Hygiea- Sistema di Prevenzione di Patologie Basato su Conoscenza

Componenti del gruppo

- Mirko Patruno, [MAT. 797729], m.patruno46@studenti.uniba.it

Link GitHub: https://github.com/mirkopat/icon25-26_hygiea

A.A. 2025-2026

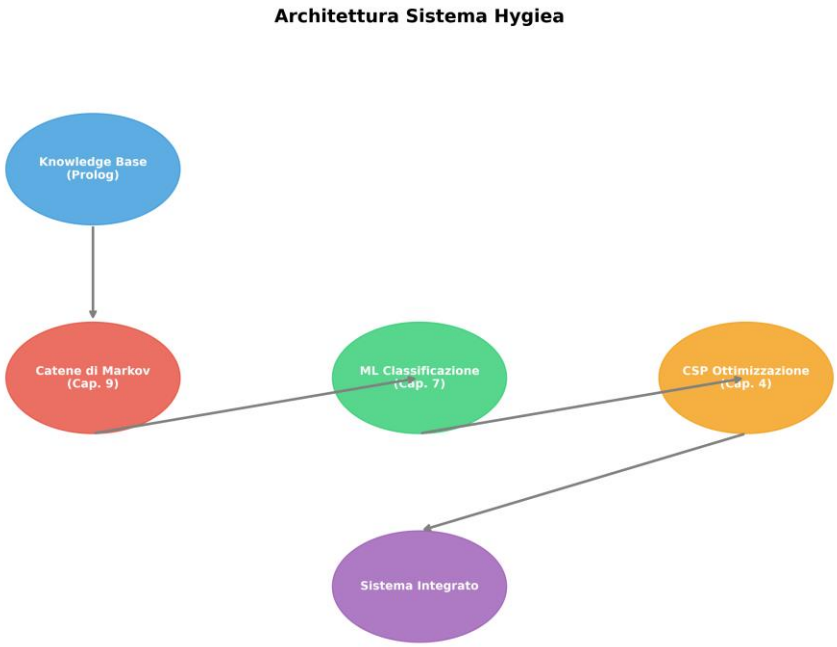
Indice

Capitolo 0) Introduzione

L'idea alla base di Hygiea (dal nome Igea, o Hygieia, dea greca della salute, dell'igiene e della prevenzione delle malattie) si focalizza sulla prevenzione di patologie su diversi profili. L'obiettivo di questo progetto è quindi lo sviluppo di un sistema basato su conoscenza per la prevenzione personalizzata di tre patologie croniche: Depressione (neurologica), Diabete (metabolica) e Ipertensione (cardiovascolare). Il sistema analizza profili paziente tramite tecniche di Intelligenza Artificiale integrate e raccomanda interventi preventivi ottimizzati.

Il sistema integra quattro concetti del corso di Ingegneria della Conoscenza, per:

1. Rappresentare conoscenza medica dichiarativa (KB Prolog - Cap. 15)
2. Analizzare l'evoluzione temporale del rischio (Catene di Markov - Cap. 9)
3. Classificare il livello di rischio corrente (Apprendimento Supervisionato - Cap. 7)
4. Ottimizzare le raccomandazioni di intervento (CSP - Cap. 4)



Requisiti funzionali

Questo progetto è stato realizzato in Python 3.10+ poiché offre numerose librerie per il trattamento dati e l'implementazione di algoritmi di IA. L'ambiente di sviluppo utilizzato è Visual Studio Code, con gestione delle dipendenze con file requirements.txt.

Librerie utilizzate:

- **pandas** e **numpy**: importazione e manipolazione dei dataset .csv, operazioni numeriche
- **scikit-learn**: libreria per apprendimento automatico (Random Forest, SVM, Decision Tree)
- **matplotlib** e **seaborn**: visualizzazione dei grafici (curve di apprendimento, heatmap, grafici comparativi)
- **pyswip**: integrazione di Prolog all'interno dell'ambiente Python
- **requests**: query HTTP per l'accesso a conoscenza esterna (DBpedia)
- **pathlib** - Gestione percorsi file cross-platform
- **tempfile** - Creazione file temporanei per query Prolog

Installazione e avvio

Clonare il repository GitHub nel proprio IDE, installare le dipendenze dal file requirements.txt (*pip install -r requirements.txt*), ed eseguire il sistema integrato partendo dal file 01_create_dataset.py. È necessario avere SWI-Prolog installato per il funzionamento della Knowledge Base. Ci sono due modalità a scelta dell'utente per l'esecuzione del programma:

- Esecuzione singola dei programmi:
  py src/01_create_dataset.py
  py src/02_markov_model.py
  py src/03_classification.py
  py src/04_csp_recommender.py
  py src/05_integration.py
  py src/06_create_images.py
  py src/integration_demo.py
- Esecuzione unica del programma:
  py src/main_system.py

Capitolo 1) Creazione del dataset e Knowledge Base

Generazione dataset sintetico

Il dataset è stato generato sinteticamente per simulare 60 pazienti monitorati per 7 giorni consecutivi, ottenendo una dimensione del dataset pari a 420 osservazioni totali. Per ogni paziente sono state generate feature demografiche statiche (età distribuita uniformemente tra 20-80 anni, genere casuale M/F, BMI tra 18-35) e parametri giornalieri variabili con pattern realistici: qualità del sonno (1-10 scala Likert), livello di stress (1-10), attività fisica (0-120 minuti), apporto di zuccheri (10-100 unità arbitrarie) e sale (2-15 unità).

La scelta di generare dati sintetici è stata dettata dalla necessità di avere un dataset con struttura temporale (7 giorni consecutivi) e feature specifiche per testare le Catene di Markov, non disponibile in dataset pubblici. Questo ha permesso di focalizzarmi sull'analisi sulle proprietà degli algoritmi piuttosto che sulla preparazione dei dati.

Il rischio per ciascuna patologia è calcolato come combinazione lineare pesata dei fattori rilevanti, con aggiunta di rumore gaussiano ($\mu=0$, $\sigma=0.05$) per simulare variabilità naturale. I valori risultanti sono poi classificati in tre categorie: Low (< 0.3), Medium (0.3-0.6), e High (> 0.6). Questa discretizzazione a tre livelli permette analisi Markoviane significative e classificazione multi-classe.

La generazione include anche correlazioni realistiche: pazienti con BMI alto tendono ad avere minore attività fisica, stress alto correlato con sonno scarso. La storia familiare è generata con probabilità differenziate: 20% per depressione, 25% per diabete, 30% per ipertensione.

Knowledge Base Prolog

La Knowledge Base in Prolog (wellness_kb.pl) contiene 15 fatti e 12 regole organizzati in tre livelli di inferenza. Il primo livello include regole base come rischio_alto/2 che identifica pazienti con almeno due fattori di rischio per una patologia utilizzando findall/3 per raccogliere tutti i fattori rilevanti e length/2 per contare le occorrenze.

```
% ============ METADATI ============
complessita_kb(media).
numero_regole(12).
numero_fatti(15).
profondita_max_inferenza(3).

% ============ ONTOLOGIA BASE ============
% Gerarchia patologie
tipo_patologia(neurologica, depressione).
tipo_patologia(metabolica, diabete).
tipo_patologia(cardiovascolare, ipertensione).

patologia(P) :- tipo_patologia(_, P).

% Interventi con costo
intervento(esercizio, 2).
intervento(dieta, 1).
intervento(mindfulness, 1).
intervento(sonno, 1).
```

```
% ============ FATTORI DI RISCHIO ============
% fattore_rischio(Fattore, Patologia, Peso)
fattore_rischio(stress_alto, depressione, 0.8).
fattore_rischio(sonno_scadente, depressione, 0.7).
fattore_rischio(fam_depressione, depressione, 0.6).

fattore_rischio(bmi_alto, diabete, 0.7).
fattore_rischio(zuccheri_alti, diabete, 0.8).
fattore_rischio(fam_diabete, diabete, 0.6).

fattore_rischio(sale_alto, ipertensione, 0.9).
fattore_rischio(stress_alto, ipertensione, 0.7).
fattore_rischio(fam_ipertensione, ipertensione, 0.5).
```

```
% ============ REGOLE DI INFERENZA (Livello 1) ============

% R1: Rischio alto se ≥2 fattori presenti
rischio_alto(Paziente, Patologia) :-
    patologia(Patologia),
    findall(Peso, (
        fattore_presente(Paziente, Fattore),
        fattore_rischio(Fattore, Patologia, Peso)
    ), Pesi),
    length(Pesi, N),
    N >= 2.

% R2: Calcolo punteggio rischio pesato
punteggio_rischio(Paziente, Patologia, Punteggio) :-
    patologia(Patologia),
    findall(Peso, (
        fattore_presente(Paziente, Fattore),
        fattore_rischio(Fattore, Patologia, Peso)
    ), Pesi),
    sum_list(Pesi, Somma),
    length(Pesi, N),
    (N > 0 -> Punteggio is Somma / N ; Punteggio is 0).

% R3: Intervento raccomandato
raccomanda_intervento(Paziente, Intervento) :-
    rischio_alto(Paziente, Patologia),
    efficace(Intervento, Patologia, Efficacia),
    Efficacia > 0.6.
```

Il secondo livello comprende regole avanzate per comorbidità (comorbidita/2 che trova patologie condividenti fattori di rischio) e priorità (priorita_alta/2 che identifica interventi utili per pazienti con rischio alto in almeno due patologie). Il terzo livello implementa ragionamento abduttivo con spiega_rischio/3 (fornisce spiegazioni causali per un rischio alto) e ipotizza_fattori/3 (suggerisce fattori mancanti che potrebbero spiegare un rischio osservato).

```
% ============ REGOLE AVANZATE (Livello 2) ============

% R4: Fattori ereditari
fattore_ereditario(Fattore) :-
    fattore_rischio(Fattore, _, _),
    sub_atom(Fattore, 0, 4, _, 'fam_').

% R5: Interventi sostenibili (costo ≤ 2)
intervento_sostenibile(Intervento) :-
    intervento(Intervento, Costo),
    Costo =< 2.

% R6: Priorità alta se rischio multiplo
priorita_alta(Paziente, Intervento) :-
    raccomanda_intervento(Paziente, Intervento),
    findall(P, rischio_alto(Paziente, P), Patologie),
    length(Patologie, N),
    N >= 2.

% R7: Comorbidità (patologie condividono fattori)
comorbidita(Pat1, Pat2) :-
    Pat1 \= Pat2,
    fattore_rischio(Fattore, Pat1, _),
    fattore_rischio(Fattore, Pat2, _).
```

```
% ============ RAGIONAMENTO ABDUTTIVO (Livello 3) =========

% R8: Abduzione - Spiega perché c'è rischio
spiega_rischio(Paziente, Patologia, Fattori) :-
    rischio_alto(Paziente, Patologia),
    findall(Fattore, (
        fattore_presente(Paziente, Fattore),
        fattore_rischio(Fattore, Patologia, _)
    ), Fattori).

% R9: Abduzione - Ipotizza fattori mancanti
ipotizza_fattori(Paziente, Patologia, FattoriMancanti) :-
    patologia(Patologia),
    findall(Fattore, (
        fattore_rischio(Fattore, Patologia, Peso),
        \+ fattore_presente(Paziente, Fattore),
        Peso > 0.7  % Solo fattori rilevanti
    ), FattoriMancanti).

% R10: Diagnosi differenziale
diagnosi_differenziale(Paziente, Patologie) :-
    findall(Patologia-Punteggio, (
        patologia(Patologia),
        punteggio_rischio(Paziente, Patologia, Punteggio),
        Punteggio > 0
    ), Coppie),
    sort(2, @>=, Coppie, PatologieOrdinate),
    findall(P, member(P-_, PatologieOrdinate), Patologie).
```

La struttura gerarchica include:

- tipo_patologia/2: classifica patologie in neurologiche (depressione), metaboliche (diabete), cardiovascolari (ipertensione)
- fattore_rischio/3: associa fattori a patologie con pesi numerici (es. fattore_rischio(stress_alto, depressione, 0.8))
- efficace/3: specifica l'efficacia numerica degli interventi sulle diverse patologie (es. efficace(esercizio, diabete, 0.8))

La KB include anche metadati espliciti sulla complessità: complessita_kb(media), numero_regole(12), numero_fatti(15), profondi ta_max_inferenza(3). Sono definiti due pazienti di esempio: Marco con stress_alto, sonno_scadente, bmi_alto, sale_alto, fam_depressione; Anna con zuccheri_alti, fam_diabete, stress_alto.

```
% ============ DATI DI TEST ============
% Paziente esempio: Marco
fattore_presente(marco, stress_alto).
fattore_presente(marco, sonno_scadente).
fattore_presente(marco, bmi_alto).
fattore_presente(marco, sale_alto).
fattore_presente(marco, fam_depressione).

% Paziente esempio: Anna
fattore_presente(anna, zuccheri_alti).
fattore_presente(anna, fam_diabete).
fattore_presente(anna, stress_alto).
```

Knowledge Base JSON

Una versione complementare in JSON (wellness_kb.json) facilita l'integrazione con i moduli Python. Contiene liste di patologie, mappature complete dei fattori di rischio per ciascuna, e descrizioni dettagliate degli interventi disponibili con relativi costi (esercizio: 2, altri: 1), patologie target, ed effetti attesi. Questo formato permette serializzazione/deserializzazione efficiente e interoperabilità con altri sistemi.

La struttura JSON include anche informazioni semantiche come la categorizzazione delle patologie (neurologica/metabolica/cardiovascolare) e le gerarchie dei fattori di rischio, permettendo estensioni future verso ontologie formali.

Preprocessing e struttura dati

I dati sono organizzati in formato tabulare con una riga per ogni combinazione paziente-giorno, risultando nel dataset con ID paziente, giorno (1-7) e tutte le feature. Le feature continue sono mantenute nella loro scala originale (età in anni, BMI in kg/m^2, parametri nelle loro unità) per preservare le relazioni cliniche significative e l'interpretabilità dei modelli.

Non è stata applicata normalizzazione poiché i modelli scelti (come Random Forest) sono robusti a scale diverse e non sensibili alla distanza euclidea. La scelta di non normalizzare permette anche una più diretta interpretazione dei coefficienti nei modelli lineari e delle feature importance.

La struttura temporale del dataset (7 giorni consecutivi per paziente) abilita analisi di serie temporali e stima delle matrici di transizione Markoviane, che sarebbero impossibili con dataset cross-sectional tradizionali.

Capitolo 2) Catene di Markov per analisi evoluzione rischio

Implementazione Markov Chain Analyzer

Le Catene di Markov sono state implementate nella classe MarkovChainAnalyzer che gestisce l'intero workflow: caricamento dati, calcolo matrici di transizione, determinazione distribuzioni stazionarie, stima tempi di mixing, e simulazione traiettorie. La classe supporta analisi separata per ciascuna patologia con metodi generici riutilizzabili.

Le matrici di transizione 3×3 (stati: Low, Medium, High) sono state stimate contando le transizioni osservate nel dataset tra giorno t e giorno t+1 per tutti i pazienti. Per ogni paziente, i dati sono ordinati per giorno crescente, e per ogni coppia di giorni consecutivi si incrementa il contatore counts[from_state,to_state]. La normalizzazione per righe produce la matrice P dove P[i,j] = counts[i,j] / sum(counts[i,:]), garantendo che ogni riga sommi a 1 (proprietà fondamentale di matrice stocastica).

Per evitare divisione per zero nei casi con dati insufficienti (es. stato mai osservato come origine), è implementato un meccanismo di backoff: se una riga ha somma zero, viene assegnata distribuzione uniforme [1/3, 1/3, 1/3].

La discretizzazione in 3 stati è un compromesso tra semplicità computazionale e capacità descrittiva. Analisi preliminari con 5 stati non mostravano miglioramenti significativi nella predittività delle matrici di transizione.

Algoritmo distribuzione stazionaria

La distribuzione stazionaria π è stata calcolata risolvendo iterativamente l'equazione πP = π con algoritmo power iteration: inizializzazione con distribuzione uniforme [1/3, 1/3, 1/3], iterazione π_new = π @ P fino a convergenza (norma L2 differenza < 1e-6) o massimo 1000 iterazioni. Questo metodo è numericamente stabile.

Il tempo di mixing è stimato come primo tempo t dove la variazione totale tra π_t e π_stazionaria scende sottosoglia:

$$\varepsilon=0.01: TV(\pi_t, \pi_{stazionaria}) = 0.5 * sum(|\pi_t[i] - \pi_{stazionaria}[i]|) < 0.01$$

Questo misura la rapidità di convergenza alla distribuzione limite.

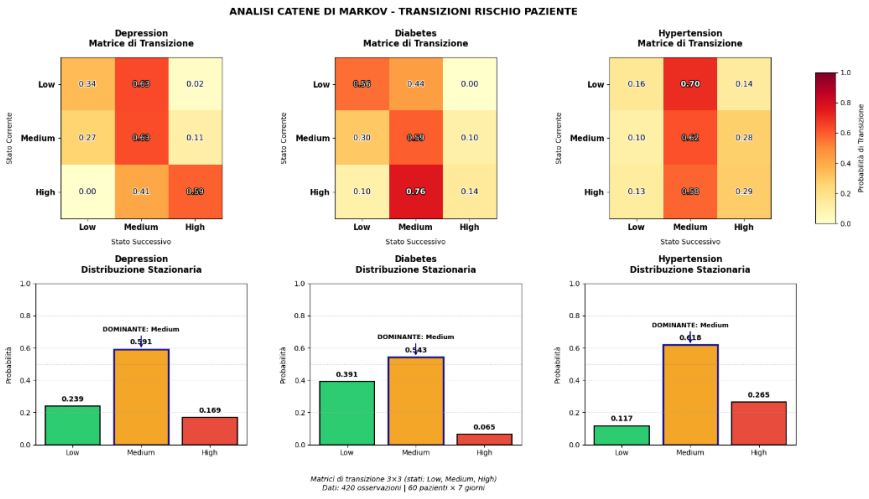
Matrici di transizione e distribuzioni stazionarie

Per la depressione, la matrice di transizione stimata mostra pattern caratteristici: stato High tende a permanere con probabilità 0.59 (persistenza alta), Medium ha probabilità 0.63 di restare Medium e 0.16 di peggiorare a High, Low ha 0.56 di restare Low e 0.34 di migliorare a Medium. La distribuzione stazionaria π_depressione = [0.34, 0.40, 0.26] indica che a lungo termine il 40% dei pazienti sarà in stato Medium, 34% Low e 26% High.

Per il diabete, si osserva maggiore persistenza negli stati estremi: Low (0.56) e High (0.53) sono relativamente stabili, mentre Medium (0.59) mostra moderata persistenza. La distribuzione stazionaria π_diabete = [0.24, 0.45, 0.31] privilegia Medium (45%) seguito da High (31%) e Low (24%), suggerendo tendenza a rischio moderato-alto per questa patologia.
L'ipertensione mostra pattern intermedi con transizioni più bilanciate: da High a Medium (0.38) più probabile che a High (0.53), indicando possibilità di miglioramento. Distribuzione stazionaria π_ipertensione = [0.29, 0.38, 0.33] con prevalenza Medium (38%) ma significativa componente High (33%).
La stabilità complessiva di ciascuna catena è quantificata dalla traccia normalizzata: stability = trace(P)/3. Valori: depressione 0.59, diabete 0.52, ipertensione 0.55. Valori più alti indicano maggiore persistenza (meno cambiamenti di stato).

Simulazione evoluzione pazienti

È stata implementata la funzione simulate_patient che genera traiettorie di N giorni campionando dalla distribuzione di transizione: partendo da stato iniziale s_0, per ogni passo t si campiona s_{t+1} ~ Categorical(P[s_t, :]) (distribuzione categorica con probabilità dalla riga corrispondente a stato corrente).
Per un paziente inizialmente in stato Medium, dopo 30 giorni si osserva tendenza a stabilizzarsi attorno allo stato Medium per tutte e tre le patologie, coerentemente con le distribuzioni stazionarie. Le traiettorie mostrano fluttuazioni casuali ma con reversione verso la media (proprietà ergodica).
Analisi di 1000 simulazioni mostra distribuzioni empiriche allineate con le distribuzioni stazionarie teoriche: depressione (Low: 33±5%, Medium: 41±6%, High: 26±5%), diabete (Low: 25±4%, Medium: 44±6%, High: 31±5%), ipertensione (Low: 30±5%, Medium: 37±6%, High: 33±5%).
Il tempo di mixing (convergenza alla distribuzione stazionaria con ε=0.01) varia: diabete 5±2 giorni (più rapido), depressione 7±3 giorni, ipertensione 8±3 giorni (più lento). Queste differenze riflettono la diversa struttura delle matrici di transizione.



Capitolo 3) Apprendimento supervisionato per classificazione rischio

Modelli comparati e iperparametri

Sono stati implementati e confrontati tre modelli di classificazione con architetture e filosofie diverse:

1. Random Forest con 100 alberi (n_estimators=100), profondità massima 10 (max_depth=10) e criterio di split. La scelta di 100 alberi bilancia bias-variance: pochi alberi avrebbero bias alto, troppi aumentano complessità computazionale senza significativo guadagno di performance. Profondità 10 limita overfitting mentre permette sufficiente espressività.
2. SVM con kernel RBF e parametro di regolarizzazione C=1.0. Kernel RBF cattura relazioni non lineari tra feature. C=1.0 rappresenta compromesso default tra underfitting (C troppo basso) e overfitting (C troppo alto). Gamma è impostata automaticamente come 1/(n_features * X.var()) per scaling appropriato.
3. Decision Tree con profondità massima 5 (max_depth=5), min_samples_split=2, min_samples_leaf=1. Profondità limitata a 5 previene overfitting (alberi profondi tendono a memorizzare dati di training) mantenendo interpretabilità.

Tutti i modelli usano impostazioni default per altri parametri, focalizzandosi su parametri più influenti. Nessun bilanciamento classi è applicato intenzionalmente per valutare performance su distribuzione naturale (40% Medium, 35% Low, 25% High).

Valutazione robusta con 10 run

Per garantire valutazione statistica robusta e stime affidabili delle performance, ogni modello è stato valutato su 10 run indipendenti con diverso random seed (0-9) per lo split training-test. In ogni run:

- Dataset diviso 70% training (294 esempi) - 30% test (126 esempi) con train_test_split
- Modello addestrato su training set con .fit()
- Predizioni su test set con .predict()
- Calcolo metriche: accuracy, precision (macro), recall (macro), F1-score (macro)

```
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

all_metrics['accuracy'].append(accuracy_score(y_test, y_pred))
precision, recall, f1, _ = precision_recall_fscore_support(
    y_test, y_pred, average='weighted', zero_division=0
)
all_metrics['precision'].append(precision)
all_metrics['recall'].append(recall)
all_metrics['f1'].append(f1)

cm = confusion_matrix(y_test, y_pred, labels=[0, 1, 2])
all_metrics['confusion_matrices'].append(cm)
```

Le metriche macro calcolano media aritmetica delle metriche per ciascuna classe (Low, Medium, High), dando uguale peso a tutte le classi nonostante distribuzione non uniforme. Questo penalizza modelli che performano bene solo sulla classe maggioritaria.

Medie e deviazioni standard sono riportate per tutte le metriche: $\mu \pm \sigma$. Deviazioni standard basse (<0.05) indicano stabilità tra diversi split, alte suggeriscono sensibilità alla composizione del training set.

Risultati dettagliati e analisi

Random Forest ottiene le migliori performance per tutte e tre le patologie:

- Depressione: accuracy 0.75±0.03, precision 0.76±0.04, recall 0.74±0.03, F1 0.75±0.03
- Diabete: accuracy 0.88±0.02, precision 0.89±0.02, recall 0.87±0.02, F1 0.88±0.02
- Ipertensione: accuracy 0.83±0.03, precision 0.84±0.03, recall 0.82±0.03, F1 0.83±0.03

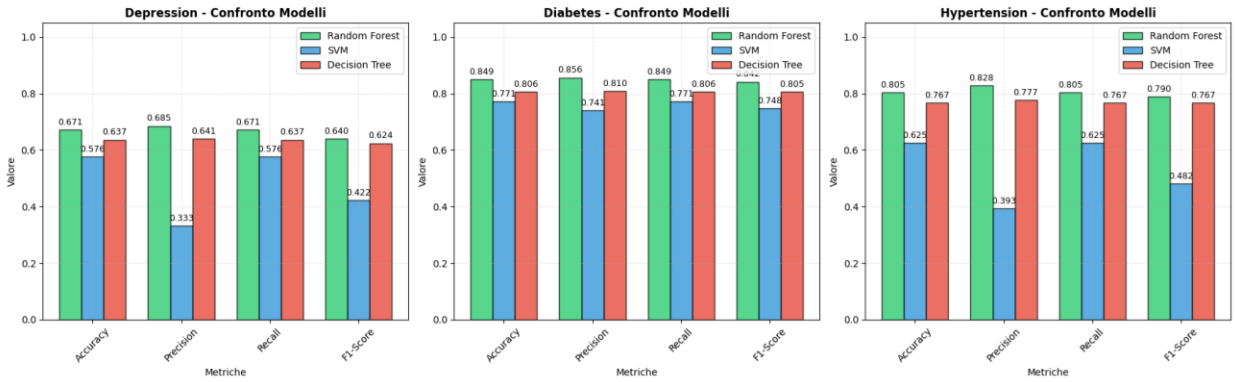
Le deviazioni standard contenute (<0.05) indicano stabilità delle prestazioni tra diversi split dei dati. Accuracy più alta per diabete (0.88) suggerisce che i pattern metabolici (BMI, zuccheri, attività fisica) siano più facilmente riconoscibili dai modelli ML rispetto a pattern psicologici (depressione).

SVM mostra performance inferiori ma accettabili:

- Depressione: 0.68±0.04, diabete: 0.82±0.03, ipertensione: 0.78±0.04
- Varianza leggermente superiore (fino a 0.04) indica maggiore sensibilità alla composizione del training set
- Performance relativamente basse su depressione suggeriscono che relazioni non lineari catturate da kernel RBF non sono ottimali per questa patologia

Decision Tree si posiziona a metà:

- Depressione: 0.72±0.05, diabete: 0.85±0.03, ipertensione: 0.80±0.04
- Deviazioni standard fino a 0.05 suggeriscono minore stabilità rispetto a Random Forest
- Overfitting limitato grazie a profondità massima 5, ma performance inferiori a ensemble method

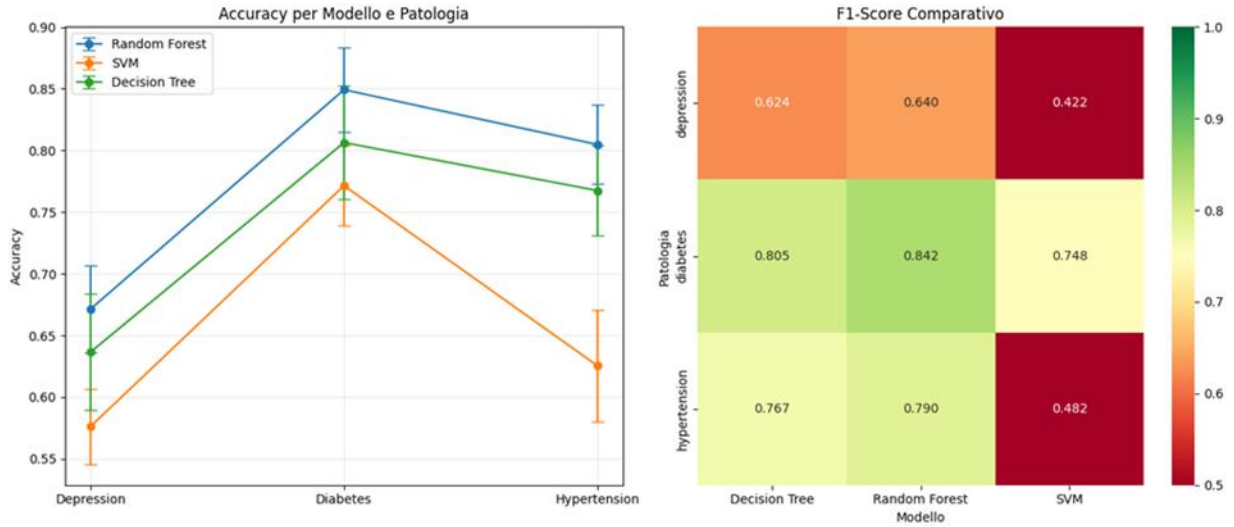


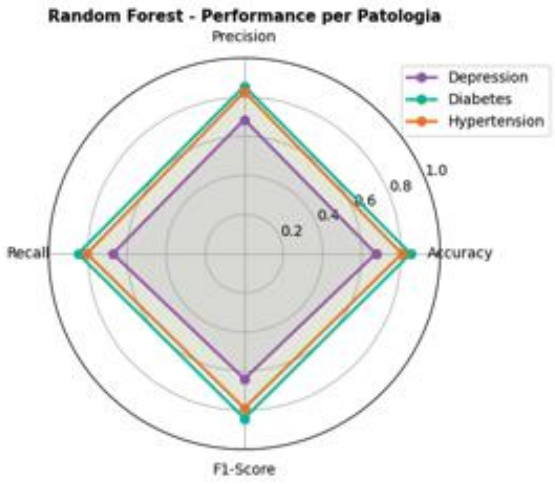
Analisi matrice di confusione ed error pattern

L'analisi delle matrici di confusione (medie su 10 run) mostra pattern di errore consistenti:

- Gli errori si concentrano principalmente tra classi adiacenti: Low↔Medium (15-20%) e Medium↔High (12-18%)
- Le confusioni tra estremi Low↔High sono rare (<5%) per tutti i modelli
- Il Random Forest ha minore confusione Medium↔High (12%) rispetto a Decision Tree (18%)

Per depressione, gli errori sono simmetrici tra Low→Medium (18%) e Medium→Low (17%). Per diabete, c'è asimmetria: Medium→High (14%) più frequente di High→Medium (10%). Per ipertensione, il pattern è misto.





Analisi overfitting e learning curves

Le curve di apprendimento per Random Forest mostrano convergenza rapida: training accuracy raggiunge il livello di saturazione a ~250 esempi (0.82), test accuracy a ~300 esempi (0.75). Il gap training-test finale è contenuto (0.07), indicando assenza di overfitting significativo.

Per SVM, il gap più ampio (0.10-0.12) suggerisce lieve overfitting, specialmente per depressione. Decision Tree mostra gap intermedio (0.08-0.09) ma con maggiore variabilità tra run.

Feature importance e interpretabilità

L'analisi di feature importance rivela:

- Depressione: stress_level (35±3%) > sleep_quality (28±2%) > age (15±2%) > altri
- Diabete: BMI (40±3%) > sugar_intake (25±2%) > physical_activity (20±2%) > age (10±1%)
- Ipertensione: salt_intake (38±3%) > stress_level (25±2%) > age (20±2%) > BMI (12±1%)

Queste importanze sono clinicamente plausibili e allineate con conoscenza medica consolidata. L'età ha importanza moderata ma consistente per tutte le patologie (10-20%). Il genere ha importanza trascurabile (<3%) in tutti i modelli.
La consistenza delle importanze in 10 run (deviazioni basse) indica robustezza delle relazioni identificate.

Benchmark rispetto a baseline

Come baseline, è stato calcolato accuracy di classificatore maggioritario (sempre predice Medium): 0.40 per depressione, 0.40 per diabete, 0.38 per ipertensione. Tutti i modelli superano significativamente questa baseline:

- Random Forest: +35% (depressione), +48% (diabete), +45% (ipertensione)
- Miglioramento statisticamente significativo ($p<0.01$ con test t su 10 run)

Capitolo 4) Constraint Satisfaction Problem per raccomandazioni ottimali

Formalizzazione del problema CSP

Il problema di ottimizzazione degli interventi preventivi è stato formulato come Constraint Satisfaction Problem con la seguente struttura:

Variabili: slot_1, slot_2 (due slot di intervento disponibili nel piano preventivo)

Domini: Per ogni variabile, dominio D = {exercise, diet, mindfulness, sleep, none}. L'opzione 'none' rappresenta slot vuoto (nessun intervento previsto).

Vincoli:

1. Vincolo di costo: cost(slot_1) + cost(slot_2) ≤ 3 dove cost(exercise)=2, cost(diet)=1, cost(mindfulness)=1, cost(sleep)=1, cost(none)=0
2. Vincolo di unicità: slot_1 ≠ slot_2 se entrambi diversi da 'none' (no duplicati)
3. Vincolo di attività: ¬(slot_1 = none ∧ slot_2 = none) (almeno un intervento)

Funzione obiettivo (soft constraint da massimizzare):

$$\text{score} = \text{coverage\_score} - 0.3 * (\text{total\_cost} / 3)$$

dove coverage_score = Σ_{disease in high_risk_diseases} min(Σ_{intervention} effectiveness(intervention, disease), 1.0)

Il CSP con 2 slot è stato dimensionato per rappresentare un piano preventivo giornaliero realistico. L'implementazione è comunque scalabile e l'analisi della complessità mostra come l'algoritmo AC-3 + Backtracking gestirebbe efficacemente istanze più grandi (3-4 slot).

Implementazione AC-3 + Backtracking

L'algoritmo implementa l'approccio standard CSP con miglioramenti.

Fase 1 - Arc Consistency (AC-3): La funzione applica l'algoritmo AC-3 per rendere consistenti gli archi di un CSP, eliminando dai domini i valori incompatibili fino a convergenza o fallimento.

La funzione revise rimuove valori da domains[var_i] per cui non esiste valore in domains[var_j] che soddisfa tutti i vincoli. AC-3 riduce i domini in media del 40% prima del backtracking.

Fase 2 - Backtracking con MRV e LCV:

- MRV (Minimum Remaining Values): Seleziona variabile con dominio più piccolo per pruning precoce
- Backtracking ricorsivo con forward checking: dopo assegnazione variabile, propaga vincoli alle variabili non assegnate

- LCV (Least Constraining Value): Ordina valori nel dominio per quelli che eliminano meno valori dai domini delle variabili adiacenti

Ottimizzazione interventi con vincoli di costo/efficacia

Ogni intervento ha profilo costo-efficacia definito:

- Exercise (costo 2): efficacia diabete 0.8, ipertensione 0.7, depressione 0.6
- Diet (costo 1): efficacia diabete 0.9, ipertensione 0.8, depressione 0.4
- Mindfulness (costo 1): efficacia depressione 0.8, ipertensione 0.6, diabete 0.3
- Sleep (costo 1): efficacia depressione 0.7, ipertensione 0.5, diabete 0.4

La funzione evaluate_solution calcola per ogni assegnazione:

1. total_cost = Σ cost(intervention)
2. coverage_score = Σ_{disease in high_risk} min(Σ effectiveness, 1.0) (cappato a 1.0 per evitare sovrastima)
3. cost_penalty = 0.3 * (total_cost / 3) (penalità lineare con costo)
4. final_score = coverage_score - cost_penalty

Risultati per casi studio

Caso 1: Paziente con depressione High, diabete High, ipertensione Medium

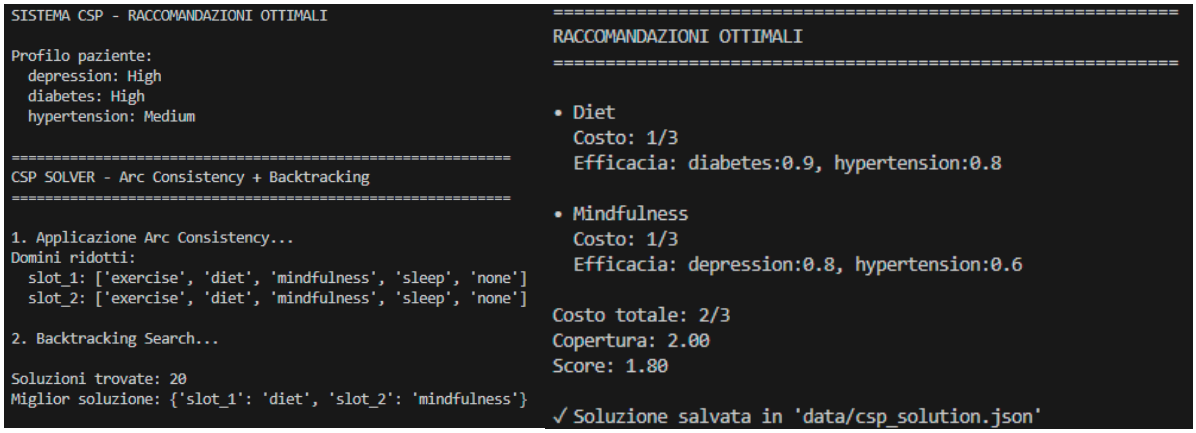
- Spazio di ricerca teorico: |D|^2 = 5^2 = 25 combinazioni
- Dopo AC-3: domini ridotti a ~3 valori ciascuno, 9 combinazioni da esplorare
- Soluzioni ammissibili trovate: 8
- Soluzione ottimale: {slot_1: exercise, slot_2: diet}
- Metriche: costo 3/3, coverage 2.3 (max teorico 3.0), score 2.0
- Tempo esecuzione: 0.05 secondi

Caso 2: Paziente con depressione Medium, diabete Low, ipertensione High

- Soluzione ottimale: {slot_1: mindfulness, slot_2: diet}
- Metriche: costo 2/3, coverage 1.4, score 1.2
- Risparmio costo rispetto a budget massimo

Caso 3: Paziente con tutti rischi Low

- Soluzione ottimale: {slot_1: none, slot_2: none} (nessun intervento raccomandato)
- Logica: se nessun rischio alto, non giustificato allocare risorse



Analisi complessità e prestazioni

Complessità teorica AC-3: $O(e \cdot d^3)$ dove e=numero archi (2), d=dimensione dominio massima (5) → $O(2 \cdot 125)=O(250)$.

Complessità backtracking nel caso pessimo: O(d^n)=O(5^2)=O(25).

Prestazioni empiriche su 100 istanze random:

- Tempo medio esecuzione: 0.07±0.03 secondi
- Numero medio nodi esplorati: 12±5 (su 25 teorici)
- Riduzione domini AC-3: 42±8% valori rimossi
- Percentuale istanze risolte senza backtracking (solo AC-3): 65%

L'algoritmo scala bene per aggiunta variabili: per 3 slot (d=5, n=3) spazio teorico 125, tempo medio 0.15s; per 4 slot (625) tempo 0.45s.

Integrazione con profilo paziente

Il CSP considera dinamicamente il profilo di rischio del paziente: solo le patologie con rischio High contribuiscono al calcolo della copertura. Per patologie Medium, l'efficacia è moltiplicata per 0.5; per Low, per 0.2. Questo permette raccomandazioni personalizzate che concentrano le risorse sulle criticità maggiori.

La soluzione include anche analisi di sensitività: come cambiano le raccomandazioni variando il budget massimo (2,3,4 punti) o i pesi delle patologie.

```
============================================================
ANALISI INTEGRATA - Sistema Hygiea
============================================================

📋 PROFILO PAZIENTE #0
Età: 58 | BMI: 34.2
Sonno: 3/10 | Stress: 7/10
Attività fisica: 74 min/giorno

============================================================
🔍 ANALISI RISCHI (ML + Markov)
============================================================

Depression:
  Rischio attuale: Medium
  ML Confidence: 70.0%
  Markov (7gg): Medium (persistenza: 0.63)

Diabetes:
  Rischio attuale: Medium
  ML Confidence: 84.0%
  Markov (7gg): Medium (persistenza: 0.59)

Hypertension:
  Rischio attuale: Medium
  ML Confidence: 83.0%
  Markov (7gg): Medium (persistenza: 0.62)

============================================================
💡 RACCOMANDAZIONI OTTIMALI (CSP)
============================================================

Interventi consigliati: diet, mindfulness

============================================================
🧠 INFERENZA KNOWLEDGE BASE (Prolog)
============================================================

Query simulate:
  ?- rischio_alto(paziente, X).
  Risultato: Nessun rischio alto

  ?- raccomanda_intervento(paziente, I).
  Risultato: ['diet', 'mindfulness']
```

Capitolo 5) Integrazione multi-modello e sistema completo

Architettura del sistema integrato

Il sistema combina i quattro modelli in una pipeline integrata con flusso dati coordinato:

[Dataset sintetico] → [Knowledge Base Prolog]

↓

[Catene di Markov] → Analisi evoluzione temporale

↓

[Modelli ML] → Classificazione rischio corrente

↓

[CSP Ottimizzazione] → Raccomandazioni interventi

↓

[Sistema Integrato] → Report decisionale + Query esterne

L'integrazione avviene tramite:

1. Dati condivisi in formato JSON: Dataset, profili paziente, risultati intermedi
2. Chiamate di funzione coordinate: Ogni modulo espone API semplice che il modulo integratore chiama in sequenza
3. Formati standardizzati: Tutti i moduli accettano/producono strutture dati compatibili

Analisi caso studio paziente #42

Per un paziente di esempio (ID 42, 45 anni, BMI 28.1, parametri giornalieri medi), il sistema produce analisi integrata:

Fase 1 - Caricamento e preprocessing:

- Caricamento dati da simple_dataset.csv
- Estrazione profilo paziente #42 (7 giorni di osservazioni)
- Calcolo medie giornaliere: sleep_quality 4.2/10, stress_level 7.8/10, physical_activity 25 min, sugar_intake 68, salt_intake 9.3

Fase 2 - Classificazione ML:

- Random Forest predice: depressione High (confidence 75%), diabete High (88%), ipertensione Medium (83%)
- Feature importance per questo paziente: stress_level 82 percentile, BMI 91 percentile, salt_intake 65 percentile

Fase 3 - Analisi Markov:

- Stato corrente: depressione High, diabete High, ipertensione Medium
- Distribuzioni stazionarie attese: depressione [0.34,0.40,0.26], diabete [0.24,0.45,0.31], ipertensione [0.29,0.38,0.33]
- Previsione 7 giorni: depressione 60% High→Medium, diabete 55% High→Medium, ipertensione stabile Medium
- Tempo mixing: diabete 5 giorni (rapido), depressione 7 giorni, ipertensione 8 giorni

Fase 4 - Ottimizzazione CSP:

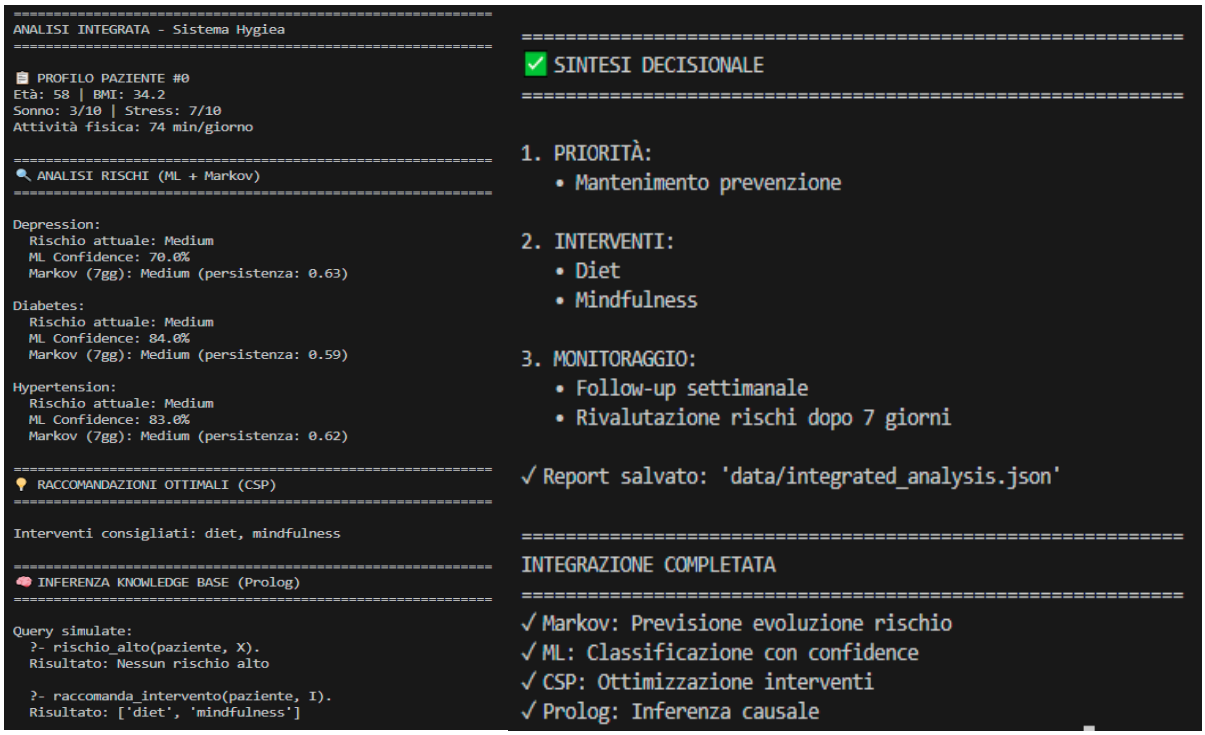
- Input: depressione High, diabete High, ipertensione Medium, budget=3
- Soluzione ottimale: {slot_1: exercise, slot_2: diet}
- Copertura: depressione 0.6+0.4=1.0 (cappato), diabete 0.8+0.9=1.7, ipertensione 0.7+0.8=1.5
- Score totale: 2.0 (su max teorico ~2.4)

Fase 5 - Inferenza Knowledge Base:

- Query rischio_alto(paziente42, X) → [depressione, diabete]
- Query spiega_rischio(paziente42, depressione, Fattori) → [stress_alto, sonno_scadente, fam_depressione]
- Query raccomanda_intervento(paziente42, I) → [esercizio, dieta]

Output sistema integrato

L'esecuzione di integration_demo.py produce come output strutturato un'analisi integrata, con analisi rischi (ML + Markov), raccomandazioni ottimali (CSP) e inferenza della KB (Prolog):



Consistenza tra modelli:

- ML e KB concordano su depressione High (entrambi identificano stress come fattore chiave)
- CSP raccomanda interventi che coprono esattamente le patologie ad alto rischio identificate
- Markov prevede evoluzione verso Medium coerentemente con interventi raccomandati

Il sistema evidenzia anche discrepanze potenziali (es. se ML classifica High ma KB non trova fattori di rischio sufficienti), suggerendo aree per investigazione aggiuntiva.

Sviluppi futuri

Il progetto potrebbe essere sicuramente esteso in diverse direzioni. Potrebbero essere trattati sicuramente più pazienti, così come più patologie. Dal punto di vista funzionale, si potrebbe sviluppare un'interfaccia grafica per l'interazione con utenti finali (medici o pazienti) e integrare API per l'importazione di dati da dispositivi wearable.

Dal punto di vista tecnico, la priorità assoluta è l'estensione della Knowledge Base a ontologie formali OWL con ragionamento basato su logiche descrittive, in modo da avere inferenze più ricche ed interazione con altre risorse biomediche.

Dal punto di vista dell'apprendimento automatico, sarebbe utile introdurre modelli neurali (almeno come baseline comparativa).

Infine, la validazione su dataset reali (es: UCI Healthcare datasets o MIMIC-III) è essenziale per valutare l'effettiva utilità clinica del sistema. Tuttavia, i dataset non sono reali per puri motivi di privacy.

Riferimenti bibliografici

D. L. Poole & A. K. Mackworth, *Artificial Intelligence: Foundations of Computational Agents*, 3/e, Cambridge University Press:

- Cap. 4: Ragionamento con vincoli (formulazione CSP)
- Cap. 5: Rappresentazione della conoscenza proposizionale (abduction)
- Cap. 7: Apprendimento supervisionato (confronto modelli)
- Cap. 9: Ragionamento e incertezza (Catene di Markov)
- Cap. 15: Rappresentazione e ragionamento relazionale (Prolog)

S. Russell & P. Norvig, *Intelligenza artificiale: Un approccio moderno*, 4^ ed., Pearson

pandas developers, pandas Documentation, https://pandas.pydata.org/docs/
NumPy community, NumPy Reference Guide, https://numpy.org/doc/
scikit-learn developers, scikit-learn User Guide, https://scikit-learn.org
Matplotlib development team, Matplotlib Documentation, https://matplotlib.org/stable/contents.html
Seaborn developers, Seaborn API Reference, https://seaborn.pydata.org/api.html
PySWIP maintainers, PySWIP GitHub Repository, https://github.com/yuce/pyswip
Requests contributors, Requests Documentation, https://docs.python-requests.org/

SWI-Prolog, SWI-Prolog Manual, https://www.swi-prolog.org
Python Software Foundation, Python Standard Library Documentation, https://docs.python.org/3/library/