Documentazione Progetto ICON

Consulente per il tempo libero e lo sport

Alessandro Carrisi (MAT. 736830)

URL Repository: https://github.com/carrisi/ICON_24-25

Anno Accademico: 2024-2025

Sommario

1 Introduzione

1.1 Introduzione al problema

Organizzare il proprio tempo libero può risultare complicato quando le condizioni meteorologiche influenzano le attività che è possibile svolgere. Questo sistema ha l'obiettivo di suggerire all'utente attività ricreative e sportive compatibili con il meteo attuale e con le sue preferenze personali.

Il sistema consiglia attività sia all'aperto (*outdoor*) che al chiuso (*indoor*), tenendo conto delle condizioni atmosferiche e dell'eventuale disponibilità di strutture coperte. Inoltre, in caso di condizioni meteorologiche particolarmente sfavorevoli, il sistema notifica l'utente e suggerisce attività alternative più adatte per evitare disagi dovuti a fattori ambientali.

Essendo il progetto molto specifico, non erano disponibili dataset pubblici già pronti sul dominio. Per questo motivo, sono stati creati dataset personalizzati raccogliendo informazioni da articoli e statistiche riguardanti la compatibilità tra condizioni meteo e attività. Ad esempio, è stato applicato un semplice principio logico:

vento molto forte + pioggia intensa = sconsigliato sport outdoor

per orientare la raccolta dei dati e la definizione delle regole.

1.2 Approccio Utilizzato

Le attività vengono consigliate basandosi sulle informazioni fornite dall'utente. Queste informazioni sono raccolte attraverso una serie di domande che permettono di identificare la situazione meteorologica, la disponibilità di strutture indoor e il tipo di attività preferito. Il sistema sfrutta il concetto di ***Forward Chaining*** in un motore di regole logiche (similmente a Prolog, ma implementato in Python).

Durante l'interazione, il sistema pone domande all'utente per raccogliere i dati necessari e, sulla base delle risposte ricevute, decide quale attività suggerire e se segnalare eventuali condizioni meteorologiche avverse. Per valutare l'idoneità di un'attività in base al meteo, il sistema utilizza delle **regole di derivazione** basate sul **Modus Ponendo Ponens**.

In termini logici, la regola di inferenza Modus Ponens afferma che:

se p è una clausola specificata nella base di conoscenza e tutte le sue condizioni sono soddisfatte, allora si può derivare p.

Formalmente:

$$\textit{attività consigliata} \leftarrow \textit{condizione meteo favorevole} \land \textit{preferenza utente}$$

Si definisce **Regola** se il numero di condizioni nel LHS è maggiore di 0,

mentre si definisce **Fatto** se il numero di condizioni è uguale a 0.

La base di conoscenza del sistema che determina la compatibilità delle attività con il meteo è basata su alcune clausole logiche fondamentali, ad esempio:

attività outdoor consigliata ← *cielo sereno* ∧ *temperatura mite*
attività outdoor sconsigliata ← *pioggia intensa* ∨ *vento forte*
attività indoor consigliata ← *condizioni meteo avverse*

L'idoneità delle attività è quindi influenzata sia dalle condizioni meteorologiche sia dalle preferenze personali dell'utente. Se il sistema rileva anomalie climatiche, verrà valutata la probabilità di insoddisfazione e suggerite alternative più appropriate (come descritto nei capitoli successivi).

1.3 Elenco Argomenti di interesse

Questo progetto integra diversi concetti trattati nel corso di Ingegneria della Conoscenza, distribuiti tra più sezioni del programma. I principali argomenti affrontati e la loro applicazione nel sistema sono i seguenti:

- **Clausole di Horn e rappresentazione della conoscenza (Capitolo 2 - Sistema Esperto)** – Il motore inferenziale utilizza regole logiche basate su clausole di Horn per selezionare le attività consigliate in base alle condizioni meteorologiche fornite.
- **Ragionamento automatico (Capitolo 3 - Ontologie)** – Il sistema sfrutta un'ontologia per strutturare e organizzare la conoscenza relativa alle attività disponibili e alle condizioni ambientali, permettendo inferenze aggiuntive sulle attività.
- **Apprendimento e incertezza, ragionamento su KB distribuite (Capitolo 4 - Reti Bayesiane)** – Il modello probabilistico implementato con le reti bayesiane permette di gestire l'incertezza e di stimare la probabilità che un'attività risulti soddisfacente per l'utente, date le condizioni correnti.

Questi concetti sono stati integrati in un unico sistema in modo da renderlo flessibile e adattabile, migliorando la qualità delle raccomandazioni in base alle preferenze dell'utente e alle condizioni climatiche.

2 Sistema Esperto

2.1 Descrizione generale

Un **sistema esperto** è un programma progettato per riprodurre le capacità decisionali di un esperto umano in un determinato dominio. Il sistema sviluppato in questo progetto si occupa di suggerire attività ricreative e sportive basandosi sulle condizioni meteorologiche e sulle preferenze dell'utente.

L'obiettivo è fornire suggerimenti che massimizzino il grado di soddisfazione dell'utente, evitando attività scomode o non realizzabili a causa del meteo. In altre parole, il sistema cerca di proporre attività appropriate per il contesto ambientale corrente, simulando il ragionamento che farebbe un consulente umano esperto di tempo libero e meteorologia.

2.2 Realizzazione

Il sistema esperto è stato implementato utilizzando Python e la libreria **Experta**, che permette di definire regole logiche e di gestire un motore inferenziale basato sul forward chaining. Le decisioni vengono prese in base a un insieme di *regole* e *fatti*, dove:

- **LHS (Left Hand Side)** rappresenta le condizioni che devono essere soddisfatte affinché una regola venga attivata.
- **RHS (Right Hand Side)** indica l'azione da eseguire quando la regola viene applicata (cioè quando le condizioni del LHS risultano tutte vere).

Esempio LHS e RHS dal codice:

```python
# Regola per la richiesta della modalità di ricerca (online)
@Rule( *args: Fact(azione="chiedereOnline"), salience=1)
def chiedere_online(self):
    # Chiede all'utente se vuole cercare i dati meteo online e dichiara il fatto
    # corrispondente.
    self.declare(Fact(azione=interfacciaConUtente.chiedi_online()))
```

Figura 2.2.1: Richiesta modalità di immissione dei dati

LHS:

```python
# Regola per la richiesta della modalità di ricerca (online)
@Rule( *args: Fact(azione="chiedereOnline"), salience=1)
```

RHS:

```python
def chiedere_online(self):
    # Chiede all'utente se vuole cercare i dati meteo online e dichiara il fatto
    # corrispondente.
    self.declare(Fact(azione=interfacciaConUtente.chiedi_online()))
```

Per ciascuna operazione prevista è stata definita una regola che garantisce un percorso operativo ininterrotto, adattabile allo stato del sistema e alle esigenze dell'utente. Il sistema esperto si avvia richiedendo all'utente di scegliere la modalità di immissione dei dati necessari per identificare le condizioni meteorologiche (*Figura 2.2.1*)

Nel caso in cui l'utente scelga di cercare online, il sistema utilizzerà le API di **OpenWeatherMap** (tramite geocoding con *GeoPy* e chiamata HTTP con *Requests*) per ottenere temperatura, condizioni del cielo e vento in tempo reale. Questa integrazione con un'API esterna arricchisce la base di fatti iniziale in modo automatico, migliorando l'esperienza utente e l'accuratezza dei suggerimenti (*Figura 2.2.2*)

```
Vuoi effettuare la ricerca online con il nome della tua città? (si/no) SI
Dove ti trovi? BARI

---------------------- DATI METEO RECUPERATI ----------------------
Città -> BARI
Fascia oraria -> mattina
Temperatura (grezza) -> freddo
Vento (km/h) -> 2
Pioggia -> 0
-------------------------------------------------------------------
```

Figura 2.2.2: Esecuzione del sistema – Recupero dati Online

```
# ================================================================================
# Funzione: ricerca_previsioni_online()
# Costruisce l'URL per l'API "/weather" e processa la risposta per
# estrarre le informazioni meteo essenziali.
# ================================================================================
1 usage
def ricerca_previsioni_online(lat, lon, api_key):
    global temp, vento, tipo, fascia, pioggia
    # Costruisce l'URL dell'API usando le coordinate e l'API key
    url = f"https://api.openweathermap.org/data/2.5/weather?lat={lat}&lon={lon}&appid={api_key}&units=metric"
```

Figura 2.2.3: interfacciaUtente.py – creazione URL per recupero dati Online

altrimenti il sistema esperto raccoglierà informazioni dall'utente attraverso una serie di domande (*Figura 2.2.4*) iniziali riguardanti la condizione atmosferica della città in cui si trova, ovvero:

- La **fascia oraria** (mattina/sera).
- Le **condizioni meteorologiche** attuali (es. sereno, nuvoloso, pioggia, ecc.).
- La **temperatura** corrente.
- La presenza e l'intensità del **vento**.
- La **preferenza** dell'utente tra tipologie di attività: sportiva, culturale o ricreativa.
- L'eventuale accesso a strutture **indoor** (sì/no).

```
Vuoi 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): SCOPERTO
Inserisci la temperatura in gradi Celsius: 18
Inserisci quanto forte ti sembra il vento (non presente, moderato, teso, fresco, forte, molto forte): MODERATO
```

Figura 2.2.4: Esecuzione del sistema - Raccolta dati

```python
# Regola per acquisire informazioni in modalità offline
@Rule( *args: OR( *args: Fact(scelta="si"), Fact(azione="trovareInformazioniOffline")), salience=0)
def chiedere_informazioni_offline(self):
    # Chiede all'utente la fascia oraria
    fascia = interfacciaConUtente.chiedi_fascia_oraria()
    # Chiede all'utente le condizioni del meteo
    meteo = interfacciaConUtente.chiedi_meteo()
    self.declare(Fact(fascia_oraria=fascia))
    self.declare(Fact(meteo=meteo))
    # Se le condizioni meteo sono "rovesci", chiede anche l'intensità della pioggia.
    if meteo.strip().lower() == "rovesci":
        self.declare(Fact(pioggia=interfacciaConUtente.chiedi_pioggia()))
    # Chiede all'utente la temperatura
    self.declare(Fact(temperatura=interfacciaConUtente.chiedi_temperatura()))
    # Chiede all'utente il vento
    self.declare(Fact(vento=interfacciaConUtente.chiedi_vento()))
    # Passa all'azione successiva, cioè la richiesta del tipo di attività.
    self.declare(Fact(azione="chiediAttivita"))
```

Figura 2.2.5: sistemaEsperto.py – Raccolta dati

Con queste informazioni, il sistema determinerà le condizioni meteorologiche attuali e chiederà all'utente di indicare quale tipo di attività intende praticare durante la giornata e se ha accesso a palestre o strutture indoor in generale (*Figura 2.2.6*).

```
Quale tipo di attività preferisci oggi? (sportiva/culturale/ricreativa) SPORTIVA
Hai accesso a una palestra o a una struttura indoor? (si/no) NO
```

Figura 2.2.6: Esecuzione del sistema – Raccolta dati su attività e strutture

```python
# Regola per acquisire il tipo di attività e l'accesso a strutture indoor,
# quindi valutare l'allerta meteo tramite rete bayesiana.
@Rule( *args: Fact(azione="chiediAttivita"), salience=0)
def chiedere_attivita(self):
    # Chiede al'utente il tipo di attività preferita
    attivita = interfacciaConUtente.chiedi_attivita()
    self.declare(Fact(attivita=attivita))
    # Chiede all'utente se ha accesso a strutture indoor
    indoor_risposta = interfacciaConUtente.chiedi_indoor()
    self.declare(Fact(indoor=indoor_risposta))
    # Valuta il rischio meteo attraverso la rete bayesiana.
    rischio_alto = interfacciaConUtente.stampa_allerta_meteo()
    if rischio_alto:
        # Se viene rilevata un'anomalia meteo, chiede all'utente di scegliere la rete bayesiana.
        self.declare(Fact(azione="chiediTipoRete"))
    else:
        # Altrimenti, passa direttamente alla stampa dell'attività consigliata.
        self.declare(Fact(azione="stampaAttivita"))
```

Figura 2.2.7: sistemaEsperto.py – Raccolta dati su attività e strutture

Se il sistema rileva condizioni meteorologiche avverse, utilizzerà la rete Bayesiana per calcolare il rischio di insoddisfazione in percentuale nello svolgere le attività proposte con le condizioni meteo selezionate, chiedendo la tipologia di rete da utilizzare per calcolarla (*Figura 2.2.8*).

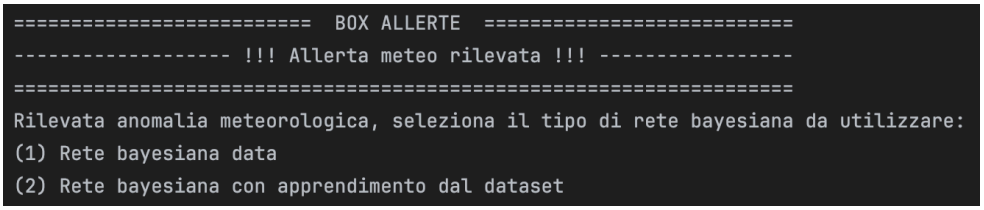

Figura 2.2.8: Esecuzione del sistema – Selezione tipo di rete Bayesiana

Infine, viene proposto all'utente il rischio di insoddisfazione, una lista di attività consigliate, una di attività non consigliate e una serie di accessori consigliati (*Figura 2.2.9*).

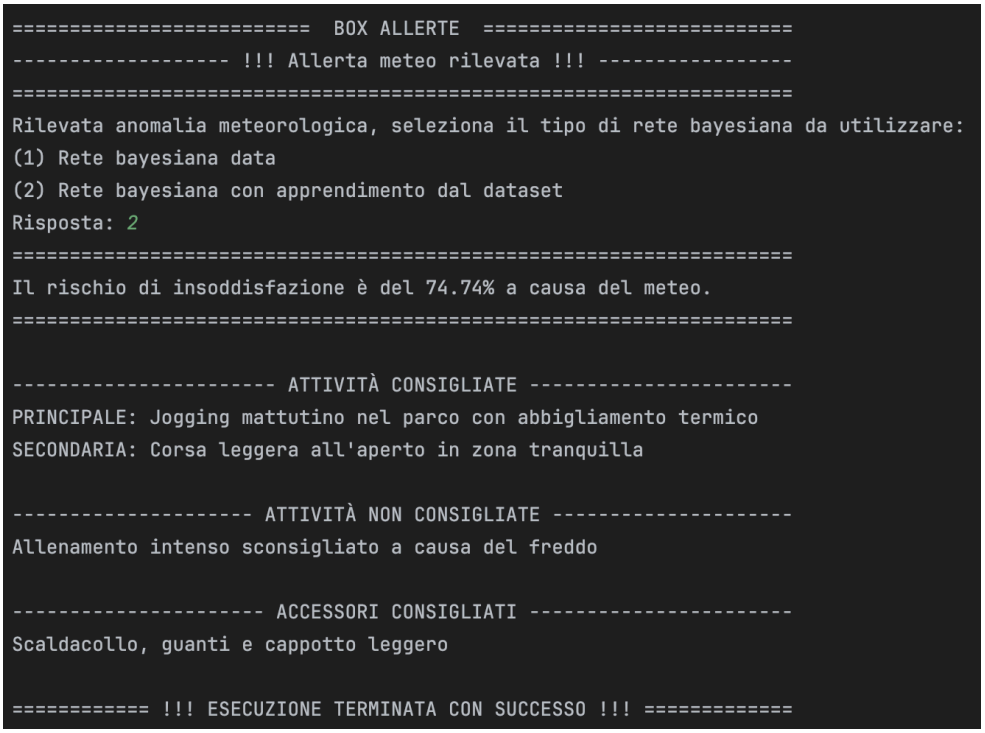

Figura 2.2.9: Esecuzione del sistema – Output finale

3 Ontologie

3.1 Descrizione generale

In ambito informatico, un'**ontologia** rappresenta una struttura formale per organizzare e gestire la conoscenza in un determinato dominio. Nel nostro progetto, l'ontologia viene utilizzata per classificare le attività ricreative e sportive in relazione alle condizioni meteorologiche e alle preferenze dell'utente. In altri termini, definisce concetti (classi di attività, condizioni di tempo, tipologie di ambienti) e relazioni tra questi concetti, permettendo al sistema di *ragionare* sulla conoscenza del dominio in modo più espressivo di una semplice base di dati.

L'impiego di un'ontologia consente di modellare gerarchie di attività (ad esempio, distinguere tra **AttivitàOutdoor** e **AttivitàIndoor**, o tra **AttivitàSportiva** e **AttivitàCulturale**) e di specificare i legami tra attività e contesto meteorologico. Questo arricchisce la conoscenza del sistema, che può così effettuare inferenze aggiuntive o trovare attività alternative appartenenti alla stessa categoria in caso di necessità.

3.2 Sviluppo

L'ontologia di dominio è stata sviluppata utilizzando lo strumento **Protégé**, un software open-source che permette di creare modelli ontologici basati su logiche descrittive (OWL). L'ontologia creata è strutturata in diverse **classi**, ognuna delle quali rappresenta una particolare combinazione di fattori: tipologia di attività, ambiente (indoor/outdoor), fascia oraria e condizioni meteorologiche. Ciascuna classe contiene un **individuo** (istanza) che racchiude le proprietà descrittive per quella situazione specifica, ad esempio il tipo di attività consigliata, l'eventuale attività alternativa e gli accessori suggeriti.

Esempio: si consideri la situazione *"attività sportiva all'aperto con clima caldo"*. Nell'ontologia esiste una classe che rappresenta questo scenario; un individuo appartenente a tale classe potrebbe avere proprietà (*Figura 3.2.1*) come **haAttivitaPrincipale = nuoto** (come attività sportiva principale consigliata con tempo caldo), **haAttivitaSecondaria = corsa** (un'attività alternativa all'aperto) e **haAccessorioConsigliato = cappello** (accessorio consigliato per proteggersi dal sole). Analogamente, per una classe *"attività culturale indoor con clima freddo"*, l'individuo associato potrebbe indicare come attività principale la visita a un museo, come secondaria la lettura in biblioteca, e consigliare come accessorio una bevanda calda. Queste strutture permettono di collegare contesto meteorologico e raccomandazioni di attività in modo dichiarativo.

```
<haAttivitaPrincipale rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
  Corsa serale leggera con pausa idrica in zone ombreggiate
</haAttivitaPrincipale>
```

Figura 3.2.1: ontologiaAttivita.owl – esempio di una proprietà

L'uso di Protégé ha consentito di definire in maniera accurata le relazioni tra le entità (attività, condizioni meteo, preferenze, ecc.) e di verificare la coerenza dell'ontologia prima della sua integrazione nel sistema. In fase di sviluppo, la validazione ontologica ha garantito che per ogni combinazione di condizioni contemplata esistesse una corrispondente istanza con raccomandazioni appropriate.

3.3 Libreria

Per l'integrazione dell'ontologia nel sistema esperto, è stata utilizzata la libreria **Owlready2** in Python. Questa libreria permette di svolgere diverse operazioni utili:

- **Caricare e leggere l'ontologia** da un file OWL esterno.
- **Recuperare informazioni** sulle classi e le proprietà degli individui definiti nell'ontologia.
- **Utilizzare la nomenclatura** delle classi/istanze per estrarre direttamente le informazioni richieste senza dover effettuare ricerche manuali sull'intero grafo ontologico.

In pratica, dopo aver caricato l'ontologia dal file (ontologiaAttivita.owl), il sistema utilizza la struttura definita (classi, istanze e proprietà) per selezionare l'attività più appropriata in base al meteo e alle preferenze dell'utente. Ad esempio, il motore inferenziale può costruire una chiave di ricerca, composta da "tipo di attività", "ambiente", "fascia oraria", "temperatura categorizzata" e "condizione meteo" (*Figura 3.3.1*) per poi trovare nell'ontologia l'individuo corrispondente a quella combinazione, ottenendo così direttamente le attività consigliate e gli eventuali accessori.

```
# Costruisce la chiave per la ricerca nell'ontologia
chiave = ("attivita_"
          + attivita.strip().lower() + "_"
          + luogo_fallback + "_"
          + fascia_oraria.strip().lower() + "_"
          + temperatura.strip().lower() + "_"
          + meteo_fallback)
```

Figura 3.3.1: interfacciaUtente.py – Costruzione chiave per la ricerca

```
<owl:NamedIndividual rdf:about="#attivita_culturale_outdoor_mattina_normale_scoperto">
```

Figura 3.3.2: ontologiaAttività.owl – esempio di chiave

L'integrazione dell'ontologia consente al sistema di organizzare e recuperare le informazioni in modo strutturato, migliorando l'efficienza del processo decisionale.

4 Reti Bayesiane

4.1 Descrizione Generale

Una **Rete Bayesiana** è un modello grafico probabilistico che rappresenta un insieme di variabili casuali e le loro dipendenze condizionali tramite un grafo diretto aciclico (DAG - Directed Acyclic Graph). Le reti bayesiane consentono di effettuare inferenze probabilistiche, aggiornando le credenze sulle variabili ignote a partire da evidenze osservate, grazie al Teorema di Bayes.

Nel nostro progetto, la rete bayesiana viene utilizzata per stimare la probabilità di soddisfazione o di successo di un'attività consigliata, date le condizioni meteorologiche e le preferenze dell'utente. In pratica, aggiunge un livello di valutazione quantitativa al suggerimento qualitativo fornito dal sistema esperto.

Ogni **nodo** della rete rappresenta una variabile rilevante nel dominio, come ad esempio:

- Condizioni meteo (es. temperatura categorizzata, intensità del vento, presenza di pioggia).
- Tipologia di attività selezionata o contesto (es. attività outdoor vs indoor, o specifiche categorie di attività).
- Livello di soddisfazione o rischio di insoddisfazione dell'utente rispetto all'attività (es. alto, medio, basso, oppure un punteggio di consiglio da 0 a 4).

Le relazioni tra queste variabili sono modellate attraverso probabilità condizionate. La rete bayesiana sfrutta tali relazioni per calcolare, ad esempio, quanto è probabile che un'attività proposta non soddisfi l'utente date certe condizioni avverse, oppure la probabilità che un'attività outdoor sia effettivamente praticabile e gradita in base al meteo del momento. Questo approccio consente di gestire l'incertezza intrinseca nel problema (ad esempio, l'utente **potrebbe** comunque divertirsi nonostante un po' di pioggia, ma con una certa probabilità di disagio).

4.2 Descrizione Reti Implementate

Nel sistema sono state sviluppate **due reti bayesiane**, ciascuna con un obiettivo specifico legato alla valutazione delle attività:

- **Rete 1 – Compatibilità Outdoor:** La prima rete viene utilizzata per determinare se un'attività **all'aperto** sia consigliabile o meno nelle condizioni date. La probabilità che un'attività outdoor sia adatta (espressa

nel nodo *CompatibilitàOutdoor* o similare) è influenzata da fattori come la temperatura (nodo *Temperatura*) e la presenza di vento forte o precipitazioni (nodo *CondizioniMeteo*).

- **Rete 2 – Rischio di Insoddisfazione:** La seconda rete è impiegata per stimare la probabilità di **insoddisfazione dell'utente** a seguito della scelta di un'attività. In questa rete, il nodo centrale (ad es. *Insoddisfazione*) dipende sia dal tipo di attività scelta (nodo *AttivitàScelta*, che può riflettere se l'attività è appropriata o meno) sia dalle condizioni meteorologiche avverse (nodo *MeteoAvverso*, derivato da combinazioni di vento/pioggia/temperature estreme). Questa configurazione permette di calcolare la probabilità che l'utente rimanga scontento dell'attività proposta a causa del meteo.

Per l'implementazione delle reti bayesiane è stato utilizzato il modulo Python retiBayesiane.py, sfruttando la libreria **bnlearn**. Questo framework ha permesso di definire la struttura delle reti come DAG, assegnare manualmente le tabelle di probabilità condizionata (CPD – Conditional Probability Distribution) ai nodi e inferire, date alcune evidenze, la probabilità di determinati eventi (come un'attività compatibile o un alto rischio di insoddisfazione).

Oltre a definire manualmente le tabelle CPD, il sistema consente anche di **apprenderle dai dati** raccolti. In particolare, la libreria bnlearn offre due metodi di stima dei parametri a partire da dati osservati:

- **Stimatore di Bayes** – Utilizza una distribuzione a priori (con smoothing di Laplace) durante la stima, evitando probabilità nulle e incorporando preconoscenza bayesiana.
- **Stimatore di massima verosimiglianza (MLE)** – Stima i parametri che massimizzano la verosimiglianza dei dati. Questo secondo approccio è stato scelto nel progetto per la sua efficienza nell'adattare il modello ai dati reali disponibili.

In questo modo, si può partire da una rete con probabilità definite a mano e poi raffinarla attraverso l'apprendimento automatico se si dispone di un dataset di esempi sufficientemente rappresentativo.

4.3 Implementazione

Il primo passo nella costruzione di ciascuna rete bayesiana consiste nella definizione della sua struttura **DAG**. A tal fine, nel codice viene creato un insieme di coppie ordinate di stringhe, in cui ogni coppia rappresenta un arco (collegamento diretto) tra due nodi della rete. Questa rappresentazione consente di modellare in modo programmatico le dipendenze tra le variabili che influenzano la scelta delle attività in base alle condizioni meteorologiche (*Figura 4.3.1*).

```
# Definizione degli archi (Bordi) del DAG: ciascuna evidenza influenza "Consiglio"
self.Bordi = [
    ('Vento', 'Consiglio'),
    ('Freddo', 'Consiglio'),
    ('Pioggia', 'Consiglio')
]
```

Figura 4.3.1: retiBayesiane.py – Definizione dei bordi del DAG

Definita la struttura, si distinguono due approcci per specificare le tabelle CPD dei nodi:

Tabelle CPD predefinite – In questo approccio, le tabelle di probabilità condizionata vengono impostate manualmente dall'esperto. Per ogni nodo si descrive il comportamento probabilistico in funzione dei suoi genitori. Il processo tipico prevede i seguenti passaggi:

1. **Identificare le evidenze (nodi genitori)** che influenzano il nodo corrente (*Figura 4.3.1*).
2. **Definire gli stati possibili** in cui il nodo può trovarsi (*Figura 4.3.2*).
3. **Costruire la tabella di probabilità**: ogni colonna rappresenta una combinazione degli stati dei nodi genitori, e ogni riga indica la probabilità che il nodo figlio assuma uno specifico stato data quella combinazione di evidenze (*Figura 4.3.3*)

Una volta costruita ciascuna tabella, essa viene associata al rispettivo nodo all'interno della rete bayesiana. In fase di inizializzazione, il codice integra queste tabelle predefinite nella struttura del DAG, cosicché siano pronte per l'inferenza. *(Figura 4.3.2 e Figura 4.3.3).*

```
# ------------------------------------------------------------------------
# CPD per Vento: rappresenta la distribuzione a priori dei livelli di vento.
# I livelli più alti (valore 4) sono più probabili.
# ------------------------------------------------------------------------
self.CPD_vento = TabularCPD(
    variable='Vento', variable_card=5,
    values=[
        [0.05],
        [0.10],
        [0.20],
        [0.25],
        [0.40]
    ],
    state_names={'Vento': [0, 1, 2, 3, 4]}
)
```

Figura 4.3.2: retiBayesiane.py – nodo genitore "Vento"

```python
def genera_cpd_consiglio():
    values = [[], [], [], [], []]  # liste per p(Consiglio=0..4)
    # Cicla su tutte le possibili combinazioni delle evidenze (5x5x5 = 125 combinazioni)
    for vento in range(5):
        for freddo in range(5):
            for pioggia in range(5):
                massimo = max(vento, freddo, pioggia)
                if massimo == 0: rischio = 0.0    # condizioni ottimali: rischio 0%
                elif massimo == 1: rischio = 0.25   # condizioni lievemente sfavorevoli: 25%
                elif massimo == 2: rischio = 0.40   # condizioni moderate: 40%
                elif massimo == 3: rischio = 0.60   # condizioni difficili: 60%
                else: rischio = 0.80   # condizioni estreme: 80%
                # Suddivide il rischio: metà per lo stato 3, metà per lo stato 4.
                p3 = rischio / 2
                p4 = rischio / 2
                restante = 1 - rischio
                if restante < 0:
                    restante = 0.0
                p0 = p1 = p2 = restante / 3 if restante > 0 else 0.0
                # Aggiunge le probabilità per questa combinazione
                values[0].append(p0); values[1].append(p1); values[2].append(p2)
                values[3].append(p3); values[4].append(p4)
    return values

self.CPD_consiglio = TabularCPD(
    variable='Consiglio', variable_card=5,
    values=genera_cpd_consiglio(),
    evidence=['Vento', 'Freddo', 'Pioggia'],
    evidence_card=[5, 5, 5],
    state_names={
        'Consiglio': [0, 1, 2, 3, 4],
        'Vento': [0, 1, 2, 3, 4],
        'Freddo': [0, 1, 2, 3, 4],
        'Pioggia': [0, 1, 2, 3, 4]
    }
)
```

Figura 4.3.3: retiBayesiane.py – Definizione possibili stati

Completata la definizione della rete bayesiana (struttura + CPD), il sistema procede all'inferenza utilizzandola assieme ai dati raccolti dall'utente. Quando l'utente interagisce col sistema esperto (*Capitolo 2 della docuementazione)* inserendo le informazioni su meteo e preferenze, queste vengono anche trasformate in **evidenze** per la rete bayesiana. In particolare, dopo che il motore di regole ha scelto un'attività candidata, la rete bayesiana calcola la probabilità che tale scelta sia ottimale. Ad esempio, il sistema calcola la probabilità che un'attività suggerita sia poco gradita all'utente a causa del maltempo e, se tale probabilità supera una certa soglia, può avvisare l'utente o passare a un'alternativa (*4.3.4)*.

```python
def stampa_allerta_meteo():
    global vento, temp, fascia, rete, tipo, indoor, meteo, reteAggiornata
    # ------------------------------------------------------------------------
    # Caso "freddo"
    # ------------------------------------------------------------------------
    if tipo == "freddo":
        # Determina il luogo (indoor/outdoor)
        luogo = "indoor" if indoor.strip().lower() == "si" else "outdoor"
        import pandas as pd
        # Carica il dataset del ramo freddo (il dataset attuale non ha "Pioggia" per il ramo freddo)
        dataset = pd.read_csv("src/ClassiSupporto/dataset_consulente_freddo_ottimale.csv")
        dataset = dataset[['Vento', 'Freddo', 'Consiglio']]
        # Prepara l'evidenza includendo anche la pioggia (per coerenza, sebbene il dataset non la usi)
        evidenza = {'Vento': int(vento), 'Freddo': int(temp), 'Pioggia': int(pioggia)}
        if DEBUG:
            print("Evidenza per inferenza (freddo):", evidenza)
        from src.ReteBayesiana import retiBayesiane as rb
        rete_bayesiana_temp = rb.BayesianaInsoddisfazione()
        risultato_default = rb.ottieni_risultato_query(rete_bayesiana_temp.inferenza(evidenza))
        p_default = risultato_default["p"]
        probabilita_rischio_default = (p_default.iloc[3] + p_default.iloc[4]) * 100
        if probabilita_rischio_default < 35:
            print("\n=========================  BOX ALLERTE  ==========================")
            print("--------------- !!! Nessun'allerta meteo rilevata !!! --------------")
            print("================================================================")
            return False
        else:
            print("\n=========================  BOX ALLERTE  ==========================")
            print("------------------ !!! Allerta meteo rilevata !!! -----------------")
            print("================================================================")
            reteAggiornata = rete_bayesiana_temp
            return True
```

Figura 4.3.4: interfacciaUtente.py - Chiamata all'inferenza e gestione del risultato

Tabelle CPD apprese da dataset – In alternativa al setting manuale, è possibile far **apprendere automaticamente** le tabelle CPD utilizzando dati reali raccolti in uno o più dataset. Nel nostro progetto, dopo aver costruito la struttura della rete, è stato previsto un modulo che carica dati da file CSV contenenti esempi di situazioni meteo e relative valutazioni di soddisfazione. Ad esempio, per le attività outdoor, è stato creato un dataset storico (es. *dataset_consulente_freddo_ottimale.csv)* contenente informazioni su attività svolte in diverse condizioni climatiche e il relativo esito in termini di soddisfazione. Allo stesso modo, per il caso di condizioni fredde estreme, è stato predisposto un dataset con indicatori di malessere/insoddisfazione raccolti da fonti mediche e sportive.

Utilizzando l'algoritmo di massima verosimiglianza menzionato sopra, la rete bayesiana può aggiornare automaticamente le distribuzioni di probabilità dei propri nodi sulla base di questi dati, invece di affidarsi solo a valori impostati manualmente. Questo approccio conferisce al sistema una maggiore adattabilità e garantisce una stima più precisa delle probabilità di soddisfazione (o insoddisfazione) dell'utente, man mano che si incorporano dati più realistici (*Figura 4.3.5 e Figura 4.3.6*).

```python
def stampa_rischio_finale():
    global vento, temp, tipo, reteAggiornata, rete, indoor, pioggia
    if reteAggiornata is None:
        print("Nessuna rete aggiornata disponibile.")
        return
    # Aggiorna la rete con il dataset in base alla scelta dell'utente
    if rete == "2":
        import pandas as pd
        if tipo == "freddo":
            dataset = pd.read_csv("src/ClassiSupporto/dataset_consulente_freddo_ottimale.csv")
            dataset = dataset[['Vento', 'Freddo', 'Pioggia', 'Consiglio']]
        elif tipo == "caldo":
            dataset = pd.read_csv("src/ClassiSupporto/dataset_consulente_caldo_ottimale.csv")
            dataset = dataset[['Attività', 'Vento', 'Pioggia', 'Consiglio']]
        reteAggiornata.impara_dataset(dataset, "bayes")
```

Figura 4.3.5: interfacciaConUtente.py - Caricamento del dataset .csv

```python
# ------------------------------------------------------------------------
# Metodo impara_dataset: aggiorna i parametri della rete utilizzando un dataset
# Se il dataset contiene la colonna 'Attività' (ramo caldo) la usa, altrimenti usa 'Freddo' (ramo freddo).
# ------------------------------------------------------------------------
def impara_dataset(self, dataset, metodo="bayes"):
    if 'Attività' in dataset.columns:
        dataset = dataset[['Attività', 'Vento', 'Pioggia', 'Consiglio']]
    else:
        dataset = dataset[['Vento', 'Freddo', 'Pioggia', 'Consiglio']]
    self.DAG = bnlearn.make_DAG(self.Bordi, verbose=0)
    # MODIFICA: utilizza il metodo bayesiano con smoothing (Laplace smoothing, α=1)
    self.DAG = bnlearn.parameter_learning.fit(
        self.DAG, dataset,
        methodtype=metodo,
        verbose=0
    )
```

Figura 4.3.6: retiBayesiane.py - Aggiornamento delle CPD con bnlearn

In concreto, l'applicazione permette all'utente di scegliere se utilizzare la rete bayesiana con i parametri predefiniti o applicare l'apprendimento dal dataset prima di eseguire l'inferenza finale. Successivamente, fornisce un responso probabilistico: ad esempio, "*Il rischio di insoddisfazione è del 75% a causa del meteo*" (*Figura 4.3.7*), oppure viceversa "*Nessuna allerta meteo rilevata, condizioni ottimali*" (*Figura 4.3.8*).

```
==========================  BOX ALLERTE  ==========================
------------------ !!! Allerta meteo rilevata !!! -----------------
======================================================================
Rilevata anomalia meteorologica, seleziona il tipo di rete bayesiana da utilizzare:
(1) Rete bayesiana data
(2) Rete bayesiana con apprendimento dal dataset
Risposta: 2
======================================================================
Il rischio di insoddisfazione è del 32.0% a causa del meteo.
======================================================================
```

Figura 4.3.7: Esecuzione del sistema - Allerta meteo rilevata

```
==========================  BOX ALLERTE  ==========================
-------------- !!! Nessun'allerta meteo rilevata !!! --------------
======================================================================
```

Figura 4.3.8: Esecuzione del sistema – Nessun'allerta meteo rilevata

5 Articoli Utilizzati

Durante la progettazione sono state consultate diverse fonti per raccogliere conoscenza di background sul dominio **meteo & attività**. In particolare, si sono considerati articoli e report riguardanti l'impatto delle condizioni climatiche sulla praticabilità di sport all'aperto e sulla preferenza degli utenti per attività indoor in caso di maltempo. Ad esempio, dati statistici su come la pioggia o temperature estreme influenzino la frequentazione di parchi, palestre o musei hanno guidato la definizione sia delle regole nel sistema esperto sia dei dataset utilizzati per l'addestramento della rete bayesiana. Le seguenti fonti sono state indicative durante lo sviluppo:

- Articoli meteorologici divulgativi sulle relazioni tra clima e attività sportive (ad es. consigli per fare jogging con diverse temperature e livelli di umidità).
- Report e studi di settore sul turismo e le attività ricreative in condizioni climatiche avverse (che evidenziano come pioggia o ondate di calore incidono sulle scelte di svago).
- Statistiche pubblicate da organizzazioni sportive relative alla partecipazione ad eventi outdoor in funzione del meteo.

Queste informazioni sono servite da riferimento qualitativo nella creazione della base di conoscenza e nella simulazione di dati realistici per il sistema.

6 Conclusioni

In conclusione, il progetto realizzato dimostra come l'integrazione di tecniche di **Ingegneria della Conoscenza** possa contribuire a costruire un sistema di raccomandazione intelligente e sensibile al contesto. Attraverso il sistema esperto basato su regole logiche, un'ontologia di dominio per la rappresentazione strutturata della conoscenza e modelli probabilistici (reti bayesiane) per la gestione dell'incertezza, il sistema offre consigli personalizzati per il tempo libero adattandosi alle condizioni meteorologiche in tempo reale.

L'architettura sviluppata risulta flessibile e modulare: le regole esperte gestiscono la **fase decisionale iniziale** (cosa proporre), l'ontologia fornisce un **vocabolario condiviso** e consente di arricchire le raccomandazioni (fornendo dettagli come attività alternative e accessori consigliati), e le reti bayesiane aggiungono una **valutazione quantitativa** del consiglio proposto

(allertando l'utente su possibili rischi o confermando l'adeguatezza della scelta). Questa sinergia di approcci migliora la qualità delle raccomandazioni rispetto a un semplice sistema deterministico, poiché considera sia la conoscenza esperta codificata sia l'esperienza derivata dai dati.

Il risultato è un assistente in grado di suggerire all'utente l'attività migliore per le sue esigenze, spiegando eventuali avvertenze (ad esempio, allerta meteo) e mostrando di poter apprendere da nuove informazioni. In futuro, il progetto potrebbe essere esteso includendo un numero maggiore di attività e condizioni (ampliando l'ontologia), nonché integrando algoritmi di apprendimento automatico avanzato per affinare ulteriormente le previsioni di soddisfazione dell'utente. In definitiva, questa esperienza conferma l'efficacia di un approccio ibrido che combina knowledge engineering e machine learning per risolvere problemi complessi nel campo delle raccomandazioni intelligenti.