Sistema di diagnostica sul diabete

Chiedi Pietro, Matricola: **718854**
Email: p.chiedi@studenti.uniba.it
`https://github.com/chdptr/ICON`

Indice

1 Introduzione

Il diabete è una condizione caratterizzata da livelli elevati di glucosio nel sangue, noti come iperglicemia. L'eccesso di glucosio può essere dovuto a una produzione insufficiente di insulina dall'ormone responsabile della regolazione del glucosio nel sangue o a un'azione inadeguata di queste ultime. Il diabete può essere suddiviso in due forme principali: l'insulino-dipendente, in cui si verifica un'assenza di produzione di insulina e non insulino-dipendente in cui la produzione da parte del fegato dei muscoli e dei tessuti adiposi o la sensibilità a questi ultimi viene ridotta, riducendo così la produzione di insulina dal pancreas. I miei modelli di diagnosi del diabete sono basati su due componenti principali, l'uso di algoritmi di apprendimento supervisionato e di un agente intelligente che tiene conto delle risposte dell'utente per la diagnosi.

2 Knowledge base

Ho quindi deciso di sviluppare un sistema esperto in grado di diagnosticare il diabete.

Un sistema esperto è un programma in grado di associare un insieme di **fatti** a un insieme di **regole**, ed eseguire alcune azioni in base alle regole di corrispondenza.

Dopo aver condotto alcune ricerche, ho scoperto e appreso l'utilizzo di una libreria Python chiamata Experta, che ha contribuito alla creazione del mio sistema esperto. Di seguito vengono mostrati dettagli e spiegazioni del sistema.

2.1 Fatti

1. La classe Fact è una sottoclasse di dict.

```
>>> f = Fact(a=1, b=2)
>>> f['a']
1
```

2. Un Fatto, quindi, non mantiene un ordine interno di elementi.

```
>>> Fact(a=1, b=2)
Fact(b=2, a=1)
```

3. A differenza di dict, è possibile creare un Fact senza chiavi (solo valori) e Fact creerà un indice numerico per i tuoi valori.

```
>>> f = Fact('x', 'y', 'z')
>>> f[0]
'x'
```

4. Puoi combinare i valori autonumerici con i valori-chiave, ma l'autonumeric deve essere dichiarato prima:

```
>>> f = Fact('x', 'y', 'z', a=1, b=2)
>>> f[1]
'y'
>>> f['b']
2
```

5. È possibile sottoclassare Fact per esprimere diversi tipi di dati o estenderli con la funzionalità personalizzate.

```
class Alert(Fact):
    """The alert level."""
    pass

class Status(Fact):
    """The system status."""
    pass

f1 = Alert('red')
f2 = Status('critical')
```

```
class Alert(Fact):
    """The alert level."""
    pass

class Status(Fact):
    """The system status."""
    pass

f1 = Alert('red')
f2 = Status('critical')
```

6. I campi dei fatti possono essere convalidati automaticamente, se li definisci utilizzando Field. Il campo utilizza la libreria Schema internamente per la convalida dei dati. Inoltre, un campo può essere dichiarato obbligatorio o avere un valore predefinito.

2.2 Regole

La regola nella libreria Experta è una regola logica o un insieme di regole utilizzate all'interno di un sistema esperto.
Le regole hanno due componenti, LHS (lato sinistro) e RHS (lato destro).

- Il LHS descrive (usando **patterns**) le condizioni in base alle quali la regola * dovrebbe essere eseguita (o attivata).
- L' RHS è l'insieme delle azioni da eseguire quando la regola viene attivata.

Affinché un Fact corrisponda a un Pattern , tutte le restrizioni del pattern devono essere **True** quando il Fact viene valutato rispetto ad esso.

```python
class MyFact(Fact):
    pass

@Rule(MyFact())  # This is the LHS
def match_with_every_myfact():
    """This rule will match with every instance of `MyFact`."""
    # This is the RHS
    pass

@Rule(Fact('animal', family='felinae'))
def match_with_cats():
    """
    Match with every `Fact` which:

      * f[0] == 'animal'
      * f['family'] == 'felinae'

    """
    print("Meow!")
```

È possibile utilizzare operatori logici per esprimere condizioni LHS complesse.

```python
@Rule(
    AND(
        OR(User('admin'),
           User('root')),
        NOT(Fact('drop-privileges'))
    )
)
def the_user_has_power():
    """
    The user is a privileged one and we are not dropping privileges.

    """
    enable_superpowers()
```

Affinchè una **regola** sia utile, è necessario che sia un metodo di una sottoclasse **KnowledgeEngine**.

2.3 Fatti vs Modelli

La distinzione tra fatti e modelli è sottile. In realtà, i modelli sono essenzialmente fatti che includono elementi condizionali del modello anziché dati regolari. Vengono utilizzati esclusivamente nella parte sinistra (LHS) di una regola. Se non si fornisce il contenuto di un modello come PCE, Experta automaticamente includerà il valore in un LiteralPCE. Inoltre, non è possibile dichiarare alcun fatto che contenga un PCE; se lo si tenta, si otterrà un'eccezione.

```
>>> ke = KnowledgeEngine()
>>> ke.declare(Fact(L("hi")))
Traceback (most recent call last):
  File "<ipython-input-4-b36cff89278d>", line 1, in <module>
    ke.declare(Fact(L('hi')))
  File "/home/experta/experta/engine.py", line 210, in declare
    self.__declare(*facts)
  File "/home/experta/experta/engine.py", line 191, in __declare
    "Declared facts cannot contain conditional elements")
TypeError: Declared facts cannot contain conditional elements
```

2.4 Difetti

Spesso, i sistemi esperti richiedono la presenza di una serie di fatti affinché il sistema funzioni correttamente. A questo scopo, si utilizza il decoratore DefFacts.

```python
@DefFacts()
def needed_data():
    yield Fact(best_color="red")
    yield Fact(best_body="medium")
    yield Fact(best_sweetness="dry")
```

Tutti i DefFacts all'interno di un KnowledgeEngine vengono richiamati ogni volta che viene eseguito il metodo reset.

2.5 Motore della conoscenza

Il primo passo consiste nel creare una sottoclasse e utilizzare Rule per decorare i suoi metodi. Successivamente, puoi creare un'istanza, popolarla con i fatti e, infine, eseguirla.

```
from experta import *

class Greetings(KnowledgeEngine):
    @DefFacts()
    def _initial_action(self):
        yield Fact(action="greet")

    @Rule(Fact(action='greet'),
          NOT(Fact(name=W())))
    def ask_name(self):
        self.declare(Fact(name=input("What's your name? ")))

    @Rule(Fact(action='greet'),
          NOT(Fact(location=W())))
    def ask_location(self):
        self.declare(Fact(location=input("Where are you? ")))

    @Rule(Fact(action='greet'),
          Fact(name=MATCH.name),
          Fact(location=MATCH.location))
    def greet(self, name, location):
        print("Hi %s! How is the weather in %s?" % (name, location))

engine = Greetings()
engine.reset()  # Prepare the engine for the execution.
engine.run()  # Run it!
```

```
$ python greet.py
What's your name? Roberto
Where are you? Madrid
Hi Roberto! How is the weather in Madrid?
```

Figura 2.5.1: Output dell'esempio di sopra

2.6 Gestione dei fatti

I seguenti metodi vengono utilizzati per manipolare l'insieme di fatti di cui il motore è a conoscenza.

2.6.1 Dichiarare

Aggiunge un nuovo fatto all'elenco dei fatti (l'elenco dei fatti noti al motore).

```
>>> engine = KnowledgeEngine()
>>> engine.reset()
>>> engine.declare(Fact(score=5))
<f-1>
>>> engine.facts
<f-0> InitialFact()
<f-1> Fact(score=5)
```

Nota: Lo stesso fatto non può essere dichiarato due volte a meno che $fact.duplication$ non sia impostato su True.

2.6.2 Ritrattare

Rimuove un fatto esistente dall'elenco dei fatti.

```
>>> engine.facts
<f-0> InitialFact()
<f-1> Fact(score=5)
<f-2> Fact(color='red')
>>> engine.retract(1)
>>> engine.facts
<f-0> InitialFact()
<f-2> Fact(color='red')
```

2.6.3 Modificare

Ritira alcuni fatti dall'elenco dei fatti e ne dichiara uno nuovo con alcune modifiche. Le modifiche vengono passate come argomenti.

```
>>> engine.facts
<f-0> InitialFact()
<f-1> Fact(color='red')
>>> engine.modify(engine.facts[1], color='yellow', blink=True)
<f-2>
>>> engine.facts
<f-0> InitialFact()
<f-2> Fact(color='yellow', blink=True)
```

2.6.4 Duplicare

Aggiunge un nuovo fatto all'elenco utilizzandone uno esistente come modello e apportando alcune modifiche.

```
>>> engine.facts
<f-0> InitialFact()
<f-1> Fact(color='red')
>>> engine.duplicate(engine.facts[1], color='yellow', blink=True)
<f-2>
>>> engine.facts
<f-0> InitialFact()
<f-1> Fact(color='red')
<f-2> Fact(color='yellow', blink=True)
```

2.7 Procedura di esecuzione del motore

Ecco il procedimento standard per eseguire un **KnowledgeEngine**:

1. In primo luogo, è necessario istanziare la classe.
2. Successivamente, occorre chiamare il metodo di ripristino:
   - Questo processo dichiara il fatto speciale InitialFact, essenziale per il corretto funzionamento di alcune regole.
   - Inoltre, dichiara tutti i fatti generati dai metodi decorati con @DefFacts.
3. Infine, è essenziale chiamare il metodo run, che avvia il ciclo di esecuzione.

2.8 Ciclo di esecuzione

Nella programmazione convenzionale, il programmatore definisce esplicitamente il punto di inizio, il punto di fine e la sequenza delle operazioni. Con Experta, il flusso del programma non richiede una definizione così esplicita. La conoscenza (Regole) e i dati (Fatti) sono separati, e il KnowledgeEngine viene utilizzato per applicare la conoscenza ai dati.
Il ciclo di esecuzione di base procede come segue:

1. Se il limite di attivazione delle regole è stato raggiunto, l'esecuzione viene interrotta.
2. Viene selezionata la regola in cima per l'esecuzione. Se non ci sono regole, l'esecuzione viene interrotta.

3. Le azioni RHS della regola selezionata vengono eseguite (tramite l'invocazione del metodo). Di conseguenza, le regole possono essere attivate o disattivate. Le regole attivate (quelle il cui condizioni sono attualmente soddisfatte) vengono messe in agenda. La posizione è determinata dalla rilevanza della regola e dalla strategia corrente di risoluzione dei conflitti. Le regole disattivate vengono rimosse dall'agenda.

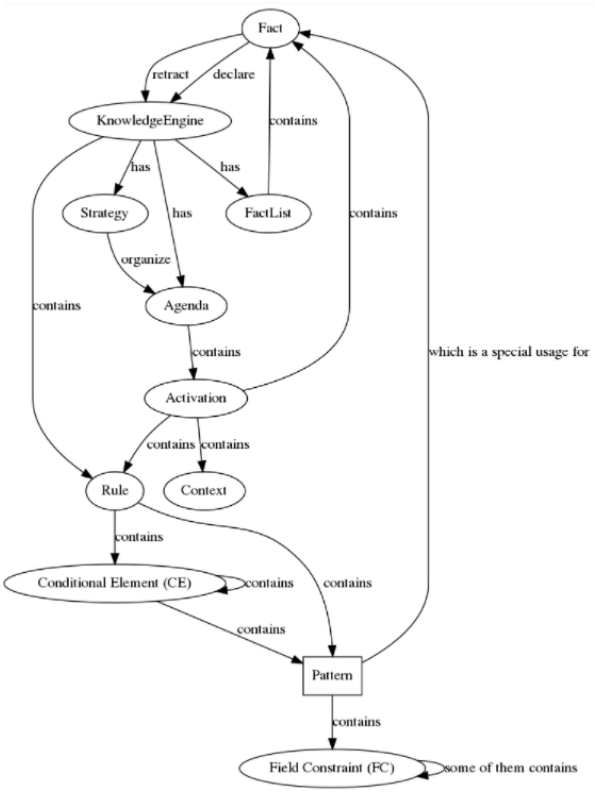
2.9 DefFacts vs Declare

Entrambi servono per dichiarare fatti sull'istanza del motore, tuttavia:

- "declare" aggiunge i fatti direttamente in memoria.
- I generatori dichiarati con "@DefFacts" vengono invocati dal metodo "reset" e tutti i fatti generati vengono aggiunti in memoria utilizzando "require".

2.10 Diagramma

Il diagramma qui sotto rappresenta tutti i componenti del sistema e le relazioni tra di essi.



2.11 Esempio di come funziona il sistema esperto di diagnostica sul diabete

Il sistema esperto funziona nel seguente modo: vengono poste domande all'utente e, in base alle risposte fornite, il sistema dichiara dei fatti, ovvero delle proposizioni che verranno quindi considerate nel processo di ragionamento. Questo consentirà al sistema di formulare una diagnosi relativa al diabete.

2.12 CSP

Numerosi problemi nell'ambito dell'Intelligenza Artificiale possono essere classificati come Problemi di Soddisfacimento di Vincoli (Constraint Satisfaction Problem o CSP). Formalmente, un CSP si basa su un insieme finito di variabili $(X_1, X_2, \ldots, X_n)$, i cui valori appartengono a domini finiti di definizione $(D_1, D_2, \ldots, D_n)$, e su un insieme di vincoli $(C_1, C_2, \ldots, C_n)$. Un vincolo su un insieme di variabili

rappresenta una restrizione sui valori che tali variabili possono assumere contemporaneamente. Concettualmente, un vincolo può essere visto come un insieme che contiene tutte le combinazioni di valori delle variabili che sono compatibili tra loro. Ad esempio, un vincolo tra k variabili $C(X_{i1}, X_{i2}, \ldots, X_{ik})$ è un sottoinsieme del prodotto cartesiano dei domini delle variabili coinvolte, che specifica quali valori delle variabili sono compatibili tra loro. Questo insieme può essere rappresentato in vari modi, come ad esempio tramite matrici, equazioni, disuguaglianze o relazioni. Il grado di una variabile è il numero di vincoli a cui è sottoposta.

Un problema di soddisfacimento di vincoli parte da un assegnamento iniziale, cioè un insieme di variabili già vincolate. Questo assegnamento iniziale può anche essere vuoto. La risoluzione del problema avanza estendendo l'assegnamento iniziale, ovvero assegnando via via valori alle variabili ancora libere. La soluzione di un CSP è un assegnamento completo e coerente di valori alle variabili, ottenuto estendendo l'assegnamento iniziale in modo che soddisfi tutti i vincoli e non lasci alcuna variabile libera.

Ho utilizzato la libreria denominata "constraint", che ha consentito di implementare un CSP semplice (nel mio caso, non prendo in considerazione altri utenti) per mostrare la disponibilità degli studi convenzionati nel caso in cui il sistema preveda la prescrizione di analisi.
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 il proprio dominio associato in modo esplicito.
- In base alle risposte dell'utente, il sistema decide se prescrivere o meno delle analisi.
- Se il sistema decide di prescrivere delle analisi, allora entra in gioco il CSP.
- Il CSP riguarda gli orari di apertura del laboratorio delle analisi.
- Esempio, il laboratorio per l'insulina effettua le analisi dalle ore 8 alle 14.
- Il sistema quindi indica gli orari possibili a cui l'utente può presentarsi per sottoporsi alle analisi.

2.12.1 Un possibile esempio:

```
Benvanuto in Diabetes Expert, un sistema esperto per la diagnosi e la cura del diabete di tipo 1
Ti senti molto assetato di solito (sopratutto di notte) ? [si/no]

no
Ti senti molto stanco? [si/no]

si
Stai perdendo peso e massa muscolare? [si/no]

no
Senti prurito? [si/no]

no
Hai la vista offuscata? [si/no]

si
Consumi spesso bevande zuccherate? [si/no]

no
Hai fame costantemente? [si/no]

no
Hai spesso la bocca asciutta? [si/no]

no


Inserisci l'altezza in centimetri


180
Inserisci il peso in kilogrammi


90
Hai fatto l'esame della pressione sanguigna?

si
Inserisci il valore della pressione sanguigna

190
Il valore della pressione e' maggiore o uguale a quella dei diabetici
Potresti avere il diabete, rivolgiti ad un medico
```

Figura 2.12.1: **Esempio:** Output del programma

```
L'agente ragiona con i seguenti fatti:

<f-0>: InitialFact()
<f-1>: Fact(inizio='si')
<f-2>: Fact(chiedi_sintomi='si')
<f-3>: Fact(molto_stanco='si')
<f-4>: Fact(vista_offuscata='si')
<f-5>: Fact(chiedi_imc='si')
<f-6>: Fact(esami_pressione='si')
<f-7>: Fact(esame_pressione_eseguito='si')
<f-8>: Fact(diagnosi_pressione_diabete='si')
<f-9>: Fact(diagnosi_diabete_incerta='si')
```

Figura 2.12.2: **In base alle risposte dell'utente** date nell'esempio, l'agente ragiona con i seguenti fatti

2.13 Ontologie

In informatica, un'ontologia rappresenta formalmente e in modo condiviso una concezione esplicita di un determinato dominio di interesse. In termini più precisi, si tratta di una teoria assiomatica del primo ordine che può essere espressa in una logica descrittiva.

Il concetto di ontologia formale è entrato in uso nel campo dell'intelligenza artificiale e della rappresentazione della conoscenza. Si riferisce alla maniera in cui vari schemi vengono combinati per creare una struttura dati che contiene tutte le entità rilevanti e le loro relazioni in un determinato dominio. I programmi informatici possono sfruttare l'ontologia per una serie di scopi, tra cui il ragionamento induttivo, la classificazione e diverse tecniche per la risoluzione di problemi.

L'ontologia è stata creata utilizzando il tool **Protege** (vedi Figura 2.13.1), mentre la sua lettura avviene tramite la libreria Python **Owlready2** (vedi Figura 2.13.2).

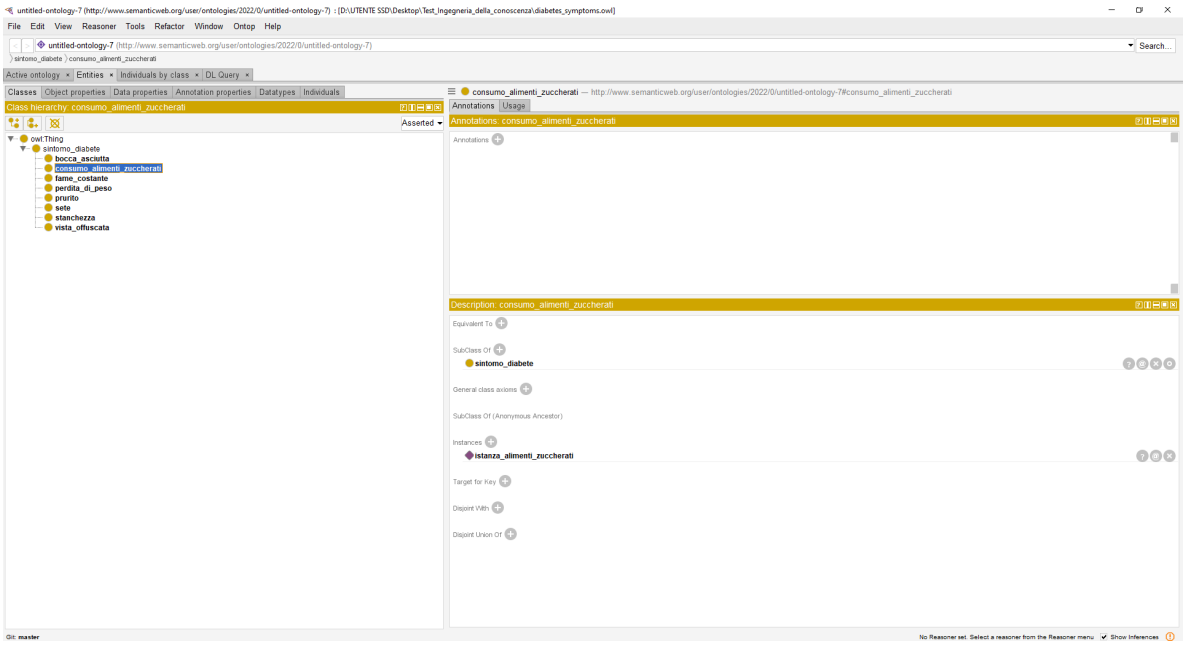


Figura 2.13.1: tool protege

```
* Owlready2 * Warning: optimized Cython parser module 'owlready2_optimized' is not available, defaulting to slower Python implementation
Benvanuto in Diabetes Expert, un sistema esperto per la diagnosi e la cura del diabete di tipo 1
---------->MENU<----------
[1] Mostra i possibili sintomi del diabete
[2] Esegui una diagnosi
[3] Esci
1
Sintomo [1]: Nome: alimenti_zuccherati
Sintomo [2]: Nome: bocca_asciutta
Sintomo [3]: Nome: fame_costante
Sintomo [4]: Nome: perdita_peso
Sintomo [5]: Nome: prurito
Sintomo [6]: Nome: sete
Sintomo [7]: Nome: stanchezza
Sintomo [8]: Nome: vista_offuscata

Seleziona il sintomo di cui vuoi conosere la descrizione, inserisci il numero del sintomo
3
Sintomo: fame_costante, descrizione: Le patologie che si possono associare ad una sensazione continua di fame comprendono il diabete, l'ipertiroidismo e l'insulinoma (tumore
 del pancreas). La fame dev'ssere distinta dal desiderio di mangiare e dal falso appetito, ossia dal bisogno psicologico di consumare alimenti la cui ingestione procura piac
ere.
```

Figura 2.13.2: Dall'ontologia viene creato un dizionario con i sintomi e la relativa descrizione

3 Dataset

Questo dataset proviene originariamente dall'Istituto Nazionale per il Diabete e le Malattie Digestive e Renali. L'obiettivo del dataset è predire diagnosticamente se un paziente ha il diabete, basandosi su determinate misurazioni diagnostiche incluse nel dataset. Sono state poste diverse restrizioni sulla selezione di queste istanze da un database più ampio. In particolare, tutte le pazienti qui presenti sono donne di almeno 21 anni di età di discendenza indiana Pima. Dal dataset presente nel file (.csv) possiamo trovare diverse variabili, alcune delle quali sono indipendenti (diverse variabili predittive mediche) e solo una variabile dipendente target (Outcome).

3.1 Analisi del dataset

3.1.1 Controllo del bilanciamento del dataset

È importante controllare che i dati in un dataset siano bilanciati perché avere un dataset bilanciato per un modello permetterebbe di ottenere modelli più accurati, una maggiore accuracy bilanciata e un tasso di rilevamento bilanciato più elevato. Di conseguenza, è fondamentale avere un dataset bilanciato per un modello di classificazione. Ciò significa che ogni classe viene considerata con la stessa importanza e può migliorare le prestazioni del modello. Se i dati sono sbilanciati, ciò può portare a risultati inaccurati e potrebbe essere necessario compiere ulteriori passaggi per bilanciare i dati prima dell'analisi.
Per prima cosa, per condurre un'analisi di pre-processing accurata, verifichiamo se il dataset è sbilanciato.

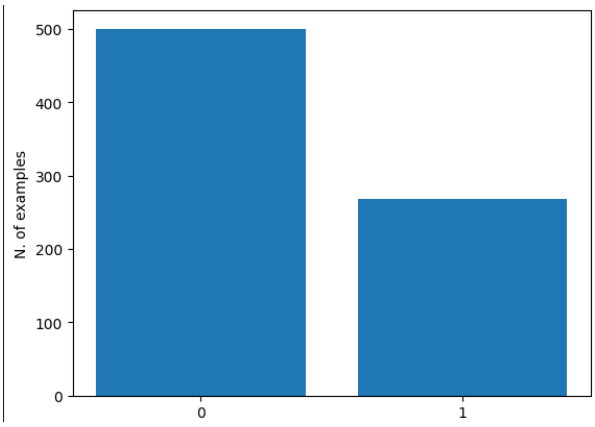


Figura 3.1.1: rappresentazione grafologica della distribuzione dei dati tra le due classi

Come possiamo notare, il dataset non è affatto bilanciato. Ci sono più istanze della classe 0 che della classe 1.

3.1.2 Controllo dei valori nulli

Naturalmente, verifichiamo la presenza di valori nulli nel dataset per individuare eventuali dati mancanti o incompleti. Questo può influenzare l'accuratezza dei risultati di un'analisi dei dati e potrebbe richiedere ulteriori fasi di pre-processing per gestire tali valori mancanti. In questo caso, non sono stati trovati valori nulli, quindi non è necessario eseguire alcuna operazione per risolvere questo problema.

3.1.3 Matrice di correlazione

Una matrice di correlazione è una tabella che evidenzia i coefficienti di correlazione tra le variabili. Ogni cella di questa tabella rappresenta la correlazione tra due variabili. Si tratta di uno strumento potente per riepilogare un insieme di dati e per individuare e visualizzare i modelli presenti nei dati. La matrice di correlazione è la seguente:

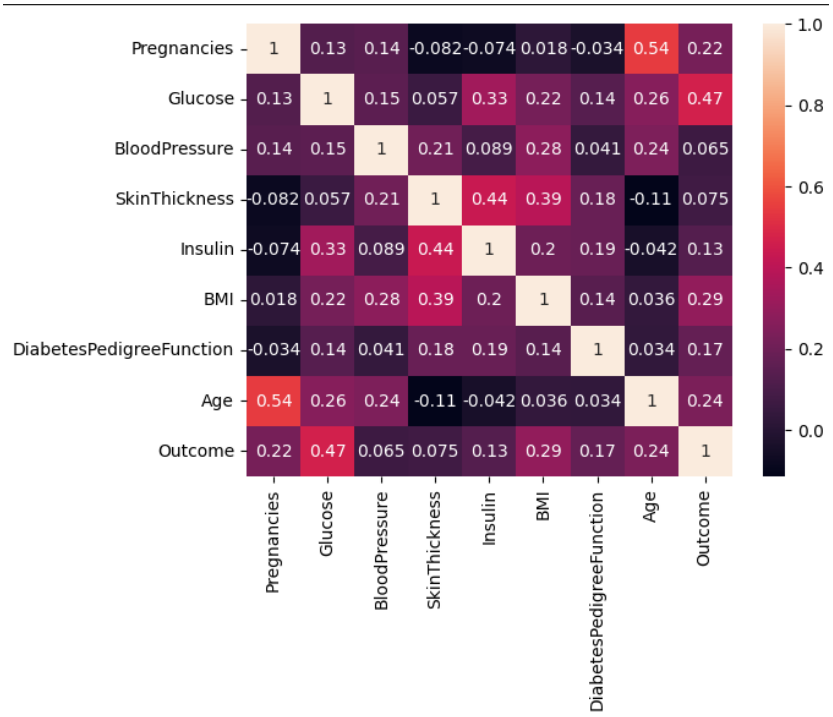


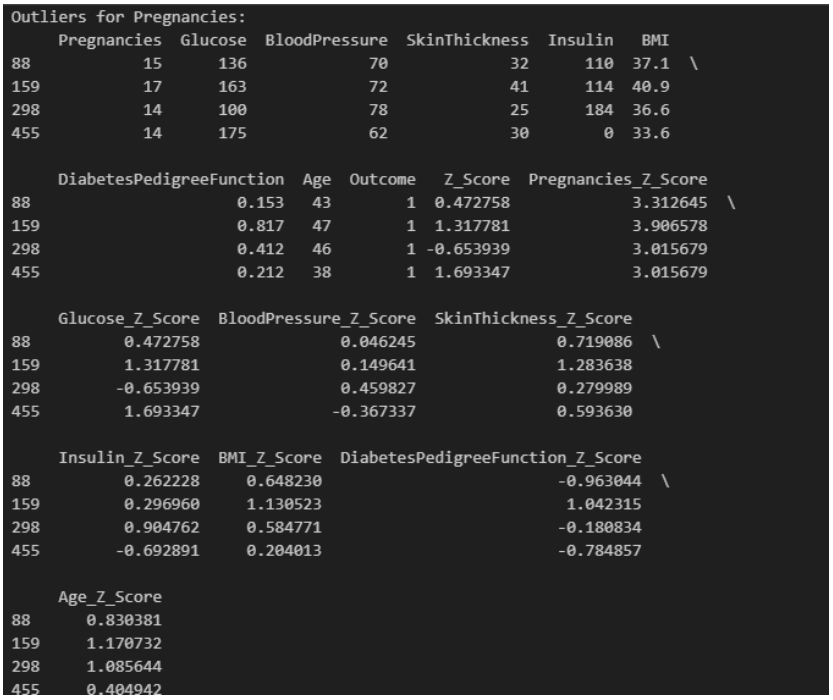
Figura 3.1.2: Matrice di correlazione

Dalla matrice emerge che alcune caratteristiche presentano una correlazione più marcata tra loro rispetto ad altre. Ad esempio, *Age* e *Pregnancies* mostrano una correlazione positiva relativamente forte di 0,544341, mentre *SkinThickness* e *Age* evidenziano una correlazione negativa relativamente debole di -0,113970. La variabile *Outcome*, presumibilmente indicativa della presenza o meno del diabete in un individuo, mostra la correlazione positiva più significativa con il glucosio, pari a 0,466581. Procediamo quindi con un'analisi più approfondita.

3.1.4 Z-Score

Il Z-Score rappresenta il numero di deviazioni standard di un data point rispetto alla media di un dataset. Si ottiene sottraendo la popolazione media da uno score individuale grezzo e dividendo la differenza per la deviazione standard della popolazione. Gli Z-Score possono essere sia positivi che negativi: un valore positivo indica che il data point è al di sopra della media, mentre un valore negativo indica che è al di sotto della media.

Gli Z-Score sono utili per confrontare i risultati con una popolazione "normale" e per individuare gli outlier in un dataset. Nel codice, gli Z-Score vengono calcolati per ogni colonna del dataset (ad eccezione della colonna degli esiti) e successivamente utilizzati per identificare gli outlier, ovvero i data point che si discostano di più di 3 deviazioni standard dalla media.



```
Outliers for Pregnancies:
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI
88            15      136             70             32      110  37.1  \
159           17      163             72             41      114  40.9
298           14      100             78             25      184  36.6
455           14      175             62             30        0  33.6

     DiabetesPedigreeFunction  Age  Outcome   Z_Score  Pregnancies_Z_Score
88                      0.153   43        1  0.472758             3.312645  \
159                     0.817   47        1  1.317781             3.906578
298                     0.412   46        1 -0.653939             3.015679
455                     0.212   38        1  1.693347             3.015679

     Glucose_Z_Score  BloodPressure_Z_Score  SkinThickness_Z_Score
88          0.472758               0.046245               0.719086  \
159         1.317781               0.149641               1.283638
298        -0.653939               0.459827               0.279989
455         1.693347              -0.367337               0.593630

     Insulin_Z_Score  BMI_Z_Score  DiabetesPedigreeFunction_Z_Score
88          0.262228     0.648230                         -0.963044  \
159         0.296960     1.130523                          1.042315
298         0.904762     0.584771                         -0.180834
455        -0.692891     0.204013                         -0.784857

     Age_Z_Score
88      0.830381
159     1.170732
298     1.085644
455     0.404942
```

Figura 3.1.3: Output Z-Score

Dai risultati emerge che nel dataset ci sono alcune righe in cui il valore della variabile "Glucose" è 0 e non solo. Questo suggerisce che tali valori non siano corretti, poiché un valore di 0 per la glicemia non è plausibile. Inoltre, notiamo che queste righe presentano uno Z-score molto basso (-3,783654) per la variabile "Glucose", indicando che tali valori si discostano significativamente dalla media in termini di deviazione standard. Di conseguenza, li identifichiamo come outlier.

3.1.5 T-Student Test

Ho optato per un test statistico utilizzando il T-Student test. Ho scelto questo test in quanto mi trovo di fronte a un problema di classificazione binaria, pertanto tale test può essere utile per stabilire se vi sia una differenza statisticamente significativa tra le medie di una variabile continua tra due gruppi (il gruppo con diabete e il gruppo senza diabete).
Calcoliamo se esiste una differenza significativa tra le medie di ciascuna variabile tra i due gruppi:

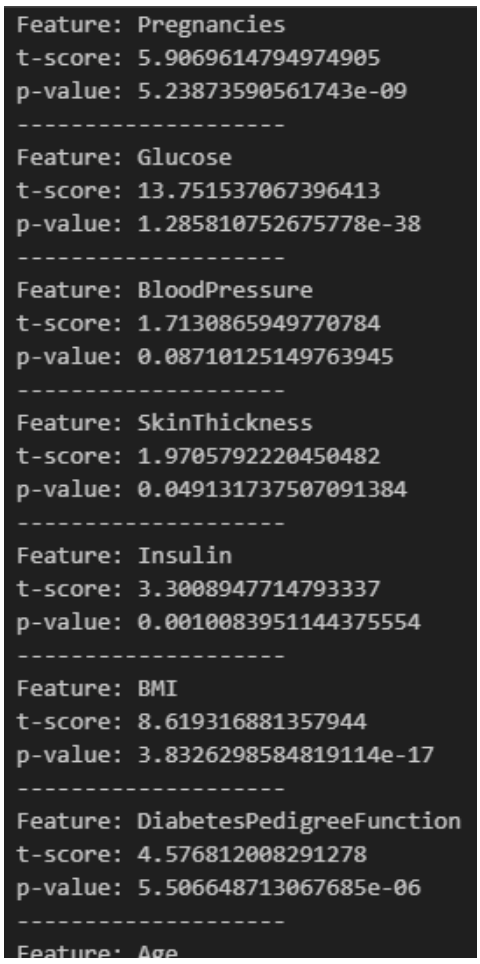


Figura 3.1.4: alcuni risultati del T-Student test

Come si può vedere nella figura (3.1.4), i risultati mostrano i t-scores e i p-values del T-Student test per le variabili del dataset. Un t-score elevato e un p-value basso indicano che esiste una differenza statisticamente significativa tra le medie dei due gruppi per quella variabile. Ad esempio, per la variabile Glucose, il t-score è 13,751537067396413 e il p-value è 1,285810752675778e-38, a indicare che esiste una differenza molto significativa tra le medie dei due gruppi per questa variabile. Analogamente, per la variabile Pregnances, il t-score è 5,9069614794974905 e il p-value è 5,23873590561743e-09, a indicare che anche per questa variabile esiste una differenza significativa tra le medie dei due gruppi. D'altra parte, per la variabile BloodPressure, il t-score è 1,7130865949770784 e il p-value è 0,08710125149763945, indicando che non vi è alcuna differenza statisticamente significativa tra le medie dei due gruppi per questa variabile.

4 Pre-Processing

Passiamo ora alla parte di pre-processing tenendo conto di quanto emerso nella parte di analisi del dataset.
Come abbiamo visto dalle analisi precedenti, nel dataset non ci sono valori nulli, ma ci sono valori errati impostati a 0 (outlier). Quindi, non appena leggiamo i dati, la prima cosa da fare è sostituire questi valori mancanti. Utilizziamo un oggetto *SimpleImputer* con la strategia di imputazione impostata su "mean" (media) e il valore mancante impostato su 0. Questo oggetto viene utilizzato per imputare i valori mancanti in tutte le colonne del dataset, tranne la colonna "Outcome". L'imputazione avviene richiamando il metodo *fit_transform* dell'oggetto imputer sulle colonne rilevanti del dataset.
Una volta imputati i valori mancanti, separiamo le capacità di input dalla classe di output eliminando la colonna "Outcome" dal dataset e memorizzandola in una variabile chiamata features. La colonna

"Outcome" viene memorizzata in una variabile separata chiamata labels.
Successivamente aumentiamo la rilevanza delle feature significative trovate nell'analisi del dataset precedente, che sono: 'Pregnancies', 'Glucose', 'BMI', 'DiabetesPedigreeFunction', 'Age'.

4.1 Normalizzazione dei dati

Successivamente, si crea un oggetto **MinMaxScaler** e lo si adatta alle feature di input richiamando il suo metodo fit. Le feature di input vengono quindi trasformate utilizzando lo scaler, richiamando il suo metodo transform e memorizzando il risultato in una variabile chiamata *features_normalized*. La normalizzazione è un passo importante nella fase di pre-processing dei dati per l'apprendimento automatico, perché può contribuire a migliorare le prestazioni degli algoritmi. La normalizzazione trasforma i valori delle colonne numeriche del dataset in una common scale, senza distorcere le differenze negli intervalli di valori. Ciò può migliorare l'accuratezza dei risultati e ridurre la complessità temporale degli algoritmi. Inoltre, la normalizzazione può aiutare a prevenire l'overfitting e a migliorare la stabilità dell'addestramento del modello.

4.2 Splitting datas

I dati vengono quindi suddivisi in training, validation e test set utilizzando la funzione di scikit-learn *train_test_split*. La dimensione del test set è impostata a 0,3 (30%). Il training set è il restante 70% che viene ulteriormente suddiviso in un training set e un validation set con una dimensione di validazione di 0,2 (30% del training set).
Il validation set è un data set utilizzato per valutare la capacità di generalizzazione del modello durante l'addestramento. È viene utilizzato per ottimizzare gli iperparametri del modello. Si chiama validation set perché serve a convalidare i risultati ottenuti nel training set. Se le prestazioni sono scarse, dobbiamo modificare gli iperparametri del modello e ricominciare l'addestramento fino a quando il risultato della validazione ci soddisfa. In questo modo, possiamo evitare il fenomeno dell'overfitting, in cui il modello è in grado di prevedere perfettamente i dati utilizzati nell'addestramento ma non è in grado di generalizzare su nuovi dati.

4.3 Bilanciamento delle classi

Poiché sappiamo che il dataset non è bilanciato, la ponderazione delle classi viene calcolata utilizzando la funzione *compute_class_weight* di scikit-learn con il peso delle classi impostato su "balanced", le classi impostate sulle unique labels delle classi in train_labels e y impostata su train_labels. I pesi delle classi risultanti sono memorizzati in un dizionario *chiamato weight_class_dict*.

4.4 Salvataggio dei dati

Infine, tutti i dati pre-elaborati vengono salvati nella cartella **Pre_Processed_Data** utilizzando la funzione *save* di numpy.

Quindi abbiamo:

- class_weights.npy
- test_features.npy
- test_labels.npy
- train_features.npy
- train_labels.npy
- val_features.npy
- val_labels.npy

5 Algoritmi di classificazione

Per questo dataset ho scelto di sperimentare 3 algoritmi di classificazione, che sono:

- K-nearest-neighbor
- Decision Tree e Random Forest

5.1 K-nearest-neighbor

L'algoritmo dei k-Nearest Neighbors (k-NN) è un tipo di instance-based learning, o lazy learning, in cui la funzione viene approssimata solo localmente e tutti i calcoli vengono rimandati fino alla funzione di valutazione. È un metodo non parametrico utilizzato per la classificazione e la regressione.
Nella classificazione K-NN, l'output è una membership class. Un oggetto viene classificato a maggioranza dai suoi vicini e viene assegnato alla classe più comune tra i suoi k vicini (k è un numero intero positivo, in genere piccolo).

5.2 Oversampling

Per questo modello ho deciso di applicare SMOTE per bilanciare i dati, invece di prendere in considerazione i pesi della classe di minoranza calcolati nella parte di pre-processing.
SMOTE è l'acronimo di Synthetic Minority Oversampling Technique. Si tratta di una tecnica di oversampling per risolvere il problema dello squilibrio delle classi nei training set. Invece di duplicare semplicemente gli esempi della classe minoritaria, SMOTE genera nuovi esempi sintetici.
SMOTE funziona selezionando esempi vicini nello spazio delle features, tracciando una linea tra gli esempi nello spazio delle features e disegnando un nuovo campione in un punto lungo quella linea. In particolare, viene prima scelto casualmente un esempio dalla classe di minoranza. Successivamente vengono individuati k dei vicini più prossimi per quell'esempio (tipicamente k=5). Viene quindi generato un nuovo esempio sintetico come combinazione lineare dell'esempio scelto e di uno dei suoi vicini più prossimi.
Questa tecnica può essere molto efficace nel fornire informazioni aggiuntive al modello per aiutarlo ad apprendere il confine decisionale tra le classi.
Quindi, SMOTE viene applicato al training set per bilanciare le classi. Ciò avviene generando nuove istanze sintetiche della classe minoritaria fino a quando il numero di istanze per ciascuna classe è uguale.

5.2.1 Grid search

Utilizzo una grid search per trovare i migliori iperparametri per il modello sul validation set.
La Grid Search è un metodo per selezionare i migliori iperparametri per un modello di machine learning. Per ogni iperparametro viene definita una griglia di valori possibili e il modello viene addestrato e valutato per ogni combinazione di iperparametri della griglia. La combinazione di iperparametri che produce le migliori prestazioni del modello viene selezionata come migliore. Nel codice, la griglia di iperparametri è definita per il parametro *n_neighbors* del classificatore k-NN e ogni combinazione di valori della griglia viene testata utilizzando il **validation set** per valutare le prestazioni del modello. La migliore combinazione di iperparametri trovata viene quindi utilizzata per addestrare il modello finale.
Questo è ciò che accade nel codice:

1. Viene specificata una griglia di valori per l'iperparametro n del classificatore K-NN.
2. Viene inizializzata una variabile per tenere traccia dei migliori iperparametri e del miglior punteggio f1 trovati durante la ricerca della griglia.
3. Il codice esegue un loop su ogni valore di n_neighbors nella griglia e addestra un modello k-NN con quel valore di *n_neighbors* utilizzando il training set bilanciato. Le prestazioni del modello vengono valutate sull'insieme di validazione utilizzando il punteggio f1 come parametro.
4. Se l'f1-score del modello sul validation set è migliore dell'attuale f1-score trovato, i migliori iperparametri e il migliore f1-score vengono aggiornati con i valori attuali.

5. Infine, viene addestrato un modello k-NN finale utilizzando i migliori iperparametri trovati e l'insieme di allenamento bilanciato e le previsioni vengono fatte sull'insieme di test.

Ho scelto di utilizzare l'F1-score come metrica per valutare le prestazioni sul validation set perché esso tiene conto sia dei falsi positivi (persone sane classificate come malate) sia dei falsi negativi (persone malate classificate come sane) e può fornire una visione più completa delle prestazioni del modello rispetto all'accuratezza, che potrebbe non essere la scelta migliore come metrica per trovare i migliori iperparametri poiché potrebbe essere influenzata dallo sbilanciamento delle classi.

5.3 Decision Tree

Un Decision Tree è un tipo di algoritmo di apprendimento supervisionato utilizzato per task di classificazione e regressione. Ha una struttura gerarchica ad albero che consiste in un nodo radice, rami, nodi interni e nodi foglia.
Nel contesto della classificazione, un decision tree funziona dividendo ricorsivamente i dati in sottoinsiemi basati sui valori delle feature in ingresso. A ogni nodo interno dell'albero, viene eseguito un test su una delle feature in ingresso per determinare quale ramo seguire. Il test confronta il valore della feature con una soglia e invia i dati al nodo figlio sinistro o destro, a seconda che il valore sia inferiore o superiore alla soglia.
L'albero viene fatto crescere finché tutti i dati di un nodo foglia non appartengono alla stessa classe o finché non viene soddisfatto un criterio di arresto. Una volta cresciuto, l'albero può essere utilizzato per fare previsioni su nuovi dati partendo dal nodo radice e seguendo i rami fino a un nodo foglia in base ai valori delle feature di input. La label della classe al nodo foglia viene quindi restituita come predizione.

5.3.1 Algoritmo Genetico per trovare i migliori iperparametri

L'algoritmo genetico è un metodo di ottimizzazione ispirato alla selezione naturale e alla genetica. L'algoritmo genetico viene utilizzato per trovare i migliori iperparametri per il decision tree sul validation set.
L'algoritmo inizia creando una popolazione di individui, dove ogni individuo rappresenta un insieme di iperparametri. Gli iperparametri nel codice sono *max_depth* e *min_samples_split*. La popolazione viene quindi valutata utilizzando la funzione fitness, che in questo caso calcola l'F1-score del modello sul validation set utilizzando gli iperparametri correnti. Naturalmente, per selezionare gli iperparametri utilizziamo ancora una volta l'F1-score, poiché si tratta di un problema di classificazione binaria.
Successivamente, l'algoritmo esegue una serie di operazioni genetiche per creare una nuova generazione di individui. Queste operazioni comprendono la selezione (per scegliere gli individui più adatti alla riproduzione), il crossover (per scambiare informazioni genetiche tra due individui) e la mutazione (per introdurre variazioni casuali negli individui).
Questo processo viene ripetuto per un numero predefinito di generazioni. Alla fine, l'algoritmo restituisce il miglior individuo trovato, che rappresenta l'insieme di iperparametri che massimizza la funzione fitness.
All'interno del codice, l'algoritmo genetico è implementato utilizzando la libreria DEAP. Gli operatori genetici (valutazione, crossover, mutazione e selezione) vengono registrati e l'algoritmo genetico viene eseguito utilizzando la funzione *eaSimple* della libreria DEAP.
Il metodo utilizzato nell'algoritmo genetico è la tournament selection. In questo metodo, un gruppo casuale di individui viene scelto dalla popolazione e l'individuo con il fitness-value più alto viene selezionato per la riproduzione. Questo processo viene ripetuto più volte per scegliere altri individui da allevare.
Nel codice viene utilizzato il crossover a due punti, che seleziona due punti casuali e scambia le informazioni genetiche tra i due per creare due nuovi individui.
Viene utilizzata anche la mutazione uniforme, che cambia casualmente il valore di uno o più geni dell'individuo all'interno di un intervallo predefinito.
Il parametro tournsize controlla il numero di individui che partecipano a ciascun tournament. Un valore più alto di *tournsize* aumenta la pressione selettiva, favorendo gli individui con fitness-value più elevati. Un valore più basso di tournsize riduce invece la pressione selettiva e aumenta la diversità genetica della popolazione.

Il metodo di selezione dei tornei viene registrato con la funzione *selTournament* e il parametro tournsize è impostato su 3.
Questo processo viene ripetuto per un numero predefinito di generazioni. Alla fine, l'algoritmo restituisce il miglior individuo trovato, che rappresenta l'insieme di iperparametri che massimizza la funzione di fitness.

5.3.2 Training del modello

Dopo aver trovato i migliori iperparametri con l'algoritmo genetico, ho istanziato un modello di decision tree passando come feature le classi pesate e i migliori parametri trovati con l'algoritmo genetico, ho addestrato il modello tramite la funzione *fit* e infine ho usato il modello addestrato per fare previsioni sul set test.

5.4 Random Forest

Dopo il Decision Tree, ho deciso di costruire anche un Random Forest, utilizzando lo stesso approccio di ricerca degli iperparametri usato per il Decision Tree e, naturalmente, tenendo conto anche delle classi pesate.
Il Random Forest è un metodo di ensemble learning, il che significa che consiste in una serie di classificatori (in questo caso, decision tree) e le loro previsioni vengono aggregate per identificare l'outcome più prevalente. Ciò può contribuire a migliorare le prestazioni del modello rispetto all'utilizzo di un singolo decision tree.

6 Risultati sul test set

Come metriche per valutare i risultati sul test set, ho deciso di utilizzare le seguenti:

- **Matrice di confusione:** La matrice di confusione è una tabella spesso utilizzata per descrivere le prestazioni di un modello di classificazione. Mostra il numero di true positive (TP), false positive (FP), true negative (TN) e false negative (FN) per ogni classe.
- **Accuracy:** è una misura delle prestazioni di un modello di classificazione. È definita come il rapporto tra il numero di previsioni corrette e il numero totale di previsioni. In altre parole, misura quanti casi sono stati classificati correttamente dal modello.
- **Classification report:** Il Classification report fornisce una ripartizione dettagliata delle prestazioni dei modelli per ciascuna classe. Include diverse metriche come precision, recall e f1-score. La precision è il rapporto tra i true positive e la somma dei true positive e dei false positive. Misura quante delle previsioni positive fatte dal modello sono effettivamente positive. Il recall è il rapporto tra i true positive e la somma dei true positive e dei false negative. Misura quanti dei casi effettivamente positivi sono stati identificati correttamente dal modello. L'F1-score è la media armonica di precision e recall. Fornisce un'unica misura che bilancia sia la precision che il recall.
- **ROC:** ROC è l'acronimo di Receiver Operating Characteristic. È un grafico che illustra le prestazioni di un classificatore binario al variare della sua soglia di discriminazione. La curva ROC viene creata tracciando il tasso di true positive (TPR) rispetto al tasso di false positive (FPR) a varie impostazioni di soglia.
- **AUC:** L'AUC può essere interpretato come la probabilità che un caso positivo scelto a caso venga classificato dal classificatore come superiore a un caso negativo scelto a caso. Un'AUC elevata indica che il classificatore è in grado di discriminare efficacemente tra casi positivi e negativi.
- **Intervallo di confidenza:** L'intervallo di confidenza è uno strumento statistico utilizzato per stimare l'intervallo in cui si trova il valore vero di un parametro sconosciuto con un certo livello di confidenza. Può essere utilizzato per stimare l'intervallo in cui si trova il vero tasso di errore di un modello su un set di dati target.
  L'intervallo di confidenza è particolarmente utile quando si dispone solo di un piccolo test set

(come nel mio caso), poiché fornisce una stima dell'incertezza associata alla stima del tasso di errore.
Ho deciso di impostare l'intervallo di confidenza al 95%

- **K-fold cross validation** sul training set: è una tecnica utilizzata per valutare le prestazioni di un modello e per mettere a punto i suoi iperparametri. Si tratta di dividere i dati in k subsets, quindi di addestrare e valutare il modello k volte, utilizzando ogni volta un fold diverso con validation set.

6.1 Risultati sul KNN

L' accuracy sul test set è 0.70, questo significa che il 70% dei casi di test è stato classificato correttamente.
L'intervallo di confidenza è $[0.6977, 0.8088]$.
Il report di classificazione in fig.(6.1.1) mostra come il KNN abbia una precision di 0.87 e una recall di 0.64 per la classe 0, e una precision di 0.55 e una recall di 0.82 per la classe 1. La precision per la classe 0 è di 0.87, questo significa che di tutti i casi che si prevedeva fossero di classe 0, solo l' 87% erano di classe 0. La precision per la classe 1 è di 0.55, questo significa che di tutti i casi che si prevedeva fossero di classe 1, solo il 55% erano di classe 1.
Il recall per la classe 0 è di 0.64, questo significa che di tutti i casi di classe 0, il 64% sono stati correttamente previsti come classe 0. Il recall per la classe 1 è di 0.82, questo significa che di tutti i casi previsti come classe 1, l' 82% sono stati previsti come classe 1.
L' f1-score per la classe 0 è di 0.74 e per la classe 1 è di 0.66.

Classification Report:

	precision	recall	f1-score	support
0	0.87	0.64	0.74	151
1	0.55	0.82	0.66	80
accuracy			0.70	231
macro avg	0.71	0.73	0.70	231
weighted avg	0.76	0.70	0.71	231

Figura 6.1.1: Classification Report

La matrice di confusione in fig.(6.1.2) mostra come di 151 casi negativi (class 0), solo 96 sono stati correttamente predetti come tali (true negatives) e 55 sono stati erroneamente predetti come positivi (false positives). Su 80 casi positivi effettivi (classe 1), 66 sono stati correttamente predetti come positivi (veri positivi) e 14 sono stati erroneamente predetti come negativi (falsi negativi).

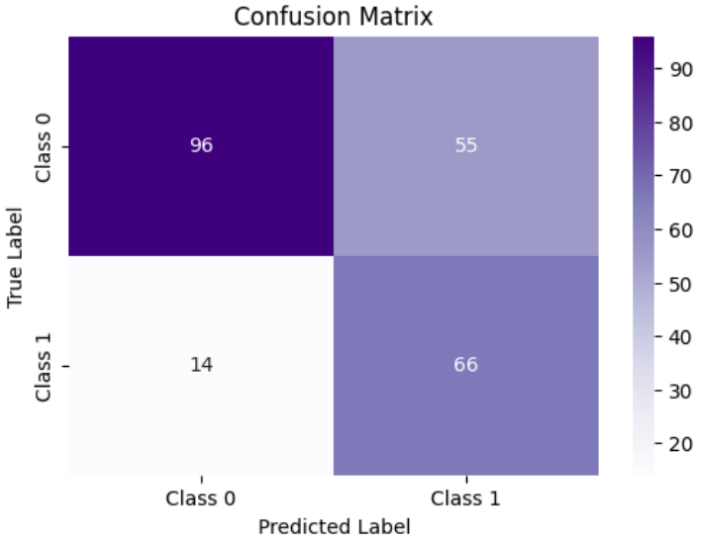


Figura 6.1.2: Confusion Matrix

L'area sotto la curva ROC in fig.(6.1.3) è 0.73, che indica che il modello ha una buona capacità di discriminazione.

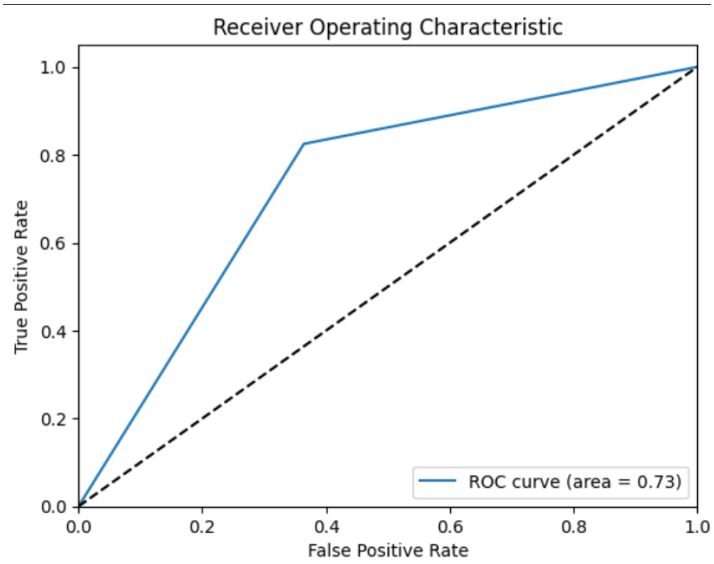


Figura 6.1.3: ROC

Guarando ai risultati della k-fold cross validation sul training set, possiamo notare che in media, il classificatore knn classifica correttamente approssimativamente il 76.93% dei casi in ogni fold. L'accuratezza varia leggermente da fold a fold, con un minimo di 73.26% a un massimo di 80.23%.

```
Accuracy scores for each fold: [0.73255814 0.80232558 0.73255814 0.77906977 0.8        ]
Mean accuracy: 0.7693023255813953
```

Figura 6.1.4: Risultati della cross validation

Questi risultati ci fanno comprendere come il classificatore knn sia in grado di ottenere prestazioni ragionevolmente buone su questo genere di problema.

6.2 Risultati sul Decision Tree e Random Forest

6.2.1 Risultati sul Decision Tree

Dai risultati forniti, l'algoritmo del decision tree ha ottenuto un'accuracy sul test set pari a 70%, indicando che il modello è in grado di classificare correttamente il 70% delle istanze nel test set. L'intervallo di confidenza è $[0,6349, 0,7526]$. La matrice di confusione fig.(6.2.1) indica i seguenti risultati:

- 100 istanze sono state classificate correttamente come "0" (veri negativi).
- 51 istanze sono state classificate erroneamente come "1" (falsi positivi).
- 19 istanze sono state classificate erroneamente come "0" (falsi negativi).
- 61 istanze sono state classificate correttamente come "1" (veri positivi).

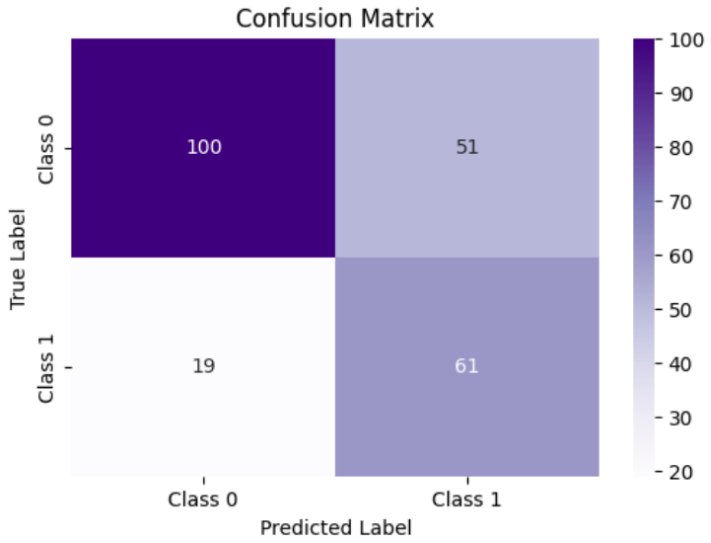


Figura 6.2.1: Matrice di confusione

La precision in fig.(6.2.2) per la classe "0" è di 84%, il che significa che di tutte le istanze classificate come "0", 84% di esse sono effettivamente "0". Il recall per la classe "0" è pari a 66%, il che indica che il 66% di tutte le istanze "0" presenti nel set di test sono state correttamente identificate come tali dal modello.

Per la classe "1", la precision in fig.(6.2.2) è di 54%, il che indica che di tutte le istanze classificate come "1", solo 54% di esse sono effettivamente "1". Il recall per la classe "1" è di 76%, il che significa che 76% di tutte le istanze "1" nel set di test sono state correttamente identificate come tali dal modello.

L' f1-score in fig.(6.2.2) per la classe "0" è 0,74, mentre per la classe "1" è 0,64. La media ponderata dell'accuracy è di 0,70, che tiene conto della distribuzione delle classi nell'insieme di test.

Classification Report:

	precision	recall	f1-score	support
0	0.84	0.66	0.74	151
1	0.54	0.76	0.64	80
accuracy			0.70	231
macro avg	0.69	0.71	0.69	231
weighted avg	0.74	0.70	0.70	231

Figura 6.2.2: Classification Report

L'area sotto la curva ROC (AUC) fig.(6.2.3) è pari a 0,712, indicando una buona capacità del modello di classificare correttamente le istanze positive rispetto a quelle negative.

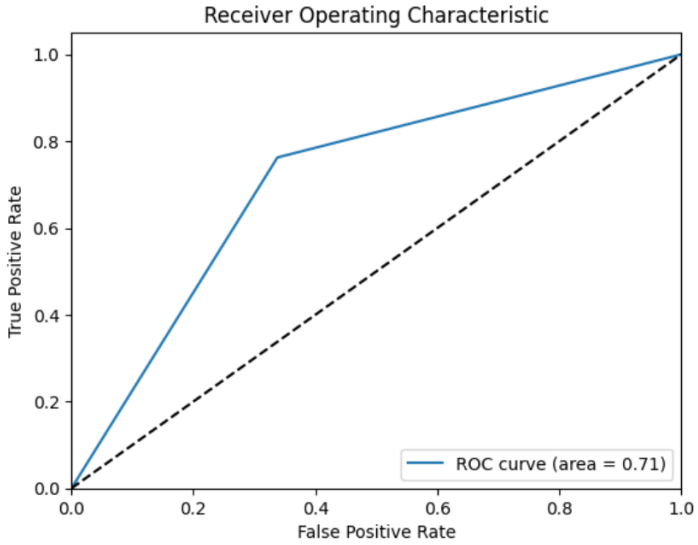


Figura 6.2.3: ROC

Concludiamo dicendo che i risultati mostrano che il Decision Tree ha una performance buone nel classificare il diabete nel dataset utilizzato. Vediamo ora i risultati del Random Forest.

6.2.2 Risultati del Random Forest

L'algoritmo Random Forest ha ottenuto un'accuratezza sul test set del 73%, leggermente superiore all'algoritmo Decision Tree.
L'intervallo di confidenza è $[0.6619, 0.7767]$.
La matrice di confusione in fig.(6.2.4) mostra i seguenti risultati:

- 105 istanze sono state classificate correttamente come "0" (veri negativi).
- 46 istanze sono state classificate erroneamente come "1" (falsi positivi).
- 16 istanze sono state classificate erroneamente come "0" (falsi negativi).
- 64 istanze sono state classificate correttamente come "1" (veri positivi).

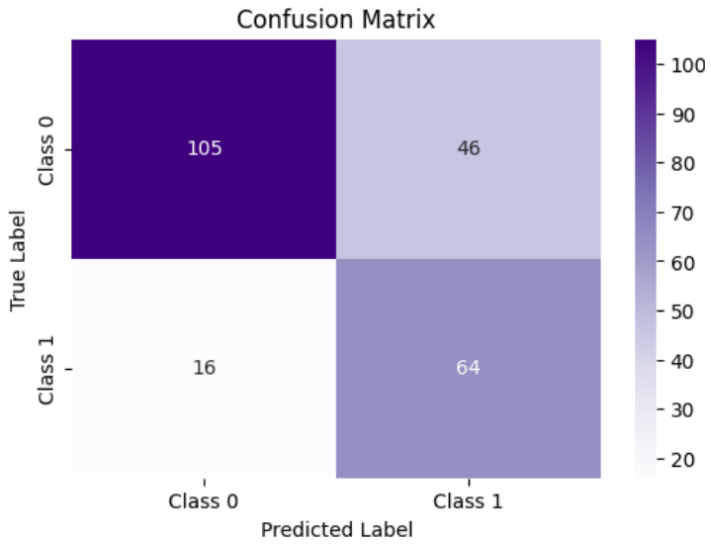


Figura 6.2.4: Confusion Matrix

La precision fig.(6.2.5) per la classe "0" è pari a 87%, indicando che di tutte le istanze classificate come "0", 87% di esse sono effettivamente "0". Il recall per la classe "0" è di 70%, il che indica che 70% di tutte le istanze "0" nel test set sono state correttamente identificate come tali dal modello.
Per la classe "1", la precision è di 58%, a indicare che di tutte le istanze classificate come "1", solo 58% di esse sono effettivamente "1". Il recall per la classe "1" è di 80%, il che significa che 80% di tutte le istanze "1" nel test set sono state correttamente identificate come tali dal modello.
L'F1-score per la classe "0" è di 0,77, mentre per la classe "1" è di 0,67. La media ponderata dell'accuratezza è di 0,73, che tiene conto della distribuzione delle classi nel test set.

Classification Report:

	precision	recall	f1-score	support
0	0.87	0.70	0.77	151
1	0.58	0.80	0.67	80
accuracy			0.73	231
macro avg	0.72	0.75	0.72	231
weighted avg	0.77	0.73	0.74	231

Figura 6.2.5: Classification report

L'area sotto la curva ROC (AUC) fig.(6.2.6) è pari a 0,747, indicando una buona capacità del modello di classificare correttamente le istanze positive rispetto a quelle negative.

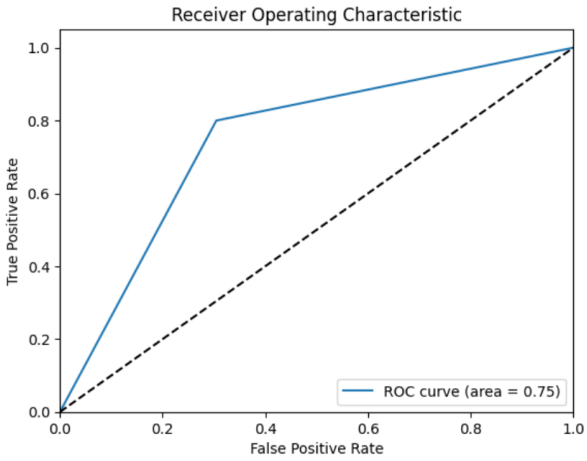


Figura 6.2.6: ROC

Pertanto, possiamo confermare che il Random Forest ha ottenuto risultati migliori rispetto al Decision Tree.

6.2.3 Risultati della Cross-Validation

- Decision Tree:
  - Accuracy scores per ogni fold: [0.75581395, 0.6744186, 0.72093023, 0.69767442, 0.65882353]
  - Accuracy media: 0.701532147742818
- Random Forest:
  - Accuracy scores per ogni fold: [0.6627907, 0.70930233, 0.72093023, 0.70930233, 0.74117647]
  - Accuracy media: 0.7087004103967168

Considerando l'accuracy media, si osserva che quella del Random Forest è leggermente superiore a quella del Decision Tree (0,7087 contro 0,7015). Ciò suggerisce che il Random Forest potrebbe avere una maggiore capacità di generalizzare e adattarsi ai dati. In effetti, il calcolo delle metriche conferma che il Random Forest è leggermente migliore del Decision Tree in questo contesto.

6.3 Conclusioni generali

In generale, possiamo concludere che tutti e tre i modelli riescono a classificare in modo discreto tutti i dati del dataset. Facciamo più fatica con le istanze della classe 1, probabilmente perché il set di test è più popolato da istanze della classe 0.