SISTEMA DI DIAGNOSTICA DI UNA MALATTIA

Gruppo di lavoro

Nome Cognome	Matricola	Email
Leonardo Colucci	758298	l.colucci23@studenti.uniba.it
Gianfranco De Vincenzo	758318	g.devincenzo14@studenti.uniba.it
Andrea Chiariello	717771	a.chiariello9@studenti.uniba.it

Link GitHub

Sistema di diagnostica

Il sistema del progetto mira a diagnosticare la presunta malattia del "**virus di fantasia**" basandosi sui dati forniti dall'utente. L'utente è coinvolto nel processo rispondendo a una serie di domande mirate, finalizzate a identificare i sintomi che l'utente potrebbe manifestare e le possibili correlazioni con il presunto virus. Questo processo serve a stabilire la probabilità che l'utente sia affetto dal patogeno.

La procedura diagnostica adottata dal sistema segue una logica di backward chaining. In un linguaggio logico come Prolog, ciò significherebbe interrogare lo stato di malattia e, notando atomi con valori sconosciuti, porre automaticamente domande relative a tali atomi definiti come 'askable'. Tuttavia, a causa delle limitazioni della libreria utilizzata, il sistema è costretto a operare con una modalità di forward chaining. In questa modalità, vengono poste domande iniziali, e ciascuna domanda controlla le sue condizioni per essere formulata, per poi determinare se è presente o meno una forma di malattia nell'individuo.

L'approccio generale si basa su una regola di derivazione, che è una forma generalizzata della regola di inferenza nota come modus ponens:
Se "$h \leftarrow a1 \wedge \ldots \wedge am$" è una clausola definita nella base di conoscenza e ogni ai è stato derivato, allora h può essere derivato.
Dove:

- h è la testa dell'atomo,
- $a1 \wedge \ldots \wedge am$ è il corpo della clausola, formato da ai atomi

Se $m > 0$, la clausola è detta regola, se $m = 0$, il corpo è vuoto e la clausola è detta clausola atomica o fatto, e tutte le clausole atomiche nella base di conoscenza sono sempre derivate in maniera diretta. Nel caso del sistema di diagnostica del virus la base di conoscenza è stata organizzata dalle seguenti clausole:

$$sintomi\_base \Leftarrow perdita\_di\_peso$$
$$sintomi\_base \Leftarrow diarrea$$
$$vomito \Leftarrow nausea$$
$$cisti \Leftarrow esame\_positivo$$
$$cisti \Leftarrow no\_esame \wedge dolore\_addominale \wedge rigonfiamento$$
$$ulcera \Leftarrow dolore\_addominale \wedge acidità\_di\_stomaco \wedge nausea$$
$$malattia\ lieve \Leftarrow sintomi\_base \wedge cisti \wedge vomito$$
$$malattia\ grave \Leftarrow sintomi\_base \wedge ulcera$$

La malattia presenta due forme distintive: una forma lieve e una forma grave. Entrambe condividono sintomi di base, tra cui perdita di peso improvvisa e episodi di diarrea. La forma lieve è caratterizzata da sintomi aggiuntivi come la presenza di cisti e episodi di vomito, mentre la forma grave è contrassegnata dalla presenza di ulcere.

La presenza di cisti può essere confermata in modo certo se l'utente ha sostenuto un esame medico con risultato positivo. In caso contrario, il sistema prende in considerazione sintomi correlati, come rigonfiamento e dolore addominale, per valutare la probabilità della presenza di cisti. Analogamente, la presenza di ulcere viene identificata in base alle risposte dell'utente riguardo a sintomi come acidità di stomaco, nausea e dolore addominale.

Sistema esperto/Sommario

Un sistema esperto è un'applicazione di intelligenza artificiale che si impegna nella risoluzione di problemi, cercando di emulare i comportamenti di esperti in un determinato campo di attività. La sua struttura fondamentale include una knowledge base che rappresenta e conserva informazioni e regole riguardanti il dominio in questione. L'inference engine è responsabile dell'applicazione pratica delle nozioni acquisite dalla base di conoscenza, mentre l'interfaccia utente facilita l'interazione tra il sistema e l'utente.

Implementazione del sistema esperto

Abbiamo sviluppato un sistema esperto in Python utilizzando la libreria Experta. Questo sistema si basa su regole che associano fatti accaduti a condizioni (LHS) e azioni (RHS). Ad esempio, per ogni sintomo, abbiamo definito una regola che chiede all'utente se riscontra quel sintomo e aggiorna il fatto corrispondente di conseguenza.

Il sistema analizza i sintomi dell'utente e, in base alle risposte, attiva altre regole correlate ai sintomi. A seconda dei sintomi attivati, vengono considerate regole aggiuntive relative a condizioni più complesse come ulcere, cisti e vomito. Alla fine, il sistema applica una regola finale che comunica la diagnosi basata sui sintomi dell'utente.

Inizialmente, il sistema chiede all'utente dei sintomi di base. Se l'utente presenta almeno uno dei sintomi basilari, come perdita di peso o diarrea, il sistema procede nell'analisi per determinare la forma della malattia. Se l'utente mostra sintomi della forma lieve del virus, come nausea e eventualmente vomito, il sistema cerca di confermare la presenza di cisti.

Se l'utente mostra tutti i sintomi della forma lieve, riceverà una diagnosi positiva. Se manca un sintomo tra cisti e vomito, il sistema esamina la possibilità che l'utente sia affetto dal secondo stadio della malattia. In questo caso, il sistema cerca di determinare la presenza di ulcere, considerando sintomi come dolore addominale, acidità di stomaco e nausea. Se l'utente presenta tutti e tre questi sintomi, il sistema conclude che l'utente ha contratto la forma grave del virus.

Esempi tratti dal codice

```
# MALATTIA LIEVE
@Rule(AND(Fact(sintomi_base=True), Fact(vomito=True), Fact(cisti=True)))
def malattia_lieve(self):
    print("I sintomi indicano che potresti aver contratto il virus.")
    print("È consigliabile recarsi da un medico da un medico per ulteriori accertamenti.")
    self.reset()
```

Questa regola è responsabile di comunicare la diagnosi all'utente quando il sistema ha determinato che l'utente ha contratto la forma lieve della malattia. Nella parte sinistra della regola (LHS), le condizioni necessarie per l'applicazione della regola sono che i fatti relativi ai sintomi di base, al vomito e alle cisti siano tutti impostati a True. Nella parte destra della regola (RHS), il sistema si occupa di comunicare la diagnosi all'utente attraverso stampe o messaggi appropriati.

```
# NAUSEA
@Rule(Fact(sintomi_base=True))
def ask_nausea(self):
    self.declare(Fact(nausea=ask_question("Avverti un senso di nausea?")))
```

Questa regola viene attivata quando i fatti nella parte sinistra (LHS) sono valutati come True. Una volta attivata, il sistema porrà una domanda all'utente e, in base alla risposta, imposterà il fatto relativo al sintomo come True o False.

```
# VOMITO
@Rule(Fact(nausea=True))
def ask_vomito(self):
    self.declare(Fact(vomito=ask_question("Hai avuto attacchi di vomito ultimamente?")))
```

Se il sintomo "nausea" è stato identificato come True nella regola precedente, questa regola successiva viene attivata per determinare se l'utente sta vivendo attacchi di vomito.

Rete Bayesiana

Abbiamo adottato un approccio basato su reti bayesiane, che sono grafi aciclici direzionati in cui ogni caratteristica è rappresentata come un nodo e le dipendenze tra le caratteristiche sono indicate tramite archi direzionati. Questo modello ci consente di stabilire che l'attivazione di una caratteristica dipende dall'attivazione di una o più altre caratteristiche, che chiamiamo genitori. Per fare ciò, stabiliamo un ordine tra le caratteristiche.

Le reti bayesiane sfruttano le probabilità e il teorema di Bayes per calcolare la probabilità di un evento specifico in base ad altri eventi (probabilità condizionata). In una rete bayesiana, ogni nodo con genitori ha una probabilità condizionata $P(node|parents(node))$ dove $parents(node)$ restituisce i nodi genitore del nodo. Di conseguenza, ogni nodo ha una tabella delle probabilità condizionate rispetto ai suoi genitori. Utilizziamo la formula generale per calcolare le probabilità di ogni nodo.

$$P(X_1, X_2, \ldots, X_n) = \prod_{i=1}^{n} P(X_1|X_1, \ldots, X_{n-1})$$

dove X_i sono le feature/nodi della rete bayesiana.

Ad esempio , abbiamo nodi come "Malattia", "Perdita di peso" e "Cisti", con archi che partono da "Malattia" e arrivano agli ultimi due nodi. La "Ciste" dipende anche dalla presenza della "Malattia", ma non c'è un collegamento diretto tra "Ciste" e "Perdita di peso", quindi queste due feauture sono indipendenti.

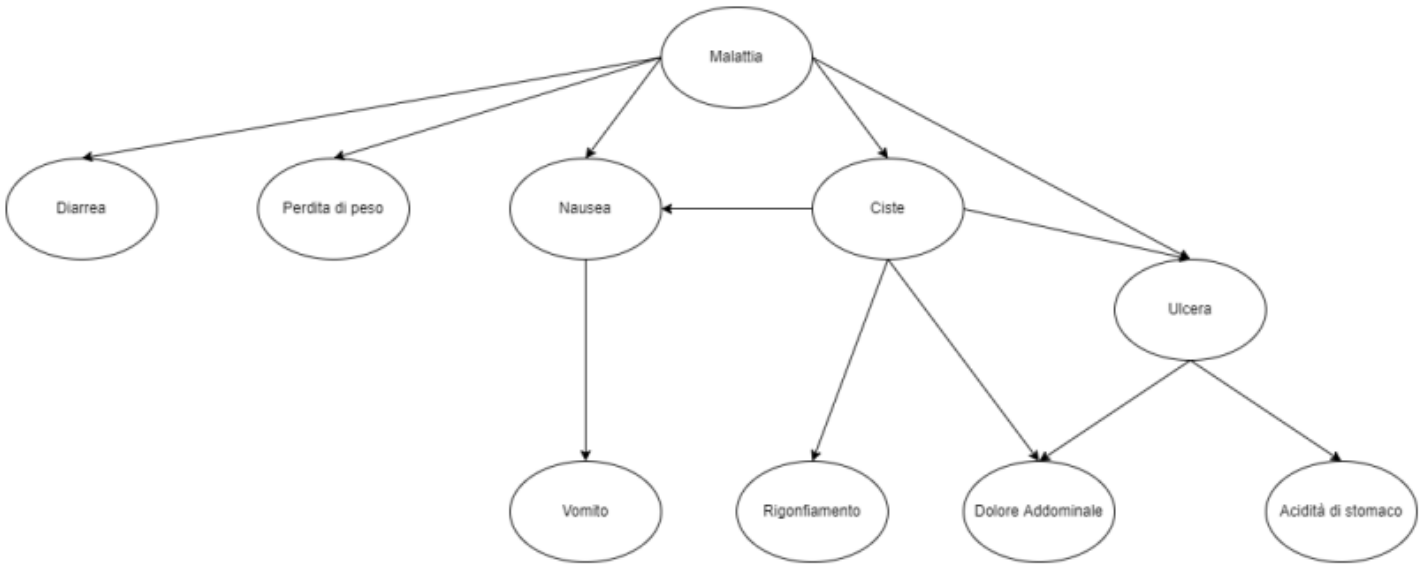
Abbiamo la flessibilità di espandere il modello includendo fattori di rischio per la malattia, come una dieta squilibrata o un ambiente malsano, per indicare possibili cause. Questo ci permette di migliorare l'accuratezza e l'efficacia del modello diagnostico.

Dopo aver costruito la rete bayesiana con la sua struttura e le tabelle per la probabilità condizionata, possiamo inferire la probabilità di eventi specifici osservando altri eventi. L'inferenza può essere esatta o approssimata. Nell'inferenza esatta, enumeriamo i mondi coerenti con le osservazioni e utilizziamo algoritmi come l'eliminazione di variabili per calcolare la probabilità esatta dell'evento. Nell'inferenza approssimata, stimiamo la probabilità di un evento.

Se non conosciamo a priori le tabelle delle probabilità condizionate degli eventi, possiamo derivarle analizzando i dati e utilizzando algoritmi come stimatori della massima verosimiglianza o stimatori di Bayes. Questi ultimi combinano la credenza precedente con i dati osservati per ottenere probabilità a posteriori. Possiamo anche utilizzare metodi come le regressioni o gli alberi decisionali per stimare le probabilità condizionate dei nodi in una DAG.

Implementazione

Sfruttando il fatto che stiamo lavorando con una malattia di fantasia abbiamo la libertà di poter sperimentare con una rete bayesiana con una struttura e delle tabelle date. Per prima cosa abbiamo la struttura della rete bayesiana così descritta:



Il grafo mostra chiaramente quali sono i sintomi della malattia, qual è la causa dei sintomi e anche delle variabili intermedie che a loro volta causano dei sintomi. Per implementare la rete bayesiana in Python abbiamo fatto uso di bnlearn, una libreria che fa da wrapper ad un’altra libreria pgmpy che è il vero cuore pulsante.

bnlearn ci permette di creare una DAG, creare ed assegnare delle tabelle di probabilità condizionata per singolo nodo (con la classe TabularCPD) e poi di inferire le probabilità per un nodo della DAG andando a segnalare le osservazioni (ossia quali sintomi l’utente riporta al sistema), tramite il metodo della eliminazione delle variabili. Inoltre, bnlearn permette di imparare anche le tabelle delle probabilità

andando a stimare da un dataset che gli forniamo in input. In questo caso ci permette di scegliere due opzioni: stimatore di massima verosimiglianza e stimatore di Bayes BDeu (ossia la credenza precedente è costruita usando una distribuzione uniforme di Dirichlet e andando a confrontarla con i dati osservati normalizzati con dei pseudoconteggi). Infine, ci permette di apprendere anche la struttura usando diversi metodi di ricerca (come la ricerca esaustiva, la ricerca Hillclimb, Chow-liu, etc.) che invece noi non useremo.

La prima cosa che dobbiamo fare (avendo una struttura data) è la creazione della DAG, per far ciò andiamo a creare un vettore di coppie, dove ogni coppia è formata da due nodi del grafo che vanno ad indicare un arco che li collega. Come secondo punto (se stiamo lavorando con la distribuzione di probabilità data) andiamo a creare e ad assegnare le tabelle delle probabilità: ogni tabella e associata ad un nodo che andiamo ad esplicitare; andiamo anche ad esplicitare le evidenze per quel nodo (se il nodo ha genitori), e poi a dichiarare i possibili stati del nodo (nel nostro caso sono 2: presente, non presente) e infine a inserire le tabelle delle probabilità vere e proprie.

La tabella ha come righe gli stati del nodo preso in esame e come colonne le combinazioni degli stati dei nodi genitori; così andiamo ad assegnare per ogni cella della tabella la probabilità che il nodo si trovi in un determinato stato dato la combinazione degli stati dei genitori. La somma degli elementi di una colonna deve essere uguale ad 1. Una volta creata una tabella per ogni nodo le andiamo ad assegnare alla DAG.

Arrivati a questo punto avremo una rete bayesiana funzionante, quello che ci rimane da fare è prendere le osservazioni del paziente (i sintomi che dichiara di avere) ed inferire la probabilità che il paziente abbia la malattia conoscendo lo stato dei sintomi. Per far ciò chiediamo (attraverso delle domande) i sintomi che il paziente ha; ad ogni sintomo osservato gli assegniamo il valore 0 se non è presente e 1 se invece è presente. Dopodiché andiamo ad usare la funzione di inferenza di bnlearn che va ad usare l'eliminazione di variabili per ottenere la probabilità della malattia, che andiamo poi a mostrare all'utente insieme ad un avvertimento se la malattia ha una probabilità maggiore o uguale al 50%. Altrimenti come alternativa, se non abbiamo le tabelle di probabilità per ogni nodo, possiamo apprendere le probabilità usando diversi stimatori (descritti precedentemente). Semplicemente diamo una DAG in input combinata con un dataset: ogni riga del dataset rappresenta un paziente osservato e testato per la malattia a cui associamo i vari sintomi che possiede. Una volta passati questi input decidiamo il tipo di stimatore da usare (nel nostro programma lasciamo la libertà all'utente) e così otteniamo nuovamente una rete bayesiana funzionante ma questa volta le probabilità sono apprese e non date. Questa è sicuramente una situazione più realistica del primo caso, in quanto è impossibile sapere a priori quali sono le probabilità che una malattia abbia un certo sintomo. Nella realtà ciò ci è sconosciuto e tutto quello che abbiamo, non sono altro che osservazioni parziali della realtà.

Valutazione

Potremmo interrogarci sulle differenze tra i due metodi di stima delle probabilità e chiederci se uno dei due sia superiore. Nella realtà, raramente ci troviamo di fronte a malattie con probabilità note; più spesso dobbiamo interpretare osservazioni parziali e conoscere i pro e i contro dei due metodi di stima delle probabilità è cruciale per medici e pazienti che potrebbero utilizzare strumenti simili.

Per rispondere a questa domanda, abbiamo eseguito numerose simulazioni sui dati a nostra disposizione, dividendoli in due set: uno per l'addestramento del modello e l'altro per il test. Abbiamo notato che lo stimatore di verosimiglianza sembra generalmente più accurato e preciso rispetto a quello di Bayes, anche se quest'ultimo ha un richiamo maggiore. In altre parole, lo stimatore di verosimiglianza è più affidabile quando identifica un positivo, mentre quello di Bayes è più propenso a individuare positivi, a costo di aumentare i falsi positivi.

Verosomiglianza (cambio database test)				
Numero	Accuratezza	Precisione	Richiamo	F1
1	0,9977	0,7273	0,4848	0,5818
2	0,9967	0,6818	0,3659	0,4762
3	0,9974	0,6923	0,5000	0,5806

Bayes (cambio database test)				
Numero	Accuratezza	Precisione	Richiamo	F1
1	0,9969	0,5312	0,5152	0,5231
2	0,9967	0,6429	0,4390	0,5217
3	0,9969	0,5758	0,5278	0,5508

Tuttavia, il punteggio F1, che combina precisione e richiamo, è migliore in due casi su tre per lo stimatore di verosimiglianza. È importante notare che entrambi gli stimatori hanno vantaggi e svantaggi, e la scelta dipende dal contesto specifico. Ad esempio, in caso di malattie gravi come il cancro, potrebbe essere preferibile un maggior richiamo anche a costo di più falsi positivi. Al contrario, in situazioni in cui una diagnosi errata è costosa, la precisione potrebbe essere prioritaria.

Verosomiglianza: 1		Predizione		Somma
		Malato	Non Malato	10000
Realtà	Malato	16	17	33
	Non Malato	6	9961	9967

Bayes: 1		Predizione		Somma
		Malato	Non Malato	10000
Realtà	Malato	17	16	33
	Non Malato	15	9952	9967

Verosomiglianza: 2		Predizione		Somma
		Malato	Non Malato	10000
Realtà	Malato	15	26	41
	Non Malato	7	9952	9959

Bayes: 2		Predizione		Somma
		Malato	Non Malato	10000
Realtà	Malato	18	23	41
	Non Malato	10	9949	9959

Verosomiglianza: 3		Predizione		Somma
		Malato	Non Malato	10000
Realtà	Malato	18	18	36
	Non Malato	8	9956	9964

Bayes: 3		Predizione		Somma
		Malato	Non Malato	10000
Realtà	Malato	19	17	36
	Non Malato	14	9950	9964

L'overfitting è un problema da considerare, specialmente quando i dati di addestramento non rappresentano accuratamente la realtà. In tali casi, lo stimatore di Bayes potrebbe essere preferibile poiché si adatta meno ai dati di addestramento.

Come secondo caso di test invece andiamo a testare qual è la quantità ottimale di dati da fornire al sistema nella fase di training, in questa fase del test andiamo quindi a variare la dimensione del dataset di training ma manteniamo costante il dataset di test. Cercheremo, quindi, di capire qual è la quantità sufficiente di dati da fornire al modello per ottenere dei risultati accettabili (questo test è molto utile soprattutto quando non si hanno molti dati su cui basare l'addestramento). Quindi eseguiamo 5 test con dataset di training di dimensione uguale a: 1.000, 5.000, 10.000, 25.000, 50.000. Mandando in esecuzione i 5 test per modello, otteniamo i seguenti dati:

Verosomiglianza (cambio database training)				
Dimensione training set	Accuratezza	Precisione	Richiamo	F1
1.000	0,9973	0,6667	0,3636	0,4706
5.000	0,9975	0,7000	0,4242	0,5283
10.000	0,9977	0,7273	0,4848	0,5818
25.000	0,9977	0,7273	0,4848	0,5818
50.000	0,9977	0,7273	0,4848	0,5818

Bayes (cambio database training)				
Dimensione training set	Accuratezza	Precisione	Richiamo	F1
1.000	0,9968	0,5152	0,5152	0,5152
5.000	0,9969	0,5312	0,5152	0,5231
10.000	0,9969	0,5312	0,5152	0,5231
25.000	0,9969	0,5312	0,5152	0,5231
50.000	0,9972	0,5862	0,5152	0,5484

La dimensione limitata dei test è un altro problema, limitando la significatività statistica dei risultati. Inoltre, abbiamo esaminato la quantità ottimale di dati per l'addestramento, notando che 10.000 elementi sembrano sufficienti per lo stimatore di verosimiglianza, mentre per quello di Bayes, l'aumento dei dati potrebbe migliorare la precisione senza influenzare il richiamo in modo significativo.

In sintesi, non esiste uno stimatore migliore in assoluto; la scelta dipende dal contesto e dalla disponibilità dei dati. In basso sono riportare le matrici di confusione:

Verosomiglianza: 1000		Predizione		Somma
		Malato	Non Malato	10000
Realtà	Malato	12	21	33
	Non Malato	6	9961	9967

Bayes: 1000		Predizione		Somma
		Malato	Non Malato	10000
Realtà	Malato	17	16	33
	Non Malato	16	9951	9967

Verosomiglianza: 5000		Predizione		Somma
		Malato	Non Malato	10000
Realtà	Malato	14	19	33
	Non Malato	6	9961	9967

Bayes: 5000		Predizione		Somma
		Malato	Non Malato	10000
Realtà	Malato	17	16	33
	Non Malato	15	9952	9967

Verosomiglianza: 10000		Predizione		Somma
		Malato	Non Malato	10000
Realtà	Malato	16	17	33
	Non Malato	6	9961	9967

Bayes: 10000		Predizione		Somma
		Malato	Non Malato	10000
Realtà	Malato	17	16	33
	Non Malato	15	9952	9967

Verosomiglianza: 25000		Predizione		Somma
		Malato	Non Malato	10000
Realtà	Malato	16	17	33
	Non Malato	6	9961	9967

Bayes: 25000		Predizione		Somma
		Malato	Non Malato	10000
Realtà	Malato	17	16	33
	Non Malato	15	9952	9967

Verosomiglianza: 50000		Predizione		Somma
		Malato	Non Malato	10000
Realtà	Malato	16	17	33
	Non Malato	6	9961	9967

Bayes: 50000		Predizione		Somma
		Malato	Non Malato	10000
Realtà	Malato	17	16	33
	Non Malato	12	9955	9967

Conclusione

Per future estensioni del progetto, sarebbe interessante esplorare ulteriori metodi di apprendimento automatico per migliorare la precisione e la generalizzazione del sistema. Inoltre, potrebbe essere utile considerare una maggiore varietà di dati di test e valutare l'impatto delle dimensioni del dataset di training sulla performance del sistema. Infine, potrebbero essere esplorate altre tecniche di inferenza e apprendimento per ottimizzare ulteriormente la diagnosi del virus di fantasia.