

Caso di studio del corso Ingegneria della conoscenza

Movies Recommender

Individuare il miglior film per il tuo intrattenimento



Autori:

Domenico Scalise

Matricola: 758458

E-mail: d.scalise@studenti.uniba.it

Gabriele Paladini

Matricola: 758417

E-mail: g.paladini7@studenti.uniba.it

AA: 2023-2024

https://github.com/Gabrielepaladini/Movies_Recommender.git

Sommario

1. Introduzione	3
1.1 Dataset utilizzati e librerie	4
1.2 Pre-processing del dataset.....	4
1.3 Gestione delle celle 'null' per ogni colonna.....	6
1.4 Rappresentazioni grafiche	6
1.5 Individuazione di eventuali correlazioni.....	9
2. Recommender System.....	11
2.1 Sommario	11
2.2 Scelta metrica da impiegare	11
2.3 Realizzazione.....	12
2.4 Classificazione	13
3. KB.....	18
3.1 Introduzione.....	18
3.2 Gestione della KB	18
3.3 Fatti	18
3.4 Regole.....	20
3.5 Trova film date determinate caratteristiche	20
3.6 Sulla base delle preferenze dell'utente, trova la piattaforma migliore per i film selezionati	22
4. Ontologia	25
4.1 Sommario	25
4.2 Protégé.....	25
4.3 Owlready2.....	30
5. Conclusioni.....	35

1. Introduzione

Lo streaming di servizi, quali film e serie tv, è oggi diventato una parte integrante delle nostre vite.

Le piattaforme di streaming sono oramai uno degli strumenti più utilizzati per l'accesso a film e serie tv. L'abbondanza di contenuti disponibili e la crescente competizione di queste piattaforme richiedono un'analisi approfondita ed intelligente per comprendere meglio le dinamiche di utilizzo e le preferenze degli utenti.

Visto l'uso elevato, e in quanto noi stessi fruitori di queste piattaforme di streaming abbiamo deciso di indirizzare il nostro progetto alla realizzazione di un sistema che possa dare supporto agli utenti nella scelta di un titolo da visualizzare o nella scelta della migliore piattaforma a cui abbonarsi.

A tale scopo abbiamo realizzato:

- un recommender system progettato per consigliare nuove serie tv e film in base alle preferenze personali degli utenti. Con l'enorme quantità di contenuti disponibili, può essere difficile scegliere cosa guardare. Il nostro sistema utilizza algoritmi avanzati per analizzare le abitudini di visione degli utenti e suggerire i contenuti più adatti ai loro gusti individuali.
- un modello di classificazione in grado di prevedere la categoria dei ratings assegnati dagli utenti ai vari contenuti. Questo ci aiuterà a comprendere meglio come vengono valutati i film e le serie tv, e a migliorare ulteriormente i nostri suggerimenti basati sulle valutazioni previste.
- una Knowledge base interattiva per permettere agli utenti di acquisire informazioni dettagliate su serie tv e film.
- un'ontologia del dominio che descrive il dominio delle piattaforme di streaming, specificando il significato dei vari simboli e concetti utilizzati nel sistema.

Una volta che l'utente avvia il programma potrà in qualsiasi momento interagire con esse per scegliere l'opzione che più può accontentare i suoi bisogni del momento, avendo la possibilità di spostarsi rapidamente tra il recommender system, la knowledge base e l'ontologia.

```
Benvenuto in Movies Recommender!
```

```
Sei nel Menù principale
```

```
Seleziona un'operazione:
```

```
1) Recommender System
```

```
2) Interagisci con la KB
```

```
3) Interagisci con l'ontologia
```

```
4) Esci
```

```
Inserisci un valore:
```

1.1 Dataset utilizzati e librerie

Per realizzare questo sistema abbiamo utilizzato il seguente dataset: <https://www.kaggle.com/datasets/farhantanveerhasan/movie-stream-df>

A cui abbiamo applicato una serie di operazioni di “data cleaning” che saranno illustrate nel paragrafo dedicato al pre-processing del dataset.

Tutto il lavoro è stato svolto in linguaggio Python sfruttando i code editor VScode e Jupyter.

Abbiamo fatto, inoltre, largo uso delle librerie **Pandas** e **NumPy**.

1.2 Pre-processing del dataset

Dopo aver eseguito un’attenta osservazione degli elementi che costituiscono il dataset, passiamo alla fase di “pre-processing” o “cleaning” dello stesso.

Prima operazione che abbiamo eseguito è la rimozione delle colonne che non sono utili ai nostri scopi quali:

```
In [9]: columns_to_drop = ['type', 'seasons', 'imdb_id', 'genres', 'budget', 'nconst', 'primaryName']
movies = movies.drop(columns=columns_to_drop, errors='ignore')
movies.head()
```

```
Out[9]:
```

	Unnamed: 0	id	title	description	release_year	runtime	production_countries	imdb_score	imdb_votes	tmdb_popularity	tmdb_score	streaming
--	------------	----	-------	-------------	--------------	---------	----------------------	------------	------------	-----------------	------------	-----------

0	0	tm1000001	The Lucky Texan	Jerry Mason, a young Texan, and Jake Benson, a...	1934	61	US	5.6	1213.0	4.1	4.7	
1	1	tm1000022	Boonie Bears: The Wild Life	Bear brothers Briar and Bramble set off on an ...	2021	99	CN	5.4	117.0	6.1	3.8	
2	2	tm1000169	Bad Cupid	Archie is a God on a mission to ensure that tr...	2021	81	US	4.4	181.0	3.3	4.8	
3	3	tm1000186	Carol's Christmas	Scrooge encounters the ghost of her late busin...	2021	70	US	2.0	48.0	0.6	6.0	
4	4	tm1000203	Digging to Death	David Van Owen moves into a mysterious house a...	2021	96	US	4.5	464.0	3.5	5.8	

Sono state eliminate le seguenti colonne:

type: in quanto il dataset fa riferimento esclusivamente a film, non è utile la distinzione per tipo.

seasons: dato che gli elementi del dataset sono esclusivamente film non presenteranno stagioni, la colonna presenta infatti solo valori nulli.

imdb_id: non è necessaria l'identificazione univoca del titolo sulla piattaforma imdb.

genres: abbiamo pensato che la ricerca del titolo debba avvenire solo in base al genere che maggiormente rispecchia il titolo indicato.

budget: questo campo non è utile alla nostra ricerca e presenta un grande numero di valori nulli.

nconst: identificativo che non è utile al nostro scopo.

primaryName: riteniamo che la ricerca del titolo in base ad un attore che vi è presente debba avvenire tra gli elementi dell'insieme 'name' e non basarsi solo sull'attore protagonista.

Dalla analisi del dataset abbiamo notato che le metriche “imdb_score”, “tmdb_popularity”, “tmdb_score”, che rappresentano rispettivamente le valutazioni dei titoli delle piattaforme “imdb” e “tmdb”, presentano alcuni valori che non vengono approssimati correttamente.

In particolare, le metriche “imdb_score” e “tmdb_score” presentano alcuni valori che non vengono approssimati alla prima cifra decimale.

Per cui procediamo ad approssimarle correttamente:

```
In [8]: df = pd.DataFrame(movies)
df['imdb_score'] = df['imdb_score'].round(1)
df['tmdb_score'] = df['tmdb_score'].round(1)
print(df[['imdb_score', 'tmdb_score']].to_string(index=False))
df.to_csv('pre_processed_dataset.csv', index=False)
```

imdb_score	tmdb_score
5.6	4.7
5.4	3.8
4.4	4.8
2.0	6.0
4.5	5.8
7.5	7.4
4.0	8.0
6.9	5.4
6.9	6.3
4.5	5.0
5.1	6.0
4.7	5.6
4.1	4.4
8.0	8.2
4.0	4.5
5.7	6.0
3.5	5.1
6.1	6.0
6.3	6.3

Procediamo con il rinominare le due colonne “main_genre” e “name” in “genre” e “actors” in modo che il loro scopo sia più chiaro.

```
In [9]: df = pd.read_csv('pre_processed_dataset.csv')
df.rename(columns={'main_genre': 'genre', 'name': 'actors'}, inplace=True)
df.to_csv('preProcessed_dataset.csv', index=False)
```

Rinominiamo la colonna “subscription_cost” in “monthly_subscription_cost” e convertiamo i valori di questa colonna in float con due cifre decimali.

```
In [10]: df = pd.read_csv('preProcessed_dataset.csv')
df.rename(columns={'subscription_cost': 'monthly_subscription_cost'}, inplace=True)
df['monthly_subscription_cost'] = df['monthly_subscription_cost'].replace(to_replace=r'\D', value='', regex=True).astype(float)
df.to_csv('pre_processed_dataset.csv', index=False)
print(df['monthly_subscription_cost'])
```

0	14.99
1	14.99
2	14.99
3	14.99
4	14.99
...	
18714	4.99
18715	4.99
18716	4.99
18717	4.99
18718	4.99

Name: monthly_subscription_cost, Length: 18719, dtype: float64

1.3 Gestione delle celle 'null' per ogni colonna

```
In [14]: movies.isnull().sum()
```

```
Out[14]: Unnamed: 0      0
id          0
title       0
description  0
release_year 0
runtime     0
production_countries 0
imdb_score  0
imdb_votes  0
tmdb_popularity 0
tmdb_score  0
streaming_service 0
main_genre  0
name        0
monthly_subscription_cost 0
dtype: int64
```

Da questa funzione risulta che il nostro dataset non presenta alcun valore da gestire.

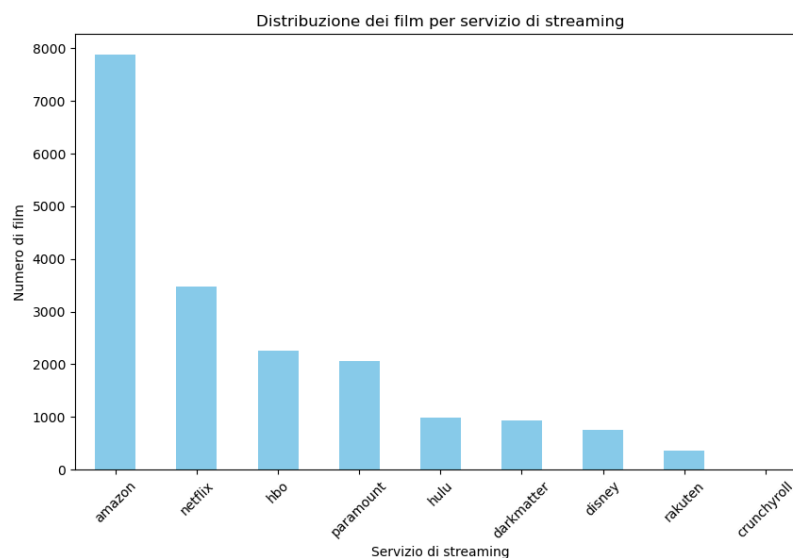
1.4 Rappresentazioni grafiche

Istogramma che rappresenta la distribuzione dei titoli cinematografici tra le varie piattaforme di streaming presenti nel dataset:

```
In [15]: df = pd.DataFrame(movies)
streaming_counts = df['streaming_service'].value_counts()

#creazione dell'istogramma
plt.figure(figsize=(10, 6))
streaming_counts.plot(kind='bar', color='skyblue')
plt.title('Distribuzione dei film per servizio di streaming')
plt.xlabel('Servizio di streaming')
plt.ylabel('Numero di film')
plt.xticks(rotation=45)
```

```
Out[15]: (array([0, 1, 2, 3, 4, 5, 6, 7, 8]),
 [Text(0, 0, 'amazon'),
  Text(1, 0, 'netflix'),
  Text(2, 0, 'hbo'),
  Text(3, 0, 'paramount'),
  Text(4, 0, 'hulu'),
  Text(5, 0, 'darkmatter'),
  Text(6, 0, 'disney'),
  Text(7, 0, 'rakuten'),
  Text(8, 0, 'crunchyroll')])
```



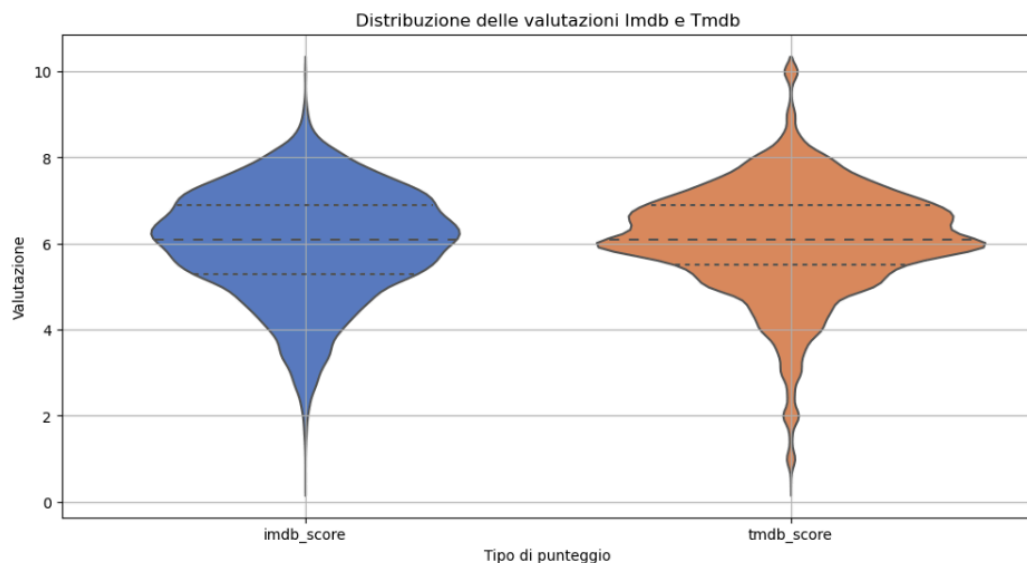
Dal grafico realizzato possiamo cogliere visivamente la distribuzione dei titoli tra le varie piattaforme di streaming che sono state considerate all'interno del dataset. Questi dati ci fanno rendere conto che la maggior parte dei titoli sono detenuti dalle piattaforme "Amazon" e "Netflix", per cui le raccomandazioni saranno, con probabilità più elevate delle altre, indirizzate verso queste ultime.

Grafico a violino che mostra la distribuzione delle valutazioni imdb_score e tmdb_score:

```
In [16]: df = pd.DataFrame(movies)

#convertiamo i dati in un formato 'long' adatto per seaborn
df_long = pd.melt(df, id_vars=['title'], value_vars=['imdb_score', 'tmdb_score'], var_name='score_type', value_name='score')
#creazione del grafico a violino
plt.figure(figsize=(12, 6))
sns.violinplot(x='score_type', y='score', data=df_long, inner='quartile', palette='muted')
plt.title('Distribuzione delle valutazioni Imdb e Tmdb')
plt.xlabel('Tipo di punteggio')
plt.ylabel('Valutazione')
plt.grid(True)

plt.show()
```



Dal grafico abbiamo rilevato una distribuzione delle votazioni per ciascun livello di valutazione. Ciascun violino rappresenta la variazione delle valutazioni assegnate dalle due piattaforme (imdb e tmdb) rispetto a ciascun titolo.

Da questi due grafici possiamo notare che la maggior parte delle valutazioni assegnate assumono un valore vicino al 6.

Istogramma che rappresenta la distribuzione dei prezzi tra le varie piattaforme di streaming:

```
In [17]: df = pd.DataFrame(movies)

streaming_services=['Amazon', 'Netflix', 'Disney', 'Hulu', 'Hbo', 'Darkmatter', 'Paramo

plt.figure(figsize=(10, 6))
hist = sns.histplot(data=df, x='monthly_subscription_cost', hue='streaming_service', mu
plt.title('Distribuzione dei costi di abbonamento alle piattaforme di streaming')
plt.xlabel('prezzo')
plt.ylabel('conteggio')
plt.grid(True)

handles, labels = hist.get_legend_handles_labels()

if not handles:

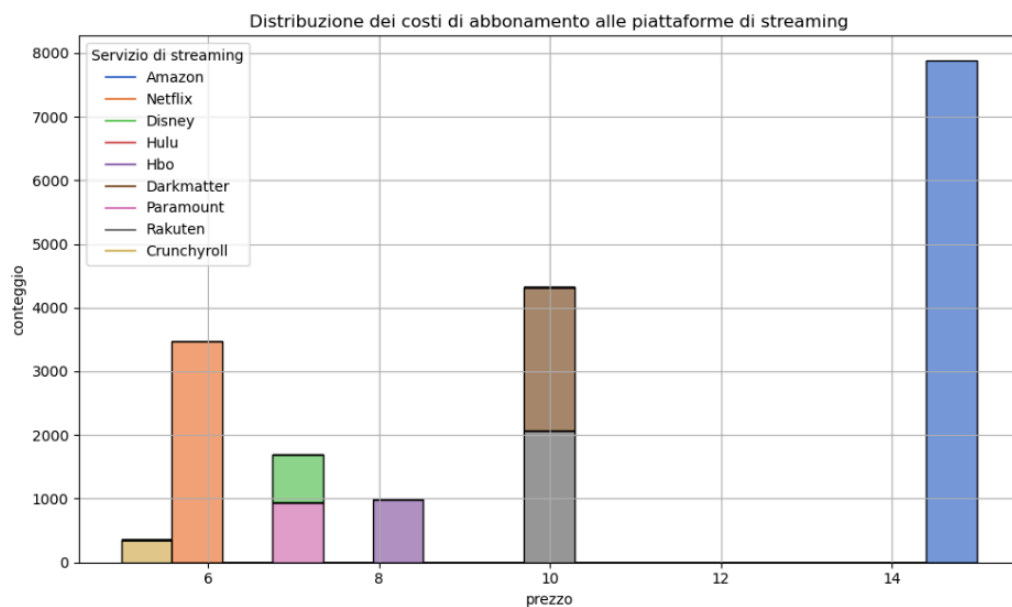
    #recuperiamo i colori della palette usata da seaborn
    palette = sns.color_palette('muted', len(streaming_services))
    color_dict = dict(zip(streaming_services, palette))

    #creiamo i gestori manualmente con i colori corretti
    for service in streaming_services:
        handles.append(plt.Line2D([], [], color=color_dict[service], label=service))

plt.legend(handles=handles, title='Servizio di streaming')

plt.tight_layout()
plt.show()

/opt/anaconda3/lib/python3.11/site-packages/seaborn/_oldcore.py:1119: FutureWarning: u
se_inf_as_na option is deprecated and will be removed in a future version. Convert inf
values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



Dall'istogramma realizzato possiamo cogliere la diversa distribuzione dei prezzi degli abbonamenti messi a disposizione delle varie piattaforme di streaming considerate all'interno del dataset.

1.5 Individuazione di eventuali correlazioni

Andiamo a sfruttare la seguente funzione per visualizzare il numero dei film messi a disposizione da ciascuna piattaforma di streaming presa in considerazione all'interno del dataset, con lo scopo di analizzare la correlazione tra il numero di film presente in ciascuna piattaforma e i generi maggiormente presenti in queste ultime.

```
In [18]: df = pd.DataFrame(movies)

film_count_per_service = df.groupby('streaming_service').size().reset_index(name='number_of_movies')
print(film_count_per_service)
```

	streaming_service	number_of_movies
0	amazon	7875
1	crunchyroll	1
2	darkmatter	940
3	disney	758
4	hbo	2264
5	hulu	987
6	netflix	3477
7	paramount	2062
8	rakuten	355

Per ciascuna piattaforma di streaming andiamo ad analizzare la distribuzione dei generi dei vari titoli presenti all'interno delle loro raccolte.

```
In [25]: import math

df = pd.DataFrame(movies)

streaming_services = df['streaming_service'].unique()

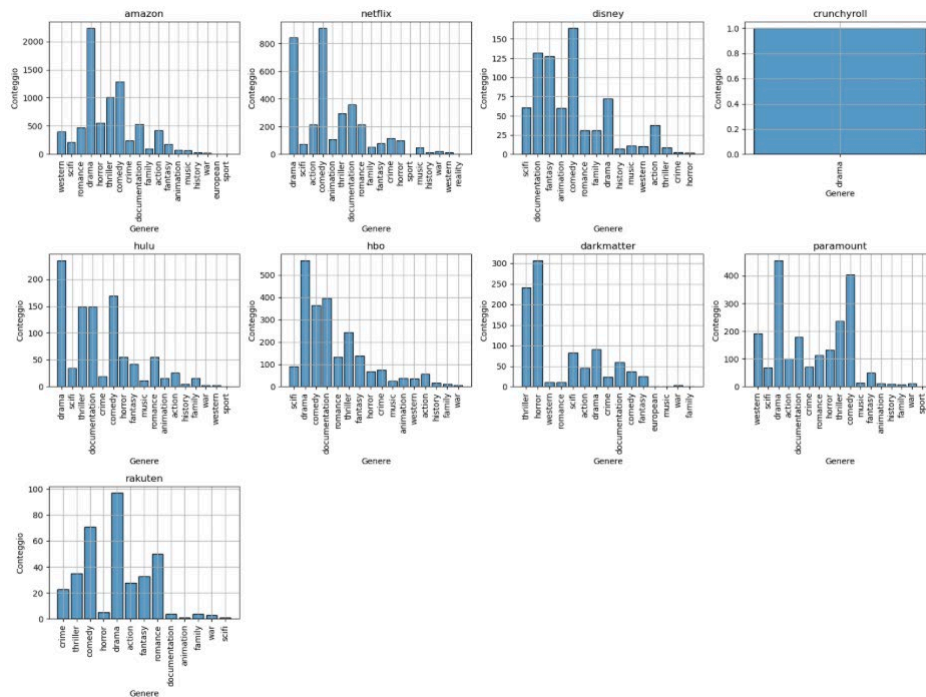
df.replace([float('inf'), -float('inf')], pd.NA, inplace=True)

num_services = len(streaming_services)
num_cols = 4
num_rows = math.ceil(num_services / num_cols)

plt.figure(figsize=(16, num_rows * 4))
for i, service in enumerate(streaming_services, 1):
    plt.subplot(num_rows, num_cols, i)
    subset = df[df['streaming_service'] == service]
    sns.histplot(data=subset, x='main_genre', discrete=True, shrink=0.8)

    plt.title(f'{service}')
    plt.xlabel('Genere')
    plt.ylabel('Conteggio')
    plt.xticks(rotation=90)
    plt.grid(True)

plt.tight_layout()
plt.show()
```



Restituiamo, alla fine, il conteggio dei generi principali per ciascuna piattaforma di streaming presa in considerazione.

```
In [5]: df = pd.DataFrame(movies)

genre_counts = df.groupby(['streaming_service', 'main_genre']).size().reset_index(name=
print(genre_counts)
```

	streaming_service	main_genre	count
0	amazon	action	419
1	amazon	animation	78
2	amazon	comedy	1289
3	amazon	crime	244
4	amazon	documentary	535
..
125	rakuten	horror	5
126	rakuten	romance	50
127	rakuten	scifi	1
128	rakuten	thriller	35
129	rakuten	war	3

[130 rows x 3 columns]

2. Recommender System

2.1 Sommario

Un sistema di raccomandazione, o **motore di raccomandazione**, è un software progettato per filtrare i contenuti e fornire suggerimenti personalizzati agli utenti, facilitando così le loro decisioni. Ci sono tre principali modalità di creazione delle raccomandazioni: l'approccio **collaborativo**, l'approccio **basato sui contenuti** e l'approccio **ibrido**.

Abbiamo scelto di adottare l'approccio del **Content-based Filtering**, in quanto il dataset selezionato ci ha permesso di accedere ad una vasta gamma di informazioni dettagliate sui film, organizzate in categorie. Questo metodo si basa sull'analisi del contenuto di un elemento e sul confronto con il profilo dell'utente. Il contenuto di un elemento include la sua descrizione, i suoi attributi, le parole chiave e le etichette associate. Questi dati vengono confrontati con il profilo dell'utente, che riflette le sue preferenze.

Nel nostro caso specifico, non disponendo di un profilo utente predefinito, abbiamo optato per un'interazione diretta con l'utente. Quest'ultimo inserisce i dati relativi al videogioco che ha scelto o che preferisce, e riceve una raccomandazione personalizzata basata su tali informazioni.

```
Inserisci un valore: 1
Recommender System

BENVENUTO NEL RECOMMENDER SYSTEM

Digita le caratteristiche del film su cui vuoi che si avii la raccomandazione
Inseisci i dati con la lettera maiuscola iniziale

Inserisci il titolo:
The Lucky Texan
Inserisci il genere:
Western
Inserisci l'anno di uscita:
1934
```

```
Questi sono i dati del film che hai inserito:

      title    genre  year
0  The Lucky Texan  Western  1934

È corretto? (y/n): y

Inizio ricerca dei film...
```

2.2 Scelta metrica da impiegare

Prima di procedere con lo sviluppo del recommender system, ci siamo dedicati allo studio delle metriche per il calcolo della similarità.

Abbiamo optato per l'utilizzo della "Correlazione di Pearson".

L'indice di correlazione di Pearson è un metodo utilizzato per esaminare la possibile relazione lineare tra due variabili quantitative continue. Questa misura indica sia la forza che la direzione di una relazione lineare. I valori che può assumere variano da -1 a +1. Un valore pari a uno di questi estremi indica una correlazione lineare perfetta, rispettivamente positiva o negativa.

2.3 Realizzazione

Per realizzare il recommender system Content-based filtering, abbiamo fatto uso della libreria open source python **SKLearn**, che ci ha permesso di utilizzare una serie di algoritmi dedicati al "Natural Language Processing".

Per calcolare le raccomandazioni, abbiamo utilizzato tre categorie del dataset: "title", "genre" e "release_year". Tuttavia, poiché i dati di ciascuna categoria non erano nel formato adatto per confrontare la similarità tra diversi film, è stato necessario trasformare queste tre categorie in vettori di parole. Questi vettori rappresentano una versione vettoriale delle parole presenti in un "documento", in cui ogni vettore trasmette un significato semantico associato a ciascuna parola.

Utilizzando la tecnica TF-IDF (term frequency-inverse document frequency), i vettori vengono calcolati generando una matrice in cui ogni colonna rappresenta una parola ottenuta dalla concatenazione delle categorie selezionate e, allo stesso tempo, rappresenta il film stesso.

In sostanza, il punteggio TF-IDF misura la frequenza di una parola in un documento, ponderata in base al numero di documenti in cui quella parola compare. Questo approccio serve a diminuire l'importanza delle parole che appaiono spesso, riducendo così il loro impatto nel calcolo finale. La libreria SKLearn fornisce una classe integrata chiamata 'TfidfVectorizer', che si occupa di generare la matrice TF-IDF.

```
1 usage
def vectorize_data(movie_data):
    vectorizer = TfidfVectorizer(analyzer='word')
    tfidf_matrix = vectorizer.fit_transform(movie_data['all_content'])
    return tfidf_matrix
```

Una volta ricavata possiamo procedere al calcolo della similarità, con l'utilizzo della correlazione di Pearson.

Si creerà, così, una matrice in cui verranno riportati i punteggi di similarità tra ciascun film e tutti gli altri. In altre parole, ogni film sarà rappresentato nella matrice da una colonna-vettore, in cui ogni elemento indicherà il punteggio di similarità rispetto ad ogni altro film del dataset.

```
tfidf_matrix = vectorize_data(movie_data)
tfidf_matrix_array = tfidf_matrix.toarray()

print("\nInizio ricerca dei film...")

indices = pd.Series(movie_data['title'].index)
id = indices[index]

correlation = []

for i in range(len(tfidf_matrix_array)):
    correlation.append(pearsonr(tfidf_matrix_array[id], tfidf_matrix_array[i]))
correlation = list(enumerate(correlation))
sorted_corr = sorted(correlation, reverse=True, key=lambda x: x[1])[1:6]
movie_index = [i[0] for i in sorted_corr]
```

Per effettuare la raccomandazione vera e propria, useremo un “reverse mapping” dei nomi dei film e degli indici presenti nel dataframe, ossia un sistema che permetta di trovare l’indice del film a partire dal suo nome.

Verranno mostrati, in conclusione, i primi 5 film più simili a quello inserito in input dall’utente.

```
Ecco una lista di 5 film più simili a quello indicato, con una predizione sulla categoria star:
```

	title	release_year	...	star	star_prediction
16302	The Lucky Texan	1934	...	3.0	3.0
835	Lucky Terror	1936	...	3.0	3.0
16486	Lucky Terror	1936	...	3.0	3.0
272	The Tall Texan	1953	...	3.0	3.0
6330	Lucky Ghost	1942	...	3.0	3.0

```
[5 rows x 6 columns]
```

2.4 Classificazione

Oltre al Recommender System, abbiamo introdotto una classificazione degli elementi del dataset basata su una nuova categoria chiamata “star”. Questa categoria è stata creata da noi utilizzando categorie già esistenti relative al rating dei film.

```
#creiamo la categoria star
movie_data['star'] = ((movie_data['imdb_score'] + movie_data['tmdb_score']) / 2)

#assegniamo i dati alla sezione corretta
movie_data.loc[(movie_data['star'] >= 7.5), ['star']] = 5
movie_data.loc[(movie_data['star'] < 7.5) & (movie_data['star'] >= 5), 'star'] = 4
movie_data.loc[(movie_data['star'] < 5) & (movie_data['star'] >= 3), 'star'] = 3
movie_data.loc[(movie_data['star'] < 3) & (movie_data['star'] >= 1.5), 'star'] = 2
movie_data.loc[(movie_data['star'] < 1.5)] = 1
```

Nello specifico, abbiamo scelto di mantenere tra le categorie presenti nel dataset “imdb_score” e “tmdb_score”.

Il valore della categoria “star” si basa sulla media aritmetica dei valori di questi due campi. A seconda del valore della media ottenuto abbiamo stabilito 5 range all’interno dei quali far rientrare i dati già presenti nel dataset.

Inoltre, sono state condotte una serie di valutazioni sul modello scelto, seguita da una fase di ottimizzazione dello stesso. Per quanto riguarda la scelta del modello di classificazione, abbiamo scelto di utilizzare il KNeighborsClassifier, disponibile nella libreria SKLearn.

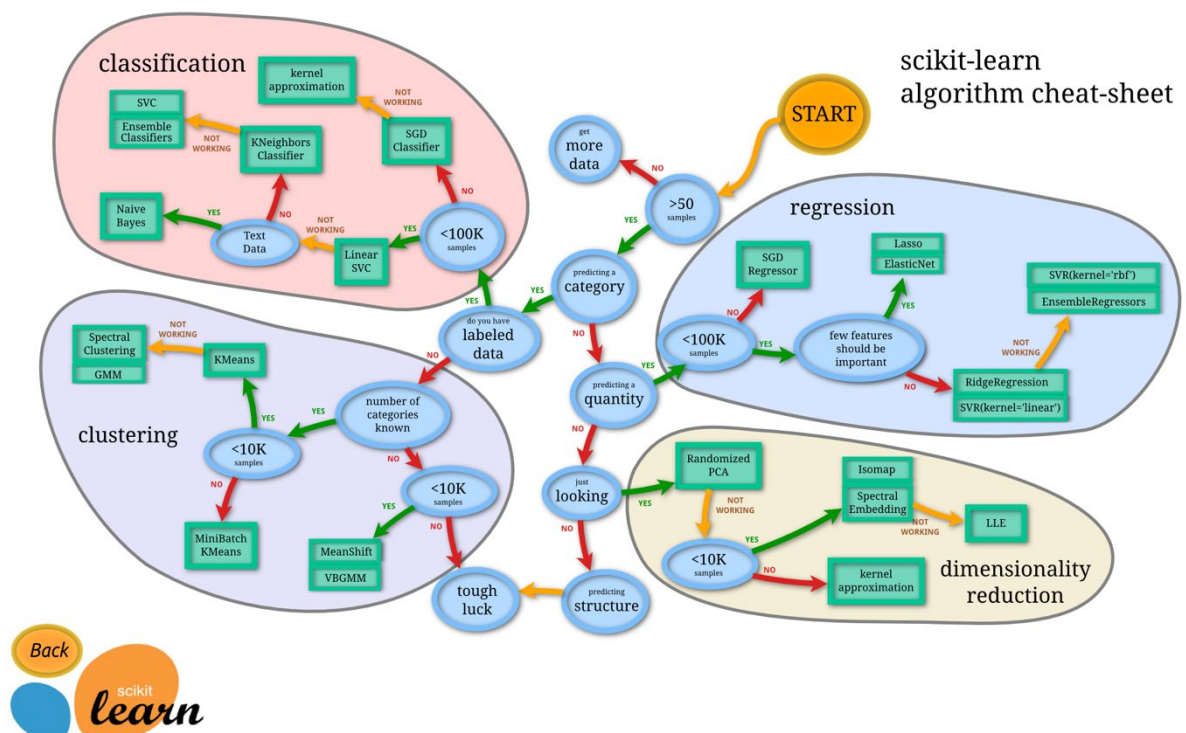
Modello adottato

Il K-nearest neighbors, o K-NN, è un algoritmo impiegato nel riconoscimento di pattern per classificare oggetti in base alle caratteristiche dei loro vicini. In pratica, l’input consiste nei k esempi di addestramento più vicini all’oggetto in questione all’interno dello spazio delle funzionalità.

Sebbene possa essere applicato sia a problemi di regressione che di classificazione, è comunemente utilizzato per la classificazione, basandosi sull’assunto che oggetti simili tendano ad essere vicini tra loro. L’output del processo è l’assegnazione di una classe.

La classificazione avviene tramite un voto di maggioranza tra i vicini più prossimi: l’oggetto viene assegnato alla classe che è più frequente tra i suoi k vicini più vicini (dove k è un numero intero positivo, generalmente piccolo). Nel caso in cui k sia pari a 1, l’oggetto viene semplicemente classificato nella stessa classe del suo vicino più prossimo.

Per selezionare il modello, abbiamo utilizzato una mappa disponibile sul sito della libreria Sklearn. Questa mappa guida nella ricerca del miglior estimatore di machine learning in base agli obiettivi specifici.



Si può osservare che, seguendo le linee che conducono al KNN, viene delineata la situazione del nostro progetto: previsione di una categoria (star), disponibilità di dati categorizzati e impiego di dati non testuali.

Dataset

Per ottenere un dataset adatto all'uso del modello selezionato e per la classificazione, esso è stato diviso in due parti: training e test. La sezione di training verrà utilizzata per addestrare il modello, che sarà poi valutato sulla parte rimanente, ossia il test. Nel nostro caso, abbiamo deciso di riservare l'80% dei dati per il training e il 20% per il test.

Un aspetto fondamentale è l'assegnazione casuale degli elementi del dataset alle due parti, assicurando che entrambe siano rappresentative del dataset originale e ben distribuite. Questo si può ottenere impostando il parametro "random-state" a un numero intero qualsiasi, dato che non è un parametro da ottimizzare.

Infine, poiché alcuni problemi di classificazione presentano categorie con numero di esempi sbilanciato, è preferibile suddividere il dataset in training e test in modo che mantenga la stessa proporzione di esempi per ogni categoria presente nel dataset originale. Questa tecnica è nota come "split stratificato" e può essere implementata modificando il parametro "stratify" utilizzando la componente "y" del dataset originale.

```
#dividiamo il dataset in due parti, 80% destinato al training e 20% destinato al testing
X_train, X_test, y_train, y_test = train_test_split(*arrays: x, y, test_size=0.2, random_state=1, stratify=y)
```

Pre-processing dei dati

Abbiamo eseguito la scalatura dei dati per uniformare il range di variazione delle caratteristiche del dataset. Questo processo ha contribuito a migliorare la qualità dei risultati finali, poiché la scalatura riduce il tempo necessario all'algoritmo di apprendimento per convergere al risultato finale.

```
#traformiamo i dati per renderli adeguati
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
predict_data = scaler.transform(predict_data)
```

Successivamente, il modello viene allenato sulla porzione di training utilizzando i parametri predefiniti, sfruttando le funzioni messe a disposizione dalla libreria SKLearn.

```
print("\n\nComposizione iniziale del modello con iper-parametri di base...")
knn = KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', p=2, metric='minkowski', metric_params=None, n_jobs=None)
knn.fit(X_train, y_train)
```


Miglioramento del modello e Hyperparameters Tuning

Dopo aver sperimentato il modello costruito con i parametri predefiniti, abbiamo tentato di elaborare delle predizioni per la categoria “star”. Successivamente, abbiamo valutato il modello per verificare l’accuratezza delle previsioni ottenute.

```
Composizione iniziale del modello con iper-parametri di base...

Predizioni dei primi 5 elementi: [3. 3. 3. 3. 3.] Valori effettivi: [3. 3. 3. 3. 3.]

Valutazione del modello...

Classification Report:

```

	precision	recall	f1-score	support
1.0	1.00	1.00	1.00	1
2.0	0.78	0.82	0.80	34
3.0	1.00	1.00	1.00	3709
accuracy			1.00	3744
macro avg	0.93	0.94	0.93	3744
weighted avg	1.00	1.00	1.00	3744

```
ROC Score: 0.9994215045119063

Proviamo a migliorare il nostro modello determinando gli iper-parametri ottimali con 'Grid Search':
```

Abbiamo utilizzato un “Classification Report” della libreria sklearn.metrics per individuare diverse metriche di valutazione e successivamente calcolato il ROC AUC score, ottenendo un valore di 0.99.

Il ROC AUC score (area sotto la curva ROC) rappresenta la misura dell’area bidimensionale sotto l’intera curva ROC e fornisce una valutazione complessiva delle prestazioni di un modello di classificazione su tutte le possibili soglie di classificazione.

Successivamente, abbiamo migliorato il nostro modello avviando una ricerca dei parametri ottimali, con l’obiettivo di incrementare il ROC score e altre metriche di valutazione (quali precision, recall, ...).

Abbiamo deciso di utilizzare una RandomizedSearchCV per la ricerca dei parametri, combinata con una cross-validation tramite RepeatedKFold. La RandomizedSearchCV è una tecnica che esplora combinazioni casuali di iper-parametri per trovare la migliore configurazione per il modello di classificazione. Rispetto alla “Grid Search”, è più rapida, poiché seleziona le combinazioni di parametri in modo casuale, e spesso più efficace, riuscendo ad individuare parametri significativamente migliori.

```
cvFold = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)
randomSearch = RandomizedSearchCV(estimator=knn, cv=cvFold, param_distributions=hyperparameters)
```

Una volta scelto l’approccio da utilizzare, sono stati creati una serie di parametri di ricerca, ovvero il numero k di neighbors, la tipologia di “weights” e la tipologia di metrica da utilizzare.


```
n_neighbors = list(range(1, 30))
weights = ['uniform', 'distance']
metric = ['euclidean', 'manhattan', 'hamming']
```

La ricerca è stata eseguita 15 volte, con ogni iterazione in cui i parametri e il ROC score ottenuto venivano memorizzati in un dizionario appositamente creato. Successivamente, il dizionario è stato ordinato in base al ROC score per identificare il miglior set di parametri tra tutte le esecuzioni. Una volta individuati i parametri ottimali, il modello è stato riconfigurato con questi e utilizzato per generare predizioni sulla categoria “star”.

Proviamo a migliorare il nostro modello determinando gli iper-parametri ottimali con 'Grid Search':

GRID SEARCH:

Best Weights: distance
Best Metric: manhattan
Best n_neighbours: 25

Ricomponiamo il modello utilizzando i nuovi iper-parametri:

ROC Score: 0.9998801625677819

Il risultato del ROC score risulta effettivamente incrementato, da 0.9994 a 0.9998.

Predizione su nuovi dati

Dopo aver ottimizzato i parametri del nostro modello, abbiamo utilizzato la nuova configurazione per effettuare predizioni sui film consigliati all'utente nella categoria star.

Ora possiamo procedere alla fase di recommendation...

Ecco una lista di 5 film più simili a quello indicato, con una predizione sulla categoria star:

	title	release_year	genre	streaming_service	star	\
16302	The Lucky Texan	1934	western	paramount	3.0	
835	Lucky Terror	1936	western	amazon	3.0	
16486	Lucky Terror	1936	western	paramount	3.0	
272	The Tall Texan	1953	western	amazon	3.0	
6330	Lucky Ghost	1942	comedy	amazon	3.0	

Possiamo osservare che, sebbene le predizioni non siano perfette al 100% rispetto ai risultati precedenti, hanno comunque mostrato dei miglioramenti.

3. KB

3.1 Introduzione

Una Knowledge Base KB è una raccolta organizzata di informazioni e regole che un sistema può impiegare per risolvere problemi, rispondere a domande o prendere decisioni attraverso la conoscenza che contiene.

Essa rappresenta **fatti** del mondo sulle quali applica **regole**, ovvero linee guida che definiscono come tali fatti possono essere utilizzati o combinati per derivare nuove regole.

Possiamo dunque definire una KB come un insieme di proposizioni, dette assiomi, le quali si presume siano vere, e che consentono di creare un ambiente capace di raccogliere, organizzare e diffondere la conoscenza.

Nel contesto del nostro caso di studio, la KB viene impiegata per permettere all'utente di formulare domande relative all'ambito di interesse approfondito.

3.2 Gestione della KB

Per la gestione della KB abbiamo impiegato l'estensione **Pyswip** del linguaggio Python, che si basa sul linguaggio di programmazione logica **Prolog**.

Grazie alle funzionalità offerte da Prolog abbiamo creato e popolato la knowledge base con fatti, che vengono generati automaticamente prelevando i dati pertinenti dal dataset.

Successivamente, abbiamo implementato le regole basandoci sui fatti presenti nel KB e su come potrebbe avvenire l'interazione con l'utente.

3.3 Fatti

I fatti rappresentano gli assiomi sempre veri della KB e costituiscono la base delle informazioni su cui operano le regole.

- 1) "title (id, title)"
Rappresenta la relazione tra un film e il suo titolo.
- 2) "description (id, description)"
Rappresenta la relazione tra un film e la sua trama.
- 3) "release_year (id, release_year)"
Rappresenta la relazione tra un film e l'anno del suo rilascio.
- 4) "runtime (id, runtime)"
Rappresenta la relazione tra un film e la sua durata.
- 5) "genre (id, genre)"
Rappresenta la relazione tra un film e il suo genere.
- 6) "production_countries (id, production_countries)"
Rappresenta la relazione tra un film e il suo paese di produzione.
- 7) "streaming_service (id, streaming_service)"
Rappresenta la relazione tra un film e la piattaforma di streaming su cui vederlo.
- 8) "actors (id, actors)"
Rappresenta la relazione tra un film e il nome degli attori che vi recitano.
- 9) "monthly_subscription_cost (id, monthly_subscription_cost)"
Rappresenta la relazione tra un film ed il costo di abbonamento mensile della piattaforma per vederlo in streaming.

```

import pandas as pd

def clean_string(s):
    """Pulisce e normalizza una stringa per l'uso in Prolog."""
    return s.replace("'", "").encode('ascii', 'ignore').decode()

class KnowledgeBase:
    def createKnowledgeBase(self):
        # Lettura del dataset
        df = pd.read_csv('pre_processed_dataset.csv')

        # Pulizia dei dati
        columns_to_string = ['title', 'description',
                             'production_countries', 'streaming_service', 'genre', 'actors']
        df[columns_to_string] = df[columns_to_string].apply(lambda x:
x.apply(clean_string))

        # Creazione del file Prolog
        with open('KB.pl', 'w') as f:
            # Direttiva per predicati discontinui
            f.write(":- discontinuous title/2.\n")
            f.write(":- discontinuous description/2.\n")
            f.write(":- discontinuous release_year/2.\n")
            f.write(":- discontinuous runtime/2.\n")
            f.write(":- discontinuous genre/2.\n")
            f.write(":- discontinuous actors/2.\n")
            f.write(":- discontinuous production_countries/2.\n")
            f.write(":- discontinuous streaming_service/2.\n")
            f.write(":- discontinuous monthly_subscription_cost/2.\n")

            # Scrittura dei fatti
            for index, row in df.iterrows():
                f.write(f"title({row['id']}, '{row['title']}').\n")
                f.write(f"description({row['id']},
' {row['description']}').\n")
                f.write(f"release_year({row['id']},
{row['release_year']}).\n")
                f.write(f"runtime({row['id']}, {row['runtime']}).\n")
                f.write(f"genre({row['id']}, '{row['genre']}').\n")
                f.write(f"actors({row['id']}, '{row['actors']}').\n")
                f.write(f"production_countries({row['id']},
' {row['production_countries']}').\n")
                f.write(f"streaming_service({row['id']},
' {row['streaming_service']}').\n")
                f.write(f"monthly_subscription_cost({row['id']},
{row['monthly_subscription_cost']}).\n")
            # Creazione della base di conoscenza
kb = KnowledgeBase()
kb.createKnowledgeBase()

```

Abbiamo utilizzato la direttiva **:- discontinuous** in quanto le clausole dei predicati definiti all'interno della KB sono organizzate in modo non contiguo nel file, poiché raggruppate in modo contiguo per ogni film e non per predicato.

Esempio di sezione della knowledge base:

```
title(100001, 'The Lucky Texan').
description(100001, 'Jerry Mason, a young Texan, and Jake Benson, an old
rancher, become partners and strike it rich with a gold mine. They then
find their lives complicated by bad guys and a woman.').
release_year(100001, 1934).
runtime(100001, 61).
genre(100001, 'western').
actors(100001, '[John Wayne, Barbara Sheldon, Lloyd Whitlock, "George Gabby
Hayes", Yakima Canutt, Eddie Parker, Gordon De Main, Earl Dwire, Jack
Rockwell, Artie Ortego, Tex Palmer, Tex Phelps, George Morrell, Robert N.
Bradbury]').
production_countries(100001, 'US').
streaming_service(100001, 'amazon').
monthly_subscription_cost(100001, 14.99).
```

3.4 Regole

Le regole rappresentano il metodo attraverso il quale l'utente può interagire con la knowledge base, mediante domande, senza la necessità di conoscere la sintassi precisa.

Le regole sfruttano i fatti presenti nella KB per fornire all'utente le informazioni da lui richieste.

Sono stati previsti due task principali che l'utente potrebbe richiedere alla knowledge base:

Scegliere quale funzione eseguire:

- 1) Trova un film date determinate caratteristiche
- 2) Sulla base delle preferenze dell'utente, trovare la piattaforma migliore per i film selezionati.
- 3) Esci

3.5 Trova film date determinate caratteristiche

Il **primo task** consiste nell'ottenere tutti i film che soddisfano determinate caratteristiche indicate dall'utente.

Per fare ciò, sono state create una serie di regole corrispondenti a tutte le possibili scelte possibili da parte dell'utente.

Se ne presenta una per ogni tipo come esempio.

```
"film_recente(ID) :- relase_year(ID,Uscita), Uscita > 2010."
```

La regola afferma che un film con ID è considerato recente se il suo anno di uscita è successivo al 2010.

```
"film_western(ID) :- main_genre(ID, 'western')."
```

La regola afferma che un film con ID è considerato di un determinato genere se il suo genere principale è (in questo caso) 'western'.

```
"film_breve_durata(ID) :- runtime(ID, Durata), Durata <= 60."
```

La regola afferma che un film con ID è considerato di breve durata se la sua durata è minore o uguale a 60.

```
"recente_western(ID) :- film_recente(ID), film_western(ID)."
```

La regola afferma che un film con ID è considerato recente e di un determinato genere se soddisfa contemporaneamente le condizioni di film recente e di genere specifico.

"recente_western_breve(ID) :- recente_western(ID) , film_breve_durata(ID) ."

La regola afferma che un film con ID è considerato recente, di un determinato genere e di breve durata se soddisfa contemporaneamente le condizioni di film breve e di un determinato genere, e la condizione di breve durata.

Questa regola rappresenta la regola finale che a sua volta chiama le precedenti in base alle occorrenze, per poter rappresentare tutte le combinazioni vi sono 171 regole di questo tipo.

Esempio di esecuzione:

```
Inserisci il periodo relativo all'anno di uscita del film
1) recente
2) tra 2000 e 2010
3) pre 2000
3
Inserisci il genere cui vorresti appartenga il film
1) western
2) scifi
3) romance
4) drama
5) horror
6) thriller
7) comedy
8) crime
9) documentation
10) family
11) action
12) fantasy
13) animation
14) music
15) history
16) war
17) european
18) sport
19) reality
1
```

```

Inserisci la durata del film
1) breve
2) media
3) lunga
2
Gli ID dei film che rispettano i filtri sono:
100001
100402
102960
104416
104421
105772
107399
107469

```

3.6 Sulla base delle preferenze dell'utente, trova la piattaforma migliore per i film selezionati

Il **secondo task** consiste nell'ottenere la piattaforma di streaming migliore da consigliare all'utente per la selezione dei film.

Per fare ciò, si tiene conto delle scelte dell'utente effettuate al primo task per individuare i film target, inoltre è stata creata una nuova regola che tenga conto del costo di sottoscrizione delle piattaforme.

```

def find_best_streaming_platform(prolog, film_ids):
    platform_count = {}
    price_filters = []

    # Selezione del prezzo
    while not price_filters:
        try:
            price_input = int(input("Seleziona il tipo di
sottoscrizione:\n" "1) economica\n2) media\n3) costosa\n"))
            if price_input == 1:
                price_filters = ["prezzo_economy"] # Solo economico
            elif price_input == 2:
                price_filters = ["prezzo_economy", "prezzo_medio"] #
Economico o medio
            elif price_input == 3:
                price_filters = ["prezzo_economy", "prezzo_medio",
"prezzo_costoso"] # Economico, medio o costoso
            else:
                print("Input non valido. Utilizzare solo 1, 2 o 3.")
        except ValueError:
            print("Input non valido. Inserisci un numero.")

    # Iteriamo su tutti gli ID dei film trovati e cerchiamo su quale

```

```

piattaforma si trovano
    for film_id in film_ids:
        query = f"streaming_service({film_id}, Piattaforma)" # Usare
il predicato corretto per la piattaforma
        try:
            results = list(prolog.query(query))
            for result in results:
                piattaforma = result["Piattaforma"]

                # Verifichiamo il prezzo della piattaforma con uno
dei predicati selezionati
                price_match = False
                for price_filter in price_filters:
                    price_query = f"{price_filter}({film_id})"
                    price_results = list(prolog.query(price_query))
                    if price_results: # Se il film è disponibile per
quel tipo di sottoscrizione
                        price_match = True
                        break

                if price_match:
                    if piattaforma in platform_count:
                        platform_count[piattaforma] += 1
                    else:
                        platform_count[piattaforma] = 1
        except Exception as e:
            print(f"Errore durante l'esecuzione della query per
{film_id}: {e}")

    if platform_count:
        # Troviamo la piattaforma con il maggior numero di film
        best_platform = max(platform_count, key=platform_count.get)
        print(f"La migliore piattaforma consigliata è:
{best_platform}")
        print(f"Numero di film disponibili su {best_platform}:
{platform_count[best_platform]}")
    else:
        print("Non è stato trovato nessun film su alcuna piattaforma
con il prezzo selezionato.")

```

E' stata implementata una funzione denominata *"find_best_streaming_platform"* la quale tiene conto del set di film target processato in precedenza e restituisce la miglior piattaforma sulla base del numero di film target contenuti in ciascuna piattaforma, e sul prezzo di sottoscrizione di ciascuna di esse.

La miglior piattaforma restituita sarà dunque quella che, sulla base del prezzo di sottoscrizione indicato dall'utente, contiene il maggior numero di film target.

Di seguito le regole per la definizione del prezzo di sottoscrizione:

"prezzo_economy(ID) :- monthly_subscription_cost(ID, Cost), Cost < 9.99."

La regola afferma che una piattaforma di streaming, contenente film con ID, presenta un prezzo di sottoscrizione mensile economico se il suo costo è inferiore a 9,99.

"prezzo_medio(ID) :- monthly_subscription_cost(ID, Cost), Cost = 9.99."

La regola afferma che una piattaforma di streaming, contenente film con ID, presenta un prezzo di sottoscrizione mensile medio se il suo costo è uguale a 9,99.

"prezzo_costoso(ID) :- monthly_subscription_cost(ID, Cost) , Cost > 9.99."

La regola afferma che una piattaforma di streaming, contenente film con ID, presenta un prezzo di sottoscrizione mensile costoso se il suo costo è superiore a 9,99.

Esempio di esecuzione:

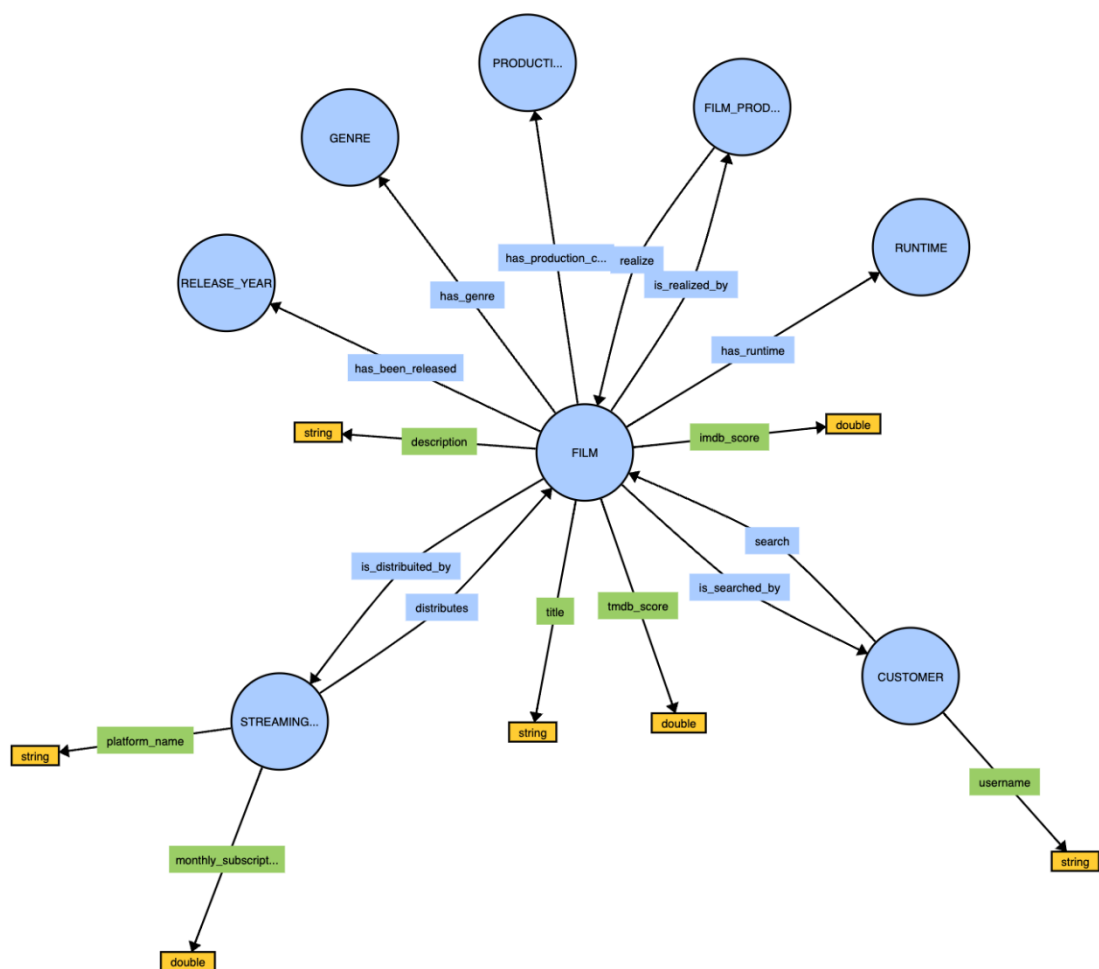
```
Scegliere quale funzione eseguire:
1) Trova un film date determinate caratteristiche
2) Sulla base delle preferenze dell'utente, trovare la piattaforma migliore per i film selezionati.
3) Esci
2
Seleziona il tipo di sottoscrizione:
1) economica
2) media
3) costosa
1
La migliore piattaforma consigliata è: disney
Numero di film disponibili su disney: 4
```


4. Ontologia

4.1 Sommario

In informatica un'ontologia è un sistema di rappresentazione formale utilizzato per descrivere la realtà e la conoscenza. È una struttura dati che permette di definire le entità (come oggetti, idee, ecc...) e le loro intenzioni all'interno di un determinato ambito di conoscenza.

Abbiamo scelto di sviluppare un'ontologia che potesse rappresentare i differenti servizi di streaming di film online. Questo consente agli utenti di valutare le piattaforme di streaming migliori (Amazon, Netflix, Disney+, ...) e di visualizzare informazioni come generi presenti, anni di uscita dei titoli e prezzi degli abbonamenti, ...

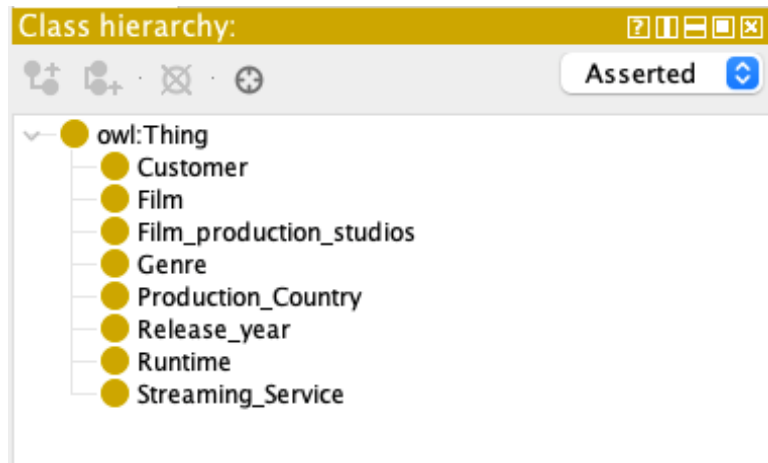


4.2 Protégé

Per la realizzazione dell'Ontologia abbiamo scelto di utilizzare l'editor di ontologie open source Protégé. L'Ontologia prevede la presenza di diverse classi, la cui maggior parte rappresenta

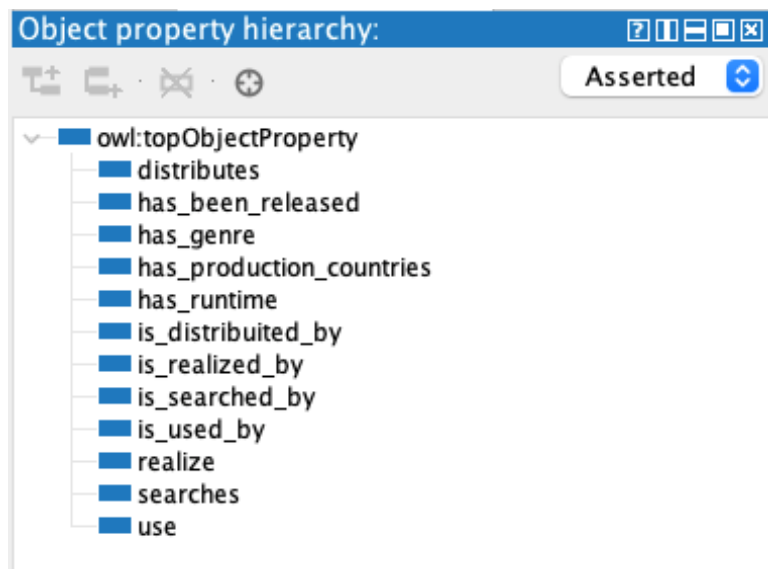
una categoria legata al film, che lo descrive nel dettaglio (release_year, genre, streaming_service,...).

Sono presenti anche le classi rappresentanti gli “attori” che fanno uso delle diverse piattaforme (customer, film_production_studios).



Classi dell'Ontologia

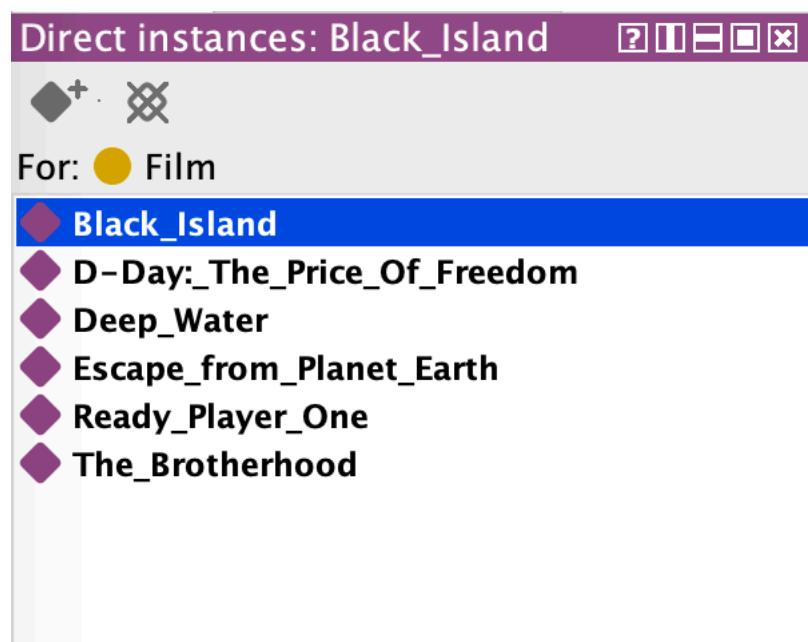
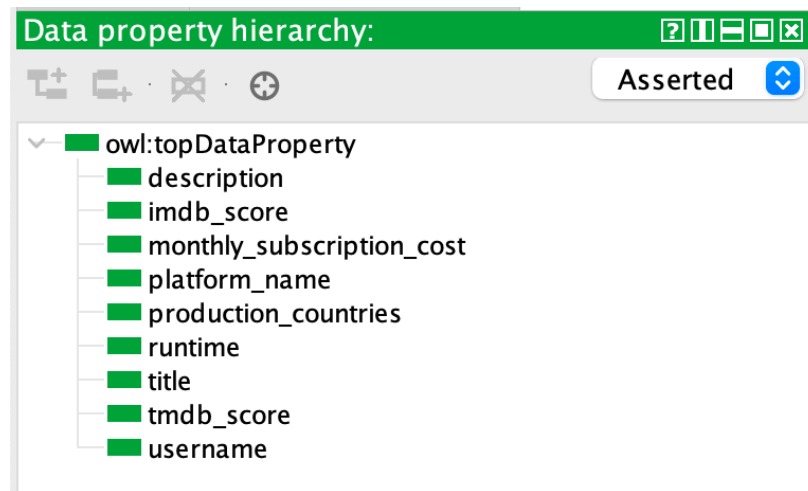
Dopo aver creato le classi abbiamo creato, inoltre, una serie di proprietà: object-properties e data-properties. Queste proprietà servono a connettere due individui appartenenti a classi uguali o differenti (object properties) o a collegare un individuo a un tipo di dato primitivo (data properties).



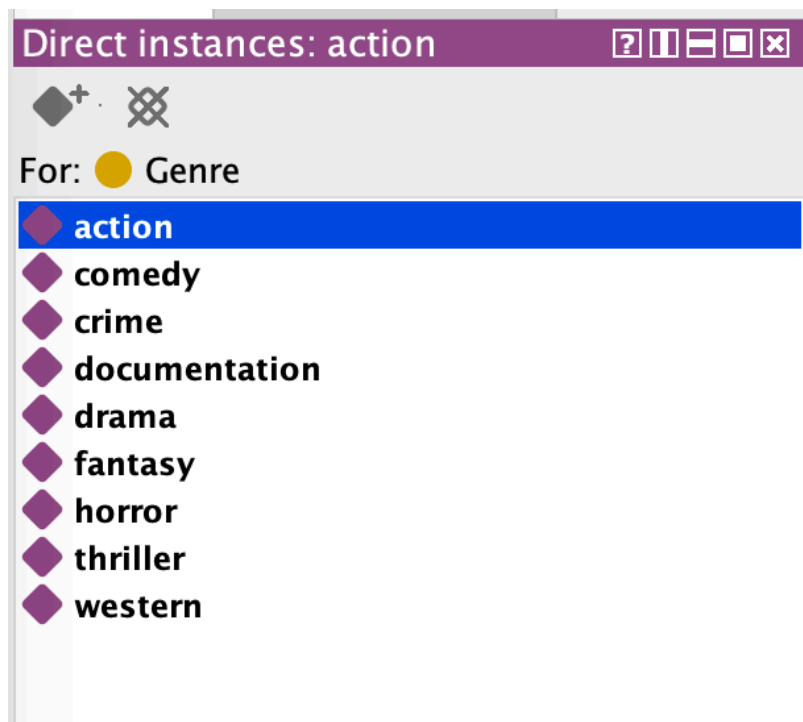
Object-properties

Data-properties

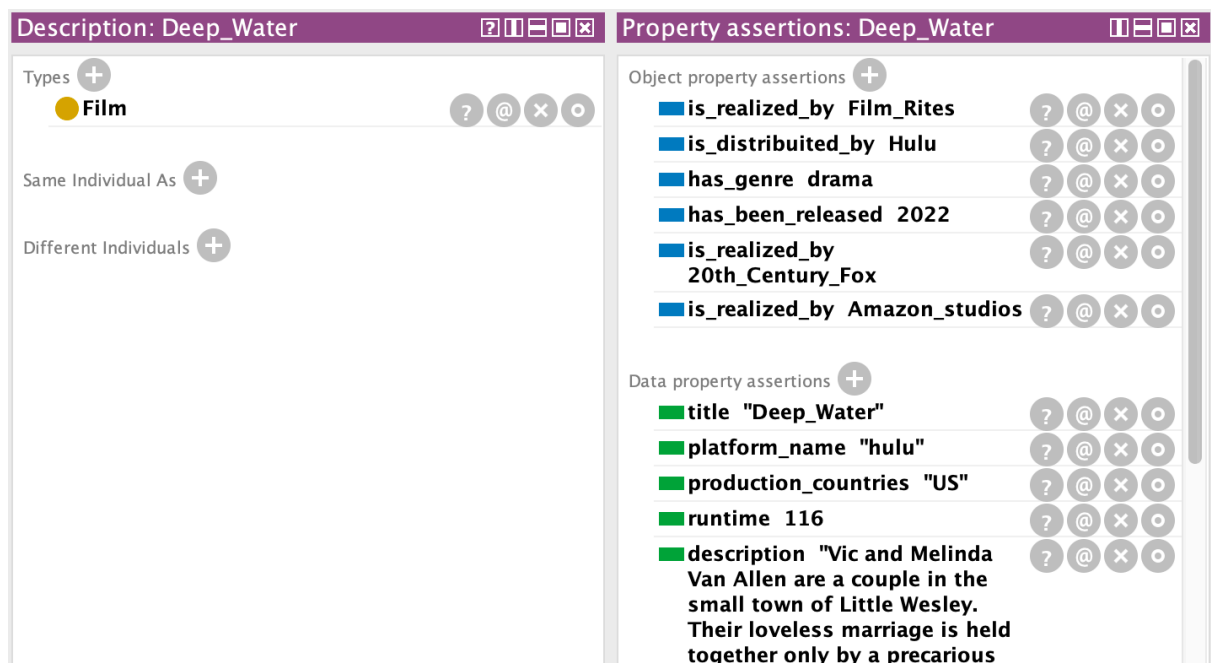
Successivamente siamo passati alla realizzazione degli individui stessi che rappresentano alcuni esempi di film presenti nel dataset.



Individui della classe Film



Individui della classe Genre



Esempio di individuo con properties annesse

Possiamo anche interrogare l'ontologia tramite l'utilizzo di due tipologie differenti di query, DL query e SPARQL query.

SPARQL query:		
<pre> PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> PREFIX owl: <http://www.w3.org/2002/07/owl#> PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#> PREFIX xsd: <http://www.w3.org/2001/XMLSchema#> PREFIX O: <http://www.semanticweb.org/gabrielepaladini/ontologies/2024/6/untitled-ontology-7#> SELECT ?Film ?production_countries ?runtime WHERE { ?Film O:production_countries ?production_countries. ?Film O:runtime ?runtime. }</pre>		
Film	production_countries	runtime
Deep_Water	"US"	"116"^^<http://www.w3.org/2001/XMLSchema#decimal>
The_Brotherhood	"US"	"84"^^<http://www.w3.org/2001/XMLSchema#decimal>
Black_Island	"DE"	"105"^^<http://www.w3.org/2001/XMLSchema#decimal>
Ready_Player_One	"US"	"140"^^<http://www.w3.org/2001/XMLSchema#decimal>
Escape_from_Planet_Earth	"CA"	"89"^^<http://www.w3.org/2001/XMLSchema#decimal>
D-Day_The_Price_Of_Freedom	"US"	"53"^^<http://www.w3.org/2001/XMLSchema#decimal>

SPARQL query con visualizzazione dei Film, Production_countries e runtime

SPARQL query:	
<pre> PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> PREFIX owl: <http://www.w3.org/2002/07/owl#> PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#> PREFIX xsd: <http://www.w3.org/2001/XMLSchema#> PREFIX O: <http://www.semanticweb.org/gabrielepaladini/ontologies/2024/6/untitled-ontology-7#> SELECT ?Film ?is_distributed_by WHERE { ?Film O:is_distributed_by ?is_distributed_by. ?Film O:imdb_score 5.8 }</pre>	
Film	is_distributed_by
Escape_from_Planet_Earth	Netflix
Deep_Water	Hulu

SPARQL query con visualizzazione dei film che hanno come object properties "is_distributed_by" e un "imdb_score" pari a 5.8

DL query: ⏏ ⏏ ⏏

Query (class expression)

Film **that** has_genre **value** scifi **and** has_been_released **value** 2018|

Execute Add to ontology

Query results

Instances (1 of 1)

Ready_Player_One

?

Query for

☐ Direct superclasses
☐ Superclasses
☐ Equivalent classes

DL-query con ricerca di film che hanno come genere “scifi” e come anno di rilascio “2018”

DL query: ⏏ ⏏ ⏏

Query (class expression)

Film **that** has_genre **value** scifi **or** has_genre **value** thriller

Execute Add to ontology

Query results

Instances (4 of 4)

Black_Island

?

Escape_from_Planet_Earth

?

Ready_Player_One

?

The_Brotherhood

?

Query for

☐ Direct superclasses
☐ Superclasses
☐ Equivalent classes
☐ Direct subclasses

DL-query con ricerca di film che hanno come genere “scifi” oppure “thriller”

4.3 Owlready2

Abbiamo utilizzato la libreria Owlready2 per interrogare l’ontologia in Python. Questo strumento ci ha consentito di effettuare query sull’ontologia, che era stata precedentemente creata con l’editor Protégé, in modo facile e intuitivo.

Durante l’esecuzione del programma, l’utente potrà optare per visualizzare le classi, le object properties, le data properties o alcune query di esempio.

Segue il menù iniziale che consente di esplorare la nostra ontologia, doc'è possibile effettuare le varie scelte:

```
BENVENUTO NELL'ONTOLOGIA

Seleziona un'operazione:
1) Visualizzazione Classi
2) Visualizzazione proprietà d'oggetto
3) Visualizzazione proprietà dei dati
4) Esegui query
5) Esci dall'Ontologia

Inserisci un valore: |
```

Selezionando la prima opzione, potremmo visualizzare le classi presenti all'interno della nostra ontologia:

```
Inserisci un valore: 1
CLASSI PRESENTI NELL'ONTOLOGIA:
- Ontologia.owx.Customer
- Ontologia.owx.Film
- Ontologia.owx.Film_production_studios
- Ontologia.owx.Genre
- Ontologia.owx.Release_year
- Ontologia.owx.Streaming_Service

Vorresti esplorare meglio una delle seguenti classi?

1) Film
2) Release_year
3) Streaming_service
4) Genre
5) Customer
6) Film_production_studios
7) No

Inserisci un valore: |
```

Possiamo, inoltre, esaminare da vicino le classi selezionando quella di interesse tra quelle visualizzate a schermo. Nell'immagine di esempio abbiamo selezionato la classe 'Streaming_Service'. Verrà quindi mostrato a schermo un elenco contenente tutti i servizi di streaming presenti all'interno della nostra ontologia:

```
Inserisci un valore: 3

LISTA DEI SERVIZI DI STREAMING PRESENTI:

- Ontologia.owx.Streaming_Service
- Ontologia.owx.Amazon
- Ontologia.owx.Hbo
- Ontologia.owx.Hulu
- Ontologia.owx.Netflix
- Ontologia.owx.Disney+

Vuoi esaminare un'altra classe? (y/n): |
```

Verrà richiesto alla fine dell'analisi di ciascuna delle classi proposte se l'utente desidera esaminare un'altra classe oppure no.

Abbiamo anche la possibilità di visualizzare la 'object properties' contenute all'interno della nostra ontologia, selezionando la seconda opzione:

```
Inserisci un valore: 2

PROPRIETÀ D'OGGETTO PRESENTI NELL'ONTOLOGIA:

- Ontologia.owx.distributes
- Ontologia.owx.has_been_released
- Ontologia.owx.has_genre
- Ontologia.owx.has_production_countries
- Ontologia.owx.has_runtime
- Ontologia.owx.is_distributed_by
- Ontologia.owx.is_realized_by
- Ontologia.owx.is_searched_by
- Ontologia.owx.is_used_by
- Ontologia.owx.realize
- Ontologia.owx.searches
- Ontologia.owx.use
```


Allo stesso modo selezionando la terza opzione abbiamo la possibilità di esaminare le 'data properties' presenti nell'ontologia:

```
Inserisci un valore: 3

PROPRIETÀ DEI DATI PRESENTI NELL'ONTOLOGIA:
- Ontologia.owx.description
- Ontologia.owx.imdb_quotes
- Ontologia.owx.imdb_score
- Ontologia.owx.imdb_votes
- Ontologia.owx.monthly_subscription_cost
- Ontologia.owx.platform_name
- Ontologia.owx.production_countries
- Ontologia.owx.runtime
- Ontologia.owx.title
- Ontologia.owx.tmdb_popularity
- Ontologia.owx.tmdb_quotes
- Ontologia.owx.tmdb_score
- Ontologia.owx.username
```

Adesso presentiamo la possibilità di visualizzare alcune query di esempio, scegliendo la quarta opzione.

Successivamente, ci verrà anche chiesto di selezionare quale delle query disponibili desideriamo eseguire:

```
Seleziona un'operazione:
1) Visualizzazione Classi
2) Visualizzazione proprietà d'oggetto
3) Visualizzazione proprietà dei dati
4) Esegui query
5) Esci dall'Ontologia

Inserisci un valore: 4

1) Lista film presenti su 'Amazon'
2) Lista film di genere 'scifi'
3) Studi di produzione del film 'Deep Water'
4) Torna indietro

Inserisci un valore: |
```

Mostriamo, ora, l'esecuzione delle tre query messe a disposizione dal sistema:

PRIMA QUERY

```
Inserisci un valore: 1
FILM PRESENTI SU AMAZON:
- Ontologia.owx.The_Brotherhood
- Ontologia.owx.D-Day:_The_Price_Of_Freedom
```

SECONDA QUERY

```
Inserisci un valore: 2
FILM DI GENERE SCI-FI:
- Ontologia.owx.Escape_from_Planet_Earth
- Ontologia.owx.Ready_Player_One
- Ontologia.owx.The_Brotherhood
```

TERZA QUERY

```
Inserisci un valore: 3
STUDI DI PRODUZIONE DEL FILM 'DEEP WATER':
- Ontologia.owx.Amazon_studios
- Ontologia.owx.Film_Rites
- Ontologia.owx.20th_Century_Fox
```

5. Conclusioni

Il progetto Movies Recommender rappresenta una solida base per la creazione di un sistema di raccomandazione di film, sfruttando le tecniche implementate.

Tuttavia, vi sono ampi margini di miglioramento ed estensione del progetto. Un'evoluzione naturale consisterebbe nell'ampliamento del dataset attuale, integrando ulteriori dataset provenienti da altre piattaforme di streaming.

La combinazione di questi dataset consentirebbe di arricchire il sistema con una gamma più vasta di film e contenuti, aumentando così la qualità e la varietà delle raccomandazioni. In questo modo, il "Movies Recommender" potrebbe diventare una soluzione ancora più completa, capace di rispondere alle esigenze di un pubblico diversificato e di adattarsi alle nuove tendenze del mercato.