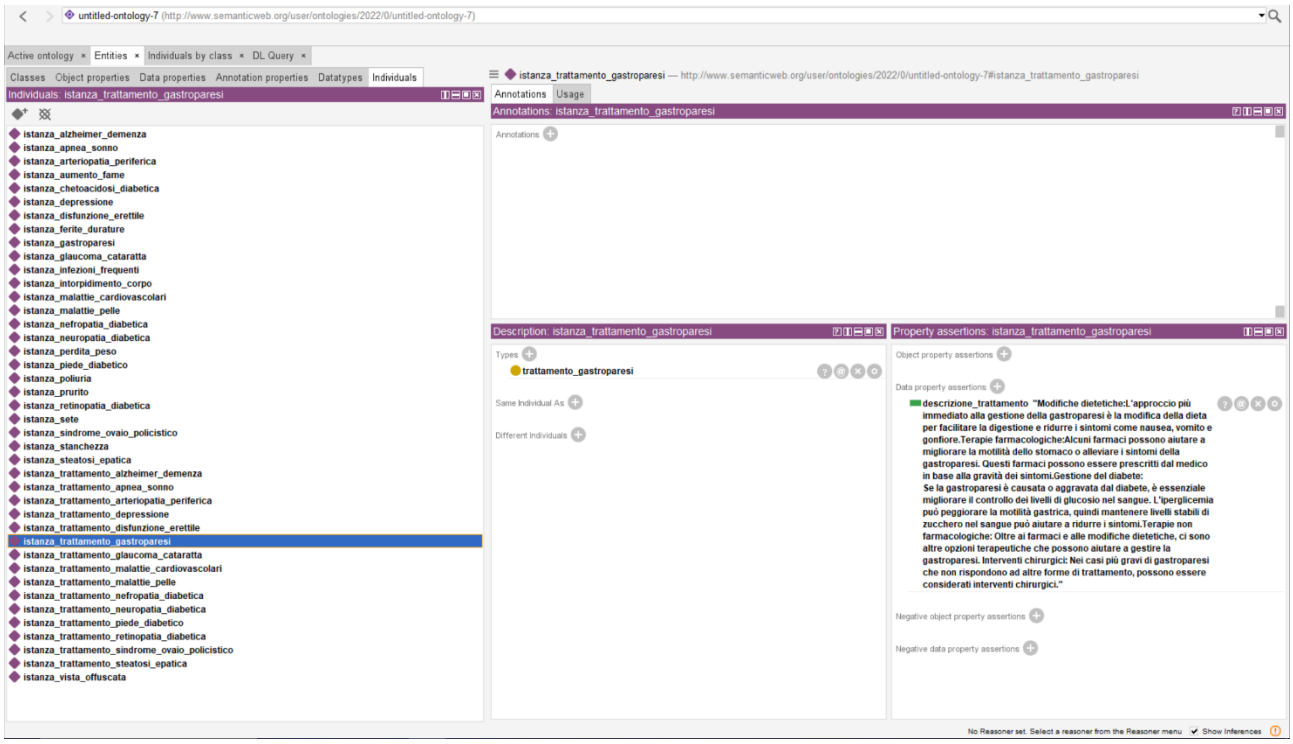


Figura 2.5.3: Individui ontologia (Protege)

Successivamente ho creato “sintomi_diabete.owl” in modo da gestire l’ontologia tramite la libreria **Owlready2** presente in python.

Usando l’ontologia abbiamo 3 funzionalità:

1. La visione dei vari sintomi del diabete e la possibilità di scegliere di quale sintomo vogliamo sapere la descrizione:

```
Benvenuto in Esperto Diabete, un sistema esperto per la diagnosi del diabete
----------->MENU<-----------
[1] Mostra i possibili sintomi del diabete
[2] Esegui una diagnosi
[3] Mostra le malattie correlate
[4] Trattamenti possibili di malattie correlate
[5] Esci
1
Sintomo [1]: Nome: aumento_fame
Sintomo [2]: Nome: chetoacidosi_diabetica
Sintomo [3]: Nome: ferite_durature
Sintomo [4]: Nome: infezioni_frequenti
Sintomo [5]: Nome: intorpidimento_corpo
Sintomo [6]: Nome: perdita_peso
Sintomo [7]: Nome: poliuria
Sintomo [8]: Nome: prurito
Sintomo [9]: Nome: sete
Sintomo [10]: Nome: stanchezza
Sintomo [11]: Nome: vista_offuscata

Seleziona il sintomo di cui vuoi conosere la descrizione, inserisci il numero del sintomo
7
Sintomo: poliuria, descrizione: La poliuria è un sintomo caratterizzato da un aumento anomalo della produzione di urina. Tecnicamente, viene definita come l'e
screzione di oltre 2,5-3 litri di urina al giorno in un adulto. In condizioni normali, un adulto urina circa 1-2 litri al giorno, quindi la poliuria rappresen
ta un aumento significativo di questo volume.
```

Figura 2.5.4: Funzionalità mostra i sintomi

2. La visione delle malattie correlate e la possibilità di scegliere di quale malattia vogliamo sapere la descrizione:

```
----------->MENU<-----------
[1] Mostra i possibili sintomi del diabete
[2] Esegui una diagnosi
[3] Mostra le malattie correlate
[4] Trattamenti possibili di malattie correlate
[5] Esci
3
Sintomo [1]: Nome: alzheimer_demenza
Sintomo [2]: Nome: apnea_sonno
Sintomo [3]: Nome: arteriopatia_periferica
Sintomo [4]: Nome: depressione
Sintomo [5]: Nome: disfunzione_erettile
Sintomo [6]: Nome: gastroparesi
Sintomo [7]: Nome: glaucoma_cataratta
Sintomo [8]: Nome: malattie_cardiovascolari
Sintomo [9]: Nome: malattie_pelle
Sintomo [10]: Nome: nefropatia_diabetica
Sintomo [11]: Nome: neuropatia_diabetica
Sintomo [12]: Nome: piede_diabetico
Sintomo [13]: Nome: retinopatia_diabetica
Sintomo [14]: Nome: sindrome_ovaio_policistico
Sintomo [15]: Nome: steatosi_epatica

Seleziona la malattia correlata di cui vuoi conosere la descrizione, inserisci il numero della malattia
6
Malattia: gastroparesi, descrizione: Un rallentamento della digestione dovuto a danni ai nervi che controllano lo stomaco.
```

Figura 2.5.5: Funzionalità mostra le malattie correlate

3. La ricerca del possibile trattamento di una malattia correlata tramite nome della malattia correlata.

```
----------->MENU<-----------
[1] Mostra i possibili sintomi del diabete
[2] Esegui una diagnosi
[3] Mostra le malattie correlate
[4] Trattamenti possibili di malattie correlate
[5] Esci
4

Scrivi il nome della malattia correlata di cui vuoi sapere i possibili trattamenti :
gastroparesi
["Modifiche dietetiche:L'approccio più immediato alla gestione della gastroparesi è la modifica della dieta per facilitare la digestione e ridurre i sintomi come nausea
, vomito e gonfiore.Terapie farmacologiche:Alcuni farmaci possono aiutare a migliorare la motilità dello stomaco o alleviare i sintomi della gastroparesi. Questi farmac
i possono essere prescritti dal medico in base alla gravità dei sintomi.Gestione del diabete: Se la gastroparesi è causata o aggravata dal diabete, è essenziale miglior
are il controllo dei livelli di glucosio nel sangue. L'iperglicemia può peggiorare la motilità gastrica, quindi mantenere livelli stabili di zucchero nel sangue può aiu
tare a ridurre i sintomi.Terapie non farmacologiche: Oltre ai farmaci e alle modifiche dietetiche, ci sono altre opzioni terapeutiche che possono aiutare a gestire la g
astroparesi. Interventi chirurgici: Nei casi più gravi di gastroparesi che non rispondono ad altre forme di trattamento, possono essere considerati interventi chirurgic
i."]
```

Figura 2.5.6: Funzionalità ricerca trattamento malattie correlate

2.6 CSP (Constraint Satisfaction Problem)

Molti problemi nell'ambito dell'Intelligenza Artificiale sono classificabili come Problemi di Soddisfacimento di Vincoli (Constraint Satisfaction Problem o CSP); Formalmente, un CSP può essere definito su un insieme finito di variabili (X1,X2, . . . ,Xn) i cui valori appartengono a domini finiti di

definizione(D1,D2, . . . ,Dn) e su un insieme di vincoli(C1,C2, . . . ,Cn). Un vincolo su un insieme di variabili è una restrizione dei valori che le variabili possono assumere simultaneamente. Concettualmente, un vincolo può essere visto come un insieme che contiene tutti i valori che le variabili possono assumere contemporaneamente: un vincolo tra k variabili C(Xi1,Xi2, . . . ,Xik), è un sottoinsieme del prodotto cartesiano dei domini delle variabili coinvolte Di1 ,Di2 , . . . ,Dik che specifica quali valori delle variabili sono compatibili con le altre. Questo insieme può essere rappresentato in molti modi, come ad esempio, matrici, equazioni, disuguaglianze o relazioni.

La libreria che ho utilizzato è “constraint” che mi ha permesso di realizzare un semplice CSP (è un caso molto semplice, non si tiene conto di altri utenti) in grado di mostrare la disponibilità dello studio medico convenzionato, nel caso in cui il sistema preveda la prescrizione delle analisi del sangue. Il funzionamento è il seguente:

• Alla base vi è una sottoclasse di Problem (classe già definita in constraint, che modella un CSP).

• Vengono aggiunte variabili con proprio dominio associato in maniera esplicita.

• In base alle risposte dell’ utente, il sistema decide se prescrivere o meno delle analisi.

• Se il sistema decide di prescrivere delle analisi, allora qui entra in gioco il CSP.

• Il CSP è relativo agli orari e giorni di apertura del laboratorio delle analisi.

• Per esempio: il laboratorio per l’analisi della glicemia nel sangue, fa le analisi dalle ore 8 alle 14 di lunedì.

• Il sistema quindi indica i possibili orari a cui l’ utente può presentarsi per sottoporsi alle analisi.

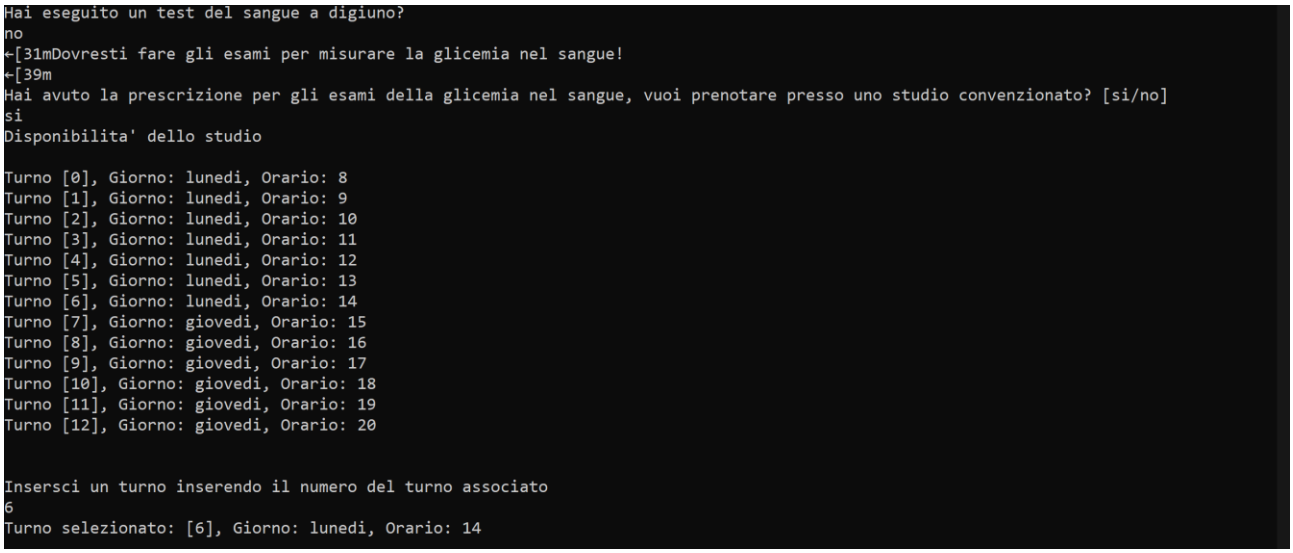


Figura 2.6.1: Esempio di prenotazione degli esami della glicemia nel sangue.

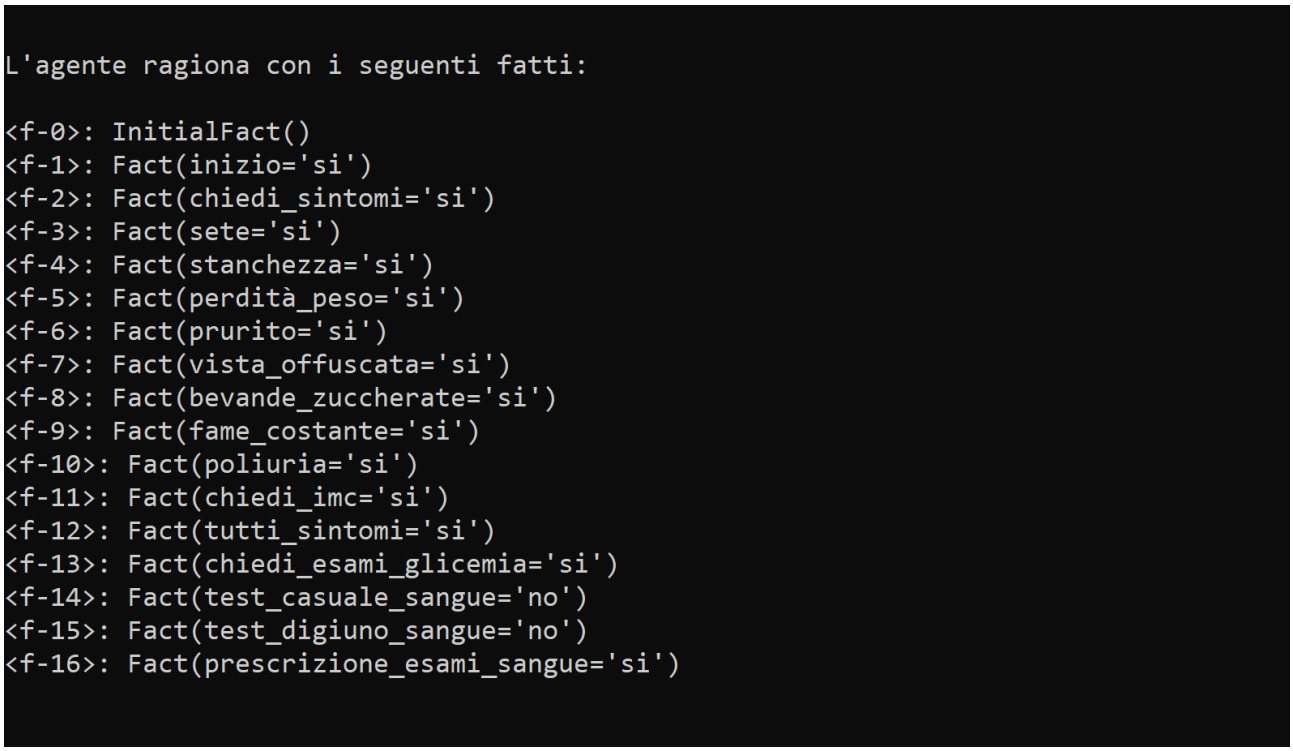


Figura 2.6.2: In base alle risposte fornite dall'utente, l'agente ragiona con i seguenti fatti

3 Apprendimento Supervisionato

In questa sezione di progetto sono stati implementati diversi algoritmi di Apprendimento Supervisionato appartenenti, in modo specifico, alla categoria degli Algoritmi di Classificazione.

Gli algoritmi utilizzati sono:

• Regressione logistica

• Albero di decisione

• K-nearest neighbor

3.1 Implementazione dei modelli di apprendimento supervisionato in sklearn

La libreria scelta per l' implementazione di questi algoritmi è sklearn, che permette tramite le classi e i metodi presenti di implementare, allenare e testare i modelli di apprendimento supervisionato.

1. Nel main (main_modello.py), per ogni modello si istanzia la relativa classe.

2. Per ogni modello viene eseguita la predizione.

3. Successivamente viene mostrata la matrice di confusione.

4. Vengono quindi calcolate le metriche (accuratezza, precision, recall, F1_precision).

5. Il calcolo di queste metriche avviene tramite l' ausilio di metodi messi a disposizione da Sklearn.

Le metriche che vengono utilizzate sono:

1. **Recall**: Misura la capacità del modello di identificare correttamente le istanze positive. Rappresenta la proporzione di veri positivi sul totale delle effettive classi positive.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

2. **Precision**: Misura la qualità delle predizioni positive del modello, ovvero la proporzione di veri positivi sul totale delle predizioni positive.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

3. **Accuracy**: Rappresenta la proporzione di tutte le predizioni corrette (sia positive che negative) sul totale delle istanze valutate.

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

4. **F1-score**: È la media armonica tra precision e recall, utilizzata per bilanciare questi due aspetti in presenza di un dataset con classi sbilanciate.

$$F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

3.2 Regressione logistica

La regressione logistica è un modello statistico usato negli algoritmi di classificazione del machine learning per ottenere la probabilità di appartenenza a una determinata classe. L'algoritmo si basa sull'utilizzo della funzione logistica (sigmoid).Il risultato della funzione sigmoide è un valore compreso tra 0 e 1, che può essere interpretato come la probabilità che l'osservazione appartenga a una classe specifica (solitamente la classe "positiva").

- Se la probabilità calcolata è superiore a una certa soglia (tipicamente 0,5), l'osservazione viene classificata come appartenente alla classe positiva.
- Se la probabilità è inferiore alla soglia, l'osservazione viene classificata come appartenente alla classe negativa.

Nella fase di addestramento l'algoritmo riceve in input un dataset di training composta da N esempi.

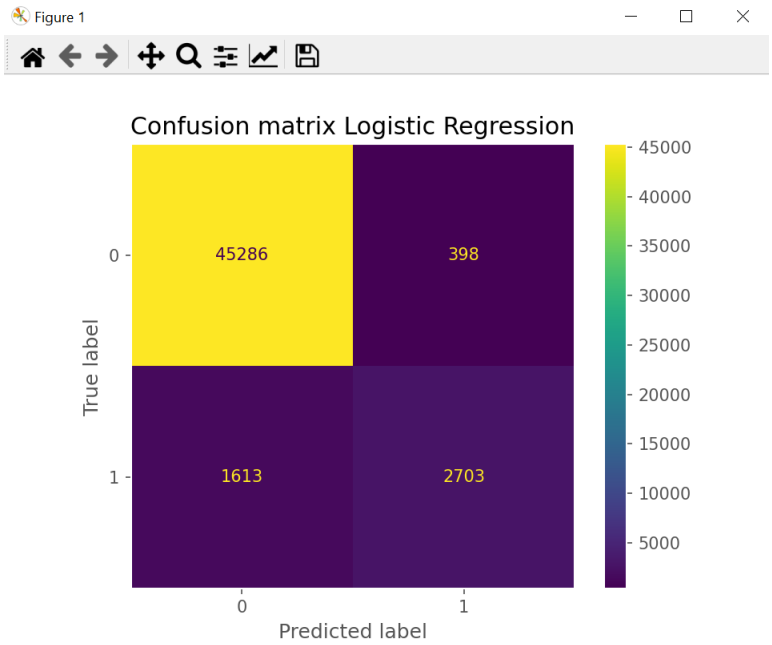
Ogni esempio è composto da m attributi X e da un'etichetta y che indica la corretta classificazione.

Nel nostro caso il dataset viene prima pre-processato creando l'oggetto di classe diabetes_data, tramite questa classe si elimina le feature relativa al sesso e ai precendeti con il fumare, successivamente definisce tramite il metodo "get training data", che restituisce i dati relativi a input features e target features. Nel nostro caso la target feature è una sola, cioè "diabetes". Questa feature indica se un soggetto è malato o meno di diabete.

Qui di seguito vengono mostrati a video i risultati delle metriche calcolate:

```
Logistic Regression metrics
Accurancy : 0.960
Precision : 0.872
Recall : 0.626
F1_precision : 0.729
```

La Matrice di confusione della regressione logistica è la seguente:



in questo caso abbiamo 2703 casi true-positive, ovvero che appartengono alle persone la cui predizione sul diabete ha dato esito positivo. Abbiamo 45286 true negative, quindi il modello ha predetto correttamente che 45286 soggetti non presentano il diabete, mentre i false-positive(398) ed i false-negative(1613) non sono stati predetti correttamente dal modello.

Analisi delle metriche al variare del parametro preso in input.

Nel caso della regressione logistica, il parametro che viene cambiato è il numero di iterazioni fatte dall' algoritmo di classificazione.

3.3 Albero di decisione

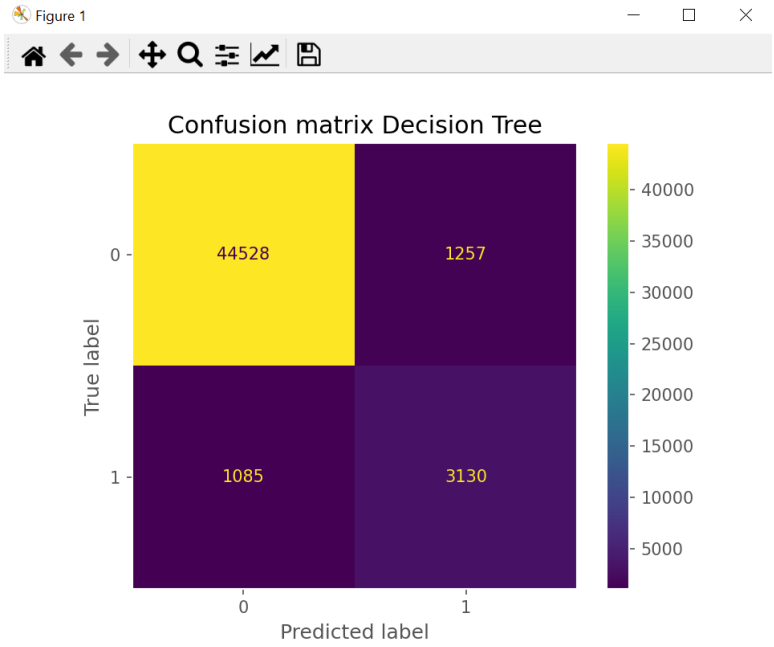
Gli alberi di decisione sono una tecnica di apprendimento automatico (machine learning) che viene utilizzata per la classificazione o la regressione di dati. Si basano su una struttura ad albero in cui ogni nodo rappresenta una caratteristica (o una variabile) del dataset e ogni arco rappresenta una regola di decisione basata su quella caratteristica.

Dal nodo dipendono tanti archi quanti sono i possibili valori che la caratteristica può assumere, fino a raggiungere le foglie che indicano la categoria associata alla decisione.

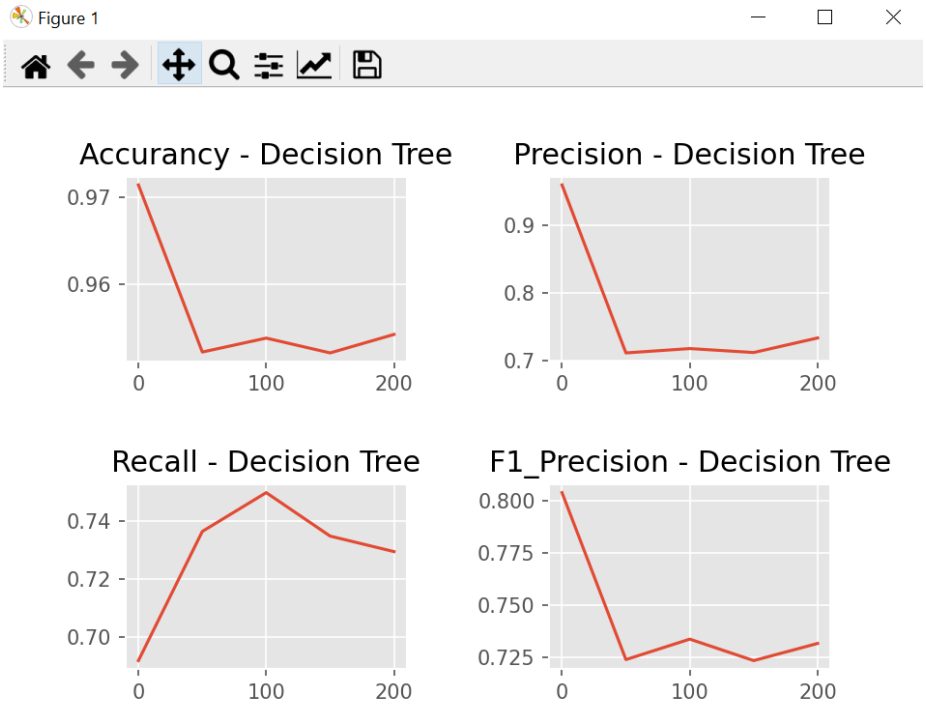
Qui di seguito vengono mostrati a video i risultati delle metriche calcolate:

```
Decision tree metrics
Accurancy : 0.953
Precision : 0.713
Recall : 0.743
F1_precision : 0.728
```

Utilizziamo nuovamente una Matrice di Confusione per analizzare gli errori compiuti dal modello.



in questo caso abbiamo 3130 casi true-positive, ovvero che appartengono alle persone la cui predizione sul diabete ha dato esito positivo. Abbiamo 44528 true negative, mentre il false-positive(1257) ed il false-negative(1085) non sono stati predetti correttamente dal modello.



Nell’albero di decisione varia la profondità.

Le performance scendono verso la profondità 50 per accuratezza precisione e F1_precision.Mentre per il recall sono basse all'inizio ma alte a profondità maggiore.

3.4 K-nearest neighbor

Il **K-Nearest Neighbors (KNN)** è un algoritmo di machine learning utilizzato per problemi di **classificazione** e **regressione**. Il KNN classifica una nuova osservazione in base alle classi delle k osservazioni più vicine (i "vicini") nel dataset di addestramento.

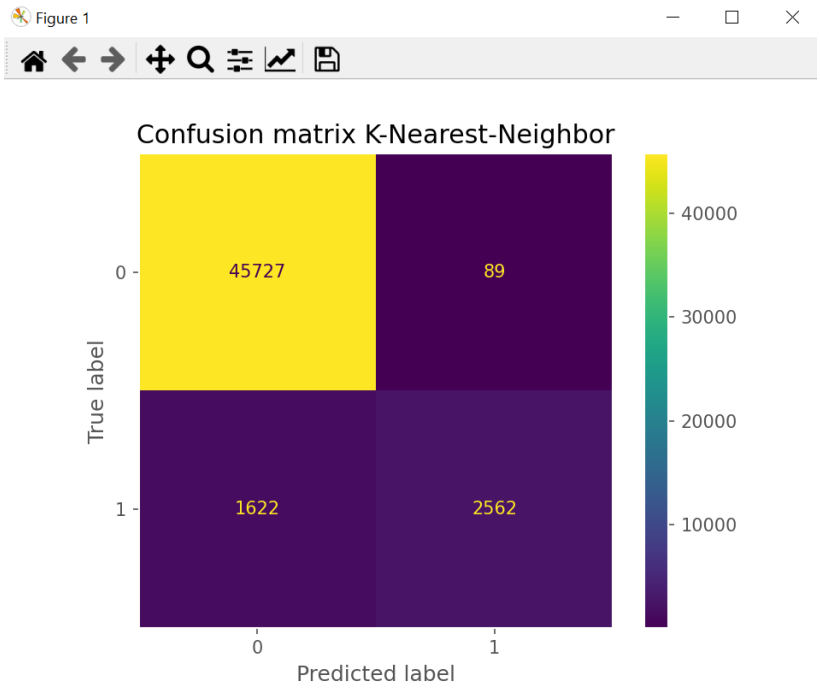
Passaggi fondamentali:

1. **Scegliere il numero di vicini (k)**: L'utente specifica il numero di vicini più prossimi k che verranno considerati per classificare un nuovo punto.
2. **Calcolare la distanza**: Per ogni nuova osservazione, KNN calcola la distanza tra questa osservazione e tutte le osservazioni del dataset di addestramento. La distanza più comunemente usata è la distanza euclidea, ma si possono usare altre metriche come la similarità del coseno.
3. **Identificare i K vicini più vicini**: Una volta calcolate le distanze, KNN seleziona i k punti dati più vicini alla nuova osservazione.
4. **Assegnare una classe** (per classificazione): L'osservazione viene assegnata alla classe più frequente tra i suoi vicini.

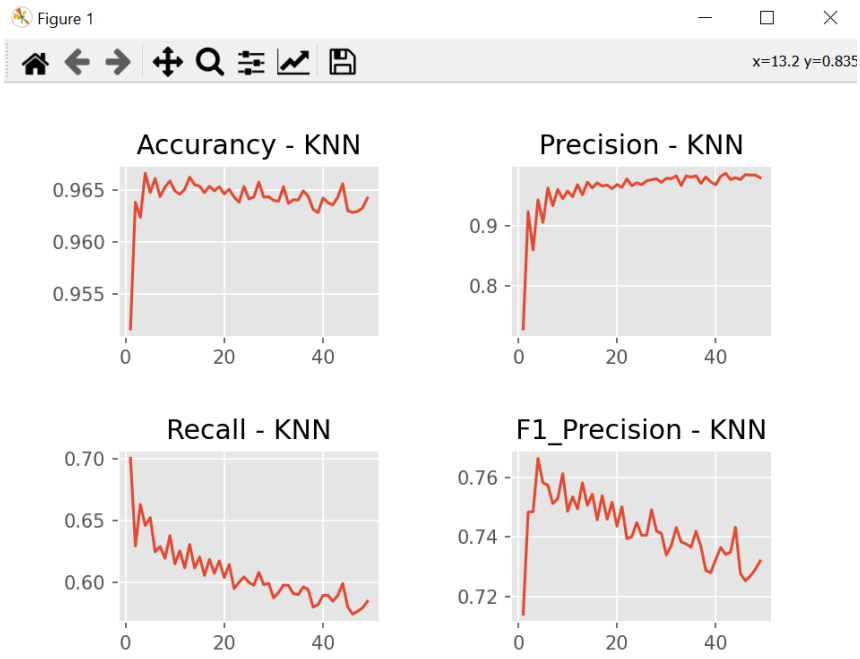
Qui di seguito vengono mostrati a video i risultati delle metriche calcolate:

```
Knn tree metrics
Accurancy : 0.966
Precision : 0.966
Recall : 0.612
F1_precision : 0.750
```

La Matrice di confusione del K-nearest neighbor è la seguente:

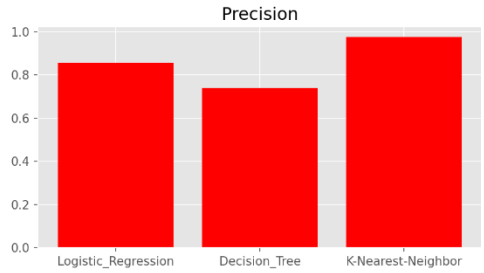


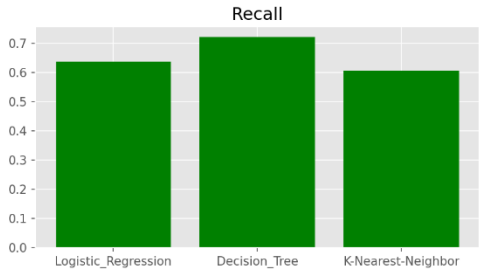
in questo caso abbiamo 2562 casi true-positive, ovvero che appartengono alle persone la cui predizione sul diabete ha dato esito positivo. Abbiamo 45727 true negative, mentre il false-positive(89) ed il false-negative(1622) non sono stati predetti correttamente.

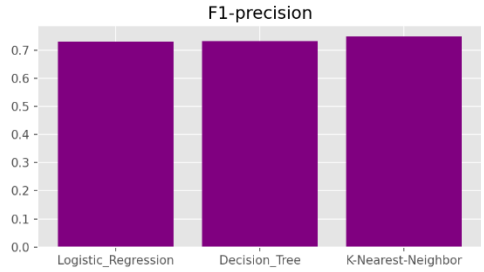


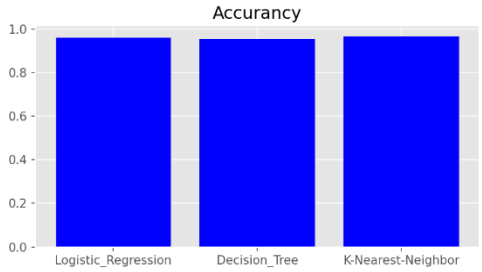
In questo caso variano i neighbors presi in considerazione.

3.5 Confronto modelli di apprendimento supervisionato









In linea generale le prestazioni dei modelli sono simili con qualche differenza soprattuto tra le misure di precision e recall.

4 Apprendimento supervisionato k-fold cross-validation

La **k-fold cross-validation** è una tecnica utilizzata nel campo **machine learning** per valutare l'accuratezza di un modello predittivo e prevenire il problema dell'overfitting. Questa tecnica suddivide il dataset in **k** sottoinsiemi o "folds". Ogni fold viene usato una volta come set di test e le restanti **k-1** folds vengono usate come set di addestramento. L'idea è di ottenere una stima più accurata delle performance del modello sfruttando più divisioni del dataset.

Passaggi della k-fold cross-validation:

1. Il dataset viene suddiviso in **k** parti (**k = 5** nel nostro caso).
2. Per ciascuna delle **k** iterazioni:
    - Un fold viene selezionato come set di test.

- I restanti **k-1** folds vengono usati come set di addestramento.
- Il modello viene addestrato sul set di addestramento e valutato sul set di test.

3. Alla fine, si calcola la media e deviazione standard delle performance come: precision, recall, acuracy, f1-score.

Ho utilizzato gli stessi classificatori usati in precedenza ma utilizzando la tecnica del k-fold cross-validation su ognuno di essi.

I fold che verranno creati con questa tecnica inizieranno sempre da un punto diverso del dataset, in modo da non andare ad usare sempre le stesse suddivisioni del dataset, grazie ad una funzionalità della libreria sklearn che permette di iniziare da un punto diverso dall'inizio.

Ogni modello avrà la propria suddivisione in fold diversa da quella degli altri per esecuzione.

Per utilizzare i classificatori con il k-fold cross-validation bisogna eseguire il file main_kfold.py

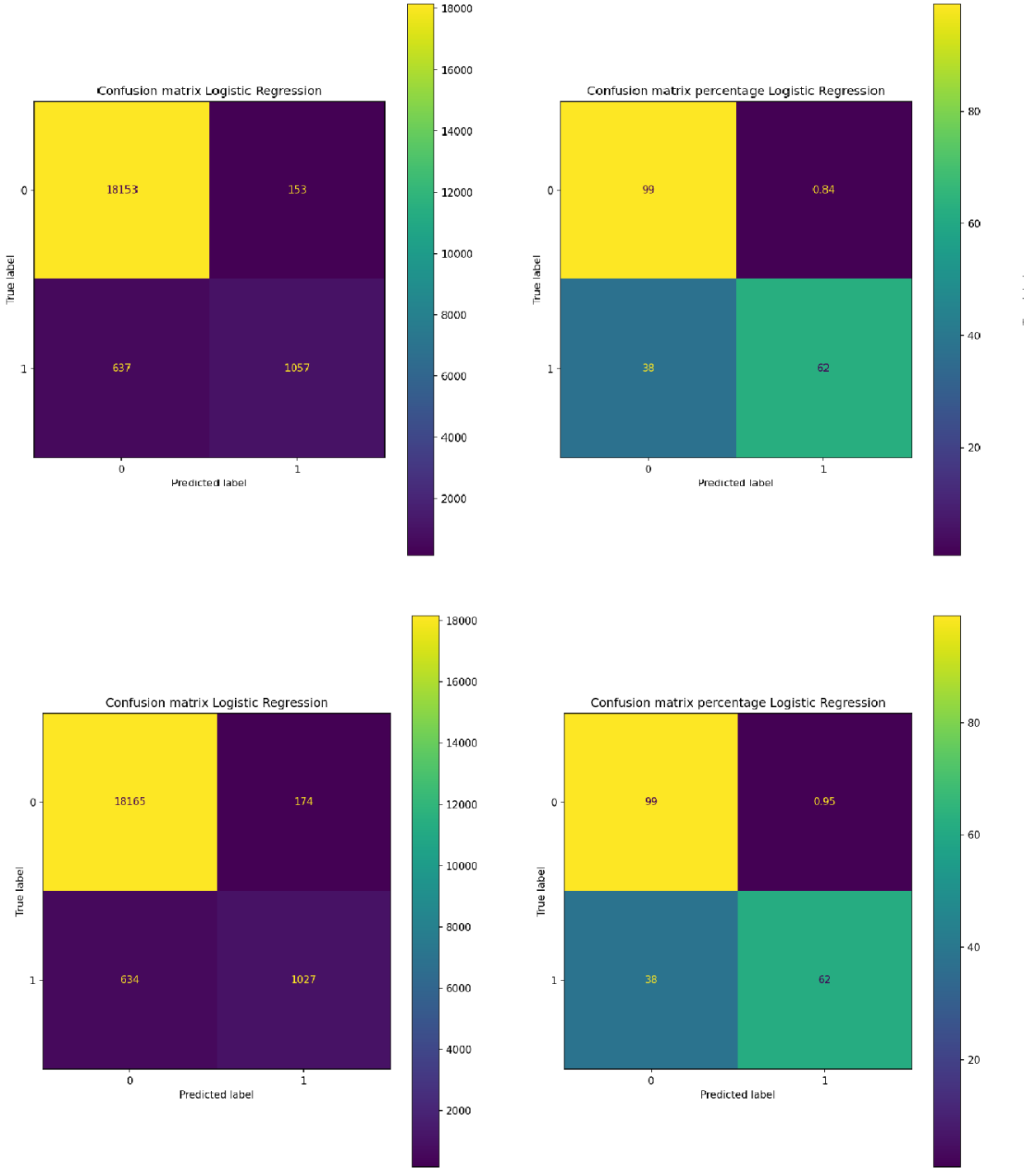
4.1 Funzionalità implementate

Le funzionalità implementate in questa parte del progetto sono:

1) Calcolo della misure di accuracy, precision, recall, f1-score per ogni singolo fold preso come test set, in base al classificatore usato :

```
Logistic Regression     Decision Tree           Knn
K fold 1                K fold 1                K fold 1
Accuracy: 0.958650      Accuracy: 0.954750      Accuracy: 0.963000
Precision : 0.913435    Precision : 0.853796    Precision : 0.972032
Recall: 0.802492        Recall: 0.858484        Recall: 0.788150
F1-Score: 0.847479      F1-Score: 0.856120      F1-Score: 0.853647

K fold 2                K fold 2                K fold 2
Accuracy: 0.959600      Accuracy: 0.951950      Accuracy: 0.964900
Precision : 0.910698    Precision : 0.839041    Precision : 0.973883
Recall: 0.804407        Recall: 0.851429        Recall: 0.793062
F1-Score: 0.847962      F1-Score: 0.845093      F1-Score: 0.858249

K fold 3                K fold 3                K fold 3
Accuracy: 0.960000      Accuracy: 0.954900      Accuracy: 0.962950
Precision : 0.923703    Precision : 0.860873    Precision : 0.970943
Recall: 0.808027        Recall: 0.852229        Recall: 0.792766
F1-Score: 0.854698      F1-Score: 0.856484      F1-Score: 0.857010

K fold 4                K fold 4                K fold 4
Accuracy: 0.961800      Accuracy: 0.954300      Accuracy: 0.965750
Precision : 0.920491    Precision : 0.852055    Precision : 0.971637
Recall: 0.816923        Recall: 0.854315        Recall: 0.804092
F1-Score: 0.859848      F1-Score: 0.853180      F1-Score: 0.866305

K fold 5                K fold 5                K fold 5
Accuracy: 0.960500      Accuracy: 0.953250      Accuracy: 0.964800
Precision : 0.919826    Precision : 0.847129    Precision : 0.971822
Recall: 0.807805        Recall: 0.854735        Recall: 0.797833
F1-Score: 0.853333      F1-Score: 0.850880      F1-Score: 0.861429
```

2) Mostra le matrici di confusione(normale, in percentuale) per ogni singolo fold, in base al classificatore usato:

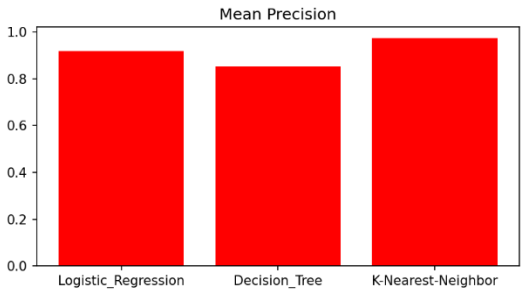


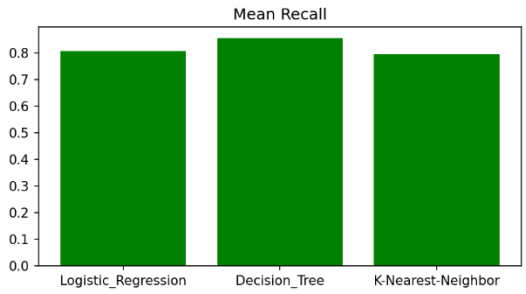
Mostrerà in totale 15 schermate di matrici di confusione delle quali 5 per ogni fold del k-fold in base al modello usato. Le matrici di confusione saranno in totale 30, 15 matrici di confusione e anche la relativa matrice di confusione in percentuale.

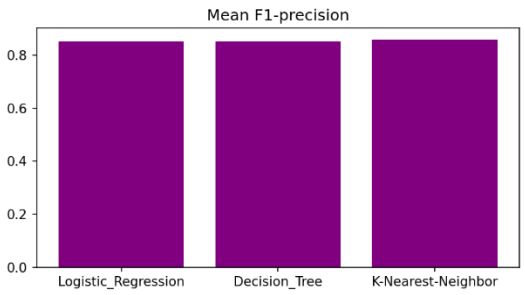
3) Calcolo media delle misure dei fold e della deviazione standard per ogni misura.

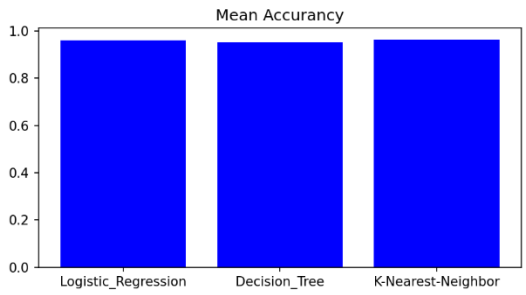
```
Metriche Logistic Regression
Accuracy media: 0.960110, Deviazione standard: 0.001015
Precisione media: 0.960110, Deviazione standard: 0.004708
Recall media: 0.807876, Deviazione standard: 0.004853
F1-Score medio: 0.852645, Deviazione standard: 0.004503

Metriche Decision Tree
Accuracy media: 0.954000, Deviazione standard: 0.001383
Precisione media: 0.954000, Deviazione standard: 0.008136
Recall media: 0.854908, Deviazione standard: 0.002555
F1-Score medio: 0.852937, Deviazione standard: 0.004773

Metriche Knn
Accuracy media: 0.964280, Deviazione standard: 0.001116
Precisione media: 0.964280, Deviazione standard: 0.000981
Recall media: 0.795180, Deviazione standard: 0.005407
F1-Score medio: 0.859328, Deviazione standard: 0.004287
```

4) Confronto medie delle misure dei fold in base ai modelli utilizzati:

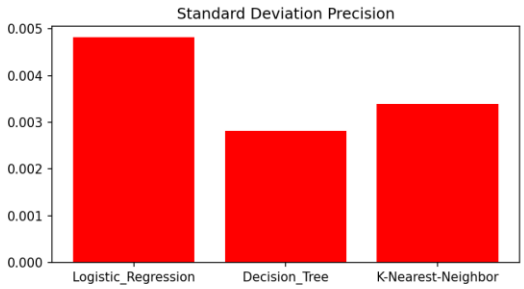


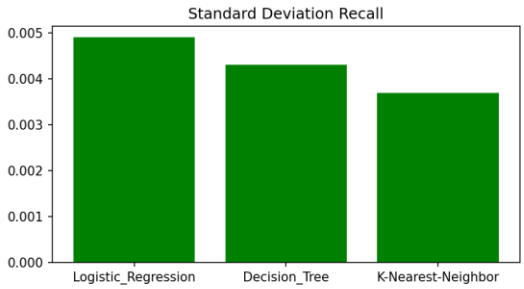


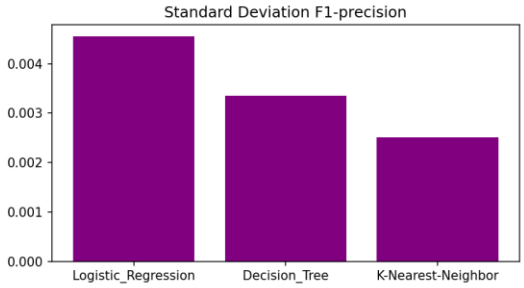


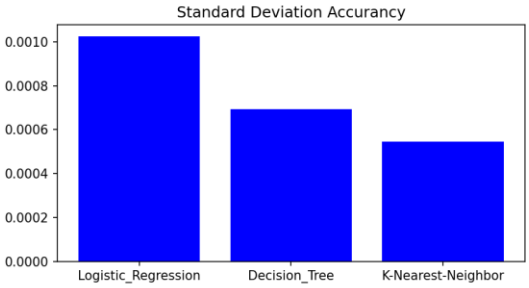


5) Confronto Deviazione standard in base ai modelli utilizzati:









6) Iperparametri scelti per modello:

```
Migliori iperparametri trovati dalla Grid Search per Logistic Regression:
 {'C': 0.1, 'penalty': 'l1', 'solver': 'liblinear'}
Migliori iperparametri trovati dalla Randomized Search per Decision Tree:
 {'criterion': 'entropy', 'max_depth': 73, 'max_leaf_nodes': 53, 'min_samples_leaf': 8, 'min_samples_split': 16}
Migliori iperparametri trovati con la Random Search per Knn:
 {'weights': 'uniform', 'p': 2, 'n_neighbors': 9, 'algorithm': 'auto'}
```

4.2 Configurazione degli Iperparametri

Gli **iperparametri** nei modelli di apprendimento(**machine learning**) sono parametri che devono essere impostati **prima** dell'addestramento del modello e **non** vengono appresi dai dati. Questi parametri influenzano direttamente il funzionamento del modello e la sua capacità di generalizzare sui dati nuovi.

Gli iperparametri dei modelli usati vengono ottimizzati tramite tecniche di ricerca come la **Grid Search** e la **Randomized Search:**

- Il **Grid search** è una tecnica utilizzata in machine learning per ottimizzare i parametri di un modello. Consiste nella ricerca sistematica di una combinazione di parametri che massimizzano (o minimizzano) una funzione obiettivo, come l'accuratezza del modello su un set di dati.

❖ Il **Randomized Search** è una tecnica di ottimizzazione dei parametri iperparametrici in machine learning. A differenza del Grid Search, che esplora tutte le combinazioni possibili di parametri in una griglia predefinita, il Randomized Search seleziona casualmente un sottoinsieme di combinazioni di parametri da testare.

a) Regressione Logistica:

Per la Regressione logistica ho utilizato la **Grid Search** per quanto riguarda la scelta di:

- **C**: Il parametro ***C*** controlla la quantità di regolarizzazione applicata al modello. Nella regressione logistica regolarizzata, l'algoritmo cerca di bilanciare il compromesso tra l'accuratezza del modello sui dati di addestramento e la sua capacità di generalizzare su nuovi dati.

- **penalty**: È possibile scegliere il tipo di regolarizzazione da applicare. I due principali tipi sono:
  L1: Regolarizzazione di tipo Lasso, che tende a portare i pesi di alcune feature a zero, favorendo la selezione delle feature più rilevanti.
  L2: Regolarizzazione di tipo Ridge, che riduce i valori dei coefficienti ma non li annulla. È utile per prevenire l'overfitting mantenendo tutte le feature.

- **solver**: Il solver è l'algoritmo utilizzato per ottimizzare la funzione di costo. I solver più comuni sono:
  liblinear: Ottimizza il problema per piccole dimensioni di dataset e supporta la regolarizzazione L1 e L2.
  saga: Un solver robusto per grandi dataset, compatibile sia con L1 che L2.

b) Decision Tree:

Per il Decision Tree ho utilizato il **Randomized Search** per quanto riguarda la scelta di:

- **max_depth** (Profondità massima dell'albero): Limita la profondità dell'albero, ossia il numero massimo di livelli di nodi che l'albero può avere. Un albero con profondità maggiore può catturare meglio la complessità dei dati, ma può anche sovradattarsi (overfitting).

- **min_samples_split** (Minimo numero di campioni per effettuare uno split): Numero minimo di campioni richiesti per dividere un nodo. Controlla quando un nodo verrà diviso. Se il numero di campioni in un nodo è inferiore a questo valore, lo split non verrà effettuato.

- **min_samples_leaf** (Minimo numero di campioni in una foglia): Numero minimo di campioni che deve essere presente in una foglia (nodo finale). Se

un nodo ha meno campioni rispetto a questo valore, non può essere ulteriormente diviso.

- **max_leaf_nodes** (Numero massimo di nodi foglia): Numero massimo di foglie (nodi finali) che l'albero può avere. Limitare il numero di foglie aiuta a controllare la complessità del modello.

- **criterion** (Criterio di impurezza): Questo iperparametro specifica la funzione utilizzata per misurare la qualità degli split. I valori comuni sono:
  "gini": Utilizza l'impurità di Gini per decidere gli split.
  "entropy": Utilizza l'informazione entropica (Guadagno di informazione).

c) Knn:

Per il Knn ho utilizato il **Randomized Search** per quanto riguarda la scelta di:

- **n_neighbors**: Numero di vicini più prossimi da considerare per la classificazione

- **weights**: Metodo di pesatura dei vicini che può essere:
  '***uniform***': Tutti i vicini hanno lo stesso peso.
  '***distance***': I vicini più vicini hanno un peso maggiore (inversamente proporzionale alla distanza).

- **algorithm:** Algoritmo da utilizzare per calcolare i vicini più prossimi che può essere:
  'auto': Sceglie automaticamente l'algoritmo in base ai dati.
  'kd_tree': Utilizza un albero KD per la ricerca.

- **p:** Potenza dell'ordine della distanza di Minkowski.
  p=1: Distanza di Manhattan (L1).
  p=2: Distanza Euclidea (L2)

5 Considerazioni finali

In conclusione, il sistema realizzato soddisfa gli obiettivi da me stabiliti inizialmente. Per il sistema esperto con la relativa gestione dell'ontologia, ho riscontrato una funzionalità soddisfacente sia nell'implementazione del CSP, con cui gestistco le prenotazioni delle visite del sangue, sia con la diagnostica dei sintomi dei pazienti relativa all'ontologia.

Inoltre, ho utilizato diversi modelli di apprendimento supervisionato rispetto al dataset, per poterli confrontare su uno stesso dataset per un determinato caso di studio. Ho utilizzato la k-fold cross-validation per prevenire il problema dell' overfitting, ed avere un utilizzo del dataset più efficente. Ho scelto gli Iperparametri dei modelli attraverso delle tecniche, per ottimizare i modelli. In fine ho potuto confrontare le varie medie delle misure e deviazioni standard per i modelli utilizzati riscontrando dei valori simili nelle medie delle misure e diversi per le deviazioni standard dei modelli.