Documentazione del caso di studio

Gruppo di lavoro:

- Saracino Lorenzo – 746459 - l.saracino20@studenti.uniba.it
- Rosmarino Fabrizio – 758447 – f.rosmarino@studenti.uniba.it

Repository github: https://github.com/s4rac1no/gamepy

AA 2023-2024

UNIVERSITÀ DEGLI STUDI DI BARI ALDO MORO

INDICE

INTRODUZIONE

Per lo sviluppo del progetto gamepy è stato utilizzato il linguaggio di programmazione Python, ponendo l'obiettivo principale sull'analisi dei videogame. I dati relativi sono stati raccolti dal sito Metacritic, il quale è un sito che aggrega recensioni dedicate a videogiochi, e non solo, anche film, musica e serie TV. Il dataset utilizzato per lo sviluppo è stato preso dal sito di Kaggle, si riporta il link: https://www.kaggle.com/datasets/brunovr/metacritic-videogames-data/data.
I dati sono stati raccolti, così come riportato dalla descrizione dell'autore del dataset, attraverso il "web scraping", il quale è un processo automatizzato di estrazione di informazioni da siti web. Questo è stato fatto attraverso uno script in Python utilizzando librerie come requests, bs4.BeautifulSoup, re and pandas. il quale ha raccolto i dati presenti nel csv.
Si riporta la struttura del dataset, in particolare si riportano feature e il loro significato:

- **name:** il nome del gioco.
- **platform:** la piattaforma disponibile per il gioco. (Playstation, Xbox, Nintendo..)
- **r-date:** la data di rilascio del gioco.
- **score:** media dei voti dati dai critici del sito metascore.
- **user score**: media dei voti dagli utenti nel sitoweb.
- **developer:** la casa produttrice del gioco.
- **genre:** il/i genere/generi del videogioco.
- **players:** modalità di gioco (1 player, up to 23...alcuni giochi potrebbero non avere questa informazione).
- **critics:** il numero di critiche che il gioco ha ricevuto.

Infine, tutti i dati nel dataset sono stati collezionati il 10 Novembre 2020, ed esso contiene giochi che vanno dal 1996 al 2020.

Gli obiettivi posti per questo progetto sono quelli di predire e determinare il successo dei videogiochi, analizzare le caratteristiche di questi e individuare

quali giochi potrebbero andare in tendenza e ritrovare ulteriori relazioni implicite presenti tra i dati e utilizzabili per produrre nuova conoscenza.

ELENCO DI ARGOMENTI DI INTERESSE

Gli argomenti che sono stati oggetto del corso di Ingegneria della Conoscenza e che sono stati riproposti e applicati in gamepy sono:

Apprendimento non supervisionato: è stato utilizzato il K-Means algoritmo di hard-clustering, la quale è una forma di clustering più rigida in cui ogni punto appartiene completamente a un cluster specifico. I cluster hanno fatto emergere particolari caratteristiche dove ad ognuno di esse si è attribuito un risultato. Infine, il K-Means ha fatto emerge alcune anomalie.

Apprendimento non supervisionato: sono state adottate diverse tecniche di machine learning supervisionato per affrontare compiti di classificazione, inoltre oltre alle varie tecniche di pre-processing e pulizia dei dati, si addestrano i modelli di machine learning per predire se un videogioco avrà successo, e si visualizzano i risultati delle predizioni attraverso metriche di valutazione. Gli algoritmi utilizzati sono stati K-Nearest Neighbors (KNN), Random Forest, Support Vector Machine (SVM), Decision Tree e Gradient Boosting Classifier.

Rappresentazione e ragionamento: in questa soluzione, viene costruita una base di conoscenza, integrando il Prolog in Python grazie alle librerie Pyswip. L'approccio utilizzato è stato quello di creare delle query le quali hanno consentito di inferire nuove informazioni a partite da quelle ricavabili in modo standard, arricchendo così la comprensione e l'analisi dell'obiettivo di interesse.

Risoluzione di un CSP: attraverso il Random Walk e il Simulated Annealing i quali sono entrambi algoritmi di ricerca e ottimizzazione che possono essere adattati per risolvere problemi di soddisfacimento dei vincoli, abbiamo creato un elenco variabile che rappresenta una playlist di giochi che rispetta determinati vincoli stabiliti.

Apprendimento non supervisionato

L'obiettivo è stato quello di esplorare e raggruppare i videogiochi in cluster omogenei basati su caratteristiche chiave come le valutazioni dei giocatori (score), il numero di giocatori e la piattaforma di gioco. Utilizzando l'algoritmo K-Means, abbiamo cercato di identificare pattern e segmentazioni all'interno dei dati per comprendere meglio il panorama dei videogiochi in base a queste caratteristiche.

Caricamento e Pulizia dei Dati

Inizialmente, abbiamo caricato i dati da un file CSV contenente informazioni dettagliate sui videogiochi, comprese le colonne per piattaforma, score, numero di giocatori e altre informazioni pertinenti. Abbiamo eseguito una pulizia preliminare eliminando le righe con valori mancanti nelle colonne cruciali come score.

Unificazione e filtraggio delle Piattaforme

Nel dataset originale, i giochi sono associati a diverse piattaforme di gioco. Tuttavia, alcune di queste piattaforme sono versioni diverse della stessa famiglia, come ad esempio Xbox One e Xbox 360, o PlayStation 2, PlayStation 3 e PlayStation 4. Quindi, per semplificare l'analisi e migliorare la coerenza nei dati, abbiamo deciso di raggruppare le diverse piattaforme di gioco in categorie più ampie:

- Le piattaforme Xbox One e Xbox 360 sono state unite sotto la categoria 'Xbox'.
- Le piattaforme PlayStation 2, PlayStation 3 e PlayStation 4 sono state unite sotto la categoria 'PlayStation'.
- Le piattaforme Wii, Wii U, Nintendo DS e Nintendo 3DS sono state raggruppate sotto la categoria 'Nintendo'.
- La piattaforma PC è rimasta come 'PC'.

Dopo l'unificazione, è stato necessario limitare l'analisi a un sottoinsieme di piattaforme di gioco principali per garantire che i dati fossero più gestibili e

focalizzati sulle piattaforme più comuni. Si è deciso quindi di restringere il tipo di piattaforme solo a quelle riportate precedentemente, raggruppate nelle rispettive categorie.

Selezione delle Caratteristiche

Dopo aver unificato le piattaforme, abbiamo selezionato le caratteristiche rilevanti per l'analisi:

- **Numeriche:**

a. score: Il punteggio assegnato al videogioco.
b. users: Il numero di utenti che hanno valutato il videogioco.

- **Categoriche:**

c. platform: La piattaforma su cui è stato pubblicato il videogioco, che include categorie unificate come 'Xbox', 'PlayStation', 'Nintendo' e 'PC'.

Queste caratteristiche sono state scelte perché offrono una base significativa per il clustering e l'analisi dei dati sui videogiochi. Le caratteristiche numeriche come il punteggio e il numero di utenti possono indicare la popolarità e la qualità del videogioco. Se un gioco possiede un ottimo score, ma è stato recensito da un numero molto basso di persone, lo score diventa poco affidabile, quindi sono stati considerate queste due caratteristiche, fortemente legate tra di loro mentre la caratteristica categorica della piattaforma consente di esaminare differenze e similitudini tra i giochi pubblicati su piattaforme diverse. Esistono giochi utilizzabili su più piattaforme o anche solo su alcune che possiedono però diversi livelli di score e quindi, un diverso livello di successo o di insuccesso.

Preprocessing dei Dati

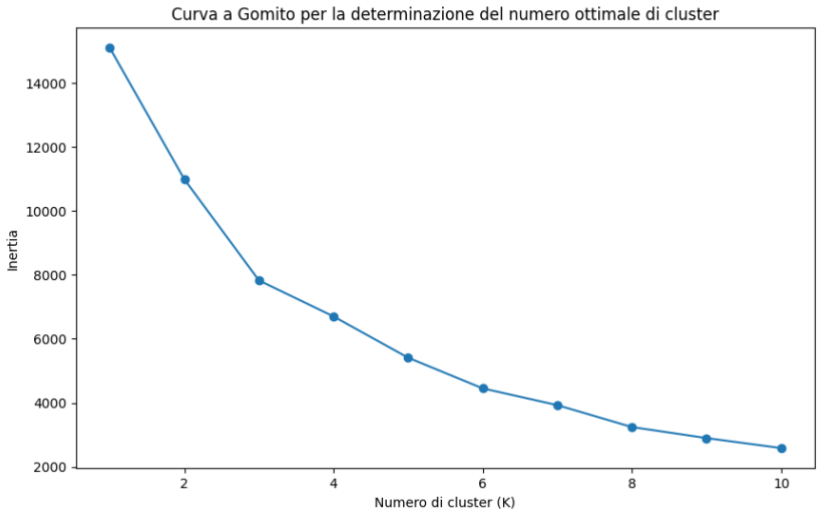
Abbiamo applicato il preprocessing dei dati per preparare le caratteristiche numeriche e categoriche per l'analisi di clustering:

Le caratteristiche numeriche 'score' e ‘users’ sono state standardizzate utilizzando StandardScaler per garantire che avessero la stessa scala e non dominassero il processo di clustering.

Le caratteristiche categoriche sono state trasformate utilizzando OneHotEncoder per gestire le variabili categoriche come la piattaforma di gioco, trasformandole in una forma numerica adatta all'analisi.

Determinazione del Numero Ottimale di Cluster (K)

Per identificare il numero ottimale di cluster da utilizzare con l'algoritmo K-Means, abbiamo eseguito un'analisi della curva a gomito. Questo ci ha permesso di valutare come l'inerzia (somma dei quadrati delle distanze dei punti dai centroidi) varia al variare del numero di cluster K. Sulla base di questa analisi, abbiamo scelto un valore ottimale di K che bilanciasse la complessità del modello con la sua capacità di identificare pattern significativi nei dati.



Clustering con K-means

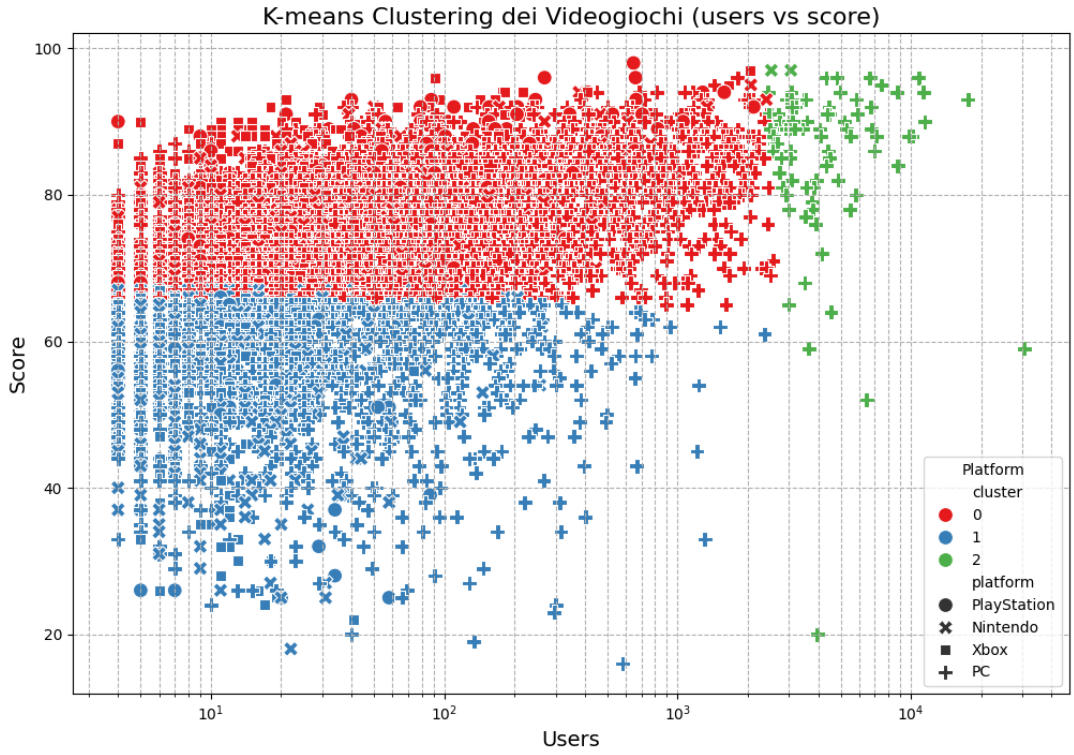
Utilizzando il numero ottimale di cluster identificato (3), abbiamo creato una pipeline che include il preprocessing dei dati e l'applicazione dell'algoritmo K-means. Abbiamo adattato questa pipeline ai dati per assegnare ciascun videogioco al suo cluster corrispondente in base alle caratteristiche di score, numero di users e piattaforma.

Interpretazione dei Risultati

I risultati dell'analisi di clustering sono stati interpretati visualmente attraverso grafici come lo scatter plot. Questi grafici hanno mostrato la distribuzione dei videogiochi all'interno dei cluster identificati, con punti colorati in base al cluster di appartenenza e con stili diversi per identificare facilmente la piattaforma di gioco. Questo ci ha aiutato a comprendere come i giochi sono

raggruppati in base alle caratteristiche di score, users e platform, nonché a identificare eventuali tendenze o segmentazioni significative tra le diverse piattaforme. Utilizzando le distanze euclidee dai centroidi, è possibile identificare i giochi che sono "anomalie" rispetto agli altri nel loro cluster. Questo può portare a una valutazione più approfondita di giochi eccezionali o atipici.

Interpretazione del grafico



1. **Assi:**

- L'asse delle ascisse (x) rappresenta il numero di giocatori che hanno recensito il gioco (users). Dato l'alto numero di recensioni rilasciate per alcuni giochi, per questioni di rappresentazione abbiamo deciso di rappresentare i dati usando una rappresentazione di tipo logaritmica.
- L'asse delle ordinate (y) rappresenta il valore della valutazione posseduta dal gioco (score).

2. **Colori dei punti:**

- I punti sono colorati in base al cluster a cui appartengono:
  - **Cluster 0**: Rosso

        - **Cluster 1**: Blu
        - **Cluster 2**: Verde
    3. **Stile dei punti:**
    - I punti sono stilizzati in base alla piattaforma del gioco:
        - **PlayStation**: Cerchio
        - **Xbox**: Quadrato
        - **PC**: +
        - **Nintendo**: X

Dalla visualizzazione del clustering dei videogiochi in base ai loro punteggi (score), al numero di utenti recensori (users), e le piattaforme di appartenenza, possiamo fare diverse deduzioni:

- **Cluster 0 (Rosso)**:
    - Generalmente contiene giochi con un punteggio medio-alto (score circa tra 70 e 90).
    - La maggior parte dei giochi ha un numero di recensioni medio (users compreso circa tra 50 e 1500 recensioni).

    Questo cluster può rappresentare giochi che sono generalmente accettati ma giocati da un numero di persone non troppo elevato. Potrebbero essere giochi non di nicchia, che interessano solo un numero non troppo elevato di utenti un pubblico limitato, ma che comunque hanno successo per una determinata fascia di mercato.

- **Cluster 1 (Blu)**:
    - Include giochi con un punteggio basso-medio (score tra 20 e 60).
    - I giochi in questo cluster hanno un numero di recensioni medio (Users meno di compreso circa tra 50 e 1500).

    Questi giochi potrebbero essere considerati meno popolari o di qualità inferiore. Potrebbero non avere un grande impatto sul mercato e potrebbero essere titoli che non sono stati bene accolti dai giocatori. Sono quindi giochi ritenuti non di successo, dato il numero di recensioni non favorevoli.

- **Cluster 2 (Verde)**:
    - Comprende giochi con un punteggio alto (score sopra 60, molti tra 80 e 100).

- I giochi in questo cluster tendono ad avere un numero di recensioni molto alto (Users maggiori circa a 3000 recensioni).

Questo cluster rappresenta i giochi di grande successo, sia in termini di qualità che di popolarità. Questi titoli hanno sia un alto punteggio che un grande numero di giocatori, indicando un forte apprezzamento e una vasta base di utenti.

Rilevazione di Dati anomali

Ci sono giochi che, nonostante punteggi bassi, hanno un numero alto di giocati, o viceversa, questi potrebbero essere considerati anomali. Per esempio, giochi nel cluster 1, presentano un numero di giocatori molto più alto del previsto, questo potrebbe indicare un'anomalia nel modo in cui i dati sono stati raccolti o potrebbe essere un particolare titolo che attira i giocatori nonostante le recensioni negative.
In particolare, abbiamo deciso di impostare una soglia threshold 95 percentile, la quale ci permette di identificare gli outliers che non rientrano nella maggior parte dei dati. L'approccio utilizzato si basa sulla selezione dei dati che si trovano oltre il 95° percentile delle distanze dei centri dei cluster. La soglia scelta è comune in task di questo tipo; infatti, con questa soglia evitiamo di classificare troppi punti come anomalie (scegliendo ad esempio 90° percentile) o troppi pochi punti (scegliendo ad esempio 99° percentile), rendendo così 95° percentile un giusto compromesso.

Conclusioni

L'analisi di clustering dei videogiochi utilizzando K-means ha fornito una panoramica generale di alcune relazioni nascoste tra i dati. In particolare, si nota che i giochi che hanno il successo più ampio, quindi con score e numero di giocatori maggiore sono proprio quelli dedicati alla piattaforma PC. Alcuni giochi utilizzabili su PC potrebbero essere utilizzati su altre piattaforme, ottenendo un discreto successo ma giocati da un basso numero di persone, oppure non avere proprio successo. Questo risultato può quindi essere utilizzato nel momento in cui, un'azienda o un privato decide di creare un nuovo gioco, e decidere per quali piattaforme progettarlo, in modo tale da avere una maggiore probabilità che il gioco abbia successo.

Apprendimento supervisionato

Introduzione

Nel nostro caso di studio, l'apprendimento supervisionato è stato utilizzato con scopo di classificare i giochi, presi da un dataset di partenza, il quale è stato arricchito con alcune features realizzate con la creazione della Knowledge Base, in due classi giochi di successo o non di successo.

Per questa parte di progetto, sono state utilizzate le seguenti librerie:

- *Pandas* e *Numpy* per la manipolazione e gestione dei dati
- *Scikit-learn (sklearn)* per gli strumenti di machine learning e l'analisi dei dati.
- *Matplotlib* e *Seaborn* per la creazione di grafici per visualizzare i dati e i risultati del modello, come la curva di apprendimento, la curva di ROC, matrici di confusione e per la visualizzazione avanzata dei dati statistici.

Per raggiungere lo scopo prefissato, si sono attraversate diverse fasi:

Caricamento e Pulizia dei Dati

Il dataset dei videogiochi è stato caricato da un file CSV e successivamente pulito per rimuovere righe con valori mancanti nelle colonne critiche come 'user score', 'genre', 'critics' e 'users', 'user score'.

Preprocessing dei Dati

Le variabili categoriali sono state codificate utilizzando *LabelEncoder*, mentre le variabili numeriche sono state standardizzate utilizzando *StandardScaler*.

I dati sono stati poi suddivisi in training set e test set utilizzando *train_test_split*, con una proporzione del 70% per l'addestramento e del 30% per il test.

```
# Dividiamo i dati in training e test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

Scelta dei modelli:
Nel nostro caso di studio, l'apprendimento supervisionato è stato utilizzato con scopo di classificazione. Per effettuare ciò, abbiamo deciso di utilizzare cinque modelli di apprendimento:

- Random Forest
- Support Vector Machine
- Decision Tree
- Gradient Boosting Classifier
- K-Nearest Neighbors(KNN)

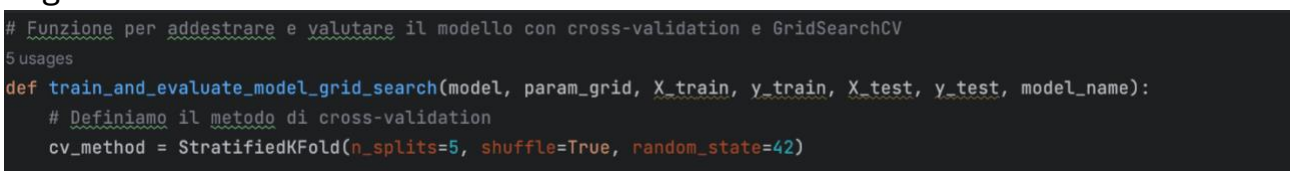
Scelta degli iper-parametri

Per ottimizzare le prestazioni dei modelli di machine learning utilizzati per predire il successo dei giochi nel dataset, abbiamo adottato il metodo della **Grid Search** in combinazione con la **cross-validation**.

Questo approccio è stato selezionato per determinare i migliori iperparametri dei modelli, garantendo una robusta valutazione delle loro prestazioni. La Grid Search è un metodo sistematico che permette di esplorare una griglia predefinita di iperparametri del modello. Ogni combinazione di iperparametri è valutata attraverso una procedura di cross-validation per determinare quale combinazione massimizza una specifica metrica di valutazione, in questo caso l'accuratezza.

Abbiamo utilizzato, per la cross-validation, il metodo *StratifiedKFold*, che suddivide il dataset in k-fold mantenendo la distribuzione delle classi. Questo processo viene ripetuto k volte (ogni volta utilizzando un fold diverso come set di validazione) per ottenere una stima più affidabile delle prestazioni del modello.

Questo approccio è particolarmente adatto per il nostro problema, in cui è necessario gestire un dataset con classi sbilanciate (giochi di successo vs non di successo). Il numero di splits stabilito è 5, come si riporta nell'immagine in seguito:



Iperparametri ricercati e utilizzati

Random Forest:

- **n_estimators:** Il numero di alberi nella foresta. Sono stati esplorati tre valori: 100, 200, e 300.
- **max_depth**: La massima profondità di ciascun albero decisionale. Abbiamo considerato quattro possibilità: None (senza limiti alla profondità), 10, e 20.
- **min_samples_split**: Il numero minimo di campioni richiesti per suddividere un nodo interno. Abbiamo testato i valori 2, 5, e 10.
- **min_samples_leaf:** Il numero minimo di campioni richiesti per essere in un nodo foglia. Sono stati esplorati i valori 1, 2, e 4.

```
rf_param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
```

Support Vector Machine (SVM):

- **C:** Questo parametro controlla quanto il modello penalizzi gli errori di classificazione durante l'addestramento. Valori più alti di C significano che il modello è meno tollerante agli errori nel set di addestramento, cercando di massimizzare la precisione anche a costo di margini decisionali più stretti. Valori testati sono [0.1, 1, 10].
- **kernel:** Il kernel determina il tipo di funzione matematica utilizzata per trasformare lo spazio delle caratteristiche dei dati. Questa trasformazione aiuta SVM a separare i dati che non sono linearmente separabili nello spazio originale delle caratteristiche. Sono stati testati, a riguardo, due tipi di kernel: *'linear'* e *'rbf'*. Il kernel 'linear' è utilizzato quando si vuole una separazione lineare tra le classi nel dataset, mentre il kernel 'rbf' è più flessibile e può gestire separazioni non lineari attraverso la creazione di decisioni basate su funzioni radiali.

```
svm_param_grid = {
    'C': [0.1, 1, 10],
    'kernel': ['linear', 'rbf']
}
```

Decision Tree:

Per la creazione dei singoli alberi, abbiamo deciso di utilizzare gli stessi iperparametri del modello RandomForest, ad esclusione dell'iperparametro "n_estimator", che non è un iperparametro per questo tipo di modello.

```
dt_param_grid = {
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
```

Gradient Boosting Classifier:

- **n_estimators**: Indica il numero di alberi da costruire nel processo di boosting. Più alberi possono migliorare la performance del modello, ma aumentano anche il tempo di addestramento. Per trovare il giusto compromesso, abbiamo scelto i seguenti valori: [50 100 200]
- **learning_rate**: Rappresenta la velocità con cui il modello apprende dagli errori corretti dal modello precedente durante il boosting. Valori più bassi rendono il modello più conservativo, riducendo il rischio di overfitting. Sono stati per questo scelti i seguenti valori: [0.1 0.05 0.01]
- **max_depth**: Specifica la massima profondità di ciascun albero nel boosting. Limitare la profondità può aiutare a controllare la complessità del modello e migliorare la sua capacità di generalizzazione. I valori assegnati sono: [3 4 5]

```
gb_param_grid = {
    'n_estimators': [50, 100, 200],
    'learning_rate': [0.1, 0.05, 0.01],
    'max_depth': [3, 4, 5]
}
```

K-Nearest Neighbors (KNN):

- **n_neighbors:** Specifica il numero di vicini da considerare per la classificazione di un punto. Valori più alti di n_neighbors rendono il modello meno sensibile al rumore nei dati, ma possono anche ridurre la capacità del modello di catturare dettagli più fini.

- **weights:** Indica il peso da assegnare ai punti vicini durante la classificazione. I valori comuni includono:
  - '*uniform*': tutti i punti vicini hanno lo stesso peso.
  - 'distance': i punti vicini hanno un peso inversamente proporzionale alla loro distanza dal punto da classificare.

- **metric:** Specifica la metrica di distanza utilizzata per misurare la distanza tra i punti. Le opzioni comuni includono:
  - *'euclidean'*: la distanza euclidea standard.
  - *'manhattan'*: la distanza di Manhattan, che è la somma delle differenze assolute tra le coordinate.

```
knn_param_grid = {
    'n_neighbors': [3, 5, 7, 9],
    'weights': ['uniform', 'distance'],
    'metric': ['euclidean', 'manhattan']
}
```

Addestramento e Valutazione dei Modelli
Per effettuare l'addestramento, abbiamo deciso di utilizzare la funzione *train_and_evaluate_model*, la quale svolge un ruolo cruciale.

Questa funzione prende in input un modello specifico, addestra quest'ultimo, utilizzando la Cross-Validation con StratifiedKFold e la GridSearchCV, effettuando una ricerca esaustiva sugli iperparametri specificati nella griglia per il modello in questione. Dopo aver trovato i migliori iperparametri, utilizzando il set di addestramento (X_train e y_train), esegue le previsioni sul set di test (X_test), calcola l'accuratezza delle previsioni e genera un report dettagliato delle prestazioni del modello.

Per visualizzare e valutare i risultati ottenuti dal modello, abbiamo utilizzato la funzione *print_results*, la quale stampa diversi risultati:

- Accuracy: l'accuratezza del modello sui dati di test.

- Classification Report: una panoramica dettagliata di precision, recall, F1-score e support per ciascuna classe di output, con relative medie.

- Training e Test Accuracy, : l'accuratezza del modello sui dati di addestramento e test.

- Feature Importances: mostra l'importanza delle caratteristiche utilizzate dal modello per prendere decisioni, queste sono visualizzate solo per i modelli supportati (Random Forest, Decision Tree, Gradient Boosting Classifier).

- Confusion Matrix: una matrice che mostra i veri positivi, i falsi positivi, i veri negativi e i falsi negativi.

- ROC Curve: una curva che visualizza il tasso di veri positivi rispetto al tasso di falsi positivi, con l'Area Under the Curve (AUC) indicata nel titolo del grafico.

Si ritengono di particolare importanza le seguenti metriche:

- Cross-Validation Scores: Mostra i punteggi di accuratezza ottenuti da ciascun fold della cross-validation. Per effettuare ciò si è deciso di utilizzare, in particolare, una cross-validation stratificata con 5 fold, implementata utilizzando 'StratifiedKFold' dal modulo 'sklearn.model.selection', il quale assicura che ogni fold abbia una distribuzione simile delle classi target come nel dataset originale.

- Mean CV Accuracy: è la media delle dei k valori ottenuti durante la cross-validation Scores.

- Standard Deviation CV Accuracy: rappresenta la deviazione standard delle accuratezze ottenute durante la cross-validation. Indica quanto variano i punteggi di accuratezza nei diversi fold.
- Geometric Mean Average Precision (GMAP): rappresenta la media geometrica delle AP per ciascuna classe. E' una misura della precisione media del modello per tutte le classi considerate insieme.
  Questa metrica è seguita dalla deviazione standard della GMAP (±), che indica quanto variano le misure di precisione medie tra i fold della cross-validation.

Esecuzione e Visualizzazione dei Risultati

Il menu principale guida l'utente attraverso la selezione del modello e l'esecuzione dell'addestramento e valutazione del modello scelto. I report di classificazione e altre metriche vengono stampati direttamente nella console. **Si valutano i diversi modelli di apprendimento automatico e si riportano le considerazioni su quelle metriche che si ritengono importanti per valutare il modello non sulla singola iterazione ma sui 5 fold stabili in fase di sviluppo:**

Random Forest

```
Risultati del modello Random Forest:

Classification Report:
                      precision    recall  f1-score   support

gioco non di successo       1.00      1.00      1.00      4770
    gioco di successo       1.00      0.99      0.99       614

             accuracy                           1.00      5384
            macro avg       1.00      0.99      1.00      5384
         weighted avg       1.00      1.00      1.00      5384

Risultati della Cross-Validation:

Cross-Validation Scores: [0.99880573 0.99840764 0.99800955 0.99880573 0.99880573]
Mean CV Accuracy: 0.9985668789808917
Standard Deviation CV Accuracy: 0.0003184713375796344

Accuracy on Test Set: 0.9986998514115899
Training Accuracy: 1.0
Test Accuracy: 0.9986998514115899

Geometric Mean Average Precision (GMAP): 0.9999298157003291 (± 2.2653022390929956e-05)

Feature Importances:
      Feature  Importance
3       users    0.528942
0  user score    0.236737
2       score    0.138193
1     critics    0.096128

Confusion Matrix:
[[4770    0]
 [   7  607]]

Parametri ottimizzati per Random Forest:
{'max_depth': 10, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200}
```

Cross-Validation:
durante il modello ha mostrato una media di accuratezza del 99.86%, con una deviazione standard molto bassa di circa 0.0003. Ciò conferma che il modello è coerentemente preciso su diversi sottoinsiemi di dati.

Geometric Mean Average Precision (GMAP):
la GMAP è 0.99993, il che significa che il modello ha una precisione media molto alta su tutte le classi considerate insieme. La deviazione standard della GMAP è estremamente piccola (circa 0.00002), indicando una consistenza elevata tra i fold della cross-validation.

Valutazione Generale del modello

l modello Random Forest ha dimostrato eccellenti capacità predittive con una precisione e un'accuratezza estremamente elevate. Ha gestito efficacemente la classificazione tra giochi di successo e non di successo, evidenziando una capacità di generalizzazione significativa sia durante la cross-validation che sui dati di test. Le feature più importanti identificate forniscono anche insight su quali aspetti dei giochi influenzano maggiormente il loro successo, il che può essere prezioso per decisioni future nel settore dei giochi.

Support Vector Machine (SVM)

```
Risultati del modello Support Vector Machine:

Classification Report:
                      precision    recall  f1-score   support

gioco non di successo       0.99      1.00      1.00      4770
    gioco di successo       0.99      0.95      0.97       614

             accuracy                           0.99      5384
            macro avg       0.99      0.97      0.98      5384
         weighted avg       0.99      0.99      0.99      5384

Risultati della Cross-Validation:

Cross-Validation Scores: [0.99283439 0.99004777 0.98925159 0.99243631 0.99004777]
Mean CV Accuracy: 0.990923566878981
Standard Deviation CV Accuracy: 0.0014331210191082622

Accuracy on Test Set: 0.9925705794947994
Training Accuracy: 0.9977707006369426
Test Accuracy: 0.9925705794947994

Geometric Mean Average Precision (GMAP): 0.9746674609858819 (± 0.002706586686362229)

Confusion Matrix:
[[4762    8]
 [  32  582]]

Parametri ottimizzati per Support Vector Machine:
{'C': 100, 'gamma': 1, 'kernel': 'rbf'}
```

Cross-Validation:
durante il modello ha mostrato una media di accuratezza del 99.1%, con una deviazione standard molto bassa di circa 0.0014. Ciò conferma che il modello ha una buona coerenza delle prestazioni del modello su diverse ripartizioni dei dati.

Geometric Mean Average Precision (GMAP):
la GMAP è 0.975, il che significa che il modello ha una precisione media molto alta su tutte le classi considerate insieme. La deviazione standard della GMAP è di circa 0.0027 indicando una buona precisone media per le due classi, con una bassa variabilità tra i fold della cross-validation.

Valutazione generale del modello

Il modello SVM ha ottenuto risultati eccellenti con un'accuratezza elevata su entrambi i set di dati (training e test). Ha dimostrato buone prestazioni di generalizzazione e una buona capacità di discriminare tra le due classi di gioco. La precisione, il recall e l'F1-score sono tutti alti, indicando una predizione robusta e bilanciata per entrambe le classi.

Decision Tree

```
Risultati del modello Decision Tree:

Classification Report:
                      precision    recall  f1-score   support

gioco non di successo       1.00      1.00      1.00      4770
    gioco di successo       1.00      0.99      0.99       614

             accuracy                           1.00      5384
            macro avg       1.00      1.00      1.00      5384
         weighted avg       1.00      1.00      1.00      5384

Risultati della Cross-Validation:

Cross-Validation Scores: [0.99880573 0.99880573 0.99800955 0.99840764 0.99761146]
Mean CV Accuracy: 0.9983280254777069
Standard Deviation CV Accuracy: 0.0004642477623284445

Accuracy on Test Set: 0.9986998514115899
Training Accuracy: 0.9996019108280255
Test Accuracy: 0.9986998514115899

Geometric Mean Average Precision (GMAP): 0.990845464214108 (± 0.0021863416195923567)
```

```
Feature Importances:
      Feature  Importance
3       users    0.605922
0  user score    0.292404
2       score    0.066537
1     critics    0.035138

Confusion Matrix:
[[4768    2]
 [   5  609]]

Parametri ottimizzati per Decision Tree:
{'criterion': 'gini', 'max_depth': 10, 'min_samples_leaf': 1, 'min_samples_split': 10}
```

Cross-Validation:
durante la cross-validation il modello ha mostrato una media di accuratezza del 99.83%, con una deviazione standard bassa 0.00046. Questo indica che i punteggi di accuratezza della cross-validation sono vicini alla media, suggerendo una buona coerenza delle prestazioni del modello su diverse ripartizioni dei dati.

Geometric Mean Average Precision (GMAP):
la GMAP è 0.991, con una deviazione standard di circa 0.0027. Questo indica una buona precisone media per le due classi, con bassa variabilità tra i k fold.

Valutazione generale del modello
Il modello Decision Tree ha ottenuto risultati eccellenti con un'accuratezza elevata su entrambi i set di dati (training e test). Ha dimostrato buone prestazioni di generalizzazione e una buona capacità di discriminare tra le due classi di gioco. La precisione, il recall e l'F1-score sono tutti alti, indicando una predizione robusta e bilanciata per entrambe le classi. Le feature "users" e "user score" hanno mostrato di essere le più importanti nel processo decisionale del modello.

K-Nearest Neighbors (KNN):

```
Risultati del modello K-Nearest Neighbors:

Classification Report:
                      precision    recall  f1-score   support

gioco non di successo       0.97      0.99      0.98      4770
    gioco di successo       0.95      0.73      0.83       614

             accuracy                           0.96      5384
            macro avg       0.96      0.86      0.90      5384
         weighted avg       0.96      0.96      0.96      5384

Risultati della Cross-Validation:

Cross-Validation Scores: [0.9669586  0.9633758  0.96656051 0.96218153 0.96656051]
Mean CV Accuracy: 0.9651273885350318
Standard Deviation CV Accuracy: 0.001959957583613252

Accuracy on Test Set: 0.9647102526002972
Training Accuracy: 1.0
Test Accuracy: 0.9647102526002972

Geometric Mean Average Precision (GMAP): 0.9179959860300304 (± 0.0036138427443212677)
```

```
Confusion Matrix:
[[4746   24]
 [ 166  448]]

Parametri ottimizzati per K-Nearest Neighbors:
{'metric': 'manhattan', 'n_neighbors': 5, 'weights': 'distance'}
```

Cross-Validation:
durante la cross-validation il modello ha mostrato una media di accuratezza del 96.51%, con una deviazione standard bassa 0.00196.

Geometric Mean Average Precision (GMAP):
la GMAP è 0.918, con una deviazione standard di circa 0.0036. Questo indica una buona precisone media per le due classi, con bassa variabilità tra i k fold.

Valutazione generale del modello

Il modello KNN ha ottenuto risultati solidi. Tuttavia, presenta una precisione leggermente inferiore per la classe "gioco di successo" rispetto alla classe "gioco non di successo", indicando che potrebbe esserci spazio per

miglioramenti specifici per la classe meno rappresentata. Infatti, questo modello si è discostato dai risultati degli altri modelli indicando una performance minore.

Gradient Boosting Classifier

```
Risultati del modello Gradient Boosting Classifier:

Classification Report:
                      precision    recall  f1-score   support

gioco non di successo       1.00      1.00      1.00      4770
    gioco di successo       1.00      1.00      1.00       614

             accuracy                           1.00      5384
            macro avg       1.00      1.00      1.00      5384
         weighted avg       1.00      1.00      1.00      5384

Risultati della Cross-Validation:

Cross-Validation Scores: [0.99880573 0.99920382 0.99880573 0.99761146 0.99920382]
Mean CV Accuracy: 0.9987261146496815
Standard Deviation CV Accuracy: 0.0005850692060787655

Accuracy on Test Set: 0.9992570579494799
Training Accuracy: 1.0
Test Accuracy: 0.9992570579494799

Geometric Mean Average Precision (GMAP): 0.9998643821020012 (± 0.0001084406968998159)
```

```
Feature Importances:
      Feature  Importance
3       users    0.616802
0  user score    0.295840
2       score    0.060880
1     critics    0.026478

Confusion Matrix:
[[4769    1]
 [   3  611]]

Parametri ottimizzati per Gradient Boosting Classifier:
{'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 200}
```

Cross-Validation:
durante la cross-validation il modello ha mostrato una media di accuratezza del 98.87%, con una deviazione standard bassa 0.00059, suggerendo una buona coerenza delle prestazioni su diverse ripartizione dei dati.

Geometric Mean Average Precision (GMAP):
la GMAP è 0.99986, con una deviazione standard di circa 0.00011. Questo indica una eccellente precisone media per le due classi, con una bassissima variabilità tra i k fold della cross-validation.

Valutazione generale del modello

Il modello Gradient Boosting Classifier ha ottenuto risultati ottimi. Presenta una performance perfetta per entrambe le classi, con precision e recall del 100%. La capacità di generalizzazione del modello è eccellente, come indicato dall'alta concordanza tra le performance sul set di training e sul set di test, e dalla bassissima variabilità dei punteggi di cross-validation.

Conclusioni

Questo approccio integrato per l'analisi e la valutazione dei modelli di classificazione dei videogiochi fornisce una panoramica completa delle prestazioni dei modelli selezionati. L'uso di tecniche avanzate di machine learning e visualizzazione dei dati permette di comprendere meglio quali caratteristiche influenzano il successo dei videogiochi e come i modelli possono essere utilizzati per prendere decisioni informative nel settore dei videogiochi e determinare e distinguere giochi di successo.

Inoltre nel path “gamepy/img/Apprendimento_supervisionato” abbiamo salvato la generazione dei grafici e curve dei modelli utilizzati:

- Matrice di confusione
- Roc Curve
- Learning Curve

Si è scelto di non riportare queste informazioni nella documentazione, perché essi riguardano la singola iterazione di un modello. Dunque, sono serviti durante la fase di sviluppo per verificare il funzionamento, ma non costituiscono informazioni valide per valutare complessivamente i modelli.

Knowledge Base

Introduzione

Nel nostro caso di studio, la Knowledge Base è stata utilizzata pe andare a definire relazioni implicite tra i dati, tra cui i giochi di successo, giochi in tendenza, definizione delle case produttrici con maggiori giochi in trend e la creazione di playlist di giochi appartenenti ad alcuni particolari generi, opportunamente scelti.

Creazione della Knowledge Base (createKB)

Il file sorgente createKB.py è stato implementato per la creazione della Knowledge Base, la quale si concentra sull'elaborazione di un dataset di videogiochi in formato CSV e sulla sua trasformazione in fatti utilizzabili in Prolog.
Iniziamo con un'analisi dettagliata di ciò che il codice fa e come contribuisce alla creazione della KB.

1.Normalizzazione dei Nomi dei Giochi, dei Generi e dei Developer: Il primo passo è la normalizzazione dei nomi dei giochi, dei generi e i nomi dei developer associati a questi. Questo processo include la sostituzione degli apostrofi con underscore, la conversione di punti e due punti in trattini e l'eliminazione di eventuali underscore iniziali o finali contenuti per uniformare il formato.

2. Pulizia e Trasformazione dei Generi: Ogni gioco può appartenere a più generi separati da virgole nel dataset CSV. Il codice separa questi generi e li pulisce da spazi vuoti e caratteri non desiderati, come virgolette singole.

3. Scrittura dei Fatti Prolog: I dati puliti vengono quindi utilizzati per generare fatti Prolog, sfruttati per utilità diverse. Tra questi ritroviamo diverse associazioni:

- **Nome gioco e relativo genere:** vengono riportati i nomi dei giochi e il relativo genere di appartenenza. Poiché ogni gioco può appartenere a più generi simultaneamente, abbiamo deciso di assegnare ad ogni gioco un unico genere, riportando quindi il nome del gioco e un solo genere, per tutti quelli presenti. Questa rappresentazione viene successivamente utilizzata per creare fatti nel file Prolog più complessi.
- **Nome gioco e relativo developer:** vengono riportati i nomi dei migliori giochi e i relativi developers.
- **Nome gioco e relativo anno di uscita**: se il gioco è ritenuto di successo. Per definire il gioco come di successo, abbiamo deciso di raccogliere i giochi con una user Score pari almeno a 90 e almeno 100 critiche.
- **Nome gioco e relativa modalità di gioco**. Data l'alto numero di modalità di gioco disponibili nel dataset di partenza, abbiamo riscontrato la necessità di diminuire la casistica e di effettuare un mapping di queste, utilizzando la funzione classify_game_mode(), la quale va a distinguere le

modalità dei giochi in sole 4 classi: Single player, Multiplayer, co-op mode e no mode. Abbiamo deciso di mantenere la classe no mode in quanto effettuare una determinazione di questa, in base ai dati disponibili e l'alto numeri di modalità, non rendeva possibile la determinazione di questo campo, in quanto questa racchiude una classe di giochi che solitamente possiedono più modalità di gioco possibili allo stesso tempo.

```python
def classify_game_mode(mode):  3 usages
    singleplayer_modes = ['1 Player', '1-2 ', 'No online Multiplayer']
    co_op_modes = ['1-2']
    game_no_mode = ['No info']
    if mode in singleplayer_modes:
        return 'singleplayer'
    elif mode in co_op_modes:
        return 'co-op mode'
    elif mode in game_no_mode:
        return 'no mode'
    else:
        return 'multiplayer'
```

- **Nome gioco e relativo peso in base ai generi:** Abbiamo deciso di riportare questa relazione per individuare i giochi che, in base all'appartenenza a specifiche classi di generi, possono andare in tendenza e quindi essere utili ad un possibile utente che vuole comprare un gioco che soddisfi i suoi gusti. Per fare ciò, abbiamo classificato i generi usando la funzione assign_genre_wieght(), assegnando un peso maggiore ai generi dei giochi più di tendenza, effettuando le relative distinzioni, e assegnando un punteggio basso a tutti i generi dei giochi poco giocati o che raramente vanno in tendenza sulle principali piattaforme videoludiche. Per essere ritenuti in tendenza, abbiamo imposto la regola di un punteggio minimo di peso per essere ritenuto tale.

```python
# Funzione per assegnare pesi ai generi
def assign_genre_weight(genre):  1 usage
    genre_weights = {
        'action': 1,
        'adventure': 1,
        'action adventure': 2,
        'strategy': 0.50,
        'simulation': 0.40,
        'role-playing': 0.70,
        'sports': 0.90,
        'racing': 0.50
    }
    return genre_weights.get(genre.lower(), 0.20)  # Restanti generi con peso 0.20
```

```python
    if weight >= 2.40:
        fact = f"gioco_peso_generi('{game.strip().lower()}', {weight:.2f})"
```

- **Nome del gioco possibile in tendenza e relativo developer:** Abbiamo deciso di riportare questa relazione per individuare le case produttrici aventi giochi realizzati che sono andati in tendenza. Questo fa sì che, un individuo, nel momento in cui decide di voler acquistare un gioco, potrebbe ricevere un gioco o una lista di questi analizzando le case produttrici e consigliandone una che produce molti giochi di successo, aumentando la possibilità di soddisfare la richiesta di un individuo. Per fare ciò abbiamo utilizzato un contatore che andasse a salvare, per ogni developer, il numero di giochi in tendenza ad esso associati, utilizzando la libreria *"Collections >Counter"* e riportando i risultati nel file "*trending_developers_playlist.csv*", all' interno della cartella datasets, riportando il nome dei developers e il relativo contatore, ordinandoli partendo dal developer con contatore più alto.

- **Nome del gioco**: in questo caso abbiamo deciso di riportare esclusivamente il nome del gioco, ma collegandolo a specifiche famiglie di giochi consigliati, che vengono utilizzate in alcuni sistemi automatizzati per suggerire playlist di giochi in base alle loro caratteristiche. Abbiamo preso in considerazione le seguenti classi di appartenenza:
  - **Intensi**
  - **Cooperativi**
  - **Rilassanti**

  Ogni gioco, per appartenere ad una specifica classe, deve rispettare delle condizioni, definite nelle funzioni:
  - **is_intense_game()**
  - **is_coop_game()**
  - **is_relaxing_game()**

  Ognuna di queste funzioni effettua un'analisi sui generi di appartenenza del gioco e viene utilizzata per effettuare in maniera automatica la classificazione dei giochi in queste 3 classi.

4.Salvataggio della KB: il modo per gestire i fatti e le regole è stato effettuato tramite il salvataggio diretto nel file "*games_kb.pl*", avendo una base di conoscenza relativamente stabile.

Utilizzo della Knowledge Base (useKB):
Dopo aver creato la KB, si è implementato il file sorgente useKB.py per utilizzare questa base di conoscenza interattivamente, per effettuare query e inferire nuova conoscenza sui videogiochi

Una volta avviato il file useKB.py, all'utente viene presentato un menu interattivo, il quale propone le seguenti azioni possibili:

- **Mostrare 10 giochi di un genere scelto:** L'utente può selezionare un genere tra quelli disponibili e ottenere una lista di giochi appartenenti a quel genere.
- **Mostrare 10 giochi con maggiore successo a partire da un certo anno:** L'utente può specificare un anno compreso tra il 1996 e il 2020 e ricevere una lista di giochi con un punteggio superiore a 80 e un numero significativo di critiche, usciti da quell'anno in poi.
- **Mostrare 5 migliori giochi di un developer scelto:** L' utente può inserire il nome di una casa produttrice tra quelle disponibili e ottenere una lista dei 5 migliori giochi prodotti da questa. I giochi presentati saranno quelli prodotti dalla casa produttrice inserita, e aventi un successo molto alto sul mercato (aventi uno score maggiore o uguale a 90, basato su un numero esiguo di recensioni e votazioni ricevute dagli utenti).
- **Mostrare 10 giochi di una modalità di gioco scelta:** L' utente può inserire il tipo di modalità di gioco, scegliendo tra Singleplayer, Multiplayer, co-op mode e no mode, e ottenendo una lista di giochi della modalità inserita.
- **Mostrare 5 giochi che potrebbero essere di tendenza:** Viene restituita una piccola lista contenente 5 giochi che potrebbero essere di tendenza, forniti casualmente tra quelli di tendenza presenti nel file Prolog, per evitare di monopolizzare i risultati su solo alcuni dei giochi di tendenza.
- **Mostrare alcuni giochi di un developer:** Vengono considerati i primi 5 developers ritenuti maggior produttori di giochi di tendenza, ottenuti in precedenza e successivamente vengono restituiti, per alcuni di essi, qualche gioco di tendenza.
- **Creare playlist con 10 giochi rilassanti/intensi/cooperativi:** vengono create, sulla base dei giochi presenti del file Prolog, delle playlist di 10 giochi attinenti alla categoria scelta.

Risultati e Considerazioni Finali.
Abbiamo raggiunto diversi obiettivi importanti, ottenendo informazioni alcune informazioni in parte note sui giochi, ma anche conoscenza nuova analizzando le relazioni tra i vari giochi, come predire i giochi di tendenza, i developers e creare playlist dedicate. Inoltre, con tali informazioni possiamo anche analizzare le situazioni del mercato dei videogiochi e con la relativa analisi di queste caratteristiche, è possibile realizzare considerazioni sui giochi in uscita nel futuro.

Risoluzione di un CSP – soddisfazione dei vincoli

L'obiettivo era quello di creare una playlist di giochi con particolari caratteristiche. Per conseguire tale scopo, l'approccio adottato ha trattato il problema come una questione di ottimizzazione dei vincoli, che includono:

1. Esattamente 10 giochi nella selezione
2. Numero totale di critics non superiore a 300.
3. Media del punteggio degli utenti almeno 7.0.
4. Non più di due giochi dello stesso genere.

Per ottenere la seguente playlist sono stati utilizzati due algoritmi di ottimizzazione:

Random Walk, il quale seleziona casualmente 10 giochi dal dataset e valuta il numero di violazioni dei vincoli. Se trova una combinazione con meno violazioni rispetto alla migliore trovata finora, questa ultima viene selezionata come la migliore. L'algoritmo continua per un numero fisso di iterazioni (max_iter) o fino a quando non trova una selezione che soddisfa tutti i vincoli (zero violazioni).

Simulated Annealing, L'algoritmo Simulated Annealing inizia con una soluzione casuale e utilizza una tecnica di annealing per esplorare nuove soluzioni. La temperatura iniziale è impostata ad un valore elevato (temp), fattore di energia(1000), e viene gradualmente ridotta ad ogni iterazione moltiplicandola per un fattore (alpha), fattore di raffreddamento (0.99). Questo consente all'algoritmo di accettare occasionalmente soluzioni peggiori per evitare di rimanere bloccati in minimi locali. Se una nuova soluzione ha meno violazioni rispetto alla migliore soluzione trovata finora, questa viene aggiornata come la migliore. L'algoritmo termina dopo un numero fisso di iterazioni (max_iter) o quando la temperatura (che controlla la probabilità di accettazione di soluzioni peggiori) raggiunge un valore molto basso.

Inoltre, si è deciso di eseguire 10 iterazioni per ogni algoritmo per ottenere una stima più accurata delle prestazioni dei due algoritmi e delle loro capacità di trovare soluzioni che soddisfano i vincoli. Poiché gli algoritmi utilizzano elementi casuali nella selezione dei giochi (in particolare il Random Walk), eseguire più iterazioni riduce la variabilità nei risultati e fornisce una valutazione media che è più rappresentativa delle capacità dell'algoritmo.

Un esempio di output del seguente file sorgente è:

l'utente può scegliere tra Random Walk e Simulated Annealing.
Dopo 10 esecuzioni dell'algoritmo selezionato, il programma stamperà:

- La media del tempo di esecuzione dell'algoritmo.
- La media delle violazioni dei vincoli.
- Salverà la migliore selezione trovata nella path "results\nomeAlgoritmo_game_with_constraints".

È importante soffermarsi sulla media delle violazioni, quest'ultima è una metrica per valutare la qualità delle soluzioni prodotte dagli algoritmi di ottimizzazione in termini di soddisfacimento dei vincoli. In particolare, si ha:

- **Violazioni dei vincoli**: Ogni volta che una selezione di giochi non soddisfa uno dei vincoli specificati (ad esempio, superare il numero massimo di critici, avere un punteggio medio degli utenti inferiore a 7.0, ecc.), si conta come una violazione. Ogni violazione aumenta il conteggio totale delle violazioni.
- **Media delle violazioni**: Dopo aver eseguito l'algoritmo di ottimizzazione più volte (ad esempio, 10 volte), si calcola la media delle violazioni dei vincoli su tutte le esecuzioni.

Esempi di risultati ottenuti:

```
Inserisci il numero della tua scelta: 1
Media del tempo di esecuzione: 0.1149 secondi
Media delle violazioni dei vincoli: 0.00
Inserisci il numero della tua scelta: 2
Media del tempo di esecuzione: 0.1669 secondi
Media delle violazioni dei vincoli: 0.15

Media del tempo di esecuzione: 0.1189 secondi
Media delle violazioni dei vincoli: 0.10
I vincoli sono stati soddisfatti.
Media del tempo di esecuzione: 0.1523 secondi
Media delle violazioni dei vincoli: 0.03
I vincoli sono stati soddisfatti.
```

In genere durante i vari test fatti la media delle valutazioni non hai mai prodotto risultati lontani dallo zero, questo significa che, in media, le soluzioni trovate dagli algoritmi di ottimizzazione violano i vincoli in misura molto ridotta. In altre parole, su 10 esecuzioni dell'algoritmo, il numero medio di violazioni è ad esempio 0.10, il che implica che le soluzioni sono generalmente molto vicine a soddisfare tutti i vincoli. Una media vicino a 0 è un indicatore di buone prestazioni dell'algoritmo, suggerendo che la maggior parte delle soluzioni trovate sono valide e rispettano quasi tutti i vincoli e quindi l'algoritmo ha buone performance.

Conclusioni

Con l'implementazione di due algoritmi avremo sicuramente casualità nella scelta delle soluzioni grazie al Random Walk, ma avremo anche un graduale miglioramento grazie al Simulated Annealing, il quale incorpora una strategia più sofisticata che permette una graduale miglioramento delle soluzioni. Infatti quest'ultimo è in grado di superare minimi locali grazie alla sua capacità di accettare soluzioni peggiori, migliorando le possibilità di trovare una soluzione globalmente ottima rispetto a molti problemi di ottimizzazione.

CONCLUSIONI E SVILUPPI FUTURI

Abbiamo notato come in generale, l'utilizzo delle tecniche di machine learning possano aiutare sia durante la progettazione di un videogioco che in fase di analisi, aiutando a prendere le giuste scelte per creare un gioco che, dati alla mano, potrebbe diventare di successo, o anche tendenza.

Si può pensare di espandere il progetto attraverso l'integrazione di API di alcune piattaforme di distribuzione digitale, per espandere l'analisi e la capacità predittive dei modelli, anche sui videogiochi recenti. Inoltre, possiamo integrare delle recensioni per estrarre informazioni sui sentimenti dei videogiocatori. Potrebbe essere una implementazione futura quella di consigliare, in base a dei profili utenti, giochi piaciuti da utenti simili a quelli presi in considerazione. Infine, si potrebbe sfruttare i modelli di apprendimento supervisionato per estrarre dai giochi di successo i generi e sfruttarli per creare nuove playlist, divise in nuove categorie, variabili nel tempo.