UNIVERSITÀ
DEGLI STUDI DI BARI
ALDO MORO

Sistema Intelligente di Raccomandazione e Analisi del Dominio Manga

Caso di Studio di “Ingegneria della Conoscenza”

Studente: *Antonello Isabella*

Matricola: *737827*

E-Mail: *a.isabella1@studenti.uniba.it*

URL Repository: *LINK*

AA 2024-2025

Indice

Introduzione .....
Obiettivi e contesto del progetto .....
Approccio ibrido: logico + machine learning .....

Architettura del sistema .....
Struttura del progetto e strumenti utilizzati .....

Raccolta e preparazione dei dati .....
Accesso API MyAnimeList (OAuth2) .....
Dataset generati (top_manga.csv, mangalist.csv) .....
Preprocessing e costruzione dataset_ml.csv .....

Knowledge Base Prolog .....
Fatti manga/8 e lettura_utente/5 .....
Generazione da CSV .....

Motore logico simbolico .....
Regole di raccomandazione .....
Menu interattivo in Prolog .....

Ontologia OWL (modulo dimostrativo) .....
Struttura dell'ontologia manga.owl .....
Ragionamento con HermiT .....

Machine Learning supervisionato .....
Classificazione con 6 modelli .....
Target Piace .....
Grafici, confusion matrix, radar plot .....

Clustering KMeans .....
Clustering base e ottimizzato (PCA, silhouette) .....
Visualizzazione dei cluster .....

Risultati e confronto finale .....
Analisi simbolico vs statistico .....
Riflessioni sui modelli .....

Problemi e soluzioni .....
Blocco CAPTCHA MyAnimeList .....
Note tecniche sul flusso OAuth .....

Conclusioni .....
Riepilogo del lavoro .....
Estensioni future .....

Appendice .....
Codice selezionato .....
Output aggiuntivo .....
Riferimenti .....

Introduzione

Il presente elaborato descrive la progettazione e lo sviluppo di un sistema intelligente ibrido per l'analisi e la raccomandazione di contenuti nel dominio dei *manga*.
L'obiettivo principale del progetto è combinare approcci **simbolici** e **subsimbolici** dell'intelligenza artificiale, sfruttando dati reali ottenuti tramite API pubbliche, tecniche di machine learning supervisionato e non supervisionato, e un motore logico basato su Prolog.

L'integrazione tra AI simbolica (basata su conoscenza esplicita e regole logiche, come in Prolog e OWL) e AI subsimbolica (che impiega modelli statistici per l'apprendimento dai dati) consente di realizzare un sistema **interpretabile, adattabile e alimentato da conoscenza reale**.

Il progetto si distingue per la sua architettura modulare e flessibile, che unisce componenti di raccolta dati, pre-processing, rappresentazione della conoscenza, ragionamento logico e analisi predittiva. Il dominio scelto, ovvero quello dei manga, si presta particolarmente bene a un'applicazione concreta di tecniche di raccomandazione, grazie alla sua struttura semantica ricca e alla forte componente soggettiva delle preferenze utente.

Obiettivi e contesto del progetto

Il progetto nasce con l'obiettivo di progettare e realizzare un sistema intelligente capace di analizzare e raccomandare *manga* in base alle preferenze esplicite e implicite di un utente reale. In un contesto in cui la produzione di contenuti multimediali è in costante crescita, i sistemi di raccomandazione intelligenti ricoprono un ruolo chiave per migliorare l'esperienza personalizzata degli utenti, e il dominio dei manga rappresenta un ambito ideale per sperimentare tecniche avanzate di AI grazie alla ricchezza semantica delle opere e alla forte componente soggettiva nei gusti degli utenti.

Il progetto si inserisce all'interno del programma formativo del corso ICON e si propone di applicare in modo integrato:

- Tecniche simboliche, basate su rappresentazione della conoscenza, logica deduttiva e sistemi dichiarativi (es. Prolog, OWL);
- Tecniche subsimboliche, basate su apprendimento dai dati, classificazione e clustering (es. modelli supervisionati e KMeans).

L'obiettivo didattico è quello di dimostrare la complementarità tra approcci basati su conoscenza esplicita e modelli appresi dai dati, creando un sistema raccomandatore ibrido, interpretabile e aggiornato, costruito a partire da dati reali ottenuti tramite API pubbliche (MyAnimeList).

In particolare, il progetto si propone di:

- Automatizzare il recupero di dataset personalizzati tramite autenticazione OAuth2;
- Costruire una knowledge base Prolog contenente fatti estratti dai dati reali;

- Implementare un motore logico di raccomandazione basato su regole;
- Eseguire analisi supervisionate e non supervisionate con metodi di machine learning;
- Integrare un esempio di ontologia OWL a fini dimostrativi per il ragionamento semantico;
- Produrre output visuali e report esplicativi che evidenzino le caratteristiche e le prestazioni dei modelli;
- Fornire una valutazione critica tra i diversi approcci, evidenziandone limiti e potenzialità.

Questo progetto si configura quindi come una dimostrazione pratica e completa dell'integrazione tra AI simbolica e subsimbolica, fondata su strumenti reali e replicabili, in linea con le finalità accademiche e professionali del corso.

Approccio ibrido: logico + machine learning

Il progetto si fonda su un'architettura **ibrida**, che combina due paradigmi distinti ma complementari dell'intelligenza artificiale:

1. **Approccio simbolico**:
   Questo approccio si basa su **rappresentazioni esplicite della conoscenza** tramite simboli, relazioni e regole logiche. Nel progetto, l'intelligenza simbolica è realizzata attraverso:
   - Una **knowledge base Prolog** contenente fatti logici (es. manga/8, lettura_utente/5);
   - Un **motore di raccomandazione logico**, che genera suggerimenti sulla base di regole dichiarative;
   - Un'**ontologia OWL** minimale, utile per dimostrare come la conoscenza semantica possa arricchire il ragionamento automatico.

Questo approccio garantisce **trasparenza e spiegabilità** delle decisioni, e consente all'utente o sviluppatore di comprendere le motivazioni alla base di una raccomandazione.

2. **Approccio subsimbolico**:
   In contrapposizione, l'AI subsimbolica non utilizza regole esplicite, ma apprende **pattern nascosti nei dati** tramite algoritmi statistici. Nel progetto, ciò si traduce in:
   - Un modulo di **apprendimento supervisionato** per classificare i manga in base alla probabilità che piacciano all'utente, addestrato su dati reali con tecniche come AdaBoost, Random Forest, KNN e altri;
   - Un modulo di **clustering non supervisionato**, che individua gruppi di manga simili sfruttando KMeans e tecniche di riduzione dimensionale (PCA).

Questo approccio fornisce **adattabilità e potere predittivo**, permettendo al sistema di migliorare nel tempo con l'aumento dei dati.

L'integrazione di questi due approcci all'interno di un unico sistema consente di:

- Bilanciare **precisione predittiva e trasparenza logica**;
- Sviluppare un sistema di raccomandazione **più robusto e versatile**;
- Valorizzare al massimo le competenze e i contenuti affrontati nel corso.

Architettura del sistema

L'architettura del progetto è **modulare**, facilmente estendibile e suddivisa in componenti indipendenti, ciascuno responsabile di una specifica fase del flusso di lavoro. Questo approccio consente di separare nettamente la logica di raccolta dati, il pre-processing, il ragionamento simbolico e l'analisi statistica, favorendo sia la chiarezza che il riuso del codice.

Struttura del progetto e strumenti utilizzati

Il progetto è stato sviluppato seguendo un approccio modulare e separato per livelli funzionali, al fine di garantire chiarezza, manutenibilità e facilità di testing. Ogni fase del sistema è supportata da una specifica cartella, con script dedicati o file generati.

Struttura delle directory

📁 `CLUSTERING/`

Script Apprendimento Supervisionato e Non:

- `clustering_runner.py` : clustering base con PCA (k=3)
- `kmeans_improvement.py` : ottimizzazione KMeans con silhouette/elbow
- `main.py` : esecuzione unificata delle analisi
- `model_builder.py` : factory dei modelli ML
- `param_config.py` : griglia parametri per tuning
- `plot_tools.py` : funzioni di visualizzazione dei modelli
- `preprocessing.py` : pulizia e codifica del dataset per il clustering (KMeans)
- `report_utils.py` : valutazione finale modelli (AdaBoost)
- `supervised_runner.py` : pipeline apprendimento supervisionato (classificazione)

📁 `DATASET/`

Contiene i CSV generati:

- `dataset_ml.csv` : dataset unificato con campi numerici e binarizzati
- `mangalist.csv` : lista letta dall'utente
- `top_manga.csv` : 1000 manga più popolari da MyAnimeList

📁 `DOCUMENTAZIONE/`

Contiene la documentazione nel formato docx e pdf:

- `Sistema Intelligente di Raccomandazione e Analisi del Dominio Manga.docx` : documentazione nel formato docx
- `Sistema Intelligente di Raccomandazione e Analisi del Dominio Manga.pdf` : documentazione nel formato pdf

📁 KB/

Knowledge base Prolog:

- `kb_creator.py` : genera la KB Prolog
- `knowledge_base.pl` : contiene fatti `manga/8` e `lettura_utente/5`
- `system.pl` : menu interattivo in Prolog

📁 OWL/

Esempio Ontologia OWL:

- `manga.owl` : struttura semantica con classi `Manga` , `Seinen` , `AwardWinning`
- `ontology_example.py` : script Python per ragionamento OWL (Hermit)

📁 PNG/

Grafici generati:

- Accuratezze, metriche per modello, radar chart, clustering PCA, ecc.

📁 PYTHON_DATASET/

Script estrazione dati da MyAnimeList:

- `mangalist_extended.py` : versione estesa che arricchisce il dataset con mean, rank, popolarità per ogni manga
- `top_manga.py` : scarica i manga top da MAL
- `user_manga.py` : scarica la lista dell'utente da MAL

Tecnologie e strumenti utilizzati

Categoria	Tecnologie
Linguaggi	Python 3, Prolog, OWL/XML
Librerie Python	requests, pandas, scikit-learn, xgboost, matplotlib, seaborn, owlready2
Reasoner OWL	HermiT via owlready2
Ambiente logico	SWI-Prolog
Autenticazione API	OAuth 2.0 (MyAnimeList)
IDE consigliati	VS Code, PyCharm, SWI-Prolog

Raccolta e preparazione dei dati

La fase di raccolta e preparazione dei dati è fondamentale per garantire la qualità e l'efficacia delle analisi successive. In questo progetto, l'obiettivo è ottenere un dataset affidabile e strutturato, partendo da fonti reali, per alimentare sia i moduli di apprendimento automatico che quelli di ragionamento simbolico.

Accesso API MyAnimeList (OAuth2)

Per accedere ai dati utente tramite le API di **MyAnimeList (MAL)**, è necessario seguire il flusso di autorizzazione **OAuth 2.0**. Questo protocollo consente all'utente di autorizzare un'applicazione ad accedere alle proprie risorse (es. lista dei manga letti) senza esporre le credenziali.
Per poter utilizzare le API è necessario accedere al seguente link e richiedere l'ID client.

```
# --- Credenziali dell'applicazione ---
CLIENT_ID = '823135212a297d25238a81ee65b9e53b'  # ID applicazione registrata su MAL
CLIENT_SECRET = '5ce0b51e70b4df89c3bc9d9e7102755e46cb679150fc99cb4a0a95a6dd1cdbd1'  # Chiave segreta
REDIRECT_URI = 'http://localhost:8080'  # URI di redirect registrato su MAL
```

Il processo si articola in 3 fasi principali:

1. Generazione del code verifier e apertura del browser:

   Lo script genera un *code_verifier* e costruisce l'URL di autorizzazione, che viene poi aperto nel browser predefinito dell'utente:

```
# --- Genera code_verifier per PKCE ---
def generate_code_verifier(length=64):
    chars = string.ascii_letters + string.digits + "-._~"
    return ''.join(secrets.choice(chars) for _ in range(length))
```

```
# --- Passo 1: Apertura URL per autorizzazione via browser ---
def open_authorization_url(code_verifier):
    auth_url = (
        f"https://myanimelist.net/v1/oauth2/authorize?"
        f"response_type=code&client_id={CLIENT_ID}&state=1234&"
        f"redirect_uri={urllib.parse.quote(REDIRECT_URI)}&"
        f"code_challenge={code_verifier}&code_challenge_method=plain"
    )
    print("\nAprendo il browser per autorizzare...")
    webbrowser.open(auth_url)
```

   L'utente viene così reindirizzato a MyAnimeList per autorizzare l'applicazione. Dopo il consenso, MAL reindirizza al server locale con un parametro code;

2. Cattura del codice di autorizzazione tramite un server HTTP locale:

   Un piccolo server Python in ascolto su localhost:8080 cattura il codice:

```python
# --- Server HTTP per intercettare la redirect OAuth ---
class OAuthCallbackHandler(BaseHTTPRequestHandler):
    authorization_code = None

    def do_GET(self):
        parsed_path = urllib.parse.urlparse(self.path)
        params = urllib.parse.parse_qs(parsed_path.query)

        if 'code' in params:
            OAuthCallbackHandler.authorization_code = params['code'][0]
            self.send_response(200)
            self.end_headers()
            self.wfile.write(b"<h1>Autorizzazione completata! Puoi chiudere questa finestra.</h1>")
        else:
            self.send_response(400)
            self.end_headers()
            self.wfile.write(b"<h1>Errore: codice mancante!</h1>")
```

3. Scambio del codice con un access token valido:

   Dopo aver ricevuto il code, lo si invia tramite POST per ottenere l'access token:

```python
# --- Passo 2: Scambio del codice per ottenere l'access token ---
def get_access_token(auth_code, code_verifier):
    token_url = 'https://myanimelist.net/v1/oauth2/token'
    data = {
        'client_id': CLIENT_ID,
        'client_secret': CLIENT_SECRET,
        'grant_type': 'authorization_code',
        'code': auth_code,
        'redirect_uri': REDIRECT_URI,
        'code_verifier': code_verifier
    }
    print("Payload inviato:")
    print(data)
    headers = {
        'Content-Type': 'application/x-www-form-urlencoded',
        'User-Agent': 'Mozilla/5.0'
    }
    response = requests.post(token_url, data=data, headers=headers)

    if response.status_code == 200:
        print("Access Token ottenuto con successo!")
        return response.json()['access_token']
    else:
        print("Errore durante il recupero dell'access token:")
        print(response.status_code, response.text)
        return None
```

   Con un token valido, è ora possibile accedere ai dati privati dell'utente (es. lista dei manga, punteggi, ecc.).

Dataset generati (top_manga.csv, mangalist.csv)

L'interazione con le API di **MyAnimeList** ha portato alla generazione di due dataset principali, entrambi in formato CSV e archiviati nella cartella DATASET/. Questi file rappresentano la base informativa da cui si sviluppano sia la knowledge base simbolica che i modelli di apprendimento automatico.

top_manga.csv

Questo file viene prodotto dallo script *top_manga.py* e contiene una selezione dei **1000 manga più popolari**, ottenuti tramite richiesta pubblica alle API (/v2/manga/ranking). Per ciascun manga sono raccolti:

- ID numerico;
- Titolo;
- Generi (es. action, drama);
- Punteggio medio (media delle valutazioni della community);
- Rank generale;
- Indice di popolarità;
- Stato di pubblicazione (es. finished, publishing);
- Autori (nome e cognome).

Questo dataset ha una funzione **oggettiva**: fornisce una visione globale del panorama manga, utile per il confronto, l'analisi e la costruzione di raccomandazioni.

Primi cinque manga estratti dalla top 1000:

	ID	Titolo	Generi	Punteggio Medio	Rank	Popolarità	Stato	Autori
2	2	Berserk	Action, Adventure, Award Winning, Drama, Fantasy, Gore, Horror, Military, Psychological, Seinen	9.47	1	1	currently_publishing	Kentarou Miura, Studio Gaga
3	1706	JoJo no Kimyou na Bouken Part 7: Steel Ball Run	Action, Adventure, Historical, Mystery, Seinen, Shounen, Supernatural	9.32	2	23	finished	Hirohiko Araki
4	656	Vagabond	Action, Adventure, Award Winning, Historical, Samurai, Seinen	9.27	3	13	on_hiatus	Takehiko Inoue, Eiji Yoshikawa
5	13	One Piece	Action, Adventure, Fantasy, Shounen	9.22	4	4	currently_publishing	Eiichiro Oda
6	1	Monster	Adult Cast, Award Winning, Drama, Mystery, Psychological, Seinen	9.16	5	28	finished	Naoki Urasawa

Nello script *top_manga.py* sono presenti le funzioni *get_top_manga* e *save_manga_to_csv*, che servono rispettivamente a estrarre i manga e a salvarli in un file in formato CSV.

mangalist.csv

Generato tramite user_manga.py (o mangalist_extended.py), questo file rappresenta la **lista personale dell'utente autenticato**. È ottenuto accedendo all'endpoint /v2/users/{username}/mangalist, previa autenticazione OAuth2. Per ciascun manga letto, vengono salvate:

- ID e titolo;
- Generi;
- Stato di lettura (es. reading, completed, plan_to_read);
- Punteggio assegnato dall'utente (score).

Il file consente di **modellare le preferenze personali** dell'utente, utile sia per il ragionamento simbolico (Prolog) sia per l'etichettatura supervisionata (piace / non piace) nel machine learning.

Primi dieci manga estratti dell'utente:

	ID	Titolo	Generi	Punteggio	Stato
2	3	20th Century Boys	Award Winning, Drama, Historical, Mystery, Psychological, Sci-Fi, Seinen	9	completed
3	743	21st Century Boys	Award Winning, Drama, Mystery, Psychological, Sci-Fi, Seinen	0	completed
4	1224	3-gatsu no Lion	Award Winning, Childcare, Drama, Iyashikei, Seinen, Slice of Life, Strategy Game	6	on_hold
5	147337	66,666 Years: Advent of the Dark Mage	Fantasy, Reincarnation	4	dropped
6	134678	A Business Proposal	Adult Cast, Comedy, Drama, Romance, Workplace	5	completed
7	148457	A Dance of Swords in the Night	Action, Fantasy, Martial Arts	5	dropped
8	41723	A Fairytale for the Demon Lord	Action, Adventure, Fantasy, Romance	0	plan_to_read
9	158496	A Modern Man Who Got Transmigrated Into the Murim World	Action, Fantasy, Isekai, Martial Arts, Reincarnation	0	plan_to_read
10	132247	A Returner's Magic Should Be Special	Action, Fantasy, School, Time Travel	6	completed
11	50027	About Death	Psychological, Slice of Life, Supernatural	8	completed

Nello script *user_manga.py* sono presenti le funzioni *get_user_mangalist* e *save_to_csv*, che servono rispettivamente a estrarre i manga e a salvarli in un file in formato CSV.

Preprocessing e costruzione dataset_ml.csv

Il file dataset_ml.csv rappresenta la versione **pulita, integrata e numericamente codificata** dei dati provenienti da mangalist.csv e top_manga.csv, ed è il **punto di partenza per tutte le analisi di machine learning** supervisionato e non supervisionato.

La costruzione del dataset è realizzata tramite lo script mangalist_extended.py, il quale:

1. **Autentica l'utente tramite OAuth2;**
2. **Scarica la lista dei manga personali;**
3. **Arricchisce ogni voce con metadati aggiuntivi**, prelevati tramite chiamate dedicate per ogni ID manga:
    - mean: punteggio medio globale;
    - rank: posizione nella classifica;
    - popolarity: indice di popolarità nel network MyAnimeList.

Una volta ottenute le informazioni, viene avviato un processo di **pulizia e trasformazione** per produrre una struttura compatibile con gli algoritmi di apprendimento automatico:

- **Filtraggio**: vengono considerati solo i manga valutati dall'utente (score > 0);
- **Tokenizzazione dei generi**: i generi vengono convertiti in una lista di stringhe uniformate (es. "Action, Drama" → ["action", "drama"]);
- **Binarizzazione dei generi**: grazie a MultiLabelBinarizer, ogni genere diventa una colonna booleana (1 se presente, 0 altrimenti);
- **Normalizzazione numerica**: punteggio medio, rank e popolarità vengono inseriti come **variabili numeriche continue.**

Questo avviene nel file preprocessing.py per l'apprendimento non supervisionato:

```python
# --- Carica e pre-processa i dati per clustering con KMeans ---
def load_and_preprocess_kmeans(filepath='DATASET/dataset_ml.csv'):
    # Carica il dataset da CSV
    df = pd.read_csv(filepath)

    # Filtra righe con punteggio utente valido (>0 e non NaN)
    df = df[df['Punteggio_Utente'].notna() & (df['Punteggio_Utente'] > 0)]

    # Trasforma la colonna 'Generi' in liste pulite di stringhe (lowercase + underscore)
    df['Generi'] = df['Generi'].fillna('').apply(
        lambda x: [g.strip().lower().replace(' ', '_') for g in x.split(',') if g]
    )

    # Codifica multilabel in binario (una colonna per ogni genere)
    mlb = MultiLabelBinarizer()
    generi_encoded = pd.DataFrame(mlb.fit_transform(df['Generi']),
                                  columns=mlb.classes_,
                                  index=df.index)

    # Combina codifica dei generi con colonne numeriche per clustering
    X = pd.concat([generi_encoded, df[['Punteggio_Medio', 'Rank', 'Popolarita']]], axis=1).fillna(0)

    return X, df    # Ritorna i dati pronti per il clustering e il DataFrame originale
```

E nel file supervised_runner.py per l'apprendimento supervisionato:

```python
# --- Pre-processing del dataset ---
df = pd.read_csv('DATASET/dataset_ml.csv')
df = df[df['Punteggio_Utente'] > 0] # Rimuove voti nulli o 0
df['Piace'] = df['Punteggio_Utente'].apply(lambda x: 1 if x >= 7 else 0)    # Target binario
df['Generi'] = df['Generi'].fillna('').apply(lambda x: [g.strip().lower().replace(' ', '_') for g in x.split(',') if g])


mlb = MultiLabelBinarizer()
generi_encoded = pd.DataFrame(mlb.fit_transform(df['Generi']), columns=mlb.classes_, index=df.index)
X = pd.concat([generi_encoded, df[['Punteggio_Medio', 'Rank', 'Popolarita']]], axis=1).fillna(0)
y = df['Piace']
```

Il file risultante è un dataset **matriciale** con colonne che uniscono:

- Feature binarie (una per ogni genere);
- Feature numeriche (Punteggio Medio, Rank, Popolarità);
- Target opzionale (Punteggio Utente e/o classe binaria *Piace* per la classificazione).

Primi dieci manga estratti dell'utente:

	ID	Titolo	Generi	Punteggio_Utente	Stato_Utente	Punteggio_Medio	Rank	Popolarita
2	3	20th Century Boys	Award Winning, Drama, Historical, Mystery, Psychological, Sci-Fi, Seinen	9	completed	8.94	17	24
3	743	21st Century Boys	Award Winning, Drama, Mystery, Psychological, Sci-Fi, Seinen	0	completed	8.43	200	143
4	1224	3-gatsu no Lion	Award Winning, Childcare, Drama, Iyashikei, Seinen, Slice of Life, Strategy Game	6	on_hold	8.85	28	225
5	147337	66,666 Years: Advent of the Dark Mage	Fantasy, Reincarnation	4	dropped	7.31	4781	1431
6	134678	A Business Proposal	Adult Cast, Comedy, Drama, Romance, Workplace	5	completed	7.94	892	1888
7	148457	A Dance of Swords in the Night	Action, Fantasy, Martial Arts	5	dropped	7.07	8053	4986
8	41723	A Fairytale for the Demon Lord	Action, Adventure, Fantasy, Romance	0	plan_to_read	7.09	7784	2086
9	158496	A Modern Man Who Got Transmigrated Into the Murim World	Action, Fantasy, Isekai, Martial Arts, Reincarnation	0	plan_to_read	6.96	9693	7704
10	132247	A Returner's Magic Should Be Special	Action, Fantasy, School, Time Travel	6	completed	7.59	2427	344
11	50027	About Death	Psychological, Slice of Life, Supernatural	8	completed	8.05	673	1395

Questo dataset viene infine utilizzato per:

- **Classificare** i manga in base al gradimento personale (Piace / Non piace);

- **Clusterizzare** opere simili basandosi sui generi e sugli indicatori numerici;
- **Generare grafici** e valutazioni metriche dei modelli predittivi.

Knowledge Base Prolog

La **Knowledge Base (KB)** rappresenta la componente simbolica del sistema, costruita con il linguaggio logico **Prolog**, noto per la sua potenza nell'esprimere conoscenze e regole in forma dichiarativa.

Questa base di conoscenza raccoglie e organizza in forma strutturata le informazioni ottenute dai dataset generati, come ad esempio i manga letti dall'utente, i punteggi assegnati, i generi associati e altri attributi significativi.

L'obiettivo di questa sezione è:

- Rappresentare i dati in forma di **fatti logici** e interrogabili;
- Consentire al motore inferenziale di **ragionare simbolicamente** sulle preferenze dell'utente;
- Preparare le basi per l'attivazione di **regole di raccomandazione**, affini a un sistema esperto.

La KB viene generata automaticamente dallo script Python kb_creator.py, che estrae i dati da top_manga.csv e mangalist.csv, e li converte in due tipi di fatto Prolog:

- manga/8: rappresenta un manga presente nella top globale, con informazioni come titolo, generi, punteggio medio, popolarità e autori;
- lettura_utente/5: rappresenta un manga letto dall'utente, con il relativo punteggio assegnato, stato di lettura e generi.

Questa rappresentazione simbolica permette di effettuare query flessibili e interpretabili, come ad esempio:

- *"Esistono manga premiati che l'utente non ha ancora letto?";*
- *"Quali generi compaiono più frequentemente nella sua cronologia?"*.

La KB diventa quindi la base per il successivo **motore di raccomandazione logico**, che utilizza queste informazioni per proporre titoli affini ai gusti dell'utente o per esplorare generi ancora non letti.

Fatti manga/8 e lettura_utente/5

La **base di conoscenza logica** è composta da due tipi principali di fatti, che rappresentano rispettivamente:

1. **Le informazioni oggettive sui manga** (estratte da top_manga.csv);
2. **Le preferenze personali dell'utente** (estratte da mangalist.csv).

manga/8

Questo predicato descrive ogni manga pubblico presente nella classifica globale. La struttura è:

- ID: identificativo numerico del manga;
- Titolo: nome del manga (come atomo testuale);
- Generi: lista di atomi che rappresentano i generi (es. [action, fantasy]);
- Mean: punteggio medio globale (numero);
- Rank: posizione nella classifica generale;
- Popolarita: indice di popolarità su MAL;
- Stato: stato editoriale (es. finished, publishing);
- Autori: lista di nomi (come atomi).

lettura_utente/5

Questo predicato rappresenta i manga letti (o pianificati) dall'utente, con le sue valutazioni e il suo stato personale. La struttura è:

- ID: identificativo del manga;
- Titolo: titolo normalizzato;
- Stato: stato di lettura (es. reading, completed, plan_to_read);
- PunteggioUtente: valore da 1 a 10 assegnato dall'utente;
- Generi: lista dei generi associati.

Questi due insiemi di fatti costituiscono l'intera base informativa interrogabile dal motore simbolico, consentendo inferenze personalizzate, raccomandazioni, raggruppamenti per generi e altro ancora.

Generazione da CSV

La costruzione automatica della base di conoscenza logica è affidata allo script Python kb_creator.py. Questo modulo legge i file CSV generati nella fase di raccolta dati (top_manga.csv e mangalist.csv) e li converte in una serie di **fatti Prolog** scritti nel file knowledge_base.pl.

L'obiettivo è trasformare informazioni tabellari in **predicati logici**, che possano essere successivamente utilizzati per ragionamenti simbolici, query e raccomandazioni.

Input

- top_manga.csv: contiene manga della top 1000 globale;
- mangalist.csv: contiene i manga letti e valutati dall'utente.

Operazioni principali

1. **Lettura dei file CSV** tramite csv.DictReader;

2. **Pulizia dei dati**: normalizzazione dei nomi (safe_string), rimozione spazi, gestione di valori mancanti;
3. **Parsing dei campi multipli** (generi e autori come liste);
4. **Scrittura su file** in formato .pl, generando:
    - Fatti manga/8 per ogni riga di top_manga.csv;
    - Fatti lettura_utente/5 per ogni riga di mangalist.csv.

Lo script utilizza la funzione safe_string() per garantire che i nomi siano compatibili con la sintassi Prolog (niente spazi o virgolette non gestite).

Questo processo rende la knowledge base **totalmente automatizzabile e sincronizzata con i dati reali**, evitando la scrittura manuale di fatti e riducendo il rischio di errori sintattici.

Ogni riga del file top_manga.csv viene trasformata in un fatto manga/8, che contiene informazioni strutturate e interrogabili:

- ID, titolo (in formato compatibile con Prolog), lista di generi;
- Punteggio medio, rank, indice di popolarità;
- Stato di pubblicazione e autori.

Esempio:

```
manga(5394, 'kiss/hug', ['romance', '_shoujo'], 7.79, 1400, 1264, finished, ['kako_mitsuki']).
```

La lista utente, invece, produce fatti del tipo lettura_utente/5, utili per il ragionamento sulle preferenze personali:

- ID e titolo;
- Stato di lettura (es. completed, reading);
- Punteggio assegnato dall'utente;
- Generi associati all'opera.

Esempio:

```
lettura_utente(143991, 'after_god', plan_to_read, 0.0, ['action', '_fantasy']).
```

Motore logico simbolico

Il **motore logico simbolico** rappresenta il cuore del componente deduttivo del sistema. Sviluppato in **Prolog**, esso consente di sfruttare la conoscenza esplicita codificata nella base di fatti per eseguire **ragionamenti interpretabili** e **raccomandazioni personalizzate**.

Attraverso un insieme di **regole logiche** definite manualmente, il motore è in grado di:

- Identificare i **generi preferiti** dell'utente;
- Consigliare manga non letti ma **affini ai gusti personali**;
- Evidenziare **opere di qualità sottovalutate**;
- Valutare la compatibilità tra manga e lettore sulla base delle frequenze di genere.

Il motore è accessibile tramite un **menu interattivo** (system.pl) che guida l'utente tra le varie funzionalità simboliche del sistema, offrendo una modalità di esplorazione **esplicativa e controllabile** dei contenuti raccomandati.

Regole di raccomandazione

Il motore simbolico definisce una serie di regole logiche implementate in **Prolog** all'interno del file system.pl. Ogni regola sfrutta i fatti della knowledge base (manga/8 e lettura_utente/5) per generare **raccomandazioni personalizzate** o analizzare i comportamenti dell'utente.

Di seguito si descrivono le principali regole del sistema.

Il menu interattivo è implementato nel predicato menu/0, che presenta una serie di opzioni numerate all'utente. Ogni opzione corrisponde a una specifica funzionalità del sistema di raccomandazione.

Per avviare il menu, è necessario consultare i file knowledge_base.pl e system.pl, quindi digitare il comando menu. nella console Prolog:

```
1 ?- consult('knowledge_base.pl').
true.

2 ?- consult('system.pl').
true.

3 ?- menu.
```

Dopodiché apparirà il seguente menu:

```
=== SISTEMA DI RACCOMANDAZIONE MANGA ===
1. Visualizza i generi preferiti (ordinati per frequenza)
2. Consiglia 5 manga basati sui tuoi gusti piÃ¹ frequenti (random)
3. Consiglia 5 manga di qualitÃ  ma poco popolari
4. Consiglia 5 manga dalla tua lista "plan_to_read" con generi familiari
5. Consiglia 5 manga premiati compatibili con i tuoi generi preferiti
6. Consiglia 5 manga con almeno 2 generi completamente nuovi per te
7. Consiglia 5 manga che combinano generi noti e generi mai letti
8. Valuta la compatibilitÃ  di una lista di generi rispetto alle tue preferenze
9. Esci dal programma
Scelta (1-9):
```

Ecco una descrizione dettagliata di ciascuna opzione, inclusi esempi di utilizzo e output atteso:

Visualizza i generi preferiti (ordinati per frequenza)

Mostra i generi dei manga letti dall'utente, ordinati per frequenza decrescente.

```
% Calcola la frequenza di ciascun genere
frequenza_generi(Frequenze) :-
    findall(Genere, genere_letto(Genere), ListaGeneri),
    sort(ListaGeneri, GeneriUnici),
    findall(Genere-Conta,
        (member(Genere, GeneriUnici),
         aggregate_all(count, genere_letto(Genere), Conta)),
        Frequenze).
```

Raccoglie in ListaGeneri tutti i generi dei manga che l'utente ha effettivamente letto (escludendo quelli "plan_to_read" sfruttando genere_letto(GenerePulito)).

```
% Estrai i generi dei manga letti (escludendo quelli ancora da leggere)
genere_letto(GenerePulito) :-
    lettura_utente(_, _, Stato, _, Generi),
    Stato \= plan_to_read,
    member(Genere, Generi),
    normalizza_genere(Genere, GenerePulito).
```

Inoltre, rimuove i duplicati ordinando alfabeticamente.

Ed infine conta quante volte ogni genere compare tra quelli letti, e li restituisce come coppie genere-numero.

```
% Ordina i generi per frequenza (decrescente)
generi_ordinati(GeneriOrdinati) :-
    frequenza_generi(Frequenze),
    sort(2, @>=, Frequenze, GeneriOrdinati).
```

Ordina la lista Frequenze in ordine decrescente rispetto alla seconda componente (il numero), fornendo i generi ordinati per preferenza.

Esempio di output:

```
--- Generi preferiti (ordinati) ---
action-221
fantasy-189
drama-91
seinen-91
comedy-87
adventure-86
school-76
shounen-73
romance-70
martial_arts-46
reincarnation-43
award_winning-41
isekai-41
psychological-41
supernatural-41
slice_of_life-31
time_travel-30
```

Consiglia 5 manga basati sui tuoi gusti più frequenti (random)

Suggerisce **manga non ancora letti**, appartenente ai **generi preferiti** dell'utente letti almeno dieci volte, ma **scegliendolo in modo casuale** tra quelli compatibili.

```
raccomanda_random(TitoloLeggibile) :-
    generi_ordinati(Generi),
    member(Genere-Count, Generi),
    Count >= 10,
    findall(ID-Titolo, (
        manga(ID, Titolo, GeneriManga, _, _, _, _, _),
        member(Genere, GeneriManga),
        \+ lettura_utente(ID, _, _, _, _)
    ), Candidati),
    list_to_set(Candidati, Unici),
    random_permutation(Unici, Mischiati),
    member(_-TitoloGrezzo, Mischiati),
    formatta_titolo(TitoloGrezzo, TitoloLeggibile).
```

Calcola i generi letti più frequentemente dall'utente, per poi scorre uno alla volta i generi preferiti. Crea una lista di tutti i manga con quel genere **non ancora letti**, salvandone ID e titolo. Seleziona **casualmente** i manga tra quelli trovati. Rende il titolo leggibile (rimuove underscore ecc.).

Esempi di output con la stessa KB:

```
--- Manga consigliati in base ai tuoi gusti (randomizzati) ---
iryuu: team medical dragon
last game
boku no chikyuu wo mamotte
adekan
steins;gate: eigou kaiki no pandora
```

```
--- Manga consigliati in base ai tuoi gusti (randomizzati) ---
josee to tora to sakana-tachi
sword art online: progressive
mob psycho 100
adolf ni tsugu
mairimashita! iruma-kun: if episode of mafia
```

Consiglia 5 manga di qualità ma poco popolari

Trova manga non letti con voto medio >= 8 e popolarità (valore numerico alto = poco noti) superiore a 1500.

```
manga_qualita_nascosto(TitoloLeggibile) :-
    manga(ID, Titolo, _, Mean, _, Pop, _, _),
    number(Mean), Mean >= 8,
    number(Pop), Pop > 1500,
    \+ lettura_utente(ID, _, _, _, _),
    formatta_titolo(Titolo, TitoloLeggibile).
```

Ottiene punteggio medio e popolarità per ogni manga. Seleziona solo manga **ben valutati (mean ≥ 8)** e **non troppo popolari** (pop > 1500) e che non siano stati già letti dall'utente.

Esempi di output con la stessa KB:

```
--- Manga di qualitÃ  poco popolari (randomizzati) ---
zense coupling
chichi to hige-gorilla to watashi
sakamichi no apollon: bonus track
saint seiya: the lost canvas - meiou shinwa gaiden
komatta toki ni wa hoshi ni kike
```

```
--- Manga di qualitÃ  poco popolari (randomizzati) ---
ghost hunt: akumu no sumu ie
itou junji jisen kessakushuu
koi dano ai dano
josee to tora to sakana-tachi
bungou stray dogs wan!
```

Consiglia 5 manga dalla tua lista "plan_to_read" con generi familiari

Suggerisce manga presenti nella lista "plan_to_read" dell'utente che condividono almeno un genere con i manga già letti.

```
consiglia_plan_to_read(TitoloLeggibile) :-
    lettura_utente(_, Titolo, plan_to_read, _, GeneriPlan),
    findall(G,
        (lettura_utente(_, _, Stato, _, GeneriLetti),
         Stato \= plan_to_read,
         member(G, GeneriLetti)),
        ListaGeneriLetti),
    intersection(GeneriPlan, ListaGeneriLetti, Comune),
    Comune \= [],
    formatta_titolo(Titolo, TitoloLeggibile).
```

Scorre tutti i manga che l'utente ha nel suo "plan to read", creando una lista di generi già letti dall'utente, escludendo i "plan_to_read". Se ci sono generi in comune tra quelli già letti e quelli del manga nel plan-to-read, allora lo consiglia.

Esempi di output con la stessa KB:

```
--- Consigliati tra i PLAN_TO_READ (randomizzati) ---
choujin x
chichi to ko
banana fish
mushishi
jinmen
```

```
--- Consigliati tra i PLAN_TO_READ (randomizzati) ---
umi ga hashiru end roll
katabami to ougon
akatsuki no yona
saihate no paladin
mushishi
```

Consiglia 5 manga premiati compatibili con i tuoi generi preferiti

Raccomanda manga non letti che sono "award_winning" e che contengono almeno un genere tra quelli preferiti.

```
manga_premiato(TitoloLeggibile) :-
    generi_ordinati(Generi),
    member(Genere-_, Generi),
    manga(ID, Titolo, GeneriManga, _, _, _, _, _),
    member(Genere, GeneriManga),
    member(award_winning, GeneriManga),
    \+ lettura_utente(ID, _, _, _, _),
    formatta_titolo(Titolo, TitoloLeggibile).
```

Scorre i generi preferiti dell'utente, cerca manga che abbiano sia quel genere, sia il tag award_winning. Consiglia solo se l'utente non li ha già letti.

Esempi di output con la stessa KB:

```
--- Manga premiati nei tuoi generi preferiti (randomizzati) ---
maiko-san chi no makanai-san
kocchi muite! miiko
boku no hatsukoi wo kimi ni sasagu
life
kimi wa pet
```

```
--- Manga premiati nei tuoi generi preferiti (randomizzati) ---
5-toubun no hanayome
ookiku furikabutte
mystery to iu nakare
capeta
hana yori dango
```

Consiglia 5 manga con almeno 2 generi completamente nuovi per te

Suggerisce manga non letti con almeno 2 generi mai letti. Serve per esplorare novità.

```
manga_genere_nuovo(TitoloLeggibile) :-
    findall(Genere,
        (manga(_, _, Generi, _, _, _, _, _), member(Genere, Generi)),
        TuttiGeneri),
    sort(TuttiGeneri, GeneriTotali),

    findall(GenereLetto,
        (lettura_utente(_, _, Stato, _, GeneriLetti),
         Stato \= plan_to_read,
         member(GenereLetto, GeneriLetti)),
        GeneriLetti),
    sort(GeneriLetti, GeneriUtente),
    subtract(GeneriTotali, GeneriUtente, GeneriMaiLetti),

    manga(ID, Titolo, GeneriManga, _, _, _, _, _),
    intersection(GeneriManga, GeneriMaiLetti, Nuovi),
    intersection(GeneriManga, GeneriUtente, Noti),
    length(Nuovi, LN), LN >= 2,
    length(Noti, LO), LO =< 1,
    \+ lettura_utente(ID, _, _, _, _),
    formatta_titolo(Titolo, TitoloLeggibile).
```

Raccoglie tutti i generi esistenti nei manga, rimuove duplicati e confronta l'elenco con quelli già letti. Calcola la differenza: generi mai letti e restituisce manga **composti solo da generi nuovi**.

Esempi di output con la stessa KB:

```
--- Manga di un genere mai letto (randomizzati) ---
hanakoi tsurane
ashita no ousama
```

Consiglia 5 manga che combinano generi noti e generi mai letti

Trova manga che combinano almeno un genere mai letto con almeno un genere già letto (anche se visto una sola volta).

Serve per espandere i gusti restando in parte nella propria comfort zone.

```
manga_misto_generi_nuovi(TitoloLeggibile) :-
    findall(Genere,
        (manga(_, _, Generi, _, _, _, _, _), member(Genere, Generi)),
        TuttiGeneri),
    sort(TuttiGeneri, GeneriTotali),

    findall(GenereLetto,
        (lettura_utente(_, _, Stato, _, GeneriLetti),
         Stato \= plan_to_read,
         member(GenereLetto, GeneriLetti)),
        GeneriLettiRaw),
    sort(GeneriLettiRaw, GeneriUtente),

    subtract(GeneriTotali, GeneriUtente, GeneriMaiLetti),

    manga(ID, Titolo, GeneriManga, _, _, _, _, _),
    intersection(GeneriManga, GeneriUtente, ComuneLetti),
    intersection(GeneriManga, GeneriMaiLetti, ComuneNuovi),
    ComuneLetti \= [],
    ComuneNuovi \= [],
    \+ lettura_utente(ID, _, _, _, _),
    formatta_titolo(Titolo, TitoloLeggibile).
```

Molto simile alla precedente, ma il manga deve contenere **almeno un genere già letto** e **almeno uno mai letto**.

Esempi di output con lo stesso KB:

```
--- Manga che mischiano generi giÃ  letti e generi mai letti (randomizzati) ---
karasugaoka don't be shy!!
gakuen alice
aqua
toumei na ai no utsuwa
hydra
```

```
--- Manga che mischiano generi giÃ  letti e generi mai letti (randomizzati) ---
mahou shoujo madoka★magica
kodomo no omocha
toradora!
karasugaoka don't be shy!!
blue lock: episode nagi
```

Valuta la compatibilità di una lista di generi rispetto alle tue preferenze

Dato un elenco di generi, li confronta con quelli ordinati per frequenza e valuta quanto sono compatibili con i gusti dell'utente.

```
valuta_compatibilita(GeneriForniti) :-
    generi_ordinati(GeneriOrdinati),  % Prende i generi ordinati per frequenza
    length(GeneriOrdinati, TotGeneri),
    Half is TotGeneri // 2,
    Quarter is TotGeneri // 4,

    findall(Punteggio,
        (member(Genere, GeneriForniti),
         nth1(Posizione, GeneriOrdinati, Genere-_),
         ( Posizione =< Half -> Punteggio = 2    % Prima metà
         ; Posizione =< Quarter * 3 -> Punteggio = 1 % Tra 25% e 50%
         ; Punteggio = 0                          % Ultimo quarto
         )
        ),
        ListaPunteggi),

    sum_list(ListaPunteggi, Somma),
    length(GeneriForniti, NGen),
    (NGen =:= 0 -> Media = 0 ; Media is Somma / NGen),

    % Media finale valutata
    ( Media >= 1.5 ->
        writeln('Questo manga è MOLTO compatibile con i tuoi gusti!')
    ; Media >= 0.75 ->
        writeln('Questo manga è ABBASTANZA compatibile con i tuoi gusti.')
    ;
        writeln('Questo manga è POCO compatibile con i tuoi gusti.')
    ).
```

Divide i generi ordinati in 3 fasce di preferenza, assegnando punteggi 2 (molto compatibile), 1 (abbastanza) o 0 (non compatibile) in base alla posizione del genere. Calcola la media del punteggio tra i generi forniti e stampa il giudizio finale in base al valore della media.

Esempi di output con la stessa KB:

```
Inserisci i generi separati da virgola (es: action, fantasy, drama):
|: samurai, avant garde, vampire
Questo manga Ã¨ POCO compatibile con i tuoi gusti.
```

```
Inserisci i generi separati da virgola (es: action, fantasy, drama):
|: samurai, avant garde, seinen, drama
Questo manga Ã¨ ABBASTANZA compatibile con i tuoi gusti.
```

```
Inserisci i generi separati da virgola (es: action, fantasy, drama):
|: seinen, drama, workplace, sports
Questo manga Ã¨ MOLTO compatibile con i tuoi gusti!
```

Menu interattivo in Prolog

Il sistema di raccomandazione logico-simbolico dispone di un **menu testuale interattivo**, implementato in Prolog nel file system.pl, che permette all'utente di esplorare i dati e ottenere suggerimenti intelligenti in base alla propria esperienza di lettura.

Funzionamento

Il menu viene attivato con la semplice query:

```
?- menu.
```

A quel punto, viene mostrata una lista numerata di opzioni, ciascuna corrispondente a una regola di raccomandazione o analisi. Dopo aver selezionato un'opzione (inserendo il numero corrispondente), il sistema esegue la regola associata e mostra il risultato.

Gestione delle scelte

Ogni opzione è gestita da un predicato esegui_scelta(N) dove N è il numero digitato. Esempio:

```
esegui_scelta(2) :-
    writeln('--- Manga consigliati in base ai tuoi gusti (randomizzati) ---'),
    findall(Titolo, raccomanda_random(Titolo), Tutti),
    list_to_set(Tutti, Unici),
    random_permutation(Unici, Mischiati),
    primi_n(5, Mischiati, Top5),
    stampa_lista(Top5), nl,
    menu.
```

Questo codice:

- Richiama la regola raccomanda_random/1;
- Estrae i primi 5 risultati casuali;
- Li stampa a video;
- Richiama nuovamente il menu.

Ontologia OWL (modulo dimostrativo)

In questa sezione si presenta un **esempio dimostrativo** di rappresentazione della conoscenza tramite **ontologia OWL**, con l'obiettivo di **arricchire il progetto con un approccio semantico simbolico**.
L'uso dell'ontologia **non è parte integrante del sistema di raccomandazione**, ma è stato introdotto **a scopo espositivo e completamento formativo**, per dimostrare l'utilizzo di tecnologie legate al **Semantic Web** e alla **logica descrittiva (Description Logics)**.

L'ontologia definisce le seguenti entità nel dominio dei manga:

- la classe Manga;
- il genere Seinen come sottoclasse o istanza di un concetto generico;
- la proprietà hasGenre;
- la proprietà hasAward;
- e l'individuo Berserk, classificato come AwardWinning.

Il ragionamento è stato implementato tramite **Owlready2** in Python e l'invocazione del **reasoner Hermit**, che consente di inferire automaticamente relazioni semantiche.

Struttura dell'ontologia manga.owl

L'ontologia manga.owl è un **modulo dimostrativo** sviluppato per rappresentare in modo simbolico alcune proprietà semantiche del dominio dei manga. È modellata secondo i principi di OWL (Web Ontology Language) e rappresenta una **base concettuale minimale**, pensata per illustrare l'integrazione tra ragionamento logico e conoscenza esplicita.

Classi principali

- **Manga**: classe generica che rappresenta un'opera manga.
- **Seinen**: genere narrativo rivolto a un pubblico adulto, modellato come concetto (e.g. può essere trattato come individuo o sottoclasse).
- **AwardWinning**: concetto che indica manga premiati, utilizzato come oggetto della proprietà hasAward.

Proprietà (ObjectProperty)

- **hasGenre**: collega un'istanza di Manga a uno o più generi (es. Seinen).
- **hasAward**: collega un'istanza di Manga a un concetto premiale (es. AwardWinning).

Esempio nel file manga.owl:

```
<!-- Istanza (individuo) della classe Manga -->
<owl:NamedIndividual rdf:about="#Berserk">
  <rdf:type rdf:resource="#Manga"/> <!-- Berserk è un manga -->
  <hasGenre rdf:resource="#Seinen"/> <!-- Genere: Seinen -->
  <hasAward rdf:resource="#AwardWinning"/> <!-- Ha vinto un premio -->
</owl:NamedIndividual>
```

Esempio output del codice:

```
=== Classi disponibili ===
manga.Manga
manga.Seinen
manga.AwardWinning
```

Ragionamento con HermiT

Il ragionamento logico sulle ontologie è stato implementato tramite **HermiT**, un motore compatibile con OWL 2, integrato in Python grazie alla libreria Owlready2.
L'obiettivo è mostrare un esempio di **deduzione automatica di conoscenza** partendo da relazioni dichiarative esplicite.

Codice per estrarre i manga premiati:

```
# --- Stampa dei manga che hanno ricevuto un premio ---
print("\n=== Manga premiati ===")
for manga in onto.Manga.instances():    # Cicla tutte le istanze della classe Manga
    if onto.hasAward in manga.get_properties(): # Controlla se hanno la proprietà hasAward
        for prop in manga.get_properties():
            for value in prop[manga]:   # Per ogni valore della proprietà
                if "AwardWinning" in str(value):    # Se la stringa contiene AwardWinning
                    print(manga.name)   # Stampa il nome del manga premiato
```

Esempio output del codice:

```
=== Manga premiati ===
Berserk
```

L'utilizzo di Owlready2 in combinazione con il reasoner **HermiT** consente di eseguire **inferenze logiche automatiche** a partire da un'ontologia espressa in OWL. Questo permette di **derivare conoscenza implicita**, rendendo più potente e flessibile il modello simbolico.

Nel contesto del progetto, dopo aver caricato l'ontologia manga.owl e attivato il reasoner, si può ottenere un **output inferenziale** mediante semplici interrogazioni sugli individui della classe Manga.