Caso di studio del corso di Ingegneria della Conoscenza

Cinelogic

Sistema di raccomandazione cinematografica che sfrutta una knowledge base logica e un'ontologia OWL per supportare decisioni intelligenti e personalizzate

Gruppo di lavoro:

- Abderrahmane Cherchari, 775262, a.cherchari@studenti.uniba.it
- Giuseppe Pio Impagnatiello, 776480, g.impagnatiello2@studenti.uniba.it

Link progetto GitHub: https://github.com/ramensonoio/ProgettoIcon25-26.git

AA 2025-26

Sommario

1. Introduzione

Negli ultimi anni i servizi di streaming di film e serie televisive sono diventati una componente centrale dell'intrattenimento digitale. Le piattaforme dedicate alla distribuzione di contenuti audiovisivi rappresentano oggi uno degli strumenti più utilizzati per l'accesso a film e serie TV, offrendo cataloghi sempre più vasti e diversificati.

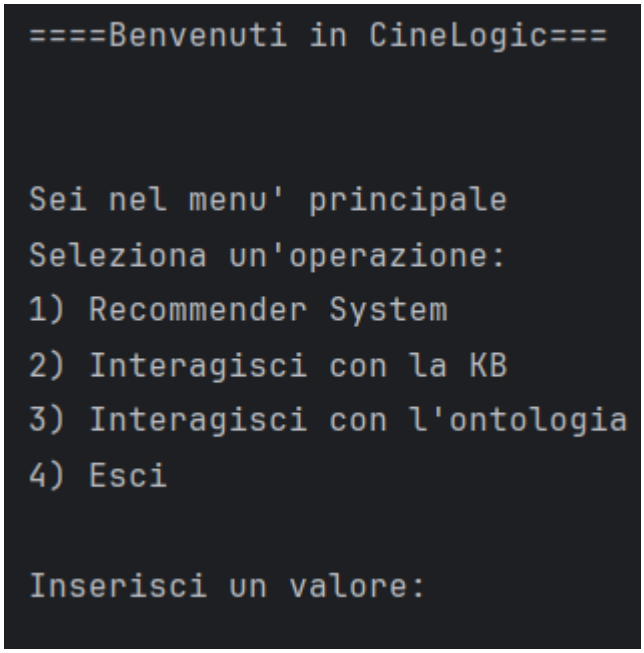
L'abbondanza di contenuti disponibili, unita alla crescente competizione tra le diverse piattaforme, rende tuttavia complesso per gli utenti orientarsi nella scelta del titolo da visualizzare o della piattaforma più adatta alle proprie esigenze. In questo contesto emerge la necessità di strumenti intelligenti in grado di analizzare le preferenze degli utenti e supportarli nelle loro decisioni.

Considerato l'ampio utilizzo di tali servizi e in quanto noi stessi fruitori delle piattaforme di streaming, abbiamo deciso di sviluppare un sistema che potesse fornire un supporto concreto nella selezione di contenuti audiovisivi e nell'individuazione della piattaforma più conveniente o coerente con i propri gusti.

A tal fine, il progetto si articola nei seguenti componenti:

- Un Recommender System, progettato per suggerire film e serie TV sulla base delle preferenze personali dell'utente. Data l'elevata quantità di contenuti disponibili, la scelta può risultare difficoltosa; il sistema implementato utilizza algoritmi di analisi dei dati per individuare pattern nelle preferenze e proporre contenuti coerenti con i gusti individuali.
- Un modello di classificazione, in grado di prevedere la categoria dei rating assegnati dagli utenti ai contenuti. Questo modulo consente di comprendere meglio le dinamiche di valutazione dei film e delle serie TV e di migliorare ulteriormente la qualità dei suggerimenti forniti dal sistema.
- Una Knowledge Base interattiva, che permette agli utenti di interrogare il sistema per ottenere informazioni dettagliate su film, serie TV e relative caratteristiche.
- Un'ontologia di dominio, finalizzata a rappresentare formalmente il dominio delle piattaforme di streaming, definendo concetti, relazioni e significati dei simboli utilizzati all'interno del sistema.

Una volta avviato il programma, l'utente può interagire dinamicamente con ciascuna componente, scegliendo l'opzione più adeguata alle proprie necessità e passando agevolmente dal recommender system alla knowledge base o all'ontologia, in un ambiente integrato e coerente.



1.1. Dataset utilizzati e librerie

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

Sono state applicate una serie di operazioni di *pre-processing*, descritte in seguito.

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

Nello sviluppo del progetto, si è fatto largo uso delle librerie **Pandas** e **NumPy.**

1.2. Pre-processing del dataset

Dopo aver osservato gli elementi che costituiscono il dataset, passiamo alla fase di *pre-processing* del dataset.

La prima operazione che abbiamo eseguito è la rimozione delle colone che non sono utili ai nostri scopi quali:

- **type:** in quanto il dataset fa riferimento esclusivamente a film, non è utile la distinzione per tipo
- **Season:** dato che gli elementi del dataset sono univocamente film
- **imdb_id:** poiché 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.
- **ncost:** identificativo che non è utile al nostro scopo
- **primaryName:** la ricerca del titolo in base ad un attore che vi è presente debba avvenire tra gli elementi dell'insieme 'name' e non basarsi sull'attore protagonista

```
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	streamin
0	0	tm100001	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	

Dall'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)
```

Rinominare 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).as
         df['monthly_subscription_cost'] = df['monthly_subscription_cost'] / 100
         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()
```

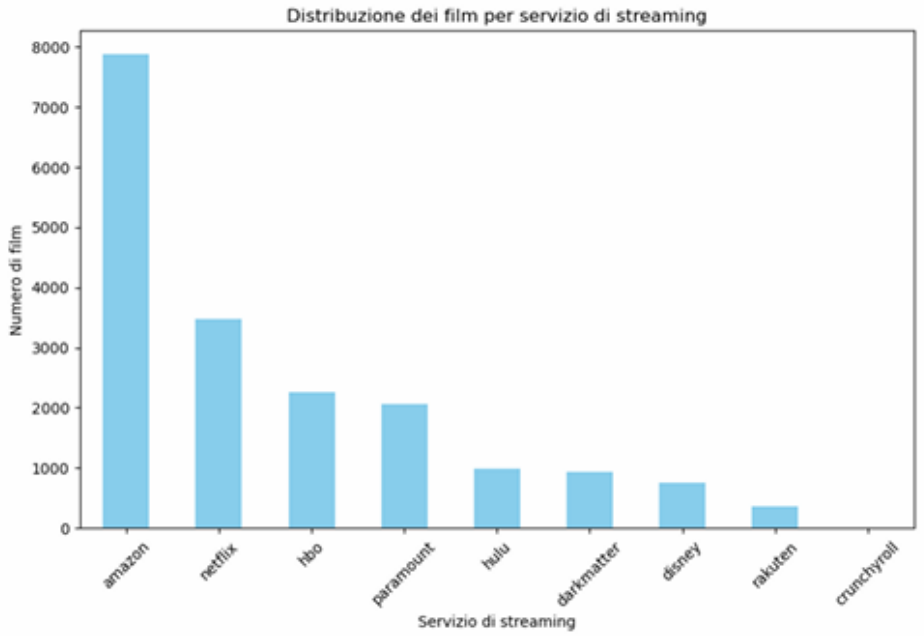
```
Out[14]: Unnamed: 0                0
         id                        0
         title                     0
         description               0
         release_year              0
         runtime                   0
         production_countries      0
         imdb_score                0
         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:

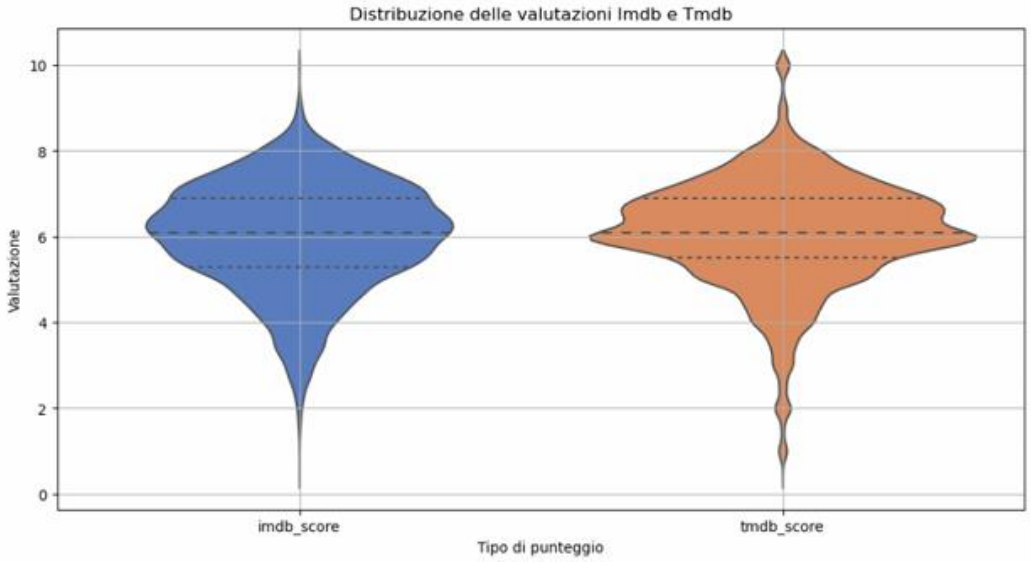
```
In [15]: df = pd.DataFrame(movies)

         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_n
         #creazione del grafico a violino
         plt.figure(figsize=(12, 6))
         sns.violinplot(x='score_type', y='score', data=df_long, inner='quartile', palette='mute
         plt.title('Distribuzione delle valutazioni Imdb e Tmdb')
         plt.xlabel('Tipo di punteggio')
         plt.ylabel('Valutazione')
         plt.grid(True)

         plt.show()
```



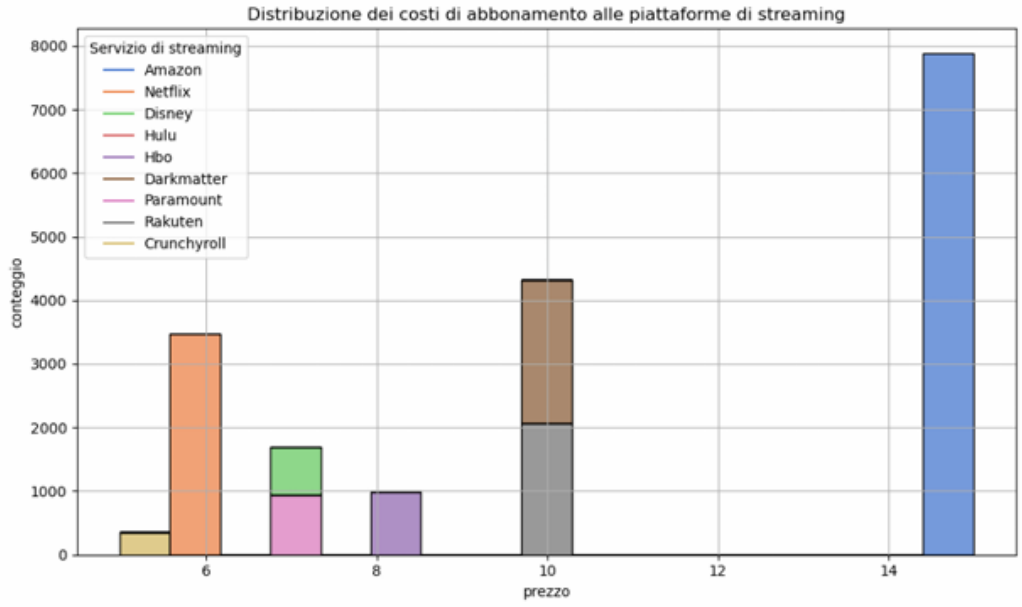
Dal grafico abbiamo rilevato una distribuzione delle votazioni per ciascun livello di valutazione. Ciascun violino rappresenta la variazione delle valutazioni assegnate dalle due piattaforme (imdb e tmdb) rispetto a ciascun titolo.

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:

    #recuperariamo 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

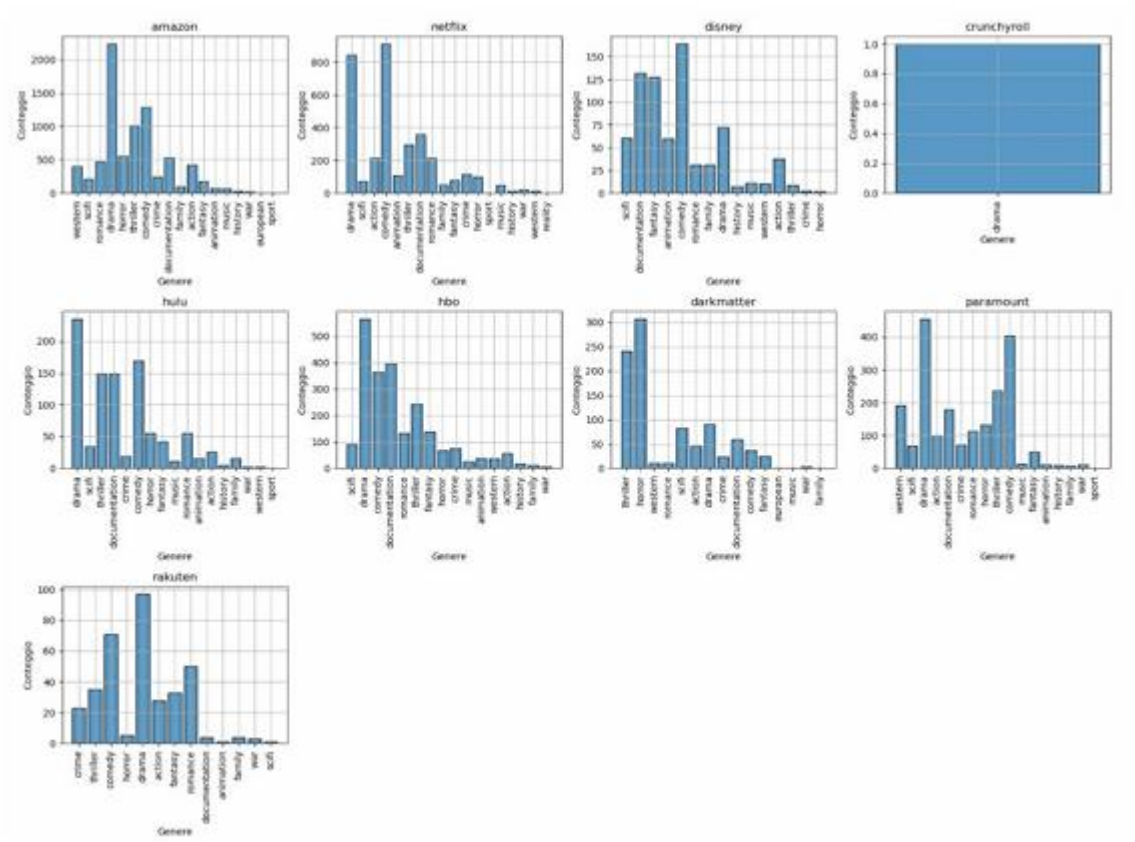
Andiamo a sfruttare la seguente funzione per visualizzare il numero dei film messi a disposizione da ciascuna piattaforma di streaming presa in considerazione all'interno del dataset, con lo scopo di analizzare la correlazione tra il numero di film presente in ciascuna piattaforma e i generi maggiormente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



Infine, restituiamo 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  documentation    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 film che ha scelto o che preferisce, e riceve una raccomandazione personalizzata basata su tali informazioni.

--- Immagine ----

2.2. Scelta della metrica

Prima di procedere con lo sviluppo del recommender system, ci siamo dedicati allo studio delle metriche per il calcolo delle similarità, optando per la "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 Reccomender 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 al 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   ramensonoio
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 delle similarità, con l'utilizzo della correlazione di Pearson.

In questo modo, si creerà 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.

```
#vettorizzazione
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, abbiamo usato 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:

```
Ora possiamo procede alla fase di recommandation...

Ecco una lista di 5 film più simili a quello indicato, con una predizione sulla categoria star:
                        title  release_year          genre streaming_service  \
3893   Silukkuvarupatti Singam          2018         action            amazon
9459         Bareilly Ki Barfi          2017        romance           netflix
532         Murder in the Cove          2020  documentation            amazon
13077                     Mass          2021          drama              hulu
13905         The Great Gatsby          2013          drama               hbo

       star  star_prediction
3893    3.0             3.0
9459    3.0             3.0
532     3.0             3.0
13077   3.0             3.0
13905   3.0             3.0
```

2.4. Classificazione

Oltre al Racommender 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.

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

#assegnamo 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”.

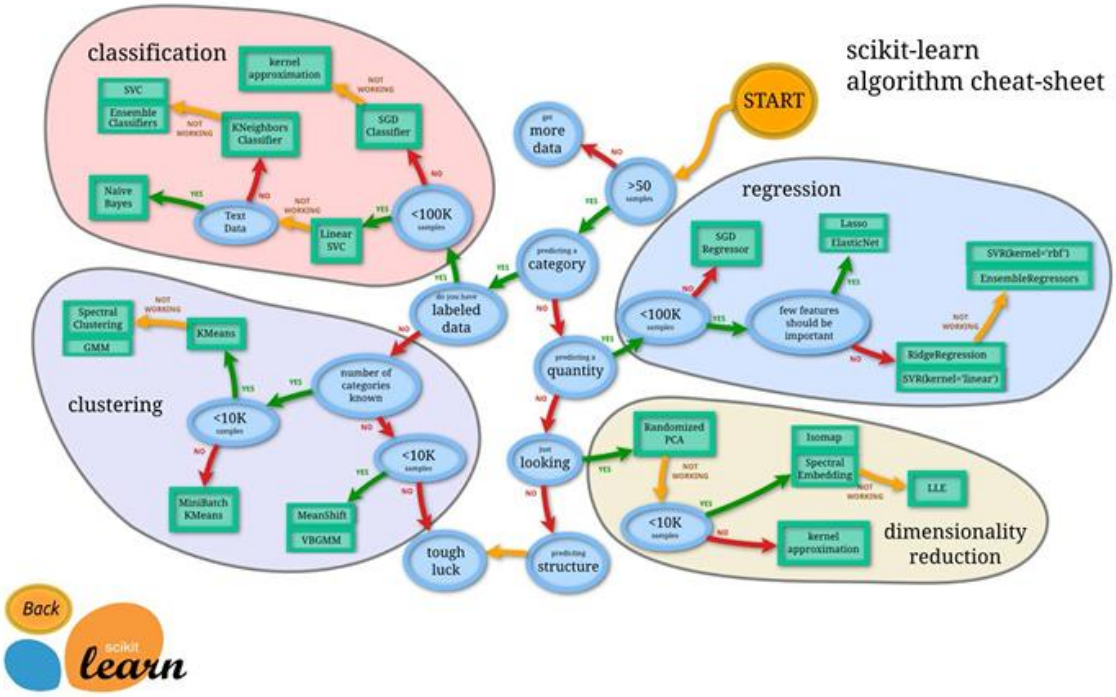
Il valore della categoria “star” si basa sulla media aritmetica dei valori di questi due campi. A seconda del valore della media ottenuto abbiamo stabilito 5 range all’interno dei quali far rientrare i dati già presenti nel dataset.

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 KNNNeighborsClassifier, disponibile nella libreria SKLearn.

2.5. Modello Adottato

Il K-Nearest Neighbors, o KNN, è un algoritmo impiegato nei 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.

2.6. Dataset

Per ottenere un dataset adatto all'uso del modello selezionato e per la classificazione, esso è stato diviso in due parti: training 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)
```

2.7. 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)
```

2.8. Miglioramento del modello e Hyperparameters Tuning

Dopo aver sperimentato il modello costruito con i parametri predefiiti, 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
```

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.

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

hyperparameters = dict(metric=metric, weights=weights, n_neighbors=n_neighbors)
```

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_eighbours:  25
```

```
ROC Score:  0.9998801625677819
```

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

2.9. 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 procede alla fase di recommandation...

Ecco una lista di 5 film più simili a quello indicato, con una predizione sulla categoria star:
                        title  release_year          genre streaming_service  \
3893   Silukkuvarupatti Singam          2018         action            amazon
9459         Bareilly Ki Barfi          2017        romance           netflix
532         Murder in the Cove          2020  documentation            amazon
13077                     Mass          2021          drama              hulu
13905         The Great Gatsby          2013          drama               hbo
```

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

3. Knowledge Base

3.1. Introduzione

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

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

Abbiamo utilizzato la direttiva :- **discontiguous** 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
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,
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:

```
Inserisci un valore: 2
Caricamento della KB... Attendi...

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
```

3.4.1. Trova film in base a 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.

Si presentano alcuni esempi presenti nel file "KnowledgeBase.py":

```
# Anno di uscita
"film_recente(ID) :- release_year(ID, Uscita), Uscita > 2010.\n",
"film_tra_2000_2010(ID) :- release_year(ID, Uscita), 2000 =< Uscita, Uscita =< 2010.\n",
"film_pre_2000(ID) :- release_year(ID, Uscita), Uscita < 2000.\n",
```

Queste regole segmentano i film in base all'anno di uscita, distinguendo tra produzioni **pre-2000**, produzioni del decennio **2000-2010** e film considerati **recenti** (post-2010).

```
# Durata del film
"film_breve_durata(ID) :- runtime(ID, Durata), Durata =< 60.\n",
"film_media_durata(ID) :- runtime(ID, Durata), 60 < Durata, Durata =< 90.\n",
"film_lunga_durata(ID) :- runtime(ID, Durata), Durata > 90.\n",
```

La regola traduce il valore numerico del runtime in categorie discrete: un film è considerato di **breve durata** sotto i 60 minuti, **media** tra 60 e 90, e di **lunga durata** oltre i 90 minuti.

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
```

```
Inserisci il periodo relativo all'anno di uscita del film
1) recente
2) tra 2000 e 2010
3) pre 2000
```

```
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
```

```
Inserisci la durata del film
1) breve
2) media
3) lunga
```

3.4.2. 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.

È 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:

```python
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.")
```

```python
# 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.")
```

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

```python
# Prezzo di sottoscrizione
"prezzo_economy(ID) :- monthly_subscription_cost(ID, Cost), Cost < 9.99.\n",
"prezzo_medio(ID) :- monthly_subscription_cost(ID, Cost), Cost = 9.99.\n",
"prezzo_costoso(ID) :- monthly_subscription_cost(ID, Cost), Cost > 9.99.\n",
```

La regola traduce il valore numerico del runtime in categorie discrete: un film è considerato di **breve durata** sotto i 60 minuti, **media** tra 60 e 90, e di **lunga durata** oltre i 90 minuti.

Di seguito un esempio di esecuzione:

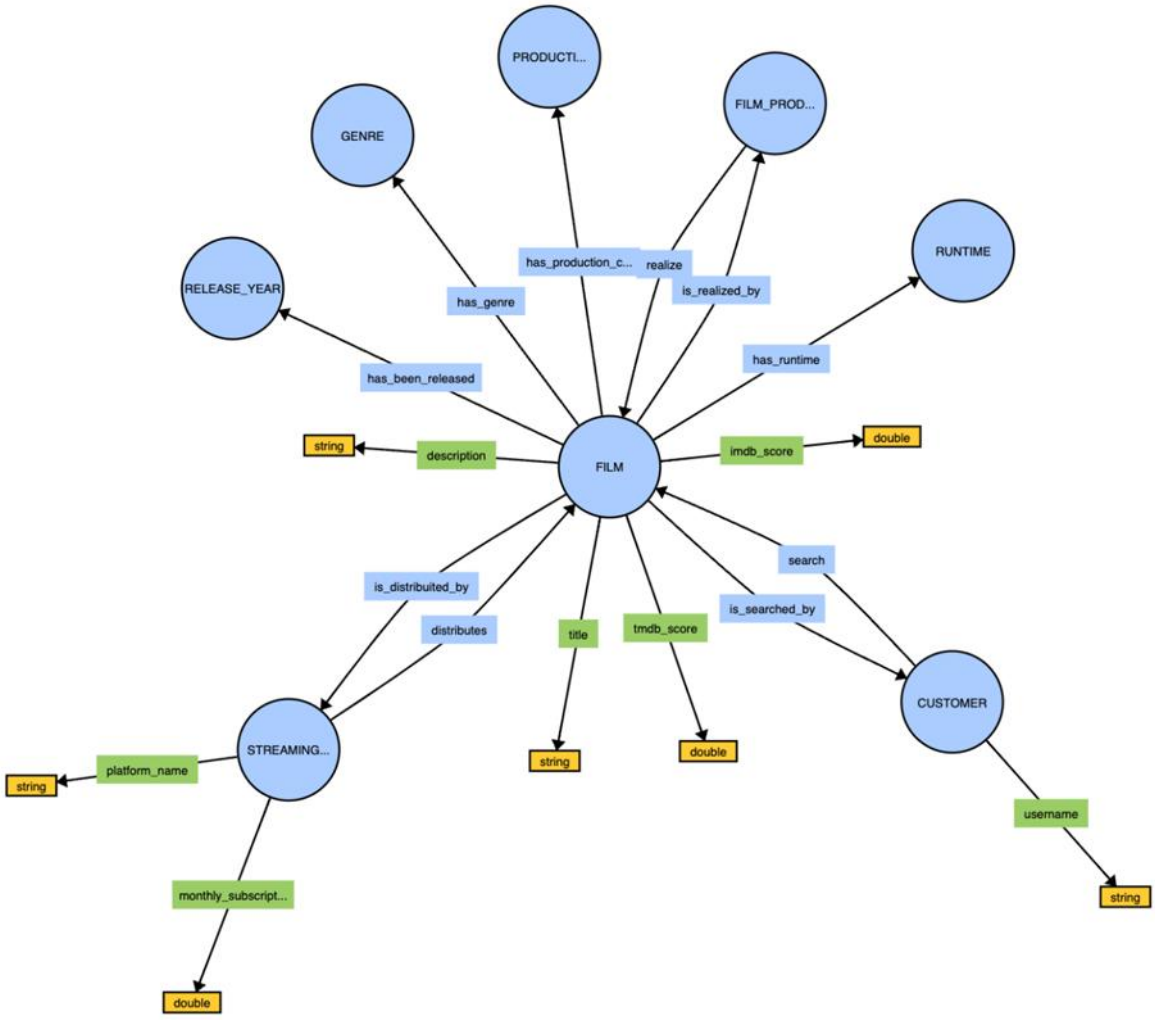
```
Scegliere quale funzione eseguire:
1) Trova un film date determinate caratteristiche
2) Sulla base delle preferenze dell'utente, trovare la piattaforma migliore per i film selezionati.
3) Esci

Inserisci un valore: 2
Seleziona il tipo di sottoscrizione:
1) economica
2) media
3) costosa
1
La migliore piattaforma consigliata è: darkmatter
Numero di film disponibili su darkmatter: 32
```

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).

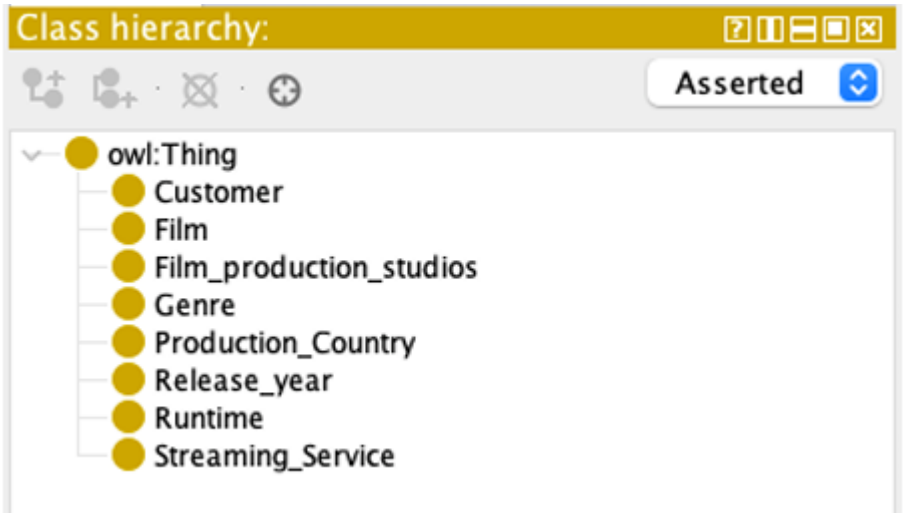


Figura 4-1: Classi dell'Ontologia

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 primitvo (data properties).

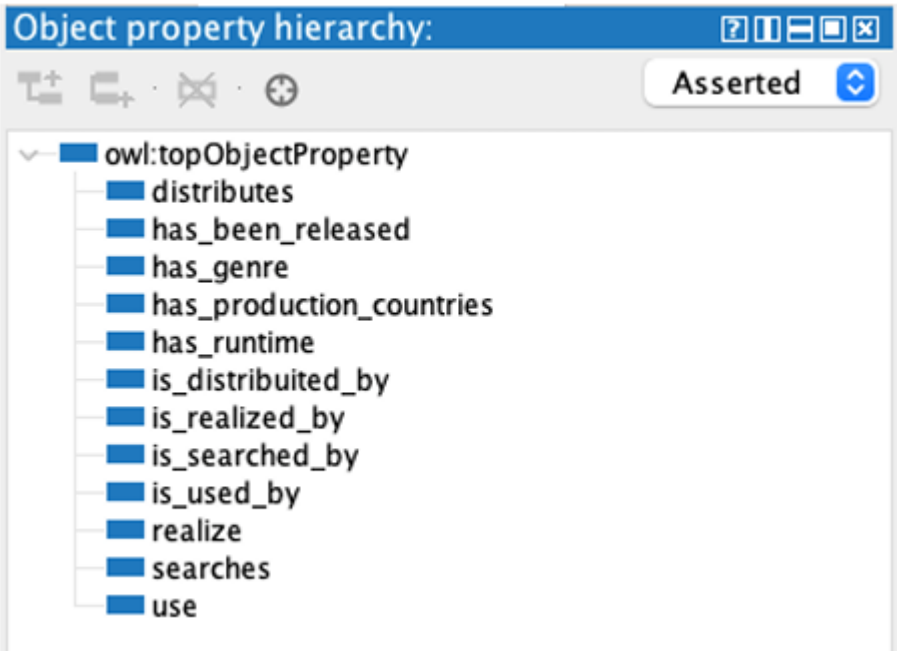


Figura 4-2: Object-Properties

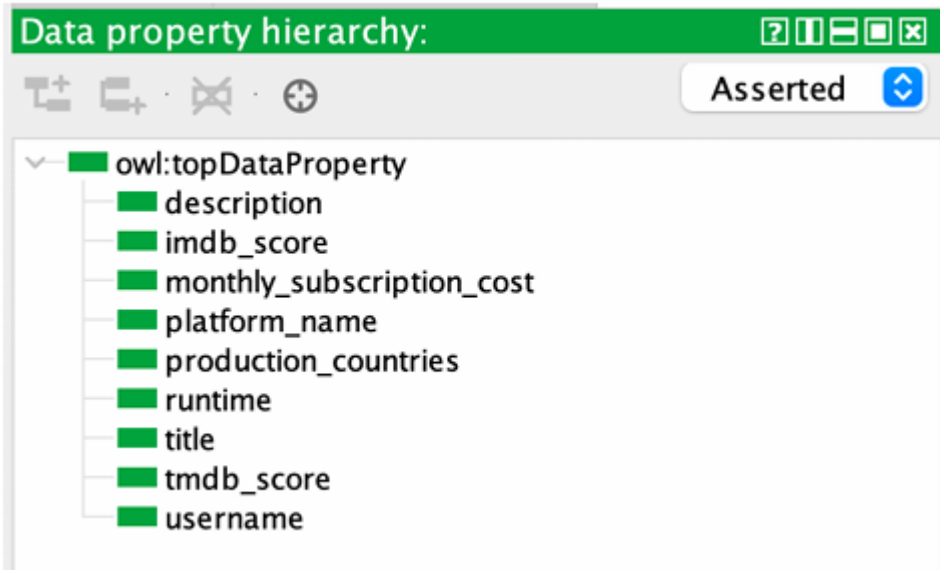


Figura 4-3: Data-Properties

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

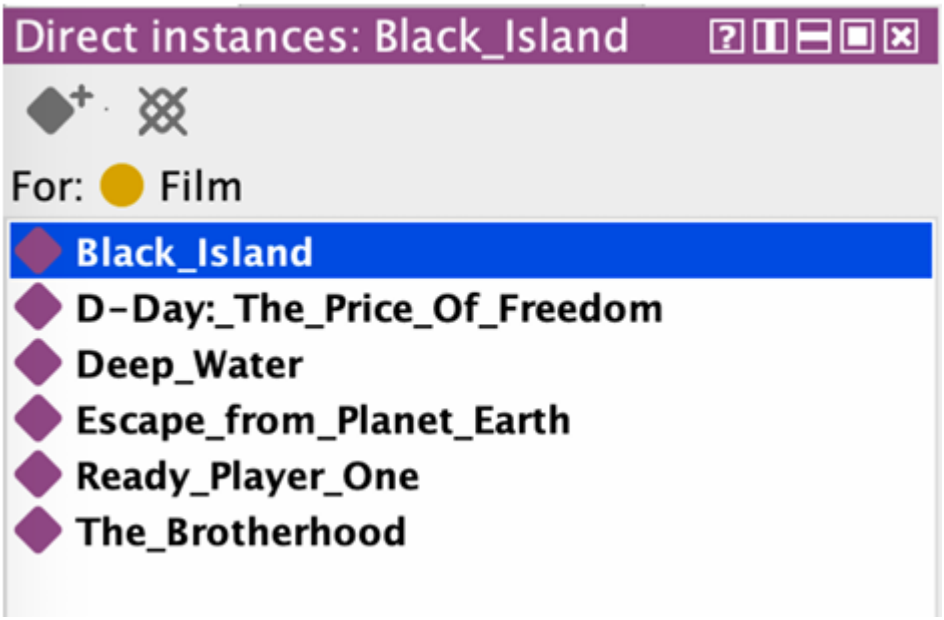


Figura 4-4: Individui della classe Film

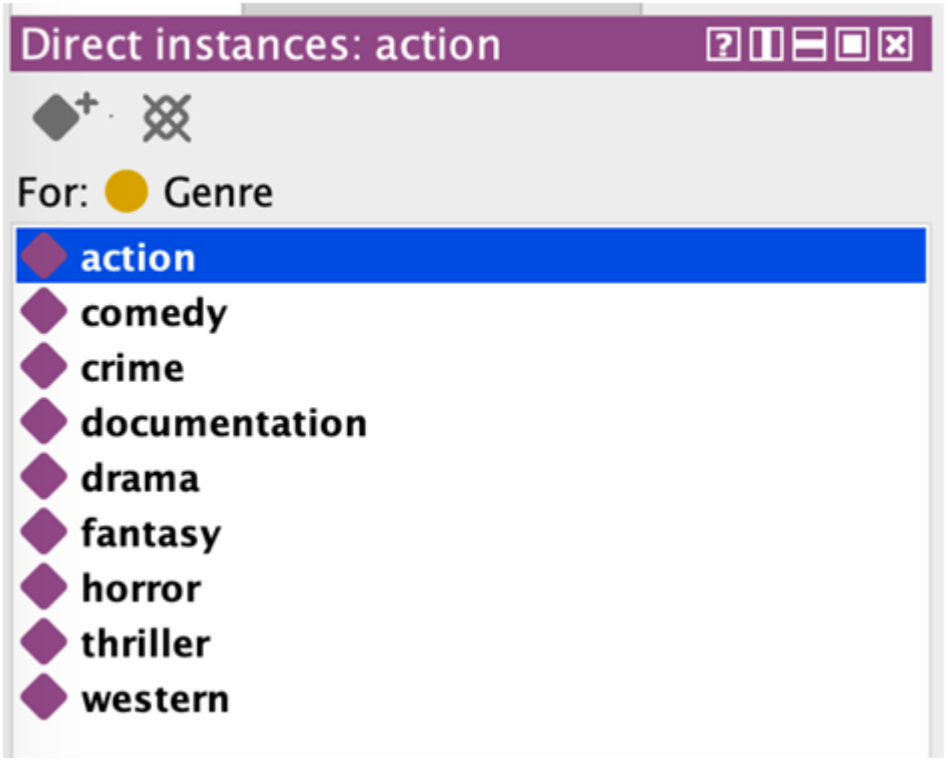


Figura 4-5: Individui della classe Genre

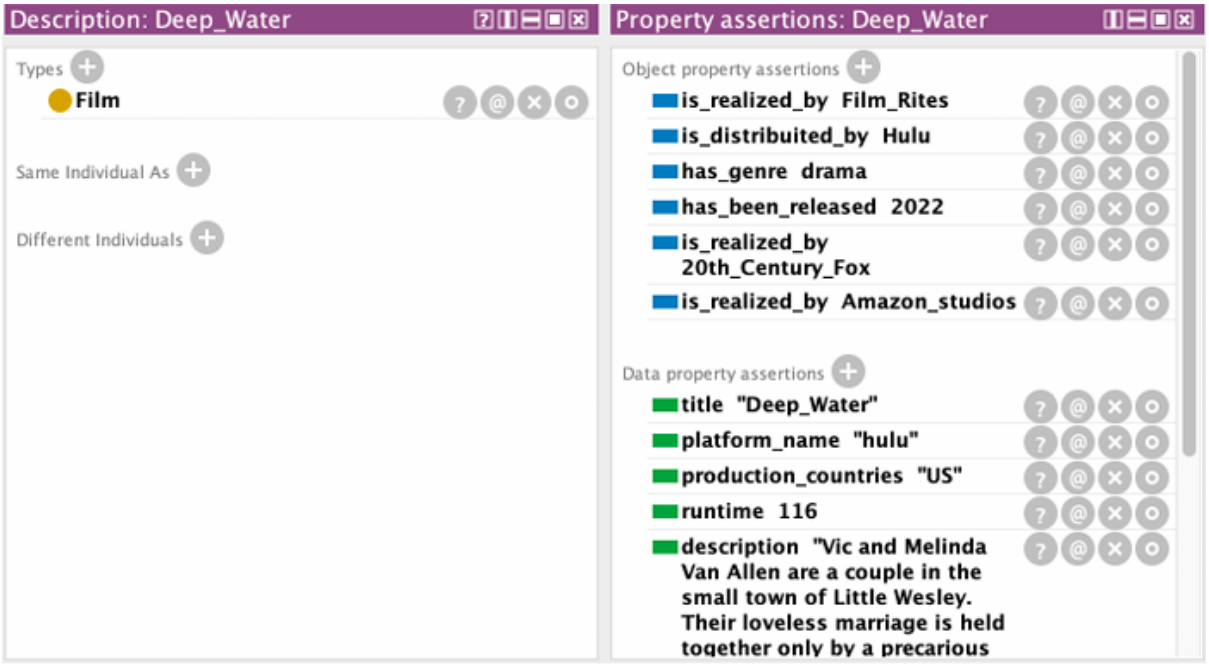


Figura 4-6: Esempio di individuo con properties annesse

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

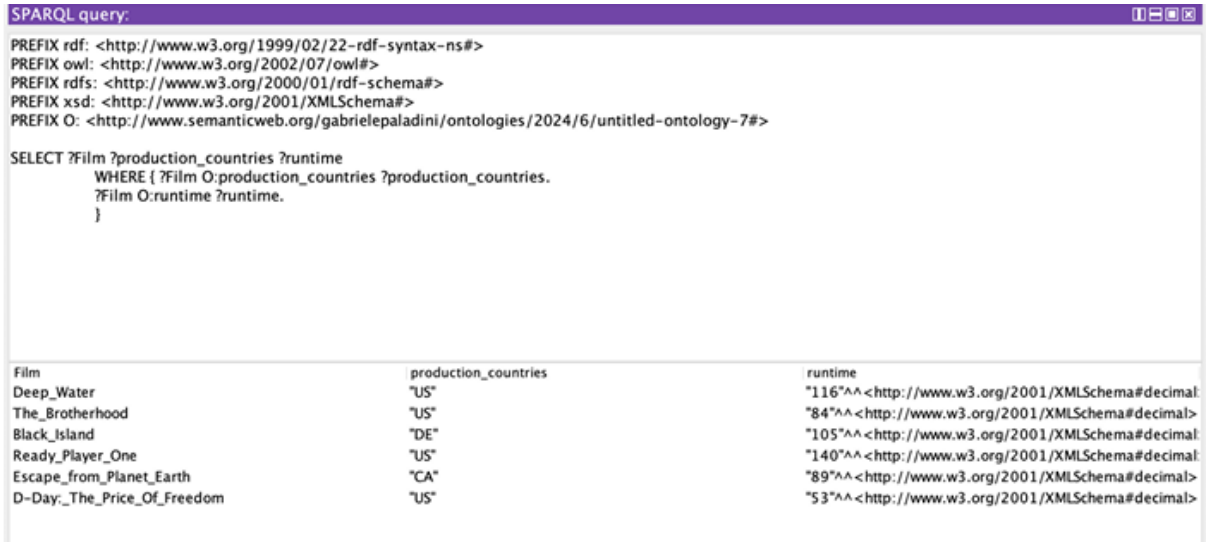


Figura 4-7: SPARQL query con visualizzazione dei Film, Production_countries e runtime

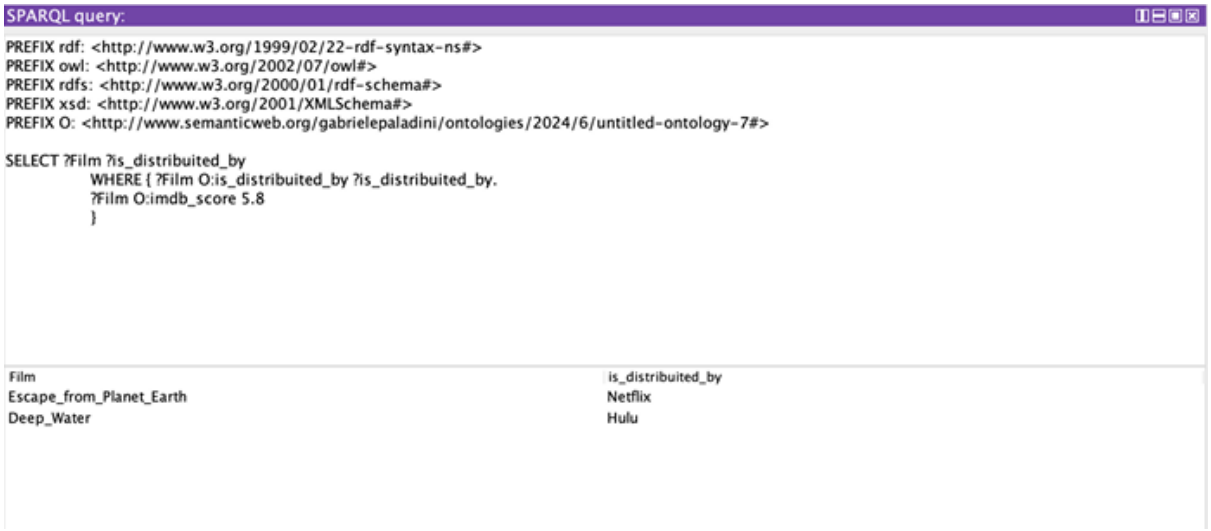


Figura 4-8: SPARQL query con visualizzazione dei Film che hanno come Object-properties "is_distribuited_by" e un "imdb_score" pari a 5.8

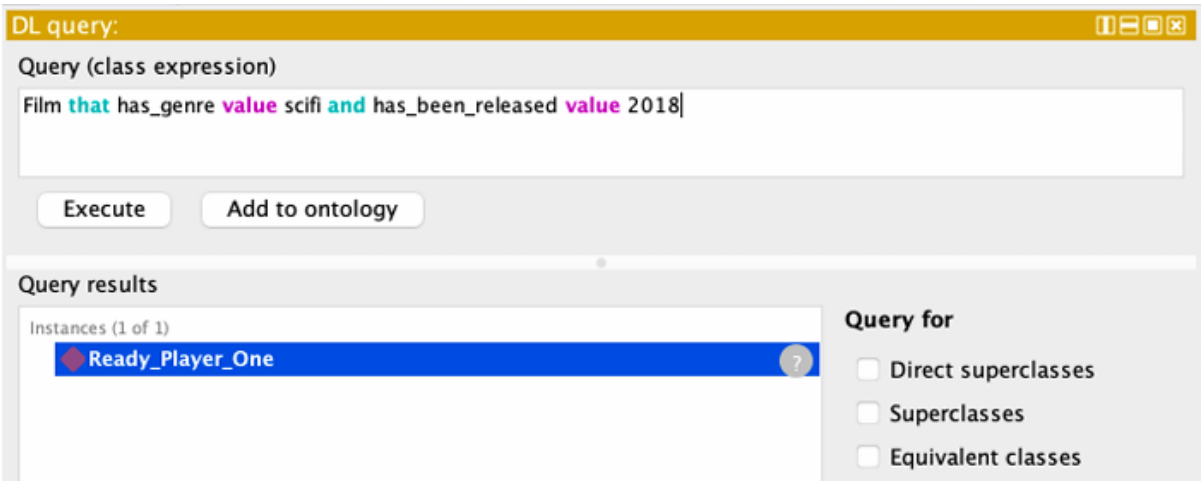


Figura 4-9: DL-query con ricerca di film che hanno come genere "scifi" e come anno di rilascio "2018"

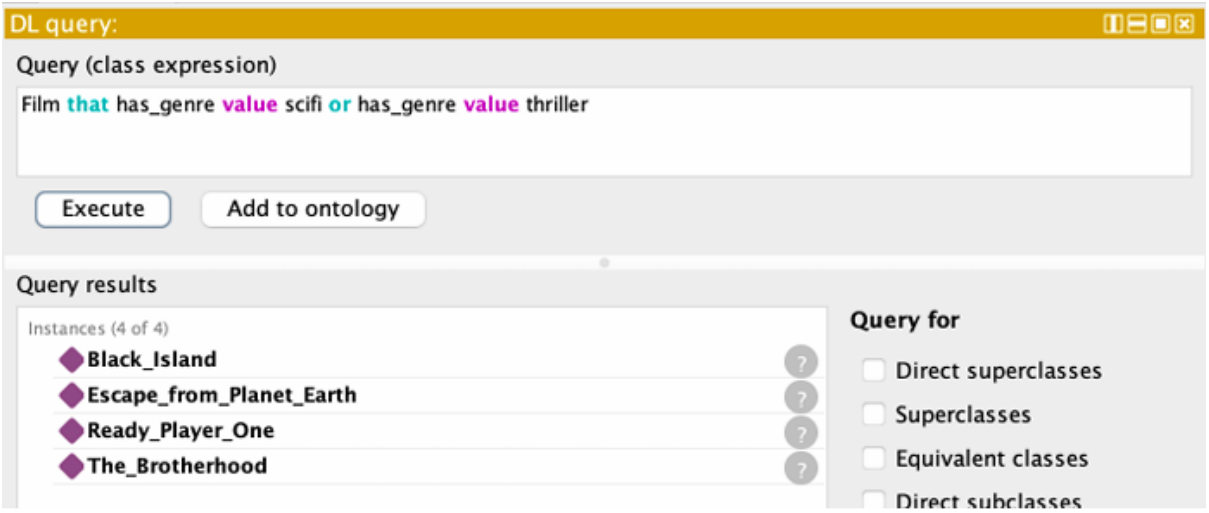


Figura 4-10:DL-query con ricerca di film che hanno come genere "scifi" oppure "thriller"

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, dov'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
```

Selezionando la prima opzione, si potranno visualizzare le classi presenti all'interno dell'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
```

Possiamo, inoltre, esaminare da vicino le classi selezionando quella di interessa tra quelle visualizzate a schermo.

1) Selezionando la prima classe, verrà mostrato a schermo la lista dei film presenti:

```
LISTA FILM PRESENTI:
- Ontologia.owx.Film
- Ontologia.owx.Black_Island
- Ontologia.owx.Deep_Water
- Ontologia.owx.Escape_from_Planet_Earth
- Ontologia.owx.Ready_Player_One
- Ontologia.owx.The_Brotherhood
- Ontologia.owx.D-Day:_The_Price_Of_Freedo
```

2) Selezionando la seconda classe, verrà mostrato a schermo un elenco degli anni di rilascio dei film:

```
LISTA ANNI DI RILASCIO PRESENTI:
- Ontologia.owx.Release_year
- Ontologia.owx.2005
- Ontologia.owx.2012
- Ontologia.owx.2016
- Ontologia.owx.2017
- Ontologia.owx.2018
- Ontologia.owx.2021
- Ontologia.owx.2022
- Ontologia.owx.2023
- Ontologia.owx.2024
```

3) Selezionando la terza classe, verrà mostrato a schermo un elenco contenente tutti i servizi di streaming presenti all'interno della nostra ontologia:

```
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+
```

4) Selezionando la quarta classe, verrà mostrato a schermo l'elenco dei generi presenti:

```
LISTA DEI GENERI PRESENTI:
- Ontologia.owx.Genre
- Ontologia.owx.action
- Ontologia.owx.comedy
- Ontologia.owx.crime
- Ontologia.owx.documentation
- Ontologia.owx.drama
- Ontologia.owx.fantasy
- Ontologia.owx.horror
- Ontologia.owx.scifi
- Ontologia.owx.thriller
- Ontologia.owx.western
```

5) Selezionando la quinta classe, verrà mostrato a schermo l'elenco dei clienti presenti:

```
LISTA DEI CLIENTI PRESENTI:
- Ontologia.owx.Customer
- Ontologia.owx.CinephileAlex
- Ontologia.owx.FlickFanFabio
- Ontologia.owx.MovieBuffMarco
- Ontologia.owx.SeriesFanAnna
```

6) Selezionando la sesta classe, verrà mostrato a schermo l'elenco degli studi di produzione presenti:

```
LISTA DEGLI STUDI DI PRODUZIONE PRESENTI:
- Ontologia.owx.Film_production_studios
- Ontologia.owx.Amazon_studios
- Ontologia.owx.Amblin_Entertainement
- Ontologia.owx.DeLine_Pictures
- Ontologia.owx.Film_Rites
- Ontologia.owx.Netflix_Studios
- Ontologia.owx.RainMaker_Entertainement
- Ontologia.owx.20th_Century_Fox
```

7) Verrà richiesto alla fine dell'analisi di ciascuna delle classi proposte se l'utente desidera esamina un'altra classe oppure no:

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

Selezionando la seconda opzione, si avrà la possibilità di visualizzare le 'proprietà d'oggetto' contenute all'interno dell'ontologia:

```
PROPRIETÁ 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_distribuited_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
```

Selezionando la terza opzione, si avrà la possibilità di esaminare le 'proprietà dei dati' presenti nell'ontologia:

```
PROPRIETÁ 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
```

Selezionando la quarta operazione, si potranno selezionare ed eseguire alcune query di esempio:

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

Figura 4-11: Esecuzione della Prima 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
```

Figura 4-12: Esecuzione della Seconda 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
```

Figura 4-13: Esecuzione della Terza Query

5. Conclusioni

Il Progetto CineLogic rappresenta una solida base per la creazione di un sistema di raccomandazioe 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, tale progetto, potrebbe diventare una soluzione ancora più completa, capace di rispondere alle esigenze di un pubblico diversificato e di adattarsi alle nuove tendenze del mercato.