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

Questa documentazione presenta la progettazione e lo sviluppo di un sistema per l'analisi e la raccomandazione di contenuti nel dominio dei manga.
L'obiettivo principale è combinare **AI Simbolica** e **AI SubSimbolica**, integrando:

- Dati reali provenienti dalle **API** pubbliche di MyAnimeList;
- Tecniche di **apprendimento supervisionato**;
- Un motore di raccomandazione basato su **regole**.

Questo permette di costruire un sistema in grado di fornire raccomandazioni basandosi sulle preferenze dell'utente.

Gli **obiettivi** includono:

- Automatizzare il recupero di dataset personalizzati;
- Generare una knowledge base con fatti strutturati da dati reali;
- Implementare un motore di raccomandazione basato su regole;
- Addestrare modelli di classificazione supervisionata per stimare la probabilità di gradimento;
- Produrre visualizzazioni chiare e metriche comparative tra approcci (le immagini generate sono state inserite nella documentazione).

2 Architettura del sistema

L'architettura del progetto è **modulare, estendibile** ed è suddivisa in componenti indipendenti, ciascuno responsabile di una fase specifica del progetto.
Questa separazione consente di isolare le attività di raccolta dati, ragionamento simbolico e analisi statistica.
Ogni componente ha una propria directory dedicata:

`APPRENDIMENTO/`

Script Apprendimento Supervisionato:

- `main.py` : esegue il flusso ML
- `crea_modello.py` : factory dei modelli ML
- `config_parametri.py` : iperparametri per ogni modello
- `grafici_modelli.py` : radar, heatmap, bar chart, ecc.
- `valutazione_finale.py` : AdaBoost finale e confusion matrix
- `apprendimento_supervisionato.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

`KB/`

Knowledge base Prolog:

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

`PNG/`

Grafici generati:

- Accuratezze, metriche per modello, ecc.

`PYTHON_DATASET/`

Script estrazione dati da MyAnimeList:

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

3 Raccolta e preparazione dei dati

La fase di raccolta e preparazione dei dati è fondamentale per garantire l'efficacia delle analisi successive.
L'obiettivo è ottenere i **dataset**, partendo da fonti reali, sia per i moduli di apprendimento automatico che per quelli simbolici.

3.1 Accesso API MyAnimeList

L'accesso ai dati dell'utente e alla top mille avviene tramite il protocollo **OAuth 2.0 con PKCE**, supportato dalle API di MyAnimeList (MAL). Questo consente di ottenere dati **aggiornati** senza esporre le credenziali dell'utente.
Utilizziamo le seguenti credenziali ottenute da MAL (apiconfig) per effettuare le successive richieste:

- **CLIENT_ID**, ID applicazione fornito da MAL;
- **CLIENT_SECRET**, secret key privata;
- **REDIRECT_URI**, dove ricevere la risposta.

Nel codice del progetto sono presenti le credenziali generate da me medesimo, così da poter eseguire i vari codici senza registrarsi al sito. Inoltre, è stata seguita la seguente guida per impostare l'accesso ai dati.

Il processo si compone di tre fasi:

1. **Autenticazione,** lo script genera un **code_verifier** (per lo scambio del token), costruisce **l'URL** (l'URL completo è visibile nello script) di autorizzazione e **apre automaticamente** il browser per permettere all'utente di concedere i permessi;

```
def generate_code_verifier(length=64):
    chars = string.ascii_letters + string.digits + "-._~"
    return ''.join(secrets.choice(chars) for _ in range(length))
def open_authorization_url(code_verifier):
    auth_url = (f"https://myanimelist.net/v1/oauth2/authorize?"f"response_type=code&client_id={CLI
    print("\nSto aprendo 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;

```
class OAuthCallbackHandler(BaseHTTPRequestHandler):
    authorization_code = None
    def do_GET(self):
        parsed_path = urllib.parse.urlparse(self.path)
        params = urllib.parse.parse_qs(parsed_path.query)
        if 'code' in params:
            OAuthCallbackHandler.authorization_code = params['code'][0]
            self.send_response(200)
            self.end_headers()
            self.wfile.write(b"<h1>Autorizzazione completata. Chiudi 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.

```
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, utili per il motore di raccomandazione.

Il primo **top_manga**, è 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).

L'API consente di estrarre 500 manga alla volta dalla top, aspettiamo 1 secondo dopo ogni richiesta per evitare il rate limit (e che il dispositivo sia contrassegnato come possibile Bot).
Questo dataset fornisce una visione oggettiva dei manga.
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

Il secondo **mangalist,** è 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.

L'API per questo tipo di richiesta consente di estrarre 100 manga alla volta dalla lista utente, come per il dataset precedente anche in questo caso attendiamo 1 secondo per evitare il rate limit.
Questo file è pensato come base per il motore di raccomandazione. Tuttavia, **non viene usato per l'apprendimento supervisionato,** dato che questo tipo di richiesta non consente di estrarre il punteggio globale, rank e popolarità, ed inoltre il dataset è molto piccolo.
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

Consiglio di leggere le due funzioni direttamente dagli script, in top_manga si chiama **get_top_manga,** mentre in user_manga si chiama **get_user_mangalist**.

3.3 Dataset Apprendimento Supervisionato

Il dataset finale "**dataset_ml**", usato per l'apprendimento supervisionato, è costruito dallo script user_estesa.py.
La costruzione del dataset è gestita dallo script, che:

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

Vengono esclusi i manga nello stato "**plan_to_read**", poiché non ancora valutati dall'utente.
Per questo script dato che l'estrazione è molto lenta, e talvolta la connessione viene interrotta, è stato aggiunto un sistema che consente di effettuare un **retry automatico** delle richieste GET con lo stesso offset, così da non ripartire da zero.

Per estrarre le informazioni l'API consente di ottenere un massimo di 100 manga alla volta, come per i dataset precedenti abbiamo un'attesa fra una richiesta ed un'altra di 1 secondo.

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
```

Come per I dataset precedenti consiglio di leggere la funzione direttamente dallo script, **get_user_mangalist_extended** è una versione estesa di **get_user_mangalist** dove ho aggiunto sia un controllo in caso di errore 5xx, generando file temporanei per salvare l'offset attuale e i manga estratti sino a quell'offset, sia una nuova richiesta per ottenere le informazioni aggiuntive necessarie per il machine learning.

4 Knowledge Base

La **Knowledge Base** rappresenta la componente simbolica del sistema, adatto alla definizione di regole e fatti logici interrogabili.
La KB ha tre finalità principali:

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

Questa rappresentazione simbolica consente di eseguire **query** come:

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

4.1 Fatti

La **base di conoscenza** 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**).

Il predicato ***manga/8*** 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 globale;**
- **Popolarità**: indice di **popolarità su MAL;**
- **Stato**: **stato editoriale** (es. finished, publishing);
- **Autori**: lista di **nomi** (come atomi).

Il predicato ***lettura_utente/5*** 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 fatti costituiscono l'intera base interrogabile dal motore, consentendo **inferenze personalizzate**, **raccomandazioni**, **raggruppamenti per generi** e tanto altro ancora.

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']).
```

4.2 Generazione da CSV

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**, scritti nel file **knowledge_base.pl,** generati automaticamente dallo script **crea_kb.py**.

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**.

5 Motore di raccomandazione

Sviluppato in **Prolog**, consente di sfruttare la **conoscenza** 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 in base ai gusti personali dell'utente;
- Valutare la compatibilità di un manga in base alla frequenza dei generi letti;
- Eseguire diverse forme di raccomandazione, come la scoperta di generi nuovi, contenuti premiati o "perle nascoste".

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

5.1 Regole di raccomandazione

Il motore, implementato nel file **system.pl**, definisce un insieme di **regole** che operano sui fatti presenti nella **knowledge base** per generare **raccomandazioni personalizzate**. L'interazione avviene tramite il predicato **menu/0**, che offre un'interfaccia testuale con otto opzioni, 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.

```
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 rimuovere l'underscore ad inizio genere;

```
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.

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 (letti almeno dieci volte, ho scelto questa misura sia per questa regola che per le successive per filtrare i manga con generi poco interessanti per un utente che ne ha letti pochi o abbastanza, confrontando la media dei lettori). 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 generi più letti, mantenendo però la casualità che permette la scoperta di titoli nuovi.

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 rispettano due condizioni: un punteggio medio elevato (≥ 8) e una popolarità bassa (> 1500, più alto è il valore, più i manga saranno poco popolari).

Il processo si articola così:

- Vengono considerati i generi letti almeno dieci volte;
- 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.

```
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 presenti nella lista "plan_to_read" dell'utente, che risultano compatibili.

Opera nel seguente modo:

- Vengono considerati i generi letti almeno dieci volte;
- Per ogni manga nella lista “plan_to_read”, il sistema confronta i suoi generi con i generi preferiti dell’utente;
- Se almeno il 50% (così da consigliare manga con generi poco letti oltre quelli letti) dei generi del manga coincide con i generi preferiti, 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).
```

La funzione *intersection/3* viene utilizzata qui e in regole successive per contare i generi in comune.

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

Consiglia manga non ancora letti che hanno ricevuto un riconoscimento (award_winning) e che condividono almeno due generi con quelli più letti dall’utente.

Il sistema procede come segue:

- Vengono considerati i generi 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 preferiti.

```
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 (rimossi 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

Consiglia manga non ancora letti, ovvero una combinazione di generi letti frequentemente e generi completamente nuovi.

Il funzionamento è il seguente:

- Viene estratta l'intera lista dei generi presenti nella knowledge base;
- Si individuano:
    - I generi preferiti, letti almeno dieci volte;
    - I generi mai letti.
- Per ogni manga non ancora letto, si verifica se:
    - contiene almeno un genere preferito;
    - contiene almeno un genere mai letto.
- I generi dei manga vengono normalizzati prima del confronto;
- Solo i manga che soddisfano entrambi i criteri vengono inclusi nelle raccomandazioni.

```
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, 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;
    - 5–9 letture → 1 punto;
    - < 5 o mai letto → 0 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 ).
```

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

```
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 il titolo di un manga e verifica se è stato già letto/non letto/vorrebbe leggerlo.

Funzionamento:

- L'utente digita il nome del manga;
- Il titolo viene **normalizzato** sostituendo gli spazi con underscore, in modo da renderlo compatibile con quanto scritto nella 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])
    ).
```

- 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.

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** offre un **menu interattivo**, sviluppato all'interno del file system.pl, che consente all'utente di ricevere **suggerimenti** in base alle proprie preferenze.

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, presenta una **lista 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
```

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:

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

6 Apprendimento supervisionato

In questo capitolo descriviamo l'applicazione di tecniche di **apprendimento supervisionato** con l'obiettivo di prevedere se un manga **possa piacere o meno**, 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 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 di raccomandazione**.

6.1 Preprocessing

Dopo l'estrazione del **dataset_ml**, 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), utile per assegnare più generi ad un singolo esempio;
- **Normalizzazione numerica**: **punteggio medio**, **rank** e **popolarità** sono trattati come **variabili numeriche continue;**
- **Oversampling della classe minoritaria** (randomico), dato che l'utente ha più punteggi minori di 7 rispetto a quelli maggiori uguali a 7.

Questa fase avviene nel file **apprendimento_supervisionato.py**:

```
#elimina righe con voti assenti o nulli
df = df[df['Punteggio_Utente'] > 0]

#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])

#per assegnare più generi contemporaneamente a un singolo esempio
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']

#suddivisione train/test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

#bilanciamento del training set (oversampling)
train_df = pd.concat([X_train, y_train], axis=1)
#divide per classe
minority = train_df[train_df['Piace'] == 1]
majority = train_df[train_df['Piace'] == 0]
```

```
#oversampling della classe minoritaria (ci sono pochi esempi con punteggio maggiore di 7)
minority_upsampled = resample(minority, replace=True, n_samples=len(majority), random_state=42)

#combina di nuovo
train_balanced = pd.concat([majority, minority_upsampled])
X_train = train_balanced.drop(columns='Piace')
y_train = train_balanced['Piace']
```

Il file risultante è un **dataset** (matrice di feature (X) con target binario (y)), 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.**

6.2 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 siti di valutazione di anime e manga (es. **MyAnimeList**, **AnimeClick**), il punteggio 7 rappresenta una soglia interpretata come **"buono"**, a differenza di voti inferiori che indicano insoddisfazione;
- L'analisi esplorativa del dataset ha mostrato che la **distribuzione dei punteggi** presenta una **curva centrata attorno a 5.5-6**, rendendo la soglia 7 un punto discriminante utile per separare classi bilanciate con sufficiente densità;
- Soglie più alte (es. ≥8) avrebbero ristretto la classe positiva, causando sbilanciamento. Soglie più basse (es. ≥6) avrebbero incluso molti titoli neutri o mediocri, riducendo la qualità predittiva.

6.3 Classificazione

Per ogni classificatore sono stati selezionati iperparametri specifici al fine di ridurre il rischio di overfitting.
Inizialmente, il problema si è manifestato in modo evidente con il dataset esteso dell'utente *beam_mopanies*, a causa delle dimensioni molto ridotte. In seguito, in misura minore, si è riscontrato un leggero overfitting anche con i dati dell'utente *Stark700*.
Grazie alla documentazione ufficiale di **scikit-learn** (anche della guida utente di sklearn) e di discussioni su **Stack Overflow**, sono stati analizzati e utilizzati i principali iperparametri disponibili, nonostante rimane qualche incertezza sull'efficacia di alcuni di essi.

Anche Kaggle a volte è stato d'aiuto (anche se in minima parte).

6.3.1 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** i sottoinsiemi risultanti.

Iperparametri principali

- **max_depth**: profondità massima dell'albero. Un valore troppo alto può portare a overfitting mentre uno troppo basso può generare underfitting;
- **min_samples_leaf**: numero minimo di campioni richiesti in un nodo foglia, per evitare splitting su pochi dati;
- **class_weight**: pesa le classi in base alla loro frequenza per gestire sbilanciamenti.

Motivazioni della scelta del modello

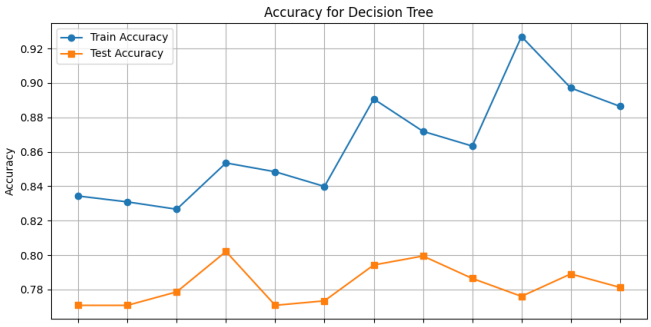
Il Decision Tree è stato selezionato come **modello** per l'analisi supervisionata perché:

- è **semplice da implementare**;
- rappresenta una **baseline efficace**, utile per confrontare le prestazioni con metodi più complessi (come RandomForest o XGBoost);

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:

- **Overfitting contenuto**:
  L'accuracy sul training set mostra valori alti (fino al 93%), ma non raggiunge mai il 100%, indica che l'albero non ha memorizzato completamente i dati.
  La distanza tra training e test accuracy è visibile, ma non esagerata: questo indica che l'overfitting è presente ma **gestito** grazie agli iperparametri introdotti;
- **Effetto della max_depth**:
  Al crescere della profondità, l'accuracy sul training set aumenta (come atteso), ma quella sul test set **non segue lo stesso andamento**.
  Le performance migliori sul test si osservano per profondità intermedie (max_depth=6–8), indicando che un modello troppo profondo ha rumore;
- **Effetto della min_samples_leaf**:
  Valori maggiori di min_samples_leaf riducono la complessità del modello, limitando gli split su insiemi molto piccoli.
  Questo 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**.

Conclusione

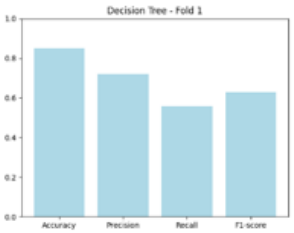
Il grafico conferma che grazie agli iperparametri è stato possibile **contenere l'overfitting** (evidente con il dataset di *beam_mopanies*).

Valutazione tramite Cross Validation (5 Fold)

Per valutare la stabilità e le capacità di predizione 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.

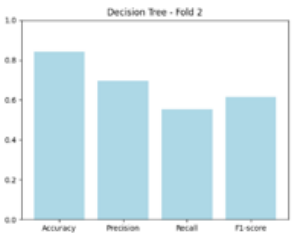
Fold 1

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



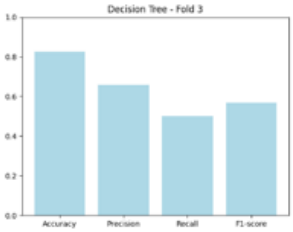
Fold 2

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



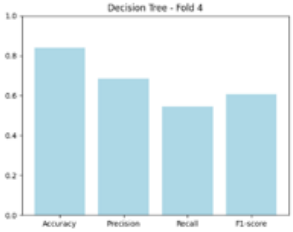
Fold 3

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



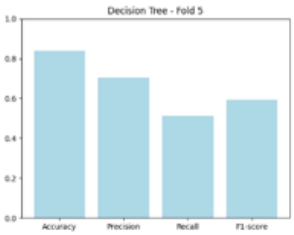
Fold 4

- **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 stabile in tutti i fold, oscillando tra **0.825 e 0.849**. Il valore massimo 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 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. Suggerisce che, quando il modello è più sicuro nella classificazione, tende a meno falsi positivi;
- **Recall**:
  Il **recall è la metrica con maggiore variabilità** tra i fold, passando da **0.500 (Fold 3)** a **0.557 (Fold 1)**. Questo evidenzia come la capacità del modello di identificare correttamente i manga graditi possa risentire della particolare suddivisione del training set;
- **F1-score**:
  L'F1-score mostra variazioni limitate, con valori tra **0.568 (Fold 3)** e **0.628 (Fold 1)**. Segue quella del recall, ma più smussata.

Considerazioni finali

Nel complesso, le prestazioni del Decision Tree risultano **coerenti fra i fold**, con variazioni presenti ma contenute.
Il modello ha una leggera tendenza a **sottostimare i positivi**.

6.3.2 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** e, a ogni nodo, utilizza solo un **sottoinsieme casuale di feature** per determinare la suddivisione.

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

Iperparametri principali

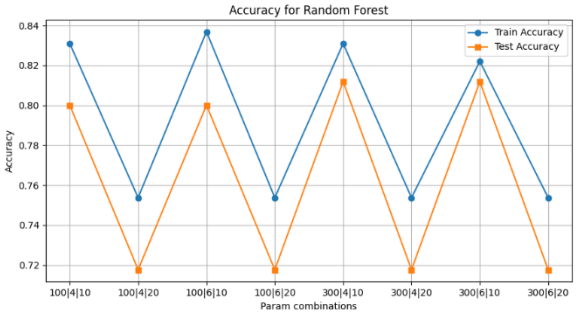
- **n_estimators**: numero di alberi nella foresta. Numero maggiore migliora la stabilità delle predizioni (rendendo l'esecuzione lenta, si nota molto quando viene eseguito lo script);
- **max_depth**: profondità massima degli alberi;
- **min_samples_leaf**: numero minimo di campioni richiesti in un 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 classificatore** dopo il Decision Tree, per analizzare quanto un approccio ensemble migliora la **capacità predittiva** e la **stabilità** del sistema.

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%, indica che il modello non ha completamente memorizzato i dati.
  La distanza tra le due curve è visibile, indicando che l'overfitting è presente, ma **controllato** grazie all'approccio ensemble del modello;
- **Effetto del n_estimators**:
  Aumenti nel numero di alberi tendono a **stabilizzare l'accuracy**, ma non sempre migliorano sul test set.

La stabilità migliora con n_estimators ≥ 300, anche se oltre un certo punto l'incremento è minimo;

- **Effetto della max_depth**:
  Profondità maggiori portano a train accuracy alta, ma anche a una maggiore distanza dal test set, quindi c'è overfitting.
  Le migliori performance sul test si osservano con max_depth=10, risultato bilanciato;
- **Effetto della min_samples_leaf**:
  Valori alti per min_samples_leaf riducono la complessità dei singoli alberi;
- **Test accuracy stabile (~80–84%)**:
  Pur non raggiungendo valori altissimi, la test accuracy si mantiene stabile in quasi tutte le combinazioni di iperparametri, confermando la **robustezza del modello**.

Conclusione

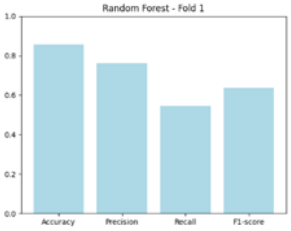
Il grafico conferma che il RandomForest, grazie all'uso combinato di sottoinsiemi casuali, e aggregazione dei risultati finali, **riesce a contenere l'overfitting**. Le accuratezze sul test set indicano che si tratta di un classificatore **affidabile e bilanciato**.

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.

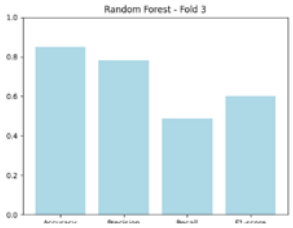
Fold 1

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



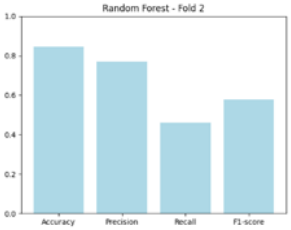
Fold 3

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



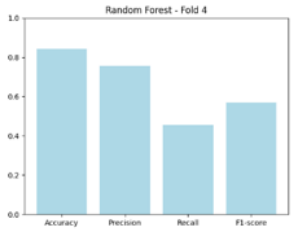
Fold 2

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



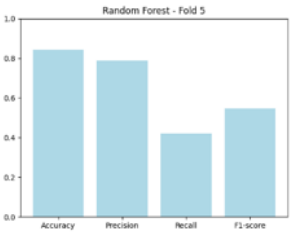
Fold 4

- 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 **stabili**, oscillando tra 0.841 e 0.857. Il modello mantiene buone prestazioni su tutte le partizioni, indicando una capacità di generalizzazione;

- **Precision**:
  Stabile e 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), quindi il modello **ha difficoltà a individuare tutti i manga apprezzati**;
- **F1-score**:
  Si mantiene tra 0.548 e 0.636, seguendo il recall. Le variazioni non sono eccessive e confermano un buon bilanciamento tra precisione e recall.

Considerazioni finali

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

6.3.3 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 molto bassa) per costruire un classificatore forte.

Il funzionamento si basa su un meccanismo:

- 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.

Iperparametri principali

- **n_estimators**: numero di classificatori deboli. Valori alti permettono una maggiore espressività del modello, ma aumentano il rischio di overfitting;
- **learning_rate**: coefficiente che controlla il peso assegnato a ciascun classificatore.

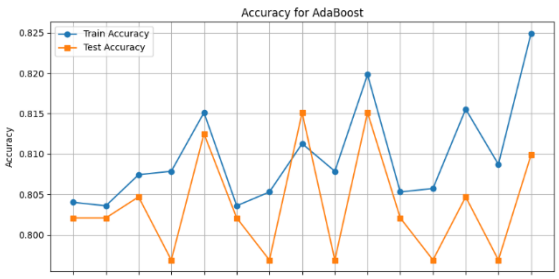
Motivazione della scelta

AdaBoost è stato selezionato per la **capacità di trasformare classificatori semplici in modelli predittivi efficaci**.

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 l'83%, e si mantiene vicina a quella

del test set. Questo comportamento è indicativo di un modello che evita la memorizzazione dei dati;

- **Effetto del n_estimators**:
  L'incremento di n_estimators porta a un miglioramento della train accuracy, ma non sempre ha effetti positivi sulla test accuracy.
  Oltre i 100 stimatori, l'efficacia tende a stabilizzarsi o addirittura a regredire in alcune combinazioni;
- **Effetto del learning_rate**:
  Valori alti (es. 1.0) possono causare variazioni nell'accuracy del test set, notato dal fatto che si adatta molto bene ai dati. I risultati più stabili si osservano con learning_rate tra **0.1 e 0.3**, che garantiscono generalizzazione;
- **Accuratezza test stabile (~80–82%)**:
  Le curve di test accuracy sono poco variabili tra le combinazioni di iperparametri, il modello è **poco sensibile agli iperparametri**, anche che la sua **capacità predittiva è limitata**.

Conclusione

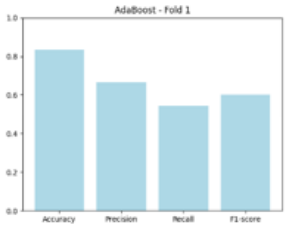
Il grafico mostra che AdaBoost è un **modello bilanciato**, capace di mantenere prestazioni costanti senza overfitting.

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:

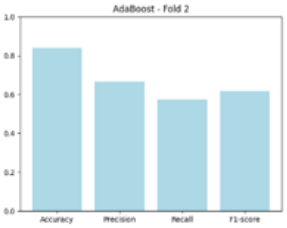
Fold 1

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



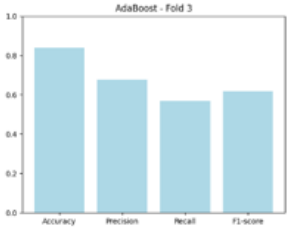
Fold 2

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



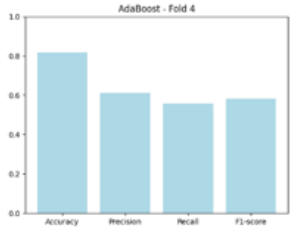
Fold 3

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



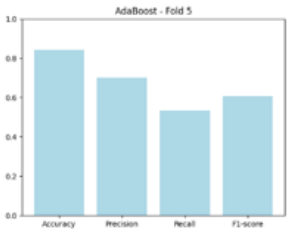
Fold 4

- 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 stabilità (0.817–0.841), 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**;
- **Recall**:
  Più sensibile alla variazione dei dati (0.534–0.575), AdaBoost riesce meglio a catturare gli esempi positivi rispetto al RandomForest;
- **F1-score**:
  Si mantiene tra 0.583 e 0.617, dimostrando un buon equilibrio tra precisione e recall.

Considerazioni finali

AdaBoost è un modello **stabile.**
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.

6.3.4 KNN

Il **K-Nearest Neighbors (KNN)** è un algoritmo di classificazione supervisionata basato sul concetto di **similarità tra esempi**.
Per classificare un nuovo esempio memorizza i dati di training e durante la fase di predizione, **identifica i *k* esempi più vicini** secondo una metrica (ad esempio la distanza euclidea).

Iperparametri principali

- **n_neighbors**: numero di vicini considerati per la classificazione. Valori alti **favoriscono la generalizzazione**.

Motivazioni della scelta del modello

Il **KNN** è utile per valutare l'efficacia della classificazione **basata su prossimità tra istanze**.

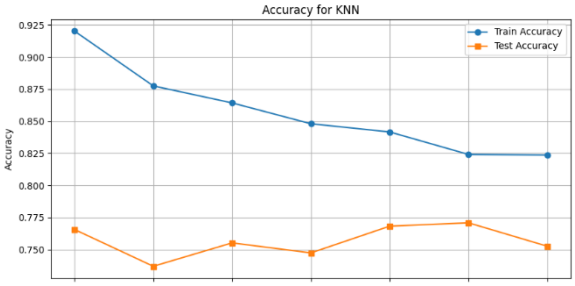
In particolare:

- Consente di analizzare la **sensibilità del sistema** rispetto alla scelta di *k*;
- 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%). Indica un **forte overfitting**: il modello si adatta troppo ai dati locali del training set;
- **Effetto del n_neighbors**:
  All'aumentare di k, la training accuracy si abbassa, 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** con l'aumento di k. Il modello fatica a generalizzare, forse a causa di feature eterogenee.

Conclusione

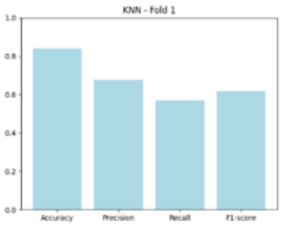
Il KNN si dimostra **sensibile alla scelta di k**, con prestazioni deboli sul test set ed una tendenza all'overfitting per valori piccoli. È inadatto a causa dei **dati sparsi**.

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:

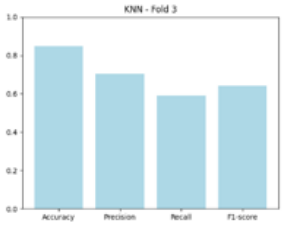
Fold 1

- Accuracy: 0.839;
- Precision: 0.676;
- Recall: 0.568;
- F1-score: 0.617.



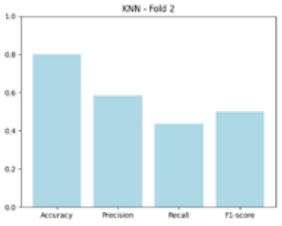
Fold 3

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



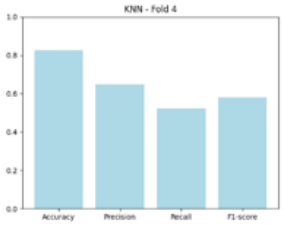
Fold 2

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



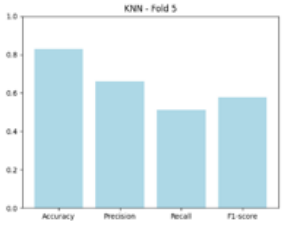
Fold 4

- 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**:
  Stabile (0. 802–0.849), con un leggero calo nel Fold 2. Il modello mostra **una capacità di generalizzazione**, ma non così tanto efficiente;
- **Precision**:
  Variabile da 0.585 (Fold 2) a 0.703 (Fold 3), conferma che **l'efficacia nel classificare correttamente i positivi dipende dalla distribuzione locale dei dati**;
- **Recall**:
  È la 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.

Considerazioni finali

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

6.3.5 Naive Bayes

Il **Naive Bayes** è un insieme di classificatori probabilistici basati sul **Teorema di Bayes**, che assume (in maniera *naïve*) l'**indipendenza tra le feature** condizionatamente alla classe.

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

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

Motivazione della scelta

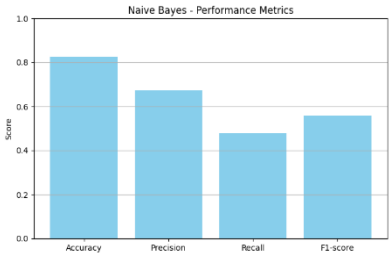
Il Naive Bayes è stato introdotto come **baseline probabilistica interpretabile**, per le seguenti ragioni:

- è tra i modelli più rapidi da addestrare ed eseguire;
- si adatta bene alla rappresentazione binaria dei generi manga;
- la tendenza a classificare positivi con maggiore libertà consente di **non escludere manga potenzialmente graditi**;
- 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:

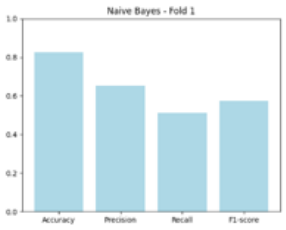
- **Accuracy stabile e buona** (~82–84%):
  Indica che il modello è in grado di classificare correttamente la maggior parte delle istanze, nonostante la semplicità;
- **Precisione elevata rispetto alla complessità del modello**:
  Dimostra che, pur con assunzioni forti di indipendenza tra feature, il modello è in grado di ridurre i falsi positivi;
- **Recall relativamente basso (~48%)**:
  Classificando positivi solo se molto "convinto", con il rischio di escludere alcuni manga;
- **F1-score medio** (~56%):
  Rappresenta un equilibrio accettabile tra precision e recall.

Valutazione tramite Cross Validation (5 Fold)

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

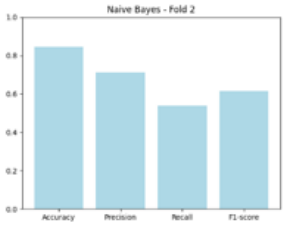
Fold 1

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



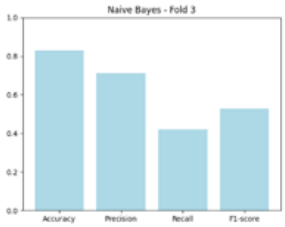
Fold 2

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



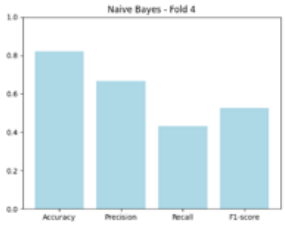
Fold 3

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



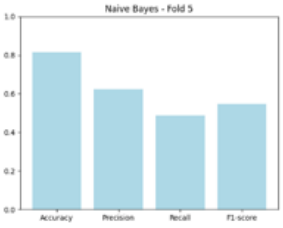
Fold 4

- 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.



Evito di commentare le variazioni tra i fold dato che l'ho già fatto sopra.

Considerazioni finali

Il Naive Bayes si conferma un **modello semplice ma efficace**.
La sua capacità di raggiungere una buona accuratezza lo rende un'ottima baseline.
Tuttavia, il **recall** suggerisce prudenza nel suo utilizzo dove è fondamentale **non perdere esempi positivi**.

6.3.6 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**:

- Riduce l'overfitting usando la **regolarizzazione**;
- Supporto a **subsampling** di dati e feature, per migliorare la generalizzazione;
- **Efficienza computazionale**, grazie all'uso di strutture dati ottimizzate.

È un algoritmo 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 ovvero 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;
- **subsample**: frazione del dataset da usare per ciascun albero. Riduce varianza e migliora la generalizzazione;
- **colsample_bytree:** serve per introdurre la casualità (0.7 indica che usa il 70% delle feature per ogni albero), in parte anche per rpevenire l'overfitting;
- **scale_pos_weight:** migliora il recall.

Motivazione della scelta

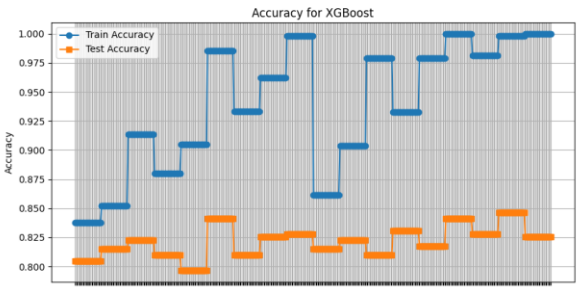
XGBoost è stato selezionato per valutare un approccio di **boosting avanzato**, con le seguenti motivazioni:

- **Capacità di generalizzazione superiore**;
- **Ottimo in presenza di rumore**;
- Utile per controllare overfitting.

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;
- **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**;
- **Stabilità delle prestazioni:**
  Nonostante l'ampio spazio di ricerca, le combinazioni ben regolarizzate mostrano una **test accuracy stabile**, con valori attorno all'83–84%.

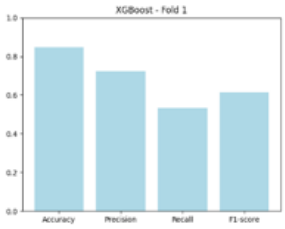
Conclusione:
XGBoost dimostra un buon equilibrio tra accuratezza e capacità di generalizzazione, pur richiedendo attenzione alla scelta degli iperparametri. Le strategie adottate si sono dimostrate efficaci nel contenere l'overfitting.

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:

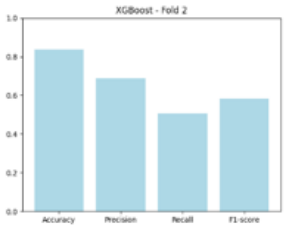
Fold 1

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



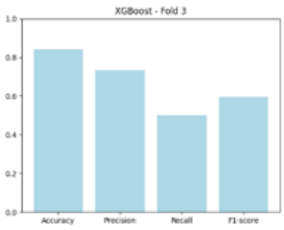
Fold 2

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



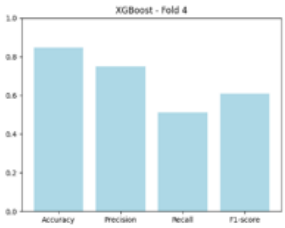
Fold 3

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



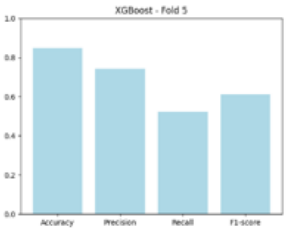
Fold 4

- 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.

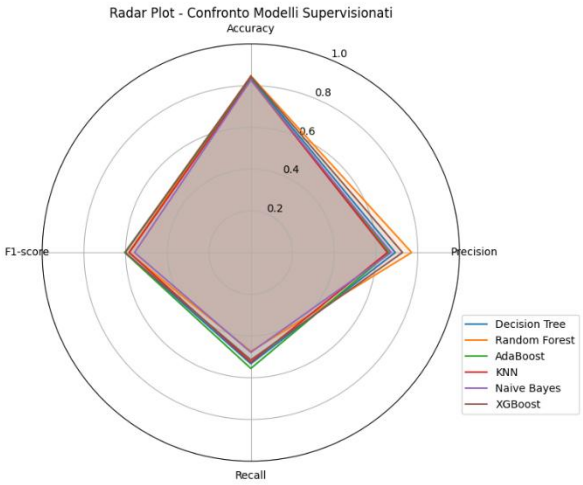
È adatto come **modello di riferimento finale**, specialmente quando si cerca un compromesso ottimale tra **bias**, **varianza** e **scalabilità**.

6.4 Radar Plot, Confusion Matrix e Deviazioni Standard

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

Osservazioni chiave:

- **Random Forest** il modello con la **precision più alta**, segno della sua tendenza 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;
- **KNN** e **Decision Tree** riportano **metriche più contenute**, ma comunque coerenti e stabili, utili come baseline interpretative.



L'approccio **statistico supervisionato** si è rivelato 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.

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:

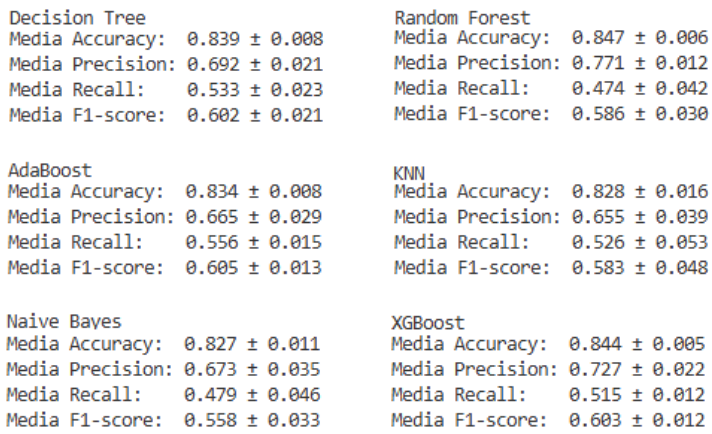
- Il modello ha **classificato correttamente 248 manga non graditi** (True Negative), ma ha anche segnalato **58 falsi positivi**.
- Dei **78 manga apprezzati**, ne ha riconosciuti **65 come tali** (True Positive), mentre **13 sono stati ignorati** (False Negative).

Questo evidenzia un compromesso 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, **si riduce il rischio di consigliare titoli non graditi**, pur riuscendo a intercettare buona parte di quelli rilevanti per l'utente.

L'immagine che segue riassume le performance dei modelli in termini di media e deviazione standard per le metriche. La **deviazione standard (±)** indica quanto i valori ottenuti nei 5 fold della cross-validation si discostano dalla media.



Fornisce una misura della **stabilità** del modello: più è bassa, più il modello è coerente nei diversi sottogruppi del dataset.

- **XGBoost e AdaBoost** hanno deviazioni molto contenute su tutte le metriche (±0.005–0.015), indicando che i risultati sono **affidabili e consistenti** indipendentemente dalla partizione dei dati. Questi modelli sono quindi **robusti**.
- **Decision Tree e Random Forest** mostrano un equilibrio: le deviazioni di precision e accuracy sono basse, mentre quella del recall è leggermente più alta. Quindi hanno una stabilità generale, ma con **variabilità nel riconoscere la classe positiva**.
- **KNN e Naive Bayes**, evidenziano **deviazioni standard elevate**, in particolare per recall e F1-score (fino a ±0.053). Le loro prestazioni **dipendono dalla composizione del training set** e possono comportarsi in modo irregolare, soprattutto nel riconoscimento della classe positiva.

7 Conclusioni

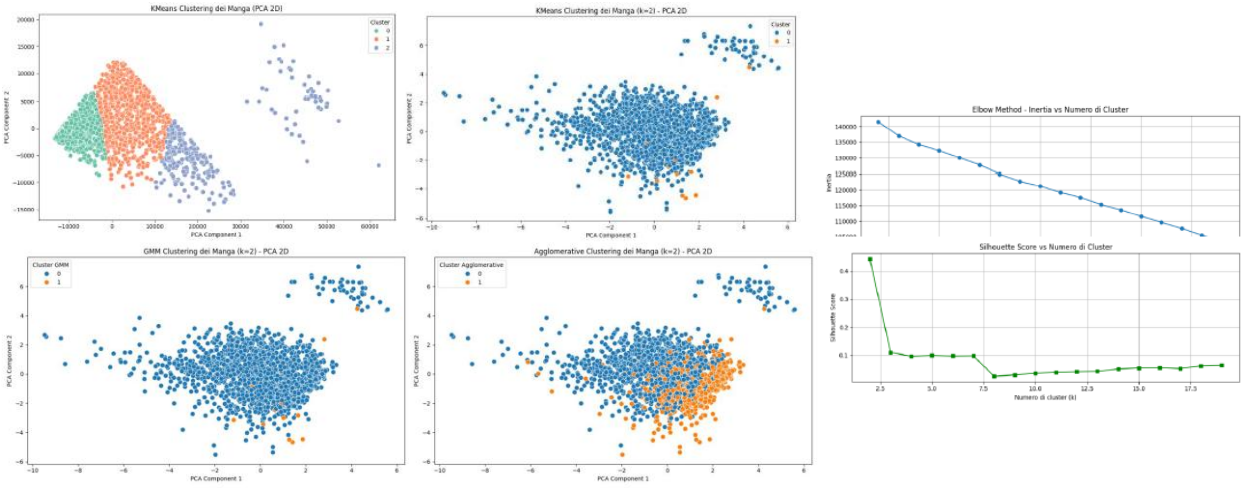
7.1 Estensioni future

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

- **Sperimentare altri modelli di classificazione,** anche deep learning (se possibile);
- Integrare una **web app interattiva** per navigare la knowledge base, generare raccomandazioni logiche in tempo reale;
- Introdurre un **modulo di feedback utente**, per migliorare le predizioni tramite apprendimento continuo.

7.2 Assenza del clustering

Inizialmente ho provato ad implementare tre algoritmi di clustering (K-Means, GMM e Agglomerative clustering), ottenendo pessimi risultati. La maggior parte dei manga veniva assegnato allo stesso cluster, probabilmente a causa dei dati poco separabili. In effetti sfruttando l'elbow method e il silhouette score si otteneva un k uguale a due (inizialmente per il KMeans si è scelto un k=3 arbitrariamente), assegnando comunque almeno il 98~99% dei manga allo stesso cluster. Eccetto per l'Agglomerative Clustering che ha suddiviso "diversamente". Riporto i grafici generati:



Quindi ho scartato l'apprendimento non supervisionato, concentrandomi sul motore di raccomandazione e sull'apprendimento supervisionato.

8 Appendice

Fonti teoriche e accademiche

- Poole, D., & Mackworth, A. – *Artificial Intelligence: Foundations of Computational Agents* Online: https://artint.info

Librerie e tool di sviluppo

- MyAnimeList API – https://myanimelist.net/apiconfig
- SWI-Prolog – https://www.swi-prolog.org/
- 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/