Caso di studio del corso
Ingegneria della conoscenza

Games Recommender

Individuare il miglior videogame per il tuo intrattenimento

***Autore*:**

Leonardo Moro

Matricola: 758405

E-mail: l.moro4@studenti.uniba.it

***AA*:** 2025-2026

https://github.com/leomoro1/Games_Recommender.git

Sommario

1. Introduzione

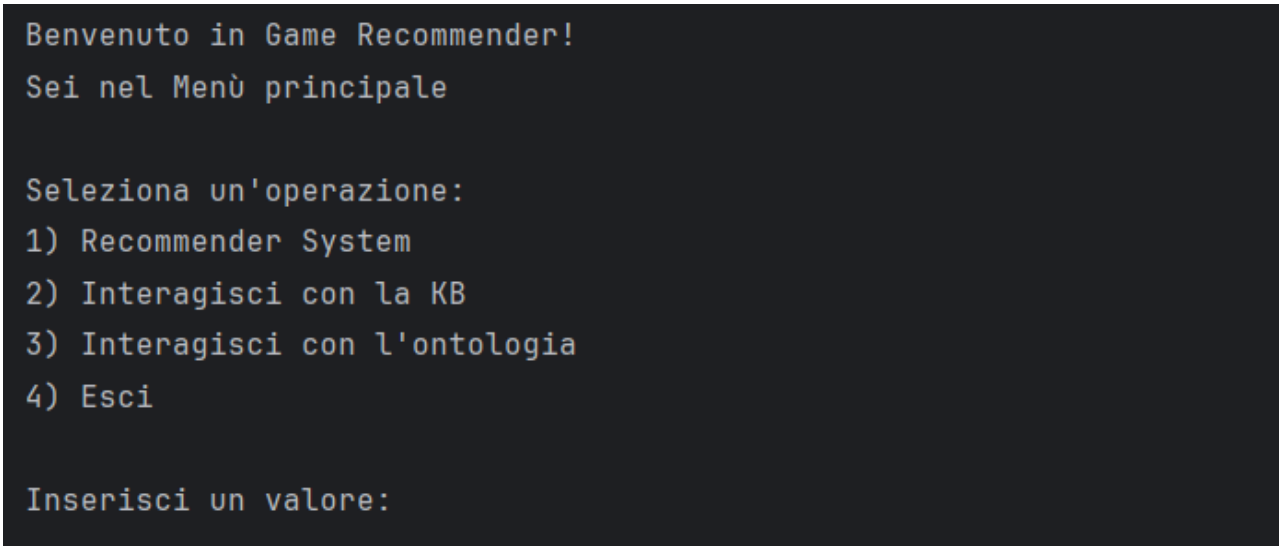
Il mondo dei videogiochi è oggi una parte integrante della vita quotidiana di milioni di persone. Con la continua espansione del mercato videoludico e l'enorme quantità di titoli disponibili su piattaforme come Steam, Epic Games Store, e molti altri, risulta sempre più complesso per i giocatori orientarsi nella scelta del gioco più adatto alle proprie preferenze.

Data la crescente popolarità del medium videoludico e in quanto appassionati di videogiochi, abbiamo deciso di indirizzare il nostro progetto verso la realizzazione di un sistema che possa supportare gli utenti nella scelta del videogioco da acquistare o provare.

A tale scopo è stato realizzato:

- **un recommender system** progettato per consigliare nuovi videogiochi in base alle preferenze personali degli utenti. Con un'offerta così vasta e variegata, scegliere cosa giocare può diventare difficile. Il nostro sistema sfrutta algoritmi avanzati per analizzare le abitudini di gioco e proporre titoli in linea con i gusti individuali del giocatore;
- **un modello di classificazione** in grado di prevedere le valutazioni assegnate dagli utenti ai vari videogiochi. Questo ci consente di comprendere meglio l'opinione generale sui titoli analizzati e migliorare ulteriormente i suggerimenti forniti dal sistema di raccomandazione;
- **una knowledge base interattiva**, per permettere agli utenti di accedere a informazioni dettagliate sui videogiochi presenti nel sistema, come genere, anno di rilascio, numero di valutazioni, achievements, prezzo e altro ancora;
- **un'ontologia del dominio** che rappresenta formalmente i concetti principali del contesto videoludico (come Videogioco, Genere, Prezzo, Data di Rilascio, Valutazione), specificandone le relazioni attraverso proprietà d'oggetto e di dati, rendendo così il sistema più interpretabile e interrogabile anche a livello semantico.

Una volta avviato il programma, l'utente potrà interagire liberamente con ciascun modulo, scegliendo l'opzione che meglio soddisfa le sue necessità del momento. Il passaggio tra il recommender system, la knowledge base e l'ontologia è fluido e intuitivo, rendendo l'esperienza completa, personalizzata e dinamica.



1.1 Dataset utilizzati e librerie

Per realizzare questo sistema è stato utilizzato il seguente dataset:
https://www.kaggle.com/datasets/nikdavis/steam-store-games?select=steam.csv
A cui è stato 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.
È stato 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 è stata eseguita è la rimozione delle colonne che non sono utili ai nostri scopi quali:

```
columns_to_drop = [
    'appid', 'english', 'developer', 'publisher',
    'categories', 'platforms', 'required_age',
    'median_playtime', 'steamspy_tags'
]
steam_df = steam_df.drop(columns=columns_to_drop, errors='ignore')
steam_df.head()
```

✓ [51] 99ms

5 rows × 5 rows × 9 cols

	name	release_date	genres	achievements	positive_ratings	ne
0	Counter-Strike	2000-11-01	Action	0	124534	
1	Team Fortress Classic	1999-04-01	Action	0	3318	
2	Day of Defeat	2003-05-01	Action	0	3416	
3	Deathmatch Classic	2001-06-01	Action	0	1273	
4	Half-Life: Opposing Force	1999-11-01	Action	0	5250	

Sono state eliminate le seguenti colonne:

appid: identificativo univoco del gioco su Steam, non utile nel contesto del nostro sistema di raccomandazione o nella rappresentazione semantica tramite ontologia;

english: tutti i giochi del dataset risultano essere in lingua inglese o presentano una codifica non coerente di questo campo; non aggiunge valore informativo;

developer e **publisher**: nel nostro progetto non si è ritenuto necessario distinguere lo sviluppatore dal publisher. Inoltre, molti giochi presentano valori nulli o duplicati tra i due attributi;

categories, **platforms**, **required_age**: queste informazioni sono generiche o ridondanti rispetto ad altri attributi più significativi (es. generi, valutazioni, playtime). Alcune colonne presentano dati non strutturati, difficili da trattare senza una pulizia approfondita;

median_playtime: si è preferito conservare un solo campo relativo al tempo medio di gioco, escludendo valori mediana o minimi, che spesso risultano incoerenti o duplicati;

steamspy_tags: sebbene potenzialmente utile, questo campo contiene valori testuali concatenati da separatori non uniformi, che rendono complessa l'elaborazione automatica senza uno specifico parsing avanzato.

Dall'analisi del dataset si può osservare che alcune metriche numeriche, come “price” “average_playtime”, presentano valori che non risultano approssimati in maniera coerente.
In particolare, il campo “price” include valori con numerose cifre decimali, mentre “average_playtime” mostra numeri con valori molto elevati che non sono stati arrotondati all'unità.

Per garantire una maggiore uniformità e leggibilità del dataset ho proceduto ad approssimare:

- la colonna “price” a due cifre decimali

- la colonna “average_playtime” a numeri interi

```
df = pd.read_csv('steam_standard_dataset.csv')
df['price'] = df['price'].round(2)
df['average_playtime'] = df['average_playtime'].round(0)
print(df[['price', 'average_playtime']].head(10).to_string(index=False))

df.to_csv('steam_rounded_dataset.csv', index=False)
[19]
```

```
 price  average_playtime
  7.19             17612
  3.99               277
  3.99               187
  3.99               258
  3.99               624
  3.99               175
  7.19              1300
  7.19               427
  3.99               361
  7.19               691
```

Procediamo a rinominare le colonne: “name”, “genres”, “price” e “average_playtime” con i nuovi titoli di: “title”, “genre”, “game_price” e “avg_playtime”.

```
df = pd.read_csv('steam_rounded_dataset.csv')
df.rename(columns={
    'name': 'title',
    'genres': 'genre',
    'price': 'game_price',
    'average_playtime': 'avg_playtime'
}, inplace=True)
df.to_csv('steam_renamed_dataset.csv', index=False)
```

1.3 Gestione delle celle ‘null’ per ogni colonna

```
steam_df.isnull().sum()
```

[21]

9 rows Length: 9, dtype: int64

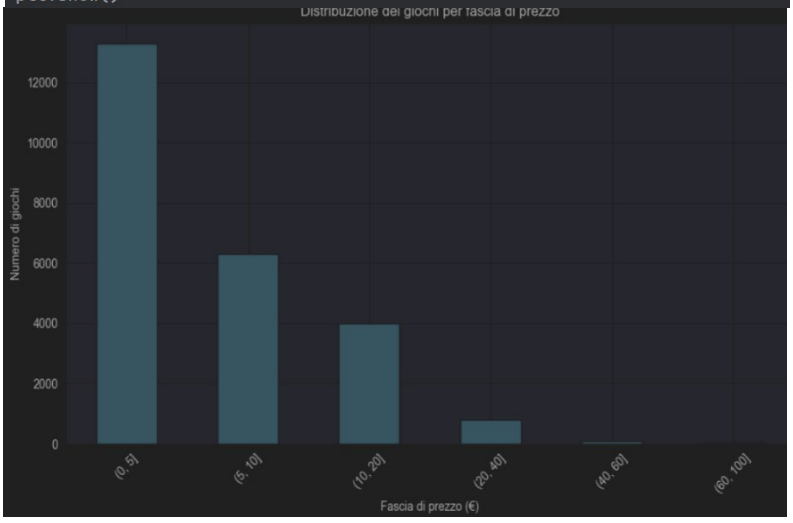
	<unnamed>
name	0
release_date	0
genres	0
achievements	0
positive_ratings	0
negative_ratings	0
average_playtime	0
owners	0
price	0

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

1.4 Rappresentazioni grafiche

Istogramma che rappresenta la distribuzione dei giochi per fascia di prezzo presenti nel dataset:

```
df = pd.read_csv("steam_renamed_dataset.csv")

# Fasce di prezzo
price_bins = pd.cut(df['game_price'], bins=[0, 5, 10, 20, 40, 60, 100])
price_counts = price_bins.value_counts().sort_index()

# Istogramma
plt.figure(figsize=(10, 6))
price_counts.plot(kind='bar', color='skyblue')
plt.title('Distribuzione dei giochi per fascia di prezzo')
plt.xlabel('Fascia di prezzo (€)')
plt.ylabel('Numero di giochi')
plt.xticks(rotation=45)
plt.tight_layout()
plt.grid(True)
plt.show()
```

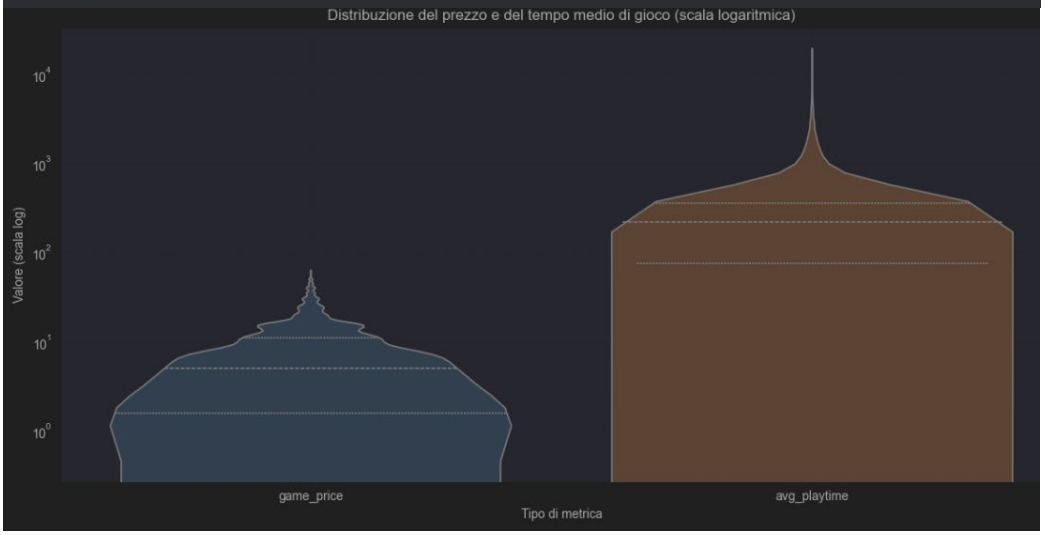


Dall'istogramma generato possiamo osservare la distribuzione dei videogiochi tra diverse fasce di prezzo.

Si nota una netta concentrazione di titoli nella fascia 0–5€, suggerendo che la maggior parte dei giochi su Steam siano economici o gratuiti.
Le fasce più alte (oltre i 40€) contengono un numero molto ridotto di titoli, il che indica che le raccomandazioni di prezzo più elevato saranno statisticamente meno frequenti.

Grafico a violino che mostra la distribuzione dei valori prezzo e tempo medio di gioco

```
df = pd.read_csv("steam_renamed_dataset.csv")

# Filtro outlier e dati non validi
df_filtered = df[(df['avg_playtime'] > 0) & (df['avg_playtime'] < 20000) & (df['game_price'] < 100)]

# Melt per struttura long
df_long = pd.melt(df_filtered, id_vars=['title'], value_vars=['game_price', 'avg_playtime'],
                  var_name='feature', value_name='value')

plt.figure(figsize=(12, 6))
sns.violinplot(x='feature', y='value', hue='feature', data=df_long, inner='quartile', palette='pastel',
 legend=False)


plt.yscale('log')
plt.title('Distribuzione del prezzo e del tempo medio di gioco (scala logaritmica)')
plt.xlabel('Tipo di metrica')
plt.ylabel('Valore (scala log)')
plt.grid(True, linestyle='--', alpha=0.5)
plt.tight_layout()
plt.show()
```

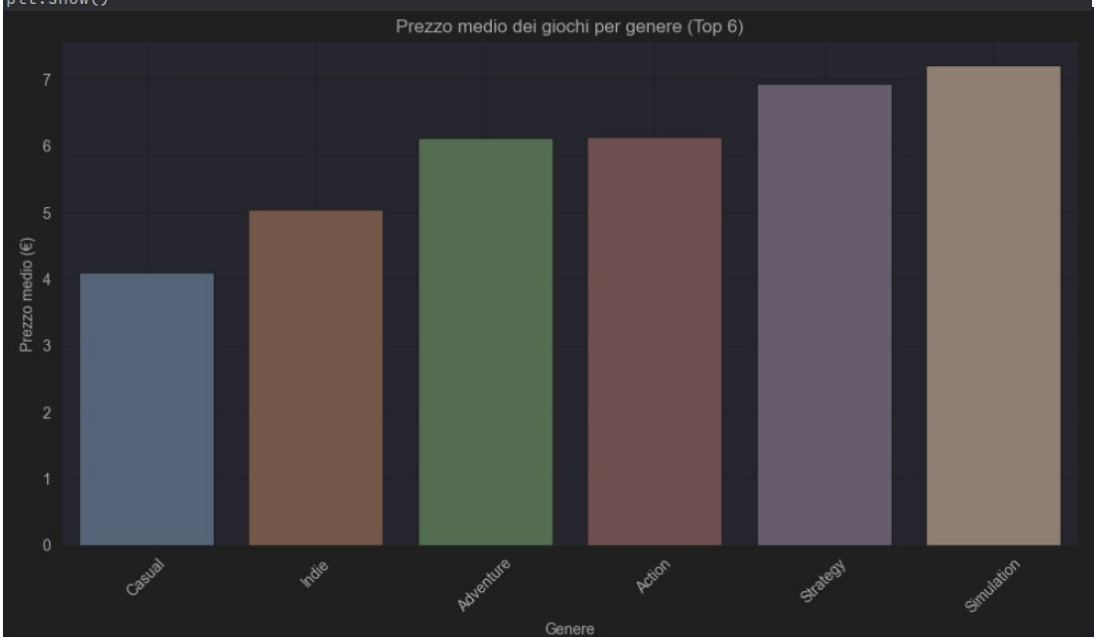


In questo grafico a violino è rappresentata la distribuzione del prezzo “game_price” e del tempo medio di gioco “avg_playtime” dei videogiochi presenti nel dataset.
Per migliorare la leggibilità, l’asse delle ordinate è rappresentato in scala logaritmica, vista la forte disparità tra le due metriche.

Come si può osservare, la maggior parte dei giochi ha un prezzo che si aggira attorno a valori contenuti, mentre il tempo medio di gioco varia notevolmente, con un picco visibile anche oltre le 10.000 unità.
La forma dei violini permette inoltre di cogliere la densità dei dati e l’intervallo interquartile, evidenziando come entrambi i valori presentino una forte asimmetria nella distribuzione.

Grafico che rappresenta la distribuzione del prezzo medio per genere

```
df = pd.read_csv("steam_renamed_dataset.csv")

# Separazione dei generi multipli
df['genre'] = df['genre'].astype(str)
df_genres = df.assign(genre=df['genre'].str.split(';')).explode('genre')

# Trova i 6 generi più frequenti
top_genres = df_genres['genre'].value_counts().nlargest(6).index

# Filtra e calcola il prezzo medio
df_top = df_genres[df_genres['genre'].isin(top_genres)]
avg_price = df_top.groupby('genre')['game_price'].mean().sort_values()

# Plot
plt.figure(figsize=(10, 6))
sns.barplot(x=avg_price.index, y=avg_price.values, hue=avg_price.index, palette='muted', legend=False)

# Etichette
plt.title('Prezzo medio dei giochi per genere (Top 6)')
plt.ylabel('Prezzo medio (€)')
plt.xlabel('Genere')
plt.xticks(rotation=45)
plt.grid(True)
plt.tight_layout()
plt.show()
```



Questo grafico mostra il **prezzo medio dei giochi** per i **sei generi più frequenti** nel dataset. Consente di confrontare visivamente quanto costano in media i giochi appartenenti a ciascun genere, evidenziando differenze significative tra generi come **Simulation** (più costosi) e **Casual** (più economici).

1.5 Individuazione di eventuali correlazioni

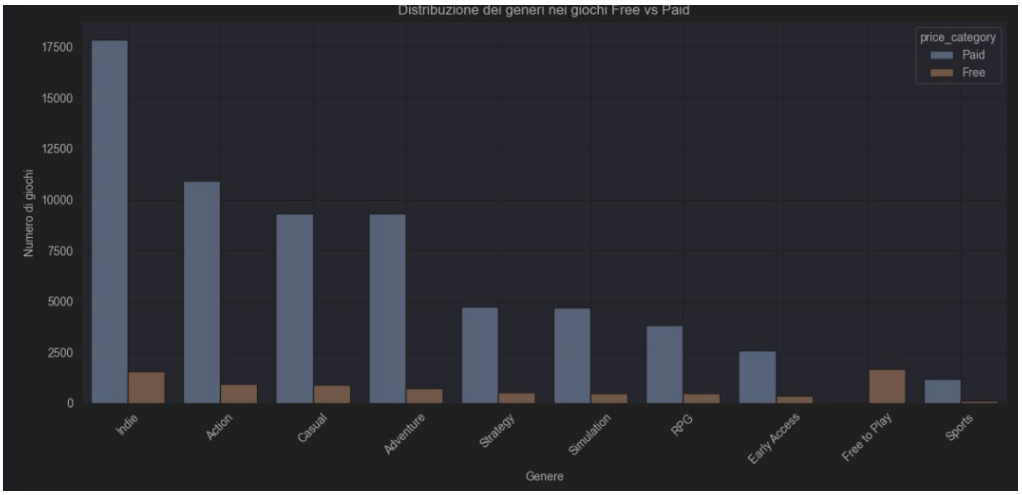
Andiamo a sfruttare la seguente procedura per visualizzare il numero di videogiochi appartenenti a ciascun genere all'interno del dataset. L'obiettivo è analizzare la distribuzione dei generi e comprendere quali tipologie di gioco siano maggiormente rappresentate, fornendo così un'indicazione utile sia per la progettazione del sistema di raccomandazione sia per lo studio delle preferenze degli utenti nel panorama videoludico.

```
df = pd.read_csv("steam_renamed_dataset.csv")
df = df.dropna(subset=['genre'])

# Separazione dei generi multipli per ogni gioco
all_genres = df['genre'].str.split(';')
exploded = all_genres.explode()

# Conteggio dei giochi per genere singolo
genre_counts = exploded.value_counts().reset_index()
genre_counts.columns = ['genre', 'number_of_games']
print(genre_counts)
```

	genre	number_of_games
0	Indie	19421
1	Action	11903
2	Casual	10210
3	Adventure	10032
4	Strategy	5247
5	Simulation	5194
6	RPG	4311
7	Early Access	2954
8	Free to Play	1704
9	Sports	1322
10	Racing	1024

Analizziamo ora la distribuzione dei generi in base alla categoria di prezzo (gratuito o a pagamento) tramite un grafico a barre.

```
import matplotlib.pyplot as plt
df = pd.read_csv("steam_renamed_dataset.csv")
# Rimuove righe con genere nullo
df = df.dropna(subset=['genre'])
# Crea colonna di categoria prezzo
df['price_category'] = df['game_price'].apply(lambda x: 'Free' if x == 0 else 'Paid')
# Divide i generi multipli in righe singole
df_exploded = df.assign(genre_split=df['genre'].str.split(';')).explode('genre_split')
# Ottieni i 10 generi più comuni
top_genres = df_exploded['genre_split'].value_counts().nlargest(10).index
df_top = df_exploded[df_exploded['genre_split'].isin(top_genres)]

# Grafico a barre
plt.figure(figsize=(12, 6))
sns.countplot(data=df_top, x='genre_split', hue='price_category', order=top_genres, palette='muted')

plt.title('Distribuzione dei generi nei giochi Free vs Paid')
plt.xlabel('Genere')
plt.ylabel('Numero di giochi')
plt.xticks(rotation=45)
plt.grid(True)
plt.tight_layout()
plt.show()
```



Infine, analizziamo la relazione tra il tipo di gioco (gratuito o a pagamento) e i diversi generi, mostrando i generi più frequenti all'interno di ciascuna categoria di prezzo.

```python
df = pd.read_csv("steam_renamed_dataset.csv")

# Classifica i giochi in Free o Paid
df['price_category'] = df['game_price'].apply(lambda x: 'Free' if x == 0 else 'Paid')
df = df.dropna(subset=['genre'])
df['genre_split'] = df['genre'].str.split(';')
df_exploded = df.explode('genre_split')

# Raggruppa per categoria di prezzo e genere
genre_counts = df_exploded.groupby(['price_category', 'genre_split']).size().reset_index(name='count')
 .sort_values(by='count', ascending=False)
print(genre_counts.head(10))
```

```
    price_category   genre_split  count
40            Paid         Indie  17864
28            Paid        Action  10946
32            Paid        Casual   9316
29            Paid     Adventure   9300
50            Paid      Strategy   4731
47            Paid    Simulation   4709
44            Paid           RPG   3835
35            Paid  Early Access   2605
9             Free  Free to Play   1671
12            Free         Indie   1557
```

2. Recommender System

2.1 Sommario

Un sistema di raccomandazione, o ***recommender system***, è un software progettato per filtrare i contenuti e fornire suggerimenti personalizzati agli utenti, supportandoli nel processo decisionale. Esistono tre principali approcci per la generazione delle raccomandazioni: **collaborativo**, **basato sui contenuti** (*content-based*) e **ibrido**.
Nel progetto, è stato scelto di adottare l'approccio **Content-based Filtering**, poiché il dataset selezionato, composto da un ampio insieme di videogiochi disponibili su Steam, ci ha consentito di lavorare su una varietà di informazioni descrittive e strutturate. Tra queste troviamo: titolo del gioco, genere, piattaforma di riferimento, numero di valutazioni positive e negative, tempo medio di gioco, prezzo e sviluppatore.
Questo approccio si fonda sull'analisi delle caratteristiche del contenuto dei videogiochi e sul confronto tra questi e le preferenze esplicite dell'utente. Tali caratteristiche vengono utilizzate per generare suggerimenti di altri giochi che presentano attributi simili a quelli preferiti dall'utente.

Nel nostro caso specifico, non disponendo di uno storico utenti, si è optato per una modalità interattiva, in cui è l'utente stesso a indicare un videogioco di riferimento. Il sistema, a partire da questo input, restituisce una serie di titoli simili per contenuto, sfruttando le informazioni presenti nel dataset per garantire raccomandazioni mirate e coerenti con i gusti dichiarati.

```
Inserisci un valore: 1
Recommender System

BENVENUTO NEL GAME RECOMMENDER SYSTEM

Inserisci i dati con la lettera maiuscola iniziale

Inserisci il titolo del gioco:
Portal 2
Inserisci il genere (es. Action):
action
Inserisci l'anno di uscita (YYYY):
2011
```

```
Dati inseriti:
      title   genre  ... positive_ratings  negative_ratings
0  Portal 2  action  ...                 0                 0

[1 rows x 7 columns]

Confermi i dati? (y/n): y
```

2.2 Scelta metrica da impiegare

Prima di procedere con lo sviluppo del sistema di raccomandazione, ci siamo dedicati allo studio delle metriche di similarità per valutare quanto due videogiochi possano essere considerati simili tra loro, sulla base delle caratteristiche contenute nel dataset.
È stata utilizzata la correlazione di Pearson, una delle misure statistiche più comuni per determinare la relazione lineare tra due variabili quantitative. Nel nostro caso, questa metrica è stata utile per confrontare valori numerici associati ai giochi, come il prezzo, il tempo medio di gioco o le valutazioni degli utenti, e

stabilire quanto siano correlati tra loro.
L'indice di Pearson può assumere valori compresi tra -1 e +1:

- Un valore vicino a +1 indica una correlazione positiva forte (es. due giochi con caratteristiche simili).
- Un valore vicino a -1 indica una correlazione negativa forte.
- Un valore vicino a 0 suggerisce assenza di relazione lineare tra le due variabili.

Questa misura si è rivelata particolarmente utile per implementare un sistema di raccomandazione che identifica e suggerisce giochi simili a quello fornito in input dall'utente.

2.3 Realizzazione

Per realizzare il sistema di raccomandazione basato sul **Content-Based Filtering**, è stata utilizzata la libreria open source **Scikit-Learn (SKLearn)**, che ci ha permesso di applicare tecniche di **Natural Language Processing (NLP)** per trasformare i dati testuali del nostro dataset di videogiochi in vettori numerici confrontabili.
Sono stati uniti più attributi del dataset, come ad esempio **titolo**, **genere** e **anno di rilascio**, all'interno di una singola colonna chiamata 'combined'. Questo ha permesso di creare una rappresentazione testuale unificata per ciascun videogioco. Per trasformare questa colonna in una forma utile al calcolo delle similarità, abbiamo utilizzato la tecnica **TF-IDF (Term Frequency – Inverse Document Frequency)**, che consente di convertire il contenuto testuale in **vettori numerici** che riflettono l'importanza relativa di ciascuna parola all'interno del corpus.
Il seguente frammento di codice mostra come abbiamo ottenuto la matrice TF-IDF:

```
# Funzione di vettorizzazione
def vectorize_data(df):  1 usage
    vectorizer = TfidfVectorizer()
    tfidf = vectorizer.fit_transform(df['combined'])
    return tfidf
```

Successivamente, la matrice TF-IDF è stata convertita in array (tfidf_array) per consentire il calcolo delle correlazioni tra vettori tramite la Correlazione di Pearson, che ci ha permesso di misurare la similarità lineare tra i videogiochi. Il seguente codice esegue il confronto tra il videogioco scelto dall'utente (tramite indice user_index) e tutti gli altri nel dataset:

```
tfidf_matrix = vectorize_data(df)
tfidf_array = tfidf_matrix.toarray()

print("\nInizio ricerca dei giochi simili...")

# Calcolo correlazione tra gioco utente e tutti gli altri
correlation = []
for i in range(len(tfidf_array)):
    corr, _ = pearsonr(tfidf_array[user_index], tfidf_array[i])
    correlation.append((i, corr))

sorted_corr = sorted(correlation, key=lambda x: x[1], reverse=True)[1:6]
game_indexes = [i[0] for i in sorted_corr]

print("\n[5 giochi più simili trovati]")
return df.iloc[game_indexes][['title', 'genre', 'release_date']]
```

In questo modo, viene generata una **matrice di similarità** tra il gioco scelto dall'utente e tutti gli altri

presenti nel dataset. Ogni valore della matrice indica quanto due giochi sono simili in base alle informazioni combinate (titolo, genere, anno). Per restituire le raccomandazioni, effettuiamo un **reverse mapping** tra titolo e indice nel dataframe, in modo da poter partire dal nome del videogioco scelto per ottenere i **5 titoli più simili**.

```
Inizio ricerca dei giochi simili...

[5 giochi più simili trovati]

Ecco i giochi consigliati:

                              title  ... release_date
23661                Adventure Portal  ...   2018-10-31
19348                     Mind Portal  ...   2018-01-12
21199  Fighting Fantasy Legends Portal  ...   2018-07-11
18453        Portal Journey: Portarius  ...   2018-04-26
17940                  EXIT 4 - Portal  ...   2018-02-12

[5 rows x 3 columns]
```

2.4 Classificazione

Oltre allo sviluppo del sistema di raccomandazione, è stato introdotto un meccanismo di classificazione dei videogiochi basato su una nuova categoria da noi definita, denominata “star”. Questa metrica è stata ideata per fornire una valutazione sintetica della qualità di un gioco, partendo da dati già presenti nel dataset, in particolare le valutazioni positive e negative ricevute da ciascun titolo.
La metrica “star” è stata calcolata applicando la formula:

```
game_data['star'] = ((game_data['positive_ratings'] /
                      (game_data['positive_ratings'] + game_data['negative_ratings'] + 1)) * 5)

# Classifichiamo i giochi in fasce di qualità
game_data.loc[(game_data['star'] >= 4.0), 'star'] = 5
game_data.loc[(game_data['star'] < 4.0) & (game_data['star'] >= 3), 'star'] = 4
game_data.loc[(game_data['star'] < 3) & (game_data['star'] >= 2), 'star'] = 3
game_data.loc[(game_data['star'] < 2) & (game_data['star'] >= 1), 'star'] = 2
game_data.loc[(game_data['star'] < 1), 'star'] = 1
```

Questa formula assegna un punteggio da 0 a 5, in cui:

- il numeratore rappresenta il numero di valutazioni positive,
- il denominatore somma le valutazioni totali (positive + negative), a cui è aggiunto 1 per evitare eventuali divisioni per zero,
- il tutto viene moltiplicato per 5, normalizzando il punteggio nella scala delle stelle (1-5).

Grazie a questa classificazione, ogni videogioco nel dataset è stato assegnato a una categoria “star” ben definita, che potrà essere utilizzata per addestrare un modello di classificazione supervisionato in grado di prevedere la qualità di altri titoli.
Per quanto riguarda la scelta del modello di classificazione, è stato utilizzato il KNNeighborsClassifier, disponibile nella libreria SKLearn.

Modello adottato

Il K-Nearest Neighbors (K-NN) è un algoritmo ampiamente utilizzato nell'ambito del machine learning per compiti di classificazione e regressione. Il suo funzionamento si basa sull'idea che un oggetto può essere classificato osservando i suoi k elementi più simili (i "vicini") all'interno di uno spazio delle caratteristiche. L'algoritmo assume che oggetti con caratteristiche simili tendano a trovarsi vicini nello spazio dei dati.
Nel caso della classificazione, un elemento viene assegnato alla classe più comune tra quelle dei suoi vicini. Ad esempio, se k=3, l'elemento verrà classificato in base alla classe presente in almeno due dei tre vicini più prossimi. Quando k=1, viene semplicemente considerato il vicino più prossimo.
Per selezionare il modello più adatto al nostro obiettivo, abbiamo consultato la mappa interattiva degli algoritmi fornita dalla libreria Scikit-learn, che suggerisce i modelli più indicati in base al tipo di problema affrontato.
Nel nostro caso, il contesto rispecchia perfettamente le condizioni suggerite per l'uso del K-NN:

- stiamo lavorando su un problema di classificazione (attribuzione della categoria "star"),
- disponiamo di dati categorizzati,
- e le informazioni utilizzate non sono di tipo testuale.

Per questo motivo, K-NN rappresenta una scelta coerente ed efficace nel nostro scenario.

Dataset

Per preparare il dataset all'addestramento del modello di classificazione, è stato suddiviso in due porzioni: una dedicata al training e una al test. Il modello verrà allenato sui dati di training e successivamente testato sulla restante parte per valutarne l'efficacia. In particolare, abbiamo scelto una ripartizione dell'80% per il training e del 20% per il test.
È importante che questa suddivisione avvenga in modo casuale, così da garantire che entrambe le porzioni siano rappresentative dell'intero dataset. Per ottenere ciò, si utilizza il parametro random_state, impostato con un numero intero a scelta. Questo parametro non influisce sull'accuratezza, ma assicura la ripetibilità della suddivisione.
Inoltre, in presenza di classi sbilanciate — ovvero categorie con un numero molto diverso di esempi — è consigliabile mantenere la stessa distribuzione delle classi sia nei dati di training che di test. Questa procedura prende il nome di split stratificato, ed è attuabile grazie al parametro stratify, a cui viene passato il vettore contenente le etichette (y) del dataset.

```
# Train/test split
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

È stata effettuata la **scalatura dei dati** per rendere omogeneo l'intervallo di variazione delle diverse caratteristiche presenti nel dataset. Questo passaggio è fondamentale, in quanto consente all'algoritmo di **apprendere in modo più efficiente**, riducendo i tempi di convergenza e migliorando la qualità complessiva dei risultati ottenuti.

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
predict_data = scaler.transform(predict_data)
```

Una volta normalizzati i dati, il **modello viene addestrato** utilizzando la porzione di training. Per l'addestramento abbiamo sfruttato le **funzionalità offerte dalla libreria SKLearn**, utilizzando i parametri di

default del classificatore selezionato.

```
print("\n\nComposizione iniziale del modello con iper-parametri di base...")
knn = KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', p=2, metric='minkowski')
knn.fit(X_train, y_train)
```

Miglioramento del modello e Hyperparameters Tuning

Dopo aver testato il modello di classificazione basato sui parametri predefiniti, abbiamo effettuato una prima valutazione delle sue performance nella previsione della categoria "star" dei videogiochi. A tal fine, abbiamo utilizzato il classification_report fornito dalla libreria sklearn.metrics, che ha restituito diverse metriche di valutazione come precision, recall e f1-score per ciascuna classe.

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

Predizioni dei primi 5 elementi:  [4. 4. 3. 1. 2.] Valori effettivi:  [4. 4. 3. 1. 2.]

Valutazione del modello...
```

```
Classification Report:
              precision    recall  f1-score   support

         1.0       0.79      0.68      0.73       206
         2.0       0.71      0.67      0.69       480
         3.0       0.73      0.76      0.74      1074
         4.0       0.80      0.84      0.82      1832
         5.0       0.92      0.88      0.90      1823

    accuracy                           0.82      5415
   macro avg       0.79      0.77      0.78      5415
weighted avg       0.82      0.82      0.82      5415

ROC Score:  0.9442022273929644
```

Inoltre, abbiamo calcolato il ROC AUC Score, ottenendo un valore iniziale di **0.94**, a testimonianza di un buon livello di accuratezza del modello.

Il ROC AUC Score rappresenta l'area sotto la curva ROC e offre una misura complessiva dell'efficacia del classificatore su tutte le possibili soglie. Un valore più vicino a 1 indica un'elevata capacità di discriminazione tra le classi.

Per migliorare ulteriormente le performance del modello, è stata introdotta una fase di **ottimizzazione degli iper-parametri**.

In particolare, è stato fatto uso di RandomizedSearchCV, in combinazione con una strategia di validazione incrociata RepeatedKFold, per testare differenti combinazioni di parametri in modo efficiente. Rispetto alla classica Grid Search, l'approccio randomizzato si è rivelato più rapido ed efficace, esplorando in maniera intelligente lo spazio delle possibili configurazioni.

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

I parametri presi in considerazione per l'ottimizzazione sono stati:

- il numero di vicini (n_neighbors) nel range da 1 a 30,
- il tipo di pesatura (weights: uniforme o basata sulla distanza),
- la metrica di distanza (metric: euclidea, manhattan o hamming).

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

Dopo aver eseguito diverse iterazioni, il modello ha individuato la combinazione ottimale: n_neighbors = 14, weights = distance, metric = manhattan. Utilizzando questi valori, abbiamo ricostruito il modello e ottenuto un significativo incremento delle metriche: l'**accuracy** è salita a **0.85** e il **ROC AUC Score** ha raggiunto un valore di **0.96**. Anche le metriche di precision e recall hanno mostrato un miglioramento per tutte le classi, dimostrando l'efficacia della fase di tuning.

```
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:  14

Ricomponiamo il modello utilizzando i nuovi iper-parametri:

Predizione dei primi 5 elementi:  [4. 4. 3. 1. 2.] Valori effettivi:  [4. 4. 3. 1. 2.]
Classification Report:
              precision    recall  f1-score   support

         1.0       0.92      0.79      0.85       206
         2.0       0.84      0.71      0.77       480
         3.0       0.81      0.78      0.79      1074
         4.0       0.81      0.89      0.85      1832
         5.0       0.91      0.90      0.91      1823

    accuracy                           0.85      5415
   macro avg       0.86      0.81      0.83      5415
weighted avg       0.85      0.85      0.85      5415

ROC Score:  0.9692177062704133
```

Predizione sui nuovi dati

Dopo aver ottimizzato gli iper-parametri del nostro modello di classificazione, è stata utilizzata la nuova configurazione per effettuare predizioni sulla qualità (categoria "star") dei videogiochi consigliati all'utente. La lista restituita comprende titoli ritenuti simili a quello di partenza, accompagnati da una stima della loro qualità basata sul modello ottimizzato.

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

Ecco una lista di giochi simili con predizione sulla qualità (star):

                              title                              genre  \
23661                Adventure Portal  Adventure;Indie;RPG;Early Access
19348                     Mind Portal            Adventure;Casual;Indie
21199  Fighting Fantasy Legends Portal               Adventure;Indie;RPG
18453       Portal Journey: Portarius            Adventure;Casual;Indie
17940                 EXIT 4 - Portal                      Casual;Indie

      release_date  star_prediction
23661   2018-10-31              3.0
19348   2018-01-12              3.0
21199   2018-07-11              4.0
18453   2018-04-26              5.0
17940   2018-02-12              4.0
```

Sebbene le predizioni non siano perfettamente identiche a quelle ottenute nella fase precedente, è possibile notare un miglioramento generale nella coerenza dei risultati, confermando l'efficacia dell'ottimizzazione effettuata.

3. KB

3.1 Introduzione

Una Knowledge Base (KB) è una struttura organizzata che raccoglie informazioni e regole, utilizzabili da un sistema per risolvere problemi, rispondere a domande o prendere decisioni in modo ragionato.
Essa rappresenta conoscenze sul mondo reale, sotto forma di fatti e regole logiche, che consentono di combinare queste informazioni per dedurre nuovi dati o supportare il processo decisionale.

In altre parole, una KB può essere vista come un insieme di proposizioni — chiamate assiomi — considerate vere, e utilizzate per costruire un ambiente in grado di raccogliere, organizzare e diffondere conoscenza.

Nel contesto del progetto, la Knowledge Base è stata implementata per offrire all'utente uno strumento interattivo, capace di rispondere a domande specifiche sull'universo dei videogiochi, facilitando così l'esplorazione del dominio trattato.

3.2 Gestione della KB

Per la gestione della Knowledge Base abbiamo utilizzato l'estensione **PySwip** per Python, che consente l'interfacciamento diretto con il linguaggio logico **Prolog**.
Attraverso le potenzialità offerte da Prolog, siamo stati in grado di creare e popolare automaticamente la KB con una serie di **fatti**, estratti direttamente dal nostro dataset.
In seguito, sono state definite delle **regole logiche** che sfruttano tali fatti per strutturare risposte e inferenze, simulando un'interazione naturale con l'utente. Questo approccio ha reso la KB dinamica e interrogabile, consentendo all'utente di porre domande specifiche e ricevere risposte basate sulle conoscenze memorizzate.

3.3 Fatti

I fatti costituiscono la base della Knowledge Base e rappresentano informazioni che si assumono sempre vere.

1) "name (id, name)"
   Rappresenta la relazione tra un videogioco e il suo titolo.

2) “genre (id, genre)”
   Indica il genere associato a ciascun videogioco.
3) “release_year (id, release_year)”
   Specifica l’anno di uscita del videogioco.
4) “average_playtime (id, average_playtime)”
   Indica il tempo medio di gioco (in minuti) associato al titolo.
5) “game_price (id, game_price)”
   Specifica il costo del videogioco in euro.

```python
import pandas as pd

def clean_string(s):  1 usage
    """Pulisce e normalizza una stringa per l'uso in Prolog."""
    return str(s).replace( _old: "'", _new: "").encode( encoding: 'ascii', errors: 'ignore').decode()

class KnowledgeBase:  1 usage
    def createKnowledgeBase(self):  1 usage
        df = pd.read_csv('steam_renamed_dataset.csv')

        # Pulizia delle stringhe
        columns_to_string = ['name', 'genre']
        df[columns_to_string] = df[columns_to_string].apply(lambda x: x.apply(clean_string))

        with open('game_kb.pl', 'w', encoding='utf-8') as f:
            f.write(":- discontiguous name/2.\n")
            f.write(":- discontiguous genre/2.\n")
            f.write(":- discontiguous release_year/2.\n")
            f.write(":- discontiguous average_playtime/2.\n")
            f.write(":- discontiguous game_price/2.\n\n")

            for _, row in df.iterrows():
                f.write(f"name({row['id']}, '{row['name']}').\n")
                f.write(f"genre({row['id']}, '{row['genre']}').\n")
                f.write(f"release_year({row['id']}, {int(row['release_year'])}).\n")
                f.write(f"average_playtime({row['id']}, {int(row['average_playtime'])}).\n")
                f.write(f"game_price({row['id']}, {float(row['game_price'])}).\n")
```

È stata utilizzata 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 gioco e non per predicato.

3.4 Regole

Le **regole** rappresentano il meccanismo che consente all’utente di interagire con la knowledge base (KB) tramite semplici comandi, senza dover conoscere la sintassi del linguaggio Prolog.
Queste regole utilizzano i **fatti** già presenti nella KB per elaborare le risposte alle richieste dell’utente. Nel progetto sono stati implementati due **task principali**, che l’utente può selezionare tramite un menu interattivo:

1. **Trovare un videogioco in base a specifiche caratteristiche** (ad esempio: genere, anno di rilascio, durata media di gioco).
2. **Determinare la fascia di prezzo più frequente** tra i giochi risultati da una determinata ricerca.

```
Scegliere quale funzione eseguire:
1) Trova un gioco in base a determinate caratteristiche
2) Trova la fascia di prezzo migliore per i giochi selezionati
3) Esci
```

3.5 Trova un gioco in base a determinate caratteristiche

Il **primo task** consente di trovare tutti i videogiochi che soddisfano una combinazione di caratteristiche specifiche fornite dall'utente.
Per implementarlo, sono state definite diverse **regole Prolog**, ognuna rappresentante un possibile filtro applicabile ai dati presenti nella knowledge base.
Di seguito vengono riportati alcuni esempi rappresentativi:

- "prezzo_economy(ID) :- monthly_subscription_cost(ID, Cost), Cost < 9.99. "
  Questa regola definisce che un videogioco è considerato **economico** se il suo costo di abbonamento mensile è inferiore a 9,99€.

- "game_recente(ID) :- release_year(ID, Uscita), Uscita > 2010."
  Un videogioco è considerato **recente** se è stato rilasciato dopo il 2010.

- "game_genre(ID, Genre) :- genre(ID, Genre)."
  Questa regola verifica se un videogioco appartiene a un determinato **genere**.

- "game_breve_durata(ID) :- average_playtime(ID, Durata), Durata =< 60."
  Il gioco è classificato come **breve** se la sua durata media di gioco è inferiore o uguale a 60 minuti.

- "recente_genre_durata(ID, Genre, DurataCat) :- game_recente(ID), game_genre(ID, Genre), game_breve_durata(ID)."
  Questa regola **combinata** definisce un gioco come recente, appartenente a un certo genere e di breve durata, solo se tutte le tre condizioni sono soddisfatte.

Queste regole vengono attivate dinamicamente a seconda delle selezioni effettuate dall'utente, permettendo di costruire query flessibili e potenti.

Esempio di esecuzione:

```
Inserisci il periodo di rilascio del gioco:
1) recente
2) tra 2000 e 2010
3) pre 2000
1

Scegli un genere tra: action, indie, rpg, strategy, simulation, sports, casual
Genere: rpg

Quanto tempo medio di gioco?
1) breve
2) medio
3) lungo
2
```

```
Giochi trovati:
game451
game521
game756
game768
game886
game990
game1320
game1926
game2095
game2133
game2796
game2861
game2967
game2968
game3436
game4281
game4946
game5246
game5781
```

3.6 Trova la fascia di prezzo migliore per i giochi selezioni

Il secondo task ha l'obiettivo di determinare la **fascia di prezzo più comune** tra un insieme di videogiochi selezionati in base alle preferenze dell'utente espresse nel primo task.
Per realizzare questa funzionalità è stata implementata la funzione find_most_common_price_range, che prende in input un insieme di ID dei giochi e interroga la knowledge base per identificare la fascia di prezzo associata a ciascun videogioco.

```python
def find_most_common_price_range(prolog, game_ids):  1 usage
    fascia_count = {}

    for game_id in game_ids:
        query = f"prezzo('{game_id}', Fascia)."
        try:
            results = list(prolog.query(query))
            for res in results:
                fascia = res["Fascia"]
                if fascia in fascia_count:
                    fascia_count[fascia] += 1
                else:
                    fascia_count[fascia] = 1
        except Exception as e:
            print(f"Errore nella query per {game_id}: {e}")

    if fascia_count:
        best_fascia = max(fascia_count, key=fascia_count.get)
        print(f"\n La fascia di prezzo più comune tra i giochi trovati è: {best_fascia}")
    else:
        print("Nessuna informazione sui prezzi disponibile per i giochi trovati.")
```

La funzione costruisce dinamicamente una query utilizzando il predicato "prezzo/2", il quale associa un videogioco a una determinata fascia di prezzo. Viene poi effettuato un conteggio delle ricorrenze per ciascuna fascia, e infine viene restituita quella con il **numero maggiore di occorrenze** tra i giochi analizzati. Questa logica permette di consigliare la fascia di prezzo più diffusa tra i giochi che l'utente ha selezionato, aiutandolo così a orientarsi tra le opzioni economiche, medie o costose.

Le fasce di prezzo vengono definite da regole nella KB come:

- prezzo_economy(ID) :- game_price(ID, Cost), Cost < 9.99.
  Indica che un gioco ha **prezzo economico** se costa meno di 9.99€.

- prezzo_medio(ID) :- game_price(ID, Cost), Cost = 9.99.
  Indica che un gioco ha **prezzo medio** se costa esattamente 9.99€.

- prezzo_costoso(ID) :- game_price(ID, Cost), Cost > 9.99.
  Indica che un gioco ha **prezzo costoso** se il prezzo è superiore a 9.99€.

In questo modo, il sistema sfrutta i **fatti presenti nella knowledge base** per guidare l'utente nella scelta più adatta alle proprie disponibilità economiche.

Esempio di esecuzione:

```
Scegliere quale funzione eseguire:
1) Trova un gioco in base a determinate caratteristiche
2) Trova la fascia di prezzo migliore per i giochi selezionati
3) Esci

Inserisci un valore: 2

 La fascia di prezzo più comune tra i giochi trovati è: media
```

4. Ontologia

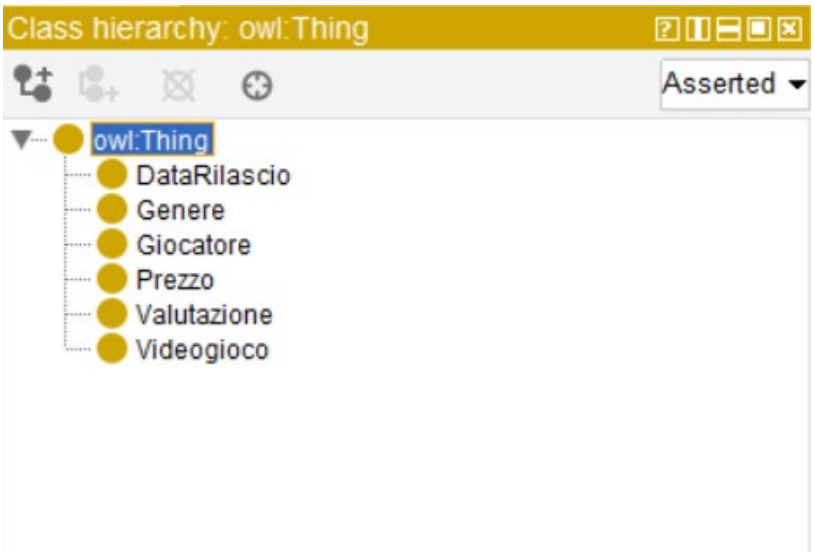
4.1 Sommario

In ambito informatico, un'ontologia è una rappresentazione formale della conoscenza relativa a un dominio specifico, in cui vengono definite le entità coinvolte, le loro caratteristiche e le relazioni tra di esse. Si tratta di uno strumento utile per strutturare, organizzare e condividere informazioni in modo semantico e coerente.
Nel progetto, è stato deciso di sviluppare un'ontologia orientata al dominio dei videogiochi. Questa permette di descrivere in modo strutturato le principali entità coinvolte, come videogiochi, generi, anno di uscita, prezzo, e valutazioni. Grazie a questa ontologia, è possibile effettuare interrogazioni mirate, ottenere informazioni dettagliate sui titoli presenti e supportare l'utente nella scelta e nell'esplorazione dei contenuti videoludici.

4.2 Protègè

Per la realizzazione dell'ontologia relativa al dominio dei videogiochi, è stato utilizzato Protégé, un editor open source per la modellazione semantica.
L'ontologia sviluppata include diverse classi, ognuna delle quali rappresenta un concetto chiave nel contesto videoludico.
Le principali classi create sono:

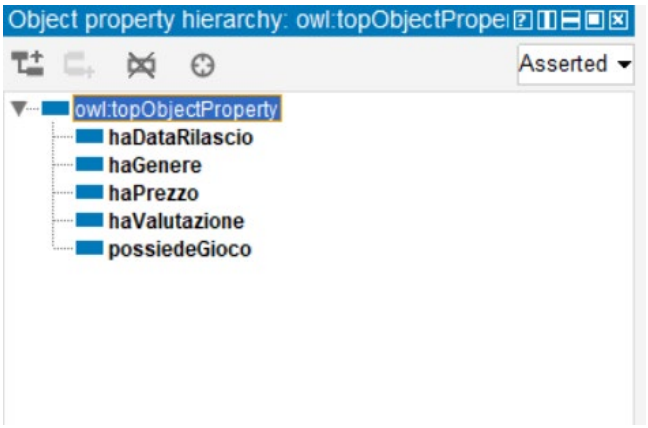
- Videogioco: elemento centrale dell'ontologia, a cui fanno riferimento tutte le altre classi.
- Genere, Prezzo, Valutazione, DataRilascio, Giocatore: categorie che descrivono nel dettaglio le caratteristiche del videogioco.

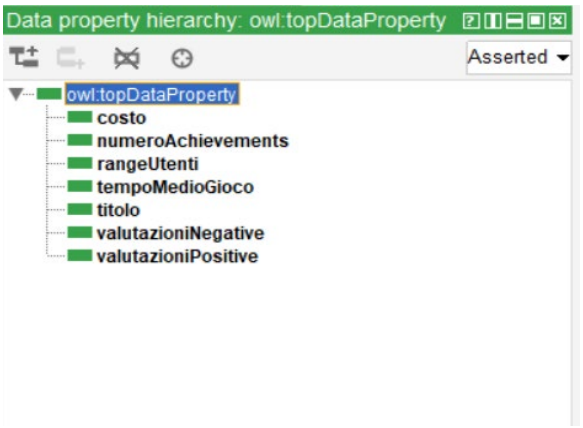


Una volta definite le classi, sono state create una serie di proprietà, sia object properties che data properties:

- Le object properties (es. haGenere, haPrezzo, haValutazione, haDataRilascio, possiedeGioco) sono utilizzate per collegare istanze di classi diverse, come ad esempio un videogioco al suo genere o al suo prezzo.
- Le data properties, invece, permettono di associare ai singoli individui dei valori primitivi (es. numeri, stringhe) per descriverne le caratteristiche quantitative o qualitative.

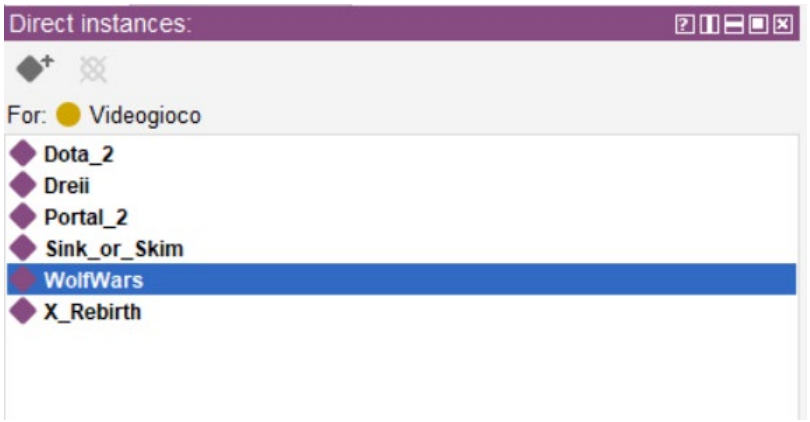
Questa struttura consente di rappresentare in modo formale e interrogabile la conoscenza relativa ai videogiochi, facilitando l'integrazione con sistemi intelligenti come motori di raccomandazione o interfacce di interrogazione.



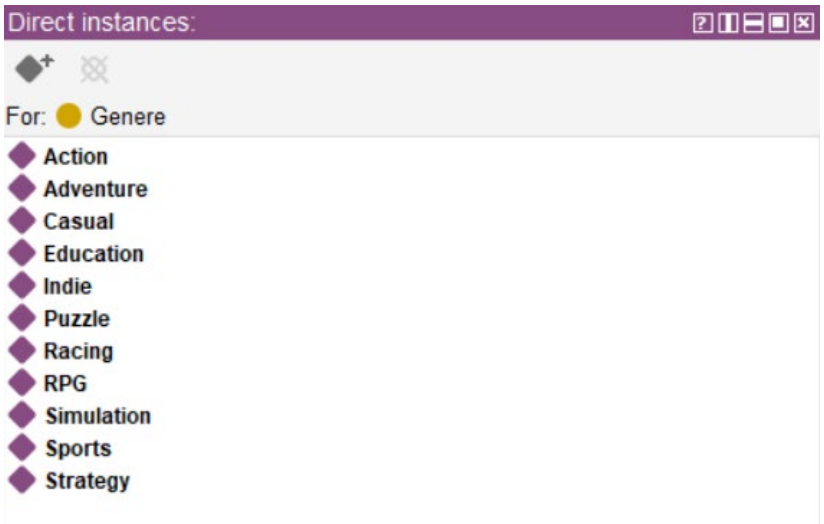


Successivamente siamo passati alla realizzazione degli individui stessi che rappresentano alcuni esempi di film presenti nel dataset.

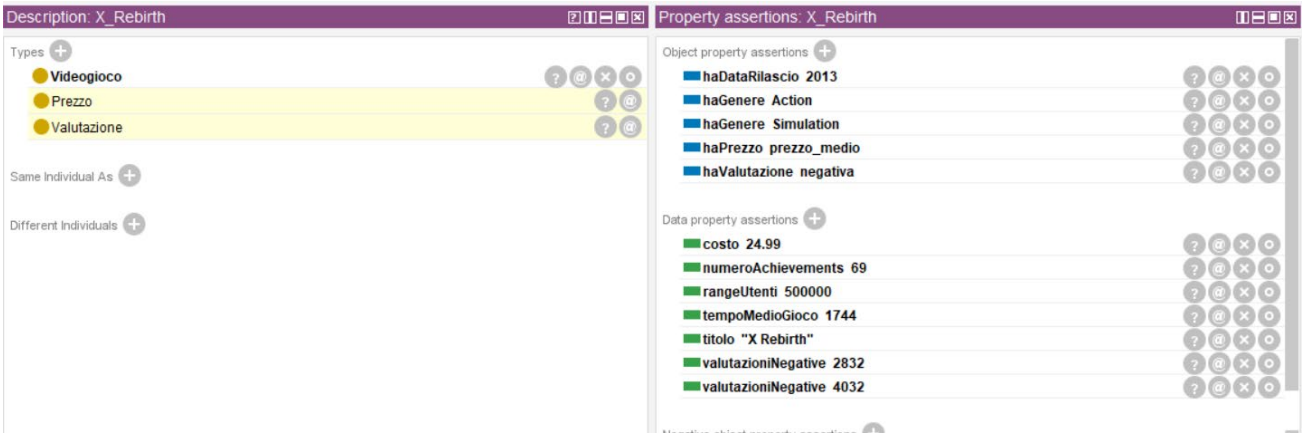
Individui classe videogioco



Individui classe genere

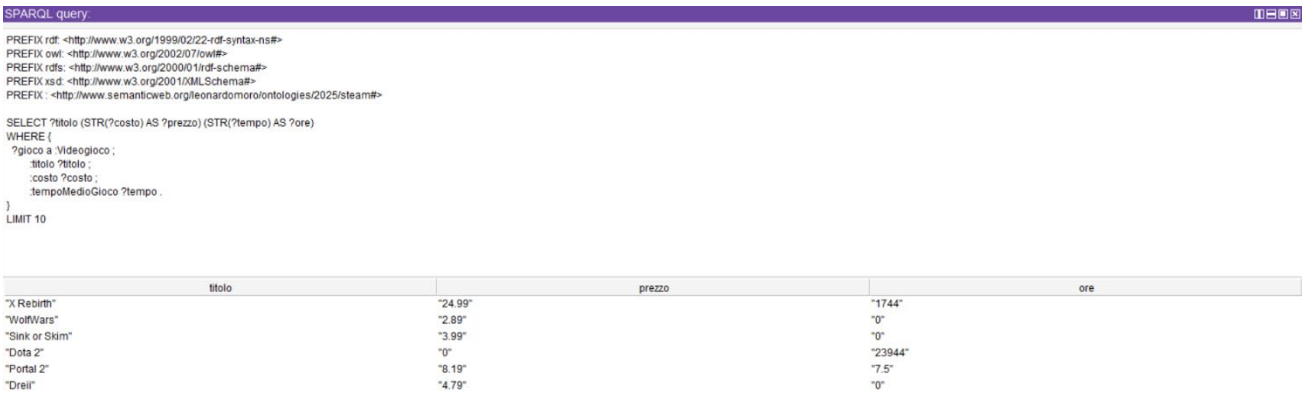


Esempio di individuo con properties annesse

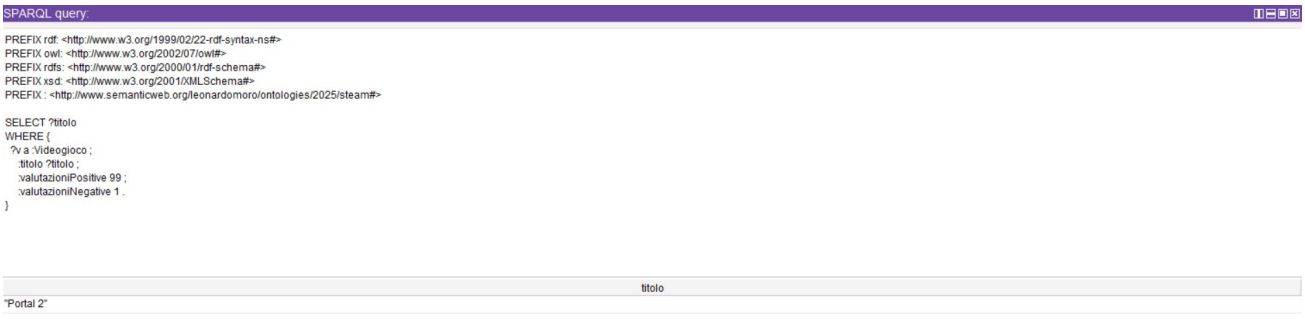


Possiamo anche interrogare l'ontologia tramite l'utilizzo di due tipologie differenti di query, DL query e SPARQL query.

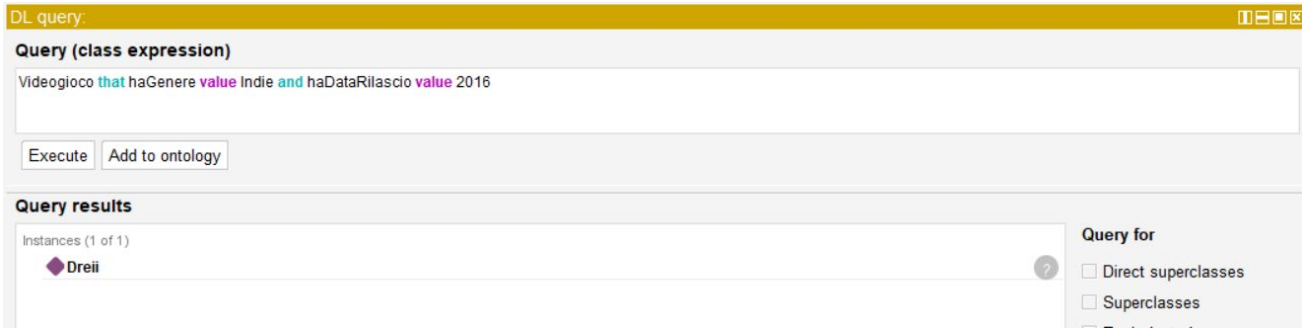
SPARQL query con visualizzazione dei titoli, prezzo e ore di gioco dei videogiochi presenti



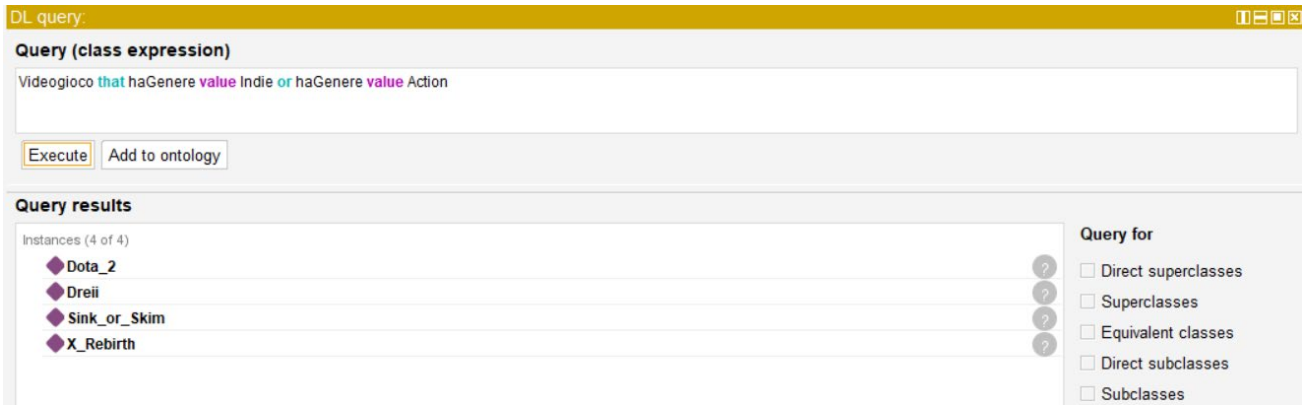
SPARQL query con visualizzazione dei giochi che hanno come valutazioni positive = 99 e valutazioni negative = 1



DL-query con ricerca dei giochi che hanno genere "Indie" e come anno di rilascio "2016"



DL-query con ricerca di giochi che hanno come genere "Action" oppure "Indie"



4.3 Owlready2

È stata utilizzata la libreria Owlready2 per interrogare l'ontologia all'interno di un ambiente Python. Questo strumento ci ha permesso di accedere e navigare l'ontologia precedentemente creata con Protégé, in maniera semplice e funzionale.
Durante l'esecuzione del programma, l'utente può scegliere di:

- visualizzare le classi presenti nell'ontologia,
- esplorare le object properties e le data properties definite,
- oppure eseguire delle query di esempio per interrogare l'ontologia e ottenere informazioni specifiche.

Segue il menù iniziale che consente di esplorare la nostra ontologia

```
BENVENUTO NELL'ONTOLOGIA DEI VIDEOGIOCHI

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, potremmo visualizzare le classi presenti all'interno della nostra ontologia:

```
Inserisci un valore: 1

CLASSI PRESENTI NELL'ONTOLOGIA:
- steam.DataRilascio
- steam.Genere
- steam.Giocatore
- steam.Prezzo
- steam.Valutazione
- steam.Videogioco

Vorresti esplorare meglio una delle seguenti classi?
1) Videogioco
2) Genere
3) Prezzo
4) Data di Rilascio
5) Valutazione
6) No
```

È possibile, inoltre, esaminare nel dettaglio le classi presenti nell'ontologia selezionando quella di interesse tra quelle visualizzate a schermo.
Nell'immagine di esempio, è stata selezionata la classe **Videogioco**. Verrà quindi mostrato a schermo un elenco contenente tutti i videogiochi presenti all'interno della nostra ontologia, ciascuno dei quali rappresenta un individuo dotato di proprietà che ne descrivono le caratteristiche principali, come **genere**, **anno di rilascio**, **valutazione** e **prezzo**.
Questa funzionalità permette all'utente di esplorare facilmente il contenuto semantico dell'ontologia e comprendere meglio la struttura e i dati rappresentati.

```
Inserisci un valore: 1

Individui della classe 'steam.Videogioco':
- ont.Dota_2
- ont.Dreii
- ont.Portal_2
- ont.Sink_or_Skim
- ont.WolfWars
- ont.X_Rebirth
```

Verrà richiesto alla fine dell'analisi di ciascuna delle classi proposte se l'utente desidera esamina 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

PROPRIETÁ D'OGGETTO PRESENTI NELL'ONTOLOGIA:
- steam.haDataRilascio
- steam.haGenere
- steam.haPrezzo
- steam.haValutazione
- steam.possiedeGioco
```

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

```
Inserisci un valore: 3

PROPRIETÁ DEI DATI PRESENTI NELL'ONTOLOGIA:
- steam.costo
- steam.numeroAchievements
- steam.rangeUtenti
- steam.tempoMedioGioco
- steam.titolo
- steam.valutazioniNegative
- steam.valutazioniPositive
```

Infine, è possibile accedere a una serie di **query di esempio** selezionando la **quarta opzione del menu**. Dopo averla scelta, il sistema ci chiederà di **indicare quale tra le query disponibili** desideriamo eseguire.

```
Inserisci un valore: 4

1) Videogiochi di genere 'Puzzle'
2) Videogiochi rilasciati nel 2011
3) Videogiochi con più di 50 valutazioni positive
4) Indietro

Scelta:
```

Esecuzione delle query messe a disposizione dal sistema:

PRIMA QUERY

```
1) Videogiochi di genere 'Simulation'
2) Videogiochi rilasciati nel 2013
3) Videogiochi con più di 50 valutazioni positive
4) Indietro

Scelta: 1

VIDEOGIOCHI DI GENERE SIMULATION
- X_Rebirth
```

SECONDA QUERY

```
1) Videogiochi di genere 'Simulation'
2) Videogiochi rilasciati nel 2013
3) Videogiochi con più di 50 valutazioni positive
4) Indietro

Scelta: 2

VIDEOGIOCHI RILASCIATI NEL 2013
- Dota_2
- X_Rebirth
```

TERZA QUERY

```
1) Videogiochi di genere 'Simulation'
2) Videogiochi rilasciati nel 2013
3) Videogiochi con più di 50 valutazioni positive
4) Indietro

Scelta: 3

VIDEOGIOCHI CON PIU' DI 50 VALUTAZIONI POSITIVE
- Dota_2
- Portal_2
```

5. Conclusioni

Il progetto **Games Recommender** rappresenta una solida base per la realizzazione di un sistema di raccomandazione nel dominio videoludico, sfruttando tecniche di classificazione e filtraggio basato sui contenuti.
Nonostante i risultati ottenuti siano soddisfacenti, esistono ampi margini di miglioramento ed estensione.
Un'evoluzione naturale del progetto potrebbe consistere nell'ampliamento del dataset attuale, integrando ulteriori fonti di dati relative ad altre piattaforme di distribuzione digitale di videogiochi.
Questa espansione permetterebbe di arricchire il sistema con una varietà più ampia di titoli, caratteristiche, valutazioni e informazioni, aumentando così la precisione e la personalizzazione delle raccomandazioni offerte.
In questo modo, il **Games Recommender** potrebbe diventare una soluzione ancora più completa e versatile, capace di adattarsi alle esigenze di un pubblico ampio e diversificato, e di rimanere al passo con l'evoluzione del mercato videoludico.