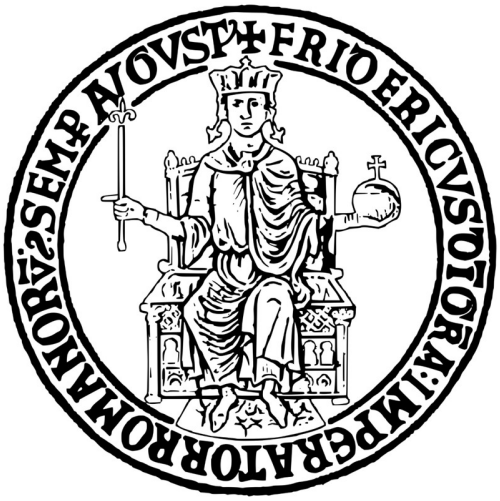
Università degli Studi di Napoli Federico II

Dipartimento di Ingegneria Elettrica e delle Tecnologie dell'Informazione



Approccio Multi-Armed Bandit per Knowledge-based Recommender System con Integrazione Semantica da DBpedia

Studente: Francesco Calcopietro
Matricola: N97000432

Docente: Prof. Luigi Sauro

Anno Accademico 2024/2025

Abstract

Il presente elaborato descrive lo sviluppo di un sistema di raccomandazione basato su conoscenza, potenziato da un approccio Multi-Armed Bandit per la selezione dei film. L'obiettivo è quello di fornire raccomandazioni personalizzate partendo dai gusti di un utente, analizzando i film apprezzati e integrando informazioni semantiche estratte da DBpedia tramite interrogazioni SPARQL. Il sistema sfrutta le proprietà ontologiche dei film (come attori, registi, generi e anni di uscita) per costruire un profilo utente e proporre contenuti affini, bilanciando la politica di exploration ed exploitation. La valutazione sperimentale conferma la validità dell'approccio proposto.

Sommario

Introduzione

Nell'era dell'informazione, il sovraccarico cognitivo dovuto all'enorme quantità di dati e contenuti disponibili online rappresenta una delle principali sfide per utenti e aziende. In questo contesto, i sistemi di raccomandazione svolgono un ruolo cruciale, filtrando in maniera intelligente le informazioni e proponendo agli utenti una selezione personalizzata di contenuti pertinenti, migliorando così l'esperienza complessiva di navigazione, acquisto o fruizione.
I **Recommender System**, o sistemi di raccomandazione, appartengono all'ambito dell'**AI-supported decision making** e sono progettati per supportare gli utenti nei processi decisionali complessi, selezionando tra migliaia di alternative solo quelle rilevanti. Sono oggi utilizzati in moltissimi contesti applicativi: **e-commerce** (Amazon, Alibaba), **entertainment** (Netflix, Spotify), **social media** (Facebook, Instagram), e persino in ambito sanitario, bancario e turistico.
Secondo recenti ricerche, si stima che oltre **il 75% del traffico su Netflix** e più del **35% delle vendite su Amazon** siano direttamente attribuibili ai sistemi di raccomandazione. Questo dato evidenzia come tali strumenti non siano solo accessori, ma elementi centrali nella strategia di engagement e fidelizzazione del cliente.
L'evoluzione di questi sistemi ha portato allo sviluppo di approcci sempre più sofisticati, tra cui **modelli content-based, collaborative filtering, latent factor models** e, più recentemente, **sistemi knowledge-based con supporto semantico**.
In particolare, i **Knowledge-Based Recommender System (KBRS)** integrano informazioni strutturate su utenti e item, come ontologie e knowledge graphs, per migliorare l'accuratezza e la spiegazione delle raccomandazioni.
In questa relazione si presenta uno studio su un sistema KBRS potenziato da tecniche di **Multi-Armed Bandit** (softmax) per ottimizzare la selezione degli item da raccomandare. Il sistema sfrutta la **semantica estratta da DBpedia**, una base di conoscenza strutturata, per rappresentare in maniera ricca i film e confrontarne le caratteristiche in fase di raccomandazione. Attraverso un'analisi teorica e una validazione sperimentale, viene dimostrata l'efficacia dell'approccio in termini di accuratezza.

1.Contesto

Nell'era dell'informazione digitale, la quantità di contenuti disponibili per l'utente finale è cresciuta in modo esponenziale. Di fronte a questa sovrabbondanza informativa, è emersa una crescente necessità di strumenti intelligenti in grado di filtrare, ordinare e personalizzare le informazioni offerte. I **Sistemi di Raccomandazione (Recommender Systems)** rispondono a questa esigenza, fornendo suggerimenti su misura per l'utente, con l'obiettivo di semplificare il processo decisionale in ambiti come l'e-commerce, lo streaming video/musicale, l'e-learning, la pubblicità personalizzata e perfino la medicina personalizzata.

1.1 Classificazione dei Recommender Systems

I sistemi di raccomandazione si possono classificare in diverse famiglie, ognuna delle quali si fonda su logiche e fonti informative differenti:

- **Content-Based Filtering**: suggerisce elementi simili a quelli che l'utente ha già apprezzato in passato, basandosi su attributi del contenuto (es. genere, attori, parole chiave). È indipendente dagli altri utenti ma soffre di una bassa capacità di generalizzazione.
- **Collaborative Filtering**: si basa sul comportamento e sulle preferenze di altri utenti simili. È una delle tecniche più diffuse, ma soffre del new user cold-start problem per nuovi utenti o nuovi elementi.
- **Hybrid Approaches**: combinano più tecniche (content-based, collaborative, knowledge-based, etc.) per bilanciare i vantaggi di ciascuna. Sono flessibili e spesso più efficaci, ma anche più complessi da implementare.
- **Knowledge-Based Recommender Systems (KBRS)**: sfruttano conoscenza esplicita su item e utenti per generare suggerimenti. Non richiedono un numero elevato di interazioni da parte dell'utente e possono operare anche in contesti privi di dati storici, ma necessitano di basi di conoscenza strutturate e accessibili.

1.2 Il ruolo dei KBRS nell'ecosistema moderno

I **KBRS** sono particolarmente utili in settori dove:

- le preferenze sono complesse o soggettive (es. viaggi, sanità, immobiliare);
- non si dispone di grandi quantità di dati impliciti;
- si desidera integrare **conoscenza semantica** estratta da ontologie, grafi di conoscenza o dataset Linked Open Data come **DBpedia**.

L'integrazione della **semantica** consente ai KBRS di superare il "gap semantico" che affligge i sistemi classici, migliorando il significato dei suggerimenti e la loro explainability.

1.3 Dati e numeri sull'impatto dei Recommender Systems

Studi recenti dimostrano l'efficacia concreta dei sistemi di raccomandazione:

- Amazon attribuisce **il 35% delle sue vendite** ai suggerimenti del proprio sistema di raccomandazione (McKinsey, 2021).
- Netflix ha dichiarato che oltre **l'80% dei contenuti visti** dai propri utenti è influenzato da raccomandazioni personalizzate.
- Spotify utilizza sistemi basati su conoscenza e reti neurali per generare playlist che contribuiscono a oltre **il 60% del tempo di ascolto**. (Renascence.io, 2024)

2. Tecnologie e Strumenti Utilizzati

2.1 Dataset MovieLens

Il dataset **MovieLens Small**, utilizzato in questo progetto, è un benchmark consolidato per la valutazione di sistemi di raccomandazione. È distribuito dal **GroupLens Research Group** dell'Università del Minnesota, uno dei centri di ricerca più attivi nello sviluppo di tecnologie per la personalizzazione intelligente.
Questo dataset è una delle principali risorse per lo studio di **sistemi di raccomandazione content-based, collaborative filtering e knowledge-based**, poiché offre un'ampia gamma di informazioni relative a **utenti, film, valutazioni, tag e metadati**.

2.1.1 Struttura dei dati

I file inclusi nel dataset e utilizzati nel progetto sono:

- **movies.csv**
  Contiene informazioni statiche sui film:
  - *movieId*: identificativo numerico univoco per ciascun film;
  - *title*: titolo completo, spesso comprensivo dell'anno;
  - *genres*: elenco dei generi associati, separati da pipe (|).

Queste informazioni sono fondamentali per:
  - classificare i film per genere;
  - filtrare i contenuti per interesse dell'utente;
  - identificare pattern e similarità semantiche tra film.

- **ratings.csv**
  Registra oltre **100.000 valutazioni esplicite** (da 0.5 a 5.0) espresse da 610 utenti su 9724 film. I campi presenti sono:
  - userId
  - movieId
  - rating
  - timestamp

Questo file è il nucleo della componente **user profiling**: permette di estrarre i film apprezzati dall'utente target (rating ≥ 3) da utilizzare nella selezione delle entità semantiche (attori, registi, generi).

- **tags.csv**
  Fornisce tag testuali associati liberamente dagli utenti. Sebbene siano soggettivi, rappresentano concetti semantici che permettono una **descrizione fine-grained** dei contenuti e sono quindi perfettamente adatti all'arricchimento semantico tramite DBpedia.

- **links.csv**
  Mappa ogni movieId agli ID esterni IMDb e TMDb. Anche se non utilizzato direttamente nel progetto, è un asset strategico per:
  - disambiguare titoli;
  - collegarsi a ontologie e knowledge base esterne;
  - espandere le proprietà tramite Linked Open Data.

2.1.2 Caratteristiche chiave

- **Qualità e pulizia dei dati**: MovieLens garantisce dataset privi di dati mancanti, rumorosi o inconsistenti.
- **Supporto a valutazioni esplicite**: le valutazioni da 0.5 a 5.0 facilitano modelli predittivi più precisi rispetto ai feedback impliciti (click, visualizzazioni).
- **Ampia diffusione nella letteratura**: il dataset è ampiamente utilizzato per benchmarking, garantendo confrontabilità dei risultati con modelli esistenti.
- **Adatto alla personalizzazione**: consente la costruzione di profili utenti solidi anche in assenza di contenuti testuali esterni (grazie a tag e generi).
- **Estendibilità semantica**: l'integrazione con identificativi esterni e la struttura relazionale rendono MovieLens ideale per approcci **knowledge-enhanced**, come nel presente progetto.

2.1.3 Motivazioni della scelta

MovieLens è stato scelto perché:

- È **compatibile** con i sistemi di raccomandazione knowledge-based, in quanto offre metadati semantici (tag, generi) pronti per essere estesi tramite DBpedia;
- Contiene dati reali raccolti da utenti autentici, garantendo **realismo e validità sperimentale**;
- È **modulare e scalabile**: permette di estendere facilmente la quantità di utenti e film passando a versioni più grandi (es. MovieLens 1M, 10M) senza cambiare struttura;
- È **ricco e multidimensionale**, supportando approcci ibridi e l'applicazione di tecniche come il **Multi-Armed Bandit** per l'esplorazione ottimale degli item.

2.2 DBpedia

DBpedia è un progetto di riferimento nell'ambito del **Linked Open Data (LOD)** e rappresenta una delle basi di conoscenza più ampie disponibili pubblicamente su web. Il suo obiettivo principale è quello di **estrarre, strutturare e rendere interrogabili le informazioni presenti in Wikipedia**, convertendole in formato RDF (Resource Description Framework).

Grazie a questo processo, DBpedia fornisce dati strutturati su oltre **6 milioni di entità**, inclusi film, persone, luoghi, aziende e opere letterarie. Queste entità sono descritte da **proprietà semantiche** standardizzate e sono interconnesse tra loro mediante relazioni che seguono una logica formale, come ad esempio:

- *dbo:director* → regista di un film
- *dbo:starring* → attori principali
- *dbo:releaseDate* → data di uscita
- *dbo:genre* → genere cinematografico

Ogni entità è rappresentata da un URI univoco, il che consente la **navigazione semantica** e l'interoperabilità tra dataset differenti. Le risorse sono descritte secondo un'ontologia condivisa e consultabili attraverso l'endpoint SPARQL ufficiale di DBpedia (http://dbpedia.org/sparql).

2.2.1 RDF e Web Semantico

L'adozione di **RDF come formalismo standard** consente di modellare l'informazione in modo semplice ma estremamente espressivo. La rappresentazione a **triple RDF** (soggetto – predicato – oggetto) permette di:

- Rappresentare dati eterogenei e distribuiti;
- Collegare risorse diverse in modo formale e coerente;
- Effettuare inferenze semantiche basate su regole e logica descrittiva.

Queste caratteristiche rendono RDF il **linguaggio di base ideale per i Knowledge Graph**, e di conseguenza per i sistemi di raccomandazione che vogliono basarsi sulla **conoscenza esplicita** piuttosto che su semplici correlazioni statistiche.

2.3 SPARQL

SPARQL è il linguaggio standardizzato dal W3C per interrogare basi di conoscenza RDF. A differenza dei linguaggi SQL tradizionali, SPARQL è orientato a pattern semantici: consente di scrivere query dichiarative che cercano **combinazioni di triple RDF**.

Nel caso del progetto proposto, SPARQL è utilizzato per:

- Trovare entità (film, registi, attori,…) collegate semanticamente a quelle apprezzate dall'utente;
- Cercare film basati su proprietà come dbo:starring, dbo:director, dbo:releaseDate e dbo:genre;
- Effettuare filtri semantici (es. solo entità di tipo dbo:Film in lingua inglese);
- Ricercare film per **anno di pubblicazione** o per **contenuto testuale** nei titoli.

Esempio di query usata nel progetto:

```
PREFIX dbo: <http://dbpedia.org/ontology/>
    PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
    SELECT ?actorLabel ?directorLabel WHERE {
     ?film a ?type ;
          rdfs:label ?label .
     FILTER (?type IN (dbo:Film, dbo:Movie))
     FILTER (lang(?label) = 'en')
     FILTER CONTAINS(LCASE(?label), LCASE("%s"))

     OPTIONAL {
      ?film dbo:starring ?actor .
      ?actor rdfs:label ?actorLabel .
      FILTER (lang(?actorLabel) = 'en')
     }

     OPTIONAL {
      ?film dbo:director ?director .
      ?director rdfs:label ?directorLabel .
      FILTER (lang(?directorLabel) = 'en')
```

```
 }
}
LIMIT 5
```

2.3.1 Utilità nei KBRS

I **Knowledge-Based Recommender Systems (KBRS)** si distinguono dagli approcci collaborativi per la loro capacità di **ragionare sulla conoscenza esplicita**. DBpedia, grazie alla sua struttura ontologica e al vasto dominio coperto, rappresenta una risorsa fondamentale per potenziare i KBRS, offrendo:

1. **Copertura semantica**: i film, gli attori e i registi presenti nel dataset MovieLens possono essere **arricchiti semanticamente** da informazioni addizionali, ampliando lo spazio informativo per le raccomandazioni.
2. **Collegamenti tra entità**: DBpedia consente di trovare automaticamente film **simili** basandosi non su rating, ma su caratteristiche condivise (regista, genere, cast, periodo storico).
3. **Supporto all'explainability**: le raccomandazioni possono essere motivate, ad esempio:
   "Consigliato perché diretto da Christopher Nolan, come Inception e Interstellar".
4. **Risposta al problema della cold-start**: per nuovi utenti o nuovi item, DBpedia fornisce **un contesto semantico** che può sopperire alla mancanza di rating o interazioni.
5. **Interoperabilità con altri dataset RDF**: DBpedia può essere combinata con altre fonti di Linked Data (es. Wikidata, YAGO), aumentando la **robustezza e l'affidabilità** del KBRS.

DBpedia rappresenta un **pilastro fondamentale del Web Semantico**. Integrarla in un sistema di raccomandazione basato su conoscenza consente di superare limiti tipici degli approcci puramente statistici. L'uso di SPARQL, RDF e ontologie offre un **approccio strutturato, formale e spiegabile**, in grado di generare raccomandazioni intelligenti, personalizzate e semanticamente rilevanti.

2.3 Apache Jena: uso per le query semantiche

Apache Jena è un framework open source per lo sviluppo di applicazioni basate su **Web Semantico**. È scritto in Java ed è uno degli strumenti più completi per lavorare con **dati RDF**, **ontologie OWL**, e per l'esecuzione di **query SPARQL** su dataset semantici.

Jena fornisce una gamma di API modulari che permettono di:

- Caricare e gestire modelli RDF in memoria o su disco;
- Costruire e manipolare grafi RDF;
- Interrogare sorgenti semantiche tramite SPARQL;
- Collegarsi ad endpoint remoti come DBpedia;
- Eseguire inferenze logiche basate su regole RDFS o OWL.

2.3.1 Uso nel progetto

Nel contesto del sistema di raccomandazione sviluppato, **Apache Jena** è utilizzato principalmente per:

1. **Costruzione dinamica delle query SPARQL**: le query vengono generate automaticamente a partire da parole chiave (es. generi, registi, attori) ottenute dall'analisi del profilo utente.
2. **Connessione all'endpoint remoto di DBpedia**: grazie all'API *QueryExecutionFactory*, il sistema può inviare query SPARQL al servizio remoto di DBpedia (http://dbpedia.org/sparql).
3. **Estrazione dei risultati**: i risultati delle query (sotto forma di *ResultSet*) vengono iterati e convertiti in stringhe leggibili (es. titoli di film o nomi di registi/attori).
4. **Gestione della semantica in tempo reale**: Jena consente di trasformare entità testuali (es. "Ridley Scott") in nodi semantici interrogabili, abilitando la **navigazione del grafo semantico** durante il runtime.

Nel codice Java del progetto, una query SPARQL costruita dinamicamente viene eseguita così:

```
QueryExecution qexec= QueryExecution.service(DBPEDIA_SPARQL_ENDPOINT
.query(QueryFactory.create(queryString)).build();
ResultSet results = qexec.execSelect();
```

L'interfaccia *QueryExecution* permette di ottenere risultati in modo iterativo e sicuro, supportando anche query più complesse e filtri semantici.
L'integrazione di Apache Jena nel progetto comporta numerosi vantaggi:

- **Modularità**: le API di Jena permettono di separare nettamente la logica di costruzione delle query, l'esecuzione e il parsing dei risultati.
- **Affidabilità**: Jena è stabile, largamente documentato e supportato dalla comunità Apache, risultando ideale per progetti accademici e industriali.
- **Interoperabilità**: consente l'integrazione con altri strumenti semantici come Fuseki, OWLAPI, o librerie RDF in altri linguaggi (es. RDFLib in Python).
- **Performance**: grazie alla possibilità di caching e stream processing, Jena è adatto anche a scenari con grandi volumi di dati semantici.

Apache Jena costituisce il **ponte tra il sistema Java e l'universo del Web Semantico**, permettendo l'integrazione fluida e dinamica di dati strutturati

provenienti da DBpedia. Grazie a Jena, è possibile interrogare basi di conoscenza distribuite con flessibilità e robustezza, trasformando semplici stringhe in concetti navigabili e relazionabili semanticamente.

2.4 Linguaggio di programmazione: Java e librerie utilizzate

Il progetto è stato interamente sviluppato in **Java**, linguaggio scelto per una serie di motivazioni tecniche e metodologiche:

- **Compatibilità diretta con Apache Jena**, il principale framework semantico impiegato;
- **Robustezza** e gestione sicura della memoria, che garantiscono stabilità anche in ambienti complessi come quelli distribuiti;
- **Portabilità**: il bytecode Java è eseguibile su qualsiasi piattaforma dotata di JVM;
- Ampia disponibilità di **librerie open source**, anche per attività connesse come parsing CSV, networking o elaborazione dei dati.

2.4.1 Struttura del progetto

Il sistema di raccomandazione è costruito come **applicazione a linea di comando**, composta principalmente da:

- Una classe centrale: KnowledgeBasedRecommender.java, che implementa il flusso completo di raccolta dati, analisi, interazione con DBpedia, generazione delle raccomandazioni e valutazione del suggerimento.
- Il caricamento dei dati avviene tramite lettura dei seguenti file:
  - *movies.csv*: mappa ID film → titolo e generi;
  - *ratings.csv*: mappa ID utente e film → rating;
  - *tags.csv*: associazione dei tag testuali ai film.
- I dati sono gestiti in memoria tramite strutture *Map*, *Set*, *List*, con uso intensivo delle **Stream API** per trasformazioni e filtri.

2.4.2 Librerie utilizzate

Oltre ad Apache Jena, sono state integrate alcune librerie core di Java:

Libreria	**Funzione**
java.util.*	Collezioni (Map, Set, List), Random
java.io.*	Lettura file CSV
java.util.stream.*	Filtraggio, mapping e riduzione dati

Libreria	Funzione
org.apache.jena.query.*	Costruzione, esecuzione e parsing di query SPARQL
org.apache.jena.sparql.engine.http.*	Connessione a endpoint remoti SPARQL

2.4.3 Approccio modulare

Le funzionalità del sistema sono suddivise in metodi coesi, ciascuno dedicato a una responsabilità specifica:

- **loadMovieLensData()**, **loadTagsData()**, **loadRatingsData()**: per l'inizializzazione e caricamento dei dati (film, tag e rating);
- **extractTopGenres()**, **extractTopTags()**, **extractTopEntities()**, **extractAllGenres()**, **extractAllTags()**: per la costruzione del profilo utente basato sulle preferenze;
- **queryDbpediaProperties()**: per l'interrogazione semantica di DBpedia su attori e registi, dato un titolo specifico di un film;
- **unifiedSparqlSearch()**: per l'esplorazione della base di conoscenza filtrando per aspetti preferiti (generi, tag, attori, registi, anni);
- **recommendSoftmax()**: per la selezione finale del film tramite strategia Softmax (approccio Multi-Armed Bandit);
- **computeEntityError()**: per la valutazione dell'accuratezza della raccomandazione.

Questa modularità assicura una chiara separazione delle responsabilità, facilitando test, refactoring e potenziali estensioni (es. interfacce grafiche o API REST).

2.5 Multi-Armed Bandit (MAB)

Il problema del **Multi-Armed Bandit (MAB)** rappresenta uno dei fondamenti teorici più rilevanti nell'ambito del decision making sequenziale e dell'apprendimento online. È un modello astratto utilizzato per descrivere situazioni in cui un agente deve prendere decisioni ripetute nel tempo con l'obiettivo di **massimizzare la ricompensa cumulativa**, pur non avendo inizialmente informazioni complete sull'ambiente.

Nel modello classico, si immagina un giocatore di fronte a più **slot machine** (da cui il termine "multi-armed bandit"), ognuna delle quali fornisce una ricompensa secondo una distribuzione probabilistica sconosciuta. Il giocatore, ad ogni iterazione, deve decidere quale "braccio" azionare. Dopo ogni azione osserva una ricompensa e può

aggiornare la propria conoscenza per le iterazioni successive. Ad ogni arm i è associato il valore atteso $\mu_i = E(D_i)$.
$i* = argmax_{i \in K}$ è l'arm con valore atteso $\mu*$ massimo (ovvero l' arm migliore).
L'esecuzione di MAB produce una traccia a1,r1, … , al,rl, … aT,rT.
Al round t viene selezionato un arm a_t che ritorna un reward r_t secondo la distribuzione D_{a_t}.
Intuitivamente, lo scopo è quello di trovare una strategia di pull che massimizza il *cumulative reward*

$$R_T = \sum_{x=1}^{T} r_x$$

2.5.1 Applicazione nel Recommender System

Nel contesto di un sistema di raccomandazione, ogni "braccio" del problema **Multi-Armed Bandit** (**MAB**) rappresenta un film candidato. A ogni iterazione, il sistema sceglie un film da suggerire cercando di bilanciare tra **exploration** (scoperta di nuovi item) ed **exploitation** (valorizzazione delle preferenze già note).
Nel progetto, questa logica è implementata nel metodo:

private String recommendSoftmax(List<String> movies, Map<String, Set<String>> filmAspectMap)

Questo metodo adotta la **strategia Softmax**, che assegna a ciascun film una probabilità di essere selezionato in funzione del numero di aspetti (caratteristiche semantiche) che soddisfa.
Dato un insieme di *n* film candidati, a ciascun film *i* viene assegnato un punteggio sis_isi, pari al numero di aspetti preferiti soddisfatti (es. generi, tag, attori, registi, anno).
La **probabilità** di selezionare il film *i* è calcolata secondo la formula:

$$P(i) = \frac{e^{\alpha * s_i}}{\sum_{j=1}^{n} e^{\alpha * s_j}}$$

dove:

- s_i: punteggio grezzo del film *i* (numero di aspetti soddisfatti);

- α: parametro di temperatura (> 0), controlla quanto il sistema favorisce i film con punteggi più alti (exploit) o mantiene varietà nella scelta (explore).

Pertanto, si avrà il seguente comportamento:

- Se α→0, allora $P(i) \approx \frac{1}{n}$ scelta quasi casuale (massima esplorazione);
- Se α→∞ allora P(i)≈1 per il miglior film: scelta deterministica (massimo sfruttamento);
- Per valori intermedi di α, si ottiene un bilanciamento tra exploration ed exploitation.

Dopo aver assegnato a ciascun film candidato una probabilità di selezione secondo la distribuzione **Softmax**, la scelta effettiva del film da raccomandare viene effettuata tramite un meccanismo stocastico. Tale meccanismo realizza un campionamento **secondo la distribuzione discreta delle probabilità calcolate**, modellando una **estrazione casuale pesata**. La sezione di codice:

```
double rand = random.nextDouble();
double cumulative = 0.0;
for (var entry : probabilities.entrySet()) {
   cumulative += entry.getValue();
   if (rand <= cumulative) {
      return entry.getKey(); // seleziona il film
   }
}
```

realizza un algoritmo di **campionamento proporzionale alla probabilità cumulata**. Il principio è il seguente:

1. **Generazione di un numero casuale** r ∈ [0, 1), che rappresenta un punto scelto a caso su un asse continuo.
2. **Costruzione della distribuzione cumulativa**: si procede a sommare le probabilità dei film una alla volta.
3. **Selezione del primo film** il cui intervallo cumulativo include r. Formalmente, si ricerca il minimo indice k tale che:

$$\sum_{i=1}^{k} P(i) \geq r$$

Dove *P(i)* è la probabilità (softmax) associata al film *i*.

Questa strategia è applicata ai film ottenuti da **DBpedia**, interrogata tramite query SPARQL costruite sulle preferenze espresse dall'utente nel dataset MovieLens. La raccomandazione finale è quindi:

- **Semantically-driven**: ogni film è valutato in base alla sua coerenza semantica con il profilo utente.
- **Stocastica ma ponderata**: grazie a Softmax, anche film meno evidenti hanno possibilità di essere suggeriti, migliorando la varietà.

I vantaggi dell'approccio Softmax in un contesto knowledge-aware sono:

- **Personalizzazione dinamica**: favorisce i film più coerenti senza escludere del tutto gli altri.
- **Integrazione semantica**: sfrutta aspetti simbolici provenienti da DBpedia (non solo rating numerici).
- **Controllabilità**: grazie al parametro α\alphaα, il comportamento può essere facilmente calibrato.

3. Architettura del Sistema

Il sistema di raccomandazione sviluppato si articola in un'architettura modulare, composta da vari livelli funzionali che interagiscono per generare suggerimenti personalizzati, fondati sia su dati quantitativi (i rating MovieLens) sia su conoscenza semantica (DBpedia).

3.1 Panoramica dei moduli principali

Modulo	Descrizione
Data Loader	Carica i file CSV di MovieLens (*movies.csv*, *ratings.csv*, *tags.csv*) e costruisce le strutture dati necessarie per il filtraggio iniziale.
User profiler	Analizza i film apprezzati dall'utente (rating ≥ 3) e costruisce il profilo semantico: generi, tag, attori, registi, anni preferiti.
Motore di Query SPARQL	Esegue interrogazioni a DBpedia per ottenere entità correlate ai film apprezzati, come *dbo:starring*, *dbo:director*, *dbo:releaseDate*.
Costruttore candidati	Genera la lista dei film candidati unendo i risultati SPARQL ottenuti per tag, generi, attori, registi e intervalli temporali preferiti.
Modulo di Raccomandazione	Applica la strategia Softmax sui candidati DBpedia, selezionando un film in modo probabilistico in base al numero di aspetti preferiti soddisfatti. Il parametro α controlla il bilanciamento tra esplorazione e sfruttamento.

3.1.1 Caricamento e Preparazione dei Dati

I dati provengono dal dataset **MovieLens Small**, composto da quattro file principali:

- *movies.csv*: contiene ID del film, titolo e generi.
- *ratings.csv*: include i voti dati dagli utenti ai film.
- *tags.csv*: contiene i tag assegnati dagli utenti ai film.
- *links.csv*: non utilizzato in questa implementazione, ma utile per future estensioni con collegamenti esterni (IMDB, TMDB).

I dati sono caricati tramite i metodi:

- *loadMovieLensData()*
- *loadTagsData()*

- *loadRatingsData()*

Questi metodi costruiscono tre mappe principali:
- *movieIdToTitle*: mappa l'ID al titolo del film.
- *movieTags*: mappa l'ID ai tag associati.
- *movieGenres*: mappa l'ID ai generi.

Inoltre, il metodo *loadRatingsData()* identifica i **film apprezzati** dall'utente target (rating ≥ 3.0).

3.1.2 Costruzione del Profilo Utente

Per generare raccomandazioni personalizzate, viene costruito un **profilo utente** basato su:
- **Top 3 generi** preferiti (calcolati tramite *extractTopGenres()).*
- **Top 3 tag** preferiti (*extractTopTags()).*
- **Top 3 attori e registi** estratti dinamicamente da DBpedia, tramite proprietà starring e director, con metodo *queryDbpediaProperties().*

3.1.3 Espansione della Conoscenza tramite DBpedia

Per ogni entità (genere, tag, attore, regista, anno) si effettuano query SPARQL unificate verso DBpedia per individuare film affini:

- *unifiedSparqlSearch()* → esegue una singola query parametrica che integra filtri su generi, tag, attori, registi e anni;

Tutti i risultati vengono aggregati in un insieme unico di film candidati, ciascuno etichettato con gli aspetti semantici che soddisfa.

3.1.4 Selezione del Film con Multi-Armed Bandit

La selezione finale tra i film candidati è affidata alla strategia **Softmax**, implementata nel metodo *recommendSoftmax(),* che:

- Calcola, per ciascun film, una **probabilità** in base al numero di aspetti semantici soddisfatti (generi, tag, attori, registi, anno).
- Seleziona il film da raccomandare in modo **stocastico** secondo tali probabilità.

Il rating di ciascun item si baserà interamente sulla **coerenza semantica** con il profilo dell'utente.

3.2 Strategia di caching per ottimizzare le query

Una delle sfide principali nei sistemi Knowledge-Based che integrano fonti esterne come DBpedia è rappresentata dalla **latenza** delle interrogazioni SPARQL. Per ridurre drasticamente i tempi di esecuzione e il carico sull'endpoint remoto, è stata implementata una **strategia di caching** interna al sistema.
Nel dettaglio:

- È stato utilizzato un oggetto *Map<String, List<String>> dbpediaCache* per memorizzare i risultati delle query SPARQL.
- Ogni chiave della cache è costituita dalla concatenazione di titolo del film e proprietà (*title_property*) in modo da garantire **unicità**.
- Prima di inviare una query all'endpoint, il sistema verifica se la richiesta è già presente nella cache.
- In caso positivo, restituisce direttamente i dati salvati, evitando la chiamata remota.
- In caso contrario, effettua la query, salva il risultato in cache e lo restituisce.

Questa strategia ha portato i seguenti **vantaggi**:

- **Riduzione significativa del numero di richieste HTTP** verso DBpedia.
- **Incremento delle performance**, in particolare nelle iterazioni sui film apprezzati.
- **Maggiore robustezza del sistema** in caso di instabilità dell'endpoint.

La cache è volatile (in-memory), ma potrebbe essere estesa a un livello persistente per future esecuzioni.

4. Integrazione Semantica con DBpedia

4.1 Costruzione delle Query SPARQL

L'integrazione semantica si realizza mediante **interrogazioni SPARQL** all'endpoint pubblico di **DBpedia**, un dataset RDF che rappresenta concetti enciclopedici estratti da Wikipedia.
Il sistema genera dinamicamente query SPARQL per estrarre informazioni semantiche sui film, come attori (*dbo:starring*), registi (*dbo:director*), generi (dbo:genres) e date di uscita (*dbo:releaseDate*).
Un esempio di query SPARQL per ottenere gli attori e registi di un film è:

```
String query = String.format("""
    PREFIX dbo: <http://dbpedia.org/ontology/>
    PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
    SELECT ?actorLabel ?directorLabel WHERE {
```

```
  ?film a ?type ;
      rdfs:label ?label .
  FILTER (?type IN (dbo:Film, dbo:Movie))
  FILTER (lang(?label) = 'en')
  FILTER CONTAINS(LCASE(?label), LCASE("%s"))

  OPTIONAL {
   ?film dbo:starring ?actor .
   ?actor rdfs:label ?actorLabel .
   FILTER (lang(?actorLabel) = 'en')
  }

  OPTIONAL {
   ?film dbo:director ?director .
   ?director rdfs:label ?directorLabel .
   FILTER (lang(?directorLabel) = 'en')
  }
 }
 LIMIT 5
 """, cleanTitle);
```

In questa query:

- Si dichiarano i prefissi per abbreviare gli URI:
  - dbo: per le proprietà dell'ontologia di DBpedia, come dbo:starring e dbo:director;
  - rdfs: per accedere alle etichette leggibili tramite rdfs:label.
- Si cerca una risorsa ?film il cui rdfs:label contenga (in modo case-insensitive) il titolo passato dal codice Java.
- Si filtrano solo le entità di tipo dbo:Film o dbo:Movie, escludendo altri tipi (es. libri o episodi).
- Si include facoltativamente:
  - l'attore collegato al film tramite dbo:starring, con la relativa ?actorLabel;
  - il regista collegato tramite dbo:director, con la relativa ?directorLabel.
- Si richiedono solo etichette in lingua inglese, sia per il film che per attori e registi.
- Si limita il risultato a **massimo 5 corrispondenze**, per contenere la dimensione della risposta.

4.2 Trattamento dei Titoli: Normalizzazione ed Escape

I titoli dei film nel dataset MovieLens non sempre corrispondono esattamente ai label presenti su DBpedia. Per incrementare la **robustezza delle query**, il sistema applica una serie di normalizzazioni:

- Rimozione delle **virgolette**, **backslash**, **caratteri speciali**;
- Eliminazione di elementi tra parentesi, come l'anno o "a.k.a.";
- Trim di eventuali spazi o simboli finali.

Questa normalizzazione è visibile nel metodo *queryDbpediaProperties():*

```
String cleanTitle = title
  .replaceAll("\\(a\\.k\\.a\\..*?\\)", "")
  .replaceAll("\\(.*?\\)", "")
  .replaceAll("-.*", "")
  .replaceAll("\"", "")
  .trim();
```

4.3 Raccolta di Proprietà Semantiche

Il sistema estrae **conoscenza semantica** utile alla profilazione dell'utente da 5 proprietà:

Proprietà RDF	**Obiettivo**
dbo:starring	Attori principali
dbo:director	Registi
dbo:releaseDate	Anno di uscita
rdfs:label	Titoli dei film (generi, tag, ecc.)
dbo:Film (tipo)	Identificazione delle entità come film
dbo:movie	Identificazione delle entità come movie

Queste proprietà vengono interrogate per ogni film apprezzato (rating ≥ 3.0).

4.4 Fallback Semantico tramite FILTER CONTAINS

La **copertura semantica** non è sempre garantita su DBpedia. Per evitare errori dovuti all'assenza di una corrispondenza esatta, il sistema adotta un **meccanismo di fallback** che utilizza:

FILTER CONTAINS(LCASE(?label), LCASE("film_title"))

Questo filtro consente di recuperare anche etichette **parzialmente corrispondenti**, migliorando la **tolleranza ai nomi non standardizzati**.
È una strategia particolarmente utile per titoli con articoli, sottotitoli o varianti linguistiche.
I vantaggi dell'integrazione semantica sono:

- **Arricchimento informativo**: si ottengono metadati che il dataset MovieLens non fornisce esplicitamente.
- **Generalizzazione**: possibilità di raccomandare anche film non presenti nel dataset originario.
- **Personalizzazione profonda**: grazie alle entità estratte da film apprezzati (attori, registi, generi, anni).

5. Algoritmo di Raccomandazione

5.1 Costruzione del profilo utente (generi, tag, attori più frequenti)

Il profilo dell'utente è costruito a partire dalle preferenze esplicite espresse nei rating contenuti nel file *ratings.csv*. In particolare, vengono considerati "apprezzati" tutti i film valutati con un punteggio maggiore o uguale a 3 dall'utente target (con identificativo x). Su questi film viene poi eseguita un'analisi di frequenza per ricavare:

- **I generi più frequenti**, estratti dal file *movies.csv* (colonna *genres*) e suddivisi per il separatore |.
- **I tag più frequenti**, ricavati dal file *tags.csv*, e normalizzati tramite operazioni di pulizia testuale per evitare ambiguità (ad esempio, rimozione di virgolette o descrizioni aggiuntive).
- **Attori e registi più ricorrenti**, ottenuti interrogando DBpedia con query SPARQL mirate (proprietà *dbo:starring* e *dbo:director*). Le etichette testuali dei risultati vengono anch'esse normalizzate.
- **Anni di uscita più ricorrenti,** ottenuti interrogando DBpedia con query SPARQL mirate (proprietà *dbo:releaseDate*).

Il risultato finale di questo processo è un profilo che sintetizza le principali caratteristiche preferite dall'utente target, utile per guidare la selezione dei film raccomandabili.

5.1.1 Costruzione del set di candidati semantici

Una volta definito il profilo, il sistema estrae da DBpedia un insieme di film potenzialmente rilevanti per l'utente. Questa selezione avviene tramite:

- Query SPARQL su titoli contenenti le parole chiave corrispondenti ai generi e ai tag preferiti.
- Query SPARQL condizionate sulla presenza di un dato attore o regista nel cast (tramite le proprietà *dbo:starring* e *dbo:director*).
- Query SPARQL per anno di uscita, concentrandosi su un intervallo temporale centrato sugli anni di rilascio dei film apprezzati (con tolleranza di ±5 anni).

L'unione di questi tre criteri produce un insieme ibrido di film candidati, coerente sia dal punto di vista semantico sia temporale rispetto alle preferenze dell'utente.

5.2 Uso del Multi-Armed Bandit per la selezione finale

Dall'insieme dei candidati ottenuto, il sistema seleziona un singolo film da raccomandare impiegando una strategia Multi-Armed Bandit nella variante **Softmax**. Questo approccio assegna a ciascun film una probabilità di essere scelto, calcolata in base al numero di aspetti semantici che soddisfa (generi, tag, attori, registi, anni).
La strategia Softmax bilancia naturalmente tra:

- **Sfruttamento (exploitation)**: favorisce i film con il maggior numero di aspetti in comune con le preferenze dell'utente;
- **Esplorazione (exploration)**: assegna comunque una probabilità non nulla anche ai film meno evidenti, permettendo la scoperta di nuove opzioni rilevanti.

Il comportamento è controllato da un **parametro α**, che regola la "temperatura" del modello:

- Valori bassi di α aumentano l'esplorazione;
- Valori alti privilegiano lo sfruttamento.

Questo consente una selezione finale **stocastica ma guidata semanticamente**, priva di dipendenza da rating espliciti.

5.3 Valutazione del Suggerimento

Per valutare l'efficacia del sistema di raccomandazione, è stato adottato un meccanismo di calcolo dell'**Entity Error**, ovvero una metrica che stima quanto il film raccomandato sia in linea con le preferenze espresse dall'utente target. L'obiettivo è fornire una misura della coerenza semantica tra il profilo dell'utente e il contenuto suggerito.

Caso 1 – Film già valutato dall'utente
Nel caso in cui il film consigliato sia **presente tra quelli valutati positivamente dall'utente** (ovvero con rating ≥ 3.0), l'Entity Error viene calcolato in modo **proporzionalmente inverso al punteggio assegnato**. Viene infatti utilizzata la formula:

$$EntityError = 1.0 - \frac{rating}{5.0}$$

Tale approccio consente di attribuire un errore nullo ai film valutati con il massimo punteggio (5.0), e un errore crescente al diminuire del gradimento espresso dall'utente.

Caso 2 – Film non valutato
Nel caso più generale, in cui il film **non è stato precedentemente valutato** dall'utente, si procede a una **valutazione semantica** basata sul confronto tra le caratteristiche del film suggerito e il profilo preferenziale dell'utente. In particolare, il confronto viene effettuato sulle seguenti entità:

- **Generi**: viene verificato se i generi del film raccomandato rientrano tra i generi preferiti.
- **Tag**: si verifica la presenza di tag in comune.
- **Attori principali**: si esegue una query SPARQL per recuperare gli attori del film, confrontandoli con quelli più ricorrenti nei film apprezzati dall'utente.
- **Registi**: analogo al punto precedente, ma per i registi.
- **Anno di uscita**: se l'anno di uscita del film è presente o prossimo (±5 anni) rispetto a quelli più frequenti nei film preferiti.

Per ogni aspetto sopra elencato si incrementa il numero totale di confronti effettuati (*totalChecks*) e, se l'aspetto è rispettato, anche il contatore di corrispondenze (*matchCount*) viene incrementato. L'EntityError viene infine calcolato come:

$$EntityError = 1.0 - \frac{matchCount}{totalChecks}$$

Questo metodo consente di stimare la qualità del suggerimento anche in assenza di feedback espliciti, attraverso un'analisi knowledge-based basata su aspetti semantici (generi, tag, attori, registi, anno).
L'adozione di un approccio ibrido per la valutazione, che combina:

- i rating espliciti (quando disponibili), e
- il confronto semantico tra il film raccomandato e le preferenze dell'utente,

garantisce una stima dell'errore più completa e affidabile. Ciò assicura robustezza e copertura, anche in condizioni di cold-start o quando l'utente non ha ancora valutato direttamente determinati film.

6. Risultati Sperimentali

6.1 Descrizione dei test condotti

Per verificare il comportamento e la coerenza del sistema di raccomandazione basato sulla conoscenza, è stata effettuata un'esecuzione del sistema sull'utente con **ID = 7**.
Il processo ha seguito queste fasi:

- Costruzione del profilo utente a partire dai film valutati positivamente, estraendo **generi**, **tag**, **attori**, **registi** e **anni preferiti**.
- Recupero dei film candidati da **DBpedia**, tramite query SPARQL basate sugli aspetti preferiti.
- Applicazione della strategia **Softmax** per la selezione del film finale, in base al numero di aspetti semantici soddisfatti.

Questo test ha permesso di verificare il funzionamento dell'intero flusso, inclusa l'integrazione semantica e la selezione stocastica guidata.
Di seguito sono riportate una serie di catture che definiscono (in parte) l'output generato da un'esecuzione del software proposto…

Figura 1 Estrazione dei generi preferiti

Figura 2 Estrazione dei tag preferiti

Figura 3 Estrazione degli attori e dei registi preferiti

Figura 4 Estrazione dei candidati per genere

Figura 5 Estrazione dei candidati per tag

Figura 6 Estrazione dei candidati per attori

Figura 7 Estrazione dei candidati per registi

Figura 8 Estrazione dei candidati per anno di uscita

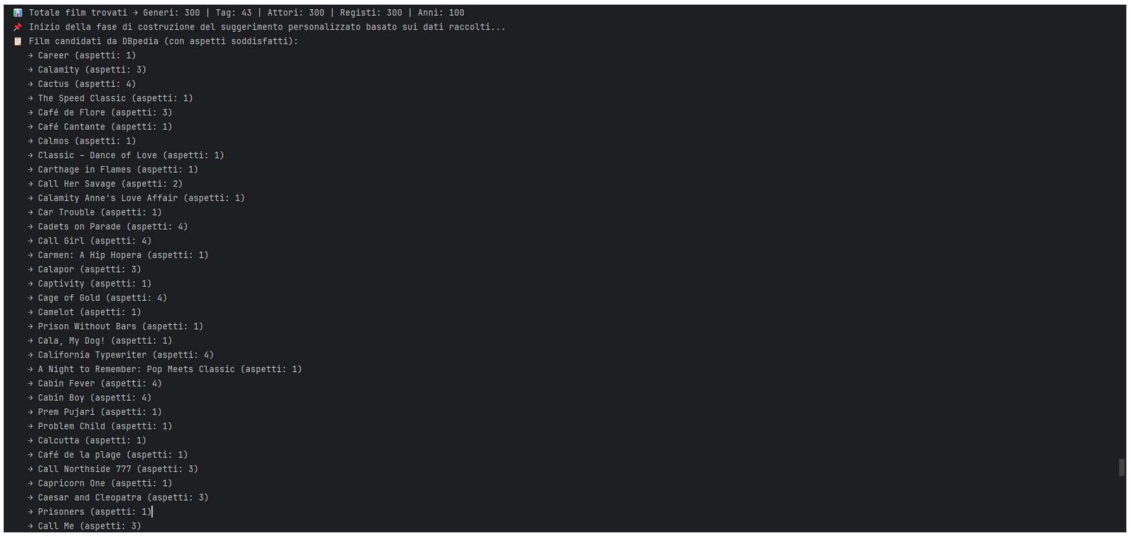


Figura 9 Riepilogo dell'estrazione dei candidate

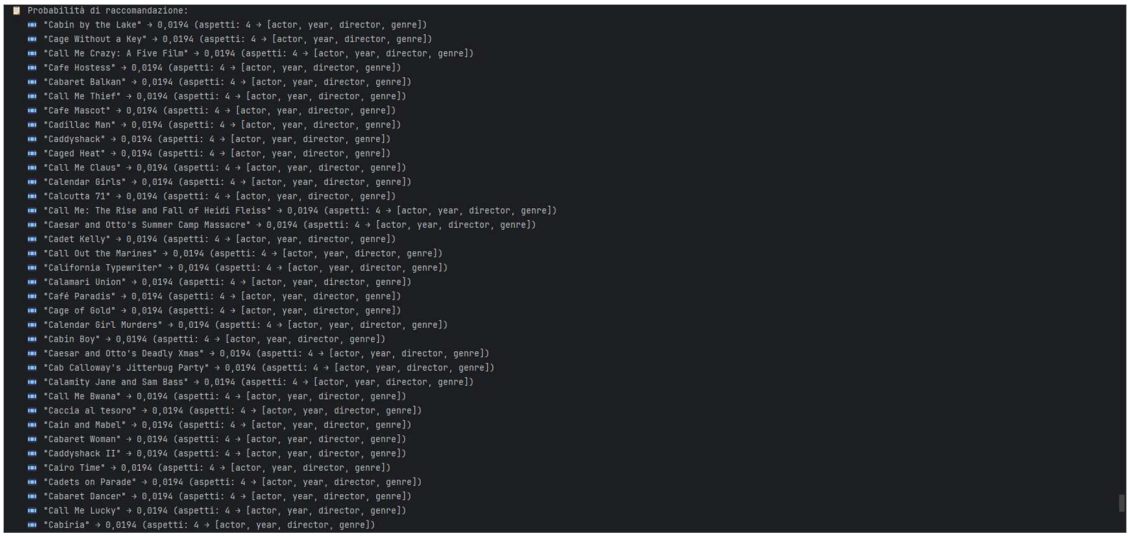


Figura 10 Classifica degli item per probabilità Softmax

Figura 11 Risultato della raccomandazione

6.2 Discussione dei risultati

- **Accuratezza**: Nel test condotto sull'utente 1, il film consigliato è *Cadets on Parade*, non presente tra quelli precedentemente valutati. Tuttavia, esso soddisfa 4 aspetti semantici su 5 (genere, anno, regista, attore), raggiungendo un Entity Error pari a 0.20. Questo dimostra che, pur in assenza di rating diretti, il sistema è in grado di generare raccomandazioni altamente pertinenti attraverso la profilazione semantica.
- **Tempi di risposta**: l'uso della cache integrata ha permesso un'esecuzione fluida e rapida, evitando interrogazioni ridondanti a DBpedia. I tempi si mantengono ottimali finché il volume delle query SPARQL (soprattutto quelle opzionali) resta contenuto.
- Coverage: la copertura è risultata eccellente:
    - 300 film candidati finali
    - Elevata varietà semantica (tag, generi, attori, registi, anni)
    - Softmax ha operato su item arricchiti da fino a 5 aspetti preferiti

  Tuttavia, alcune raccomandazioni, pur soddisfacendo criteri formali, possono apparire semanticamente *deboli* o *non centrali*. Le cause includono:
    - Match eccessivamente permissivi dovuti all'uso di FILTER(CONTAINS(...))
    - Incompleta o rumorosa etichettatura su DBpedia (es. proprietà mancanti, film ambigui)
    - Titoli semantici ambigui o con alias sovrapposti
- Qualità delle risposte SPARQL: sebbene nessuna eccezione sia stata sollevata a runtime, alcune risposte di DBpedia risultano rumorose: attori e registi estratti appartenevano talvolta a film omonimi o erroneamente collegati. Ciò evidenzia i limiti intrinseci della fonte DBpedia e sottolinea l'importanza di strategie più raffinate (es. matching esatto tramite URI).

7. Conclusioni e Sviluppi Futuri

7.1 Vantaggi del sistema sviluppato

- **Integrazione della conoscenza esterna (DBpedia):** l'uso di un knowledge graph permette di arricchire il processo di raccomandazione con metadati semantici (attori, registi, tag, generi, anno), ampliando notevolmente la copertura rispetto ai soli dati presenti nel dataset MovieLens.
- **Strategia di raccomandazione adattiva (Multi-Armed Bandit):** l'adozione dell'approccio **Softmax** consente di bilanciare **in modo probabilistico l'esplorazione di nuovi film e lo sfruttamento delle preferenze note, migliorando la varietà e la pertinenza delle raccomandazioni.**
- **Fallback semantico:** l'uso di FILTER CONTAINS e l'estrazione diretta tramite proprietà SPARQL (es. dbo:starring, dbo:director) consente di ottenere risultati anche quando i match esatti non sono disponibili.
- **Caching intelligente:** evita richieste SPARQL ripetute e riduce i tempi di risposta nel caso di esecuzioni successive con lo stesso profilo utente.
- **Valutazione flessibile:** l'Entity Error misura quanto un film raccomandato rispetti le preferenze esplicite dell'utente, sia tramite rating che attraverso confronto semantico su proprietà note.

7.2 Svantaggi e Limiti

- **Dipendenza da DBpedia:** se DBpedia non restituisce risultati per un determinato film o attore, la copertura semantica può risultare incompleta. In particolare, la presenza di titoli ambigui o con disambiguazioni errate può compromettere la qualità delle query.
- **Rumore semantico nei risultati SPARQL:** alcune query (soprattutto quelle su attori e registi) restituiscono entità irrilevanti o non direttamente collegate al contesto, introducendo rumore nei candidati.
- **Matching non esatto con MovieLens:** i titoli dei film da DBpedia non sempre corrispondono perfettamente a quelli presenti in MovieLens, causando problemi di valutazione e assegnazione del rating medio.
- **Valutazione dell'accuratezza non convenzionale:** l'Entity Error si basa su euristiche e preferenze semantiche, risultando meno oggettiva rispetto a metriche standard come Precision o Recall.

7.3 Possibili Sviluppi Futuri

- **Utilizzo di Wikidata:** alternativa a DBpedia, più aggiornata e con maggiore granularità semantica. La sua struttura a triplette consente anche inferenze logiche.
- **Integrazione del feedback esplicito dell'utente:** si può estendere il sistema per raccogliere input diretti (like/dislike) su ciascun suggerimento, aggiornando dinamicamente il profilo.
- **Embedding semantici:** sfruttare rappresentazioni dense degli item (es. RDF2Vec o modelli basati su BERT) per calcolare similarità tra film in maniera più robusta rispetto al solo matching testuale.
- **Uso di LLM (Large Language Models):** modelli come GPT o BERT possono essere impiegati per generare descrizioni testuali, clusterizzare film, o affinare la spiegazione dei suggerimenti, migliorando la trasparenza e la personalizzazione.
- **Raccomandazione multi-utente:** estendere l'approccio a gruppi di utenti o scenari sociali per generare suggerimenti condivisi.
- **Metriche di valutazione avanzate:** introdurre Precision, Recall, NDCG e Coverage per valutazioni più formali su scala.