Videogames Recommender System

Documentazione Progetto Ingegneria della Conoscenza

Gruppo di lavoro

- Nicolò Sciancalepore, 735589, n.sciancalepore2@studenti.uniba.it
- Saverio de Candia, 736578, s.decandia15@studenti.uniba.it
- Alessandro Piergiovanni, 738044, a.piergiovanni5@studenti.uniba.it

Repository progetto:

https://github.com/alessandropier/Icon-22-23

AA 2022-23

INDICE

Introduzione

Visto il crescente interesse nei confronti dei videogiochi da parte dei giovani, il gruppo ha deciso di venire incontro a tali esigenze creando questo progetto con due funzioni principali: una previsione sui limiti di età di un eventuale nuovo gioco sul mercato e un recommender system creato al fine di suggerire agli utenti giochi affini ai loro gusti.
Per far ciò si è fatto ricorso a due dataset presenti sul sito Kaggle, successivamente uniti per soddisfare le esigenze progettuali.

Strumenti

Il gruppo ha deciso di utilizzare Python come linguaggio di programmazione e Jupyter Notebook e VisualStudio Code come IDE. Per condividere i file del progetto, il gruppo ha scelto di creare e utilizzare un repository su GitHub.

Librerie

Per lo sviluppo di questo progetto sono state utilizzate le seguenti librerie:

- Pandas: una libreria per la manipolazione e l'analisi dei dati. Essa mette a disposizione strutture dati e operazioni per manipolare tabelle numeriche e serie temporali;
- Numpy: una libreria che aggiunge supporto a grandi matrici e array multidimensionali insieme ad una collezione di funzioni matematiche per operare efficientemente su tali strutture;
- Sklearn: una libreria di apprendimento automatico. Contiene algoritmi di classificazione, regressione e clustering;
- Matplotlib: una libreria per la creazione di grafici;
- Deep translator: una libreria che permette la traduzione di dati testuali in diverse lingue;
- Pyswip: una libreria che permette l'integrazione del linguaggio Prolog in ambiente Python.

Elenco argomenti di interesse

- Creazione della Knowledge Base (KB), utile all'analisi delle informazioni contenute nei dataset in modo da ottenere un sistema in grado di rispondere a determinate query;
- Apprendimento supervisionato, utilizzato per la predizione della classificazione di un videogioco fornito in input dall'utente;
- Apprendimento non supervisionato (clustering), usato per raggruppare i videogiochi in diverse classi nella realizzazione di un recommender system per consigliare all'utente 10 videogiochi sulla base di un gioco a lui gradito.

Creazione della Knowledge Base (KB)

Una knowledge base, o base di conoscenza, è un sistema organizzato e centralizzato che contiene una vasta quantità di informazioni, dati, fatti, procedure e altri tipi di conoscenza su un determinato argomento o dominio.
Le knowledge base possono assumere diverse forme e formati, inclusi database, documenti, wiki, sistemi di gestione delle conoscenze, e altro ancora. Possono essere utilizzate in una vasta gamma di contesti e sono fondamentali per migliorare l'efficienza operativa, facilitare la condivisione della conoscenza tra dipendenti e clienti, ridurre gli errori e promuovere la collaborazione.

Si è utilizzata una base di conoscenza in logica del primo ordine, ovvero un insieme di proposizioni, dette assiomi, considerate vere senza bisogno di dimostrazione.

Il processo di creazione di una base di conoscenza coinvolge i seguenti passaggi:

- Definizione del Dominio: si decide il dominio che desideriamo rappresentare, ovvero il mondo reale, un mondo immaginario o un mondo astratto, come numeri o insiemi. Nel caso del nostro progetto, decidiamo di operare sul dataset di videogiochi ottenuto dopo le attività di preprocessing;
- Definizione delle Proposizioni Atomiche, che serviranno a rappresentare il nostro dominio;
- Assiomatizzazione del Dominio: si definiscono le proposizioni che saranno considerate vere nell'interpretazione del dominio. Queste proposizioni costituiranno gli assiomi della base di conoscenza e rappresenteranno le verità fondamentali del nostro dominio.
- Interrogazione del Sistema: Una volta definita la base di conoscenza, possiamo porre domande o query al sistema. Il sistema determinerà se specifiche proposizioni sono conseguenze logiche della base di conoscenza, ossia se sono vere in tutti i modelli della base di conoscenza.

Per implementare una base di conoscenza in logica del primo ordine, è stato utilizzato il linguaggio di programmazione Prolog con la libreria pyswip. Le caratteristiche selezionate sono state rappresentate come fatti nella base di conoscenza, con la definizione dei loro domini.

Ai fini della creazione di una Knowledge Base adatta al nostro progetto, ci siamo serviti di due dataset contenenti diverse informazioni relative ad un gran numero di videogiochi. Successivamente, si sono susseguite diverse operazioni di preprocessing sui dataset affinché potessero essere utilizzati per i nostri scopi.

Preprocessing

Sono stati selezionati due dataset su Kaggle:

- Video Game Sales: https://www.kaggle.com/datasets/gregorut/videogamesales
- Video Game Reviews from JVC: https://www.kaggle.com/datasets/floval/jvc-game-reviews

Successivamente, è stato effettuato un merge tra i due dataset, poiché abbiamo ritenuto necessarie alcune colonne presenti in entrambi. Il merge è stato fatto in base alle colonne Name (contenente i nomi internazionali dei videogiochi) e Platform (contenente le varie piattaforme per cui un gioco è disponibile). Successivamente, è stata fatta una selezione sulle colonne del dataset risultante, in modo tale da tenere solo quelle necessarie ai fini del nostro progetto.
Il risultato è stato un dataset contenente le colonne:

- Name: nome del videogioco;
- Platform: piattaforma per cui quel gioco è disponibile;

- RatingOutOf20: valutazione assegnata dal pubblico al videogioco in questione;
- Publisher: nome dell'azienda di produzione del videogioco;
- Year: anno di pubblicazione del videogioco;
- Classification: limiti d'età per quel videogioco;
- Genre: singolo genere che identifica il videogioco.

Nota: Ogni videogioco appare nel dataset una volta per ciascuna piattaforma per cui esso è disponibile. Abbiamo preso questa decisione per due motivi:

- Alcuni giochi presentano descrizioni e valutazioni diverse per le loro versioni su diverse piattaforme;
- Questo ci permetterà di suggerire all'utente videogiochi affini alla console in suo possesso.

Sono state rimosse le missing values e i valori "Unknown" dalle colonne "Classification" e "Publisher". Nel caso della colonna "Genre", i valori "Unknown" e "Others" sono stati sostituiti con "Indie".

Sono state standardizzate le colonne "RatingOutOf20" e "Classification", eliminando dai valori delle celle di queste colonne, rispettivamente "/20" e "ans".

Inoltre, si è proceduto alla creazione di un dizionario che associa un valore intero a ciascun valore contenuto nelle colonne feature. Si è deciso di fare questo poiché gli algoritmi di apprendimento supervisionato e non supervisionato non supportano operazioni su stringhe, e quindi è stato necessario associare valori numerici a ciascun valore contenuto nelle colonne del dataset.

Il risultato di questa operazione è un dataset composto da soli dati numerici, come nel seguente screenshot:

	Name	Platform	RatingOutOf20	Publisher	Year	Classification	Genre	Classification_format
1	0	0	13	0	0	2	0	12
2	1	1	15	1	1	2	1	12
3	2	1	16	2	2	2	2	12
4	3	1	16	3	2	2	3	12
5	4	2	14	4	3	2	4	12
6	5	1	13	5	3	2	5	12
7	6	1	15	5	4	2	5	12
8	7	0	10	6	5	2	0	12
9	8	1	16	5	6	2	5	12
10	9	1	15	7	7	2	5	12
11	10	1	17	8	4	2	4	12
12	11	2	10	9	8	2	6	12
13	11	1	10	9	8	2	6	12
14	12	0	10	10	0	2	0	12
15	13	0	10	11	0	2	1	12
16	14	3	16	11	3	2	1	12
17	15	2	9	11	9	2	1	12
18	16	2	11	12	0	2	4	12
19	17	1	18	1	5	2	1	12
20	18	1	16	13	0	2	6	12
21	19	3	15	3	2	2	7	12
22	20	4	15	14	3	2	4	12
23	21	0	16	15	9	2	8	12
24	22	0	16	15	3	2	8	12
25	23	1	10	16	9	2	5	12
26	24	4	11	14	4	2	6	12
27	25	4	16	17	10	2	9	12
28	26	3	11	4	0	2	3	12
29	27	1	9	18	3	2	5	12
30	28	3	14	15	8	2	3	12

***Figura 1** – Dataset dei giochi dopo il preprocessing*

Rappresentazione della base di conoscenza in Prolog

Dopo aver effettuato delle operazioni di preprocessing sul dataset è stata creata una base di conoscenza a partire dal dataset ottenuto dalla fase precedente. Questo è stato possibile creando dei fatti che sarebbero stati successivamente utili per le interrogazioni che abbiamo deciso di effettuare sulla base di conoscenza.

```
54  #Apri il file KB.pl in modalità scrittura
55  with open("./Code/KB.pl", "w") as prolog_file:
56      prolog_file.write(":- discontiguous genre/2.\n")
57      prolog_file.write(":- discontiguous giochi_pegi/3.\n")
58      prolog_file.write(":- discontiguous val_publisher/2.\n")
59      prolog_file.write(":- discontiguous genre_publisher/2.\n")
60      prolog_file.write(":- discontiguous name_publisher_val/3. \n")
61
62      #Itera attraverso le righe del dataset
63      for index, row in dataset.iterrows():
64          name = row[0]
65          genre = row[6]
66          pegi = row[5]
67          publisher = row[3]
68          rating = row[2]
69
70          #Scrivo i fact nel file KB.pl
71          prolog_fact = f"genre({name}, {genre}).\n"
72          prolog_file.write(prolog_fact)
73
74          prolog_fact = f"giochi_pegi({name}, {genre}, {pegi}).\n"
75          prolog_file.write(prolog_fact)
76
77          prolog_fact = f"val_publisher({publisher}, {rating}).\n"
78          prolog_file.write(prolog_fact)
79
80          prolog_fact = f"genre_publisher({genre}, {publisher}).\n"
81          prolog_file.write(prolog_fact)
82
83          prolog_fact = f"name_publisher_val({name}, {publisher}, {rating}).\n"
84          prolog_file.write(prolog_fact)
```

Figura 2 – Creazione dei fatti in Prolog

Successivamente, si è deciso di creare alcuni predicati di ausilio ad alcune query:

- Predicato 1: count_elements([], Count).

```
count_elements([], 0).
count_elements([_|Tail], Count) :-
    count_elements(Tail, TailCount),
    Count is 1 + TailCount.
```

Predicato utilizzato per contare il numero di elementi di una lista. Il predicato è stato definito ricorsivamente e ad ogni iterazione viene salvato il risultato in TailCount. Il risultato finale sarà contenuto in Count. Se riceve in input una lista vuota, restituisce 0.

- Predicato 2: sum_list([], Sum).

```
sum_list([], 0).
sum_list([Head|Tail], Sum) :-
    sum_list(Tail, TailSum),
    Sum is Head + TailSum.
```

Predicato utilizzato per calcolare la somma degli elementi numerici in una lista. Il predicato è stato definito ricorsivamente e ad ogni iterazione viene salvato il risultato in TailSum. Il risultato finale sarà contenuto in Sum. Se riceve in input una lista vuota, restituisce 0.

- Predicato 3: average([], Average).

```
average([], 0).
average(List, Average) :-
    sum_list(List, Sum),
    count_elements(List, Count),
    Count > 0, % Evita la divisione per zero
    Average is Sum / Count.
```

Predicato che stampa la media di una lista numerica fornita in input utilizzando le funzioni sum_list e count_elements definite precedentemente.
Il predicato sum_list viene utilizzato per avvalorare la variabile "Sum" con la somma di tutti i valori della lista. Il predicato count_element viene utilizzato per avvalorare la variabile "Count" con il numero dei valori presenti nella lista.
Di seguito, viene effettuato un controllo su Count per verificare che sia maggiore di 0 in maniera da evitare una divisione impossibile.
Infine, il risultato del predicato sarà contenuto all'interno della variabile "Average".

- Predicato 4: publisher_with_most_games_of_genre(Genre, Publisher).

```
publisher_with_most_games_of_genre(Genre, Publisher) :-
    findall(P, genre_publisher(Genre, P), Publishers),
    list_max_occurrences(Publishers, Publisher, _).
```

Predicato che viene utilizzato per trovare il publisher con più giochi di un genere specifico.
Per farlo vengono utilizzati due predicati ovvero findall e list_max_occurences.
Il predicato findall è definito con la seguente sintassi: findall(+Template, +Goal, -Bag)

  - Template è una variabile che rappresenta ciò che si desidera trovare.
  - Goal è la condizione da soddisfare.
  - Bag è la lista che conterrà tutte le istanze trovate di Template.

Quindi abbiamo definito findall come segue: findall(P, genre_publisher(Genre, P), Publishers).
In questo modo salviamo tutti i publisher "P" che si trovano all'interno della relazione genre_publisher(Genre, P) all'interno di "Publishers".
Di seguito viene chiamato il predicato list_max_occurrences spiegato successivamente.

- Predicato 5: list_max_occurences(List, Max, Occurrences).

```
list_max_occurrences(List, Max, Occurrences) :-
    msort(List, Sorted), % Ordina la lista
    pack(Sorted, Packed), % Raggruppa gli elementi consecutivi
    find_max_occurrences(Packed, Max, Occurrences). % Trova il valore massimo
```

Predicato che viene utilizzato per restituire il valore massimo e il numero di occorrenze in una lista.
Per prima cosa si utilizza il predicato msort per ordinare la lista ottenuta in input. Di seguito

vengono chiamate le funzioni "pack" e "find_max_occurences" spiegate successivamente.

- Predicato 6: pack([], []).

```
pack([], []).
pack([X|Xs], [[X|Packed]|Rest]) :-
    transfer(X, Xs, Ys, Packed),
    pack(Ys, Rest).
```

Il predicato pack serve a raggruppare gli elementi consecutivi di una lista in sotto-liste.

  - La regola base pack([], []). data in input una lista vuota, allora restituisce una lista di output vuota.
  - La regola ricorsiva pack([X|Xs], [[X|Packed]|Rest]) suddivide la lista iniziale [X|Xs] in due parti:
    - Il primo elemento X diventa l'inizio di una nuova sotto-lista [X|Packed].
    - Gli elementi successivi Packed sono gli elementi consecutivi uguali a X.
    - La funzione transfer è utilizzata per estrarre gli elementi consecutivi uguali, mentre la lista restante dopo la rimozione di questi elementi viene passata ricorsivamente a pack per essere ulteriormente suddivisa.
    - Rest è il risultato dell'applicazione ricorsiva di pack sulla lista rimanente Ys.

- Predicato 7: transfer(X, [], [], []).

```
transfer(_, [], [], []).
transfer(X, [Y|Ys], [Y|Ys], []) :- X \= Y.
transfer(X, [X|Xs], Ys, [X|Packed]) :- transfer(X, Xs, Ys, Packed).
```

Il predicato transfer viene utilizzato da pack per trasferire gli elementi consecutivi uguali dalla lista di input alla sotto-lista corrente.

  - La regola base transfer(_, [], [], []). data in input una lista vuota, allora restituisce una lista di output vuota.
  - La regola transfer(X, [Y|Ys], [Y|Ys], []) :- X = Y. controlla se il primo elemento X è diverso dal successivo Y. In tal caso, significa che il raggruppamento degli elementi uguali è terminato. Quindi, la regola restituisce Ys come lista rimanente e [] come lista di elementi raggruppati fino a quel punto.
  - La regola finale transfer(X, [X|Xs], Ys, [X|Packed]) :- transfer(X, Xs, Ys, Packed). trasferisce tutti gli elementi consecutivi uguali a X nella lista Packed, lasciando il resto della lista in Ys. Questo processo è ricorsivo e continua fino a quando si verifica la regola base definita inizialmente.

Per riassumere, il predicato pack suddivide la lista in ingresso in sotto-liste contenenti elementi consecutivi uguali, mentre il predicato transfer viene utilizzato all'interno di pack per estrarre tali elementi consecutivi.

- Predicato 8: find_max_occurrences([]; Max, Occurrences).

```
find_max_occurrences([], _, 0).
find_max_occurrences([L|Ls], Max, Occurrences) :-
    length(L, Len),
    find_max_occurrences(Ls, Max1, Occurrences1),
    (Len > Occurrences1 -> Max = L, Occurrences = Len; Max = Max1, Occurrences = Occurrences1).
```

Il predicato find_max_occurrences è utilizzato per trovare il valore massimo e il numero di occorrenze in una lista di liste.

- La prima regola find_max_occurrences([], _, 0). stabilisce che, se la lista di liste in ingresso è vuota, il massimo numero di occorrenze è 0. Questo è il caso base della ricorsione.
- La seconda regola find_max_occurrences([L|Ls], Max, Occurrences) viene applicata quando la lista di liste non è vuota.
  - L rappresenta la testa della lista, ovvero la prima lista all'interno della lista di liste.
  - Ls rappresenta la coda della lista, ovvero il resto delle liste.
  - length(L, Len) calcola la lunghezza della lista L, cioè il numero di elementi in L.
  - find_max_occurrences(Ls, Max1, Occurrences1) viene chiamato ricorsivamente per trovare il massimo numero di occorrenze nelle restanti liste di liste.
  - La condizione (Len > Occurrences1 -> Max = L, Occurrences = Len; Max = Max1, Occurrences = Occurrences1) confronta la lunghezza della lista L con il massimo numero di occorrenze trovato nelle restanti liste di liste.
    - Se la lunghezza di L è maggiore di Occurrences1, allora assegnamo L a Max e Len a Occurrences. Questo significa che abbiamo trovato una nuova lista con un numero maggiore di occorrenze.
    - Altrimenti, mantieniamo il valore precedente di Max e Occurrences.

Per concludere, find_max_occurrences attraversa ricorsivamente la lista di liste e restituisce il valore massimo e il numero di occorrenze della lista con il maggior numero di elementi.

- Predicato 9: highest_rated_game(Publisher, Game).

```
highest_rated_game(Publisher, Game) :-
    name_publisher_val(Game, Publisher, Rating),
    \+ (name_publisher_val(OtherGame, Publisher, OtherRating), OtherRating > Rating, Game \= OtherGame).
```

Il predicato highest_rated_game cerca di trovare il gioco con valutazione più alta pubblicato da un determinato editore (Publisher).:

- name_publisher_val(Game, Publisher, Rating) è un fatto che associa un gioco (Game) pubblicato da un editore specifico (Publisher) a un rating (Rating).
- La riga successiva \\+ (name_publisher_val(OtherGame, Publisher, OtherRating), OtherRating > Rating, Game \\= OtherGame) usa il "negation as failure" per assicurarsi che non ci siano altri giochi pubblicati dallo stesso editore (Publisher) con un rating maggiore di quello attuale (Rating).
  - OtherGame e OtherRating rappresentano rispettivamente altri giochi e i loro rating associati.
  - OtherRating > Rating controlla se il rating di OtherGame è maggiore del rating del gioco attuale (Rating).
  - Game \\= OtherGame garantisce che il gioco OtherGame non sia lo stesso del gioco attuale (Game), in modo da non confrontare il gioco con sé stesso.

Per concludere, questa regola seleziona il gioco con il rating più alto tra quelli pubblicati da un determinato editore, garantendo che non ci siano altri giochi con un rating maggiore dello stesso editore.

- Predicato 10: is_age_appropriate(Age, Pegi).

```
is_age_appropriate(Age, Pegi) :-
    Age >= Pegi.
```

Il predicato viene utilizzato per capire se una determinata età permette di giocare un gioco di un determinato Pegi. Restituisce vero se Age è ≥ Pegi, falso altrimenti.

Successivamente sono state realizzate 7 interrogazioni che utilizzano i predicati precedentemente definiti e i fatti presenti nella base di conoscenza:

1. Chiede in input genere e classificazione d'età e dà in output la lista dei giochi di quel genere con quel pegi;
2. Chiede in input un genere e stampa quanti giochi di quel genere ci sono;
3. Chiede in input un genere e stampa i titoli dei giochi di quel genere;
4. Chiede in input un publisher e restituisce la media dei rating dei suoi giochi;
5. Chiede in input un genere e restituisce il publisher con più giochi di quel genere;
6. Chiede in input un publisher e restituisce il gioco di quel publisher con il rating più alto;
7. Chiede in input nome gioco ed età di una persona e restituisce true se può giocarlo, false altrimenti.

Di seguito un esempio di esecuzione per la query #1:

```
print("Query 1: Chiede in input genere e classificazione d'età e da in output la lista dei giochi di quel genere con quel pegi\n")

genre = input("Inserisci il genere che vuoi cercare: ").lower()
genre_format = search_format(genre)

pegi = input("Inserisci la classificazione PEGI (Esempio: +3): ").lower()
pegi_format = search_format_classification(pegi)

if pegi_format == None:         #Controllo per evitare valore "None" pegi_format che causa errori
    pegi_format = -1

query = list(prolog.query(f"giochi_pegi(X, {genre_format}, {pegi_format}).")) #Esegue la query
print(f"\nGiochi con pegi {pegi}:")
print_query_format(query)                                                     #Stampa il risultato della query
```

Figura 3 – Codice di esecuzione della query #1

Apprendimento Supervisionato

L'apprendimento supervisionato è una tecnica di apprendimento automatico in cui un algoritmo impara da un set di dati di input etichettati, cercando di trovare modelli per fare previsioni su nuovi dati.
Questo tipo di apprendimento è caratterizzato da un insieme di esempi ed un insieme di features, queste ultime suddivise in features di input e feature target. Una feature è una funzione che va dall'insieme degli esempi ad un valore.
Esistono due tipi di applicazioni per questo tipo di apprendimento:

- Classificatori: algoritmi di apprendimento automatico che prendono in input un insieme di dati e li assegna a una o più categorie o classi predefinite;
- Regressori: algoritmi di apprendimento automatico che predicono o stimano un valore numerico continuo basato su un insieme di variabili di input.

Per i nostri scopi, si è scelto di utilizzare un classificatore, in quanto il nostro scopo è quello di assegnare una classe di limiti d'età (PEGI) a giochi inseriti dall'utente in input.
Le classi in questione sono: +3, +7, +12, +16, +18 e rappresentano la nostra feature target.

Le feature di input utilizzate per la predizione sono: Nome del videogioco, Genere, Publisher, Piattaforma, Valutazione della community e Anno d'uscita.

Parametri ottimali

Per la scelta dei parametri di input dei vari algoritmi abbiamo creato delle liste di parametri da inserire in GridSearchCV, che a sua volta ha restituito i parametri ottimali per ciascun algoritmo, applicato al nostro dataset.

```
13      knn = KNeighborsClassifier()
14      parameters_knn = {
15          "n_neighbors": (1, 10, 13),
16          "weights": ("uniform", "distance"),
17          "metric": ("minkowski", "manhattan")}
18
19      grid_search_knn = GridSearchCV(
20          estimator = knn,
21          param_grid = parameters_knn,
22          scoring = "accuracy",
23          n_jobs = -1,
24          cv = 5
25      )
26
27      knn_1 = grid_search_knn.fit(x_train, y_train)
28      y_pred_knn1 = knn_1.predict(x_test)
29      print("Best params", grid_search_knn.best_params_)
```

***Figura 4** – Calcolo dei best_params_ per il KNN*

```
gau = GaussianNB()
parameters_gau = {'var_smoothing': np.logspace(0, -9, num=100)}

# GridSearch
grid_search_gau = GridSearchCV(
    estimator=gau,
    param_grid=parameters_gau,
    cv=10,
    verbose=1,
    scoring='accuracy'
)

gau_1 = grid_search_gau.fit(x_train, y_train)
y_pred_gau1 = gau_1.predict(x_test)
print("Best params", grid_search_gau.best_params_)
```

Figura 5 – Calcolo dei best_params_ per Gaussian Naive Bayes

```
rf = RandomForestClassifier()

parameters_rf = {
    'n_estimators': (100, 500, 1300),
    'max_depth': (15, 25),
    'min_samples_split': (3, 15),
    'min_samples_leaf': (1, 3)
}

grid_rf = GridSearchCV(rf, parameters_rf, cv=10, verbose=1,
                       scoring='accuracy')
rf1 = grid_rf.fit(x_train, y_train)
y_pred_rf1 = rf1.predict(x_test)
print("Best params", grid_rf.best_params_)
```

Figura 6 – Calcolo dei best_params_ per il Random Forest

Una volta effettuata questa procedura siamo arrivati ad ottenere i seguenti risultati, utilizzati poi per determinare l'algoritmo di classificazione più efficiente per il nostro progetto.

```
KNN:
Best params {'metric': 'manhattan', 'n_neighbors': 10, 'weights': 'distance'}

Gaussian Naive Bayes:
Fitting 10 folds for each of 100 candidates, totalling 1000 fits
Best params {'var_smoothing': 0.0015199110829529332}

Random Forest:
Fitting 10 folds for each of 24 candidates, totalling 240 fits
Best params {'max_depth': 25, 'min_samples_leaf': 1, 'min_samples_split': 3, 'n_estimators': 100}
```

Figura 7 – best_params_ di KNN, Gaussian Naïve Bayes e Random Forest

Valutazione degli algoritmi di classificazione

Nel machine learning uno dei task più importanti è quello di classificazione, ovvero identificare la classe di un nuovo obiettivo sulla KB estratta da un training set. Per i problemi di classificazione si ricorre all'apprendimento supervisionato, caratterizzato da un insieme di esempi ed un insieme di features (funzione che va dall'insieme degli esempi ad un valore), queste ultime suddivise in features di input e feature target, con l'obiettivo è predire i valori per le features target per gli esempi di test e gli esempi non ancora visti.

Per il nostro progetto, abbiamo suddiviso il dataset fissando il 70% di esso come training set e il 30% come test set. La predizione viene effettuata sull'attributo "Classification".

Sono stati messi a confronto 3 modelli di apprendimento supervisionato, richiamati con i parametri ottenuti dall'operazione precedente, per trovare quello con la migliore performance da utilizzare per la classificazione:

- K-NN
- Random Forest
- Gaussian Naive Bayes

K-NN

```
KNN:
Accuracy knn: 0.9789288849868305
              precision    recall  f1-score   support

           0       0.98      1.00      0.99       450
           1       1.00      0.95      0.97       149
           2       0.99      1.00      1.00       264
           3       0.97      0.94      0.96       170
           4       0.92      0.94      0.93       106

    accuracy                           0.98      1139
   macro avg       0.97      0.97      0.97      1139
weighted avg       0.98      0.98      0.98      1139
```

***Figura 8** – Output dell'accuracy del KNN*

Random Forest

```
Random Forest:
Accuracy rf: 1.0
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       459
           1       1.00      1.00      1.00       132
           2       1.00      1.00      1.00       280
           3       1.00      1.00      1.00       161
           4       1.00      1.00      1.00       107

    accuracy                           1.00      1139
   macro avg       1.00      1.00      1.00      1139
weighted avg       1.00      1.00      1.00      1139
```

***Figura 9** – Output dell'accuracy del Random Forest*

Gaussian Naive Bayes

```
Gaussian Naive Bayes:
Accuracy gau : 1.0
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       450
           1       1.00      1.00      1.00       149
           2       1.00      1.00      1.00       264
           3       1.00      1.00      1.00       170
           4       1.00      1.00      1.00       106

    accuracy                           1.00      1139
   macro avg       1.00      1.00      1.00      1139
weighted avg       1.00      1.00      1.00      1139
```

***Figura 10** – Output dell'accuracy del Gaussian Naive Bayes*

Conclusioni

Alla luce dell'analisi condotta, si può capire che gli algoritmi di classificazione più efficienti per il nostro progetto sono il Random Forest, con parametri "max_depth" = 15, "min_samples_leaf" = 1, "min_samples_split" = 3, "n_estimators" = 100, e il Gaussian Naive Bayes, con parametro "var_smoothing" = 0.0015199110829529332. In entrambi gli algoritmi efficienti, si ha complessità lineare (nel Gaussian Naive Bayes rispetto al numero di istanze di training e al numero di feature, mentre nel Random Forest rispetto al numero di alberi nella foresta).

Tuttavia, si è scelto di procedere utilizzando il Random Forest, in quanto il Gaussian Naive Bayes, a causa della sua rappresentazione approssimativa ("naive") del problema e dell'assunzione di indipendenza delle feature di input, risulta uno stimatore poco preciso, prevedendo spesso o probabilità 0 o probabilità 1. Il Random Forest, d'altro canto, tende a prevedere più spesso probabilità intermedie, a causa del suo funzionamento che prevede che la probabilità finale sia la media delle probabilità dei sottoinsiemi ottenuti dal bagging, prevedendo comunque un discreto numero di volte probabilità 0 o 1.

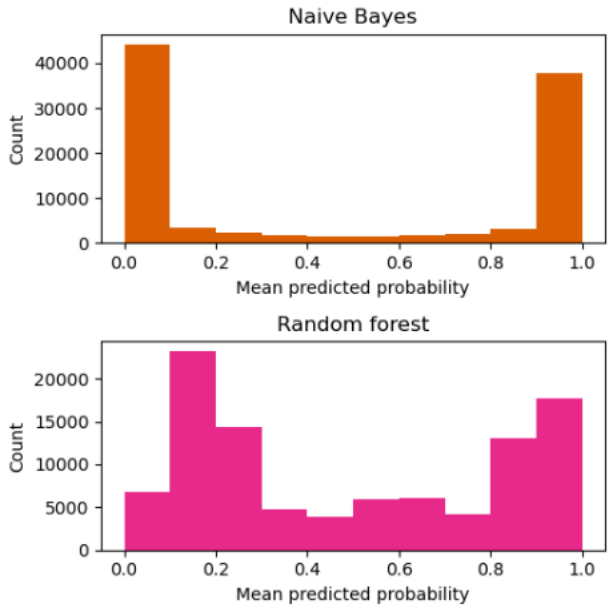

***Figura 11** – Grafico di confronto delle probabilità medie predette da Gaussian Naïve Bayes e Random Forest*

Dall'output evidenziato, inoltre, abbiamo verificato se l'eccessiva accuracy del modello potesse essere un sintomo di overfitting, che si verifica quando un modello fa previsioni basate su regolarità che compaiono negli esempi di training, ma non negli esempi di test o nel mondo da cui vengono tratti i dati. Perciò siamo andati a modificare il codice della funzione di classificazione per verificare l'accuracy sia del training set, sia del test set.

```
rf = RandomForestClassifier(max_depth=25, min_samples_leaf=1, min_samples_split=3, n_estimators=100, criterion="entropy")
rf.fit(x_train.values, y_train.values)
y_pred_rf = rf.predict(x_test.values)
y_train_rf = rf.predict(x_train.values)
print("Accuracy rf training set:", metrics.accuracy_score(y_train, y_train_rf))
print("Accuracy rf test set:", metrics.accuracy_score(y_test, y_pred_rf))
```

Figura 12 – *Codice dell'accuracy di training e test set con Random Forest*

Si è quindi, come prima cosa, nuovamente mandata in esecuzione la funzione di classificazione:

```
Random Forest:
Accuracy rf training set: 1.0
Accuracy rf test set: 1.0
```

Figura 13 – *Output dell'accuracy di training e test set con Random Forest nella funzione di classificazione*

Successivamente, si è ripetuta l'operazione nel main, fornendo in input prima un elemento presente nel dataset e poi uno esterno al dataset

```
PS C:\Users\nisci\Desktop\Progetto-ICon> & c:/Users/nisci/Desktop/Progetto-ICon/.venv/Scripts/python.exe c:/Users/nisci/Desktop/Progetto-ICon/Code/main.py
Accuracy rf training set: 0.9984939759036144
Accuracy rf test set: 0.9859525899912204
PS C:\Users\nisci\Desktop\Progetto-ICon> & c:/Users/nisci/Desktop/Progetto-ICon/.venv/Scripts/python.exe c:/Users/nisci/Desktop/Progetto-ICon/Code/main.py
Accuracy rf training set: 0.9992469879518072
Accuracy rf test set: 0.9841966637401229
```

Figura 14 – *Output dell'accuracy di training e test set con Random Forest nel main*

Pertanto, siamo giunti alla conclusione che, nel nostro caso, non si è verificato l'overfitting.

Applicazione al progetto

Dopo la serie di scelte progettuali, ecco un esempio dell'esecuzione della funzionalità di apprendimento supervisionato.

```
Vuoi che ti suggerisca un altro videogioco ? - Premi 1
Vuoi sapere la classificazione PEGI di un altro videogioco? - Premi 2
Vuoi uscire? - Premi 3
2
Qual è il nome del gioco che vuoi classificare?
fifa soccer 13
Qual è il genere del gioco che vuoi classificare?
sports
Qual è il publisher del gioco che vuoi classificare?
electronic arts
Qual è la piattaforma del gioco che vuoi classificare?
ps3
Se dovessi valutarlo, che voto gli daresti da 0 a 20?
16
Qual è l'anno di uscita del gioco che vuoi classificare?
2014
La classificazione del gioco fifa soccer 13 e':  +3
```

Figura 15 – *Esempio di esecuzione dell'algoritmo supervisionato*

Clustering & Elbow Method

Il clustering è una tecnica che partiziona gli esempi in cluster o classi. Ogni classe predice i valori delle feature per gli esempi che contiene. Ogni clustering (sistema di cluster o raggruppamento) presenta un certo errore di predizione associato. Il migliore è quello che minimizza l'errore. Gli oggetti all'interno di un cluster presentano tra loro delle similarità e, per contro, hanno delle dissimilarità con gli oggetti all'interno degli altri cluster.

Per questo progetto, si è scelto di effettuare il clustering utilizzando l'algoritmo K-Means, che suddivide gli oggetti in K cluster in base ai loro attributi. In input all'algoritmo sono dati gli esempi ed il numero di classi K. Quindi costruisce K classi, una predizione del valore di ogni feature per ogni classe ed una funzione di assegnazione degli esempi alle classi. In seguito, costruisce una nuova partizione, associando ogni punto d'ingresso al gruppo in cui il centroide è più vicino ad esso; infine vengono ricalcolati i centroidi per i nuovi gruppi e si ripete il procedimento fino a quando l'algoritmo non converge.

L'algoritmo k-Means è un algoritmo di hard clustering. Nell'hard clustering, ogni punto dati nel set di input è assegnato esclusivamente ad un singolo cluster senza alcuna ambiguità.

Per trovare il migliore K possibile per il nostro dataset, si è utilizzata la tecnica dell'Elbow Method, andando a costruire un grafico avente sull'asse delle ascisse il numero dei cluster e sull'asse delle ordinate la somma dei quadrati delle distanze tra ciascun punto dati e il centroide del cluster a cui appartiene (WCSS – Within-Cluster Sum of Squares). Il numero più efficiente di cluster è quello rappresentato dal "gomito" illustrato nel grafico.

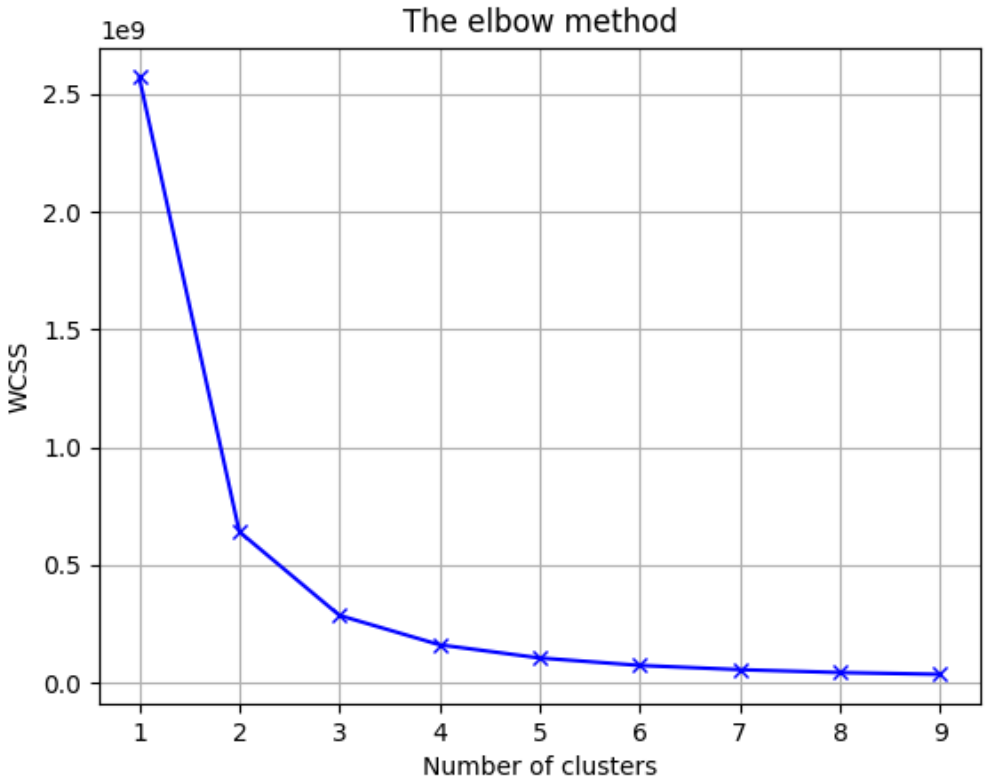

Figura 16 - Grafico Elbow Method

Dal grafico, si può notare che i punti migliori su cui applicare il K-Means sono 2, 3 e 4. Pertanto, abbiamo scelto di dividere il dataset in 3 cluster, essendo tale punto al centro del "gomito" formatosi. Tale divisione ottimizzerà di molto il funzionamento del recommender system, mostrato di seguito, ottenendo il giusto bilanciamento tra la complessità del modello (numero di cluster) e la coerenza dei dati all'interno dei cluster.

Applicazione al progetto

Difatti, si è utilizzato un recommender system per applicare l'algoritmo di hard clustering k-Means. Segue uno screenshot di un esempio dell'esecuzione di tale funzionalità.

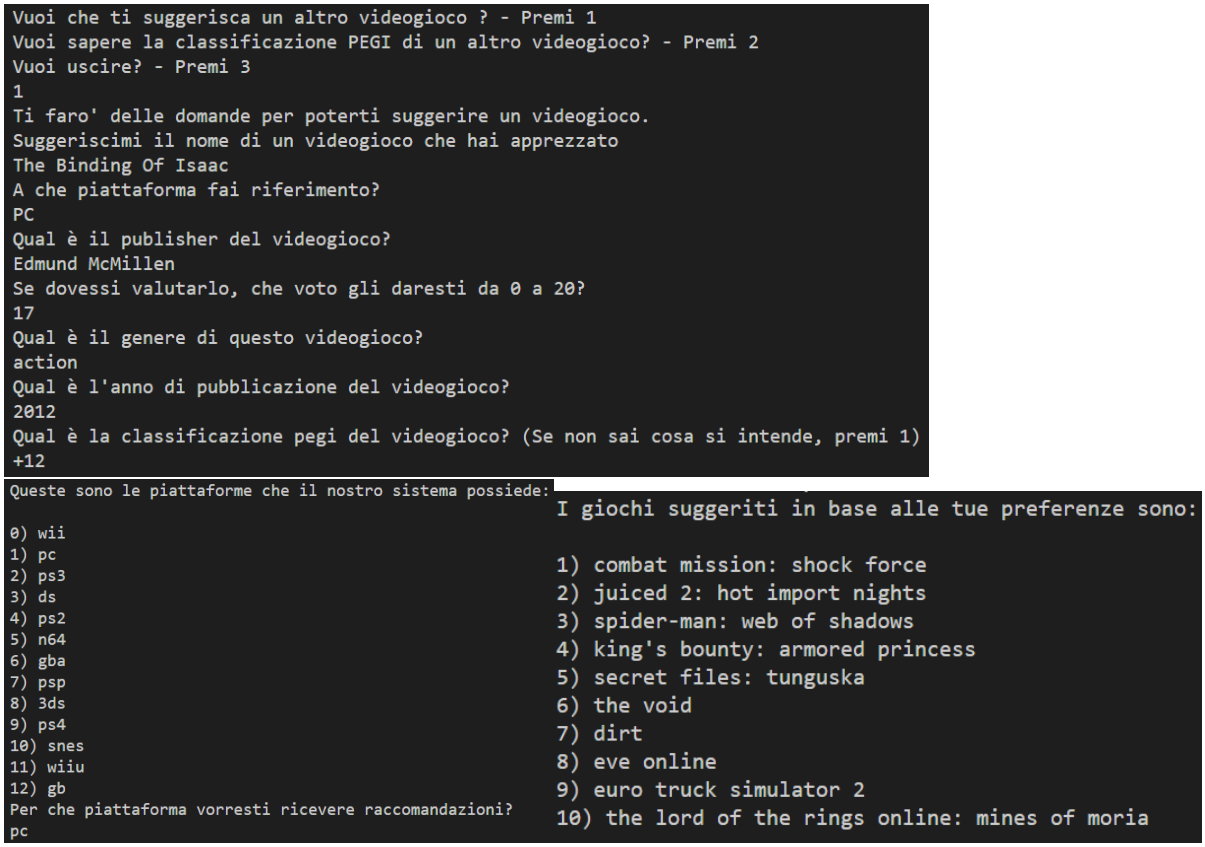

Figura 17 *– Esempio di esecuzione del recommender system che utilizza il k-Means*

Conclusioni

I requisiti iniziali preposti sono stati raggiunti con successo e il programma rispetta quanto ci si era prefissati di ottenere.

La fase di preprocessing e realizzazione della base di conoscenza ha portato ad avere un dataset adatto allo sviluppo del nostro progetto. Tuttavia, il gruppo si è reso conto a posteriori di non necessitare di Jupyter Notebook per la fase di preprocessing, in quanto, pur essendo molto comodo per la visualizzazione dei risultati di ciascun comando impartito, tale ambiente non ha memoria e quindi è necessario ri-eseguire tutti i comandi ogni volta che si riprende la codifica dello stesso file, risultando molto scomodo per le nostre esigenze. A tale disagio, si aggiunge il fatto che con Jupyter Notebook si ottengono gli stessi risultati rispetto all'utilizzo di un qualsiasi altro ambiente di sviluppo con capacità di memoria, quindi più adatto alle esigenze progettuali, come ad esempio Visual Studio Code, che permette anche operazioni interfacciate con il repository su GitHub.

L'utilizzo dell'algoritmo di apprendimento supervisionato "Random Forest" ha portato il gruppo ad ottenere un buon predittore per la classificazione PEGI di un videogioco a partire dalla base di conoscenza ottenuta nella fase precedente.
Infine, l'algoritmo di apprendimento non supervisionato "KMeans" ci ha permesso di ottimizzare il nostro recommender system dividendo l'intero dataset in più cluster, aumentando quindi la precisione del sistema.

Un'estensione possibile, attualmente non implementata per motivi di tempo, è l'ampliamento del dataset in modo da inserire più videogiochi e fornire previsioni più varie e accurate.

Un ulteriore estensione del progetto è l'inserimento di nuovi fatti all'interno della base di conoscenza con l'aggiunta di nuove regole e query per ottenere ulteriori informazioni.

Riferimenti Bibliografici

[1] David L. Poole, Alan K. Mackworth – "Artificial Intelligence"
[2] Documentazione pandas: https://pandas.pydata.org/docs/
[3] Documentazione scikit learn: https://scikit-learn.org/0.21/documentation.html