Progetto ICON

2024-2025

Andrea Di Caro a.dicaro4@studenti.uniba.it 776229

Paolo Giampietro p.giampietro2@studenti.uniba.it 780262

URL Repository: https://github.com/paologiampietro/ICon24-25.git

Sommario

1. INTRODUZIONE

Steam è una popolare piattaforma digitale sviluppata da **Valve Corporation**, specializzata nella distribuzione di un'ampia gamma di videogiochi, sia prodotti da Valve che da altri sviluppatori. Con oltre **120 milioni di utenti mensili attivi**, **Steam** si conferma come una delle principali piattaforme di distribuzione digitale a livello globale.

Considerando l'enorme utilizzo della piattaforma e la nostra esperienza diretta come giocatori, abbiamo scelto di analizzare **Steam** in questo **"caso di studio",** con l'obiettivo di sviluppare strumenti in grado di supportare gli utenti nella scoperta e nella scelta di nuovi giochi. A tal fine, abbiamo realizzato:

- Un **recommender system** che suggerisce titoli in base alle preferenze personali dell'utente, aiutandolo a orientarsi tra le migliaia di opzioni disponibili;

- **Un modello di classificazione** in grado di prevedere le valutazioni (**ratings**) degli utenti per diverse categorie di giochi;

- Una **knowledge base** interrogabile tramite un'interazione a **domanda-risposta**, che permette di verificare informazioni e ottenere dettagli sui videogiochi;

- Un'**ontology** che definisce formalmente il dominio di interesse, chiarendo il significato dei concetti e delle relazioni all'interno del sistema;

- Un **budget planner** raccomanda giochi in base al budget e alle preferenze dell'utente.

1.1 DATASET E LIBRERIE UTILIZZATI

Per questo progetto è stato utilizzato il **dataset** **Steam Store Games**, già sottoposto a un processo di data cleaning che ha incluso la rimozione di dati incompleti o inconsistenti, l'ordinamento delle informazioni e un controllo generale della qualità.

L'intero sviluppo è stato realizzato in **Python** utilizzando l'editor **VS Code**, con il supporto delle librerie presenti all'interno del file **"requirements.txt"**

1.2 MENU' PRINCIPALE

Una volta avviata l'applicazione, l'utente può navigare liberamente tra le diverse funzionalità, selezionando in qualsiasi momento quella più adatta alle sue esigenze. Grazie a un'interfaccia intuitiva, è possibile passare rapidamente da un modulo all'altro, tra:

- **Il sistema di raccomandazione**, per scoprire nuovi giochi in linea con i propri gusti;
- **La knowledge base**, per ottenere risposte immediate a domande specifiche;
- **L'ontologia**, che organizza e chiarisce le relazioni tra i concetti del dominio;
- **Il budget planner**, per gestire in modo intelligente le proprie spese sui videogiochi.

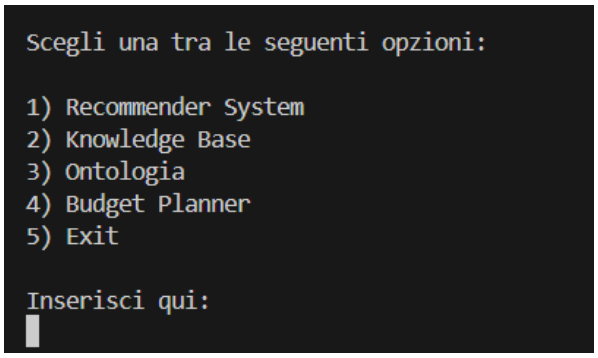

Figura 1.1 – Menù principale del programma

2. RECOMMENDER SYSTEM

Questo codice implementa un sistema di raccomandazione basato sul contenuto **(Content-based Filtering)** per videogiochi, che aiuta l'utente a trovare titoli simili a quelli che preferisce. Il sistema interagisce con l'utente, chiedendogli di inserire le caratteristiche di un gioco di riferimento (nome, sviluppatore, genere, ecc.) e verifica la correttezza dei dati prima di procedere.

Successivamente, carica un dataset di giochi esistenti, combina le informazioni testuali e le analizza utilizzando la **vettorizzazione TF-IDF e il calcolo della similarità del coseno**. In questo modo, **identifica i cinque giochi più simili a quello inserito dall'utente e li propone come raccomandazioni**. Inoltre, il sistema confronta diverse **metriche di similarità** (coseno, euclidea e Pearson) e valuta la **sparsità** dei dati per ottimizzare i risultati.

Questo approccio è particolarmente efficace quando non si dispone di un profilo utente predefinito, poiché si basa esclusivamente sulle caratteristiche del gioco fornito dall'utente per generare suggerimenti pertinenti.

```
=== INIZIO SISTEMA DI RACCOMANDAZIONE ===

Caricamento dati...

Preparazione dati...

Divisione dati...

Ottenimento raccomandazioni...
RECOMMENDER SYSTEM

Benvenuto, digita le caratteristiche del gioco su cui vuoi che si avvii la raccomandazione

(Ricorda di inserire i dati con la lettera maiuscola iniziale)

Inserisci il nome:
Counter-Strike

Inserisci lo sviluppatore:
Valve

Inserisci la casa pubblicatrice:
Valve

Inserisci le piattaforme:        (ricorda tra una parola e l'altra di mettere il simbolo ';' )
windows

Inserisci il genere:     (ricorda tra una parola e l'altra di mettere il simbolo ';' )
Action
```

Figura 2.1 Inserimento dei dati per la raccomandazione

```
Questo è il videogioco che hai inserito:

             name developer publisher platforms  genres
0  Counter-Strike     Valve     Valve   windows  Action

È corretto? (si/no):
```

Figura 2.2 Controllo dei dati inseriti

2.1 REALIZZAZIONE

Questo codice implementa un sistema di raccomandazione basato su contenuto (**Content-based Filtering**) per videogiochi, utilizzando tecniche di **Natural Language Processing (NLP)** e machine learning tramite l'utilizzo della libreria Python chiamata **SKLearn.**

```python
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
```

Figura 2.3 Libreria SKLearn

Il sistema chiede all'utente di inserire le caratteristiche di un videogioco **(nome, sviluppatore, publisher, piattaforme e generi)** e poi raccomanda i 5 giochi più simili presenti nel dataset Steam.

Questi **array lessicali** costituiscono una **codifica numerica** dei **termini** in un '**testo**', dove ogni vettore conserva un valore semantico per ognuno di essi.

$$TF(t,d) = \frac{number\ of\ times\ t\ appears\ in\ d}{total\ number\ of\ terms\ in\ d}$$
$$IDF(t) = log\frac{N}{1+df}$$
$$TF-IDF(t,d) = TF(t,d) * IDF(t)$$

Figura 2.4 Formula TF-IDF

Il funzionamento si basa sulla trasformazione delle informazioni testuali in vettori numerici tramite la tecnica **TF-IDF** (Term Frequency-Inverse Document Frequency), che assegna un peso alle parole in base alla loro importanza relativa. Questi vettori vengono poi confrontati utilizzando la similarità del coseno per trovare i giochi con caratteristiche più affini a quelle inserite dall'utente.

La libreria **SKLearn** contiene una classe built-in chiamata *TfIdfVectorizer* che si occupa di produrre la matrice TF-IDF.

```
# Funzione per vettorizzare i dati testuali usando TF-IDF
def vectorize_data(steam_data):
    # Gestione valori mancanti e vettorizzazione
    steam_data['all_content'] = steam_data['all_content'].fillna('')
    vectorizer = TfidfVectorizer(analyzer='word')
    return vectorizer.fit_transform(steam_data['all_content'])
```

Figura 2.5 Funzione per la vettorizzazione TfidfVectorizer

Una volta prodotta, si può procedere nel calcolo della similarità, con l'utilizzo della correlazione di **Pearson**.

$$Corr(x,y) = \frac{\sum_i (x_i - x)(y_i - y)}{\sqrt{\sum (x_i - \bar{x})^2}\sqrt{\sum (y_i - \bar{y})^2}}$$
$$= \frac{\langle x - x,\ y - y \rangle}{||x - x||\ ||y - y||}$$
$$= CosSim(x - \bar{x}, y - \bar{y})$$

Figura 2.6 Formula della correlazione di Pearson

Si ricava in questo modo una tabella che includerà i valori di affinità tra ciascun titolo e tutti gli altri. In altre parole, nella tabella ogni gioco corrisponde a un vettore-colonna in cui ogni elemento indica il grado di somiglianza con un altro videogioco.

```
# Calcolo similarità del coseno e selezione top 5 giochi
cosine_sim = cosine_similarity(tfidf_matrix[index:index+1], tfidf_matrix).flatten()
correlation = list(enumerate(cosine_sim))
sorted_corr = sorted(correlation, reverse=True, key=lambda x: x[1])[1:6]  # Esclude se stesso
games_index = [i[0] for i in sorted_corr]
```

Figura 2.7 Calcolo matrice

Per effettuare la raccomandazione vera e propria, impiegheremo una mappatura inversa tra i titoli dei videogiochi e gli indici estratti dal dataframe, ossia un sistema in grado di riconoscere l'indice del videogioco a partire dal suo nome.

```
Ecco a te i 5 giochi più simili a quello proposto:
                             name                 genres                      developer  price
10           Counter-Strike: Source   Action;FPS;Multiplayer                          Valve   7.19
7     Counter-Strike: Condition Zero   Action;FPS;Multiplayer                          Valve   7.19
25  Counter-Strike: Global Offensive  FPS;Multiplayer;Shooter  Valve;Hidden Path Entertainment   0.00
5                           Ricochet   Action;FPS;Multiplayer                          Valve   3.99
3                 Deathmatch Classic   Action;FPS;Multiplayer                          Valve   3.99
```

Figura 2-6: Risultato della raccomandazione

2.2 METRICHE

Prima di procedere con lo sviluppo del sistema di raccomandazione, abbiamo analizzato diverse metriche per valutare la similarità. In particolare, abbiamo confrontato tre approcci selezionati: la similarità coseno, la correlazione di Pearson e la distanza euclidea.

Tempo di esecuzione

Per iniziare, abbiamo misurato i tempi di elaborazione per ogni metrica nel sistema di raccomandazione dei giochi: utilizzando la libreria Python **Time**, è stato possibile determinare il tempo impiegato per generare una raccomandazione in base alla metrica selezionata.

Il secondo punto di questo studio è stato un confronto diretto delle caratteristiche delle 3 metriche.

Coefficiente di Pearson

Il coefficiente di Pearson è un metodo statistico utilizzato per analizzare una potenziale relazione lineare tra due variabili quantitative continue. Questo indice valuta sia l'intensità che il verso di un'associazione lineare.

I suoi valori possono variare da -1 a +1. Un risultato pari a +1 o -1 indica una correlazione lineare perfetta, rispettivamente positiva (crescita concordante) o negativa (relazione inversa).

```
Calcolo raccomandazioni con correlazione di Pearson...
Risultati con correlazione di Pearson:
1. Counter-Strike: Source (score: 0.9276)
2. Counter-Strike: Condition Zero (score: 0.8734)
3. Counter-Strike: Global Offensive (score: 0.7764)
4. Ricochet (score: 0.7339)
5. Deathmatch Classic (score: 0.7196)

Tempo di esecuzione: 57.7950 secondi
```

Figura 2.7 Risultati con correlazione di Pearson con Score annesso al gioco

Distanza Euclidea

La **distanza euclidea** rappresenta la misura della lunghezza del segmento che congiunge due punti in uno spazio. Impiegando questa metrica, lo spazio euclideo assume le proprietà di uno spazio metrico. Gli algoritmi basati su questa distanza privilegiano gli elementi con valori più piccoli, poiché ciò facilita l'identificazione di somiglianze tra di essi.

```
Calcolo raccomandazioni con distanza euclidea...
Risultati con distanza euclidea:
1. Counter-Strike: Source (score: 0.7243)
2. Counter-Strike: Condition Zero (score: 0.6653)
3. Counter-Strike: Global Offensive (score: 0.5993)
4. Ricochet (score: 0.5782)
5. Deathmatch Classic (score: 0.5718)

Tempo di esecuzione: 26.0466 secondi
```

Figura 2.8 Risultati con distanza Euclidea con Score annesso al gioco

Similarità del coseno

La similarità del coseno misura il coseno dell'angolo formato da due vettori ed è ampiamente impiegata nell'analisi testuale e documentale. Questo metodo permette di valutare la somiglianza tra documenti indipendentemente dalla loro lunghezza. Spesso, viene applicata insieme a tecniche come il TF-IDF, che trasformano i testi in vettori numerici, assegnando a ogni parola un punteggio specifico e rappresentandola lungo un asse distinto. La similarità del coseno, quindi, confronta questi vettori per stabilire quanto i documenti siano affini.

Questa metrica è particolarmente diffusa nell'ambito dell'NLP (Natural Language Processing) perché offre un'efficienza computazionale elevata quando si lavora con dati sparsi. Tuttavia, nel nostro caso, un'analisi preliminare ha rivelato che il dataset utilizzato presenta un basso livello di sparsità. Questa osservazione potrebbe giustificare i tempi di elaborazione più lunghi riscontrati nell'applicazione della similarità del coseno.

```
Calcolo raccomandazioni con similarità del coseno...
Risultati con similarità del coseno:
1. Counter-Strike: Source (score: 0.9276)
2. Counter-Strike: Condition Zero (score: 0.8734)
3. Counter-Strike: Global Offensive (score: 0.7764)
4. Ricochet (score: 0.7339)
5. Deathmatch Classic (score: 0.7196)

Tempo di esecuzione: 84.7038 secondi
```

Figura 2.9 Risultati della similarità del coseno con Score annesso al gioco

```
Sparsità del dataset: 9.31 %
```

Figura 2.10 Risultati della sparsità del dataset

Differenza tra coseno e Pearson

La similarità del coseno e la correlazione di Pearson possono essere visti come due varianti del prodotto interno (dot product).

Avendo due vettori x e y e vogliamo misurare la similarità tra loro, quella più semplice disponibile è il prodotto interno.

$$Inner(x, y) = \sum_i x_i y_i = \langle x, y \rangle$$

Figura 2.11 Formula prodotto interno

Se il vettore ***x*** tende ad essere alto dove anche ***y*** lo è e basso dove anche ***y*** lo è, il prodotto interno sarà alto a suo modo, e i vettori saranno quindi più simili tra loro.

Il prodotto interno si presenta senza limiti; un modo per poterlo limitare tra -1 e 1 è quello di dividerlo per le norme L2 dei vettori, ottenendo così la similarità del coseno.

$$CosSim(x, y) = \frac{\sum_i x_i y_i}{\sqrt{\sum_i x_i^2}\sqrt{\sum_i y_i^2}} = \frac{\langle x, y \rangle}{||x||\ ||y||}$$

Figura 2.12 Formula similarità del coseno

La similarità del coseno non è invariante ai cambiamenti. Se ***x*** venisse cambiato in ***x+1***, di conseguenza anche la similarità cambierebbe. Invece ciò che si presenta invariante è la correlazione di Pearson.

$$
\begin{aligned}
Corr(x, y) &= \frac{\sum_i (x_i - x)(y_i - y)}{\sqrt{\sum (x_i - \bar{x})^2} \sqrt{\sum (y_i - \bar{y})^2}} \\
&= \frac{\langle x - x,\ y - y \rangle}{||x - x||\ ||y - y||} \\
&= CosSim(x - \bar{x}, y - \bar{y})
\end{aligned}
$$

Figura 2.12 Formula correlazione di Pearson

La correlazione può essere definita come la somiglianza del coseno tra i vettori x e y dopo che sono stati centrati (cioè sottraendo la loro media), con valori compresi tra -1 e 1. A differenza della similarità del coseno standard, la correlazione rimane invariata sia quando i vettori vengono scalati (moltiplicati per una costante) sia quando vengono traslati (aggiunta di una costante).

Differenza tra coseno e distanza euclidea

Rispetto alla similarità del coseno, la distanza euclidea è meno utilizzata nelle applicazioni di NLP. È più adatta per variabili numeriche continue, ma ha alcuni svantaggi: non è invariante rispetto alla scala, quindi è generalmente consigliabile normalizzare i dati prima del calcolo. Inoltre, amplifica l'impatto di eventuali informazioni ridondanti presenti nel dataset.

Metriche	**Contesto NLP**	**Tipi di variabili**	**Invariante alle modifiche**	**Range di valori**	**Tempo di computazione del codice**
Coseno	Si	Vettori non vuoti con N variabili	no	[-1,1] o [0,1]	118 s.
Pearson	//	continue	Si	[-1,1]	50 s.
Euclidea	No	numeriche continue	No	[0,R]	25 s.

2.3 CLASSIFICAZIONE

Oltre al sistema di raccomandazione, abbiamo introdotto una classificazione degli elementi del dataset basata su una nuova categoria chiamata "star", da noi ideata sfruttando criteri preesistenti relativi alle valutazioni dei videogiochi.

```
# Calcola il rating a stelle basato sul rapporto recensioni negative/positive
steam_data['star'] = (steam_data['negative_ratings'] / steam_data['positive_ratings']) * 100
steam_data.loc[(steam_data['star'] >= 0) & (steam_data['star'] <= 12.5), 'star'] = 5
steam_data.loc[(steam_data['star'] > 12.5) & (steam_data['star'] <= 25), 'star'] = 4
steam_data.loc[(steam_data['star'] > 25) & (steam_data['star'] <= 37.5), 'star'] = 3
steam_data.loc[(steam_data['star'] > 37.5) & (steam_data['star'] <= 50), 'star'] = 2
steam_data.loc[(steam_data['star'] > 50), 'star'] = 1
```

Figura 2.13 Calcolo del rating a stella

Il dataset è stato suddiviso in 5 intervalli predefiniti per classificare i dati esistenti.

Successivamente, il modello selezionato è stato valutato attraverso diverse analisi, seguite da una fase di ottimizzazione e affinamento delle prestazioni.

Per quanto riguarda la scelta dell'algoritmo di classificazione, abbiamo selezionato il **KNeighborsClassifier**, disponibile nella libreria **Scikit-learn (SKLearn).**

Modello

Il **k-nearest neighbors** (K-NN) è un metodo impiegato nell'analisi dei pattern per assegnare una categoria a un oggetto, prendendo in considerazione le proprietà degli elementi più vicini ad esso. In questo algoritmo, i dati di input sono rappresentati dai *k* campioni di addestramento più prossimi nello spazio delle feature. Nonostante possa essere applicato sia a problemi di regressione che di classificazione, viene principalmente adottato per quest'ultima, fondandosi sull'ipotesi che elementi simili tendano a raggrupparsi insieme. Il risultato della classificazione è determinato dalla classe più frequente tra i *k* vicini più prossimi: l'oggetto analizzato viene assegnato alla categoria maggioritaria tra quelli circostanti. Il valore di *k* è un intero positivo, solitamente piccolo. Nel caso particolare in cui *k* = 1, l'oggetto assume direttamente la classe dell'unico elemento a esso più vicino.

Per la scelta del modello ci siamo affidati ad una mappa presa dal sito della libreria **Sklearn**, la quale mostra un percorso nella ricerca del miglior estimatore di machine learning in base ai propri scopi:

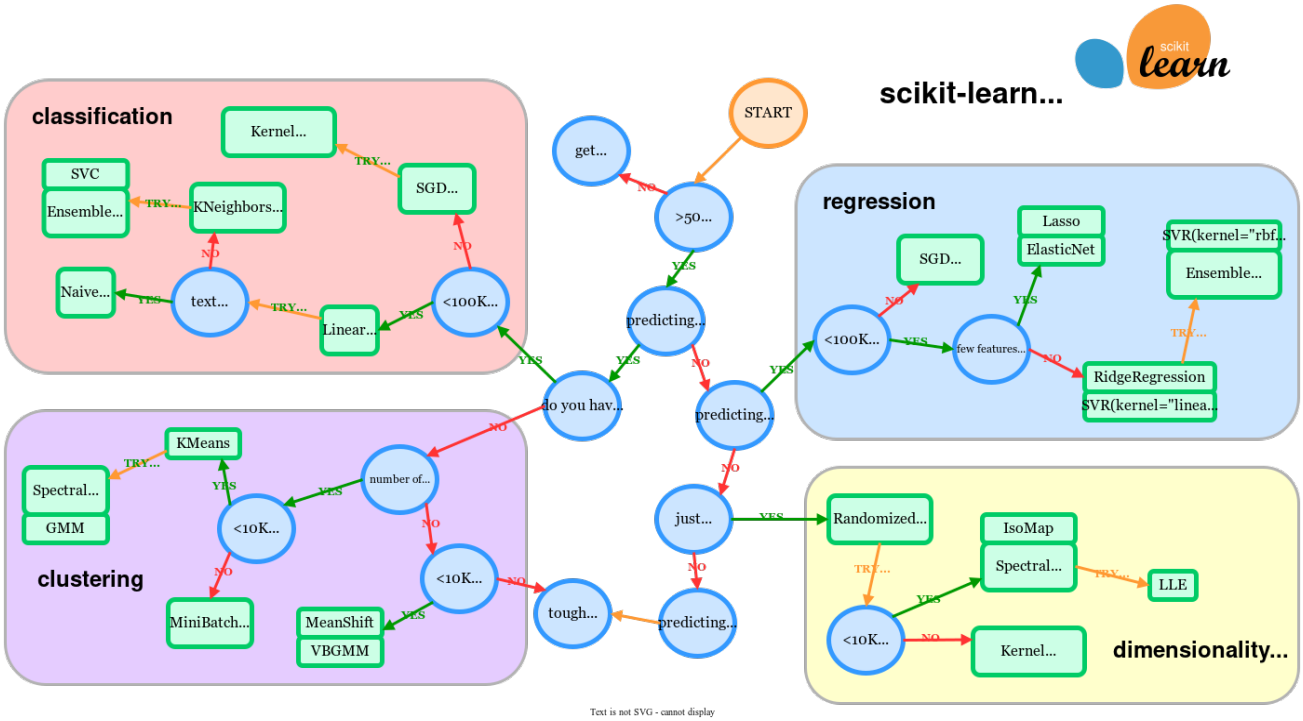

Figura 2.14 Mappa per la ricerca del miglior estimatore

Si può osservare che, analizzando i passaggi necessari per implementare il KNN, emergono le caratteristiche principali del nostro progetto: la previsione di una classe specifica (star), l'impiego di dati già classificati e l'utilizzo di informazioni non testuali.

Dataset

Per preparare un dataset adatto all'uso del modello selezionato e alla classificazione, i dati sono stati divisi in due gruppi: training e test. Il set di training (o di addestramento) servirà per insegnare al modello, mentre la parte restante (test) sarà utilizzata per valutarne le prestazioni. Nello specifico, abbiamo optato per una ripartizione dell'80% dei dati per il training e del 20% per il test.

Un altro aspetto significativo consiste nella suddivisione casuale degli elementi del dataset nelle due parti. Questo approccio garantisce che i due sotto-dataset risultanti siano rappresentativi dell'insieme originale e presentino una distribuzione equilibrata. Per ottenere questo risultato, è possibile specificare un valore intero qualsiasi nel parametro "random-state", poiché non si tratta di un parametro da ottimizzare (hyperparameter tuning).

Infine, in alcuni problemi di classificazione, le categorie possono avere un numero diseguale di campioni. Per questo motivo, è utile dividere il dataset in training set e test set mantenendo la stessa proporzione di esempi per ogni classe presente nel dataset iniziale. Questa tecnica è

nota come *split stratificato*. Per applicarla, è possibile impostare il parametro "stratify" con la variabile target "y" del dataset originale.

```
print("\nSplit train-test...")
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=1, stratify=y)
```

Figura 2.15 Split stratificato usato nel nostro codice

Pre-processing dei dati

Abbiamo proceduto alla standardizzazione dei dati per uniformare l'intervallo di variazione delle feature presenti nel dataset. Questa operazione ha permesso di ottimizzare l'affidabilità degli output ottenuti: la normalizzazione, infatti, accelera la convergenza dell'algoritmo di machine learning verso la soluzione definitiva.

```
print("\nNormalizzazione dati...")
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
predict_data = scaler.transform(predict_data)
```

Figura 2.16 Normalizzazione dei dati

Il modello viene poi allenato sulla porzione di training utilizzando i parametri predefiniti, sfruttando le funzioni messe a disposizione dalla libreria SKlearn.

```
print('\n\n=== FASE 1: Modello con parametri di base ===')
knn = KNeighborsClassifier(n_neighbors=5)

print("\nAddestramento modello base...")
knn.fit(X_train, y_train)
```

Figura 2.17 Addestramento modello base

Miglioramento del modello e Hyperparameters Tuning

Dopo aver valutato il modello iniziale con i parametri predefiniti, abbiamo esplorato diverse configurazioni per generare previsioni sulla categoria "star". Successivamente, abbiamo analizzato le performance del modello per verificarne l'accuratezza predittiva.

```
=== FASE 1: Modello con parametri di base ===

Addestramento modello base...

Predizioni test:
Predetti: [5. 5. 1. 2. 1.]
Effettivi: [1. 1. 2. 4. 4.]

Valutazione modello base...

Valutazione modello in corso...
Classification report:
              precision    recall  f1-score   support

         1.0       0.40      0.61      0.48      1818
         2.0       0.16      0.08      0.11       557
         3.0       0.15      0.10      0.12       650
         4.0       0.24      0.18      0.20       964
         5.0       0.34      0.29      0.31      1426

    accuracy                           0.33      5415
   macro avg       0.26      0.25      0.24      5415
weighted avg       0.30      0.33      0.30      5415

ROC score:  0.5502741408667262

La nostra accuratezza è bassa, procediamo a migliorare gli iperparametri
```

Figura 2.18 Modello con parametri

Per valutare le prestazioni del modello, abbiamo adottato il "Classification report" di sklearn.metrics, che ci ha permesso di considerare diverse metriche di valutazione. Inoltre, abbiamo calcolato il punteggio ROC AUC, ottenendo un valore di 0.55. L'ROC AUC (Area Under the Receiver Operating Characteristic Curve) rappresenta l'area totale sotto la curva ROC, fornendo una valutazione complessiva delle prestazioni del modello di classificazione considerando tutte le possibili soglie decisionali. Questo indicatore ci offre una misura aggregata dell'efficacia del modello.

Successivamente, abbiamo ottimizzato il nostro modello implementando una ricerca degli iperparametri più adatti, al fine di incrementare il ROC score, l'accuratezza e gli altri indicatori prestazionali.

Abbiamo scelto di impiegare una RandomizedSearchCV per l'ottimizzazione degli iperparametri, integrando anche una validazione incrociata di tipo RepeatedKFold.

Randomized Search

La **Randomized Search** è un metodo in cui si testano diverse combinazioni casuali di iperparametri per individuare la configurazione ottimale per il modello di classificazione. Rispetto alla Grid Search, questa tecnica è più rapida perché seleziona i parametri in modo aleatorio, ed è anche più efficiente, poiché spesso trova combinazioni di parametri significativamente migliori rispetto ad altri approcci.

```
# Configura la cross-validation con 10 fold ripetuti 3 volte
cvFold = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)
randomSearch = RandomizedSearchCV(estimator=knn, cv=cvFold, param_distributions=hyperparameters, n_jobs=-1, verbose=1)
```

Figura 2.19 Randomized Search

Una volta deciso quale approccio 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.

```
def HyperparametersSearch(X_train, X_test, y_train, y_test):
    #Esegue una ricerca degli iperparametri ottimali per il modello KNN.
    print("\nInizio HyperparametersSearch...")
    result = {}
    n_neighbors = list(range(1,30))
    weights = ['uniform', 'distance']
    metric = ['euclidean', 'manhattan', 'hamming']
```

Figura 2.20 Ricerca iperparametri attuali

L'esperimento è stato replicato per 3 iterazioni, durante le quali tutti i parametri e i relativi valori del ROC score sono stati memorizzati in un apposito dizionario. Successivamente, il dizionario è stato ordinato in base al ROC score per identificare la combinazione di parametri ottimale per ogni elaborazione.

Dopo aver determinato la migliore configurazione, il modello è stato riconfigurato con questi parametri e utilizzato per generare previsioni sulla categoria "star".

```
=== MIGLIORI IPERPARAMETRI TROVATI ===
n_neighbors: 23
metric: manhattan
weights: uniform
ROC score: 0.5778312816492143
```

Figura 2.21 Risultato migliori iperparametri trovati

Il risultato del ROC score risulta effettivamente incrementato, da 0.55 a 0.57.

Predizione su nuovi dati

Dopo aver ottimizzato i parametri del nostro modello, abbiamo testato le sue prestazioni generando previsioni sui giochi consigliati all'utente, concentrandoci in particolare sulla categoria principale.

```
Modello ottimizzato pronto per le raccomandazioni

Predizione su nuovi dati...

=== RISULTATI FINALI ===

Ecco a te i 5 giochi più simili a quello proposto:
                              name                  genres                      developer  price  star  star_prediction
10           Counter-Strike: Source   Action;FPS;Multiplayer                         Valve   7.19   5.0              5.0
7     Counter-Strike: Condition Zero   Action;FPS;Multiplayer                         Valve   7.19   5.0              4.0
25  Counter-Strike: Global Offensive  FPS;Multiplayer;Shooter  Valve;Hidden Path Entertainment   0.00   4.0              5.0
5                          Ricochet   Action;FPS;Multiplayer                         Valve   3.99   4.0              4.0
3                Deathmatch Classic   Action;FPS;Multiplayer                         Valve   3.99   4.0              4.0
```

Figura 2.22 Risultati finali

Possiamo notare che le predizioni, pur non essendo perfette al 100% rispetto ai risultati ottenuti precedentemente, abbiano restituito dei miglioramenti.

Conclusione

Anche se le previsioni sono state realizzate sulla categoria "star", inizialmente era disponibile un'altra opzione, cioè usare la categoria già esistente "english", che assegnava a ogni videogioco un valore binario (0 o 1). Tuttavia, abbiamo preferito utilizzare la prima categoria perché, durante la fase di valutazione del modello, la categoria "english" mostrava un'accuratezza iniziale estremamente elevata (98%), con previsioni quasi perfette. Di conseguenza, qualsiasi tentativo di ottimizzazione o ricerca di nuovi parametri non avrebbe portato a miglioramenti significativi, dato che i risultati erano già ottimali. Per questo motivo, abbiamo scelto di lavorare con la categoria "star", in modo da dimostrare un reale progresso e confermare l'utilità del nostro lavoro.

3. KNOWLEDGE BASE

Una **knowledge base** (o base di conoscenza) è un archivio strutturato che fornisce assistenza all'utente grazie ai dati e alle informazioni in esso contenuti. Rappresenta conoscenze sul mondo, permette di elaborare ragionamenti su tali conoscenze e si avvale di regole e principi logici per derivare nuove informazioni o individuare eventuali contraddizioni. In sostanza, una *KB* può essere definita come un insieme di assiomi, ovvero affermazioni considerate vere, che creano un ambiente finalizzato a semplificare la raccolta, la gestione e la condivisione della conoscenza.

Nel contesto del nostro caso di studio, la **knowledge base** viene impiegata per permettere all'utente di formulare domande specifiche relative al dominio di interesse, ottenendo risposte pertinenti e approfondite.

3.1 GESTIONE DELLE KNOWLEDGE BASE

Per la creazione e gestione della knowledge base, abbiamo adottato l'estensione **Pytholog** di Python, che si fonda sui principi del linguaggio **logico Prolog**. Partendo dalla struttura del Prolog, abbiamo prima proceduto a inserire i fatti nella knowledge base in modo automatico, estraendo direttamente le informazioni necessarie dal dataset. Questo approccio garantisce che, in caso di aggiornamenti del dataset, anche la KB venga modificata di conseguenza senza interventi manuali. Successivamente, abbiamo definito le regole logiche, tenendo conto sia della gestione dei fatti presenti nella KB sia delle possibili modalità di interazione con l'utente.

3.1.1 FATTI

I fatti costituiscono gli assiomi invariabili della knowledge base e rappresentano il fondamento su cui si basano le regole. Esistono sei diverse tipologie di fatti, ciascuna delle quali esprime una relazione tra i giochi e le relative caratteristiche.

1) “`developer (name, developer)`”

Rappresenta la relazione tra un gioco e chi lo ha sviluppato.

es. `developer(counter-strike, valve)`

2) “publisher(name, publisher)”

Rappresenta la relazione tra un gioco e chi lo ha rilasciato.

es. publisher(counter-strike, valve)

3) “prices(name, price)”

Rappresenta la relazione tra un gioco e il suo prezzo.

es. prices(counter-strike, 7.19)

4) “stars(name, star)”

Rappresenta la relazione tra un gioco e l’apprezzamento da parte degli utenti. ‘star’ è il valore in percentuale del rapporto tra i rating negativi e i rating positivi, successivamente convertiti in dei numeri interi compresi tra 1 e 5 in base al risultato del rapporto.

```
steam_data['star'] = (steam_data['negative_ratings'] / steam_data['positive_ratings']
steam_data.loc[(steam_data['star'] >= 0) & (steam_data['star'] <= 12.5), ['star']] =
steam_data.loc[(steam_data['star'] > 12.5) & (steam_data['star'] <= 25), ['star']] =
steam_data.loc[(steam_data['star'] > 25) & (steam_data['star'] <= 37.5), ['star']] =
steam_data.loc[(steam_data['star'] > 37.5) & (steam_data['star'] <= 50), ['star']] =
steam_data.loc[(steam_data['star'] > 50), ['star']] = 1
```

Figura 3.1 Calcolo del rating a stella

es. stars (counter-strike, 5)

5) “genre(name, steamspy_tags)”

Rappresenta la relazione tra un gioco e il suo genere.

es. genre(counter-strike, action;fps;multiplayer)

6) “english(name, english)”

Rappresenta la relazione tra un gioco e la presenza o meno della lingua inglese. I valori “0” e “1” sono stati rispettivamente convertiti nelle stringhe “no” e “yes”.

```
steam_data['english'] = steam_data['english'].replace({0: 'no', 1: 'yes'})
```

Figura 3.2 Relazione tra gioco e lingua inglese

es. english(counter-strike, yes)

3.1.2 REGOLE

Le **regole** costituiscono il cuore dell'interazione con l'utente, permettendogli di formulare domande alla knowledge base senza dover conoscere la sintassi specifica. Grazie ai fatti presenti nella KB, le regole consentono di recuperare e mostrare all'utente le informazioni richieste.

Le regole presenti nella KB sono le seguenti, e per ognuna è riportato un esempio:

1) "`has_price(X, Y) :- prices(Y, X)`"

Permette di ottenere informazioni sul prezzo di un gioco o su giochi di un determinato prezzo.

es. "`has_price(X, counter-strike)`" -> `X = 7.19`

2) "`quality(X, Y) :- stars(Y, X)`"

Permette di ottenere informazioni sulla qualità di un gioco, basandosi sulle stelle.

es. "`quality(X, counter-strike)`" -> `X = 5`

3) "`developed_by(X, Y) :- developer(Y, X)`"

Permette di ottenere informazioni su chi ha sviluppato un gioco.

es. "`developed_by(X, counter-strike)`" -> `X = valve`

4) "`released_by(X, Y) :- publisher(Y, X)`"

Permette di ottenere informazioni su chi ha rilasciato un gioco.

es. "`released_by(X, counter-strike)`" -> `X = valve`

5) "`is_genre(X, Y) :- genre(Y, X)`"

Permette di ottenere informazioni sul genere di un gioco.

es. "`is_genre(X, counter-strike)`" -> `X = action;fps;multiplayer`

6) “has_english(X, Y) :- english(Y, X)”

Permette di chiedere se il gioco presenta la lingua inglese o meno.

es. “has_english(X, counter-strike)” -> X = yes

7) “quality_check(X, Y, T, Z) :- stars(X, T), stars(Y, Z)”

Permette di ottenere informazioni sulla qualità di due giochi diversi, mettendoli a confronto basandosi sulle stelle.

es. “quality_check(team fortress classic, counter-strike, T, Z)” -> T = 4, Z = 5

3.1.3 LIKING PROBABILITY

Abbiamo sviluppato fatti e regole in grado di interagire tra loro, offrendo all'utente la possibilità di calcolare la probabilità che un nuovo gioco possa piacergli, basandosi sulle informazioni presenti nella knowledge base.

Stress

1) “has_work(utente, ore_di_lavoro)”

Fatto che permette di controllare le ore di lavoro dell’utente.

2) “stress(utente, Prob1) :- has_work(Persona, Prob2), Prob1 is Prob2”

Regola che permette di calcolare l’indice di stress dell’utente

Questi due fattori permettono di determinare, in funzione dello stress e delle ore di lavoro dell’utente, la probabilità che un nuovo gioco possa piacergli.

Gioco piaciuto

1) “liked_game(utente, gioco_piaciuto)”

Fatto che serve a controllare se un gioco piace all’utente.

Average Playtime

1) "`avg_playtime(name, average_playtime)`"

Fatto che rappresenta la relazione tra un gioco e l'avg_playtime di quel gioco.

2) "`has_avg_playtime(X, Y) :- avg_playtime(X, Y)`"

Regola che permette di risalire all'avg_playtime di un gioco.

3) "`avg_playtime_comp(Gioco_Piaciuto, Gioco_Nuovo, Y) :-`
`liked_game(utente, Gioco_Piaciuto),`
`has_avg_playtime(Gioco_Piaciuto, Avg1),`
`has_avg_playtime(Gioco_Nuovo, Avg2), Y is Avg1 – Avg2`"

Regola che fa la differenza tra l'avg_playtime del gioco che già piace all'utente e del gioco nuovo.

Il valore ottenuto da ***avg_playtime_comp*** viene confrontato con specifici intervalli che indicano il grado di somiglianza con ***avg_playtime***. Quanto più il valore si avvicina a zero, tanto maggiore è la somiglianza. Il valore corrispondente all'intervallo viene quindi memorizzato nella variabile ***Playtime_Similarity*.**

Genre

1) "`same_genre(genre, genre)`"

Fatto che permette di controllare se il genere di due giochi è lo stesso. Se il risultato della query è **Yes**, il fatto esiste e quindi è uguale.

2) "`has_same_genre(Gioco1, Gioco2,) :- is_genre(Gioco1, X), is_genre(Gioco2, Y), same_genre(X, Y)`"

Regola che controlla se due giochi hanno lo stesso genere. Alla conclusione di una query su questa regola, viene generata la variabile *Genre_Similarity*, assumendo il valore di 2.5 se il risultato è *Yes* e -2.5 se il risultato è ***No***.

Calcolo della probabilità

1) "`compatibility(Num1, Num2, S1) :- S1 is Num1 + Num2`"

Regola che somma i valori di Similarity di avg_playtime e genre.

2) "`to_like(utente, Num1, Num2, Probabilità) :- stress(utente, Prob1), compatibility(Num1, Num2, Score),`

Probabilità is Prob1 + Score" Regola che calcola la probabilità che un gioco nuovo possa piacere all'utente in base agli score di similarità e allo stress.

3.2 INTERAZIONE CON L'UTENTE

Durante l'esecuzione del programma, la seconda opzione permetterà all'utente di interagire con la knowledge base e porre domande riferite ai dati dei giochi.

```
KNOWLEDGE BASE

Benvenuto, qui puoi eseguire ricerche sui giochi e sulle loro caratteristiche

Ecco le ricerche che puoi eseguire:
1) Ricerche sulle caratteristiche di un gioco
2) Confronti e ricerca di giochi in base ad una caratteristica
3) Verificare delle caratteristiche
4) Vedere con quale probabilità ti possa piacere un nuovo gioco
5) Exit Knowledge Base

Quale scegli (inserisci il numero corrispondente alla tua scelta)?
```

Figura 3.3 Schermata di benvenuto KB

Qui l’utente potrà scegliere delle domande da porre alla KB:

1)Le ricerche sulle caratteristiche di un gioco fornito in input dall’utente consentono, attraverso le regole, di ottenere l’informazione desiderata.

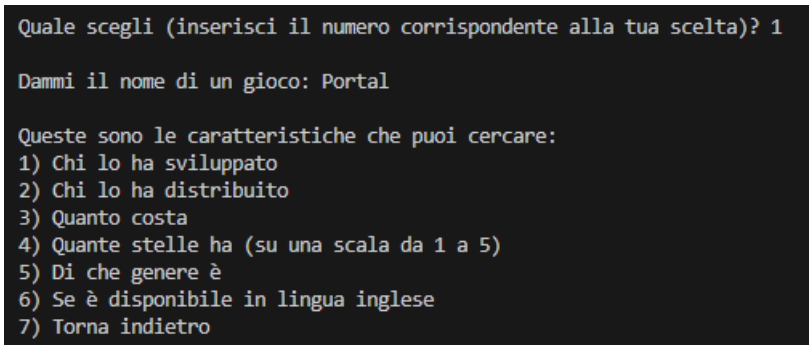

Figura 3.4 Menu mostrato alla scelta di ricerca 1

L'utente seleziona il tipo di informazione desiderata dalla knowledge base scegliendo un numero da 1 a 6. Al termine della richiesta, può effettuare una nuova ricerca nello stesso ambito o tornare al menu principale di interazione con la KB.

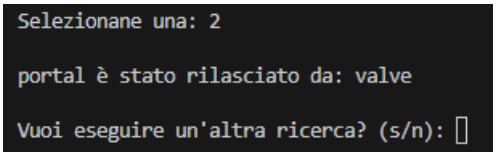

Figura 3.5 Risultato dell'operazione richiesta

2)L'utente può confrontare due giochi scelti in base alla loro qualità e, inoltre, può cercare giochi in base a una specifica caratteristica.

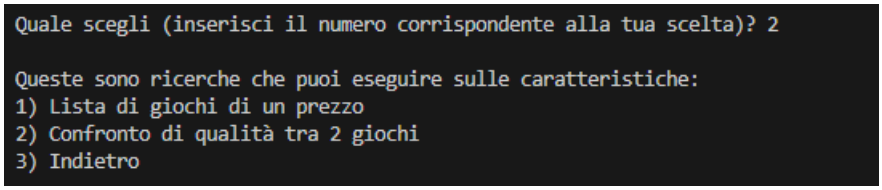

Figura 3.6 Menu mostrato alla scelta di ricerca 2

In questa sezione, l'utente può scegliere il tipo di ricerca approfondita da esplorare, selezionando un numero tra 1 e 2 per effettuare le ricerche, oppure 3 per tornare indietro.

```
Selezionane una: 2

Dimmi il nome del primo gioco: Counter-Strike
Dimmi il nome del secondo gioco: Portal

I due giochi hanno la stessa qualità: 5 stelle
```

Figura 3.7 Risultato dell'operazione richiesta

3) L'utente viene invitato a inserire manualmente il tipo di informazione che desidera verificare, corrispondente alle diverse tipologie di fatti presenti nella KB.

```
Quale scegli (inserisci il numero corrispondente alla tua scelta)? 3

Questi sono le caratteristiche che puoi verificare:
1) developer
2) publisher
3) prices
4) stars
5) genre
6) english
7) Indietro

Selezionane una:
```

Figura 3.8 inserimento dei dati da parte dell'utente e risultato dell'operazione richiesta

L'utente deve inserire il nome del gioco e l'informazione relativa al tipo di verifica che desidera eseguire. Successivamente, ha la possibilità di ripetere l'interazione o di tornare al menu principale di interazione con la KB.

```
Quale scegli (inserisci il numero corrispondente alla tua scelta)? 3

Questi sono le caratteristiche che puoi verificare:
1) developer
2) publisher
3) prices
4) stars
5) genre
6) english
7) Indietro

Selezionane una: 1

Quale gioco vuoi controllare? Counter-Strike
Inserisci un dato corrispondente alla caratteristica scelta: Valve

Risultato: Sì
```

Figura 3.9 Verifica delle caratteristiche: inserimento e risposta

4) Calcolo della probabilità che un gioco possa piacere:

All'utente viene chiesto di inserire il nome di un gioco che gli è piaciuto e un nuovo gioco per cui desidera conoscere la probabilità che gli piaccia, insieme alle informazioni relative a questi giochi e al numero di ore al giorno dedicate al lavoro o allo studio.

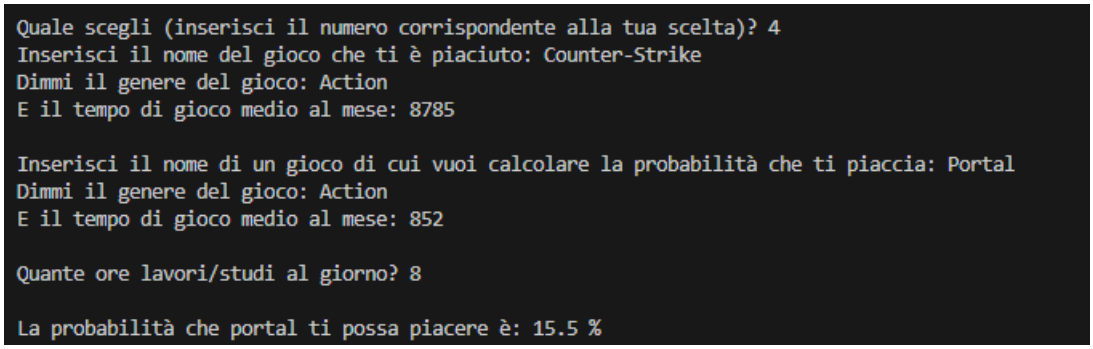

Figure 3.10 Risultati della probabilità di gradimento del gioco

4. ONTOLOGIA

In ambito informatico, un'**ontologia** è un sistema strutturato che formalizza la rappresentazione della realtà e del sapere. Si tratta di un modello organizzato che permette di definire gli elementi (come oggetti o idee) e i loro collegamenti all'interno di uno specifico campo del conoscibile.

4.1 INTRODUZIONE

Il modulo Ontologia è progettato per esplorare e interrogare l'ontologia Steam, che formalizza le relazioni tra i concetti principali del dominio dei videogiochi sulla piattaforma Steam. L'ontologia è stata sviluppata utilizzando la libreria owlready2 in Python, che permette di caricare, visualizzare e interrogare ontologie in formato OWL.

4.2 FUNZIONALITÀ PRINCIPALI

Il modulo offre le seguenti funzionalità:

1. **Visualizzazione delle Classi**: Mostra tutte le classi presenti nell'ontologia, come Agent, Game, Developer, Genre, Platform, Publisher, Costumer, e Shop.
2. **Visualizzazione delle Proprietà**:

- **Proprietà d'oggetto**: Relazioni tra istanze delle classi (es. has_developer, has_genre).
- **Proprietà dei dati**: Attributi legati alle istanze (es. price, rating).

3. **Query d'esempio**: Esegue query predefinite per estrarre informazioni specifiche, come giochi di un determinato genere o sviluppati da una certa azienda.
4. **Interazione intuitiva**: Un menu testuale guida l'utente attraverso le opzioni disponibili.

4.3 STRUTTURA DEL CODICE

Il codice è organizzato in una funzione principale, main_ontology(), che gestisce l'interazione con l'utente e le operazioni sull'ontologia. Ecco i componenti principali:

- **Caricamento dell'ontologia**: L'ontologia viene caricata dal file Steam-Ontology.owl nella cartella dataset.
- **Menu principale**: Offre opzioni per esplorare classi, proprietà, query o uscire dal modulo.
- **Sottomenu per le classi**: Permette di visualizzare le istanze di ciascuna classe (es. tutti i giochi o gli sviluppatori).
- **Query predefinite**: Include esempi come "Giochi con genere 'Classic'" o "Giochi sviluppati da 'Valve'".

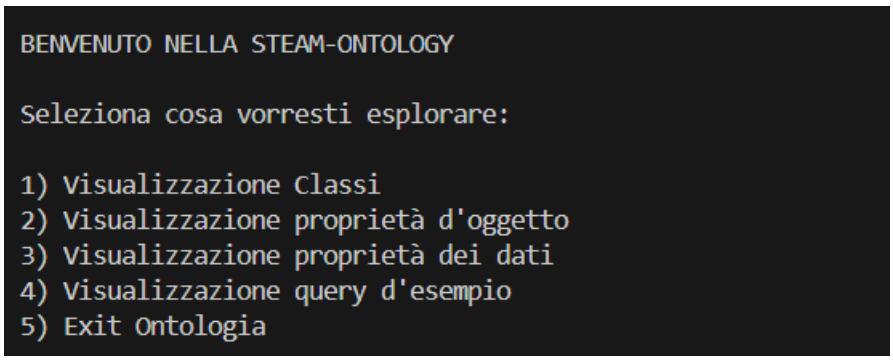

Figura 4.1 Menù ontology

4.4 ESEMPI DI UTILIZZO

1. **Visualizzazione delle Classi**:
   - L'utente seleziona l'opzione 1 dal menu principale.
   - Viene mostrato l'elenco delle classi. Successivamente, può esplorare le istanze di una classe specifica (es. Game o Developer).

```
Inserisci qui la tua scelta:    1

Classi presenti nell'ontologia:
Classi presenti nell'ontologia:

[Steam-Ontology.Agent, Steam-Ontology.Game, Steam-Ontology.Develo
per, Steam-Ontology.Genre, Steam-Ontology.Platform, Steam-Ontolog
y.Publisher, Steam-Ontology.Costumer, Steam-Ontology.Shop]

Vorresti esplorare meglio qualche classe in particolare?

1) Agent
2) Game
3) Developer
4) Genre
5) Platform
6) Publisher
7) Costumer
8) Shop
```

Figura 4.2 Classi presenti nell'ontologia

2. **Query d'esempio**:
    - L'utente seleziona l'opzione 4 per visualizzare query predefinite.

```
Inserisci qui la tua scelta:    4

Query d'esempio:

-Lista di Giochi che presentano la categoria 'Classic':

[Steam-Ontology.Grand_Theft_Auto, Steam-Ontology.Half_Life]


-Lista di Giochi che presentano lo sviluppatore 'Valve':

[Steam-Ontology.Half_Life, Steam-Ontology.Portal]


-Lista di Giochi che presentano la piattaforma 'Windows':

[Steam-Ontology.Call_of_Duty, Steam-Ontology.Grand_Theft_Auto, Steam-Ontology.Half_Life, Steam-Ontology.Portal, Steam-Ontology.Tomb_Raider:_Legend]
```

Figura 4.3 Query d'esempio presenti nell'ontologia

3**. Proprietà**:
    - Le opzioni 2 e 3 mostrano rispettivamente le proprietà d'oggetto e dei dati, utili per comprendere le relazioni e gli attributi nel dominio.

```
Inserisci qui la tua scelta:    2

Proprietà d'oggetto presenti nell'ontologia:

[Steam-Ontology.distributes, Steam-Ontology.is_distributed_by, St
eam-Ontology.has_developer, Steam-Ontology.has_genre, Steam-Ontol
ogy.has_platform, Steam-Ontology.has_publisher, Steam-Ontology.is
_searched_by, Steam-Ontology.searches, Steam-Ontology.is_situated
, Steam-Ontology.is_used_by, Steam-Ontology.uses, Steam-Ontology.
operates_in]
```

Figura 4.4 Proprietà d'oggetto presenti nell'ontologia

```
Inserisci qui la tua scelta:  3

Proprietà dei dati presenti nell'ontologia:

[Steam-Ontology.achievement, Steam-Ontology.average_playtime, Ste
am-Ontology.english, Steam-Ontology.id_Agent, Steam-Ontology.nega
tive_rating, Steam-Ontology.rating, Steam-Ontology.positive_ratin
g, Steam-Ontology.price, Steam-Ontology.release_date, Steam-Ontol
ogy.required_age, Steam-Ontology.username]
```

Figura 4.5 Proprietà dei dati presenti nell'ontologia

4.5 INTERAZIONE CON L'UTENTE

L'interfaccia è progettata per essere intuitiva:

- L'utente inserisce numeri corrispondenti alle opzioni desiderate.
- I messaggi di output sono chiari e strutturati, con esempi e spiegazioni contestuali.
- È possibile tornare indietro o continuare l'esplorazione in qualsiasi momento.

4.6 DETTAGLI TECNICI

- **Libreria utilizzata**: owlready2 per la gestione delle ontologie OWL.
- **Input**: File OWL (Steam-Ontology.owl).
- **Output**: Risultati delle query e visualizzazioni strutturate nel terminale.

4.8 CONCLUSIONI

Il modulo Ontologia fornisce uno strumento efficace per esplorare e interrogare la conoscenza strutturata del dominio Steam. La sua integrazione con gli altri moduli (come il Recommender System e la Knowledge Base) arricchisce l'esperienza utente, offrendo un accesso organizzato alle informazioni. Possibili estensioni includono l'aggiunta di nuove query o l'integrazione con interfacce grafiche per una navigazione più intuitiva.

5. BUDGET PLANNER

Il **Budget Planner Steam** è un modulo progettato per aiutare gli utenti a trovare giochi sulla piattaforma Steam che rientrino nel loro budget, tenendo conto delle loro preferenze personali. Questo strumento è particolarmente utile per gli utenti che desiderano gestire in modo intelligente le proprie spese sui videogiochi, ottenendo raccomandazioni personalizzate basate su budget, rating minimo, genere preferito e tempo di gioco massimo.

5.1 DATASET E LIBRERIE UTILIZZATE

Per questo modulo è stato utilizzato il dataset **Steam Store Games**, già sottoposto a un processo di data cleaning. Le principali librerie utilizzate sono:

- **Pandas**: Per la manipolazione e l'analisi dei dati.
- **Python Standard Library**: Per la gestione dei tipi di dati e l'interazione con l'utente.

5.2 FUNZIONALITA' PRINCIPALI

5.2.1 CARICAMENTO E PRE-ELABORAZIONE DEI DATI

Il modulo include una funzione _load_and_preprocess_data che si occupa di caricare il dataset da un file CSV. Successivamente, converte la colonna price in valori numerici, sostituendo quelli non validi con 0. La funzione calcola anche un rating a stelle compreso tra 1 e 5, basato sul rapporto tra recensioni positive e negative. Infine, mantiene solo le colonne rilevanti per le raccomandazioni, ovvero name, price, star, genres e average_playtime.

5.2.2 RACCOMANDAZIONE DI GIOCHI

La funzione **recommend** seleziona i giochi applicando i seguenti filtri:

Budget – Considera solo i giochi con un prezzo inferiore o uguale al budget indicato.

Rating minimo – Include esclusivamente i giochi che hanno un rating a stelle maggiore o uguale al valore specificato (il valore predefinito è 3.5).

Genere – Filtra i giochi in base al genere, se viene fornito.

Tempo di gioco massimo – Seleziona i giochi con un tempo di gioco medio inferiore o uguale al valore indicato.

I risultati ottenuti vengono ordinati in base al prezzo, dal più alto al più basso, e restituiti sotto forma di lista di dizionari, ciascuno contenente le informazioni relative ai giochi selezionati.

5.2.3 INTERFACCIA UTENTE INTERATTIVA

La funzione **interactive_planner** consente all'utente di inserire le proprie preferenze attraverso un'interfaccia a riga di comando. L'utente può specificare il budget disponibile, il rating minimo desiderato, il genere preferito (opzionale) e il tempo di gioco massimo (opzionale). Dopo l'inserimento dei dati, il modulo mostra una lista dei 10 giochi migliori che corrispondono ai criteri indicati. Per ogni gioco vengono forniti dettagli come nome, prezzo, rating, tempo di gioco medio e generi.

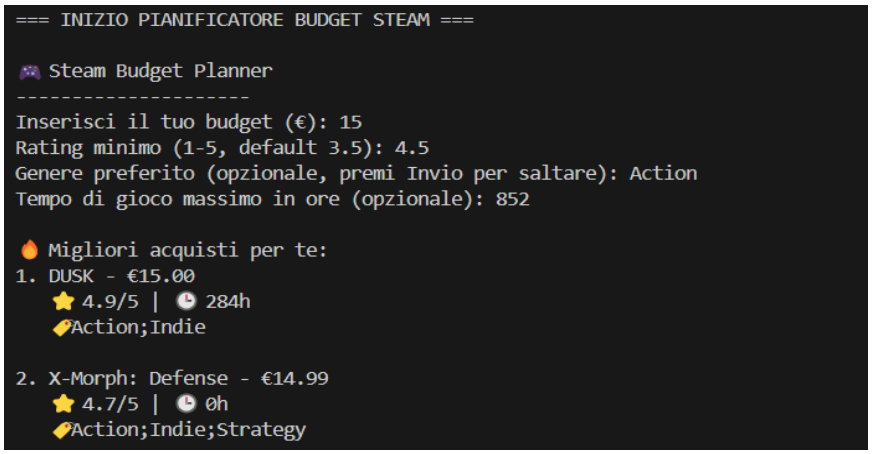

Figura 5.1 Risultato budget Planner

6. CONCLUSIONI

Delle possibili estensioni del progetto sono:

Interfaccia Utente Migliorata

Per un'esperienza più intuitiva e completa, l'interfaccia utente sarà potenziata attraverso una dashboard interattiva sviluppata con Flask/Dash o Streamlit, che permetterà di visualizzare graficamente raccomandazioni, confronti e trend. Saranno integrati filtri avanzati per consentire agli utenti di selezionare giochi in base a specifiche tecniche, come requisiti di sistema, modalità multiplayer o supporto VR. Inoltre, una visualizzazione grafica delle relazioni, realizzata tramite network graph, mostrerà le connessioni tra giochi, generi e sviluppatori, migliorando la scoperta di titoli correlati.

Analisi di Mercato

Benchmark prezzi: Confronta i prezzi dei giochi con altri store come Epic Games e GOG utilizzando tecniche di web scraping o API per ottenere dati aggiornati e competitivi.

6.1 LINK GITHUB

Il repository con tutti i file del progetto: https://github.com/paologiampietro/ICon24-25.git

La lista delle dipendenze del progetto è contenuta nel file *requirements.txt*, caricato nel repository.

Il lavoro è stato suddiviso in task assegnati ai due membri del gruppo.