UNIVERSITÀ POLITECNICA DELLE MARCHE
FACOLTÀ DI INGEGNERIA
Dipartimento di Ingegneria dell'Informazione
Corso di Laurea in Ingegneria Informatica e dell'Automazione

ANALISI E CLASSIFICAZIONE DELLE CONDIZIONI DI PESO MEDIANTE ALBERI DECISIONALI IN PROLOG

Studio comparativo tra gli algoritmi Gini e C4.5 nell'identificazione di diverse condizioni di peso

Autore

Ciotti Niccolò
Renzi Luca
Sbattella Mattia

ANNO ACCADEMICO 2024-2025

Indice

Elenco delle figure

Elenco delle tabelle

CAPITOLO 1

Introduzione

Questa sezione ha lo scopo di introdurre gli obiettivi del progetto. In particolare, verrà presentato il concetto di albero decisionale e verranno mostrati due criteri per la scelta degli attributi, il Gini e il C4.5.

1.1 Scopo del progetto

L'obiettivo del progetto è quello di sviluppare un modello previsionale in grado di apprendere delle informazioni sul peso delle persone, generare un albero decisionale usando i criteri Gini e C4.5 e mostrare le relative statistiche.

1.2 Albero decisionale

Un albero decisionale per l'apprendimento automatico è un modello di previsione che utilizza una struttura ad albero per rappresentare le decisioni e le loro possibili conseguenze. È uno degli algoritmi più semplici e intuitivi per l'analisi dei dati, ed è particolarmente utile per problemi di classificazione (dove l'obiettivo è assegnare una categoria a un dato) e regressione (dove si cerca di prevedere un valore continuo). L'albero decisionale lavora prendendo in considerazione una serie di caratteristiche dei dati di input e applicando una sequenza di condizioni per determinare quale classe (nel caso della classificazione) o valore numerico (nel caso della regressione) è il più probabile. Il processo di costruzione di un albero decisionale consiste nel selezionare la caratteristica più significativa a ciascun livello dell'albero per suddividere i dati in modo da ridurre l'incertezza (ad esempio, la entropia o l'impurità di Gini).

1.2.1 Struttura di un albero decisionale

Un albero decisionale è composto da vari nodi:

- Nodo radice: il nodo iniziale che rappresenta il punto di partenza della decisione. Il modello prende una decisione iniziale basata su una caratteristica dei dati (ad esempio, "se l'età è maggiore di 30").
- Nodi interni: ogni nodo interno rappresenta una condizione basata su una delle caratteristiche dei dati. Questi nodi dividono i dati in base a regole di decisione.
- Rami: sono le connessioni tra i nodi, che rappresentano il risultato delle condizioni. Ogni ramo indica il risultato di una decisione (ad esempio, "vero" o "falso").
- Nodi foglia: i nodi finali dell'albero, che contengono la previsione o la decisione finale. In un problema di classificazione, il nodo foglia conterrà la classe prevista, mentre in un problema di regressione conterrà il valore numerico previsto.

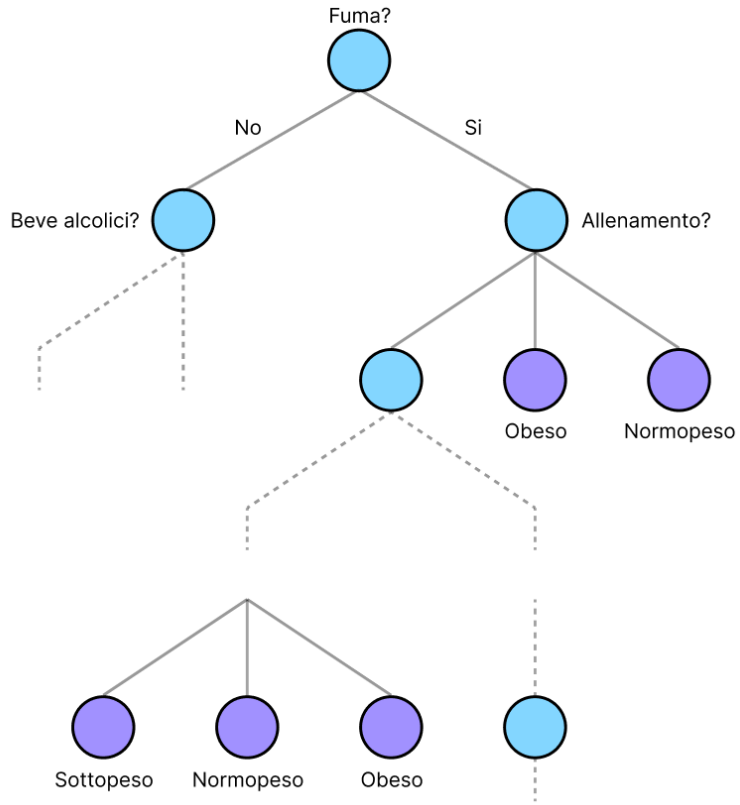

Figura 1.1: Esempio di albero decisionale

1.2.2 Criterio di Gini

Il criterio di Gini misura l'impurità di un insieme di dati e aiuta a scegliere l'attributo che meglio suddivide i dati in modo che i gruppi risultanti siano il più omogenei possibile (ovvero, contengano elementi della stessa classe).

L'indice di Gini per un nodo è calcolato come:

$$Gini(D) = 1 - \sum_{i=1}^{k} p_i^2$$

dove:

- D è l'insieme dei dati nel nodo
- p_i è la probabilità di un elemento appartenente alla classe i
- k è il numero di classi

L'indice di Gini assume valori tra 0 e 1:

- $Gini = 0$ indica che tutti gli esempi nel nodo appartengono alla stessa classe (perfetta purezza)
- $Gini = 1$ indica che gli esempi sono distribuiti equamente tra tutte le classi (massima impurità)

L'obiettivo è minimizzare l'indice di Gini per i nodi successivi. Per ogni possibile attributo, si calcola l'indice di Gini per ciascun possibile valore (o intervallo per attributi continui) e si sceglie l'attributo che riduce al minimo l'impurità.

1.2.3 Criterio C4.5

L'algoritmo *C4.5* utilizza il concetto di *guadagno di informazione* per scegliere l'attributo migliore. Il guadagno di informazione misura quanto un attributo riduce l'incertezza (o entropia) nei dati.

La formula del guadagno di informazione rispetto a un attributo A è:

$$\text{Guadagno}(D, A) = \text{Entropia}(D) - \sum_{v \in \text{valori}(A)} \frac{|D_v|}{|D|} \cdot \text{Entropia}(D_v)$$

dove:

- D è l'insieme dei dati nel nodo
- A è l'attributo che stiamo considerando per la divisione
- v sono i possibili valori di A
- D_v è il sottoinsieme di D in cui l'attributo A ha valore v

- Entropia(D) è l'entropia dell'insieme D, che misura l'incertezza dei dati
- Entropia(D_v) è l'entropia del sottoinsieme D_v

L'entropia di un insieme D si calcola come:

$$\text{Entropia}(D) = -\sum_{i=1}^{k} p_i \log_2(p_i)$$

dove p_i è la probabilità di ciascuna classe nel nodo.

Guadagno di Informazione Relativo (C4.5)

C4.5 utilizza anche il *guadagno di informazione relativo*, che normalizza il guadagno di informazione per tenere conto del numero di valori possibili di un attributo. La formula del guadagno di informazione relativo è:

$$\text{Guadagno Normalizzato}(D, A) = \frac{\text{Guadagno}(D, A)}{\text{Entropia}(A)}$$

dove Entropia(A) è l'entropia dell'attributo A, che misura l'incertezza dovuta alla varietà dei suoi valori. C4.5 preferisce gli attributi con il maggior guadagno normalizzato, poiché questi sono considerati i più informativi.

CAPITOLO 2

Implementazione Prolog

Questo capitolo si concentra sull'implementazione del sistema Prolog utilizzato nell'analisi e classificazione delle condizioni di peso. Verranno esaminate la struttura dei dati, necessaria per rappresentare termini e clausole nell'ambito degli alberi decisionali, e la struttura del codice, con particolare attenzione agli algoritmi Gini e C4.5.

2.1 Dataset

Il dataset rappresenta la base fondamentale per l'analisi e la classificazione delle condizioni di peso. Esso contiene una raccolta di dati relativi a caratteristiche, come genere, età, peso, altezza, abitudini alimentari, etc. Di seguito è riportato un esmpepio di alcuni record del dataset.

La seguente tabella riporta alcune righe selezionate dal Dataset:

Gender	Age	Height	Weight	Family History	FAVC	NCP	SMOKE	CH2O	FAF	CALC	MTRANS	Obesity
Female	Youth	Average	Normal	yes	no	3	no	2	0	no	Public Transportation	Normal Weight
Female	Youth	Average	Light	yes	no	3	yes	3	3	Sometimes	Public Transportation	Normal Weight
Male	Youth	Average	Normal	yes	no	3	no	2	2	Frequently	Public Transportation	Normal Weight
Male	Youth	Average	Heavy	no	no	3	no	2	2	Frequently	Walking	Overweight
Male	Youth	Average	Heavy	no	no	1	no	2	0	Sometimes	Public Transportation	Overweight
Male	Youth	Average	Light	no	yes	3	no	2	0	Sometimes	Automobile	Normal Weight
Female	Youth	Short	Light	yes	yes	3	Yo	2	1	Sometimes	Motorbike	Normal Weight
Male	Youth	Tall	Heavy	yes	yes	3	no	3	2	Sometimes	Public Transportation	Obesity

Tabella 2.1: Alcune righe del Dataset

dove:

- *Gender* indica il genere della persona
- *Age* indica l'età della persona

- *Height* indica l'altezza della persona
- *Weight* indica il peso della persona
- *FamilyHistory* indica se la persona ha qualche membro della famiglia obeso
- *FAVC* indica se la persona assume cibi calorici
- *NCP* indica il numero di pasti giornalieri
- *SMOKE* indica se la persona fuma
- *CH2O* indica quanta acqua assuma la persona
- *FAF* è la frequenza di attività fisica
- *CALC* è la frequenza di assunzione di alcool
- *MTRANS* indica quali mezzi di trasporto vengono usati
- *Obesity* indica il livello di peso della persona

La struttura del codice in questo progetto è organizzata in modo da permettere una gestione efficiente dei dati e un'analisi performante dei risultati, in particolare per quanto riguarda l'implementazione degli algoritmi decisionali Gini e C4.5. In questa sezione, verranno descritti i principali moduli del codice, insieme alla gestione del training set e del test set.

2.1.1 Pulizia del Dataset

Il Dataset originale include vari attributi, tra cui alcune variabili numeriche come l'età, altezza e peso e dopo un'ispezione preliminare, è stata applicata una tecnica di discretizzazione per rendere i dati più interpretabili e gestibili in un contesto simbolico. In particolare, sono stati trasformati i valori numerici dell'età in intervalli categoriali, suddividendoli nelle categorie "Youth" (per età inferiori a 30 anni), "Adult" (tra 30 e 50 anni) e "Senior" (per età superiori a 50 anni). La stessa tecnica è stata applicata per gli attributi riguardanti l'altezza ed il peso. Una successiva pulizia è stata effettuata rimuovendo alcuni attributi dalla tabella originale poichè ritenuti poco pertinenti. Il passo successivo è stato quello di trasformare ogni riga della tabella in un fatto Prolog. Sono stati creati due file, uno contente il Dataset dove ogni elemento ha la seguente struttura:

```
e(Classe,[Att=Val]).
```

e l'altro contenente tutti gli attributi con i relativi valori.

```
a(gender,['Female','Male']).
a(age,['Youth', 'Adult', 'Senior']).
a(height,['Average','Short','Tall']).
```

```
a(weight,['Heavy','Light','Normal']).
a(family,['no','yes']).
a(favc,['no','yes']).
a(ncp,[1,2,3,4]).
a(smoke,['no','yes']).
a(water,[1,2,3]).
a(activity,[0,1,2,3]).
a(alcohol,['Always','Frequently','Sometimes','no']).
a(mtrans,['Automobile','Bike','Motorbike','Public_Transportation','
    Walking'])..
```

2.1.2 Training Set

Il Training Set è utilizzato per l'addestramento degli alberi decisionali ed è costituito dal 70% dei dati del Dataset. Il Training Set è caricato in memoria tramite un processo di parsing, dove i dati vengono separati e trasformati in predicati Prolog con la struttra e(Classe, [Att=Val]). Di seguito è mostrato un esempio:

Gender	Age	Height	Weight	Family History	FAVC	NCP	SMOKE	CH2O	FAF	CALC	MTRANS	Obesity
Male	Youth	Average	Normal	no	no	3	no	3	2	no	Public Transportation	Normal Weight

Tabella 2.2: Esempio di una riga del Training Set

Dopo aver effettuato il parsing, la riga del Training Set appare come di seguito:

```
e(normopeso,[gender = 'Male', age = 'Youth', height = 'Average',
    weight = 'Normal', family = 'no', favc = 'no', ncp = 3, smoke = '
    no', water = 3, activity = 2, alcohol = 'no', mtrans = '
    Public_Transportation']).
```

2.1.3 Test Set

Il Test Set è utilizzato per il test degli alberi decisionali ed è costituito dal 30% dei dati del Dataset. Anche questo set di dati viene trasformato in predicati Prolog con la seguente struttura, s(Classe, [Att=Val]).

Gender	Age	Height	Weight	Family History	FAVC	NCP	SMOKE	CH2O	FAF	CALC	MTRANS	Obesity
Male	Adult	Average	Heavy	yes	yes	3	no	3	1	Sometimes	Automobile	Obesity

Tabella 2.3: Esempio di una riga del Test Set

Dopo aver effettuato il parsing, la riga del Test Set appare come di seguito:

```
s(obeso,[gender = 'Male', age = 'Adult', height = 'Average', weight =
    'Heavy', family = 'yes', favc = 'yes', ncp = 3, smoke = 'no',
   water = 3, activity = 1, alcohol = 'Sometimes', mtrans = '
   Automobile']).
```

2.1.4 Codice

Il programma è progettato per apprendere modelli di classificazione attraverso Alberi di Decisione, utilizzando tecniche come la disuguaglianza di Gini e l'algoritmo C4.5. Viene impiegato per classificare oggetti (ad esempio, individui con caratteristiche legate al peso come obesi, sovrappeso, normopeso o sottopeso) in base ai loro attributi.

Il predicato *induce_albero* genera un albero di decisione a partire da esempi (insiemi di oggetti con le loro classi) e un insieme di attributi. L'albero è costruito ricorsivamente in base a diverse condizioni:

- Se l'insieme degli esempi è vuoto, l'albero è nullo.
- Se tutti gli esempi appartengono alla stessa classe, l'albero è una foglia che rappresenta quella classe.
- Se gli esempi appartengono a più classi, l'albero è un nodo con un attributo che discrimina tra i vari valori.

```
induce_albero( _, [],_, null ) :- !.          % (1)
induce_albero( _, [e(Classe,_)|Esempi],_, l(Classe)) :- % (2)
      \+ ( member(e(ClassX,_),Esempi), ClassX \== Classe ),!.
induce_albero( Attributi, Esempi, Algoritmo, t(Attributo,SAlberi) ):-

% (3)
      scegli_attributo( Attributi, Esempi, Algoritmo, Attributo), !,
      del( Attributo, Attributi, Rimanenti ),
      a( Attributo, Valori ),
      induce_alberi( Attributo, Valori, Rimanenti, Algoritmo, Esempi,
          SAlberi).
induce_albero( _, Esempi, Algoritmo, l(Classi)) :-
      findall( Classe, member(e(Classe,_),Esempi), Classi).
induce_albero(_, Esempi,_, l(Classi)) :-
   findall(Classe, member(e(Classe,_), Esempi), Classi).

induce_alberi(_, [], _ , _ , _ , []) :- !.

% Caso ricorsivo: per ogni valore di un attributo, genera un
   sottoalbero
induce_alberi(Att, [Val1|Valori], AttRimasti, Algoritmo, Esempi, [
   Val1:Alb1|Alberi]) :-
   attval_subset(Att=Val1, Esempi, SottoinsiemeEsempi),
```

```
induce_albero(AttRimasti, SottoinsiemeEsempi, Algoritmo, Alb1),
   induce_alberi(Att, Valori, AttRimasti,Algoritmo, Esempi, Alberi).

attval_subset(AttributoValore,Esempi,Sottoinsieme) :-
      findall(e(C,O),(member(e(C,O),Esempi),soddisfa(O,[
         AttributoValore])),Sottoinsieme).

soddisfa(Oggetto,Congiunzione) :-
      \+ (member(Att=Val,Congiunzione),
         member(Att=ValX,Oggetto),
         ValX \== Val).

del(T,[T|C],C) :- !.
del(A,[T|C],[T|C1]) :-
      del(A,C,C1).

mostra(T) :-
      mostra(T,0).
mostra(null,_) :- writeln(' ==> ???').
mostra(l(X),_) :- write(' ==> '),writeln(X).
mostra(t(A,L),I) :-
      nl,tab(I),write(A),nl,I1 is I+2,
      mostratutto(L,I1).
mostratutto([],_).
mostratutto([V:T|C],I) :-
      tab(I),write(V), I1 is I+2,
      mostra(T,I1),
      mostratutto(C,I).
```

Il predicato *scegli_attributo* seleziona l'attributo che meglio discrimina tra le classi. Sono supportati due algoritmi di selezione:

- Gini: Si utilizza per calcolare l'indice di disuguaglianza (diversità) tra le classi per ogni attributo e utilizza il setof per ordinare gli attributi in base al valore crescente della loro disuguaglianza.
- C4.5: Utilizza il guadagno informativo, che misura quanto un attributo riduce l'incertezza nella classificazione.

```
scegli_attributo(Attributi, Esempi, Algoritmo, Attributo) :-
      (  Algoritmo == 'Gini'
      -> scegli_attributo_gini(Attributi, Esempi, Attributo)
      ;  Algoritmo == 'C45'
      -> scegli_attributo_C4_5(Attributi, Esempi, Attributo)
      ;  format('Algoritmo non supportato: ', [Algoritmo]),
            fail
      ).
```

```
scegli_attributo_gini( Attributi, Esempi, MigliorAttributo ) :-
      setof( Disuguaglianza/A,
           (member(A,Attributi) , disuguaglianza(Esempi,A,
              Disuguaglianza)),
           [MinorDisuguaglianza/MigliorAttributo|_] ).

scegli_attributo_C4_5(Attributi, Esempi, MigliorAttributo) :-
   setof(Gain/A, (member(A, Attributi), guadagno_informativo(Esempi,
      A, Gain)), [MaxGain/MigliorAttributo|_]).

guadagno_informativo(Esempi, Attributo, Gain) :-
   entropia(Esempi, EntropiaE),
   a(Attributo, Valori),
   somma_entropie(Esempi, Attributo, Valori, Somma),
   Gain is EntropiaE - Somma.

entropia(Esempi, Entropia) :-
   findall(Classe, member(e(Classe,_), Esempi), Classi),
   distribuzione_classi(Classi, Distribuzione),
   calcola_entropia(Distribuzione, Entropia).

distribuzione_classi(Classi, Distribuzione) :-
   length(Classi, N),
   findall(X, (member(X, Classi)), Unici),
   length(Unici, M),
   findall(P, (member(U, Unici), findall(1, (member(U, Classi)), S),
      length(S, L), P is L / N), Distribuzione).

calcola_entropia(Distribuzione, Entropia) :-
   findall(P * log(P), (member(P, Distribuzione), P > 0), LogTerms),
   sum_list(LogTerms, S),
   Entropia is -S.

somma_entropie(Esempi, Attributo, [Val|Valori], Somma) :-
   attval_subset(Attributo=Val, Esempi, Subset),
   entropia(Subset, EntropiaVal),
   length(Subset, NSubset),
   length(Esempi, N),
   Peso is NSubset / N,
   SommaParziale is Peso * EntropiaVal,
   somma_entropie(Esempi, Attributo, Valori, Somma1),
   Somma is SommaParziale + Somma1.
somma_entropie(_, _, [], 0).

somma_pesata( _, _, [], Somma, Somma).
somma_pesata( Esempi, Att, [Val|Valori], SommaParziale, Somma) :-
      length(Esempi,N),
      findall( C,(member(e(C,Desc),Esempi),soddisfa(Desc[Att=Val])),
```

```
          EsempiSoddisfatti ),
      length(EsempiSoddisfatti, NVal),
      NVal > 0, !,
      findall(P,
                        (bagof(1,member(_,EsempiSoddisfatti),L),
                           length(L,NVC),P is NVC/NVal),
            ClDst),
      gini(ClDst,Gini),
      NuovaSommaParziale is SommaParziale + Gini*NVal/N,
      somma_pesata(Esempi,Att,Valori,NuovaSommaParziale,Somma)
      ;
      somma_pesata(Esempi,Att,Valori,SommaParziale,Somma).

gini(ListaProbabilit,Gini) :-
      somma_quadrati(ListaProbabilit,0,SommaQuadrati),
      Gini is 1-SommaQuadrati.
somma_quadrati([],S,S).
somma_quadrati([P|Ps],PartS,S) :-
      NewPartS is PartS + P*P,
      somma_quadrati(Ps,NewPartS,S).
```

Una volta che l'albero di decisione è costruito, il predicato *classifica* permette di determinare a quale classe appartiene un oggetto in base agli attributi dell'oggetto e alla struttura dell'albero. L'algoritmo percorre l'albero partendo dalla radice, facendo delle scelte in base ai valori degli attributi, fino a raggiungere una foglia che rappresenta la classe.

```
classifica(Oggetto, nc ,t(Att,Valori)) :-
      %writeln(Oggetto),
      member(Att=Val,Oggetto),
      member(Val:null,Valori).


classifica(Oggetto,Classe,t(Att,Valori)) :-
      %writeln(Oggetto), % Aggiungi questa riga per vedere l'oggetto
      member(Att=Val,Oggetto),
      member(Val:l(Classe),Valori).


classifica(Oggetto,Classe,t(Att,Valori)) :-
      member(Att=Val,Oggetto),
      delete(Oggetto,Att=Val,Resto),
      member(Val:t(AttFiglio,ValoriFiglio),Valori),
      classifica(Resto,Classe,t(AttFiglio,ValoriFiglio)).
```

Il predicato *stampa_matrice_di_confusione* effettua il test dell'albero sui dati di test (esempi non usati per costruire l'albero) e calcola l'accuratezza del modello. I risultati sono suddivisi tramite il predicato *valuta* in vari casi come:

- Vero obeso (VO)

- Vero sovrappeso (VSO)
- Vero normopeso (VN)
- Vero sottopeso (VST)
- Obeso come sovrappeso (OSO)
- Obeso come normopeso (ON)
- Obeso come sottopeso (OST)
- Sovrappeso come obeso (SOO)
- Sovrappeso come normopeso (SN)
- Sovrappeso come sottopeso (SOST)
- Normopeso come obeso (NO)
- Normopeso come sovrappeso (NSO)
- Normopeso come sottopeso (NST)
- Sottopeso come obeso (STO)
- Sottopeso come sovrappeso (STSO)
- Sottopeso come normopeso (STN)
- Non classificato (NC)

```
stampa_matrice_di_confusione :-
      carica_albero_da_file('/Users/niccolociotti/Desktop/IA/albero.
         txt', Albero, Algoritmo),
      alb(Albero),
      findall(Classe/Oggetto,s(Classe,Oggetto),TestSet),
      %write(TestSet),
      length(TestSet,N),
      valuta(Albero,TestSet,VO,0,VSO,0,VN,0,VST,0,OSO,0,ON,0,OST,0,
         SOO,0,SN,0,SOST,0,NO,0,NSO,0,NST,0,STO,0,STSO,0,STN,0,NC,0),

      A is (VO + VSO + VN + VST) / (N - NC), % Accuratezza
      E is 1 - A,          % Errore
      write('Test effettuati :'), writeln(N),

      write('Veri obesi: '), writeln(VO),
      write('Veri sovrappeso: '), writeln(VSO),
      write('Veri normopeso: '), writeln(VN),
      write('Veri sottopeso: '), writeln(VST),
      write('Obesi classificati come sovrappeso: '), writeln(OSO),
```

```
      write('Obesi classificati come normopeso: '), writeln(ON),
      write('Obesi classificati come sottopeso: '), writeln(OST),
      write('Sovrappeso classificati come obesi: '), writeln(SOO),
      write('Sovrappeso classificati come normopeso: '), writeln(SN),
      write('Sovrappeso classificati come sottopeso: '), writeln(SOST
         ),
      write('Normopeso classificati come obesi: '), writeln(NO),
      write('Normopeso classificati come sovrappeso: '), writeln(NSO)
         ,
      write('Normopeso classificati come sottopeso: '), writeln(NST),
      write('Sottopeso classificati come obesi: '), writeln(STO),
      write('Sottopeso classificati come sovrappeso: '), writeln(STSO
         ),
      write('Sottopeso classificati come normopeso: '), writeln(STN),

      write('Test non classificati :'), writeln(NC),

      write('Accuratezza: '), writeln(A),
      write('Errore: '), writeln(E),

      Valori = [VO, VSO, VN, VST, OSO, ON, OST, SOO, SN, SOST, NO,
         NSO, NST, STO, STSO, STN, Algoritmo],
   % Costruisci il comando da passare a Python
   atomics_to_string(Valori, ' ', Args),
   % Esegui il comando Python passando i valori come argomenti
   string_concat('python3 /Users/niccolociotti/Desktop/IA/matrix.py '
      , Args, Command),
   shell(Command).

scrivi_albero_su_file(FileName, Albero, Algoritmo) :-
   open(FileName, write, Stream),
   write(Stream, Albero),
      write(Stream, '.'),
      nl(Stream),
      write(Stream, '"'),
      write(Stream, Algoritmo),
      write(Stream, '".'),
   close(Stream).

carica_albero_da_file(FileName, Albero, Algoritmo) :-
      open(FileName, read, Stream),
      read(Stream, Albero),
      read(Stream, Algoritmo),
      close(Stream).
```

Il predicato *valuta* serve per valutare le performance del modello di classificazione, basato sull'albero di decisione indotto, confrontando le predizioni fatte dal modello con i veri valori nel Test Set. Si occupa di calcolare varie metriche di valutazione come la accuratezza, l'errore e la matrice di confusione.

```
valuta(_,[],VO,VO,VSO,VSO,VN,VN,VST,VST,OSO,OSO,ON,ON,OST,OST,SOO,SOO
    ,SN,SN,SOST,SOST,NO,NO,NSO,NSO,NST,NST,STO,STO,STSO,STSO,STN,STN,
    NC,NC).
% Corretta classificazione "obeso"
valuta(Albero, [obeso/Oggetto | Coda], VO,VOA,VSO,VSOA,VN,VNOA,VST,
    VSTA,OSO,OSOA,ON,ONA,OST,OSTA,SOO,SOOA,SN,SNA,SOST,SOSTA,NO,NOA,
    NSO,NSOA,NST,NSTA,STO,STOA,STSO,STSOA,STN,STNA,NC,NCA) :-
       classifica(Oggetto, obeso, Albero), !,
    VOA1 is VOA + 1,
       writeln('VOA1: '), writeln(VOA1),
    valuta(Albero, Coda,VO,VOA1,VSO,VSOA,VN,VNOA,VST,VSTA,OSO,OSOA,ON,
       ONA,OST,OSTA,SOO,SOOA,SN,SNA,SOST,SOSTA,NO,NOA,NSO,NSOA,NST,
       NSTA,STO,STOA,STSO,STSOA,STN,STNA,NC,NCA).


% Corretta classificazione "sovrappeso"
valuta(Albero, [sovrappeso/Oggetto | Coda], VO,VOA,VSO,VSOA,VN,VNOA,
    VST,VSTA,OSO,OSOA,ON,ONA,OST,OSTA,SOO,SOOA,SN,SNA,SOST,SOSTA,NO,
    NOA,NSO,NSOA,NST,NSTA,STO,STOA,STSO,STSOA,STN,STNA,NC,NCA) :-
       classifica(Oggetto, sovrappeso, Albero), !,
       VSOA1 is VSOA + 1,
       writeln('VSOA1: '), writeln(VSOA1),
       valuta(Albero, Coda,VO,VOA,VSO,VSOA1,VN,VNOA,VST,VSTA,OSO,OSOA,
          ON,ONA,OST,OSTA,SOO,SOOA,SN,SNA,SOST,SOSTA,NO,NOA,NSO,NSOA,
          NST,NSTA,STO,STOA,STSO,STSOA,STN,STNA,NC,NCA).

% Corretta classificazione "normopeso"
valuta(Albero, [normopeso/Oggetto | Coda], VO,VOA,VSO,VSOA,VN,VNOA,
    VST,VSTA,OSO,OSOA,ON,ONA,OST,OSTA,SOO,SOOA,SN,SNA,SOST,SOSTA,NO,
    NOA,NSO,NSOA,NST,NSTA,STO,STOA,STSO,STSOA,STN,STNA,NC,NCA) :-
       classifica(Oggetto, normopeso, Albero), !,
       VNOA1 is VNOA + 1,
       writeln('VNOA1: '), writeln(VNOA1),
       valuta(Albero, Coda,VO,VOA,VSO,VSOA,VN,VNOA1,VST,VSTA,OSO,OSOA,
          ON,ONA,OST,OSTA,SOO,SOOA,SN,SNA,SOST,SOSTA,NO,NOA,NSO,NSOA,
          NST,NSTA,STO,STOA,STSO,STSOA,STN,STNA,NC,NCA).

% Corretta classificazione "sottopeso"
valuta(Albero, [sottopeso/Oggetto | Coda], VO,VOA,VSO,VSOA,VN,VNOA,
    VST,VSTA,OSO,OSOA,ON,ONA,OST,OSTA,SOO,SOOA,SN,SNA,SOST,SOSTA,NO,
    NOA,NSO,NSOA,NST,NSTA,STO,STOA,STSO,STSOA,STN,STNA,NC,NCA) :-
       classifica(Oggetto, sottopeso, Albero), !,
       VSTA1 is VSTA + 1,
       writeln('VSTA1: '), writeln(VSTA1),
       valuta(Albero, Coda,VO,VOA,VSO,VSOA,VN,VNOA,VST,VSTA1,OSO,OSOA,
          ON,ONA,OST,OSTA,SOO,SOOA,SN,SNA,SOST,SOSTA,NO,NOA,NSO,NSOA,
          NST,NSTA,STO,STOA,STSO,STSOA,STN,STNA,NC,NCA).
```

```
% Obeso classificato come sovrappeso
valuta(Albero, [obeso/Oggetto | Coda], VO,VOA,VSO,VSOA,VN,VNOA,VST,
    VSTA,OSO,OSOA,ON,ONA,OST,OSTA,SOO,SOOA,SN,SNA,SOST,SOSTA,NO,NOA,
    NSO,NSOA,NST,NSTA,STO,STOA,STSO,STSOA,STN,STNA,NC,NCA) :-
       classifica(Oggetto, sovrappeso, Albero), !,
       OSOA1 is OSOA + 1,
       writeln('OSOA1: '), writeln(OSOA1),
       valuta(Albero, Coda,VO,VOA,VSO,VSOA,VN,VNOA,VST,VSTA,OSO,OSOA1,
          ON,ONA,OST,OSTA,SOO,SOOA,SN,SNA,SOST,SOSTA,NO,NOA,NSO,NSOA,
          NST,NSTA,STO,STOA,STSO,STSOA,STN,STNA,NC,NCA).

% Obeso classificato come normopeso
valuta(Albero, [obeso/Oggetto | Coda], VO,VOA,VSO,VSOA,VN,VNOA,VST,
    VSTA,OSO,OSOA,ON,ONA,OST,OSTA,SOO,SOOA,SN,SNA,SOST,SOSTA,NO,NOA,
    NSO,NSOA,NST,NSTA,STO,STOA,STSO,STSOA,STN,STNA,NC,NCA) :-
       classifica(Oggetto, normopeso, Albero), !,
       ONA1 is ONA + 1,
       writeln('ONA1: '), writeln(ONA1),
       valuta(Albero, Coda,VO,VOA,VSO,VSOA,VN,VNOA,VST,VSTA,OSO,OSOA,
          ON,ONA1,OST,OSTA,SOO,SOOA,SN,SNA,SOST,SOSTA,NO,NOA,NSO,NSOA,
          NST,NSTA,STO,STOA,STSO,STSOA,STN,STNA,NC,NCA).

% Obeso classificato come sottopeso
valuta(Albero, [obeso/Oggetto | Coda], VO,VOA,VSO,VSOA,VN,VNOA,VST,
    VSTA,OSO,OSOA,ON,ONA,OST,OSTA,SOO,SOOA,SN,SNA,SOST,SOSTA,NO,NOA,
    NSO,NSOA,NST,NSTA,STO,STOA,STSO,STSOA,STN,STNA,NC,NCA) :-
       classifica(Oggetto, sottopeso, Albero), !,
       OSTA1 is OSTA + 1,
       writeln('OSTA1: '), writeln(OSTA1),
       valuta(Albero, Coda,VO,VOA,VSO,VSOA,VN,VNOA,VST,VSTA,OSO,OSOA,
          ON,ONA,OST,OSTA1,SOO,SOOA,SN,SNA,SOST,SOSTA,NO,NOA,NSO,NSOA,
          NST,NSTA,STO,STOA,STSO,STSOA,STN,STNA,NC,NCA).

% Sovrappeso classificato come obeso
valuta(Albero, [sovrappeso/Oggetto | Coda], VO,VOA,VSO,VSOA,VN,VNOA,
    VST,VSTA,OSO,OSOA,ON,ONA,OST,OSTA,SOO,SOOA,SN,SNA,SOST,SOSTA,NO,
    NOA,NSO,NSOA,NST,NSTA,STO,STOA,STSO,STSOA,STN,STNA,NC,NCA) :-
       classifica(Oggetto, obeso, Albero), !,
       SOOA1 is SOOA + 1,
       writeln('SOOA1: '), writeln(SOOA1),
       valuta(Albero, Coda,VO,VOA,VSO,VSOA,VN,VNOA,VST,VSTA,OSO,OSOA,
          ON,ONA,OST,OSTA,SOO,SOOA1,SN,SNA,SOST,SOSTA,NO,NOA,NSO,NSOA,
          NST,NSTA,STO,STOA,STSO,STSOA,STN,STNA,NC,NCA).

% Sovrappeso classificato come normopeso
valuta(Albero, [sovrappeso/Oggetto | Coda], VO,VOA,VSO,VSOA,VN,VNOA,
    VST,VSTA,OSO,OSOA,ON,ONA,OST,OSTA,SOO,SOOA,SN,SNA,SOST,SOSTA,NO,
    NOA,NSO,NSOA,NST,NSTA,STO,STOA,STSO,STSOA,STN,STNA,NC,NCA) :-
       classifica(Oggetto, normopeso, Albero), !,
```

```
      SNA1 is SNA + 1,
      writeln('SNA1: '), writeln(SNA1),
      valuta(Albero, Coda,VO,VOA,VSO,VSOA,VN,VNOA,VST,VSTA,OSO,OSOA,
         ON,ONA,OST,OSTA,SOO,SOOA,SN,SNA1,SOST,SOSTA,NO,NOA,NSO,NSOA,
         NST,NSTA,STO,STOA,STSO,STSOA,STN,STNA,NC,NCA).

% Sovrappeso classificato come sottopeso
valuta(Albero, [sovrappeso/Oggetto | Coda], VO,VOA,VSO,VSOA,VN,VNOA,
   VST,VSTA,OSO,OSOA,ON,ONA,OST,OSTA,SOO,SOOA,SN,SNA,SOST,SOSTA,NO,
   NOA,NSO,NSOA,NST,NSTA,STO,STOA,STSO,STSOA,STN,STNA,NC,NCA) :-
      classifica(Oggetto, sottopeso, Albero), !,
      SOSTA1 is SOSTA + 1,
      writeln('SOSTA1: '), writeln(SOSTA1),
      valuta(Albero, Coda,VO,VOA,VSO,VSOA,VN,VNOA,VST,VSTA,OSO,OSOA,
         ON,ONA,OST,OSTA,SOO,SOOA,SN,SNA,SOST,SOSTA1,NO,NOA,NSO,NSOA,
         NST,NSTA,STO,STOA,STSO,STSOA,STN,STNA,NC,NCA).

% Normopeso classificato come obeso
valuta(Albero, [normopeso/Oggetto | Coda], VO,VOA,VSO,VSOA,VN,VNOA,
   VST,VSTA,OSO,OSOA,ON,ONA,OST,OSTA,SOO,SOOA,SN,SNA,SOST,SOSTA,NO,
   NOA,NSO,NSOA,NST,NSTA,STO,STOA,STSO,STSOA,STN,STNA,NC,NCA) :-
      classifica(Oggetto, obeso, Albero), !,
      NOA1 is NOA + 1,
      writeln('NOA1: '), writeln(NOA1),
      valuta(Albero, Coda,VO,VOA,VSO,VSOA,VN,VNOA,VST,VSTA,OSO,OSOA,
         ON,ONA,OST,OSTA,SOO,SOOA,SN,SNA,SOST,SOSTA,NO,NOA1,NSO,NSOA,
         NST,NSTA,STO,STOA,STSO,STSOA,STN,STNA,NC,NCA).

% Normopeso classificato come sovrappeso
valuta(Albero, [normopeso/Oggetto | Coda], VO,VOA,VSO,VSOA,VN,VNOA,
   VST,VSTA,OSO,OSOA,ON,ONA,OST,OSTA,SOO,SOOA,SN,SNA,SOST,SOSTA,NO,
   NOA,NSO,NSOA,NST,NSTA,STO,STOA,STSO,STSOA,STN,STNA,NC,NCA) :-
      classifica(Oggetto, sovrappeso, Albero), !,
      NSOA1 is NSOA + 1,
      writeln('NSOA1: '), writeln(NSOA1),
      valuta(Albero, Coda,VO,VOA,VSO,VSOA,VN,VNOA,VST,VSTA,OSO,OSOA,
         ON,ONA,OST,OSTA,SOO,SOOA,SN,SNA,SOST,SOSTA,NO,NOA,NSO,NSOA1,
         NST,NSTA,STO,STOA,STSO,STSOA,STN,STNA,NC,NCA).

% Normopeso classificato come sottopeso
valuta(Albero, [normopeso/Oggetto | Coda], VO,VOA,VSO,VSOA,VN,VNOA,
   VST,VSTA,OSO,OSOA,ON,ONA,OST,OSTA,SOO,SOOA,SN,SNA,SOST,SOSTA,NO,
   NOA,NSO,NSOA,NST,NSTA,STO,STOA,STSO,STSOA,STN,STNA,NC,NCA) :-
      classifica(Oggetto, sottopeso, Albero), !,
      NSTA1 is NSTA + 1,
      writeln('NSTA1: '), writeln(NSTA1),
      valuta(Albero, Coda,VO,VOA,VSO,VSOA,VN,VNOA,VST,VSTA,OSO,OSOA,
         ON,ONA,OST,OSTA,SOO,SOOA,SN,SNA,SOST,SOSTA,NO,NOA,NSO,NSOA,
         NST,NSTA1,STO,STOA,STSO,STSOA,STN,STNA,NC,NCA).
```

```
% Sottopeso classificato come obeso
valuta(Albero, [sottopeso/Oggetto | Coda], VO,VOA,VSO,VSOA,VN,VNOA,
    VST,VSTA,OSO,OSOA,ON,ONA,OST,OSTA,SOO,SOOA,SN,SNA,SOST,SOSTA,NO,
    NOA,NSO,NSOA,NST,NSTA,STO,STOA,STSO,STSOA,STN,STNA,NC,NCA) :-
       classifica(Oggetto, obeso , Albero), !,
       STOA1 is STOA + 1,
       writeln('STOA1: '), writeln(STOA1),
       valuta(Albero, Coda,VO,VOA,VSO,VSOA,VN,VNOA,VST,VSTA,OSO,OSOA,
          ON,ONA,OST,OSTA,SOO,SOOA,SN,SNA,SOST,SOSTA,NO,NOA,NSO,NSOA,
          NST,NSTA,STO,STOA1,STSO,STSOA,STN,STNA,NC,NCA).

% Sottopeso classificato come sovrappeso
valuta(Albero, [sottopeso/Oggetto | Coda], VO,VOA,VSO,VSOA,VN,VNOA,
    VST,VSTA,OSO,OSOA,ON,ONA,OST,OSTA,SOO,SOOA,SN,SNA,SOST,SOSTA,NO,
    NOA,NSO,NSOA,NST,NSTA,STO,STOA,STSO,STSOA,STN,STNA,NC,NCA) :-
       classifica(Oggetto, sovrappeso, Albero), !,
       STSOA1 is STSOA + 1,
       writeln('STSOA1: '), writeln(STSOA1),
       valuta(Albero, Coda,VO,VOA,VSO,VSOA,VN,VNOA,VST,VSTA,OSO,OSOA,
          ON,ONA,OST,OSTA,SOO,SOOA,SN,SNA,SOST,SOSTA,NO,NOA,NSO,NSOA,
          NST,NSTA,STO,STOA,STSO,STSOA1,STN,STNA,NC,NCA).

% Sottopeso classificato come normopeso
valuta(Albero, [sottopeso/Oggetto | Coda], VO,VOA,VSO,VSOA,VN,VNOA,
    VST,VSTA,OSO,OSOA,ON,ONA,OST,OSTA,SOO,SOOA,SN,SNA,SOST,SOSTA,NO,
    NOA,NSO,NSOA,NST,NSTA,STO,STOA,STSO,STSOA,STN,STNA,NC,NCA) :-
       classifica(Oggetto, normopeso, Albero), !,
       STNA1 is STNA + 1,
       writeln('STNA1: '), writeln(STNA1),
       valuta(Albero, Coda,VO,VOA,VSO,VSOA,VN,VNOA,VST,VSTA,OSO,OSOA,
          ON,ONA,OST,OSTA,SOO,SOOA,SN,SNA,SOST,SOSTA,NO,NOA,NSO,NSOA,
          NST,NSTA,STO,STOA,STSO,STSOA,STN,STNA1,NC,NCA).

% Oggetto non classificato
valuta(Albero, [_/Oggetto | Coda], VO,VOA,VSO,VSOA,VN,VNOA,VST,VSTA,
    OSO,OSOA,ON,ONA,OST,OSTA,SOO,SOOA,SN,SNA,SOST,SOSTA,NO,NOA,NSO,
    NSOA,NST,NSTA,STO,STOA,STSO,STSOA,STN,STNA,NC,NCA) :-
       classifica(Oggetto, _ , Albero), !,
       NCA1 is NCA + 1,
       writeln('NCA1: '), writeln(NCA1),
       valuta(Albero, Coda,VO,VOA,VSO,VSOA,VN,VNOA,VST,VSTA,OSO,OSOA,
          ON,ONA,OST,OSTA,SOO,SOOA,SN,SNA,SOST,SOSTA,NO,NOA,NSO,NSOA,
          NST,NSTA,STO,STOA,STSO,STSOA,STN,STNA,NC,NCA1).
```

2.1.5 Esempio di albero decisionale

```
swipl -s /Users/niccolociotti/Desktop/IA/
   Obesity_Tree_Induction_C4_5.pl

?- induce_albero(Albero,'C45').
```

Di seguito è mostrata una parte dell'albero decisionale generato

```
smoke
 no
   favc
    no
     height
       Average
         age
           Youth
            mtrans
              Automobile
                gender
                 Female ==> normopeso
                 Male
                   ncp
                     1 ==> sovrappeso
                     2 ==> ???
                     3
                       water
                         1 ==> ???
                         2
                           activity
                             0 ==> obeso
                             1 ==> obeso
                             2 ==> sovrappeso
                             3 ==> ???
                       3 ==> sovrappeso
                  4 ==> normopeso
              Bike ==> normopeso
              Motorbike ==> normopeso
              Public_Transportation
                gender
                  Female
                    alcohol
                      Always ==> ???
                      Frequently ==> ???
```

```
Sometimes
  water
    1
      ncp
        1 ==> sottopeso
        2 ==> ???
        3
          weight
            Heavy ==> ???
            Light
              activity
                0 ==> normopeso
                1 ==> sottopeso
                2 ==> sottopeso
                3 ==> ???
            Normal ==> normopeso
        4 ==> sottopeso
    2
      ncp
        1
          family
            no ==> normopeso
            yes
              activity
                0 ==> sovrappeso
                1 ==> ???
                2
                  weight
                    Heavy ==> ???
                    Light ==> normopeso
                    Normal ==> sovrappeso
                3 ==> ???
        2 ==> sovrappeso
        3
          activity
            0 ==> sovrappeso
            1 ==> sottopeso
```

```
?- stampa_matrice_di_confusione.

Test effettuati :633
Veri Obesi: 192
Veri Sovrappeso: 80
Veri Normopeso: 21
Veri Sottopeso: 52
Obesi classificati come Sovrappeso: 15
Obesi classificati come Normopeso: 6
Obesi classificati come Sottopeso: 1
Sovrappeso classificati come Obesi: 5
Sovrappeso classificati come Normopeso: 11
Sovrappeso classificati come Sottopeso: 1
Normopeso classificati come Obesi: 3
Normopeso classificati come Sovrappeso: 12
Normopeso classificati come Sottopeso: 7
Sottopeso classificati come Obesi: 0
Sottopeso classificati come Sovrappeso: 2
Sottopeso classificati come Normopeso: 10
Test non classificati :215
Accuratezza: 0.8253588516746412
Errore: 0.17464114832535882
Matrice di Confusione:
[[192 15 6  1]
 [ 5  80 11  1]
 [ 3  12 21  7]
 [ 0  2  10 52]]
Matrice di confusione salvata in: /Users/niccolociotti/
   confusion_matrix_C45.png
2025-01-23 12:46:46.144 Python[3276:168578] +[IMKClient
   subclass]: chose IMKClient_Modern
2025-01-23 12:46:54.191 Python[3276:168578] +[IMKInputSession
   subclass]: chose IMKInputSession_Modern
true.
```

La matrice di confusione viene mostrata a schermo con la possibilità di essere salvata.

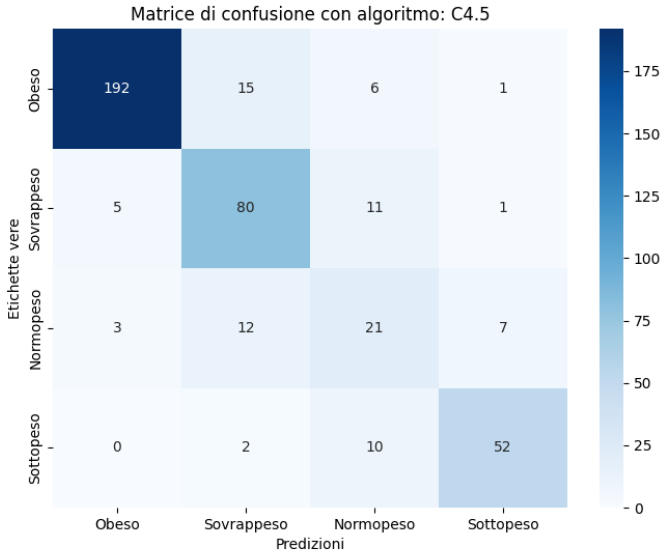

Figura 2.1: Matrice di confusione con algoritmo C4.5

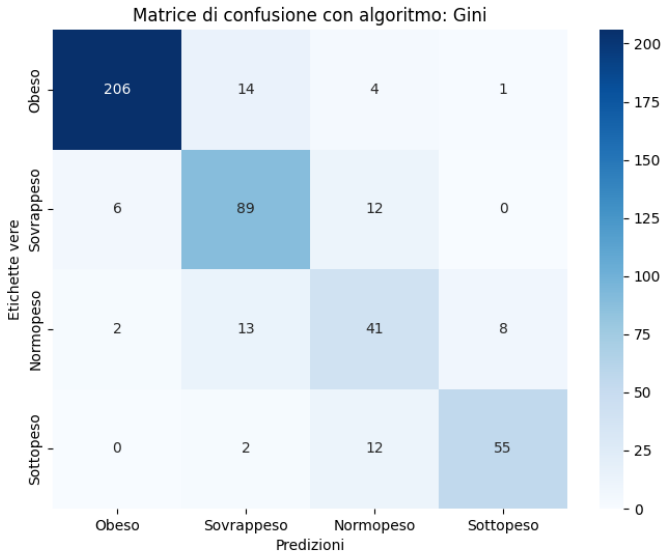

Figura 2.2: Matrice di confusione con Gini

2.1.6 Caso particolare

Per verificare il funzionamento del modello, è stato effettuato un esperimento manipolativo sui dati di input. In particolare, è stato selezionato un elemento dal Dataset di addestramento con la seguente configurazione:

```
e(sottopeso, [gender = 'Female', age = 'Youth', height = 'Average',
weight = 'Light', family = 'no', favc = 'no', ncp = 3, smoke = 'no',
water = 2, activity = 2, alcohol = 'Sometimes',
mtrans = 'Public_Transportation']).
```

Successivamente, è stata modificata l'etichetta associata all'elemento, invertendola da *sottopeso* a *obeso*, e aggiunto tale elemento al dataset di test:

```
s(obeso, [gender = 'Female', age = 'Youth', height = 'Average',
weight = 'Light', family = 'no', favc = 'no', ncp = 3, smoke = 'no',
water = 2, activity = 2, alcohol = 'Sometimes',
mtrans = 'Public_Transportation']).
```

Dopo aver eseguito il test e analizzato i risultati, è emerso che l'algoritmo ha classificato l'elemento manipolato come *sottopeso* invece di *obeso*. Questo risultato evidenzia un errore nella previsione, suggerendo che il modello abbia imparato correttamente a discriminare quella classe.

Conclusioni

Analisi

Il progetto ha raggiunto con successo l'obiettivo di sviluppare un sistema previsionale per la classificazione delle condizioni di peso utilizzando alberi decisionali implementati in Prolog. L'approccio ha dimostrato l'efficacia dell'utilizzo dei criteri Gini e C4.5, consentendo un'analisi comparativa tra i due algoritmi.

- *Accuratezza*: il criterio di Gini tende a offrire buoni livelli di accuratezza in problemi di classificazione binaria o quando si ha a disposizione un dataset relativamente semplice e bilanciato. Questo approccio si basa sull'indice di impurità di Gini, che cerca di massimizzare l'omogeneità dei dati all'interno dei nodi. Tuttavia, la mancanza di un meccanismo di normalizzazione può rappresentare un limite nei dataset complessi, caratterizzati da un elevato numero di attributi o valori variabili. In tali contesti, Gini può risultare meno accurato poiché tende a preferire attributi con numerosi valori univoci, rischiando di creare un modello sovradattato.

  Il criterio C4.5, invece, garantisce generalmente una maggiore accuratezza in problemi che coinvolgono molte classi o dataset più eterogenei. Questo risultato è reso possibile dall'uso del guadagno di informazione normalizzato, che tiene conto del numero di valori univoci per ogni attributo, riducendo così il rischio di sovradattamento. In questo modo, il criterio C4.5 consente di generare alberi decisionali più robusti, capaci di generalizzare meglio su nuovi dati. Tuttavia, nel caso in esame, l'accuratezza di C4.5 è risultata inferiore rispetto alle aspettative (0.8253 contro 0.8408 per Gini). Questo risultato potrebbe essere attribuito alla limitata variabilità e popolazione del dataset, che penalizza le caratteristiche distintive di C4.5, riducendo i vantaggi della normalizzazione.

- *Errore*: il criterio di Gini tende a presentare performance inferiori rispetto a C4.5 nei dataset complessi. Poiché Gini privilegia la costruzione di alberi rapidi e

semplici, potrebbe sacrificare la precisione nella selezione degli attributi, favorendo quelli con molti valori univoci. Questo comportamento aumenta il rischio di sovradattamento, portando a errori maggiori nella fase di test.

D'altro canto, il criterio C4.5 si dimostra più efficace nel ridurre l'errore complessivo, grazie alla sua capacità di selezionare gli attributi che realmente contribuiscono alla riduzione dell'entropia. Nel caso in esame, l'errore di Gini è stato misurato a 0.1591, inferiore rispetto all'errore di C4.5, pari a 0.1746. Questo risultato conferma che, a causa delle limitazioni del dataset, C4.5 non è stato in grado di mantenere una maggiore capacità discriminativa rispetto alla controparte, seppur teoricamente più efficente.

L'implementazione ha consentito inoltre, di preparare un Dataset discretizzato e trasformato in un formato compatibile con Prolog. Ha permesso di generare alberi decisionali ricorsivamente, partendo da un training set e testando i risultati su un test set ed infine valutare il modello mediante la matrice di confusione.

Confronto dei criteri e limiti

Per quanto riguarda il metodo di valutazione dei nodi:

- Criterio di Gini: Si concentra sull'omogeneità dei nodi calcolando l'impurità basata sulla probabilità degli elementi in ciascuna classe.
- Criterio C4.5: Si basa sul guadagno di informazione, che misura la riduzione dell'entropia nei dati dopo la divisione per un attributo.

Riguardo l'efficienza nella costruzione dell'albero:

- Criterio di Gini: richiede meno calcoli rispetto a C4.5, rendendolo più veloce in termini di performance computazionale, soprattutto su dataset di grandi dimensioni.
- Criterio C4.5: include la normalizzazione del guadagno informativo per gestire meglio gli attributi con valori multipli, aumentando la complessità del calcolo.

Per la capacità di gestione dei valori:

- Criterio di Gini: tende a preferire attributi con molti valori univoci (es., attributi numerici), il che può portare a sovradattamento del modello.
- Criterio C4.5: normalizza il guadagno di informazione in base al numero di valori univoci dell'attributo, riducendo il rischio di sovradattamento.

Per la qualità dell'albero generato:

- Criterio di Gini: produce alberi più semplici, con meno nodi e rami, ma potrebbe sacrificare la precisione in contesti con dati complessi.

- Criterio C4.5: crea alberi più accurati ma con una struttura più complessa, specialmente in presenza di molti attributi o valori

In sintesi, il progetto rappresenta un valido esempio di applicazione dell'intelligenza artificiale e degli alberi decisionali nella classificazione di dati complessi, dimostrando la robustezza e la versatilità di strumenti come Prolog.

Sviluppi futuri

Un miglioramento significativo nella costruzione degli alberi decisionali in Prolog potrebbe essere ottenuto attraverso l'introduzione di una strategia di *ricerca in ampiezza*. Questo approccio, in contrapposizione alla classica costruzione in profondità, consente di esplorare i nodi livello per livello, garantendo una distribuzione uniforme dell'elaborazione e facilitando l'individuazione di pattern su più livelli dell'albero.

Confronto con la Ricerca in Profondità

La costruzione degli alberi decisionali tradizionali si basa spesso su un'esplorazione in profondità, che valuta completamente un ramo prima di passare a quelli successivi. Sebbene questo approccio sia semplice ed efficace in molti contesti, può presentare delle criticità, tra cui:

- La tendenza a esplorare percorsi lunghi e complessi, con un alto costo computazionale per nodi che potrebbero non essere rilevanti.
- La difficoltà nell'integrare criteri di arresto o di pruning efficaci, poiché le decisioni vengono prese localmente senza una visione globale dell'albero.
- Possibile stack overflow se l'albero che si va a costruire risulta essere di grandi dimensioni.

Al contrario, la ricerca in ampiezza esplora i nodi su un livello alla volta, consentendo una costruzione più equilibrata e la possibilità di valutare l'intero albero in modo sistematico. Questo approccio offre alcuni vantaggi distintivi:

- Una maggiore flessibilità nella gestione della costruzione, con la possibilità di introdurre facilmente criteri di priorità o di arresto.
- Una migliore gestione della memoria, poiché i nodi esplorati possono essere trattati come entità separate, riducendo il rischio di duplicazioni o inefficienze.

Pseudo algoritmo

- *Inizializzazione della struttura dati per la coda:* Creare una coda (FIFO - First In, First Out) per gestire i nodi da esplorare. La coda viene inizialmente popolata con il nodo radice, che rappresenta l'intero dataset e i possibili attributi da considerare.

- *Iterazione principale sull'albero:* Mentre la coda non è vuota, eseguire i seguenti passaggi per ogni nodo:
  - *Estrazione del nodo corrente:* Prelevare il primo elemento dalla testa della coda. Questo nodo rappresenta il sottoinsieme dei dati corrente e gli attributi rimanenti da considerare.
  - *Valutazione dell'attributo migliore:* Utilizzare una metrica di selezione per scegliere l'attributo migliore su cui suddividere i dati del nodo corrente.
  - *Suddivisione dei dati:* Suddividere il sottoinsieme dei dati del nodo corrente in base ai valori dell'attributo scelto, creando i nodi figli. Ogni nodo figlio rappresenta un sottoinsieme specifico dei dati.
  - *Aggiunta dei nodi figli alla coda:* Aggiungere i nodi figli alla fine della coda, assicurando che vengano esplorati dopo i nodi dello stesso livello.
- *Creazione dei nodi foglia:* Per ogni nodo esplorato, verificare se può diventare un nodo foglia. Le condizioni includono:
  - Tutti i dati nel nodo appartengono alla stessa classe.
  - Non ci sono più attributi disponibili per la suddivisione.
  - È stato raggiunto un criterio di arresto predefinito (ad esempio, profondità massima o numero minimo di istanze per nodo).

  Se il nodo soddisfa una di queste condizioni, viene marcato come foglia e associato alla classe più rappresentata o a una previsione probabilistica.
- *Ripetizione del processo fino a esaurimento della coda:* Continuare il processo fino a quando la coda non è vuota, garantendo che tutti i nodi siano stati esplorati e i loro figli valutati.
- *Costruzione finale dell'albero:* Dopo che la coda è stata svuotata, l'albero è completo. Ogni nodo avrà figli appropriati (o sarà un nodo foglia), e l'intera struttura rifletterà le suddivisioni effettuate livello per livello.
- *Valutazione e pruning opzionali:* Una volta costruito l'albero, è possibile applicare tecniche di pruning per ridurre l'overfitting. Questo può includere:
  - Potatura dei nodi che non migliorano significativamente le prestazioni.
  - Riduzione dei rami basata su validazione incrociata.

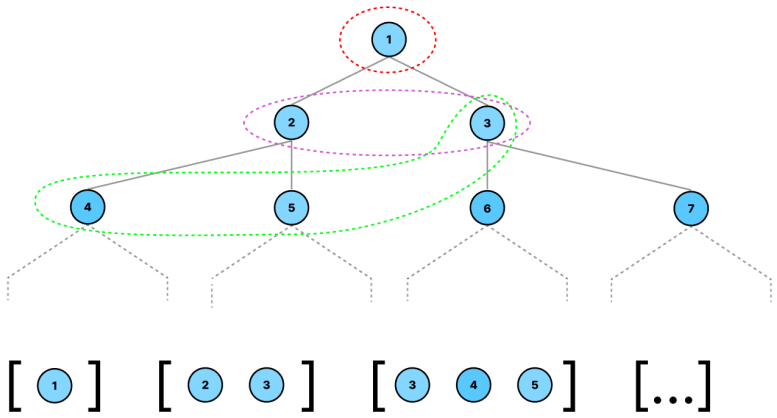

Figura 2.3: Albero in ampiezza.

Analisi Critica e Potenziale Applicativo

L'approccio in ampiezza può essere particolarmente utile in contesti in cui la profondità dell'albero rappresenta un fattore critico. Ad esempio, in un Dataset con una forte interdipendenza tra attributi su livelli diversi, la costruzione basata sulla ricerca in ampiezza consente di catturare rapidamente le relazioni senza compromettere l'integrità dei nodi più superficiali.

Tuttavia, non è priva di limitazioni. L'algoritmo richiede molto più tempo d'esecuzione e di risorse computazionali per creare l'albero, inficiando sull'usabilità dell'algoritmo.

Conclusioni e sviluppi

Nonostante le sfide, l'introduzione della ricerca in ampiezza offre un potenziale innovativo nella costruzione degli alberi decisionali. Essa apre la strada all'aggiunta futura di pruning, sfruttando la conoscenza globale dei livelli per eliminare interi rami in modo più efficace. Sebbene non sia sempre la scelta più adatta, il suo utilizzo in contesti specifici può fornire significativi vantaggi, ma nonostante le prove effettuate non siamo stati in grado di costruire un algoritmo funzionante in prolog che implementasse quanto descritto.