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

1 Introduzione

Questo elaborato presenta 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 è combinare approcci **simbolici** e **subsimbolici** dell'intelligenza artificiale, integrando:

- Dati reali provenienti da API pubbliche (MyAnimeList);
- Tecniche di apprendimento automatico supervisionato e non supervisionato;
- Un motore logico basato su regole esplicite, implementato in **Prolog**;
- Un esempio semantico tramite **ontologia OWL**.

Questa integrazione permette di costruire un sistema in grado di fornire raccomandazioni motivate e adattabili all'utente.

1.1 Obiettivi e contesto del progetto

Il progetto nasce con l'obiettivo di progettare un sistema intelligente capace di analizzare e raccomandare manga sulla base delle **preferenze esplicite** (es. punteggio dato dall'utente) e **implicite** (es. genere, popolarità, pattern nei dati).
Gli obiettivi concreti includono:

- Automatizzare il recupero di dataset personalizzati tramite OAuth2;
- Generare una knowledge base Prolog con fatti strutturati da dati reali;
- Implementare un motore logico per raccomandazioni basate su regole;
- Addestrare modelli di classificazione supervisionata per stimare la probabilità di gradimento;
- Eseguire clustering non supervisionato per individuare gruppi tematici di manga;
- Dimostrare l'uso di un'ontologia OWL a supporto del ragionamento semantico;
- Produrre visualizzazioni chiare e metriche comparative tra approcci;
- Offrire una valutazione critica sulle potenzialità e i limiti di ciascun paradigma.

1.2 Approccio ibrido

Il sistema si basa su un'**architettura ibrida**, che fonde due paradigmi fondamentali dell'IA:

Approccio simbolico

Utilizza rappresentazioni esplicite della conoscenza (simboli, regole, fatti). È implementato tramite:

- Una knowledge base Prolog (manga/8, lettura_utente/5);
- Un motore deduttivo per raccomandazioni basate su regole logiche;
- Una mini-ontologia OWL che supporta inferenze semantiche, inserita come esempio.

Tale approccio garantisce **trasparenza, tracciabilità e spiegabilità** delle decisioni.

Approccio subsimbolico

Apprende pattern dai dati attraverso algoritmi statistici. È implementato con:

- Un modulo di classificazione (supervisionato) che stima il gradimento dell'utente, tramite modelli come Random Forest, AdaBoost, KNN, XGBoost;
- Un modulo di clustering (non supervisionato) per la scoperta di gruppi coerenti, con tecniche come KMeans, PCA, Gaussian Mixture e Agglomerative Clustering.

Questo approccio consente **adattabilità** e capacità predittiva.

Integrazione dei due approcci

La loro combinazione permette di:

- Equilibrare **trasparenza logica** e **accuratezza predittiva**;
- Creare un sistema raccomandatore robusto, flessibile e didatticamente significativo;
- Dimostrare concretamente i benefici dell'IA ibrida su un dominio reale e motivante.

2 Architettura del sistema

L'architettura del progetto è **modulare, estendibile** e suddivisa in componenti indipendenti, ciascuno responsabile di una fase specifica del flusso di lavoro.
Questa separazione consente di isolare chiaramente le attività di raccolta dati, pre-processing, ragionamento simbolico e analisi statistica.

2.1 Struttura del progetto e strumenti utilizzati

Il progetto è stato organizzato secondo un approccio a livelli funzionali. Ogni componente ha una propria directory dedicata:

📁 **APPRENDIMENTO/**

Script Apprendimento Supervisionato e Non:

- `clustering_runner.py` : KMeans base con PCA
- `kmeans_improvement.py` : clustering migliorato (K ottimale, GMM, Agglomerative)
- `main.py` : esegue tutto il flusso ML (classificazione + clustering)
- `model_builder.py` : factory dei modelli ML
- `param_config.py` : iperparametri per ogni modello
- `plot_tools.py` : radar, heatmap, bar chart, ecc.
- `preprocessing.py` : feature engineering, encoding, delta score
- `report_utils.py` : AdaBoost finale e confusion matrix
- `supervised_runner.py` : classificazione con cross-validation

📁 **DATASET/**

Contiene i CSV generati:

- `dataset_ml.csv` : dataset finale per ML
- `mangalist.csv` : dataset finale per KB
- `top_manga.csv` : top manga da MAL

📁 **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 knowledge_base.pl
- `knowledge_base.pl` : fatti `manga/8` e `lettura_utente/5`
- `system.pl` : regole di raccomandazione Prolog + menu

📁 OWL/

Esempio Ontologia OWL:

- `manga.owl` : classi Manga, AwardWinning, ecc.
- `ontology_example.py` : inferenza OWL via owlready2 + HermiT

📁 PNG/

Grafici generati:

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

📁 PYTHON_DATASET/

Script estrazione dati da MyAnimeList:

- `mangalist_extended.py` : versione arricchita (mean, rank, popolarità)
- `top_manga.py` : classifica top 1000 da MAL
- `user_manga.py` : lista manga utente semplice

Tool, librerie e ambienti utilizzati

Categoria	**Tecnologie impiegate**
Linguaggi	Python 3.10+, Prolog (SWI-Prolog), OWL/XML
Machine Learning	scikit-learn, XGBoost
Analisi dati	pandas, numpy
Visualizzazione	matplotlib, seaborn
Clustering	KMeans, PCA, GMM, Agglomerative (tutti da scikit-learn)
Ontologie OWL	owlready2 (Python), HermiT (reasoner Java)
API / HTTP	requests (OAuth2 PKCE + accesso MyAnimeList API)
Ambienti di sviluppo	Visual Studio Code, PyCharm, SWI-Prolog
Fonte dati	MyAnimeList API – accesso autenticato con OAuth2 e token exchange

3 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.
L'obiettivo del progetto è ottenere un dataset **affidabile, personalizzato e strutturato**, partendo da fonti reali, per alimentare sia i moduli di apprendimento automatico che quelli simbolici.

3.1 Accesso API MyAnimeList

L'accesso ai dati personali dell'utente avviene tramite il protocollo **OAuth 2.0 con PKCE**, supportato dalle API di MyAnimeList (MAL). Questo meccanismo consente di ottenere dati **aggiornati** senza esporre le credenziali dell'utente.

```
# Credenziali dell'applicazione
CLIENT_ID = '823135212a297d25238a81ee65b9e53b'  # ID dell'applicazione registrata su MAL
CLIENT_SECRET = '5ce0b51e70b4df89c3bc9d9e7102755e46cb679150fc99cb4a0a95a6dd1cdbd1'  # Secret key privata (tecnicamente da non condividere pubblicamente)
REDIRECT_URI = 'http://localhost:8080'  # URL dove ricevere la risposta OAuth
```

Il processo si compone di tre fasi:

1. **Autenticazione:**
   Lo script genera un **code_verifier**, costruisce l'URL di autorizzazione e apre automaticamente il browser per permettere all'utente di concedere i permessi;

```
# Crea un codice sicuro alfanumerico usato nel flusso PKCE per proteggere lo scambio del token
def generate_code_verifier(length=64):
    chars = string.ascii_letters + string.digits + "-._~"
    return ''.join(secrets.choice(chars) for _ in range(length))
# Costruisce l'URL di autorizzazione e lo apre nel browser dell'utente
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)
```

2. **Cattura del codice:**
   Un server HTTP locale (su localhost:8080) riceve il parametro code da MAL dopo l'autorizzazione;

```
# Server locale per catturare il codice di autorizzazione, quando MyAnimeList reindirizza al browser, questo handler intercetta la richiesta e salva il codice
class OAuthCallbackHandler(BaseHTTPRequestHandler):
    authorization_code = None   # Valore condiviso per salvare il codice

    def do_GET(self):
        # Analizza l'URL della richiesta ricevuta dopo il login su MAL
        parsed_path = urllib.parse.urlparse(self.path)
        params = urllib.parse.parse_qs(parsed_path.query)

        # Estrazione del parametro 'code'
        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:**
   Lo script invia una richiesta POST con il codice ricevuto per ottenere un **access_token**, necessario per accedere agli endpoint utente (es. lista manga).

```python
# Scambio del codice per ottenere un token di accesso, fa una richiesta POST per ottenere l'access token da MyAnimeList dopo l'autenticazione dell'utente
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
    }
    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
```

3.2 Dataset generati

Durante l'autenticazione e le chiamate API, vengono generati due dataset principali in formato CSV:

top_manga.csv

Generato dallo script top_manga.py, contiene una classifica dei manga più popolari su MyAnimeList, ottenuta interrogando l'endpoint /v2/manga/ranking.

Ogni riga include:

- ID, Titolo, Generi;
- Punteggio medio, Rank, Popolarità;
- Stato (es. finished);
- Autori (nome e cognome).

Questo dataset fornisce una visione oggettiva del panorama manga, utile per confronti, filtri e analisi non personalizzate.

Primi cinque manga estratti dalla top mille:

	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

mangalist.csv

Generato da user_manga.py, contiene la lista personale dei manga dell'utente, ottenuta dall'endpoint /v2/users/{username}/mangalist.

Per ogni manga letto:

- ID, Titolo, Generi;
- Stato di lettura;
- Punteggio dato dall'utente.

Questo file è pensato come base per l'analisi delle preferenze utente. Tuttavia, **non viene usato direttamente per il machine learning**, ma come alternativa semplificata.

Primi cinque manga estratti dell'utente (MAL 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

3.3 Preprocessing e costruzione

Il dataset finale, usato per apprendimento automatico, è costruito esclusivamente dallo script mangalist_extended.py.
La costruzione del dataset è gestita dallo script **mangalist_extended.py**, che:

- Effettua l'autenticazione via OAuth2;
- Estrazione lista manga dell'utente;
- Enrichment di ciascun manga con mean, rank e popularity.

Vengono esclusi i manga nello stato plan_to_read, poiché non ancora valutati dall'utente.
Dopo l'estrazione, i dati vengono **puliti** e **trasformati** per risultare compatibili con gli algoritmi di **apprendimento automatico**:

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

Questa fase avviene nel file **supervised_runner.py**, dedicato all'**apprendimento supervisionato:**

```
# Pre-processing del dataset
df = pd.read_csv('DATASET/dataset_ml.csv')
df = df[df['Punteggio_Utente'] > 0] # Elimina righe con voti assenti o nulli

# Etichetta binaria: piace (1) se punteggio utente ≥ 7
df['Piace'] = df['Punteggio_Utente'].apply(lambda x: 1 if x >= 7 else 0)

# Pulizia e codifica dei generi
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)

# Costruzione matrice X (feature) e vettore y (target)
X = pd.concat([generi_encoded, df[['Punteggio_Medio', 'Rank', 'Popolarita']]], axis=1).fillna(0)
y = df['Piace']
```

E nel file **preprocessing.py**, dedicato alla fase di **apprendimento non supervisionato**:

```
# 1. Caricamento del dataset da file CSV
df = pd.read_csv(filepath)

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

# 3. Pulisce e standardizza la colonna 'Generi':
#    - divide per virgola
#    - rimuove spazi e trasforma in lowercase con underscore
df['Generi'] = df['Generi'].fillna('').apply(lambda x: [g.strip().lower().replace(' ', '_') for g in x.split(',') if g])

# 4. Codifica multilabel dei generi in one-hot encoding
mlb = MultiLabelBinarizer()
generi_encoded = pd.DataFrame(mlb.fit_transform(df['Generi']),columns=mlb.classes_,index=df.index)
```

Il file risultante è un **dataset matriciale**, in cui le **colonne** combinano:

- **Feature binarie,** una per ogni genere;
- **Feature numeriche**, come Punteggio Medio, Rank e Popolarità;
- Un **target opzionale,** costituito dal Punteggio Utente e/o dalla **classe binaria** (Piace/Non Piace) per la **classificazione.**

Primi cinque manga estratti dell'utente (MAL Utente):

```
ID,Titolo,Generi,Punteggio_Utente,Stato_Utente,Punteggio_Medio,Rank,Popolarita
43783,"""Aishiteru"", Uso dakedo.","Romance, Shoujo",0,on_hold,6.26,19129,7539
32081,"""Aoi"" Hikaru ga Chikyuu ni Ita Koro.....","School, Shounen, Supernatural",6,completed,7.22,5899,3929
36037,"""Bungaku Shoujo"" to Ue Kawaku Ghost","Drama, Psychological, Romance, Shounen",0,on_hold,6.89,11022,13132
62841,"""Fushigi"" Toriatsukaimasu: Tsukumodou Kottouten","Fantasy, Mystery",0,reading,7.71,1724,10212
682,"""Kare"" First Love","Drama, Romance, School, Shoujo, Slice of Life",6,completed,7.51,2958,1243
```

Questo **dataset** viene infine utilizzato per:

- **Clusterizzare** opere simili, basandosi su generi e indicatori numerici;
- Generare **grafici** e **valutazioni metriche** dei **modelli predittivi;**
- **Classificare** i manga in base al **gradimento personale** (Piace/Non piace).

4 Knowledge Base

La **Knowledge Base (KB)** rappresenta la componente simbolica del sistema, costruita in linguaggio logico **Prolog**, adatto alla definizione di regole e fatti logici interrogabili.
Essa raccoglie e struttura le informazioni derivanti dai dataset reali, in particolare:

- Manga presenti nella top globale;
- Manga letti dall'utente, con punteggio e stato.

La KB ha tre finalità principali:

- Rappresentare la conoscenza sotto forma di **fatti logici interrogabili**;
- Consentire **inferenze simboliche** sulle preferenze dell'utente;
- Costituire la base per un **motore esperto di raccomandazione**, sviluppato nel capitolo successivo.

La KB viene generata automaticamente dallo script **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**;
- **lettura_utente/5**.

Questa rappresentazione simbolica consente di eseguire **query flessibili e interpretabili**, come:

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

La KB diventa così la **base logica** per il **motore di raccomandazione simbolico**, che utilizza tali informazioni per:

- Proporre **titoli affini ai gusti dell'utente;**
- Esplorare **generi non ancora letti**, ampliando l'esperienza di lettura;
- E tanto altro.

4.1 Fatti

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

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

manga/8

Questo **predicato** descrive ciascun manga presente nella **classifica globale**. La struttura è la seguente:

- **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;**
- **Popolarità**: 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 **proprio stato di lettura**. 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, altrimenti 0 se l'utente non ha fornito un punteggio o non ha ancora letto il manga;
- **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 tanto altro ancora.

4.2 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**.

Input

- **top_manga.csv**: contiene i manga della **top mille 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 con **safe_string()**, rimozione spazi, gestione di **valori mancanti;**
3. **Parsing dei campi multipli**: trasformazione di **generi** e **autori** in **liste;**
4. Scrittura di ogni fatto in knowledge_base.pl.

Lo script utilizza la funzione **safe_string()** per garantire la compatibilità sintattica con **Prolog**, evitando problemi dovuti a **spazi**, **caratteri speciali** o **virgolette**.

Struttura dei fatti

Ogni riga di **top_manga.csv** viene trasformata in un fatto **manga/8**, esempio:

```
manga(145863, 'sayonara_eri', ['drama', '_shounen'], 8.64, 78, 90, finished, ['tatsuki_fujimoto']).
```

Ogni riga di **mangalist.csv** produce invece un fatto **lettura_utente/5**, esempio:

```
lettura_utente(93350, 'ayeshah\'s_secret', completed, 7.0, ['drama']).
```

5 Motore logico

Sviluppato in **Prolog**, consente di sfruttare la **conoscenza esplicita** codificata nella base dei fatti per eseguire ragionamenti interpretabili e generare raccomandazioni personalizzate.

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

- Identificare i generi preferiti dall'utente;
- Suggerire manga non letti ma affini ai gusti personali;
- Valutare la compatibilità tra manga e lettore in base alla frequenza dei generi letti;
- Eseguire diverse forme di raccomandazione simbolica, come la scoperta di generi nuovi, contenuti premiati o "perle nascoste".

Il motore è accessibile tramite un **menu interattivo**, che guida l'utente tra le funzionalità simboliche del sistema, offrendo un'esperienza **esplicativa, guidata e controllabile** nell'esplorazione dei contenuti suggeriti.

5.1 Regole di raccomandazione

Il motore simbolico, implementato in **Prolog** nel file **system.pl**, definisce un insieme di **regole logiche** che operano sui fatti presenti nella **knowledge base** per generare **raccomandazioni personalizzate** e analizzare le **abitudini dell'utente**. L'interazione avviene tramite il predicato **menu/0**, che offre

un'interfaccia testuale con otto opzioni numerate, ognuna corrispondente a una specifica **funzionalità del sistema**.

Di seguito si riportano le principali regole implementate.

1 - Visualizza i generi preferiti (ordinati per frequenza)

Questa funzionalità mostra i generi dei manga effettivamente letti dall'utente, ordinati per frequenza decrescente, rappresentando così i gusti principali del lettore.

```
generi_ordinati(GeneriOrdinati) :-
    frequenza_generi(Frequenze),
    sort(2, @>=, Frequenze, GeneriOrdinati).
```

Il processo si articola in più fasi:

- Vengono estratti tutti i generi associati ai manga letti (escludendo quelli con stato plan_to_read) tramite il predicato genere_letto(GenerePulito), che applica anche una normalizzazione per uniformare i nomi dei generi.

```
genere_letto(GenerePulito) :-
    lettura_utente(_, _, Stato, _, Generi),
    Stato \= plan_to_read,
    member(Genere, Generi),
    normalizza_genere(Genere, GenerePulito).

normalizza_genere(Genere, GenerePulito) :-
    atom_chars(Genere, ['_'|Rest]) -> atom_chars(GenerePulito, Rest) ;
    GenerePulito = Genere.
```

- I generi estratti vengono deduplicati e ordinati alfabeticamente.
- Per ciascun genere, viene calcolata la frequenza con cui appare nei manga letti, producendo una lista nella forma genere-numero.

```
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).
```

- Infine, la lista viene ordinata in ordine decrescente sulla base della frequenza, tramite il predicato generi_ordinati/1.

Questo output consente di identificare i generi più apprezzati e utilizzarli come base per le successive raccomandazioni.

Esempio di output (la lista stampata è molto più lunga):

```
      Generi preferiti (ordinati)
action-226
fantasy-193
drama-93
seinen-92
adventure-88
comedy-88
school-76
shounen-73
romance-70
martial_arts-48
```

2 - Consiglia 5 manga basati sui gusti più frequenti

Questa funzionalità suggerisce manga non ancora letti, appartenenti ai generi preferiti dell'utente, ossia quelli letti almeno dieci volte. I manga sono selezionati in modo casuale tra quelli compatibili.

```
raccomanda_random(Output) :-
    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(ID-Titolo, Mischiati),
    manga(ID, _, _, _, _, _, Stato, Autori),
    formatta_output_nome(Titolo, Autori, Stato, Output).
```

Il processo si sviluppa come segue:

- I generi vengono ordinati per frequenza tramite generi_ordinati/1 e filtrati per selezionare solo quelli con almeno dieci letture (Count >= 10);
- Per ciascuno di questi generi, si costruisce una lista di manga che contengono quel genere e che non sono ancora stati letti (\+ lettura_utente(...));
- La lista complessiva di candidati viene deduplicata e mescolata casualmente (random_permutation/2);
- Infine, vengono selezionati casualmente 5 manga da proporre all'utente.

Questa raccomandazione si basa sui gusti più dominanti, mantenendo però una componente casuale che permette la scoperta di titoli potenzialmente nuovi ma in linea con le preferenze.

Esempio di output:

```
      Manga consigliati in base ai tuoi gusti (randomizzati)
NOME MANGA: amayo no tsuki - AUTORE/I:  kuzushiro - STATO: currently_publishing
NOME MANGA: pocket monsters special - AUTORE/I: hidenori kusaka,  satoshi yamamoto,  mato - STATO: currently_publishing
NOME MANGA: ousama ranking - AUTORE/I: sousuke tooka - STATO: currently_publishing
NOME MANGA: h2 - AUTORE/I: mitsuru adachi - STATO: finished
NOME MANGA: hoshifuru oukoku no nina - AUTORE/I:  rikachi - STATO: currently_publishing
```

3 - Consiglia 5 manga di qualità ma poco popolari basati sui gusti più frequenti

Suggerisce manga non ancora letti dall'utente che soddisfano due condizioni: un punteggio medio elevato (≥ 8) e una popolarità relativamente bassa (indicata da un valore numerico > 1500).

Il processo di selezione si articola così:

- Vengono considerati i generi letti almeno dieci volte, come indicatore dei gusti consolidati dell'utente.
- Per ciascun genere, si cercano manga che:
    - abbiano una media voto (Mean) maggiore o uguale a 8;
    - abbiano una popolarità (Pop) superiore a 1500, ovvero siano meno noti;
    - non siano già stati letti (\+ lettura_utente(...)).
- I risultati vengono deduplicati, mescolati casualmente, e tra questi vengono selezionati i titoli da raccomandare.

```
manga_qualita_nascosto(Output) :-
    generi_ordinati(Generi),
    member(Genere-Count, Generi), % basta un genere in comune
    Count >= 10,
    findall(ID-Titolo,(manga(ID, Titolo, GeneriManga, Mean, _, Pop, _, _),number(Mean), Mean >= 8,number(Pop), Pop > 1500,member(Genere, GeneriManga),\+ lettura_utente(ID, _, _, _, _)),Candidati),
    list_to_set(Candidati, Unici),
    random_permutation(Unici, Mischiati),
    member(ID-Titolo, Mischiati),
    manga(ID, _, _, _, _, _, Stato, Autori),
    formatta_output_nome(Titolo, Autori, Stato, Output).
```

Esempio di output:

```
      Manga di qualità poco popolari (randomizzati)
NOME MANGA: rojica to rakkasei - AUTORE/I:  kinome - STATO: finished
NOME MANGA: love bullet - AUTORE/I:  inee - STATO: currently_publishing
NOME MANGA: amayo no tsuki - AUTORE/I:  kuzushiro - STATO: currently_publishing
NOME MANGA: natsu e no tunnel, sayonara no deguchi: gunjou - AUTORE/I:  koudon,  mei hachimoku - STATO: finished
NOME MANGA: jibaku shounen hanako-kun 0 - AUTORE/I: iro aida - STATO: finished
```

4 - Consiglia 5 manga dalla tua lista "plan_to_read" basati sui gusti più frequenti

Suggerisce titoli già presenti nella lista "plan_to_read" dell'utente, che risultano particolarmente compatibili ai suoi gusti.

Il sistema opera nel seguente modo:

- I generi letti almeno dieci volte vengono identificati come “generi forti”.
- Per ogni manga nella lista plan_to_read, il sistema confronta i suoi generi con i generi forti dell’utente.
- Se almeno il 50% dei generi del manga coincide con i generi forti, il titolo viene considerato compatibile e proposto come raccomandazione.

```
consiglia_plan_to_read(Output) :-
    generi_ordinati(Generi),
    include(over_10, Generi, Dominanti),
    maplist(arg(1), Dominanti, GeneriForti),
    lettura_utente(_, Titolo, plan_to_read, _, GeneriPlan),
    intersection(GeneriPlan, GeneriForti, Comune),
    length(Comune, NComune),
    length(GeneriPlan, NTot),
    NTot > 0,
    Ratio is NComune / NTot,
    Ratio >= 0.5,
    formatta_nome_manga(Titolo, Output).
```

Questo approccio consente di valorizzare manga che l’utente aveva già pianificato di leggere, filtrandoli in base alla loro reale affinità con i gusti dimostrati nella cronologia di lettura.

Esempio di output:

```
      Consigliati tra i PLAN_TO_READ (randomizzati)
NOME MANGA: after god
NOME MANGA: unmei no hito ni deau hanashi
NOME MANGA: monkey peak
NOME MANGA: yamada-kun to lv999 no koi wo suru
NOME MANGA: jinmen
```

5 - Consiglia 5 manga premiati basati sui gusti più frequenti

Raccomanda manga non ancora letti che abbiano ricevuto un riconoscimento (award_winning) e che condividano almeno due generi con quelli più frequentemente letti dall’utente.

Il sistema procede come segue:

- Estrae i generi “dominanti” dell’utente, cioè quelli letti almeno dieci volte.
- Esamina ciascun manga con tag award_winning e verifica che:
    - non sia già stato letto;
    - contenga almeno due generi tra quelli dominanti.

```
manga_premiato(Output) :-
    generi_ordinati(Generi),
    include(over_10, Generi, Dominanti),
    maplist(arg(1), Dominanti, GeneriForti),
    manga(ID, Titolo, GeneriManga, _, _, _, Stato, Autori),
    member(award_winning, GeneriManga),
    \+ lettura_utente(ID, _, _, _, _),
    normalizza_lista(GeneriManga, Puliti),
    intersection(Puliti, GeneriForti, Comune),
    length(Comune, NComune),
    NComune >= 2,
    formatta_output_nome(Titolo, Autori, Stato, Output).
```

- Prima del confronto, i generi vengono normalizzati per evitare problemi di formato (es. underscore iniziali).
- I manga che soddisfano i criteri vengono proposti come raccomandazioni, includendo titolo, autori e stato.

Esempio di output:

```
      Manga premiati nei tuoi generi preferiti (randomizzati)
NOME MANGA: kimi wa pet - AUTORE/I: yayoi ogawa - STATO: finished
NOME MANGA: 5-toubun no hanayome - AUTORE/I: negi haruba - STATO: finished
NOME MANGA: cross game - AUTORE/I: mitsuru adachi - STATO: finished
NOME MANGA: capeta - AUTORE/I: masahito soda - STATO: finished
NOME MANGA: ore monogatari!! - AUTORE/I: kazune kawahara,   aruko - STATO: finished
```

6 - Consiglia 5 manga che combinano generi basati sui gusti più frequenti e generi mai letti

Suggerisce manga non ancora letti che presentano una combinazione di generi familiari e generi completamente nuovi rispetto alle abitudini dell’utente.

Il funzionamento è il seguente:

- Viene estratta l'intera lista dei generi presenti nella knowledge base;
- Si individuano:
    - I generi dominanti, letti almeno dieci volte;
    - I generi mai letti, ottenuti per differenza tra tutti i generi e quelli dell'utente.
- Per ogni manga non ancora letto, si verifica se:
    - contiene almeno un genere dominante;
    - contiene almeno un genere mai letto.
- I generi del manga vengono normalizzati prima del confronto per evitare incongruenze;
- Solo i manga che soddisfano entrambi i criteri vengono inclusi nelle raccomandazioni.

Questa funzionalità favorisce la scoperta controllata, offrendo novità bilanciate da elementi già apprezzati.

```
manga_misto_generi_nuovi(Output) :-
    findall(Genere, (manga(_, _, Generi, _, _, _, _, _), member(Genere, Generi)), TuttiGeneri),
    sort(TuttiGeneri, GeneriTotali),
    frequenza_generi(Freq),
    include(over_10, Freq, GeneriFrequenti),
    maplist(arg(1), GeneriFrequenti, GeneriDominanti),
    findall(GenereLetto, genere_letto(GenereLetto), GeneriLetti),
    sort(GeneriLetti, GeneriUtente),
    subtract(GeneriTotali, GeneriUtente, GeneriMaiLetti),
    manga(ID, Titolo, GeneriManga, _, _, _, Stato, Autori),
    normalizza_lista(GeneriManga, Normalizzati),
    intersection(Normalizzati, GeneriDominanti, ComuneLetti),
    intersection(Normalizzati, GeneriMaiLetti, ComuneNuovi),
    ComuneLetti \= [],
    ComuneNuovi \= [],
    \+ lettura_utente(ID, _, _, _, _),
    formatta_output_nome(Titolo, Autori, Stato, Output).
```

Esempio di output:

```
       Manga che mischiano generi già letti e generi mai letti (randomizzati)
NOME MANGA: wondance - AUTORE/I:  coffee - STATO: currently_publishing
NOME MANGA: w-juliet - AUTORE/I:  emura - STATO: finished
NOME MANGA: konya mo nemurenai - AUTORE/I: kotetsuko yamamoto - STATO: finished
NOME MANGA: one room angel - AUTORE/I:  harada - STATO: finished
NOME MANGA: tadaima, okaeri - AUTORE/I: ichi ichikawa - STATO: currently_publishing
```

7 - Valuta la compatibilità di una lista di generi rispetto alle preferenze

Dati uno o più generi forniti dall'utente, il sistema valuta quanto questi siano compatibili con i gusti di lettura già espressi, in base alla frequenza con cui ciascun genere è stato letto in passato.

La procedura funziona così:

- Viene calcolata la frequenza di ciascun genere letto dall'utente (escludendo i manga in plan_to_read).
- Ogni genere fornito viene normalizzato (rimozione underscore iniziali) e confrontato con le frequenze note.
- A ciascun genere viene assegnato un punteggio:
    - ≥ 20 letture → 3 punti
    - 10–19 letture → 2 punti

```
valuta_genere(Frequenze, Genere, Punteggio) :-
    normalizza_genere(Genere, G),
    ( member(G-Conta, Frequenze) ->
        ( Conta >= 20 -> Punteggio = 3
        ; Conta >= 10 -> Punteggio = 2
        ; Conta >= 5  -> Punteggio = 1
        ; Punteggio = 0 )
    ;   Punteggio = 0 ).
```

- 5–9 letture → 1 punto
- < 5 o mai letto → 0 punti

- Viene calcolata la media aritmetica dei punteggi ottenuti.
- In base alla media risultante, il sistema restituisce un giudizio sintetico:
  - Media ≥ 2 → Molto compatibile
  - Media ≥ 1 → Abbastanza compatibile
  - Media < 1 → Poco compatibile

Questa funzione è utile, ad esempio, per valutare un manga prima della lettura sulla base dei suoi generi.

```
valuta_compatibilita(GeneriForniti) :-
    frequenza_generi(Frequenze),
    maplist(valuta_genere(Frequenze), GeneriForniti, Punteggi),
    sum_list(Punteggi, Somma),
    length(Punteggi, N),
    (N =:= 0 -> Media = 0 ; Media is Somma / N),
    (
        Media >= 2 -> writeln('Questo manga è MOLTO compatibile con i tuoi gusti!')
    ;   Media >= 1 -> writeln('Questo manga è ABBASTANZA compatibile con i tuoi gusti.')
    ;   writeln('Questo manga è POCO compatibile con i tuoi gusti.')
    ).
```

Esempio di output:

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

```
Inserisci i generi separati da virgola (es: action, fantasy, drama):
|: memoir, mecha, seinen
Questo manga è ABBASTANZA compatibile con i tuoi gusti.
```

```
Inserisci i generi separati da virgola (es: action, fantasy, drama):
|: space, shoujo
Questo manga è POCO compatibile con i tuoi gusti.
```

8 - Verifica lettura di un manga specifico

Permette all'utente di inserire manualmente il titolo di un manga e verifica se è presente nel proprio storico di lettura.

Funzionamento:

- L'utente digita il nome del manga (anche con spazi).
- Il titolo viene **normalizzato** sostituendo gli spazi con underscore, in modo da renderlo compatibile con il formato della knowledge base.

```
ha_letto_manga :-
    write('Inserisci il nome del manga: '),
    read_line_to_string(user_input, Input),
    normalize_input(Input, TitoloNorm),
    (   lettura_utente(_, TitoloNorm, Stato, Voto, _)
    ->  format('Hai letto ~w. Stato: ~w, Punteggio: ~w~n', [TitoloNorm, Stato, Voto])
    ;   format('Non hai letto ~w.~n', [TitoloNorm])
    ).
```

- Il sistema verifica se esiste un fatto lettura_utente/5 corrispondente al titolo.
  - Se sì, mostra lo **stato della lettura** (es. reading, completed) e il **punteggio assegnato**.
  - Se no, comunica che il manga non è stato letto;
  - Se è un manga plan_to_read, avvisa l'utente di ciò.

Questa funzione è utile per controllare rapidamente lo stato di un manga prima di iniziare una nuova lettura.

Esempio di output:

```
Inserisci il nome del manga: berserk
Hai letto berserk. Stato: dropped, Punteggio: 10.0
```

```
Inserisci il nome del manga: touge oni
Non hai letto touge_oni.
```

```
Inserisci il nome del manga: dr. stone
Vorresti leggere dr._stone (presente in plan_to_read)
```

5.2 Menu interattivo

Il **sistema di raccomandazione logico-simbolico** offre un **menu testuale interattivo**, sviluppato in **Prolog** all'interno del file system.pl, che consente all'utente di **esplorare i dati** e ricevere **suggerimenti personalizzati** in base alla propria cronologia di lettura.

Funzionamento

Per avviare il menu, è necessario consultare il file system.pl, quindi digitare i seguenti comandi:

```
PS C:                                   \ICON> cd KB
PS C:                                   \ICON\KB> swipl
Welcome to SWI-Prolog (threaded, 64 bits, version 9.2.9)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

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

2 ?- menu.
```

Una volta eseguito, il sistema presenta una **lista numerata di opzioni**, ognuna delle quali attiva una specifica **regola di raccomandazione o analisi**. Selezionando un'opzione (tramite il numero corrispondente, ovvero per la prima opzione scrivere '1.' senza le virgolette), il sistema esegue la regola associata e **mostra il risultato a video**.

```
      SISTEMA DI RACCOMANDAZIONE MANGA
1. Visualizza i generi preferiti (ordinati per frequenza)
2. Consiglia 5 manga basati sui gusti più frequenti
3. Consiglia 5 manga di qualità ma poco popolari basati sui gusti più frequenti
4. Consiglia 5 manga dalla tua lista "plan_to_read" basati sui gusti più frequenti
5. Consiglia 5 manga premiati basati sui gusti più frequenti
6. Consiglia 5 manga che combinano generi basati sui gusti più frequenti e generi mai letti
7. Valuta la compatibilità di una lista di generi rispetto alle preferenze
8. Verifica lettura di un manga specifico
9. Esci dal programma
Scelta (1-9): 1.
      Generi preferiti (ordinati)
action-226
fantasy-193
```

Gestione delle scelte

Le opzioni del menu sono gestite tramite il predicato **esegui_scelta(N)**, dove N rappresenta l'opzione selezionata. Ad esempio, l'invocazione di:

```
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.
```

esegue il seguente flusso:

- attiva la regola raccomanda_random/1;
- seleziona **cinque risultati casuali** tra quelli compatibili;
- li visualizza a schermo;
- infine, **richiama il menu principale** per consentire nuove interazioni.

6 Ontologia OWL (modulo dimostrativo)

Questa sezione presenta un **modulo dimostrativo** di rappresentazione della conoscenza mediante un'ontologia OWL, sviluppata con l'obiettivo di arricchire il progetto.

L'ontologia, pur **non essendo integrata direttamente,** dimostra come concetti e relazioni del dominio possano essere formalizzati in **linguaggio OWL** e successivamente utilizzati per inferenze logiche.

6.1 Struttura dell'ontologia

L'ontologia manga.owl è progettata per rappresentare simbolicamente alcune proprietà essenziali nel dominio dei manga. Modellata secondo le specifiche OWL 2, fornisce una base concettuale su cui è possibile applicare ragionamenti automatici.

Classi principali:

- Manga: classe base per i manga;
- Seinen: rappresenta un genere narrativo destinato a un pubblico adulto; può essere modellato come sottoclasse o istanza;
- AwardWinning: classe associata a opere che hanno ricevuto un premio.

Proprietà (ObjectProperty):

- hasGenre: collega un manga a uno o più generi (es. Seinen);
- hasAward: collega un manga a un concetto di riconoscimento (es. AwardWinning).

Individui di esempio:

- Berserk: definito come istanza della classe Manga, collegato a AwardWinning tramite hasAward.

```
<!-- Identificatore dell'ontologia -->
<owl:Ontology rdf:about="http://example.org/manga"/>

<!-- Definizione delle classi OWL: entità concettuali -->
<owl:Class rdf:about="#Manga"/> <!-- Classe principale per i manga -->
<owl:Class rdf:about="#Seinen"/> <!-- Genere specifico (es. Seinen) -->
<owl:Class rdf:about="#AwardWinning"/> <!-- Categoria per manga premiati -->

<!-- Proprietà oggetto: relazioni tra individui -->
<owl:ObjectProperty rdf:about="#hasGenre"/> <!-- Associa un manga a un genere -->
<owl:ObjectProperty rdf:about="#hasAward"/> <!-- Associa un manga a un premio -->

<!-- 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>
```

6.2 Ragionamento con HermiT

Il ragionamento sull'ontologia è stato effettuato utilizzando **HermiT**, un reasoner per OWL 2, integrato in Python tramite la libreria **Owlready2**.

L'**obiettivo** è dimostrare l'uso di ragionamento semantico per dedurre conoscenze implicite a partire da relazioni esplicitamente dichiarate.

Una volta attivato il reasoner, è possibile eseguire interrogazioni sugli individui della classe Manga e ottenere risultati dedotti automaticamente, come ad esempio:

```
# 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
```

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

In questo caso, l'appartenenza di berserk alla classe dei Manga premiati è stata **inferita automaticamente** a partire dalla proprietà hasAward, senza che fosse esplicitamente dichiarata come istanza di AwardWinning.

Conclusione

Il modulo OWL è stato inserito a scopo **didattico e illustrativo**, per mostrare l'integrazione possibile tra IA simbolica e tecnologie del **Semantic Web**.
Pur non essendo integrato, evidenzia come sia possibile arricchire i sistemi intelligenti con componenti semantiche orientate all'interoperabilità, esplicitazione della conoscenza e ragionamento automatico.

7 Apprendimento supervisionato

In questa sezione si descrive l'applicazione di tecniche di **apprendimento supervisionato** con l'obiettivo di prevedere il **gradimento di un manga** da parte dell'utente, sulla base di caratteristiche **numeriche** e **semantiche**.

L'obiettivo è costruire un **classificatore binario** capace di stimare se un manga possa piacere (1) o meno (0) a un determinato utente, utilizzando come feature:

- i **generi** associati all'opera (codificati binariamente);
- e alcune **metriche globali** tratte dal dataset (punteggio medio, rank, popolarità).

Questa fase rappresenta la componente **subsimbolica** del sistema: l'apprendimento si basa sull'analisi dei dati storici dell'utente attraverso modelli statistici in grado di generalizzare le sue preferenze.
I risultati costituiscono una base **predittiva** utile sia per la **raccomandazione diretta**, sia per l'**integrazione con il motore logico-simbolico**.

Fasi operative:

- Costruzione del dataset supervisionato a partire da dataset_ml.csv;
- Binarizzazione dei generi tramite MultiLabelBinarizer;
- Definizione della variabile target Piace, con valore 1 per manga valutati ≥ 7, 0 altrimenti;
- Addestramento e confronto di diversi modelli di classificazione;
- Valutazione delle performance tramite **cross-validation 5-fold**, usando le principali metriche.

Durante il preprocessing:

- Vengono eliminate tutte le righe con Punteggio_Utente non valido (NaN o ≤ 0);
- I generi mancanti sono riempiti con liste vuote;
- Dopo l'applicazione dell'One-Hot Encoding, tutti i NaN residui vengono impostati a 0, evitando problemi nei modelli che non supportano valori nulli.

Output dell'analisi

L'analisi produce:

- grafici comparativi (accuracy, precisione, ecc.);
- matrici di confusione;
- radar plot aggregati.

Questi strumenti consentono di **identificare i modelli più adatti al profilo dell'utente**, offrendo un **supporto predittivo solido** per le attività successive di raccomandazione e integrazione con moduli simbolici.

7.1 Classificazione

7.1.1 Decision Tree

Il **Decision Tree** è un algoritmo di classificazione supervisionata che rappresenta il processo decisionale tramite una struttura ad **albero gerarchico**.
In questa struttura:

- ogni **nodo interno** rappresenta una condizione su una feature del dataset;
- ogni **ramo** corrisponde a un valore o a una soglia della feature;
- ogni **foglia** rappresenta una classe di output (in questo caso: Piace = 1 oppure Non piace = 0).

L'albero viene costruito in modo ricorsivo: ad ogni passo, l'algoritmo seleziona la feature e la soglia che **massimizzano la purezza** dei sottoinsiemi risultanti.

Iperparametri principali

- **max_depth**: profondità massima dell'albero. Un valore troppo elevato può portare a overfitting; troppo basso può generare underfitting;
- **min_samples_leaf**: numero minimo di campioni richiesti in un nodo foglia;
- **splitter**: strategia usata per scegliere la feature su cui dividere ("best" o "random").

Motivazioni della scelta del modello

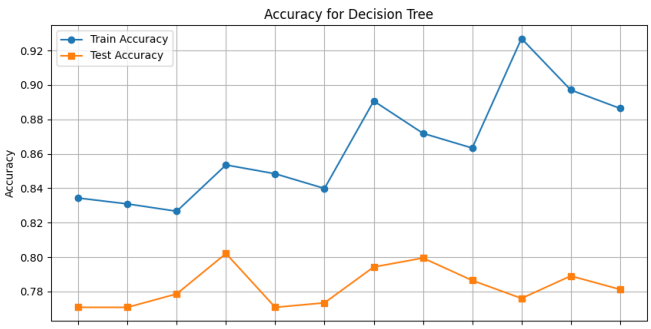
Il Decision Tree è stato selezionato come **modello iniziale** per l'analisi supervisionata in quanto:

- è **semplice da implementare** e **altamente interpretabile**;
- consente una **visualizzazione intuitiva** del processo decisionale;
- rappresenta una **baseline efficace**, utile per confrontare le prestazioni con metodi più complessi (come Random Forest o XGBoost);
- offre la possibilità di osservare con chiarezza il fenomeno di **overfitting**, facilmente controllabile tramite i suoi iperparametri principali.

Grafico delle Accuratezze

Il grafico mostra l'andamento dell'**accuracy sul training set** (linea blu) e sul **test set** (linea arancione) per diverse combinazioni di iperparametri del Decision Tree:

- max_depth: {4, 6, 8, 10};
- min_samples_leaf: {1, 5, 10};
- class_weight: 'balanced'.



Osservazioni principali:

- **Overfitting contenuto**:
  L'accuracy sul training set mostra valori elevati (fino al 93%), ma non raggiunge mai il 100%, segno che l'albero non ha memorizzato completamente i dati.
  La distanza tra training e test accuracy è visibile, ma non estrema: questo indica che l'overfitting è presente ma **ben gestito** grazie ai vincoli introdotti;

- **Effetto della profondità (max_depth)**:
  Al crescere della profondità, l'accuracy sul training set tende ad aumentare (come atteso), ma quella sul test set **non segue lo stesso andamento**.
  Le performance migliori sul test si osservano per profondità intermedie (es. max_depth=6–8), suggerendo che un modello troppo profondo cattura rumore e perde capacità di generalizzazione;
- **Effetto della dimensione minima delle foglie (min_samples_leaf)**:
  Valori maggiori di min_samples_leaf contribuiscono a ridurre la complessità del modello, limitando gli split su insiemi molto piccoli.
  Questo ha un **effetto** migliora la stabilità dell'accuracy sul test set;
- **Test accuracy stabile ma non eccelsa (~77–80%)**:
  Indica che il modello ha raggiunto un buon compromesso tra bias e varianza, ma la capacità predittiva è ancora **limitata**, giustificando l'esplorazione di modelli più sofisticati.

Conclusione

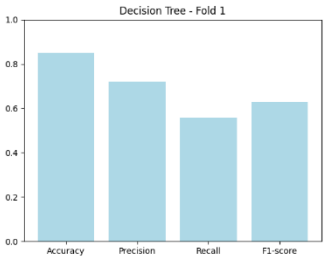
Il grafico conferma che, grazie all'uso degli iperparametri (max_depth, min_samples_leaf, class_weight='balanced'), è stato possibile **contenere l'overfitting**, mantenendo buone prestazioni sul test set.
Questa configurazione rende il Decision Tree un valido punto di partenza, sebbene siano auspicabili modelli più robusti per migliorare ulteriormente la generalizzazione.

Valutazione tramite Cross Validation (5 Fold)

Per valutare la stabilità e le capacità predittive del modello Decision Tree, è stata eseguita una **validazione incrociata a 5 fold**. Di seguito vengono riportati i risultati per ciascun fold, accompagnati da un'analisi delle variazioni osservate tra le diverse partizioni del dataset.

Fold 1

- **Accuracy**: 0.849;
- **Precision**: 0.721;
- **Recall**: 0.557;
- **F1-score**: 0.628.

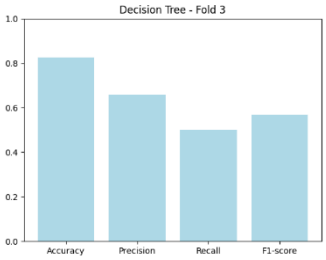


Fold 2

- **Accuracy**: 0.843;
- **Precision**: 0.696;
- **Recall**: 0.552;
- **F1-score**: 0.615.

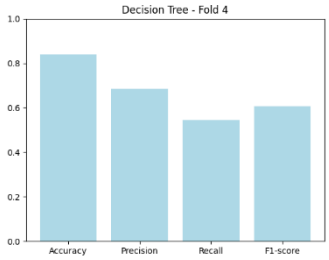
Fold 3

- **Accuracy**: 0.825;
- **Precision**: 0.657;
- **Recall**: 0.500;
- **F1-score**: 0.568.



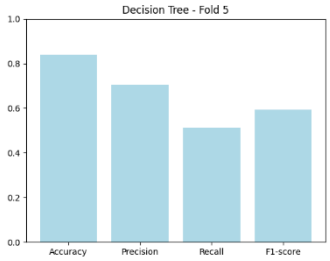
Fold 4

- **Accuracy**: 0.838;
- **Precision**: 0.686;
- **Recall**: 0.545;
- **F1-score**: 0.608.



Fold 5

- **Accuracy**: 0.838;
- **Precision**: 0.703;
- **Recall**: 0.511;
- **F1-score**: 0.592.



Variazioni tra i fold

- **Accuracy**:
  L'accuratezza si mantiene relativamente stabile in tutti i fold, oscillando in un intervallo ristretto tra **0.825 e 0.849**. Il valore massimo si registra nel Fold 1, mentre il minimo nel Fold 3. Questa stabilità indica che il modello riesce a generalizzare bene anche su suddivisioni differenti del dataset;
- **Precision**:
  Anche la precisione presenta una buona costanza, variando tra **0.657 (Fold 3)** e **0.721 (Fold 1)**. Il modello tende a essere più preciso nei fold in cui anche l'accuracy è più alta. Questo suggerisce che, quando il modello è più sicuro nella classificazione, tende anche a commettere meno falsi positivi;
- **Recall**:
  Il **recall è la metrica che mostra maggiore variabilità** tra i fold, passando da **0.500 (Fold 3)** a **0.557 (Fold 1)**. Questo comportamento evidenzia come la capacità del modello di identificare correttamente i manga graditi possa risentire della particolare suddivisione del training set. Tuttavia, non si rilevano fluttuazioni eccessive;
- **F1-score**:
  Di conseguenza, anche l'F1-score mostra variazioni limitate, con valori compresi tra **0.568 (Fold 3)** e **0.628 (Fold 1)**. La tendenza segue quella del recall, ma in forma più smussata.

Considerazioni finali

Nel complesso, le performance del Decision Tree risultano **coerenti tra i diversi fold**, con variazioni contenute.
La metrica più instabile è il **recall**, seguita dall'**F1-score**, il che conferma una leggera tendenza del modello a **sottostimare i positivi**.

7.1.2 Random Forest

La **Random Forest** è un algoritmo di classificazione supervisionata basato su un **ensemble di alberi decisionali**.
Ogni albero viene addestrato su un **sottoinsieme casuale del dataset** (tecnica del *bootstrapping*) e, a ogni nodo, utilizza solo un **sottoinsieme casuale di feature** per determinare la suddivisione (*bagging* + *feature randomness*).

La predizione finale è ottenuta mediante **voto di maggioranza** tra i singoli alberi.

Questa strategia consente di:

- **Ridurre l'overfitting** tipico del singolo Decision Tree;
- **Aumentare la robustezza e la stabilità** del modello;
- Ottenere stime **più generalizzabili**, anche in presenza di dati rumorosi.

Iperparametri principali

- **n_estimators**: numero di alberi nella foresta. Un numero maggiore migliora la stabilità delle predizioni, a scapito del tempo di calcolo;
- **max_depth**: profondità massima degli alberi;
- **min_samples_leaf**: numero minimo di campioni richiesti in un nodo foglia. Aiuta a prevenire overfitting sui singoli alberi;
- **max_features**: numero di feature considerate per ogni split. Regola la diversificazione tra gli alberi;
- **class_weight**: peso da assegnare alle classi per gestire dataset sbilanciati.

Motivazioni della scelta del modello

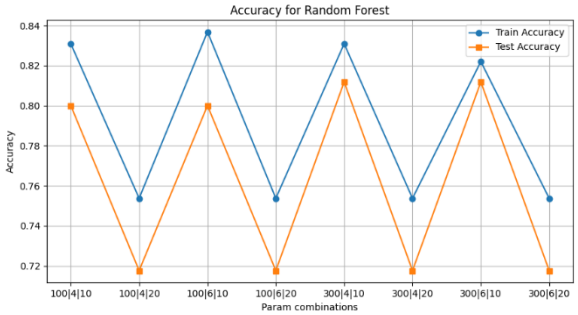
La **Random Forest** è stata selezionata come **secondo passo naturale** dopo il Decision Tree, per analizzare quanto un approccio ensemble possa migliorare la **capacità predittiva** e la **stabilità** del sistema.
Il modello risulta:

- **Robusto** anche su dataset complessi o con feature ridondanti;
- **Meno incline all'overfitting**, grazie alla diversità tra gli alberi;
- Efficiente anche con un tuning limitato, rendendolo adatto a scenari reali.

Grafico delle Accuratezze

Il grafico mostra l'andamento dell'accuracy sul training set (linea blu) e sul test set (linea arancione) per diverse combinazioni di iperparametri della Random Forest:

- n_estimators: {100, 300, 500};
- max_depth: {6, 10, 12};
- min_samples_leaf: {1, 5, 10};
- max_features: {'sqrt', 'log2', None};
- class_weight: 'balanced'.



Osservazioni principali:

- **Overfitting contenuto**:
  L'accuracy sul training set raggiunge valori molto alti (fino al 94%), ma resta sempre inferiore al 100%, segno che il modello non ha completamente memorizzato i dati.
  La distanza tra le due curve è visibile ma contenuta, indicando che l'overfitting è presente, ma **ben controllato** dalla natura ensemble del modello;
- **Effetto del numero di alberi (n_estimators)**:
  Aumenti nel numero di alberi tendono a **stabilizzare l'accuracy**, ma non sempre migliorano la

performance sul test.
La stabilità migliora con n_estimators ≥ 300, anche se oltre un certo punto l'incremento è marginale;

- **Effetto della profondità (max_depth)**:
  Profondità maggiori portano a train accuracy più alta, ma anche a una maggiore distanza dal test set, segno di overfitting.
  Le migliori performance sul test si osservano con max_depth=10, valore che bilancia espressività e generalizzazione;
- **Effetto della dimensione minima delle foglie (min_samples_leaf)**:
  Valori più alti per min_samples_leaf riducono la complessità dei singoli alberi, contribuendo a migliorare la generalizzazione e **ridurre l'instabilità tra le curve**;
- **Test accuracy stabile (~80–84%)**:
  Pur non raggiungendo valori altissimi, la test accuracy si mantiene stabile in quasi tutte le combinazioni, confermando la **robustezza del modello** anche in presenza di parametri diversi.

Conclusione

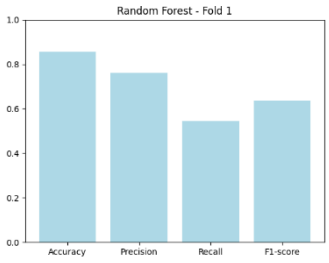
Il grafico conferma che la Random Forest, grazie all'uso combinato di bootstrapping, randomizzazione e aggregazione, **riesce a contenere l'overfitting** pur mantenendo prestazioni elevate.
Le accuratezze sul test set indicano che si tratta di un classificatore **affidabile e ben bilanciato**, ideale per un sistema di raccomandazione che richiede stabilità e generalizzazione.

Valutazione tramite Cross Validation (5 Fold)

Per valutare la stabilità e le capacità predittive della Random Forest, è stata eseguita una validazione incrociata a 5 fold. Di seguito vengono riportati i risultati per ciascun fold, accompagnati da un'analisi delle variazioni osservate tra le diverse partizioni del dataset.

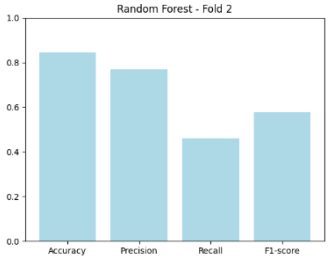
Fold 1

- Accuracy: 0.857;
- Precision: 0.762;
- Recall: 0.545;
- F1-score: 0.636.



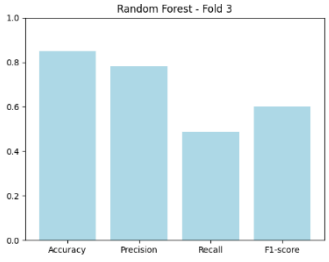
Fold 2

- Accuracy: 0.846;
- Precision: 0.769;
- Recall: 0.460;
- F1-score: 0.576.



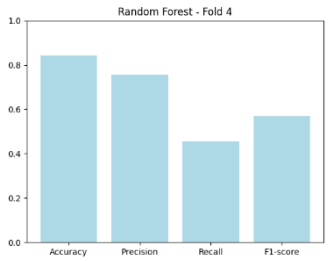
Fold 3

- Accuracy: 0.851;
- Precision: 0.782;
- Recall: 0.489;
- F1-score: 0.601.



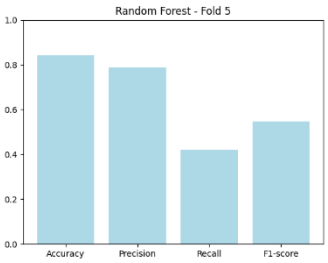
Fold 4

- Accuracy: 0.841;
- Precision: 0.755;
- Recall: 0.455;
- F1-score: 0.567.



Fold 5

- Accuracy: 0.841;
- Precision: 0.787;
- Recall: 0.420;
- F1-score: 0.548.



Variazioni tra i fold

- **Accuracy**:
  I valori sono **molto stabili**, oscillando in un range ristretto tra 0.841 e 0.857. Il modello mantiene buone prestazioni su tutte le partizioni, indicando un'elevata capacità di generalizzazione.
- **Precision**:
  Anch'essa stabile e sempre elevata (tra 0.755 e 0.787). La Random Forest tende a **limitare i falsi positivi**, classificando come "piace" solo i casi con alta probabilità.
- **Recall**:
  È la metrica più variabile (da 0.420 a 0.545), suggerendo che il modello **fatica a catturare tutti i manga apprezzati**.
- **F1-score**:
  Si mantiene tra 0.548 e 0.636, seguendo l'andamento del recall. Le variazioni non sono eccessive e confermano un buon bilanciamento tra precisione e richiamo.

Considerazioni finali

Nel complesso, la Random Forest **dimostra maggiore stabilità e performance globali superiori** rispetto al Decision Tree.
Il **recall rimane il punto debole**, ma le alte precisioni e le accuratezze stabili rendono questo modello **molto affidabile** in un contesto di raccomandazione.

7.1.3 AdaBoost

AdaBoost (*Adaptive Boosting*) è un algoritmo di classificazione supervisionata basato su un approccio **ensemble iterativo**, che combina più **classificatori deboli** (solitamente alberi decisionali a profondità uno, detti *decision stump*) per costruire un classificatore forte.

Il funzionamento si basa su un meccanismo adattivo:

- Dopo ogni iterazione, gli esempi classificati erroneamente ricevono **un peso maggiore**;
- Il classificatore successivo viene **indirizzato a correggere gli errori** del precedente;
- Ogni classificatore viene **pesato in base alla propria accuratezza**;
- Le predizioni finali si ottengono tramite **una votazione pesata** di tutti i classificatori.

Questo schema consente di concentrare l'attenzione del modello su istanze "difficili", migliorando progressivamente le performance complessive.

Iperparametri principali

- **n_estimators**: numero di classificatori deboli. Valori più alti permettono una maggiore espressività del modello, ma aumentano il rischio di overfitting;
- **learning_rate**: coefficiente che controlla il peso assegnato a ciascun classificatore. Valori bassi rallentano l'apprendimento ma favoriscono la generalizzazione;
- **base_estimator** (opzionale): permette di definire un classificatore personalizzato, ad esempio un Decision Tree con max_depth > 1, che può essere configurato con ulteriori iperparametri (es. min_samples_split, min_samples_leaf, ecc.).

Motivazione della scelta

AdaBoost è stato selezionato per la sua **capacità di trasformare classificatori semplici in modelli predittivi efficaci**, sfruttando una strategia adattiva mirata a **correggere progressivamente gli errori**.

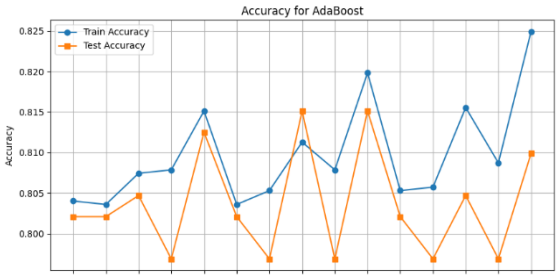
Le principali motivazioni includono:

- **Robustezza contro l'overfitting**, se adeguatamente regolarizzato (via learning_rate o limitando la complessità del base learner);
- **Adattabilità** alle caratteristiche del dataset, soprattutto in presenza di **istanze soggettive o difficili**, come nel caso delle preferenze utente;
- **Buone prestazioni empiriche** in scenari reali, anche con dataset di dimensioni moderate.

Grafico delle Accuratezze

Il grafico mostra l'andamento dell'accuracy sul training set (linea blu) e sul test set (linea arancione) per diverse combinazioni di iperparametri di AdaBoost:

- n_estimators: {50, 100, 200};
- learning_rate: {0.05, 0.1, 0.3, 0.5, 1.0}.



Osservazioni principali:

- **Overfitting contenuto**:
  L'accuracy sul training set non supera mai il 83%, e si mantiene vicina a quella del test set. Questo comportamento è indicativo di un **modello ben regolarizzato**, che evita la memorizzazione dei dati;
- **Effetto del numero di stimatori (n_estimators)**:
  L'incremento di n_estimators porta generalmente a un miglioramento della train accuracy, ma non sempre ha effetti positivi sulla test accuracy.
  Si osserva che oltre i 100 stimatori, il beneficio tende a stabilizzarsi o addirittura a regredire in alcune combinazioni;

- **Effetto del tasso di apprendimento (learning_rate)**:
  Valori troppo alti (es. 1.0) possono causare oscillazioni nell'accuracy di test, sintomo di **eccessivo adattamento** ai dati.
  I risultati più stabili si osservano con learning_rate compresi tra **0.1 e 0.3**, che garantiscono un buon compromesso tra velocità di apprendimento e generalizzazione;

- **Accuratezza test stabile (~80–82%)**:
  Le curve di test accuracy sono poco variabili tra le combinazioni, segno che il modello è **poco sensibile agli iperparametri**, ma anche che la sua **capacità predittiva è limitata**.

Conclusione

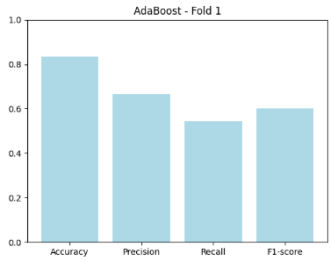
Il grafico mostra che AdaBoost è un **modello ben bilanciato**, capace di mantenere prestazioni costanti senza rischiare overfitting.
La **regolarizzazione tramite learning_rate** si rivela efficace per evitare eccessiva complessità.
Sebbene le prestazioni siano inferiori rispetto a Random Forest o XGBoost, la **semplicità e la stabilità** del modello lo rendono un'ottima scelta in contesti moderati.

Valutazione tramite Cross Validation (5 Fold)

Per valutare la stabilità del modello AdaBoost, è stata eseguita una validazione incrociata a 5 fold. Di seguito vengono riportati i risultati ottenuti:

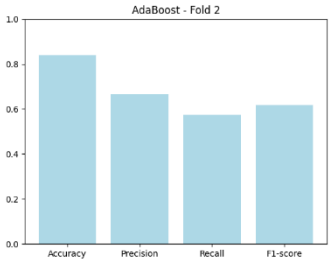
Fold 1

- Accuracy: 0.833;
- Precision: 0.667;
- Recall: 0.545;
- F1-score: 0.600.



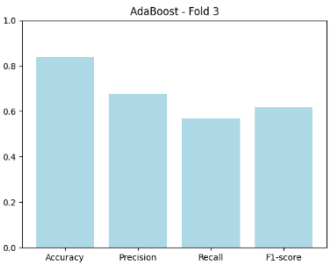
Fold 2

- Accuracy: 0.838;
- Precision: 0.667;
- Recall: 0.575;
- F1-score: 0.617.



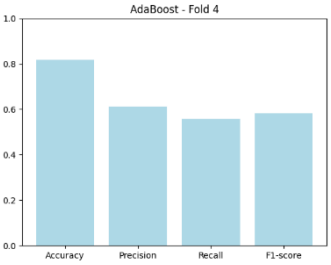
Fold 3

- Accuracy: 0.838;
- Precision: 0.676;
- Recall: 0.568;
- F1-score: 0.617.

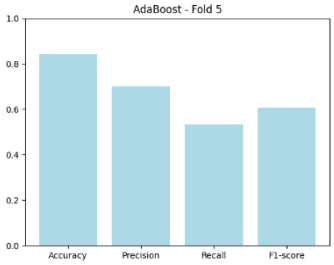


Fold 4

- Accuracy: 0.817;
- Precision: 0.613;
- Recall: 0.557;
- F1-score: 0.583.



Fold 5



- Accuracy: 0.841.
- Precision: 0.701;
- Recall: 0.534;
- F1-score: 0.606.

Variazioni tra i fold

- **Accuracy**:
  L'accuracy mostra una buona stabilità (0.817–0.841), confermando che AdaBoost è in grado di mantenere prestazioni affidabili su differenti suddivisioni del dataset.
- **Precision**:
  Varia da 0.613 a 0.701. Il modello tende a **contenere i falsi positivi**, mantenendo una precisione soddisfacente anche nei fold con performance inferiori.
- **Recall**:
  Più sensibile alla variazione dei dati (0.534–0.575), ma comunque **più alto rispetto a modelli conservativi** come Random Forest.
  AdaBoost riesce meglio a catturare gli esempi positivi.
- **F1-score**:
  Si mantiene tra 0.583 e 0.617, dimostrando un buon equilibrio tra precisione e richiamo.

Considerazioni finali

AdaBoost si dimostra un modello **coerente, stabile e ben regolarizzato**.
È meno soggetto a overfitting rispetto a modelli ensemble più complessi e fornisce una buona capacità di generalizzazione anche con un numero moderato di stimatori.

7.1.4 KNN

Il **K-Nearest Neighbors (KNN)** è un algoritmo di classificazione supervisionata basato sul concetto di **similarità tra istanze**.
Per classificare un nuovo esempio, il modello **non costruisce un modello esplicito**, ma memorizza i dati di training e, durante la fase di predizione, **identifica i *k* esempi più vicini** secondo una metrica (tipicamente la distanza euclidea). La classe viene assegnata in base alla **maggioranza dei vicini**.

Iperparametri principali

- **n_neighbors**: numero di vicini considerati per la classificazione. Valori più alti **favoriscono la generalizzazione**, ma possono rendere **meno netti i confini decisionali**;
- **weights**: strategia di ponderazione dei vicini:
  - "uniform": tutti i vicini contribuiscono allo stesso modo;
  - "distance": i vicini più prossimi pesano di più nella decisione.
- **metric**: metrica di distanza utilizzata per misurare la similarità.

Motivazioni della scelta del modello

Il **KNN è stato selezionato come benchmark non parametrico**, utile per valutare l'efficacia della classificazione **basata su prossimità tra istanze**.

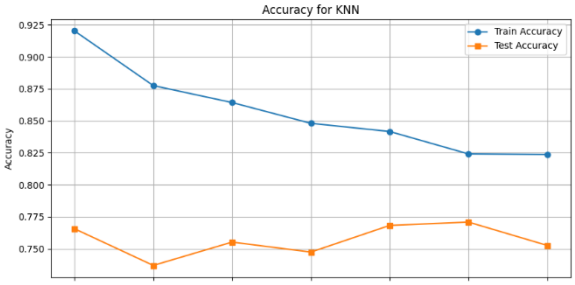
In particolare:

- Permette di osservare l'impatto delle **feature numeriche e binarizzate** sulla definizione di similarità tra manga;
- Consente di analizzare la **sensibilità del sistema** rispetto alla scelta di *k*, evidenziando eventuali fluttuazioni di performance;
- Fornisce un punto di confronto con **modelli più strutturati o ensemble**, mostrando i limiti e i punti di forza di un approccio puramente geometrico.

Grafico delle Accuratezze

Il grafico mostra l'andamento dell'accuracy sul training set (linea blu) e sul test set (linea arancione) per diverse combinazioni del parametro n_neighbors del modello KNN:

- n_neighbors: {3, 5, 7, 9, 11, 15, 21}.



Osservazioni principali:

- **Overfitting evidente per valori bassi di k**:
  Con k=3, l'accuracy sul training set raggiunge il 92%, mentre quella sul test si mantiene bassa (~76%).
  Questo indica un **forte overfitting**: il modello si adatta troppo ai dati locali del training set.
- **Effetto del numero di vicini (n_neighbors)**:
  All'aumentare di k, la curva di training accuracy si abbassa (segno di minore complessità del modello), mentre quella del test diventa **più stabile**, con un **massimo attorno a k=11**.
- **Test accuracy bassa e piatta (~74–77%)**:
  Le prestazioni sul test set **non mostrano miglioramenti significativi** con l'aumento di k. Il modello fatica a generalizzare, probabilmente a causa di feature eterogenee e rumore nei dati.

Conclusione

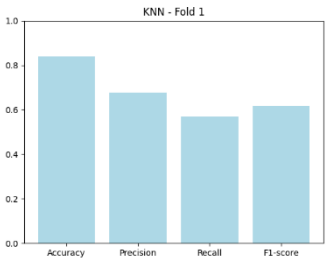
Il KNN si dimostra **molto sensibile alla scelta di k**, con prestazioni deboli sul test set e una forte tendenza all'overfitting per valori piccoli.
L'assenza di un modello interno e la dipendenza diretta dai dati rendono KNN **inadatto a problemi in cui le feature non sono normalizzate** o i dati sono sparsi.
Tuttavia, il modello resta un **benchmark utile per confrontare la classificazione basata su similarità pura**.

Valutazione tramite Cross Validation (5 Fold)

Per valutare le prestazioni del modello KNN, è stata condotta una validazione incrociata a 5 fold. Di seguito i risultati:

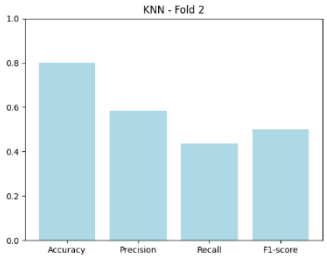
Fold 1

- Accuracy: 0.839;
- Precision: 0.676;
- Recall: 0.568;



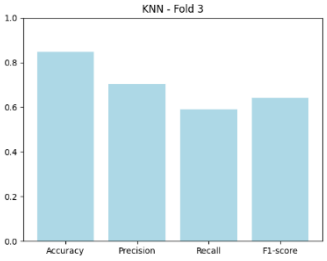
- F1-score: 0.617.

Fold 2



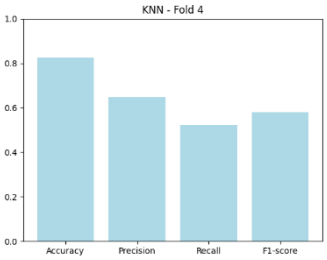
- Accuracy: 0.802;
- Precision: 0.585;
- Recall: 0.437;
- F1-score: 0.500.

Fold 3



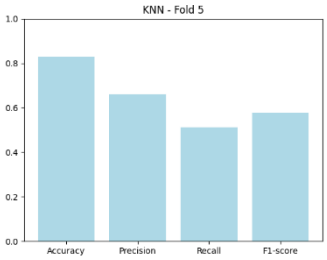
- Accuracy: 0.849;
- Precision: 0.703;
- Recall: 0.591;
- F1-score: 0.642.

Fold 4



- Accuracy: 0.825;
- Precision: 0.648;
- Recall: 0.523;
- F1-score: 0.579.

Fold 5



- Accuracy: 0.828;
- Precision: 0.662;
- Recall: 0.511;
- F1-score: 0.577.

Variazioni tra i fold

- **Accuracy**:
  Relativamente stabile (range: 0. 802–0.849), con un leggero calo nel Fold 2. Il modello mostra **una discreta capacità di generalizzazione**, ma non eccelle;
- **Precision**:
  Variabile da 0.585 (Fold 2) a 0.703 (Fold 3), confermando che **l'efficacia nel classificare correttamente i positivi dipende fortemente dalla distribuzione locale dei dati**;
- **Recall**:
  È la metrica più instabile (0.437–0.591), sintomo di **sensibilità agli outlier e ai dati poco densi**;
- **F1-score**:
  Oscilla tra 0.500 e 0.642. Rispecchia la media tra precision e recall, rivelando **prestazioni moderate** con un equilibrio appena sufficiente.

Considerazioni finali

Il KNN si conferma un **modello semplice ma debole** nel contesto del problema affrontato.
Soffre di **overfitting** con k piccoli e **scarsa generalizzazione** anche con k maggiori.

Nonostante ciò, rappresenta un utile **termine di paragone per valutare l'efficacia di modelli più sofisticati**, soprattutto quelli basati su apprendimento strutturato o ensemble.

7.1.5 Naive Bayes

Il **Naive Bayes** è una famiglia di classificatori probabilistici basati sul **Teorema di Bayes**, che assume (in maniera *naïve*) l'**indipendenza tra le feature** condizionatamente alla classe.
Nonostante questa assunzione forte e raramente verificata nei dati reali, il modello si è dimostrato **efficace in numerosi contesti pratici**, in particolare su dataset con **feature binarie o categoriali**.

Nel nostro caso, il classificatore è stato applicato su dati misti:

- Feature binarie (generi binarizzati);
- Feature numeriche (punteggio, rank, popolarità).

Iperparametri

Una delle principali caratteristiche del Naive Bayes è la **semplicità di configurazione**:

- Non è necessario effettuare tuning avanzato.
- Il modello **non ha iperparametri critici** nella configurazione standard (eccetto alpha per il Laplace smoothing, non modificato).
- La semplicità computazionale consente **addestramento e predizione rapidi**, anche su dataset estesi.

Motivazione della scelta

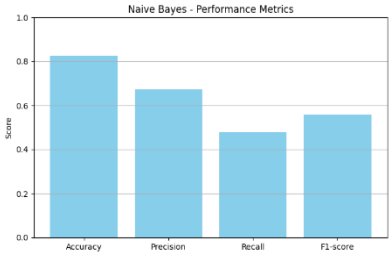
Il Naive Bayes è stato introdotto come **baseline probabilistica interpretabile**, per le seguenti ragioni:

- **Efficienza computazionale**: è tra i modelli più rapidi da addestrare ed eseguire;
- **Adattabilità a feature indipendenti**: si adatta bene alla rappresentazione binaria dei generi manga;
- **Elevato recall**: la tendenza a classificare positivi con maggiore libertà consente di **non escludere manga potenzialmente graditi**, risultando utile in un contesto di raccomandazione;
- **Trasparenza**: il modello fornisce una stima esplicita della probabilità a posteriori, facilitando l'interpretazione dei risultati.

Grafico delle Metriche

Il grafico a barre mostra le performance medie ottenute dal modello **Naive Bayes**, in termini di:

- **Accuracy**: ~82.7%;
- **Precision**: ~67.3%;
- **Recall**: ~47.9%;
- **F1-score**: ~55.8%.



Osservazioni principali:

- **Accuracy stabile e buona** (~82–84%):
  Indica che il modello è in grado di classificare correttamente la maggior parte delle istanze, nonostante la semplicità dell'approccio.

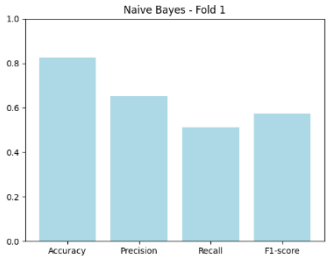
- **Precisione elevata rispetto alla complessità del modello**:
  Dimostra che, pur con assunzioni forti di indipendenza tra feature, il modello è in grado di ridurre i falsi positivi in modo consistente.
- **Recall relativamente basso (~48%)**:
  Il modello **tende a essere conservativo**, classificando positivi solo se molto "convinto", con il rischio di escludere alcuni manga che avrebbero potuto piacere.
  Questo comportamento è coerente con la natura probabilistica e la forte indipendenza assunta tra le feature.
- **F1-score medio** (~56%):
  Rappresenta un equilibrio accettabile tra precision e recall, pur penalizzato dal richiamo limitato.

Valutazione tramite Cross Validation (5 Fold)

Il modello è stato valutato tramite cross-validation a 5 fold. Di seguito i risultati per ogni partizione:

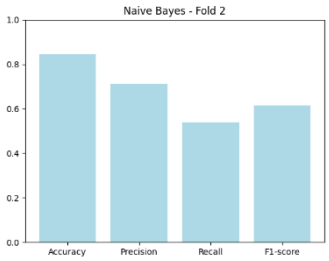
Fold 1

- Accuracy: 0.826;
- Precision: 0.652;
- Recall: 0.511;
- F1-score: 0.573.



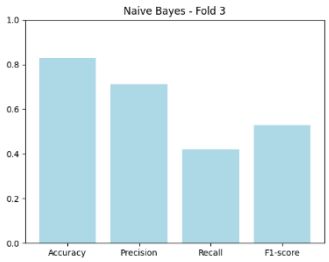
Fold 2

- Accuracy: 0.846;
- Precision: 0.712;
- Recall: 0.540;
- F1-score: 0.614.



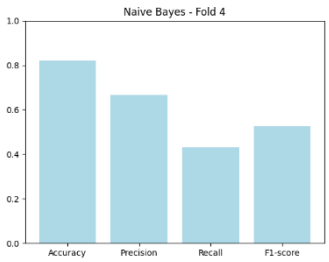
Fold 3

- Accuracy: 0.828;
- Precision: 0.712;
- Recall: 0.420;
- F1-score: 0.529.

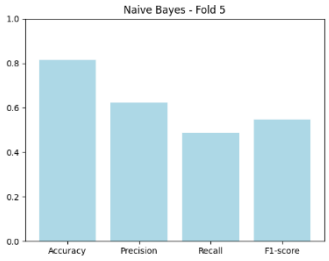


Fold 4

- Accuracy: 0.820;
- Precision: 0.667;
- Recall: 0.432;
- F1-score: 0.524.



Fold 5



- Accuracy: 0.815;
- Precision: 0.623;
- Recall: 0.489;
- F1-score: 0.548.

Variazioni tra i fold

- **Accuracy**:
  Stabile tra 81.5% e 84.6%, confermando la **robustezza del modello** anche con dati parzialmente diversi;
- **Precision**:
  Oscilla tra 0.623 e 0.712, dimostrando una **buona capacità di limitare i falsi positivi** nei vari fold;
- **Recall**:
  Più instabile (range: 0.420 – 0.540), riflette la **difficoltà del modello nel catturare tutti i positivi**, specialmente in fold più sbilanciati;
- **F1-score**:
  Varia tra 0.524 e 0.614. Segue l'andamento del recall, ma meno accentuato, confermando un bilanciamento solo moderato tra precision e recall.

Considerazioni finali

Il Naive Bayes si conferma un **modello semplice ma sorprendentemente efficace**.
La sua capacità di raggiungere una buona accuratezza **senza tuning** e con una **computazione minima** lo rende un'ottima baseline.
Tuttavia, la **limitata capacità di recall** suggerisce prudenza nel suo impiego in contesti dove è fondamentale **non perdere esempi positivi**.

7.1.6 XGBoost

XGBoost è un algoritmo di boosting avanzato basato su **alberi decisionali**.
Costruisce il modello in modo **sequenziale**, correggendo a ogni iterazione gli errori residui dei classificatori precedenti, tramite **ottimizzazione del gradiente**.

Rispetto ai metodi tradizionali, XGBoost introduce **numerose ottimizzazioni**:

- Gestione della **regolarizzazione** per ridurre l'overfitting;
- Strategia di **shrinkage** per evitare aggiornamenti troppo bruschi (tramite il parametro learning_rate);
- Supporto a **subsampling** di dati e feature, per migliorare la generalizzazione;
- Elevata **efficienza computazionale**, grazie all'uso di strutture dati ottimizzate (es. DMatrix).

È un algoritmo particolarmente indicato per problemi complessi con:

- feature numeriche e categoriche;
- rumore nei dati;
- classi sbilanciate.

Iperparametri principali

- **n_estimators**: numero totale di alberi generati. Più alberi => maggiore capacità, ma rischio di overfitting;
- **learning_rate**: tasso di apprendimento, controlla il peso dato a ogni nuovo albero;
- **max_depth**: profondità massima degli alberi, per limitare la complessità strutturale;
- **subsample**: frazione del dataset da usare per ciascun albero. Riduce varianza e migliora la generalizzazione;
- *(Altri iperparametri secondari: colsample_bytree, gamma, reg_alpha, reg_lambda, non modificati in questa analisi).*

Motivazione della scelta

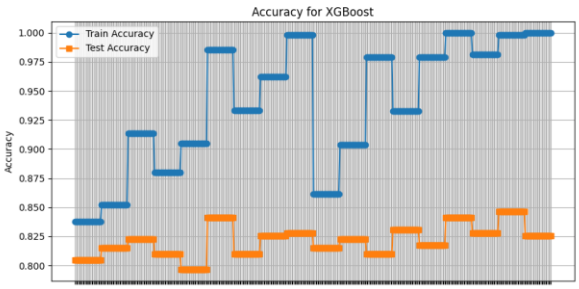
XGBoost è stato selezionato per valutare un approccio di **boosting avanzato**, con le seguenti motivazioni:

- **Capacità di generalizzazione superiore** rispetto ai singoli alberi (grazie al boosting graduale);
- **Robustezza in presenza di rumore** e distribuzioni sbilanciate;
- **Regolarizzazione integrata**, utile per controllare overfitting in contesti reali;
- **Ottime prestazioni empiriche**, spesso superiori agli ensemble classici su problemi strutturati come la classificazione delle preferenze utente.

Grafico delle Accuratezze

Il grafico mostra l'andamento delle accuratezze su **training set** (linea blu) e **test set** (linea arancione) per diverse combinazioni di iperparametri di XGBoost, tra cui:

- n_estimators: {100, 300};
- max_depth: {3, 5, 7};
- learning_rate: {0.05, 0.1, 0.3};
- subsample: {0.7, 0.9, 1.0};
- colsample_bytree: {0.7, 1.0};
- scale_pos_weight: {1, 1.5}.



Osservazioni principali:

- **Overfitting visibile ma gestito:**
  L'accuracy sul training set raggiunge spesso il 100%, ma quella sul test set resta più bassa (~81–84%).
  Questo divario è previsto nei modelli boosting ad alta capacità, ma l'uso combinato di **regolarizzazione (reg_alpha, reg_lambda)**, **subsampling** e **learning rate basso** aiuta a contenerlo.
- **Effetto di learning_rate e n_estimators:**
  L'aumento del numero di alberi (n_estimators) migliora le prestazioni solo se accompagnato da un learning_rate basso.
  Un learning_rate=0.05 con n_estimators=300 tende a produrre risultati stabili e generalizzabili.

- **Contributo della regolarizzazione:**
  L'introduzione di scale_pos_weight=1.5 ha migliorato leggermente il **recall**, compensando lo sbilanciamento tra le classi.
- **Stabilità delle prestazioni:**
  Nonostante l'ampio spazio di ricerca, le combinazioni ben regolarizzate mostrano una **test accuracy stabile**, con valori attorno all'83–84%.

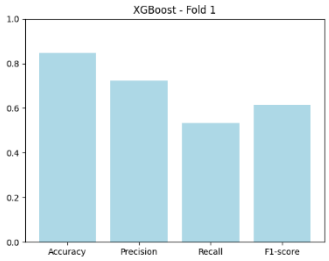
Conclusione:
XGBoost dimostra un buon equilibrio tra accuratezza e capacità di generalizzazione, pur richiedendo attenzione alla scelta degli iperparametri. Le strategie adottate sono dimostrate efficaci nel contenere l'overfitting.

Valutazione tramite Cross Validation (5 Fold)

Per analizzare la robustezza del modello, è stata condotta una validazione incrociata 5-fold. I risultati per ciascun fold sono i seguenti:

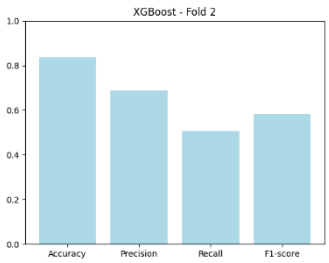
Fold 1

- Accuracy: 0.846;
- Precision: 0.723;
- Recall: 0.534;
- F1-score: 0.614.



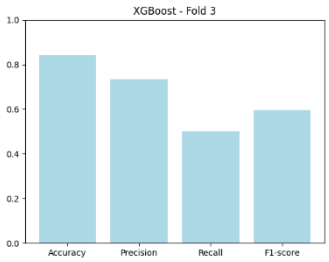
Fold 2

- Accuracy: 0.836;
- Precision: 0.688;
- Recall: 0.506;
- F1-score: 0.583.



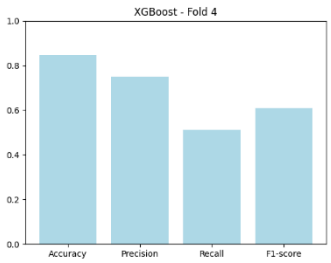
Fold 3

- Accuracy: 0.843;
- Precision: 0.733;
- Recall: 0.500;
- F1-score: 0.595.



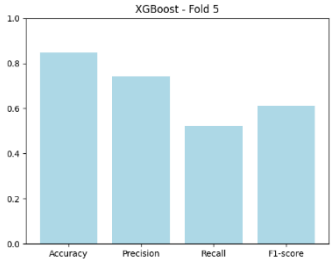
Fold 4

- Accuracy: 0.849;
- Precision: 0.750;
- Recall: 0.511;



- F1-score: 0.608.

Fold 5



- Accuracy: 0.849;
- Precision: 0.742;
- Recall: 0.523;
- F1-score: 0.613.

Variazioni tra i fold

- **Accuracy**: molto stabile tra i fold, con valori tra 0.836 e 0.849. Indica buona capacità generalizzativa.
- **Precision**: valori costantemente sopra 0.68, con un picco di 0.750 (Fold 4). Il modello è accurato nel classificare i positivi.
- **Recall**: variabile tra 0.50 e 0.53. Conferma una tendenza a privilegiare la precisione rispetto alla sensibilità.
- **F1-score**: coerente con gli altri modelli boosting (~0.58–0.61), suggerendo un bilanciamento discreto.

Considerazioni finali

XGBoost si conferma un modello **robusto e performante**, con accuratezza tra le più alte osservate. Mostra una leggera **preferenza per la precisione**, ma con configurazioni adeguate (es. bilanciamento classi), è possibile migliorare il recall.

È adatto come **modello di riferimento finale**, specialmente quando si cerca un compromesso ottimale tra **bias**, **varianza** e **scalabilità**.

7.1.7 Analisi Complessità

Per ogni modello si è scelto di limitare la complessità strutturale e di bilanciare la sensibilità alle classi per mitigare overfitting e sbilanciamenti.

Modello	**Tempo di training**	**Memoria**
Decision Tree	$O(n \cdot d \cdot \log n)$	$O(n)$
Random Forest	$O(t \cdot n \cdot d \cdot \log n)$ ($t = n_{estimators}$)	$O(t \cdot n)$
AdaBoost	$O(t \cdot T_{base})$	$O(t \cdot n)$
KNN	O(1) training; $O(n \cdot d)$ inferenza	$O(n \cdot d)$
Naive Bayes	$O(n \cdot d)$	$O(d \cdot k)$ (k = # classi)
XGBoost	$O(t \cdot n \cdot \log n)$	$O(t \cdot n)$

n = numero esempi, *d* = numero feature, *t* = numero stimatori.

Decision Tree presenta un costo di training di $O(n \cdot d \cdot \log n)$, dove n è il numero di esempi e d il numero di feature, e richiede memoria $O(n)$ è quindi piuttosto scalabile su dataset di medie dimensioni, ma cresce più velocemente se aumentano gli attributi o i campioni.

Random Forest, con t stimatori, ha tempo di training $O(t \cdot n \cdot d \cdot \log n)$ e memoria $O(t \cdot n)$; l'aumento degli alberi migliora la stabilità, ma allunga proporzionalmente i tempi e il footprint in memoria.

AdaBoost richiede $O(t \cdot T_{base})$ tempo, dove T_{base} è il costo di training del classificatore debole (tipicamente un albero poco profondo), e $O(t \cdot n)$ spazio per mantenere pesi e predizioni intermedie. È più leggero di Random Forest finché gli stadi t restano contenuti.

K-Nearest Neighbors non ha fase di training $O(1)$, ma l'inferenza costa $O(n \cdot d)$ per ogni previsione e richiede memoria $O(n \cdot d)$ per conservare l'intero dataset, il che può diventare oneroso in fase di deployment su grandi insiemi di dati.

Naive Bayes scorre linearmente sui dati con $O(n \cdot d)$ sia in tempo che in spazio $O(d \cdot k)$ (con k numero di classi): ciò lo rende estremamente efficiente e adatto a scenari real-time o vincolati in memoria.

XGBoost fonde boosting e struttura ad albero, con tempo $O(t \cdot n \cdot \log n)$ e memoria $O(t \cdot n)$; pur generando modelli molto performanti, conviene mantenere basso t e profondità per contenere l'impatto computazionale.

Le notazioni Big-O riportate per ciascun algoritmo derivano dall'analisi delle operazioni fondamentali descritte nel libro (Poole & Mackworth, capitoli dedicati agli algoritmi corrispondenti ed internet). In sostanza, le formule standard dai testi di riferimento sono state adattate ai parametri (n, d, t).

7.2 Target Piace

Per trasformare il problema del gradimento in una classificazione binaria supervisionata, è stata definita una variabile target denominata **Piace**, che assume valore:

- 1 se il manga ha ricevuto un **punteggio medio (score) maggiore o uguale a 7** da parte dell'utente;
- 0 in tutti gli altri casi.

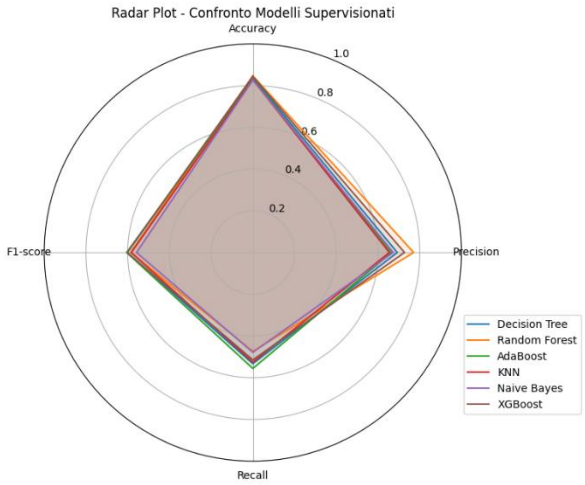
Questa soglia è stata scelta sulla base delle seguenti considerazioni:

- Nei principali portali di valutazione di anime e manga (es. **MyAnimeList**, **AniList**), il punteggio 7 rappresenta una soglia comunemente interpretata come **"buono"** o **"gradito"**, a differenza di voti inferiori che indicano disinteresse o insoddisfazione.
- L'analisi esplorativa del dataset ha mostrato che la **distribuzione dei punteggi** presenta una **curva centrata attorno a 6.5–7**, rendendo la soglia 7 un punto discriminante utile per separare classi bilanciate con sufficiente densità.
- Soglie più alte (es. ≥8) avrebbero ristretto eccessivamente la classe positiva, causando sbilanciamento e perdita di informazioni. Soglie più basse (es. ≥6) avrebbero incluso molti titoli neutri o mediocri, riducendo la qualità predittiva.

7.3 Confusion Matrix e Radar Plot

I grafici riepilogativi forniscono una visione sintetica ma efficace delle prestazioni dei modelli supervisionati analizzati.
In particolare, il **radar plot** mostra le **metriche medie** (Accuracy, Precision, Recall e F1-score) dei sei classificatori:

- **Decision Tree;**
- **Random Forest;**
- **AdaBoost;**
- **KNN;**
- **Naive Bayes;**
- **XGBoost.**



Osservazioni chiave:

- **Random Forest** si conferma il modello con la **precision più alta**, segno della sua tendenza conservativa nel classificare positivi con alta confidenza.
- **Naive Bayes** evidenzia il **recall più elevato**, riuscendo a coprire un maggior numero di positivi a scapito di un numero maggiore di falsi positivi.
- **AdaBoost** e **XGBoost** mostrano un **profilo equilibrato**, con buone performance su tutte le metriche, risultando ideali per applicazioni generaliste.
- **KNN** e **Decision Tree** riportano **metriche più contenute**, ma comunque coerenti e stabili, utili come baseline interpretative.

Questa visualizzazione consente di **valutare visivamente i compromessi tra precisione e sensibilità**, aiutando a selezionare il modello più adatto agli obiettivi: massima precisione, elevata copertura o bilanciamento.

Confusion Matrix

Accanto al radar plot, la **confusion matrix** di AdaBoost mostra il comportamento del modello su un campione di test:

	Predetto: 0	**Predetto: 1**
Reale: 0	248 (TN)	58 (FP)
Reale: 1	13 (FN)	65 (TP)

Analisi:

- Il modello ha **classificato correttamente 248 manga non graditi** (True Negative), ma ha anche segnalato **58 falsi positivi**.
- Dei **78 manga apprezzati**, ne ha riconosciuti **65 come tali** (True Positive), mentre **13 sono stati ignorati** (False Negative).

Questo evidenzia un compromesso ben bilanciato:

- La **quantità di positivi recuperati (recall)** è buona;
- I **falsi positivi** sono contenuti;
- Il modello riesce a **identificare con successo molti manga potenzialmente apprezzati**, mantenendo basso il rumore.

In un contesto di raccomandazione, questo comportamento è auspicabile: **si riduce il rischio di consigliare titoli non graditi**, pur riuscendo a intercettare buona parte di quelli rilevanti per l'utente.

8 Apprendimento non supervisionato (sperimentale)

In questa sezione sono state esplorate alcune tecniche di apprendimento non supervisionato, con l'obiettivo iniziale di segmentare i manga in gruppi omogenei sulla base delle loro caratteristiche strutturali (generi, punteggi, popolarità).
A differenza dell'apprendimento supervisionato, questi algoritmi non utilizzano etichette di gradimento, ma cercano pattern e regolarità direttamente nella distribuzione dei dati.

Questa parte del progetto è stata sviluppata a scopo sperimentale, per valutare se il clustering potesse contribuire alla fase di raccomandazione. Tuttavia, i risultati ottenuti **non hanno portato a insight significativi** né a segmentazioni utili dal punto di vista predittivo. Per questo motivo, la sezione è presentata **come test** piuttosto che come componente centrale del sistema.

Sono stati analizzati tre algoritmi (gli ultimi due sono stati cercati su internet ed implementati anch'essi grazie alle loro rispettive librerie):

- **KMeans**: clustering hard basato su centroidi;
- **Gaussian Mixture Model (GMM)**: clustering probabilistico (soft clustering);
- **Agglomerative Clustering**: metodo gerarchico bottom-up.

I dati sono stati vettorizzati combinando:

- generi (binarizzati);
- metriche numeriche (punteggio medio, rank, popolarità);
- e successivamente ridotti con PCA a due dimensioni per la visualizzazione.

L'esperimento ha permesso di valutare **la coerenza strutturale dei dati**, ma ha evidenziato **una scarsa separabilità**, suggerendo che approcci supervisionati restano preferibili per il compito specifico di predizione del gradimento.

8.1 Risultati dei metodi

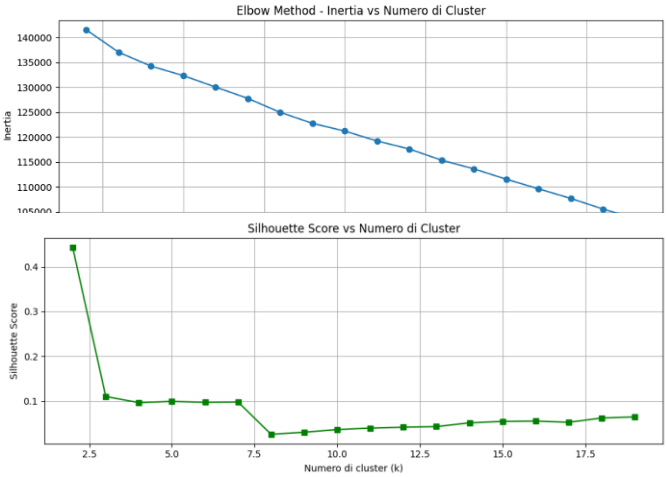
Sono stati sperimentati tre algoritmi di clustering non supervisionato su una rappresentazione vettoriale dei manga costruita a partire da:

- generi codificati binariamente;
- punteggio medio;
- rank;
- popolarità.

Per la rappresentazione visiva e la riduzione della dimensionalità, si è impiegata la **PCA**, proiettando i dati in uno spazio bidimensionale.

8.1.1 KMeans

Il metodo **KMeans** è stato valutato sia con un numero fisso di cluster, sia con selezione automatica del numero ottimale k.

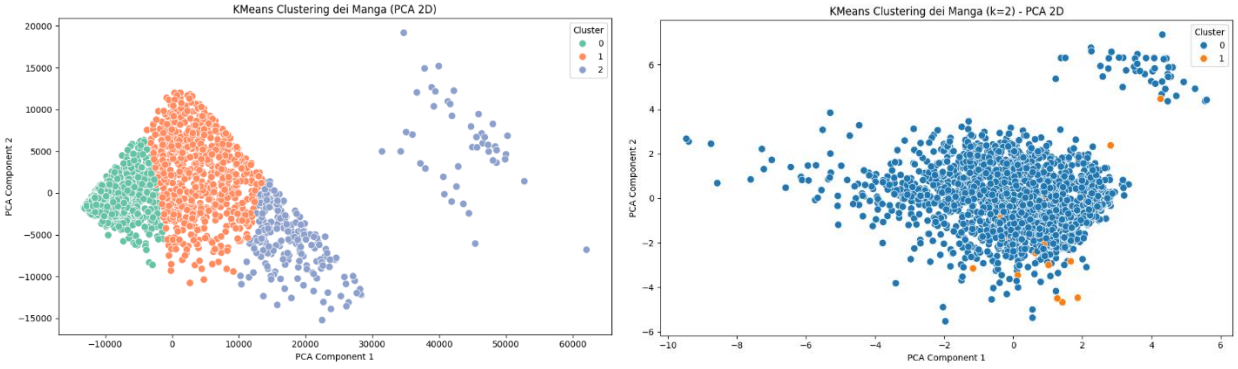


- Il **metodo del gomito** (Elbow Method) ha mostrato un andamento continuo rendendo **difficile stabilire un k ottimale chiaro**.
- L'**indice silhouette** ha indicato **k = 2** come valore con massimo relativo (silhouette ≈ 0.44), sebbene la separazione tra cluster sia risultata comunque poco marcata.

Distribuzione dei cluster (k=2):

- Cluster 0: 1898 manga
- Cluster 1: 18 manga

La visualizzazione su PCA mostra che **i punti sono fortemente sbilanciati verso un solo cluster**, suggerendo **scarsa variabilità strutturale** nel dataset.



8.1.2 Gaussian Mixture Model

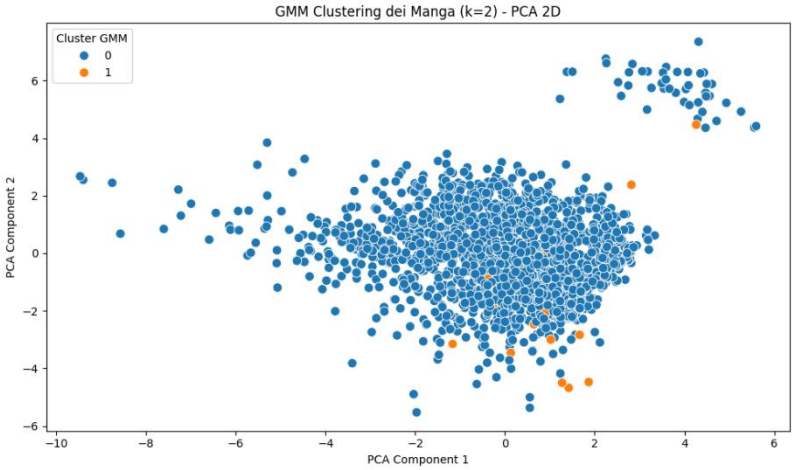
Anche il **modello GMM** ha mostrato un comportamento simile a KMeans in termini di distribuzione.

Distribuzione dei cluster:

- Cluster 0: 1898 manga
- Cluster 1: 18 manga

Nonostante GMM permetta **soft clustering**, nella pratica i risultati si sono ridotti a una partizione molto squilibrata, e la separazione visiva ha mostrato **cluster fortemente sovrapposti**.

Questo suggerisce che i dati non presentano differenze sufficienti per supportare una divisione probabilistica significativa.



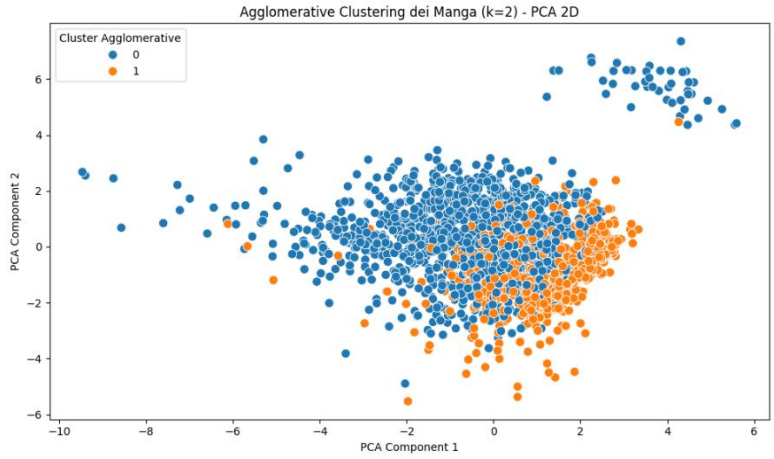
8.1.3 Agglomerative Clustering

Il metodo **Agglomerative**, a differenza degli altri due, ha fornito una suddivisione **più bilanciata**:

Distribuzione dei cluster (k=2):

- Cluster 0: 1238 manga
- Cluster 1: 678 manga

La visualizzazione su PCA mostra una segmentazione leggermente più netta, con una certa coerenza geometrica. Tuttavia, anche in questo caso **i cluster non corrispondono a gruppi evidenti** (es. per genere, popolarità o apprezzamento).



8.2 Considerazioni finali

L'analisi ha evidenziato che:

- Il dataset non supervisionato non mostra **cluster ben separabili**;
- I risultati di **KMeans e GMM** sono stati **molto sbilanciati**, con quasi tutto il dataset assegnato a un unico cluster;
- Solo **Agglomerative Clustering** ha prodotto una suddivisione visivamente interpretabile, ma non particolarmente utile in ottica di raccomandazione.

Pertanto, l'**apprendimento non supervisionato non si è rivelato efficace** per il problema specifico della predizione del gradimento.
Questa sezione è stata dunque condotta **come esperimento**.

9 Risultati e confronto finale

9.1 Analisi simbolico vs statistico

Il progetto ha integrato due approcci distinti per la raccomandazione:

- **Apprendimento simbolico** (motore Prolog): genera raccomandazioni tramite regole deduttive esplicite (es. per autore, genere, stato, premi).
- **Apprendimento statistico** (modelli supervisionati e non supervisionati): apprende dai dati attraverso pattern e correlazioni tra feature e valutazioni.

L'approccio **simbolico** è altamente interpretabile e controllabile, ma è limitato nella capacità di generalizzare oltre quanto codificato. È ideale per esprimere preferenze logiche chiare.

L'approccio **statistico supervisionato** si è rivelato più efficace nella **predizione del gradimento**, con accuracies medie tra **0.828 (KNN)** e **0.847 (Random Forest)**.
I modelli ensemble (Random Forest, AdaBoost, XGBoost) si sono distinti per l'equilibrio tra precisione e recall, fornendo **predizioni robuste e bilanciate** anche in presenza di dati eterogenei.

Al contrario, i metodi **non supervisionati (KMeans, GMM, Agglomerative)** non hanno prodotto risultati significativi:

- I cluster identificati da KMeans e GMM erano **fortemente sbilanciati**, con oltre 98% dei manga assegnati a un unico gruppo.
- L'Agglomerative Clustering ha mostrato **una segmentazione più bilanciata**, ma non coerente con gruppi rilevanti né utile per la raccomandazione.

Ciò conferma che, nel contesto del progetto, **l'apprendimento supervisionato è più adatto** a gestire il problema della previsione delle preferenze utente, mentre il clustering ha mostrato **limiti strutturali**.

9.2 Riflessioni sui modelli supervisionati

Sono stati testati sei modelli principali di classificazione supervisionata:

Modello	Accuracy	Precision	Recall	F1-score
Decision Tree	0.839	0.692	0.533	0.602
Random Forest	0.847	0.771	0.474	0.586
AdaBoost	0.834	0.665	0.556	0.605

Modello	Accuracy	Precision	Recall	F1-score
KNN	0.828	0.655	0.526	0.583
Naive Bayes	0.827	0.673	0.479	0.558
XGBoost	0.844	0.727	0.515	0.603

Considerazioni:

- **Random Forest** ha ottenuto le **migliori performance globali**, con precisione e stabilità elevate;
- **AdaBoost** e **XGBoost** hanno mostrato **ottimi compromessi** tra accuratezza e capacità di generalizzazione;
- **KNN** ha performato bene ma mostra sensibilità alla scala e al numero di vicini;
- **Naive Bayes** è risultato molto rapido ed efficiente, ma meno potente nei casi più complessi;
- **Decision Tree**, pur interpretabile, ha sofferto di una leggera tendenza all'overfitting.

Nel complesso, i modelli ensemble (Random Forest e XGBoost) si confermano i più adatti al task. La classificazione è stata svolta **su feature semplici ma informative**: generi (one-hot), punteggio medio, rank e popolarità.

9.3 Considerazioni sull'apprendimento non supervisionato

Il clustering è stato testato, ma non ha prodotto suddivisioni coerenti né cluster utili.

Anche la riduzione dimensionale con PCA ha mostrato **una struttura piatta e sovrapposta**, con **bassa separabilità intrinseca** dei dati.
Metriche come **silhouette score (< 0.45)** confermano la **scarsa qualità dei cluster**.

Questa parte si è rivelata utile **come test**, ma ha mostrato che **il dataset richiede segnali supervisionati (es. gradimento)** per essere sfruttato efficacemente.

10 Conclusioni

10.1 Riepilogo del lavoro

Il progetto ha sviluppato un sistema di raccomandazione ibrido per manga, integrando approcci **simbolici** (logica Prolog) e **statistici** (modelli di apprendimento supervisionato e clustering esplorativo).
Il sistema, progettato in modo modulare, include le seguenti componenti principali:

- Raccolta e preprocessing dei dati da **MyAnimeList**;
- Generazione automatica della **knowledge base Prolog**;
- **Motore logico-simbolico** per raccomandazioni interpretabili e filtrabili;

- Sei **modelli supervisionati** per predire il gradimento dell'utente a partire da generi e metriche numeriche;
- Un modulo sperimentale di **clustering (KMeans, GMM, Agglomerative)**, con riduzione dimensionale (PCA) e selezione automatica del numero di cluster (Silhouette, Elbow);
- Visualizzazioni finali tramite **matrici di confusione**, radar plot e proiezioni PCA.

L'integrazione dei due paradigmi ha permesso di creare un sistema intelligente, estendibile e interpretabile.
Tuttavia, ha evidenziato che **i modelli supervisionati risultano più efficaci e affidabili** rispetto agli approcci non supervisionati.

10.2 Estensioni future

Il sistema è stato progettato in modo modulare ed estensibile. Alcune possibili evoluzioni includono:

- **Sperimentare altri modelli di classificazione**;
- Integrare una **web app interattiva** per navigare la knowledge base, generare raccomandazioni logiche e visualizzare i cluster in tempo reale;
- Introdurre un **modulo di feedback utente**, per raffinare le predizioni tramite apprendimento continuo.

11 Appendice

11.1 Riferimenti

Fonti teoriche e accademiche

- Poole, D., & Mackworth, A. – *Artificial Intelligence: Foundations of Computational Agents*
  Online: https://artint.info
  Concetti chiave:
  - Logica dei predicati, Prolog
  - Apprendimento supervisionato e clustering
  - Sistemi ibridi (simbolico + statistico)

Librerie e tool di sviluppo

- MyAnimeList API – https://myanimelist.net/apiconfig
- SWI-Prolog – https://www.swi-prolog.org/
- Owlready2 – https://owlready2.readthedocs.io/
- HermiT Reasoner – https://www.hermit-reasoner.com/
- Scikit-learn – https://scikit-learn.org/
- XGBoost – https://xgboost.ai/
- Pandas – https://pandas.pydata.org/
- NumPy – https://numpy.org/
- Matplotlib – https://matplotlib.org/
- Seaborn – https://seaborn.pydata.org/