

Documentazione progetto ICON

Ivan De Cosmis
716486
i.decosmis@studenti.uniba.it

Data Inizio Stesura 18 Agosto, 2022
Data Fine Stesura 25 Agosto, 2022

Capitoli

1 Descrizione generale	2
1.1 Introduzione al problema	2
1.2 Approccio utilizzato	2
2 Sistema Esperto	3
2.1 Realizzazione del sistema esperto	3
3 Ontologie	7
3.1 Descrizione generale	7
3.2 Il tool Protege	7
3.3 La libreria Owlready2	8
4 Reti Bayesiane	9
4.1 Descrizione generale	9
4.2 Descrizione delle reti bayesiane implementate	10
4.3 Implementazione della rete bayesiana	11
4.4 Differenze tra tabella CPD date e tabella CPD apprese	11
5 Articoli utilizzati	14
6 Conclusioni	15

1 Descrizione generale

1.1 Introduzione al problema

A volte può essere fastidioso affrontare la giornata scegliendo un abbigliamento non adeguato alla situazione meteorologica. Con questo sistema punto a consigliare all'utente un abbigliamento adatto sia alle condizioni climatiche esterne sia ai gusti personali. Inoltre notifico anche in caso di situazioni meteorologiche molto avverse e suggerisco indumenti extra o informazioni per affrontare al meglio le prima citate situazioni avverse.

Essendo questo un progetto particolare non sono riuscito a trovare dataset online da sfruttare, per questo sono stati realizzati da me cercando informazioni da articoli e statistiche riguardanti questo problema(a fine documentazione), e usando anche un minimo di logica: temperatura molto bassa + vento molto forte = probabile malanno se non ci si copre in modo appropriato.

1.2 Approccio utilizzato

L'abbigliamento viene consigliato basandosi sulle informazioni inserite dall'utente. Queste informazioni sono ottenute facendo delle domande all'utente in modo da identificare la situazione meteorologica, la tipologia di giornata e lo stile desiderato. Il sistema sfrutta il concetto di **Forward Chaining** in un linguaggio logico (ES: Prolog). Esso chiede delle domande in base alla condizione in cui il sistema si trova, e alla fine decide cosa consigliare all'utente e lo avvisa nel caso di allerta meteo. Per decidere se le condizioni meteo sono pericolose, il sistema sfrutta semplici **Regole di derivazione**. Una regola di derivazione è una forma generale della regola di inferenza **Modus Ponendo Ponens**. Questa regola afferma che:

Se p è una clausola specificata nella base di conoscenza e ogni sua condizione è stata derivata, allora p può essere derivato

In notazione logica:

$$p \longleftarrow c_1 \wedge c_2 \dots \wedge c_n$$

Si chiama **Regola** se $n > 0$.

Si chiama **Fatto** se $n = 0$.

La base di conoscenza del sistema che riconosce un'allerta meteo è basata sulle seguenti **Clausole**:

$$\begin{aligned} \text{allerta meteo} &\longleftarrow \text{temperatura molto bassa} \\ \text{allerta meteo} &\longleftarrow \text{vento forte} \wedge \text{temperatura bassa} \\ \text{allerta meteo} &\longleftarrow \text{temperatura molto alta} \end{aligned}$$

L'allerta meteo viene quindi influenzata sia dalla temperatura, nel caso sia troppo bassa o troppo alta, e anche dal vento nel caso in cui sia già bassa.

2 Sistema Esperto

Tutto il progetto si basa sul sistema esperto.

Un sistema esperto è un programma che cerca di riprodurre le prestazioni di una o più persone esperte in un determinato campo di attività, ed è un'applicazione o una branca dell'intelligenza artificiale.

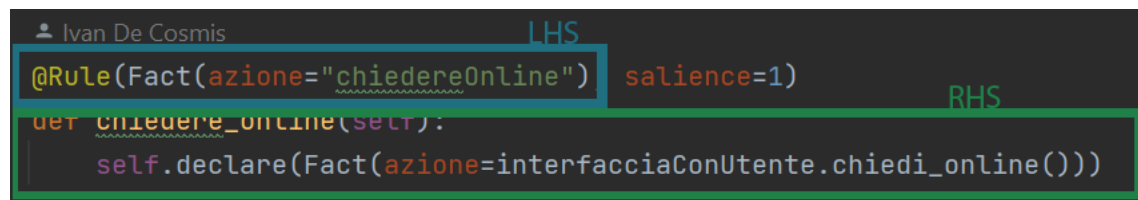
2.1 Realizzazione del sistema esperto

Ho associato dei fatti a delle regole usando una libreria del linguaggio **Python** chiamata **Experta**.

Queste regole sono basate su due concetti: **LHS** e **RHS**:

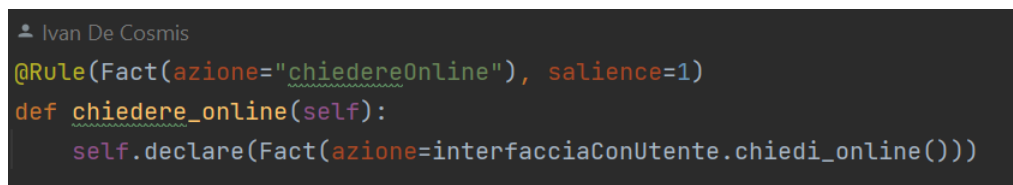
1. LHS(Left Hand Side): esprime le condizioni che devono essere verificate affinché la regola sia applicata.
2. RHS(Righ Hand Side): le operazioni eseguite quando la regola viene applicata.

Esempio LHS e RHS dal codice:



The screenshot shows a code editor with the following Python code: `@Rule(Fact(azione="chiedereOnline"), salience=1)` and `def chiedere_online(self): self.declare(Fact(azione=interfacciaConUtente.chiedi_online()))`. A blue box labeled 'LHS' highlights the condition `Fact(azione="chiedereOnline")`. A green box labeled 'RHS' highlights the function definition `def chiedere_online(self):` and its body `self.declare(Fact(azione=interfacciaConUtente.chiedi_online()))`.

Per ogni operazione da compiere è stata associata una regola che permette di seguire un flusso di operazioni continuo in base alla condizione del sistema e alle richieste dell'utente. Il sistema esperto inizia chiedendo all'utente la modalità di inserimento dei dati necessari per l'identificazione della situazione meteorologica(figura 2.1).



The screenshot shows a code editor with the following Python code: `@Rule(Fact(azione="chiedereOnline"), salience=1)` and `def chiedere_online(self): self.declare(Fact(azione=interfacciaConUtente.chiedi_online()))`.

Figura 2.1: Richiesta delle informazioni necessarie

Nel caso in cui l'utente scelga di cercare online verrà chiesto il nome della città in cui verificare le condizioni meteorologiche, altrimenti il sistema chiederà all'utente una serie di dati riguardanti la condizione atmosferica della città in cui si trova, ovvero:

1. Scegliere la fascia oraria
2. Scegliere le condizioni meteo attuali
3. Inserire la temperatura attuale
4. Inserire quanto forte il vento sembra

```

Ivan De Cosmis
@Rule(OR(Fact(scelta="si"), Fact(azione="trovareInformazioniOffline")), salience=0)
def chiedere_informazioni(self):
    self.declare(Fact(fascia_oraria=interfacciaConUtente.chiedi_fascia_oraria()))
    self.declare(Fact(meteo=interfacciaConUtente.chiedi_meteo()))
    self.declare(Fact(temperatura=interfacciaConUtente.chiedi_temperatura()))
    self.declare(Fact(vento=interfacciaConUtente.chiedi_vento()))
    self.declare(Fact(azione="chiediStile"))

```

Figura 2.2: sistemaEsperto.py, 37

```

Effettuare la ricerca online con il nome della tua città?(si/no) no
Scegli la fascia oraria(mattina/sera) mattina
Scegli le condizioni meteo attuali(nuvoloso, scoperto, rovesci) nuvoloso
Inserisci la temperatura in gradi celsius 12
Inserisci quanto forte ti sembra il vento(non presente, moderato, teso, fresco, forte, molto forte) forte

```

Figura 2.3: Citando l'esecuzione del sistema

Grazie a queste informazioni il sistema identificherà la condizione meteorologica attuale e chiederà all'utente quale stile di indumenti preferisce indossare nella giornata.

```

Ivan De Cosmis
@Rule(Fact(azione="chiediStile"), salience=0)
def chiedere_stile(self):
    self.declare(Fact(stile=interfacciaConUtente.chiedi_look()))
    self.declare(Fact(azione="chiediDurata"))

```

Figura 2.4: sistemaEsperto.py, 45

```

Quale look preferisci oggi?(casual/elegante/sportivo) casual

```

Figura 2.5: Citando l'esecuzione del sistema

In seguito chiederà se l'utente dovrà dormire a casa oppure fuori, per poter eventualmente consigliare una cambiata.

```
Ivan De Cosmis
@Rule(Fact(azione="chiediDurata"), salience=0)
def chiedere_durata(self):
    self.declare(Fact(durata=interfacciaConUtente.chiedi_durata()))
    self.declare(Fact(azione="stampaStile"))
```

Figura 2.6: sistemaEsperto.py, 50

```
Torni a casa oppure dormi fuori?(casa/fuori) fuori
```

Figura 2.7: Citando l'esecuzione del sistema

Se il sistema avrà rilevato anomalie meteorologiche utilizzerà la rete bayesiana per calcolare la probabilità di prendere un malanno uscendo con quelle condizioni meteorologiche, chiedendo la tipologia di rete bayesiana da usare.

```
if tipo == "caldo" or tipo == "freddo":
    chiedi_tipo_rete()
    src = __import__('src.ReteBayesiana.retiBayesiane', globals(), locals())
    reteBayesiana = src.ReteBayesiana.retiBayesiane
```

Figura 2.8: interfacciaConUtente.py, 234

```
-----
Rilevata anomalia meteorologia, selezionare il tipo di rete bayesiana da utilizzare:
(1)Rete bayesiana data
(2)rete bayesiana con apprendimento dal dataset
Risposta: 2
```

Figura 2.9: Citando l'esecuzione del sistema

Infine viene proposto all'utente il completo di indumenti consigliato, se è necessaria una cambiata e viene allertato in caso di anomalie meteorologiche.

```
Ivan De Cosmis
@Rule(Fact(azione="stampaStile"),
      Fact(stile=MATCH.stile),
      Fact(fascia_oraria=MATCH.fascia_oraria),
      Fact(temperatura=MATCH.temperatura),
      Fact(meteo=MATCH.meteo),
      salience=0)
def stampare_vestiti(self, stile, fascia_oraria, temperatura, meteo):
    interfacciaConUtente.stampa_risultato(stile, fascia_oraria, temperatura, meteo)
    self.declare(Fact(azione="stampaRicambi"))
```

Figura 2.10: sistemaEsperto.py, 45

```
=====
RISULTATO DELL'ANALISI
=====
L'abbigliamento consigliato per oggi e':
Busto: maglione elegante
Gambe: pantaloni lunghi
Scarpe: pelle per gli uomini, tacchi per le donne
Ombrello: sconsigliato
-----
Si consiglia di portare una cambiata per la notte
-----
E' necessario portare una sciarpa pesante per coprire il collo!
Hai il 92.66% di probabilita' di prendere un colpo di freddo
=====
Esecuzione terminata con successo!
```

Figura 2.11: Citando l'esecuzione del sistema

Si può ovviamente estendere il progetto entrando nel dettaglio per quanto riguarda i componenti del completo di vestiti consigliato, estendere le condizioni meteo disponibili per potere effettuare un'analisi più approfondita, effettuare una ricerca locale per poter permettere all'utente di acquistare indumenti che non ha oppure realizzare degli studi per raccogliere dati più approfonditi per i dataset.

3 Ontologie

3.1 Descrizione generale

In informatica, un'ontologia è una rappresentazione formale, condivisa ed esplicita di una concettualizzazione di un dominio di interesse. Più nel dettaglio, si tratta di una teoria assiomatica del primo ordine esprimibile in una logica descrittiva. Il termine ontologia formale è entrato in uso nel campo dell'intelligenza artificiale e della rappresentazione della conoscenza, per descrivere il modo in cui diversi schemi vengono combinati in una struttura dati contenente tutte le entità rilevanti e le loro relazioni in un dominio. I programmi informatici possono poi usare l'ontologia per una varietà di scopi, tra cui il ragionamento induttivo, la classificazione, e svariate tecniche per la risoluzione di problemi.

3.2 Il tool Protege

Ho creato l'ontologia usata sfruttando un tool chiamato **Protege**(figura 3.2.1).

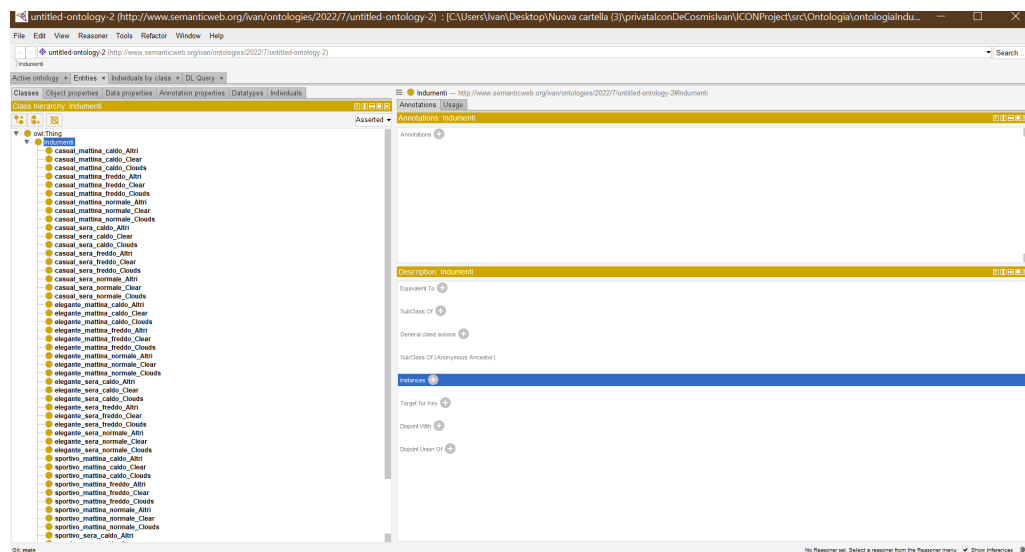


Figura 3.2.1: Il tool Protege con l'ontologia creata

L'ontologia è organizzata in diverse classi, ognuna delle quali rappresenta una configurazione di indumenti diversa, ovvero un caso che rappresenta la situazione meteorologica e le scelte dell'utente. Ogni classe contiene un individuo, con delle proprietà che rappresentano la configurazione di indumenti (figura 3.2.1).

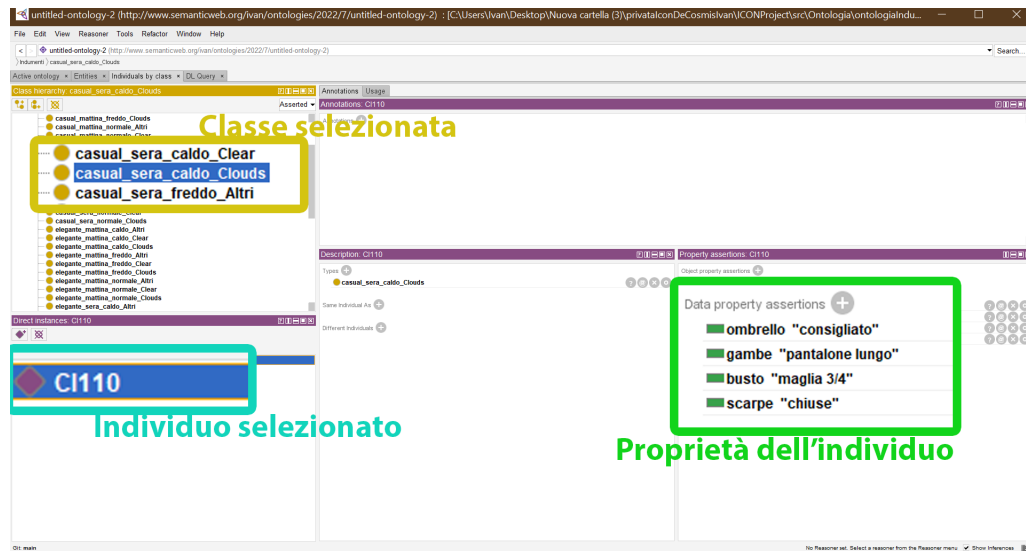


Figura 3.2.1: Il tool Protege con l'ontologia creata

3.3 La libreria Owlready2

L'ontologia viene letta usando la libreria python **Owlready2** (figura 3.3.1).

```

onto = get_ontology(path).load()
set = ["busto", "gambe", "scarpe", "ombrello"]
i = onto[stile+"_"+fascia_oraria+"_"+temperatura+"_"+meteo].instances()[0]
indice = 0
while indice < 4:
    for prop in i.get_properties():
        if set[indice] == prop.python_name:
            indice += 1
            print("%s: %s" % (prop.python_name.capitalize(), prop[i][0]))
            break

```

Figura 3.3.1: interfacciaConUtente.py, 230.

Dopo aver caricato l'ontologia da file, sfrutto la nomenclatura delle classi per poter recuperare le informazioni che mi servono senza dover effettuare una ricerca per classe. In seguito leggo le proprietà del singolo individuo nella classe e le stampo nell'ordine corretto.

4 Reti Bayesiane

4.1 Descrizione generale

Un **Rete Bayesiana** è un modello grafico probabilistico che rappresenta un insieme di variabili stocastiche con le loro dipendenze condizionali attraverso l'uso di un grafo aciclico diretto (DAG).

Formalmente le reti Bayesiane sono grafi diretti aciclici (ovvero non ha cicli diretti, scegliendo un vertice del grafo non possiamo tornare ad esso percorrendo gli archi del grafo) i cui nodi rappresentano variabili casuali in senso Bayesiano: possono essere quantità osservabili, variabili latenti, parametri sconosciuti o ipotesi. Gli archi rappresentano condizioni di dipendenza; i nodi che non sono connessi rappresentano variabili che sono condizionalmente indipendenti tra di loro. Ad ogni nodo è associata una funzione di probabilità che prende in input un particolare insieme di valori per le variabili del nodo genitore e restituisce la probabilità della variabile rappresentata dal nodo.

Sfrutta il teorema di Bayes per calcolare la probabilità di un evento che dipende (oppure non dipende) da altri eventi (da qui il nome **Probabilità Condizionata**).

In una rete bayesiana ogni nodo che deriva da uno o più genitori ha una **probabilità condizionata** di:

$$P(\text{nodo} | g(\text{nodo}))$$

La funzione **g(nodo)** ritorna tutti i genitori del nodo inserito come parametro di input.

Grazie a questo ogni nodo avrà una tabella CPD (Tabelle della probabilità condizionata).

La formula per calcolare la probabilità di ogni nodo è la seguente:

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

con X_i nodi della rete bayesiana. Dopo aver ottenuto una rete bayesiana e la relativa struttura DAG con le tabelle di CPD per ogni nodo si può inferire la probabilità che un evento specifico avvenga basando il calcolo su altri eventi.

Ci sono due tipi di inferenza:

1. Inferenza **esatta**: si enumera i mondi coerenti con le osservazioni e in seguito si utilizza un algoritmo con cui calcolare la probabilità esatta dell'evento in questione. Nel progetto verrà usato l'algoritmo delle eliminazioni delle variabili (fornito standard dalla libreria usata).
2. Inferenza **approssimata**: si va solamente a stimare la probabilità di un evento

Se non è possibile avere le tabelle CPD a priori, si possono analizzare i dati e attraverso determinati algoritmi (stimatori di Bayes, della massima verosimiglianza, ecc..) ricavare le CPD dei nodi di una DAG

4.2 Descrizione delle reti bayesiane implementate

Nel caso della rete usata per calcolare la probabilità di un malanno in caso di condizione atmosferica gelida, la probabilità di ammalarsi (nodo Colpo_di_freddo) dipende dalla temperatura (nodo Freddo) e dal vento (nodo Vento). La relazione d'ordine è facilmente individuabile dalla figura 4.2.1.

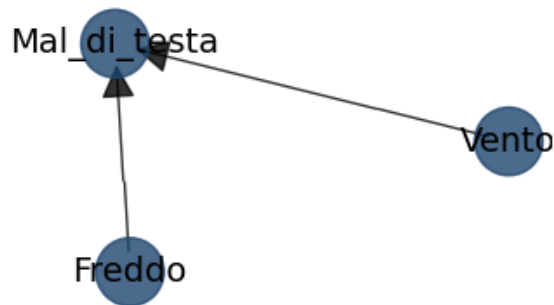


Figura 4.2.1

Invece nel caso della rete usata per calcolare la probabilità di un malanno in caso di condizione atmosferica torrida, la probabilità di ammalarsi (nodo Colpo_di_sole) dipende dalla temperatura (nodo Caldo). La relazione d'ordine è facilmente individuabile dalla figura 4.2.2

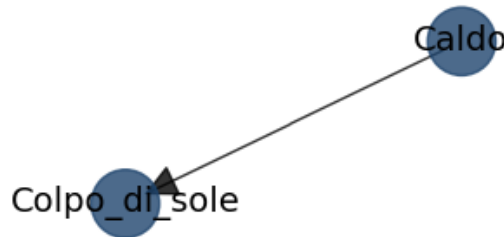


Figura 4.2.2

L'implementazione di queste due reti è specificata nel file **retiBayesiane.py** usando la libreria **bnlearn**. Con **bnlearn** è stato possibile creare una rete bayesiana con la struttura DAG, inserire delle tabelle CPD per ogni singolo nodo e inferire le probabilità di un nodo segnalando la gravità della condizione meteorologica attraverso il metodo della eliminazione delle variabili. Possiamo anche imparare le tabelle CPD **stimando un dataset**. La libreria fornisce due algoritmi: **stimatore di Bayes** e **stimatore di massima verosimiglianza**, noi useremo il secondo.

4.3 Implementazione della rete bayesiana

Come primo passaggio bisogna creare la struttura della DAG. Per fare questo andiamo a istanziare un vettore di coppie di stringhe, ogni coppia rappresenta due nodi del grafo indicando l'arco che li collega (figura 4.3.1).

```
# Creo i bordi DAG
self.Bordi = [('Vento', 'Colpo_di_freddo'),
               ('Freddo', 'Colpo_di_freddo')]
```

Figura 4.3.1: retiBayesiane.py, 11, Bayesianafreddo

Adesso abbiamo due casi:

1. Tabelle CPD date
2. Tabelle CPD da apprendere

Tabelle CPD date

Nel caso in cui le tabelle CPD siano date, andiamo a creare e assegnare le tabelle CPD.

Ad ogni nodo verrà assegnata una tabella che andremo a descrivere:

1. Vengono indicati, se li ha, i genitori del nodo (evidenze)
2. Vengono dichiarati tutti i possibili stati in cui un nodo può trovarsi
3. Vengono inserite le tabelle di verità: questa tabella ha come colonne le combinazioni degli stati delle evidenze (la somma di ogni colonna deve essere uguale a 1) e come righe gli stati del nodo considerato. In questo modo assegniamo la probabilità che il nodo si trovi in uno specifico stato dalla combinazione degli stati delle evidenze.

Dopo aver creato la tabella viene assegnata alla DAG. Tutto viene descritto nella figura 4.3.2

The image shows a snippet of Python code for setting up a Bayesian network. The code is annotated with several labels in different colors:

- Dichiaro gli stati** (yellow): Points to the `variable='Colpo_di_freddo'` argument in the `TabularCPD` constructor.
- Inserisco la tabella di verità** (orange): Points to the `values` list, which contains a 10x5 matrix of probabilities.
- Indico le evidenze** (cyan): Points to the `evidence=['Vento', 'Freddo']` and `evidence_card=[5, 5]` arguments.
- Assegno alla DAG** (green): Points to the `self.CPD_colpo_di_freddo` assignment.

The code itself is as follows:

```
self.CPD_colpo_di_freddo = TabularCPD(variable='Colpo_di_freddo', variable_card=2,
values=[[0.90, 0.70, 0.65, 0.50, 0.35,
0.80, 0.60, 0.45, 0.35, 0.15,
0.70, 0.55, 0.40, 0.20, 0.05,
0.60, 0.40, 0.30, 0.10, 0.05,
0.45, 0.30, 0.15, 0.05, 0.01],
[0.10, 0.30, 0.35, 0.50, 0.65,
0.20, 0.40, 0.55, 0.65, 0.85,
0.30, 0.45, 0.60, 0.80, 0.95,
0.40, 0.60, 0.70, 0.90, 0.95,
0.55, 0.70, 0.85, 0.95, 0.99]],
evidence=['Vento', 'Freddo'],
evidence_card=[5, 5])

# Collego il DAG con ogni CPT
self.DAG = bnlearn.make_DAG(self.Bordi, CPD=[
    self.CPD_vento,
    self.CPD_freddo,
    self.CPD_colpo_di_freddo
], verbose=0)
```

Figura 4.3.2: retiBayesiane.py, 26, BayesiananaFreddo

Dopo aver creato la rete bayesiana bisogna recuperare le informazioni dall'utente (situazione meteorologica, stile preferito) La modalità con cui vengono recuperate le informazioni dall'utente è stata descritta del capitolo **2 Sistema Esperto**.

In seguito inferiremo la probabilità che l'utente possa prendersi un malanno (figura 4.3.3) conoscendo lo stato atmosferico e segnaleremo all'utente il pericolo e la relativa contromisura (figura 2.11).

```
dati = {'Vento': vento,
        'Freddo': temp}
risultato = reteBayesiana ottieni_risultato_query(rete_bayesiana_freddo.inferenza(dati))
probabilita = (risultato["p"])[1] * 100
```

Figura 4.3.3: interfacciaConUtente.py, 278

Tabelle CPD da apprendere

È possibile anche apprendere le tabelle CPD fornendo un dataset al sistema, ad esempio nel caso di condizione atmosferiche gelide: **dataset_freddo.csv** e andando ad usare lo **stimatore di massima verosimiglianza** creiamo una rete bayesiana completa, con le probabilità apprese dal dataset piuttosto che date (figura 4.3.4).

```
Ivan De Cosmis
def impara_dataset(self, dataset, metodo):
    self.DAG = bnlearn.make_DAG(self.Bordi,
                                verbose=0)
    self.DAG = bnlearn.parameter_learning.fit(self.DAG,
                                                dataset,
                                                methodtype=metodo,
                                                verbose=0)
```

Figura 4.3.3: retiBayesiane.py, BayesianaFreddo, 60

4.4 Differenze tra tabelle CPD date e tabelle CPD apprese

Prestazioni

Basare una rete bayesiana su delle tabelle CPD date richiede molto meno sforzo computazionale piuttosto che andare a creare una rete basata su un dataset.

Precisione

Una rete bayesiana basata su dataset è più precisa a causa della grossa mole di dati su cui andare a fare le previsioni, che rispecchiando più facilmente la realtà.

Praticità

Se da una parte è più veloce creare una rete bayesiana dando a priori delle tabelle CPD, se il progetto ha una vita lunga diventa complicato andare ad aggiornare tabella per tabella con nuovi dati, è più semplice ampliare un dataset.

5 Articoli utilizzati

Ho sfruttato le informazioni nei seguenti articoli per farmi un'idea riguardante come, ad esempio, avvengono colpi di sole o malanni dovuti al freddo (come mal di gola), per poter generare i dataset su cui basare il sistema. Ovviamente sarebbe utile compiere uno studio su larga scala per raccogliere dati più realistici possibili.

Articoli riguardanti i colpi di sole:

- [https://www.mayoclinic.org/diseases-conditions/heat-stroke/symptoms-causes/syc-20353581#:~:text=Overview,\(40%20C\)%20or%20higher.](https://www.mayoclinic.org/diseases-conditions/heat-stroke/symptoms-causes/syc-20353581#:~:text=Overview,(40%20C)%20or%20higher.)
- <https://my.clevelandclinic.org/health/diseases/21812-heatstroke>
- <https://www.ncbi.nlm.nih.gov/books/NBK537135/>

Articoli riguardanti malanni collegati al freddo:

- <https://www.medicalnewstoday.com/articles/323431>
- <https://earandsinusinstitute.com/how-cold-weather-affects-your-ear-nose-throat/>
- <https://www.entslc.com/blog/runny-nose-sore-throat-ear-pain-more-during-cold-winter-weather-in-tooele-ut>
- <https://www.winchesterhospital.org/health-library/article?id=156976>

6 Conclusioni

- Linguaggio di programmazione usato: **Python**
- Documentazione realizzata in: **LaTeX**
- Link repository GitHub: <https://github.com/i-decosmis/progettoIconDeCosmisIvan>
- Studente: **De Cosmis Ivan**
- Matricola: **716486**
- Mail universitaria: **i.decosmis@studenti.uniba.it**