

Devo realizzare un progetto universitario nel quale devo implementare un content-based knowledge-based recommender system nel contesto dei film, utilizzando tecniche MAB per risolvere il dilemma del compromesso tra exploration ed exploitation. Il dataset è il MovieLens Small, e posso utilizzare delle query SPARQL per interrogare il database DBPedia ed ottenere maggiori informazioni. Una possibile idea che ho pensato è quella di realizzare questo sistema andando a vettorializzare tutti i film e, quindi, utilizzare la cosine similarity.

Rispondi alle seguenti domande:

- Pensi sia una buona idea? Può funzionare o c'è qualcosa che dovrei modificare?
- Come posso implementare le tecniche MAB in questo contesto?
- Come posso rappresentare in modo empirico la preferenza degli utenti?
- Come posso realizzare una fase di verifica sperimentale del sistema di raccomandazione? Una possibile idea è quella di simulare in modo automatico o manuale la scelta di film di un utente e, quindi, utilizzare la matrice dei rating messa a disposizione dal dataset MovieLens Small per capire quanto sto sbagliando la predizione della distribuzione di probabilità della preferenza. Ti sembra una buona proposta?



Ragionato per 22 secondi >

Ecco una proposta strutturata per affrontare il progetto di un sistema di raccomandazione content-based e knowledge-based con tecniche multi-armed bandit (MAB), sfruttando MovieLens Small e arricchimento da DBpedia. Ogni sezione include riferimenti a fonti esterne per approfondimenti.

1. Validità dell'idea generale

L'idea di combinare un approccio content-based con un knowledge-based enrichment dei film e poi integrare un meccanismo di bandit per gestire exploration vs. exploitation è sensata e allineata a pratiche comuni in letteratura:

- **Content-based filtering:** si costruisce un profilo utente basato sulle caratteristiche degli item apprezzati, e si raccomandano film simili a quelli con rating elevati
en.wikipedia.org .
- **Knowledge-based enrichment:** quando si dispone di un knowledge graph o di metadati esterni (es. DBpedia), si possono estrarre attributi semantici (regista, attori,

generi, temi, ecc.) per comporre [Vai ai contenuti](#) caratteristiche più ricco rispetto ai soli tag o descrizioni testuali [en.wikipedia.org](#).

- **Dataset MovieLens Small:** contiene circa 100.836 valutazioni e 3.683 tag applications su 9.742 film, create fino al 24 settembre 2018 [files.grouplens.org](#). Pur non essendo aggiornato oltre il 2018, rimane un benchmark adeguato per prototipi e sperimentazioni accademiche.

Quindi il workflow proposto è **plausibile**: estrarre feature dei film (metadata + arricchimento DBpedia), costruire un profilo utente content-based, e aggiungere un componente bandit per imparare online le preferenze e bilanciare esplorazione/esplorazione. Non è obbligatorio modificare drasticamente il disegno, ma alcuni aspetti meritano cura (ad esempio la grande dimensionalità delle feature semantiche e la valutazione offline di policy bandit).

2. Arricchimento con DBpedia tramite SPARQL

Per estrarre informazioni semantiche sui film (es. attributi come regista, cast, genere, anno, paese di produzione, temi, ecc.), si può usare il endpoint SPARQL di DBpedia:

- **Endpoint SPARQL:** l'endpoint pubblico si trova all'indirizzo <https://dbpedia.org/sparql> [stackoverflow.com](#), e DBpedia espone le risorse di Wikipedia normalizzate in RDF, con ontology come `dbo:Director`, `dbo:starring`, `dbo:genre` ecc. [en.wikipedia.org](#).
- **Query tipiche:** per ogni film (identificato ad es. tramite titolo o URI DBpedia corrispondente), estrarre triple rilevanti: p.es.

sparql

 Copia codice

```
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dbr: <http://dbpedia.org/resource/>
SELECT ?regista ?attore ?genere ?anno ?paese WHERE {
  dbr:Inception dbo:director ?regista ;
                dbo:starring ?attore ;
                dbo:genre ?genere ;
                dbo:releaseDate ?anno ;
                dbo:country ?paese .
}
```

- **Pre-elaborazione:** normalizzare le stringhe (es. mappare i generi in categorie controllate), gestire dati mancanti (alcuni film potrebbero non avere certe proprietà), e limitare la dimensionalità (potrebbe servire selezionare solo le proprietà più informative o raggrupparne i valori, ad esempio aggregando sottogeneri in generi più alti).
- **Caching e rate-limit:** l'endpoint pubblico ha limiti di query al secondo; meglio estrarre offline con batch di SPARQL e salvare localmente (es. in file JSON o in un grafo RDF locale) prima di costruire i vettori.

3. Rappresentazione dei film e del profilo utente

3.1 Vettorizzazione dei film [Vai ai contenuti](#)

- **Feature binarie o tf-idf:** per attributi categoriali (genere, regista, attori), si possono usare vettori one-hot o multi-hot; per testi (abstract, plot), si può applicare TF-IDF su descrizioni tratte da Wikipedia o da DBpedia.
- **Embeddings semantici:** si possono anche considerare embedding pre-addestrati (es. RDF2Vec su DBpedia) per mappare entità in uno spazio continuo; tuttavia, per un prototipo con MovieLens small, spesso si inizia con rappresentazioni sparse (one-hot) o TF-IDF semplificate [researchgate.net](#).
- **Concatenazione di feature:** un vettore film = [one-hot generi] + [one-hot regista] + [one-hot attori selezionati] + [TF-IDF descrizione] + [anno normalizzato] + ...; attenzione alla scalabilità: MovieLens Small ha circa 9.700 film, ma il numero di attori/registi distinti può crescere; si possono limitare ai top-K più frequenti o raggruppare entità minori in “altro”.

3.2 Profilo utente empirico

- **Feedback esplicito:** le valutazioni 1–5 del dataset MovieLens forniscono un segnale di gradimento. Si possono mappare in un reward binario (like/dislike) o in un reward continuo normalizzato (es. $(rating - 3) / 2$ per scala [-1, +1], se serve).
- **Profilo statico iniziale:** per un utente, si può calcolare un vettore profilo come media pesata delle feature dei film valutati positivamente (o differenza tra media di quelli valutati positivamente e media di quelli con rating basso) [en.wikipedia.org](#).
- **Aggiornamento dinamico:** in un contesto bandit, ad ogni interazione si osserva un reward su un film consigliato; si aggiorna la stima del modello (vedi sezione MAB). Il profilo utente non è “fisso” offline, ma evolve in base ai reward ottenuti.

4. Integrazione delle tecniche Multi-Armed Bandit (MAB)

4.1 Scelta dell’approccio bandit

- **Context-free vs. contextual bandit:**
 - *Context-free* (classico K-armed bandit) tratta ogni film come un braccio senza differenziare contesto: non sfrutta le feature film/utente, e in presenza di molti film (K grande) l’apprendimento è lento.
 - *Contextual bandit* sfrutta feature vettoriali (sia film che utente): alla prova t , si osserva un contesto (profilo utente corrente o contesto di sessione), e un insieme di bracci candidati (tutti i film o un sottoinsieme); si usa algoritmo come LinUCB o Thompson Sampling con modello lineare o altro per stimare reward atteso in base al prodotto tra vettore feature e parametro da apprendere [arxiv.org](#).

4.2 Algoritmi consigliati

- **LinUCB:** assume reward atteso lineare in funzione del vettore contesto; mantiene stime dei parametri tramite regressione ridge e costruisce bound di confidenza per exploration/exploitation [arxiv.org](#).

- **Thompson Sampling lineare:** [Vai ai contenuti](#) bayesiano lineare sul reward; campiona parametri a posteriori e sceglie l'azione con stima reward massima per exploration implicita.
- **Neural bandit:** per feature complesse, si può sperimentare NeuralUCB o Neural Thompson Sampling (es. Zhou et al. 2020), ma per MovieLens Small un modello lineare è già un ottimo punto di partenza [researchgate.net](#).
- **Combinatorial bandit:** se si raccomanda un set di film in batch piuttosto che uno a uno, si può considerare un contesto combinatoriale (più avanzato).

4.3 Modellazione del contesto

- **Feature di contesto:** per ogni film candidato, utilizzare il vettore film; per l'utente, si possono includere le statistiche correnti (profilo medio), oppure altre informazioni di sessione (es. genere preferito in questa sessione).
- **Azione / braccio:** ogni film è un braccio. Dato il numero relativamente grande (~9.700), in ogni turno si può ridurre lo spazio dei candidati: es. selezionare un sottoinsieme top-N tramite filtro content-based puro, e applicare bandit su N relativamente piccolo (es. N=20–50). In tal modo l'apprendimento converge più rapidamente e si mantiene la latenza bassa.

4.4 Definizione di reward

- **Reward esplicito:** rating fornito dall'utente (mapping continuo o binario). Se si simula offline, si assume che l'utente dia rating al film consigliato: usare rating storico come proxy.
- **Reward implicito:** click, visualizzazioni parziali, tempo di visione; non disponibile in MovieLens classico, ma se si simulasse un utente reale, si potrebbe modellare reward binario basato su soglia di rating ≥ 4 = successo.

5. Rappresentazione empirica della preferenza

- **Profilo iniziale:** se si simula un nuovo utente, si può partire da un profilo “vuoto” oppure da qualche preferenza iniziale random o basata su cluster di utenti esistenti. In MovieLens, spesso per cold-start utente si chiede un set di rating iniziali (es. 20 film diversificati) per costruire un profilo base [en.wikipedia.org](#).
- **Simulazione offline:** ogni utente del dataset ha già valutato alcuni film. Si può trattare ogni rating come se fosse feedback a un consiglio: in simulazione, quando la policy consiglia un film che l'utente ha già valutato, si “rivela” il reward corrispondente; se la policy consiglia un film non valutato dall'utente, si può gestire come “non osservato” (non si sa il reward) e nell'offline bandit evaluation bisogna evitare bias (vedi sezione successiva).
- **Costruzione del profilo evolutivo:** il profilo utente (parametri del contesto) si aggiorna dopo ogni interazione osservata.

6. Verifica sperimentale del sistema

6.1 Valutazione offline di pr

[Vai ai contenuti](#)

- **Sfida offline:** i dati raccolti da MovieLens non sono generati da policy random; sono valori rating raccolti in modo passivo. Per valutare una policy bandit offline senza bias, servirebbe log di consigli casuali (random) con relativo reward osservato, per applicare tecniche di Inverse Propensity Scoring (IPS) o metodi di off-policy

evaluation [arxiv.org](#) [cake.fiu.edu](#) [📄](#).

- **Raccolta di “casual logging”:** in assenza di log random, si può simulare un logging policy random in ambiente sintetico, o assumere che parte delle interazioni possa essere trattata come random se si considera il dataset intero (con cautela, rischio di bias).
- **Protocollo hold-out per simulazione:** una pratica comune in prototipi:
 1. **Divisione temporale:** ordinare i rating di ogni utente in ordine cronologico (se timestamp disponibile) o simulare ordini casuali.
 2. **Sessione simulata:** per ogni utente, considerare i primi k rating come “storico iniziale” per costruire profilo, poi per ogni step t successivo: la policy seleziona un film fra quelli non ancora raccomandati; se il film consigliato è presente nei rating storici di “verità”, ottenere reward; altrimenti, saltare quell’interazione (o assegnare reward neutro, a seconda del protocollo).
 3. **Metriche:** misurare cumulative reward (somma rating o binarizzato), o regret cumulato rispetto a una policy oracolo (es. raccomandare sempre il film con valutazione massima futura, ipotetico).
 4. **Ripetizione su utenti multipli:** aggregare performance su molti utenti per stime statisticamente significative.
- **Valutazione off-policy:** se si dispone di un logging policy casuale, si possono applicare IPS e Self-Normalized IPS per stimare reward atteso della nuova policy

senza bias [cake.fiu.edu](#) [📄](#).

6.2 Simulazione manuale vs. automatica

- **Simulazione automatica:** come descritto, usare i rating esistenti per “rispondere” alle raccomandazioni: è il metodo standard in prototipi accademici.
- **Simulazione manuale:** meno scalabile; chiedere a soggetti di dare feedback su raccomandazioni in un’interfaccia; utile per capire usabilità, ma costoso in termini di utenti e tempo. Può essere usato in un proof-of-concept limitato, ma per valutazioni statistiche conviene lavorare offline su MovieLens.
- **Proposta utente:** usare simulazione automatica sui dati MovieLens Small, con procedura di hold-out descritta. La matrice rating fornisce “ground truth” quando la policy consiglia un film già valutato. Questo è una **buona proposta** per la fase di sperimentazione iniziale. Tuttavia, bisogna considerare il bias di selezione (non tutti i

film sono stati valutati da ogni utente. Senza di logging random: quindi i
Vai ai contenuti

risultati offline vanno interpretati con cautela. [arxiv.org](#) [cake.fiu.edu](#) .

- **Alternative:** creare un modello sintetico di utente con preferenze generative (ad es. un vettore latente e modello di probabilità di rating), per poter simulare feedback anche su film non valutati nel dataset; questo permette valutare policy in modo più controllato, ma richiede assumere un modello di utente (potenzialmente mismatched con la realtà).

6.3 Metriche di performance

- **Cumulative reward:** somma dei reward (rating normalizzati) ottenuti nel corso della simulazione.
- **Regret cumulato:** differenza tra reward oracolo (se conoscessimo in anticipo i rating futuri) e reward policy.
- **Precision@k o CTR:** se simuliamo raccomandazioni top-k in ogni turno, valutare quante raccomandazioni sono “positive” (rating alto).
- **Time-to-convergence:** numero di step per cui la policy supera un baseline (es. raccomandazione casuale o content-based fisso).
- **Diversità / novelty:** misure su quanto la policy esplora film meno popolari; spesso si vuole evitare filter bubble.

7. Implementazione pratica

7.1 Ambiente e librerie

- **Lingua:** Python è adatto.
- **Estrazione e pre-elaborazione:**
 - Usare librerie SPARQL (ad es. `SPARQLWrapper`) per interrogare DBpedia offline e salvare i dati in file locali (JSON, CSV).
 - Pandas per manipolare i rating MovieLens e i vettori di feature.
- **Costruzione vettori:**
 - Scikit-learn `DictVectorizer` o `OneHotEncoder` per feature categoriali, `TfidfVectorizer` per testi.
 - Eventualmente librerie per embedding (es. `gensim`, se si vogliono embedding da descrizioni), o `rdflib` per gestione RDF.
- **Algoritmi bandit:**
 - Implementare LinUCB o Thompson Sampling lineare in Python (la logica è relativamente semplice: mantenere $A = D \times D$ matrix, $b = D \times 1$, aggiornare parametri; per TS usare posteriore gaussiana). Ci sono librerie come `mabwiser`, ma per didattica conviene scrivere da zero.

- **Valutazione offline:**

- Creare script per procedure di valutazione cronologica.
- Calcolare metriche.

7.2 Workflow esemplificativo

1. Caricamento MovieLens Small:

- Leggere `ratings.csv` e `movies.csv`.
- Filtrare utenti con abbastanza rating (es. ≥ 20) per una simulazione stabile; opzionale.

2. Estrazione metadati DBpedia:

- Per ogni film in `movies.csv`, mappare titolo a URI DBpedia (possibile matching su titolo + anno; attenzione a omonimie).
- Eseguire batch di query SPARQL per ottenere attributi chiave (regista, generi, attori principali).
- Salvare risultati in file locale.
- Pre-elaborare e vettorizzare feature film.

3. Preparazione dati utente:

- Per ogni utente selezionato, ordinare rating per timestamp; dividere in fase “warm-up” e fase di interazione.

4. Inizializzazione policy bandit:

- Inizializzare parametri (per LinUCB $A = \lambda I$, $b=0$).
- Opzionalmente, iniziare con un profilo user basato su warm-up: se durante warm-up si usa content-based puro per stimare un vettore utente iniziale, tale vettore può essere usato come contesto iniziale per LinUCB.

5. Simulazione step-by-step:

- In ogni step t : policy valuta ogni film candidato (non ancora raccomandato all'utente), calcola punteggio UCB o campione TS, seleziona film con punteggio massimo.
- Se l'utente ha fornito rating per quel film in storico “futuro”, usare quel rating come reward e aggiornare parametri (aggiornare A e b). Altrimenti, si può saltare o assegnare reward neutro; documentare la scelta.

6. Raccolta metriche:

- Calcolare cumulative reward, regret relativo a baseline (es. raccomandazione casuale), precision@k in ogni fase, velocità di miglioramento.

7. Analisi risultati:

- Confrontare diverse configurazioni: ad esempio, LinUCB con feature limitate vs. feature complete arricchite DBpedia; Thompson Sampling vs. UCB; politiche hybrid content-based + bandit (usare content-based per pre-filtrare top-20 e poi bandit) vs. bandit su spazio intero; valutare trade-off performance vs. complessità computazionale.

- Studiare sensibilità a parametri (regolarizzazione, scelte di mapping rating→reward).
Vai ai contenuti

8. Vantaggi e possibili miglitorie

- **Cold-start film/utente:** l'arricchimento DBpedia aiuta nel cold-start film perché fornisce feature note prima di avere rating. Per utenti nuovi, si possono creare questionari iniziali (es. chiedere quali generi preferisce) o usare qualche interazione random in fase warm-up.
- **Dimensionalità:** le feature semantiche possono essere molto sparse e ad alta dimensione; considerare riduzione dimensionale (PCA, autoencoder) o selezione feature, per migliorare efficienza del bandit.
- **Off-policy evaluation avanzata:** se si dispone di log di interazioni randomizzate (non certo nel dataset MovieLens), si possono applicare tecniche IPS/DR per valutazioni

meno biased cake.fiu.edu .

- **Hybridizzazione:** oltre a content-based + knowledge-based, si potrebbe combinare con elementi collaborative (es. implicit feedback di utenti simili) in un modello ibrido; ciò richiede un dataset più ampio e tecniche di ibridazione.
- **Personalizzazione del reward:** se si avessero dati reali di click/dwell time, si potrebbe definire reward più ricco; nel prototipo offline si rimane sul rating.
- **Scalabilità:** MovieLens Small è piccolo; ma concetti e prototipi possono poi essere estesi a dataset maggiori o sistemi reali.

9. Conclusioni e raccomandazioni

- **Idea generale:** è buona e allineata a letteratura di content-based + knowledge-based + contextual bandit en.wikipedia.org arxiv.org.
- **Implementazione MAB:** usare LinUCB o Thompson Sampling lineare con feature film come contesto; ridurre lo spazio candidato in ogni turno tramite filtro content-based per accelerare apprendimento.
- **Rappresentazione preferenze:** basarsi sulle valutazioni esplicite, convertite in reward; costruire profilo iniziale dal warm-up; aggiornare dinamicamente i parametri bandit.
- **Verifica sperimentale:** simulazione offline con hold-out di MovieLens Small è un'ottima fase iniziale; attenzione a bias di osservazione, mancanza di log random; integrare analisi del bias o simulazione di modelli sintetici. L'idea di utilizzare la matrice di rating per misurare quanto "si sbaglia" nella previsione della distribuzione di probabilità di preferenza è corretta, purché si definisca chiaramente la procedura di valutazione (regret, cumulative reward, IPS se possibile).
- **Step successivi:** prototipare gradualmente, analizzare risultati, documentare limiti dovuti a dati offline. Se vi è la possibilità di raccogliere dati reali (ad es. con utenti in un'implementazione web), si può validare online e applicare realmente policy bandit.

In sintesi, la proposta di vettorializzare i film con caratteristiche arricchite via DBpedia e di integrare una policy contextual bandit è solida. Presta attenzione soprattutto all'alta dimensionalità delle feature, alla corretta simulazione offline (evitando bias) e alla definizione chiara del reward. La fase sperimentale basata su simulazione con MovieLens Small è adeguata per un progetto universitario, pur riconoscendo i limiti di valutazione derivanti dalla natura passiva dei dati raccolti. Buon lavoro per l'implementazione!


Riferimenti principali citati:

- MovieLens Small dataset: circa 100.836 ratings, info fino al 24 settembre 2018
files.grouplens.org .
- Content-based & knowledge-based filtering: panoramica su Wikipedia en.wikipedia.org
en.wikipedia.org .
- DBpedia SPARQL endpoint: <https://dbpedia.org/sparql> stackoverflow.com
en.wikipedia.org .
- Contextual bandit per raccomandazione: Li et al. "A Contextual-Bandit Approach to Personalized News Article Recommendation" arxiv.org ; offline evaluation unbiased:
Li et al. WSDM 2011 cake.fiu.edu [🔗](#).

Se servono esempi di codice o dettaglio sulla pipeline Python, fammi sapere!

        Fonti

Fai una domanda

+ 🔍 |  Avvia il ragionamento x



ChatGPT può commettere errori. Assicurati di verificare le informazioni importanti. Vedi [Preferenze sui cookie](#).