Sommario

1 Introduzione: sistema attuale e sistema proposto

Come la maggior parte dei rami del mercato, quello videoludico è, al giorno d'oggi, ampiamente diffuso online. Esistono difatti molteplici piattaforme per la vendita di prodotti inerenti al mondo dei videogiochi, alcune delle quali estremamente popolari (G2A, GameStop, InstantGaming, …). Tuttavia, nonostante la loro popolarità, la maggior parte di queste piattaforme non dispone di un sistema che riesca a rendere la loro esperienza di utilizzo da parte degli utenti realmente unica. Individuato questo gap, si è deciso di tentare di colmarlo con l'ausilio di tecniche di intelligenza artificiale.

Ci si propone, pertanto, di realizzare un sistema, GameHub, che fornisca ad ogni singolo utente un'esperienza di utilizzo davvero insostituibile, interfacciandosi ad esso prima come "gamer", e poi come "cliente".

2 Descrizione dell'agente

2.1 Obbiettivi

Lo scopo quindi del progetto, è quello di realizzare un agente intelligente che sia in grado di:

- Consigliare un insieme di prodotti personalizzato, sulla base delle proprie preferenze e di quelle di utenti affini;
- Migliorare il sistema di personalizzazione nel tempo, facendolo evolvere sulla base dei feedback degli utenti;

2.2 Specifica PEAS

Diamo ora la specifica PEAS dell'agente:

PEAS	
Performance	La misura di performance dell'agente è la sua capacità di avvicinarsi quanto più possibile ad una situazione ideale nella quale vengono mostrati agli utenti esattamente i prodotti che loro intendono acquistare.
Environment	L'ambiente in cui opera l'agente è lo spazio degli utenti del sito, con le loro preferenze, unito a quello dei possibili prodotti e le loro caratteristiche. L'ambiente è: • **Dinamico**, in quanto nel corso delle elaborazioni dell'agente, un utente potrebbe effettuare un ordine, cambiando in tal modo le sue preferenze; • **Sequenziale**, in quanto le azioni passate degli utenti influenzano le decisioni future dell'agente; • L'ambiente è **discreto**, in quanto sebbene una delle caratteristiche, il prezzo di un prodotto, possa assumere un'infinità di valori, originariamente di un insieme continuo, il prezzo viene sempre approssimato a due cifre dopo la virgola, rendendo l'infinità dei suoi possibili valori numerabile; • **Completamente osservabile**, in quanto si ha accesso a tutte le informazioni relative agli utenti ed hai prodotti in ogni momento; • **Non deterministico**, in quanto lo stato dell'ambiente cambia indipendentemente dalle azioni dell'agente; • **Non noto**, in quanto l'agente non può conoscere a priori il risultato delle proprie azioni, in termini di efficienza; • A **singolo agente**, in quanto l'unico agente che opera in questo ambiente è quello in oggetto.
Actuators	Gli attuatori dell'agente consistono nella lista dei prodotti consigliati sulla base delle preferenze, e relativa pagina web.
Sensors	I sensori dell'agente consistono nel bottone della lista dei prodotti consigliati, e pagina web relativa.

2.3 Analisi del problema

Il problema avrebbe potuto essere approcciato semplicemente creando un algoritmo che associasse ad un utente i prodotti con le caratteristiche più frequenti, tra quelli che ha acquistato. Tuttavia, questa soluzione avrebbe sofferto di diverse limitazioni:

- Sarebbe stato impossibile ottenere una personalizzazione che tenesse conto di "associazioni nascoste", presenti in utenti con preferenze simili a quello da profilare, oltre che di connotati aggiuntivi (ad esempio età, paese di provenienza, …) alle sole caratteristiche dei prodotti acquistati;
- Su un numero di prodotti acquistato elevato, il calcolo sarebbe stato molto oneroso dal punto di vista temporale, e d'altra parte, se si fossero mantenute tutte le caratteristiche dei prodotti acquistati questo avrebbe comportato un costo eccessivo in termini di memoria;

Pertanto, è stato deciso di affrontare il problema in esame mediante tecniche di **machine learning** che permettessero di effettuare la cosiddetta **user segmentation**. Non avendo a disposizione alcun tipo di etichetta prestabilita da associare ai gruppi di utenti da creare, si è deciso di interpretare il problema in oggetto come un problema di **clustering**.

L'idea alla base è stata quindi quella di suddividere gli utenti in gruppi con preferenze simili, denotare le caratteristiche dei prodotti più comuni in ogni gruppo, e consigliare dei prodotti ad ogni utente in base a quelle del suo gruppo di appartenenza.

3 Raccolta, analisi e preprocessing dei dati

3.1 Scelta del dataset

Venendo al dataset necessario per la creazione del modello di machine learning, le possibili strade da seguire erano due:

1. **Creare** un dataset da zero, formulando un questionario da far riempire ad un campione di persone, per poterne carpire le preferenze;
2. **Cercare** sulla rete un dataset già formato, e adeguarlo alle nostre esigenze;

La prima opzione era soggetta a limitazioni non trascurabili:

- Scarsità di dati
- Possibile inconsistenza dei dati, dovuta alle risposte degli utenti non sempre veritiere

Si è deciso, pertanto, di procedere con la seconda soluzione, cercando in rete un dataset già creato. Dopo svariati tentativi, la nostra attenzione si è posata su un dataset proveniente da una delle piattaforme di gaming più popolari che esistano: **Steam**.

3.2 Analisi e scrematura del dataset

Il dataset in questione, reperibile a questo link, proviene da un **dump** effettuato sul database pubblico della piattaforma nel 2016, ed ha una dimensione compressa di circa 17GB, e di circa 200GB decompressa. Il dataset in questione, oltre ad essere di una dimensione eccessiva per gli scopi del progetto, e che va molto aldilà delle possibilità di calcolo dei sistemi a nostra disposizione, presentava inoltre moltissime features, di cui la maggior parte non significative per gli scopi da raggiungere. Per tali ragioni, sono state effettuate delle operazioni di taglio orizzontale sull'insieme di dati, estraendo un piccolo sottoinsieme di circa 3200 utenti, con le loro relative preferenze, e diverse operazioni di taglio verticale, per eliminare le features non rilevanti.

Per una descrizione più dettagliata delle features di cui il sistema GameHub tiene traccia, ci si può riferire alla sezione “Gestione dei dati persistenti” del System Design Document del sistema.

3.2.1 Tabella Achievement_Percentages

ACHIEVEMENT_PERCENTAGES

This table was obtained through the Web API [ISteamUserStats::GetGlobalAchievementPercentagesForApp]. It contains achievmeent completion data for all the products listed in the App_ID_Info table.

appid	The ID of the game in question
Name	The name of the achievement as it appears to players. As an internal value assigned by developers, its descriptiveness of the achievement varies.
Percentage	The percentage of players who have finished this achievement out of all total players who own this game.

Questa tabella è stata scartata in quanto nel sistema non è presente nessun sistema di “achievement”.

3.2.2 Tabella App_ID_Info

APP_ID_INFO

This table was derived from scanning the steam storefront by emulating the REST API calls issued by the Steam client's "Big Picture mode". It contains selected information for each product ("app") offered on Steam. An older version of this table from the time of our first data pull can be found with an "_Old" suffix.

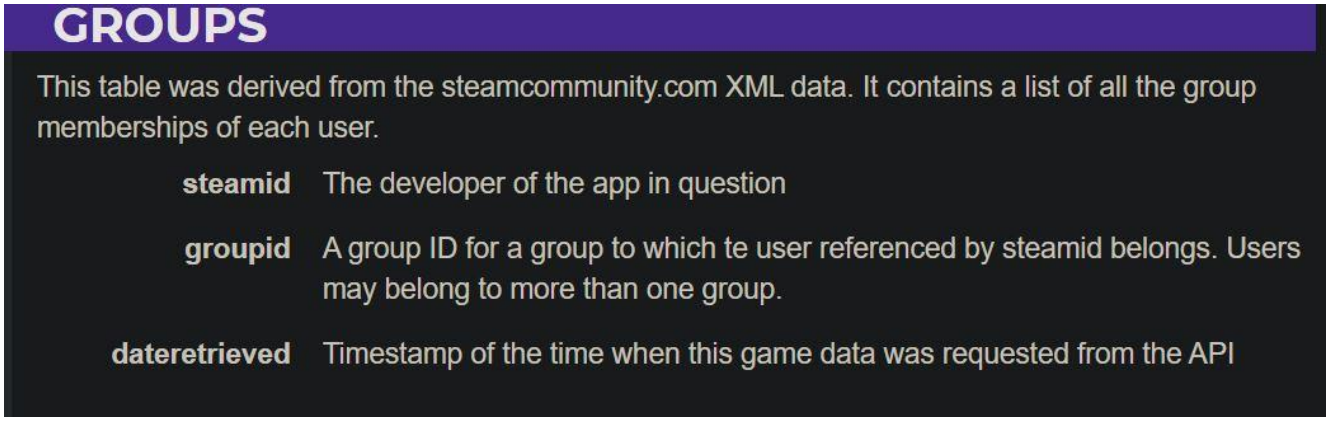
appid	The ID of the "app" in question, which is not necessarily a game.
Title	The Title of the app, as it appears to users
Type	The type of the "app". Possible values include: "demo," "dlc," "game," "hardware," "mod," and "video." Game is the most common.
Price	The current price of the "app" on the Steam storefront, in US dollars. Free items have a price of 0.
Release_Date	The date the "app" was made available via the Steam storefront. Note that apps released elsewhere originally and later published through steam carry the date of the Steam publish
Rating	The rating of the "app" on Metacritic. Set to -1 if not applicable.
Required_Age	The MSRB or PEGI-assigned age requirement for viewing this game in the Steam storefront, and, by extension, clicking the button to purchase it.
Is_Multiplayer	A value of either 0 or 1 indicating whether or not an "app" contains multiplayer content. Self-reported by developers.

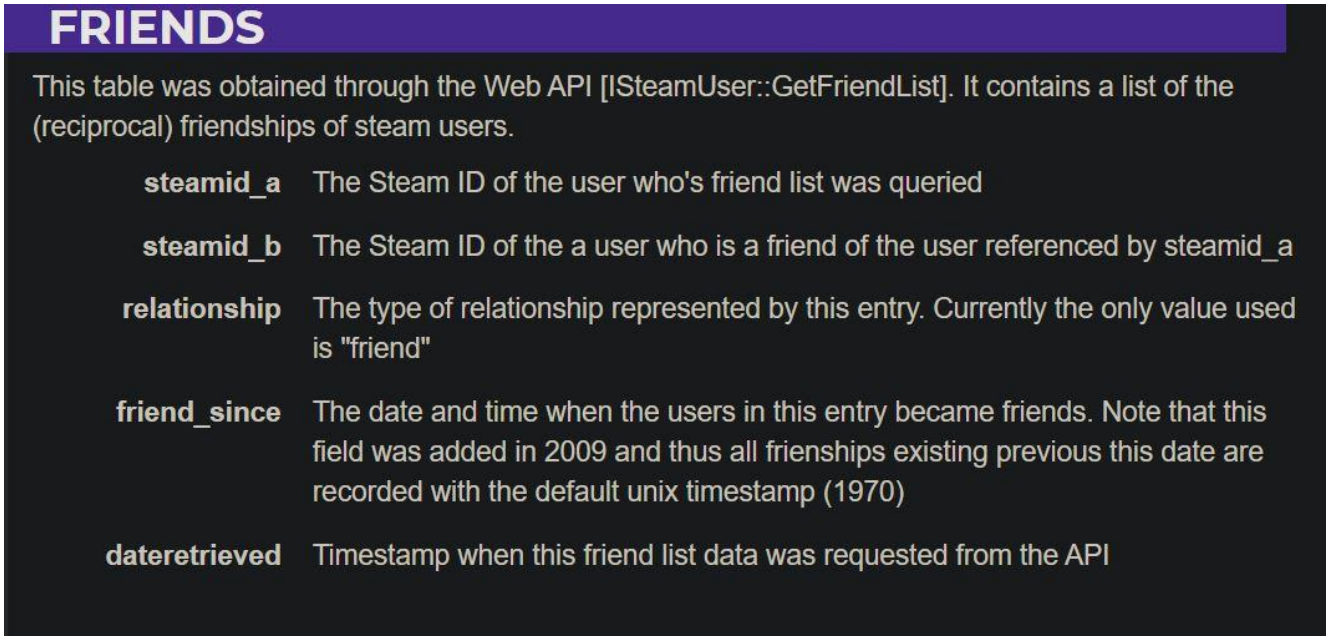
Di questa tabella, si sono considerati tutti i campi, ad eccezione di “Rating”, ma con due fondamentali differenze:

- Type è stato associato alle “Categorie” di prodotti presenti nel sistema GameHub
- Is_Multiplayer è stato associato ad uno dei “Tag” di prodotti presenti nel sistema GameHub

Si noti che il campo “Release_Date”, benché sia indicato, non è tra quelli presenti realmente nel dataset; si tratta infatti di un errore di documentazione.

3.2.3 Tabelle Friends e Groups





Queste tabelle sono state scartate in quanto il sistema GameHub non tiene traccia dei rapporti tra gli utenti.

3.2.4 Tabelle Games_1, Games_2 e Games_Daily

GAMES_1

This table was obtained through the Web API [IPlayerService::GetOwnedGames]. It contains the game data requested during our initial crawl of the Steam network.

steamid	The steam ID of the user in question
appid	The ID of a given app in the user's library
playtime_2weeks	The total time the user has run this app in the two-week period leading up to when this data was requested from the API. Values are given in minutes.
playtime_forever	The total time the user has run this app since adding it to their library. Values are given in minutes.
dateretrieved	Timestamp of the time when this game data was requested from the API

GAMES_2

This table was obtained through the Web API [IPlayerService::GetOwnedGames]. It contains the game data requested during our follow-up crawl of the Steam network.

steamid	The steam ID of the user in question
appid	The ID of a given app in the user's library
playtime_2weeks	The total time the user has run this app in the two-week period leading up to when this data was requested from the API. Values are given in minutes.
playtime_forever	The total time the user has run this app since adding it to their library. Values are given in minutes.
dateretrieved	Timestamp of the time when this game data was requested from the API

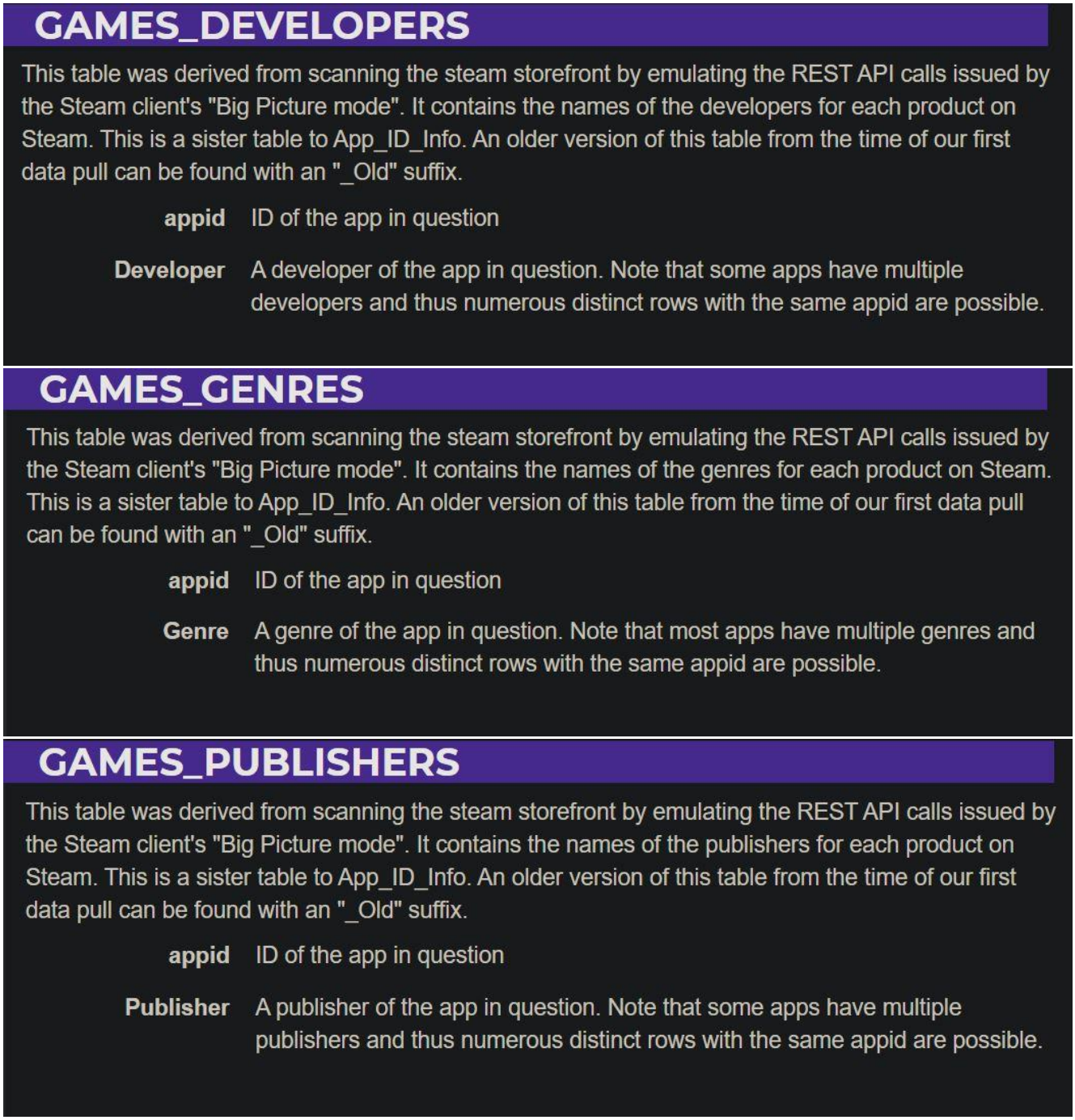
GAMES_DAILY

This table was obtained through the Web API [IPlayerService::GetOwnedGames]. It contains the game playing data for a select subset of users. Each user's data in the subset was requested repeatedly, every day for five days.

steamid	The steam ID of the user in question
appid	The ID of a given app in the user's library
playtime_2weeks	The total time the user has run this app in the two-week period leading up to when this data was requested from the API. Values are given in minutes.
playtime_forever	The total time the user has run this app since adding it to their library. Values are given in minutes.
dateretrieved	Timestamp of the time when this game data was requested from the API

Di queste tabelle, sono stati considerati i campi “steamid” e “appid”, che indicano la relazione tra il giocatore (steamid) ed il gioco (appid), mentre gli altri campi sono stati scartanti in quanto rappresentano informazioni di cui il sistema GameHub non tiene traccia. Si è considerata solo la tabella “Game_2” in quanto, rispetto a “Games_1”, contiene informazioni più aggiornate, mentre rispetto a “Games_Daily” contiene un numero di informazioni maggiore.

3.2.5 Tabelle Games_Developers, Games_Genres e Games_Publishers



Queste tabelle rappresentano le relazioni tra la tabella dei giochi e quelle degli sviluppatori, dei generi e dei publishers. Sono pertanto state considerate interamente.

3.2.6 Tabella Player_Summaries

PLAYER_SUMMARIES

This table obtained through the Web API [ISteamUser::GetPlayerSummaries]. It contains a profile summary for each Steam user.

steamid	The Steam ID of the user in question
lastlogoff	Timestamp of the time when this game data was requested from the API
primaryclanid	The groupid (Groups::groupid) of the group that the user has designated as their primary group
timecreated	Timestamp of the time when the account was created
gameid	If the user was in-game at the time of the API request, this value specifies which game they were running at the time
gameserverip	If the user was in-game at the time of the request, and playing a game using Steam matchmaking, this value specifies the IP of the server they were connected to. Is otherwise set to "0.0.0.0:0"
loccountrycode	ISO-3166 code for the country in which the user resides. Self-reported.
locstatecode	State where the user resides. Self-reported.
loccityid	Internal Steam ID corresponding to the city where the user resides. Self-reported.
dateretrieved	Timestamp of the time when this game data was requested from the API

Di questa tabella sono stati scartati i seguenti campi:

- lastlogoff, primaryclanid, timecreated, gameid e gameserverrip, in quanto non rilevanti ai fini del nostro sistema;
- locstatecode e locity code, in quanto costituiti da codici interni di Steam, per noi irrilevanti

3.3 Formattazione dei dati

Effettuati i tagli verticali ed orizzontali del dataset originario, è iniziato il processo di trasformazione dei dati in una forma che potesse essere più adatta ad un modello di apprendimento. La formattazione dei dati è risultata particolarmente complessa, ed è stata, pertanto, effettuata su più livelli ed in più fasi.

3.3.1 Livello I: Aggregazione delle informazioni

A tale scopo, si è deciso di aggregare tutte le feature provenienti da tabelle differenti in un'unica tabella, che raccoglie una riga per ogni utente, associandolo a tutte le caratteristiche dei prodotti che egli ha acquistato, oltre che al loro prezzo medio. Ove necessario, ad esempio nel caso in cui un utente abbia acquistato più prodotti, con più caratteristiche, è stato necessario concatenare tali caratteristiche in un'unica stringa.

In particolare, tutti i "Tag" sono stati raccolti sotto il nome di "Genres", le "categorie" sotto "Categories", gli sviluppatori sotto "Developers", e i publisher sotto "Publishers". Si è inoltre reso necessario eliminare eventuali ripetizioni, con l'ausilio della funzione user-defined "get_unique_items()". I dati così formattati sono stati posti sotto forma di viste interrogabili, in particolare:

- **DataSetProduct** rappresenta un prodotto associato a tutte le sue caratteristiche;
- **DataSetSample** rappresenta un utente, associato a tutte le caratteristiche dei prodotti che ha acquistato, e rappresenta pertanto il nostro campione;

```
create view DataSetProduct as
select distinct A.appid, A.Title, A.Type, A.Price,
                A.Required_Age, A.Is_Multiplayer,
                GROUP_CONCAT(DISTINCT GG.Genre) as Genres,
                GROUP_CONCAT(DISTINCT GP.Publisher) as Publishers,
                GROUP_CONCAT(DISTINCT GD.Developer) as Developers
from app_id_info A, games_genres GG, games_publishers GP,
     games_developers GD
where A.appid=GG.appid AND GG.appid=GP.appid AND GD.appid=GP.appid
group by A.appid, A.Title, A.Type, A.Price, A.Release_Date,
         A.Required_Age, A.Is_Multiplayer;

select * from DataSetProduct;

SET group_concat_max_len = 4_000_000;
drop view if exists DataSetSample;
create view DataSetSample as
    select distinct P.steamid, P.personaname, P.personastate,
                    P.communityvisibilitystate, P.profilestate,
                    P.cityid, P.loccountrycode, P.loccityid,
                    AVG(A.Price) as avgPrice,
                    MAX(A.Required_Age) as maxRequiredAge,
                    MAX(A.Is_Multiplayer) as usesMultiplayer,
                    get_unique_items( str: GROUP_CONCAT(DISTINCT A.Genres)) as Genres,
                    get_unique_items( str: GROUP_CONCAT(DISTINCT A.Publishers)) as Publishers,
                    get_unique_items( str: GROUP_CONCAT(DISTINCT A.Developers)) as Developers,
                    GROUP_CONCAT(DISTINCT A.Type) as Categories
    from DataSetProduct A, games_2 G, player_summaries P
    where A.appid=G.appid AND G.steamid=P.steamid
    group by P.steamid, P.personaname, P.personastate,
             P.communityvisibilitystate, P.profilestate,
             P.cityid, P.loccountrycode, P.loccityid;
```

3.3.2 Livello II: Trasposizione su CSV e codifica

Una volta aggregati i dati in un'unica tabella, si è reso necessario codificarli, trasformando i dati dalla forma testuale a quella numerica, per renderli comprensibili ad un algoritmo di clustering. Per la codifica dei dati, si è scelto di utilizzare la **One-Hot-Encode**, sebbene non sia efficiente dal punto di vista della memoria, in quanto i dati non avevano una semantica ordinale o che permettesse di applicare qualunque altro tipo di codifica. A tale scopo, i dati aggregati sono stati esportati in formato CSV, per permettere agli script Python, che si occuperanno della parte rimanente della formattazione, di accedervi con l'ausilio della libreria **pandas**.

```
def main():
    # Open file .csv
    dataFrame = pd.read_csv('../../Desktop/datasetRaw.csv')

    # Remove useless columns
    dataFrame.drop(columns=['communityvisibilitystate'], axis=1, inplace=True)
    dataFrame.drop(columns=['profilestate'], axis=1, inplace=True)
    dataFrame.drop(columns=['cityid'], axis=1, inplace=True)
    dataFrame.drop(columns=['loccityid'], axis=1, inplace=True)
    dataFrame.drop(columns=['personastate'], axis=1, inplace=True)

    # Rename columns
    dataFrame.rename(columns={'steamid': 'Id', 'personaname': 'Username', 'loccountrycode': 'Country'}, inplace=True)

    # Random country code generator
    encodeCountry(dataFrame)

    # Retrieve values to format
    categories = dataFrame['Categories']
    genres = dataFrame['Genres']
    developers = dataFrame['Developers']
    publishers = dataFrame['Publishers']
    countries = dataFrame['Country']

    # Create multilabelbinarizer
    one_hot = MultiLabelBinarizer()
```

```
    # Merge tables in to one
    result = pd.merge(dataFrame, resultCategories, on=dataFrame.index)
    result.drop(columns=['key_0', 'Categories'], axis=1, inplace=True)

    result = pd.merge(result, resultGenres, on=dataFrame.index)
    result.drop(columns=['key_0', 'Genres'], axis=1, inplace=True)

    result = pd.merge(result, resultDevelopers, on=dataFrame.index)
    result.drop(columns=['key_0', 'Developers'], axis=1, inplace=True)

    result = pd.merge(result, resultPublishers, on=dataFrame.index)
    result.drop(columns=['key_0', 'Publishers'], axis=1, inplace=True)

    result = pd.merge(result, resultCountries, on=dataFrame.index)
    result.drop(columns=['key_0', 'Country'], axis=1, inplace=True)

    # Save new dataset
    result.to_csv('./datasetOfficial.csv')
```

```
# Settings option to display dataset without '...'
# pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)
pd.set_option('display.width', None)
pd.set_option('display.max_colwidth', -1)

# Format values in to One Hot Encode
resultCategories = pd.DataFrame(one_hot.fit_transform(categories.dropna().str.split(',')), columns=one_hot.classes_,
                                index=dataFrame.index)
resultCategories = resultCategories.add_prefix('Ctg_')

resultGenres = pd.DataFrame(one_hot.fit_transform(genres.dropna().str.split(',')), columns=one_hot.classes_,
                            index=dataFrame.index)
resultGenres = resultGenres.add_prefix('Tag_')

resultDevelopers = pd.DataFrame(one_hot.fit_transform(developers.dropna().str.split(',')), columns=one_hot.classes_,
                                index=dataFrame.index)
resultDevelopers = resultDevelopers.add_prefix('Dvp_')

resultPublishers = pd.DataFrame(one_hot.fit_transform(publishers.dropna().str.split(',')), columns=one_hot.classes_,
                                index=dataFrame.index)
resultPublishers = resultPublishers.add_prefix('Pbl_')

resultCountries = pd.DataFrame(one_hot.fit_transform(countries.dropna().str.split(' ')), columns=one_hot.classes_,
                               index=dataFrame.index)
```

Lo script Python che si occupa della formattazione è suddiviso in tre fasi:

1. Rimozione delle colonne indesiderate, e irrilevanti ai fini del clustering, come l'id o lo username e ridenominazione delle colonne rimanenti;
2. Codifica dei dati in formato One-Hot-Encode attraverso la classe MultiLabelBinazer;
3. Applicazione dei cambiamenti fondendo tutto in un unico data frame, e salvataggio di quest'ultimo su file CSV;

3.3.3 Estrazione dei dati dal database

Una volta posto il dataset in una forma più congeniale alla creazione del modello di machine learning, si è posto il problema dei nuovi campioni da dare in input al modello, in modo tale che questo potesse inserirli in uno dei gruppi già creati, e restituire una lista di caratteristiche per poter costruire una lista di prodotti consigliati.

Ovviamente, i nuovi dati da "clusterizzare" dovevano derivare, in un modo o nell'altro dal database del sistema GameHub. Si è posto quindi un problema simile a quello che abbiamo affrontato nei punti 3.3.1 e 3.3.2, applicato però ad il database del sistema.

C'era dunque la necessità di raggruppare tutti i dati relativi ad un utente e quelli relativi ad i prodotti che ha acquistato, in un'unica tabella. Anche qui, ove necessario, ad esempio nel caso in cui un utente abbia acquistato più prodotti, con più caratteristiche, è stato necessario concatenare tali caratteristiche in un'unica stringa, ed eliminare eventuali ripetizioni. Si è pertanto realizzato un insieme di viste che, utilizzate in cascata, adempiono al compito:

- **JoinedPhysicalProduct** e **JoinedDigitalProduct** rappresentano i prodotti uniti alla lista delle loro caratteristiche (“Tag” e “Categorie”);
- **DatasetSampleDigital** e **DatasetSamplePhysical** rappresentano gli utenti che hanno effettuato rispettivamente ordini di prodotti fisici e ordini di prodotti digitali, uniti alle caratteristiche di tali prodotti;
- **DatasetSample** è invece l’unione di DatasetSampleDigital e DatasetSamplePhysical con l’insieme degli utenti che hanno effettuato ordini di entrambe le tipologie di prodotto, unito alle caratteristiche dei prodotti acquistati. Ogni riga rappresenta un campione che può essere dato in input al modello;

```sql
drop view if exists JoinedDigitalProduct;
create view JoinedDigitalProduct as
    select distinct D.id, D.name, D.publisher, D.softwareHouse, D.requiredAge, D.price,
                    GROUP_CONCAT(DISTINCT DC.tag) as Tags,
                    GROUP_CONCAT(DISTINCT DB.category) as Categories
    from digitalproduct D, digitalcharacteristic DC, digitalbelonging DB
    where D.id=DC.digitalProduct AND D.id=DB.digitalProduct
    group by D.id, D.name, D.publisher, D.softwareHouse, D.requiredAge, D.price;

select * from JoinedDigitalProduct;

drop view if exists JoinedPhysicalProduct;
create view JoinedPhysicalProduct as
    select distinct P.id, P.name, P.price,
                    GROUP_CONCAT(DISTINCT PC.tag) as Tags,
                    GROUP_CONCAT(DISTINCT PB.category) as Categories
    from physicalproduct P, physicalcharacteristic PC, physicalbelonging PB
    where P.id=PC.physicalProduct AND P.id=PB.physicalProduct
    group by P.id, P.name, P.price;

select * from JoinedPhysicalProduct;
```

```sql
/*Campione di un utente con le caratteristiche dei prodotti fisici dei suoi ordini,
  escludendo gli utenti fittizi, che hanno una '@' nello username.
*/
drop view if exists DataSetSamplePhysical;
create view DataSetSamplePhysical as
    select distinct U.username as Username, U.country as Country,
                    (AVG(JPP.price)) as avgPrice,
                    get_unique_items( str: GROUP_CONCAT(DISTINCT JPP.Tags)) as Tags,
                    get_unique_items( str: GROUP_CONCAT(DISTINCT JPP.Categories)) as Categories
    from JoinedPhysicalProduct JPP, `order` O, physicalpurchasing PP, user U
    where U.username=O.user AND O.id=PP.`order` AND PP.physicalProduct=JPP.id AND U.username not like '%@%'
    group by U.username, U.country;

select * from DataSetSamplePhysical;

/*Campione di un utente con le caratteristiche dei prodotti digitali dei suoi ordini,
  escludendo gli utenti fittizi, che hanno una '@' nello username.
*/
drop view if exists DataSetSampleDigital;
create view DataSetSampleDigital as
    select distinct U.username as Username, U.country as Country,
                    (AVG(JDP.price)) as avgPrice,
                    MAX(JDP.requiredAge) as maxRequiredAge,
                    get_unique_items( str: GROUP_CONCAT(DISTINCT JDP.Tags)) as Tags,
                    get_unique_items( str: GROUP_CONCAT(DISTINCT JDP.Categories)) as Categories,
                    GROUP_CONCAT(DISTINCT JDP.softwareHouse) as Developers,
                    GROUP_CONCAT(DISTINCT JDP.publisher) as Publishers
    from JoinedDigitalProduct JDP, `order` O, digitalpurchasing DP,  user U
    where U.username=O.user AND (O.id=DP.`order` AND DP.digitalProduct=JDP.id) AND U.username not like '%@%'
    group by U.username, U.country;
```

```
/*Creiamo la vista dei campioni del dataset, facendo la join full unendo la left e la right,
  per garantire che vengano considerate sia le persone che hanno fatto ordini su prodotti fisici,
  sia quelle sui prodotti digitali, sia quelle che li hanno fatti su entrambi.
*/
drop view if exists DataSetSample;
create view DataSetSample as
    select *
    from (
        select distinct DSSP.Username as Username, DSSP.Country as Country,
                    (DSSP.avgPrice + DSSD.avgPrice)/2 as avgPrice,
                    DSSD.maxRequiredAge as maxRequiredAge,
                    get_unique_items(
                            str: GROUP_CONCAT(distinct concat(DSSD.Tags, ',', DSSP.Tags))
                        ) as Tags,
                    get_unique_items(
                            str: GROUP_CONCAT(distinct concat(DSSD.Categories, ',', DSSP.Categories))
                        ) as Categories,
                    DSSD.Developers as Developers,
                    DSSD.Publishers as Publishers
        from DataSetSamplePhysical DSSP join DataSetSampleDigital DSSD on (DSSP.Username=DSSD.Username)
        group by DSSP.Username, DSSP.Country
        UNION
        select distinct DSSP.Username as Username, DSSP.Country as Country,
                        (DSSP.avgPrice)/2 as avgPrice,
                        DSSD.maxRequiredAge as maxRequiredAge,
                        get_unique_items(
                             str: GROUP_CONCAT(distinct DSSP.Tags)
                        ) as Tags,
                        get_unique_items(   Attornado, 10/01/2021 21:43 • Correzioni alle classi di testing delle servlet;
                             str: GROUP_CONCAT(distinct DSSP.Categories)
                        ) as Categories,
                        DSSD.Developers as Developers,
                        DSSD.Publishers as Publishers
        from DataSetSamplePhysical DSSP left join DataSetSampleDigital DSSD on (DSSP.Username=DSSD.Username)
        group by DSSP.Username, DSSP.Country
        UNION
        select distinct DSSD.Username, DSSD.Country as Country,
                        (DSSD.avgPrice) as avgPrice,
                        DSSD.maxRequiredAge as maxRequiredAge,
                        get_unique_items(
                             str: GROUP_CONCAT(distinct DSSD.Tags)
                        ) as Tags,
                        get_unique_items(
                             str: GROUP_CONCAT(distinct DSSD.Categories)
                        ) as Categories,
                        DSSD.Developers as Developers,
                        DSSD.Publishers as Publishers
        from DataSetSamplePhysical DSSP right join DataSetSampleDigital DSSD on (DSSP.Username=DSSD.Username)
        group by DSSD.Username, DSSD.Country, DSSD.avgPrice, DSSD.Developers, DSSD.Publishers
        ) as SubQuery
    group by Username;
```

3.3.4 Estrazione dei dati dal database

Si può notare come la strategia applicata per l'aggregazione di questi dati sia molto simile a quella attuata nel dataset originario, il che ha permesso di riutilizzare, con gli opportuni cambiamenti, anche lo script di formattazione.

Per formattare e codificare i dati, ancora una volta il risultato della vista è stato posto su un file CSV, chiamato **buffer.csv**, dove verrà scritto un campione alla volta per poter essere analizzato dagli script Python, attraverso Pandas. In particolare, è stata realizzata una funzione **format()**, posta nel modulo **samplePrediction**, che ha lo scopo di rimuovere i dati non necessari e formattarli con la One-Hot-Encode:

```
def format(votes=False):
    # Caricamento dataset iniziale e campione da clusterizzare

    if(votes):
        buffer = pd.read_csv(clustering.VOTES_FILENAME)
    else:
        buffer = pd.read_csv(clustering.BUFFER_FILENAME)
    df = pd.read_csv(clustering.DATASET_FILENAME)
    official = df.columns

    # Filtraggio delle colonne sulle quali eseguire la one hot encode
    categories = buffer['Categories']
    genres = buffer['Tags']
    developers = buffer['Developers']
    publishers = buffer['Publishers']
    countries = buffer['Country']

    # Esecuzione della one hot encode
    one_hot = MultiLabelBinarizer()

    resultCategories = pd.DataFrame(one_hot.fit_transform(categories.dropna().str.split(',')),
                                    columns=one_hot.classes_, index=buffer.index)
    resultCategories = resultCategories.add_prefix('Ctg_')

    resultGenres = pd.DataFrame(one_hot.fit_transform(genres.dropna().str.split(',')),
                                columns=one_hot.classes_, index=buffer.index)
    resultGenres = resultGenres.add_prefix('Tag_')

    resultDevelopers = pd.DataFrame(one_hot.fit_transform(developers.dropna().str.split(',')),
                                    columns=one_hot.classes_, index=buffer.index)
    resultDevelopers = resultDevelopers.add_prefix('Dvp_')
```

```
resultPublishers = pd.DataFrame(one_hot.fit_transform(publishers.dropna().str.split(',')),
                                columns=one_hot.classes_, index=buffer.index)
resultPublishers = resultPublishers.add_prefix('Pbl_')

resultCountries = pd.DataFrame(one_hot.fit_transform(countries.dropna().str.split(' ')),
                               columns=one_hot.classes_, index=buffer.index)

# Merge dei risultati ottenuti in un unico dataframe e rimozione delle colonne inutilizzate
result = pd.merge(buffer, resultCategories, on=buffer.index)
result.drop(columns=['key_0', 'Categories'], axis=1, inplace=True)

result = pd.merge(result, resultGenres, on=buffer.index)
result.drop(columns=['key_0', 'Tags'], axis=1, inplace=True)

result = pd.merge(result, resultDevelopers, on=buffer.index)
result.drop(columns=['key_0', 'Developers'], axis=1, inplace=True)

result = pd.merge(result, resultPublishers, on=buffer.index)
result.drop(columns=['key_0', 'Publishers'], axis=1, inplace=True)

result = pd.merge(result, resultCountries, on=buffer.index)
result.drop(columns=['key_0', 'Country'], axis=1, inplace=True)

# Aggiunta delle colonne non presenti nel campione, ma nel dataset
# Mappa per tenere traccia di eventuali nuove colonne presenti all'interno del
#  campione, ma non nel dataset
map = {}
oldColumns = result.columns
for header in official:
    if header not in oldColumns:
        result[header] = 0
    else:
        map[header] = True
```

```python
    # Salviamo il risultato con le nuove colonne, per salvarle nel dataset(in caso di voto)
    if(votes):
        resultWithOldColumns = result.drop(axis=1, columns=['Vote'])

    # Aggiunta di eventuali nuove colonne al dataframe e rimozione
    # di esse dal campione per permettere la clusterizzazione
    for c in oldColumns:
        if c not in map:
            result = result.drop(c, axis=1)
            df[c] = 0

    # Rimozione di eventuali colonne "Unnamed"
    df = df.filter(regex="^(?!Unnamed)", axis=1)

    if(votes):
        # Aggiungiamo i voti al dataset aggiornato
        df = pd.concat([df, resultWithOldColumns])

    # Salvataggio del dataset aggiornato in un file temporaneo
    df.to_csv(clustering.DATASET_TEMP_FILENAME)
    return result
```

Lo script Python che si occupa della formattazione dei nuovi campioni, è anch'esso suddiviso in quattro fasi:

1. Rimozione delle colonne indesiderate, e irrilevanti ai fini del clustering, come l'id o lo username e ridenominazione delle colonne rimanenti;
2. Codifica dei dati in formato One-Hot-Encode attraverso la classe MultiLabelBinazer;
3. Rimozione di eventuali colonne "unnamed" create nel corso delle operazioni ed aggiunta di eventuali nuove colonne che corrispondono ad eventuali nuove categorie/tag/sviluppatori/publisher che potrebbero essere stati aggiunti al sito, e salvataggio del dataset con le nuove features aggiunte in un file temporaneo, chiamato **datasetTemp.csv**. Le nuove features così create, nel dataset "aggiornato", sui vecchi campioni vengono sempre poste a 0;
4. Restituzione del campione formattato;

Si noti come è stata volutamente ignorata, nella spiegazione dello script, la presenza del parametro booleano "votes", nella funzione format(), che ha un ruolo nell'ambito dell'evoluzione del modello e dell'implementazione del sistema di feedback. Verrà, per tale ragione, affrontato nella sezione dedicata.

4 Algoritmo di clustering

4.1 Scelta dell'algoritmo di clustering

Una volta completata la formattazione dei dati, non rimaneva che scegliere l'algoritmo di clustering e scrivere lo script che lo richiamasse. Analizzando il nostro dataset, abbiamo ritenuto che potesse essere particolarmente denso, vista la presenza di numerose features di valore uguale in moltissimi campioni.

4.1.1 DBScan

La scelta dell'algoritmo di clustering sembrava a quel punto automatica: **DBscan**. L'algoritmo risulta infatti particolarmente adatto a dataset densi e, come ci si aspettava, l'algoritmo ha avuto ottime prestazioni.

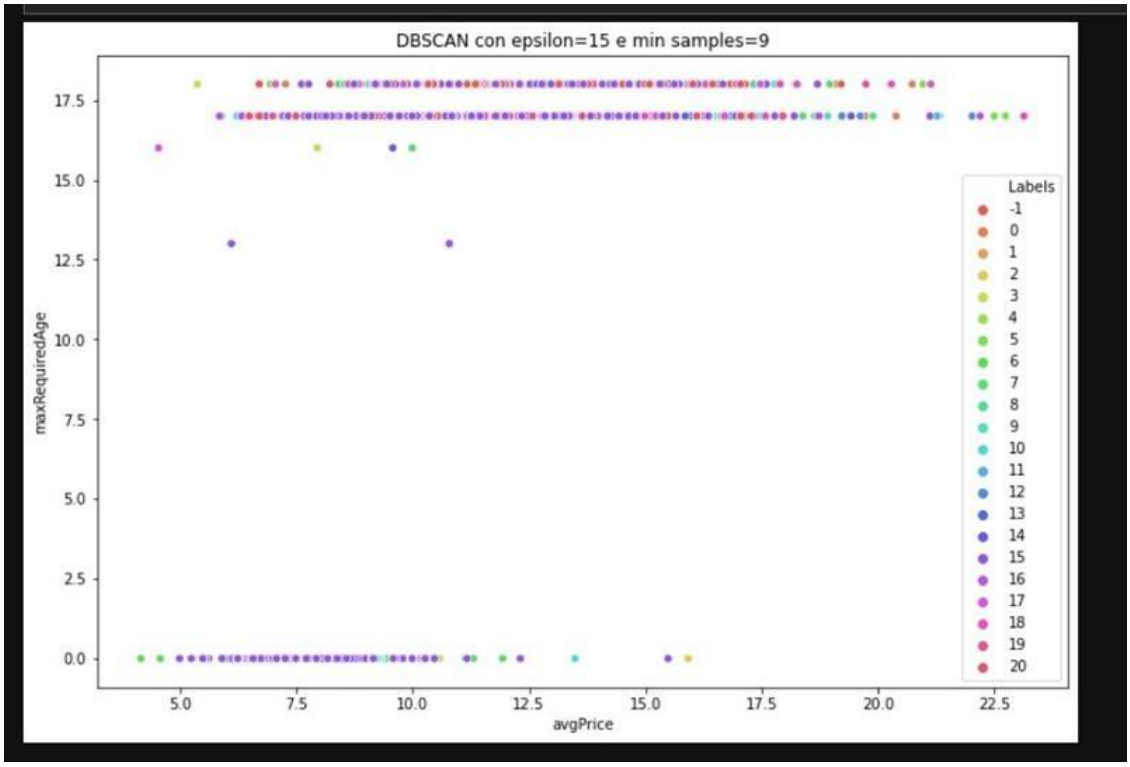
Lo script che applica il DBScan è estremamente semplice:

1. Legge il file CSV del dataset ed elimina le colonne irrilevanti (Id utente e Username);
2. Effettua la chiamata a DBScan, in questo caso con Ɛ=15 e minPts=9 (successivamente sono stati effettuati tentativi anche con Ɛ=16 e Ɛ=16.7). Il minPts è stato scelto secondo l'euristica di dare a tale parametro un valore pari almeno al numero di features;

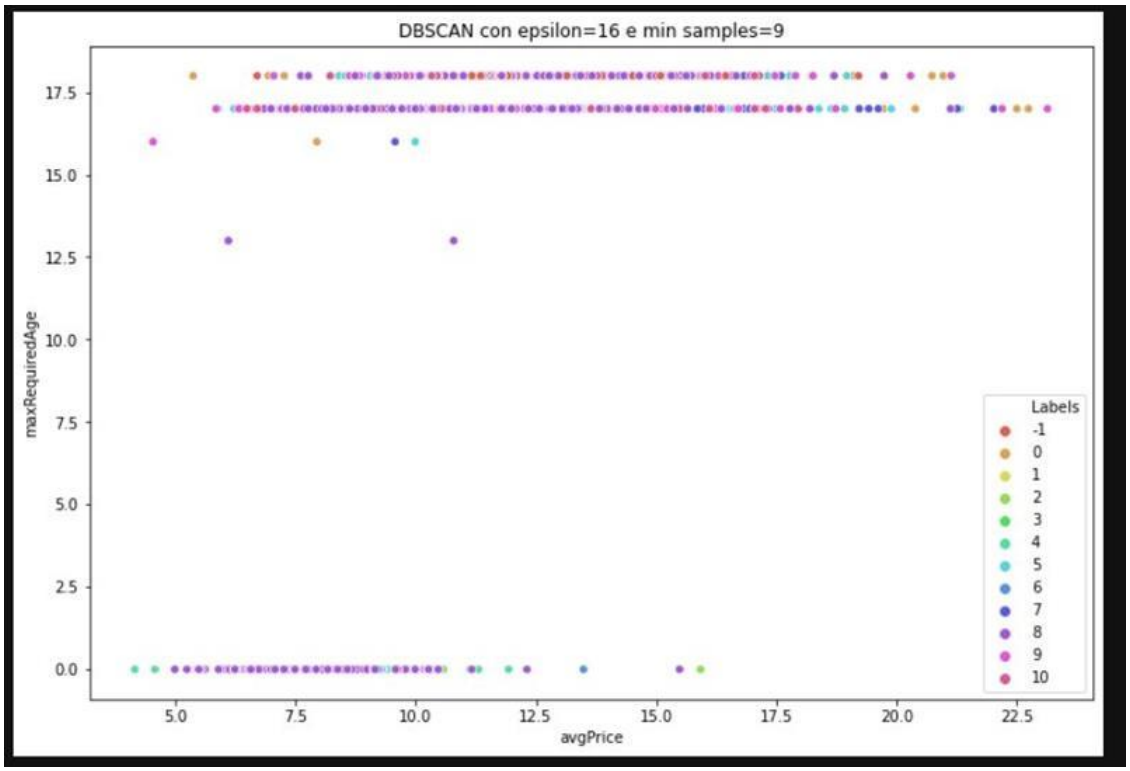
```
[*]: from sklearn.cluster import DBSCAN
     import numpy as np
     import pandas as pd
     import matplotlib.pyplot as plt
     import seaborn as sns


     df = pd.read_csv('datasetOfficial.csv')
     X = df.drop(['Id', 'Username'], axis=1)

     db = DBSCAN(eps=15, min_samples=9).fit(X)


     X['Labels'] = db.labels_
     plt.figure(figsize=(12,8))
     sns.scatterplot(X['avgPrice'], X['maxRequiredAge'], hue=X['Labels'], palette=sns.color_palette('hls', np.unique(db.labels_).shape[0]))
     plt.title('DBSCAN con epsilon=15 e min samples=9')
     plt.show()
```

Mostriamo di seguito il grafico dei campioni, colorati per appartenenza al cluster, in uno spazio di rappresentazione bidimensionale, ove l'asse x è il prezzo medio, e y è l'età richiesta dai prodotti. Anche da questa rappresentazione, è possibile denotare la densità del dataset.

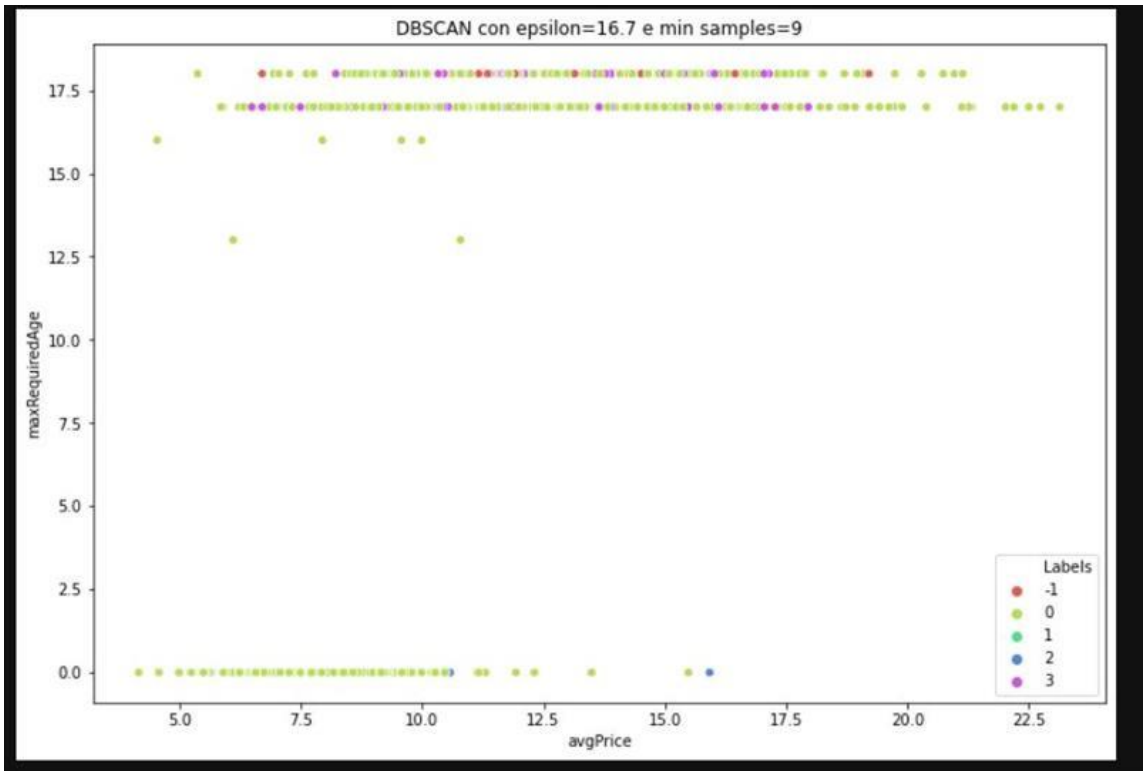
Con Ɛ=15 e minPts=9:



Con Ɛ=16 e minPts=9:



Con Ɛ=16.7 e minPts=9:



Veniamo ora al **silhouette score** dato in output dall'esecuzione:

Codice per mostrare il silhouette score:

```
fig, ax = plt.subplots(1,2,figsize=(15,5))
y_predict = db.fit_predict(X)
silhouette_vals = silhouette_samples(X, y_predict)
y_ticks = []
y_lower = y_upper = 0
for i, cluster in enumerate(np.unique(y_predict)):
    cluster_silhouette_vals = silhouette_vals[y_predict == cluster]
    cluster_silhouette_vals.sort()
    y_upper += len(cluster_silhouette_vals)

    ax[0].barh(range(y_lower, y_upper), cluster_silhouette_vals, height = 1);
    ax[0].text(-0.03, (y_lower+y_upper)/2, str(i+1))
    y_lower += len(cluster_silhouette_vals)

    avg_score = np.mean(silhouette_vals)
    ax[0].axvline(avg_score, linestyle= '--', linewidth = 2, color='green')
    ax[0].set_yticks([])
    ax[0].set_xlim([-0.1, 1])
    ax[0].set_xlabel('Silhouette coefficient values')
    ax[0].set_ylabel('Cluster labels')

    ax[1].scatter(X['maxRequiredAge'], X['avgPrice'], c = y_predict);
    ax[1].set_xlabel('Eruption time in mins')
    ax[1].set_ylabel('Waiting time to next eruption')
    ax[1].set_title('Visualization of clustered data', y=1.02)

    plt.tight_layout()
```

Grafico del silhouette score con Ɛ=15 e minPts=9:

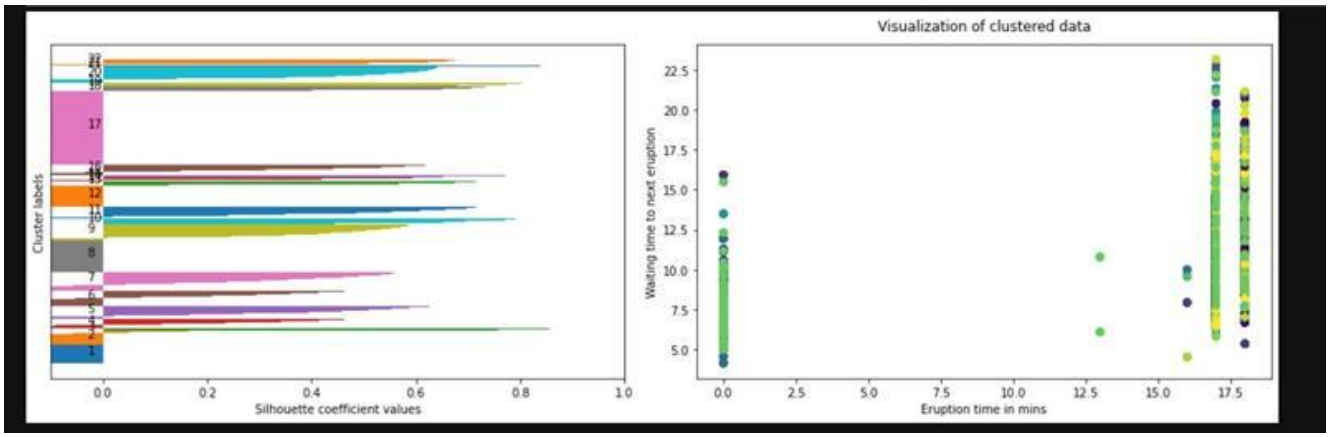


Grafico del silhouette score con Ɛ=16 e minPts=9:

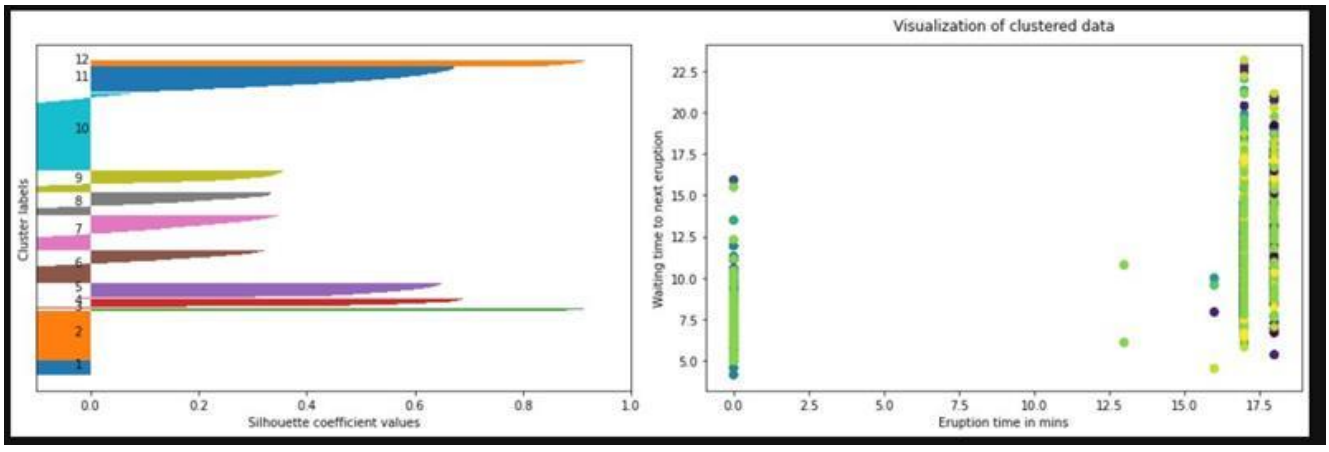
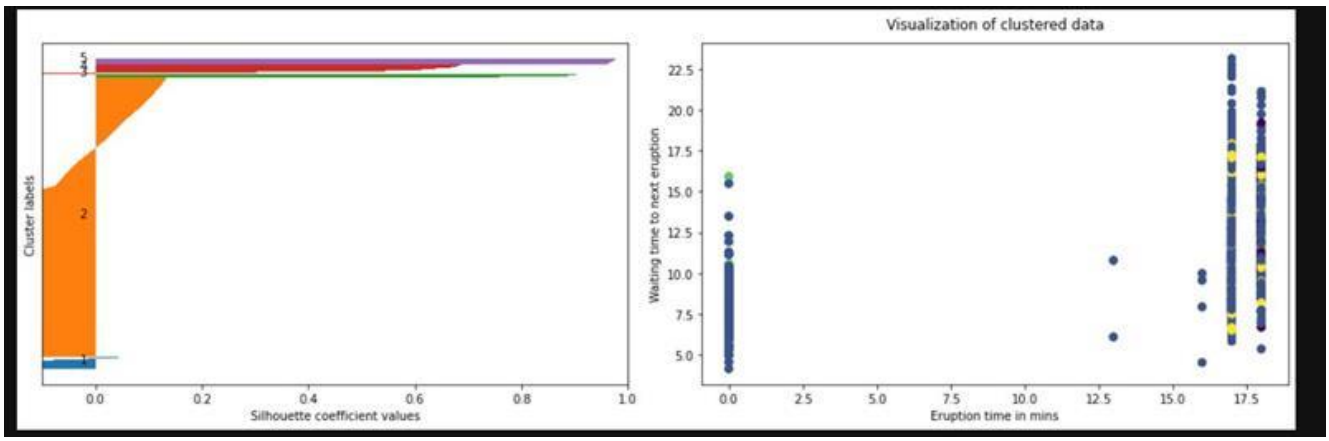


Grafico del silhouette score con Ɛ=16.7 e minPts=9:



Possiamo notare come il silhouette score aumenti nella maggior parte dei cluster aumentando il raggio, e come il numero ottimale di cluster sembra assestarsi tra 4 e 5.

Nonostante le sue ottime prestazioni, il DBScan soffre di una limitazione fondamentale ai fini del nostro sistema: non esiste nessuna rappresentazione significativa di ogni cluster, al di fuori degli elementi dello stesso. Anche infatti il tentativo di calcolare un **punto medio** (**centroide**) per ogni cluster generato da DBScan è risultato fallimentare, dato che l'algoritmo può generare cluster di qualsivoglia forma e dimensione, a patto che lo spazio sia sufficientemente denso, il più delle volte, tale punto medio avrebbe potuto trovarsi ben al di fuori del cluster stesso, facendogli perdere ogni significatività. Tale rappresentazione significativa del cluster risultava essere indispensabile poiché, per costruire la lista dei prodotti consigliati, era necessario conoscere le caratteristiche più comuni dei prodotti più comuni all'interno del cluster.

Un'altra motivazione che ha portato allo scartare il DBScan come soluzione è la difficoltà di definire un metodo di "predizione" del cluster di appartenenza per nuovi campioni, che non prevedesse la creazione di un **classificatore** basato sulle etichette generate. Inoltre, anche procedendo in questa direzione, sarebbe comunque stato possibile che un nuovo campione potesse essere classificato come "outlier", pertanto non sarebbe stato possibile ricavare alcuna informazione utile.

4.1.2 K-Means

A questo punto, il nostro occhio si è posato su un altro algoritmo, il **K-Means**, che pur non avendo le stesse prestazioni in ambiti densi (e quindi con il nostro dataset), avrebbe permesso di effettuare molto più semplicemente le operazioni di "previsione" di appartenenza al cluster e l'operazione di calcolo delle caratteristiche più comuni dei prodotti all'interno dello stesso.

Come noto a molti, il problema fondamentale del K-Means, consiste nel dover individuare a priori il numero di cluster. A tal scopo, forti anche delle considerazioni sul numero di cluster pervenute dai tentativi effettuati con il DBScan, abbiamo ritenuto che il numero ottimale di cluster si dovesse trovare tra 2 e 9.

Lo script che applica il K-Means è estremamente semplice:

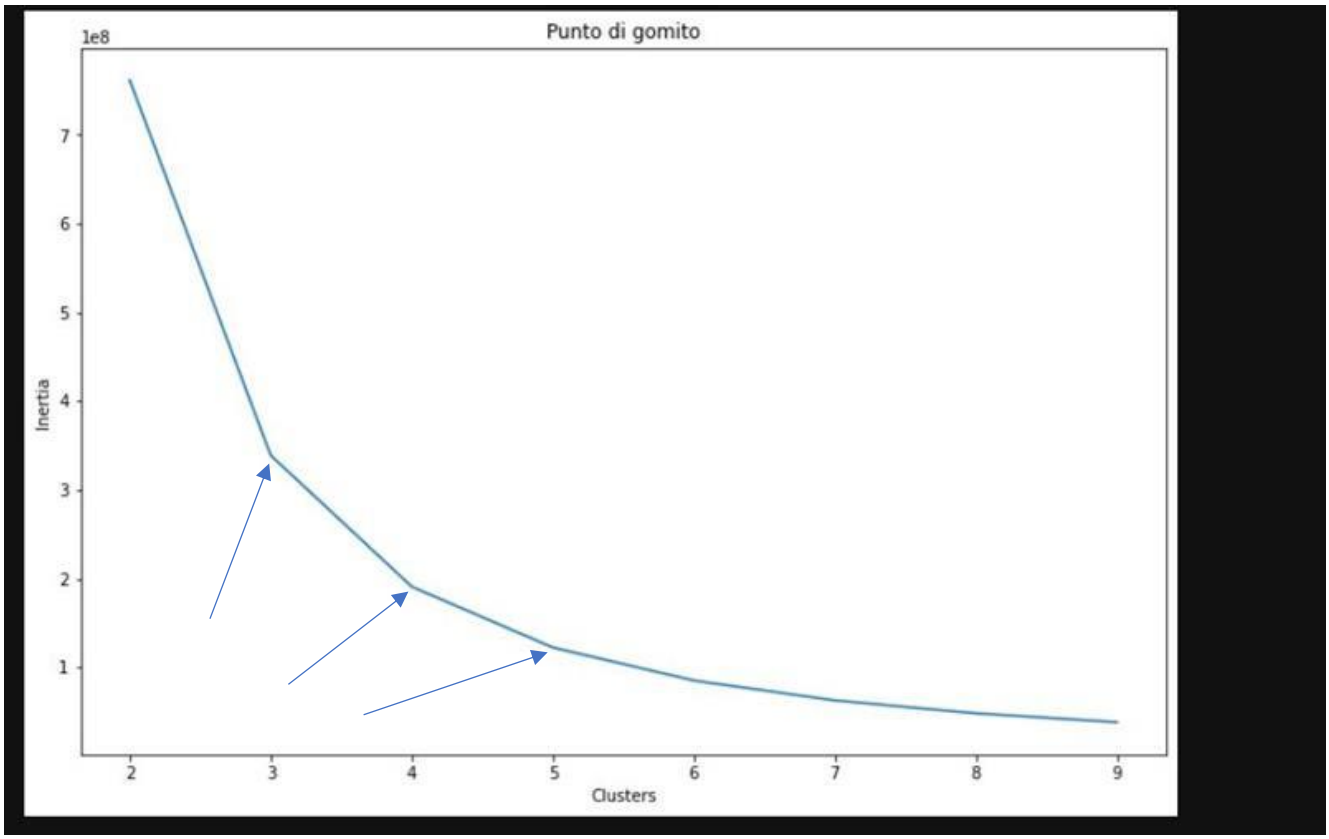
1. Legge il file CSV del dataset ed elimina le colonne irrilevanti (Id utente e Username);
2. Effettua la chiamata a KMeans, per ogni k da 2 a 9, calcolando la rispettiva somma dei quadrati degli errori finale, e stampando il silhouette score medio per quel numero di cluster;
3. Stampa il grafico della funzione della somma dei quadrati degli errori;

```python
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_samples,silhouette_score
import json
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

df = pd.read_csv('datasetOfficial.csv')
X = df.drop(['Id', 'Username'], axis=1)

clusterErrors= []

for i in range (2, 10):
    km = KMeans(n_clusters=i, max_iter=1000).fit(X)
    clusterErrors.append(km.inertia_)
    y_predict = km.fit_predict(X)
    centroids = km.cluster_centers_
    label = km.predict(X)
    print(f'Silhouette Score(n={i}): {silhouette_score(X, label)}')


fig, ax = plt.subplots(figsize=(12, 8))
sns.lineplot(x=list(range(2,10)), y=clusterErrors, ax=ax)
ax.set_title('Punto di gomito')
ax.set_xlabel('Clusters')
ax.set_ylabel('Inertia')
```

Silhouette score per i vari k da 2 a 9:

```
Silhouette Score(n=2): 0.6260771034210855
Silhouette Score(n=3): 0.5888932621796592
Silhouette Score(n=4): 0.5698454684158012
Silhouette Score(n=5): 0.5579695551307063
Silhouette Score(n=6): 0.5496381137780153
Silhouette Score(n=7): 0.5433117645118282
Silhouette Score(n=8): 0.5382132664064472
Silhouette Score(n=9): 0.5339645122559663
```

Grafico della somma dei quadrati degli errori al variare dal numero di cluster, indicati con una freccia i possibili punti a gomito:



Dal grafico della somma dei quadrati degli errori, emergono tre possibili punti di gomito: 3, 4 e 5.

Dopo una serie di considerazioni, e dopo aver valutato anche il silhouette score, abbiamo optato per k=5, in quanto abbiamo ritenuto che l'eccessiva pendenza della funzione della SSE nei punti k=3 e k=4 fosse sintomo di un possibile **overfitting** sullo scarso numero di dati a disposizione che, tra le altre cose, sono anche molto densi.

4.2 Implementazione dello script

Una volta scelto l'algoritmo di clustering, non rimaneva che creare lo script che effettuasse il training e **serializzasse** il modello, permettendone l'interrogazione in un secondo momento.

4.2.1 Allenamento e serializzazione

Analizziamo ora lo script che effettua l'allenamento del modello di clustering e la sua serializzazione, esso:

1. Legge il dataset dal file CSV ed elimina le features non rilevanti (id e username);
2. Chiama KMeans per calcolare i cluster con k=5;
3. Chiama findMostCommonTagList(), che calcola e restituisce la lista delle 5 caratteristiche (tag) più comuni all'interno del cluster, per ognuno dei 5 cluster (ne mostriamo di seguito l'implementazione);
4. Restituisce la coppia (clusterizzatore, matrice dei tag più comuni per ogni cluster)

```python
def trainClusterer() -> (KMeans, list):
    df = pd.read_csv(DATASET_FILENAME)
    X = df.drop(['Id', 'Username'], axis=1)
    km = KMeans(n_clusters=NUM_CLUSTERS, max_iter=1000).fit(X)
    mostCommonTagsList = findMostCommonTagsList(df, km)
    return km, mostCommonTagsList
```

```python
# costruiamo una lista di mappe (una per cluster), ogni mappa e' caratterizzata da: key->nomeTag, value->frequenzaTag
def findMostCommonTagsList(df: pd.DataFrame, clustererTrained: KMeans) -> list:
    df['cluster'] = clustererTrained.labels_ #prendiamo le label
    listMaps = []

    for i in range (NUM_CLUSTERS):
        tagsHead = df.filter(regex="((^Tag_.*$)|(usesMultiplayer))", axis=1).columns
        filteredDf = df.filter(regex="((^Tag_.*$)|(usesMultiplayer)|(cluster))", axis=1)

        listMaps.append({})
        for tag in tagsHead:
            #tagColumn = filteredDf.filter(like=tag, axis=1)
            buffer = filteredDf[filteredDf[tag]==1]
            listMaps[i][tag] = len(buffer[buffer['cluster']==i])

    #estraiamo i migliori TAG_AMOUNT da ogni cluster
    bestLists = [[] for l in range(NUM_CLUSTERS)] #lista di best vuoti
    for i in range(NUM_CLUSTERS):
        for j in range(TAG_AMOUNT):
            #chiave con il massimo valore, ripetuto per TAG_AMOUNT volte
            maxKey = max(listMaps[i].keys(), key=(lambda k: listMaps[i][k]))
            listMaps[i].pop(maxKey) #rimuoviamo il massimo
            bestLists[i].append(maxKey) #aggiungiamo la sua chiave alla lista dei migliori tag per il cluster specificato
    return bestLists
```

Mostriamo ora le due funzioni che si occupano di caricare salvare (serializzare) il custer e di caricarlo dal file. Molto semplicemente, la priam salva la coppia restituita da trainClusterer() su un file, e l'altra la legge e la restituisce. Entrambe le funzioni dispongono della libreria **pickle**.

```
def saveClusterer(clustererTrained: KMeans, mostCommonTagsList: list):
    couple = (clustererTrained, mostCommonTagsList)
    pickle.dump(couple, open(SERIALIZED_CLUSTER_FILENAME, 'wb'))


def loadClusterer() -> (KMeans, list):
    clusterAndTagList = pickle.load(open(SERIALIZED_CLUSTER_FILENAME, 'rb'))
    return clusterAndTagList
```

4.2.2 Predizione del cluster di appartenenza dei nuovi campioni

Ora non resta che mostrare come è stata gestita la predizione del cluster di appartenenza di un nuovo campione da dare in input al modello. Lo script relativo è, ancora una volta, piuttosto semplice, in quanto sfrutta il metodo predict() che la classe KMeans mette a disposizione. Essa permette, a partire da un modello già allenato, di prevedere il cluster di appartenenza di uno o più campioni passati:

1. Legge dal file buffer.csv il valore del nuovo campione, e chiama la format() per formattarlo;
2. Carica il clusterizzatore serializzato dal file con loadClusterer(), insieme alla matrice dei tag più comuni in ogni cluster;
3. Chiama la predict() sul campione letto per stabilirne il cluster di appartenenza;
4. In base al cluster di appartenenza, viene restituita la lista di tag più comuni in tale cluster, letta dalla matrice;

```
def predict() -> list:
    print(os.listdir())
    #leggiamo il campione dal csv e lo formattiamo con la one-hot encode
    predictValue = format()
    predictValue.drop(['Username', "Id"], axis=1, inplace=True)

    #accediamo alla coppia (clusterizzatore, listaDeiTagMiglioreDeiCluster) serializzata
    clusterer, mostCommonTagList = clustering.loadClusterer()

    #prevediamo il valore del cluster del campione
    clusterNum = clusterer.predict(predictValue)[0] #cluster più vicino al campione

    tagList = mostCommonTagList[clusterNum] #accediamo alla lista di tag del cluster
    formatTagList(tagList) #formattiamo correttamente i tag

    return tagList
```

5 Integrazione con il sistema

Per effettuare l'integrazione del modulo di Intelligenza Artificiale con il sistema GameHub inizialmente si era pensato all'utilizzo del framework **Jython**, ma durante la fase di sviluppo ci si è resi conto di come tale framework non supportasse l'import di librerie quali **Scikit-learn**, che si poggiano in gran parte su librerie scritte in linguaggi a basso livello come C o Fortran. Per questo motivo è stato preso in considerazione il framework **JEP**, che permette l'esecuzione di script Python all'interno di un ambiente interamente sviluppato in Java, supportando anche le librerie esterne adottate.

In particolare, JEP prevede la possibilità, in Java, di creare oggetti di tipo "Interpeter", rappresentanti gli interpreti CPython, dai quali è poi possibile eseguire codice Python.

L'integrazione con il progetto è stata realizzata attraverso la creazione di quattro classi:

1. DatasetSampleDAO
2. PersonalizationBridge
3. RecommendedProductList
4. RecommendedProductListBuilder

5.1 DatasetSampleDAO

Lo scopo di tale classe è quello di fornire un metodo che, dato lo username di un utente, sia in grado di selezionare dalla vista "DatasetSample", descritta al paragrafo riportato, il campione corrispondente al nome utente passato come parametro. Dopo aver effettuato una query sulla vista, il risultato viene memorizzato all'interno di un file buffer.csv.

Di seguito viene riportato il codice relativo alla classe DatasetSampleDAO.

```java
/**
 * This class's purpose is to write, on a buffer csv file, new user samples for the machine
 * learning model used to segment a classify the users.
 */
public class DatasetSampleDAO {
    public static final @NotNull String BUFFER =
            HomeServlet.EXECUTION_PATH + "/WEB-INF/personalization/buffer.csv";

    /**
     * This method buffers the user sample corresponding to the given username.
     * It is synchronized because no more than one user at time may access to the buffer file.
     *
     * @param username the username corresponding to the user sample to buffer.
     *                 It has to be non-null.
     */
    public synchronized void doBufferByUsername(@NotNull String username) {
        try {
            Connection cn = ConPool.getConnection();
            CSVWriter csvWriter = new CSVWriter(new FileWriter(BUFFER)); //creiamo il CSVWriter
            PreparedStatement ps = cn.prepareStatement(
                    sql: "SELECT * from game_hub_db.datasetsample where Username=?;"
            );
            ps.setString( parameterIndex: 1, username);
            ResultSet rs = ps.executeQuery();
            csvWriter.writeAll(rs, includeColumnNames: true); //scriviamo sul csv i nuovi dati
            csvWriter.flush();
            csvWriter.close();
        } catch (SQLException | IOException e) {
            throw new RuntimeException(e);
        }
    }
}
```

5.2 PersonalizationBridge

La classe "PersonalizationBridge" è un **Singleton**, che rappresenta inoltre un **Adapter** tra il modello di machine learning e l'applicazione Java. Il pattern Singleton, oltre all'utilizzo di **metodi synchronized** per l'intera classe, è stato adottato per evitare eventuali **race condition** sui file CSV coinvolti nell'elaborazione.

5.2.1 PersonalizationBridge – getTagList()

Il seguente metodo prende in input un oggetto di tipo User, che modella un utente, e grazie al metodo doBufferByUsername della classe DatasetSampleDAO ne effettua il salvataggio all'interno del file .csv di buffer.

Successivamente si ricorre all'utilizzo di JEP per la creazione di un interprete Python il quale effettua l'invocazione del metodo predict() descritto al paragrafo indicato.

Dopo aver invocato il metodo predict() si ottengono i nomi dei tag di possibile interesse da parte dell'utente. Infine, per ognuna delle stringhe restituite dalla chiamata, l'oggetto di tipo Tag, interrogando il database del sistema GameHub, e si restituisce il risultato finale.

```java
/**
 * This method queries a python script (and indirectly the machine learning model) to
 * get the tag list for the given user.
 * Method is synchronized because no more than one user at time may access to the buffer file.
 *
 * @param u the subject user we want to get tag list of
 * @return the tag list of the cluster of the user, according to the machine learning model
 * @throws TagPredictionException if the tag prediction fails
 */
@NotNull
public synchronized List<Tag> getTagList(@NotNull User u)  {
    List<Tag> tags = new ArrayList<>();
    TagDAO td = new TagDAO();
    dsd.doBufferByUsername(u.getUsername()); //scriviamo sul file il campione
    try {
        //creiamo l'istanza dell'interprete
        SharedInterpreter si = new SharedInterpreter();
        //gestione della path
        String path = HomeServlet.EXECUTION_PATH.replace( target: "\\",  replacement: "/");
        si.exec( s: "import os");
        si.exec( s: "os.chdir('" + path + "')");

        //prendiamoci i tag dallo script python, che chiamerà il modello di machine learning
        //tramite il metodo predict()
        si.exec( s: "from samplePrediction import predict");
        ArrayList<String> tagNames = si.getValue( s: "predict()", ArrayList.class);
        si.close();
        tagNames.forEach(t -> tags.add(td.doRetrieveByName(t)));
        return tags;
    } catch (Exception e) {
        throw new TagPredictionException(e);
    }
}
```

5.3 Recommended Product List

La classe "RecommendedProductList" modella lista di prodotti consigliati, e segue il design pattern **Proxy**. Questa classe mantiene una lista di prodotti consigliati, inizialmente vuota, il voto che l'utente ha dato riguardo essa (inizialmente null) e l'utente a cui si riferisce, oltre che un'instanza di

RecommendedProductListBuilder, il cui ruolo verrà chiarito in seguito. Ogni istanza di questa classe è associata ad uno ed un solo User (utente), e pertanto, quando l'utente effettua il login, un'istanza di questa classe viene posta nella **sessione**. Possiede tre metodi: getList(), getVote() e setVote(). I metodi getVote() e setVote() hanno un ruolo nell'evoluzione del modello e nel sistema di feedback e verranno, pertanto, descritti nella sezione apposita. Si può inolre notare come la classe abbia una variabile statica contenente l'istanza del singleton PersonalizationBridge, poiché questo ha un ruolo fondamentale nella creazione della lista dei prodotti consigliati effettiva.

```java
/**
 * This class models the recommended {@link Product} list, created based on the user interest,
 * following the indications of the machine learning model.
 * Note that this class uses the Proxy design pattern, so that when an instance of this class is
 * constructed, the actual {@link Product} list is empty, and then, when it's requested, the list
 * is filled based on the machine learning model recommendations.
 */
public class RecommendedProductList {
    //variabile static e final in quanto non più di un utente alla volta deve accedere al bridge
    @NotNull
    private static final PersonalizationBridge BRIDGE = PersonalizationBridge.getInstance();

    /**
     * Construct a new, empty {@link RecommendedProductList} for the given {@link User}.
     *
     * @param user the user we want to construct the {@link RecommendedProductList} for
     */
    public RecommendedProductList(@NotNull User user) {
        this.products = new ArrayList<>();
        this.builder = new RecommendedProductListBuilder();
        this.user = user;
        this.vote = null;
    }
```

5.3.1 RecommendedProductList - getList()

Il metodo getList() si occupa di restituire la lista dei prodotti consigliati, se essa già esiste, o di crearla nel caso in cui non esista, prima di restituirla. Difatti, quando viene invocato il metodo:

1. Se la lista è già stata inizializzata la restituisce, altrimenti;
2. Richiama getTagList() sul PersonalizationBridge, facendosi restituire la lista di tag;
3. Richiama buildList(), sull'istanza di RecommendedProductListBuilder che, passandogli la lista di tag, riempie la lista di prodotti consigliati;

5.3.2 RecommendedProductListBuilder – buildList()

Questa classe, con il solo metodo buildList(), ha l'unico scopo di riempire la lista di prodotti consigliati nel momento in cui viene richiesta, sulla base dei tag che vengono passati allo stesso metodo.

Diremo che un prodotto ha un "match" con un tag, se tale prodotto è associato a tale tag nel database.

Il concetto che sta alla base dell'algoritmo sviluppato si basa sul numero di "match" dei prodotti fisici e digitali con i tag passati come parametro.

A tale scopo si utilizza una coda a priorità laddove l'elemento con priorità massima corrisponde al prodotto che ha più match con i tags, contrariamente l'elemento con priorità minima corrisponde al prodotto che ha meno match con i tags.

```
public List<Product> buildList(@NotNull HashMap<String, Tag> tags) {
    if (tags.isEmpty()) {
        throw new IllegalArgumentException("Error: tag list must be not null!");
    }
    List<Product> products = new ArrayList<>();
    List<Tag> tagList = new ArrayList<>(tags.values());
    HashMap<String, Integer> productsPriorities = new HashMap<>();

    //creiamo una coda a priorità ordinata in base allo score dei prodotti;
    //Lo score di un prodotto p è pari ai tag che ha in comune con la lista tags passata;
    PriorityQueue<Product> pq = new PriorityQueue<>(
            Comparator.comparingInt(
                    p -> -productsPriorities.get(p.getClass().getSimpleName() + p.getId())
            )
    );

    //creiamo la lista di prodotti effettuando il merge delle due liste(fisici e digitali)
    products.addAll(ppd.doRetrieveAllByTags(tagList,  offset: 0, MAX_LIMIT));
    products.addAll(dpd.doRetrieveAllByTags(tagList,  offset: 0, MAX_LIMIT));
```

Per ogni prodotto si calcola lo score e lo si inserisce all'interno della coda a priorità.

Successivamente si estraggono i primi NUM_PRODS (variabile che indica il numero di prodotti da consigliare) con priorità massima e vengono ritornati dal metodo.

```
    //calcoliamo lo score di ogni prodotto, sulla base del numero di tag che
    //matchano con la lista passata, e inseriamolo in una coda a priorità (tempo O(nlogn))
    products.forEach(p -> {
        int score = 0;
        //se il prodotto ha il tag, aumentiamo il suo score
        for (Tag t : tagList) {
            if (p.hasTag(t.getName()) != null) {
                score++;
            }
        }
        productsPriorities.put(p.getClass().getSimpleName() + p.getId(), score);
        pq.add(p); //aggiungiamo alla coda a priorità
    });

    //se ci sono solo NUM_PRODS o meno elementi ritorniamo tutti i prodotti nella coda
    if (pq.size() <= NUM_PRODS) {
        return new ArrayList<>(pq);
    }
    List<Product> bestProducts = new ArrayList<>();
    for (int i = 0; i < NUM_PRODS; i++) {
        bestProducts.add(pq.poll());
    }
    return bestProducts;
```

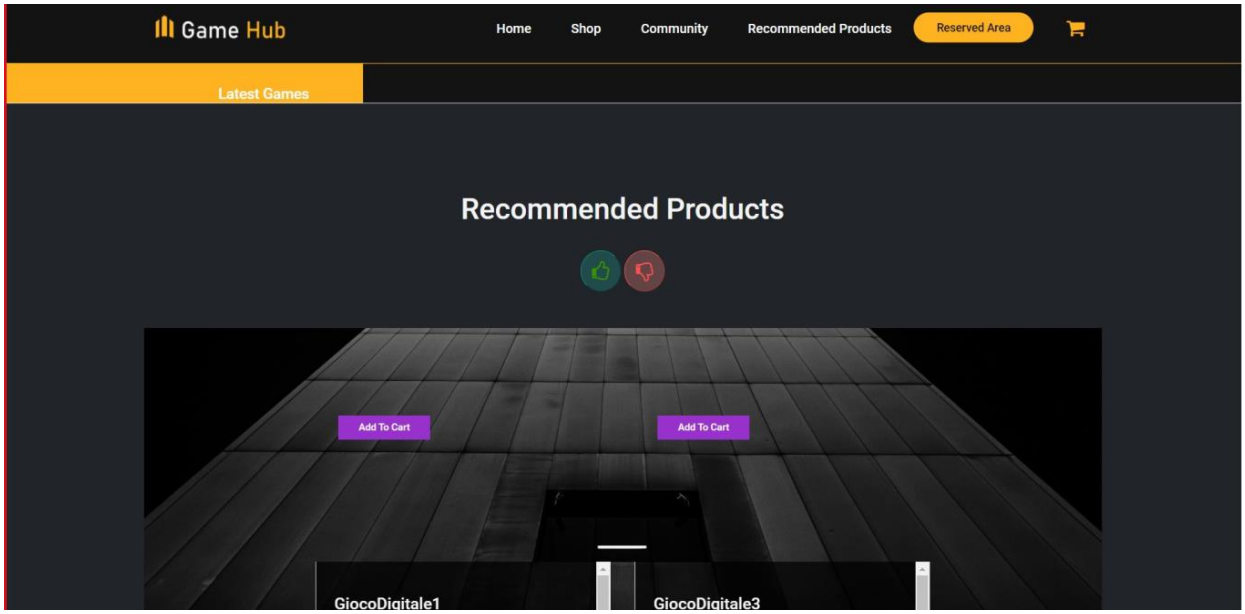
5.4 Pagina dei prodotti consigliati

Tutte le informazioni elaborate e riposte nella lista dei prodotti consigliati salvata in sessione vengono poi mostrate all'utente mediante una pagina apposita. Tale pagina viene mostrata attraverso la **servlet** ShowRecommendedProductsServlet, che chiama la getList() sulla RecommendeMostriamo salvata in sessione, e la passa alla **JSP** recommendedProducts.jsp.

Mostriamo di seguito il codice della servlet:

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    HttpSession session = request.getSession();
    User loggedUser = (User) session.getAttribute( s: "loggedUser");
    List<Product> actualProductList;
    RecommendedProductList rpl;
    rpl = (RecommendedProductList) session.getAttribute( s: "recommendedProducts");

    if (loggedUser == null) {
        throw new RequestParametersException(
                "Error: you cannot access to the recommended product list without being logged!"
        );
    }

    synchronized (session) {
        actualProductList = rpl.getList(); //calcoliamo la lista dei prodotti consigliati
    }


    request.setAttribute( s: "products", actualProductList);
    RequestDispatcher rd = request.getRequestDispatcher(
            s: "/WEB-INF/view/recommendedProducts.jsp"
    );
    rd.forward(request, response);
}
```

Ed uno screenshot della pagina:

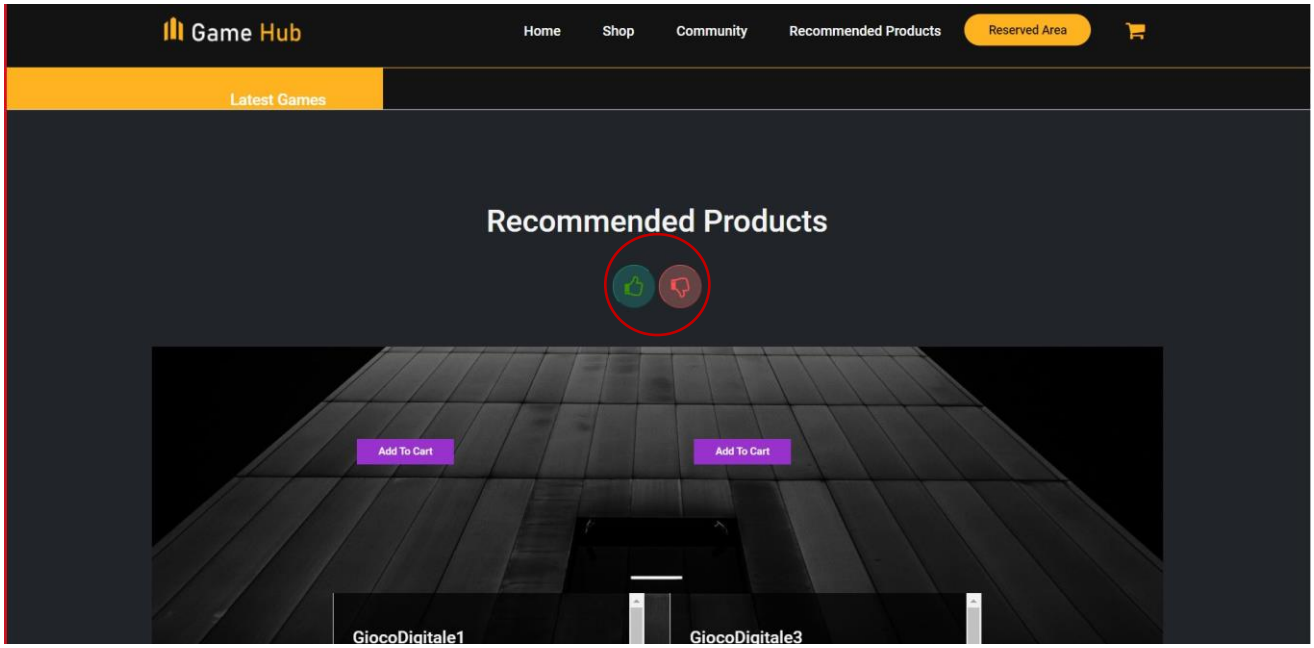


6 Evoluzione del modello

6.1 Introduzione al sistema di feedback

Come abbiamo già detto in precedenza, lo scopo dell'agente era non solo quello di consigliare ad ogni utente un insieme di prodotti che si avvicinasse quanto più possibile alle sue preferenze, ma anche quello di migliorare i consigli nel tempo sulla base della valutazione degli stessi utenti.

A tale scopo, abbiamo elaborato un sistema di feedback, che permette ad ogni utente, ogni volta che accede alla pagina dei prodotti consigliati, di esprimere una propria valutazione su di essi, attraverso un semplice bottone (pollice in su per valutazione positiva, pollice in giù per valutazione negativa):



6.2 Registrazione del voto

6.2.1 Servlet per la registrazione del voto

Il bottone di valutazione, quando cliccato richiama una servlet, VotePersonalizationServlet, la quale a sua volta richiama il metodo setVote(), che descriveremo a breve, sulla RecommendedProductList salvata in sessione:

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    HttpSession session = request.getSession();
    User loggedUser = (User) session.getAttribute( s: "loggedUser");
    boolean vote = Boolean.parseBoolean(request.getParameter( s: "vote"));
    RecommendedProductList rpl;
    rpl = (RecommendedProductList) session.getAttribute( s: "recommendedProducts");

    if (loggedUser == null) {
        throw new RequestParametersException(
                "Error: you cannot access to the recommended product list without being logged!"
        );
    }

    JsonObject responseObject = new JsonObject();
    synchronized (session) {
        try {
            rpl.setVote(vote);
            responseObject.addProperty( property: "type",  value: "success");
            responseObject.addProperty( property: "msg",  value: "Voting completed successfully!");
        } catch (Exception e) {
            responseObject.addProperty( property: "type",  value: "error");
            responseObject.addProperty( property: "msg",  value: "Voting not completed: " + e);
        } finally {
            response.getWriter().println(responseObject);
            response.flushBuffer();
        }
    }
}
```

6.2.2 RecommendedProductList – setVote()

Veniamo ora al metodo setVote() della classe RecommendedProductList, che abbiamo in precedenza tralasciato. Questo metodo richiama semplicemente il metodo registerVote() di PersonalizationBridge, il quale a sua volta richiama gli script Python che registrano il voto su un file CSV, votes.csv.

```
/**
 * This method sets the vote value of the {@link RecommendedProductList}.
 * Note that each time this method is called, the user's vote is saved on a file, for future
 * machine learning model retraining, based on it's performance.
 * Note that this method should be called iff the actual {@link Product} {@link List} has
 * already been initialized, for semantics reasons; for this reason, a getList() invocation
 * prior to this method invocation is mandatory.
 *
 * @param vote the user vote value for the list, must be either true or false
 * @throws ListNotInitializedException if the method is called while the {@link Product}
 * {@link List} it's not initialized yet (meaning the getList() method was not called yet).
 */
public void setVote(@NotNull Boolean vote) {
    if (this.products.isEmpty()) {
        throw new ListNotInitializedException("Error: setVote() method may not be called if"
                + " the product list is not initialized yet!");
    }
    this.vote = vote;
    //chiamiamo il bridge per registrare il voto
    BRIDGE.registerVote(user, vote);
}
```

6.2.3 PersonalizationBridge – registerVote()

Il metodo registerVote() di PersonalizationBridge non fa altro che richiamare l'omonima funzione Python, per registrare il voto dell'utente passato sul file votes.csv.

```java
/**
 * This method call a python script to register the vote of the user, for the recommended
 * product list generated by the system.
 * Method is synchronized because no more than one user at time may access to the buffer file.
 *
 * @param u the subject user we want to register the vote for the recommended product list
 * @throws VoteRegistrationException if the vote registration fails
 */
public synchronized void registerVote(@NotNull User u, boolean vote) {
    dsd.doBufferByUsername(u.getUsername()); //scriviamo sul file il campione

    try {
        SharedInterpreter si = new SharedInterpreter();
        //gestione della path
        String path = HomeServlet.EXECUTION_PATH.replace( target: "\\", replacement: "/");
        si.exec( s: "import os");
        si.exec( s: "os.chdir('" + path + "')");
        //passiamo il voto da registrare allo script python
        Boolean voteObj = vote;
        si.exec( s: "from samplePrediction import registerVote");
        si.set("vote", voteObj);
        Boolean success = si.getValue( s: "registerVote(vote)", Boolean.class);
        si.close();
        if (!success) {
            throw new VoteRegistrationException();
        }
    } catch (JepException e) {
        throw new VoteRegistrationException(e);
    }
}
```

6.2.4 Funzione registerVote()

La funzione Python registerVote() è estremamente semplice:

1. Legge il file buffer.csv ove è contenuto il campione relativo all'utente che ha votato;
2. Legge il file votes.csv ove sono contenuti i voti degli utenti;
3. Se il voto dell'utente è già presente nel insieme di voti, viene aggiornato, altrimenti viene aggiunto;

```
def registerVote(vote: bool) -> bool:
    try:
        # Leggiamo i dati del votante dal buffer, ed uniamoli ad i dati presenti nel dataset di voti
        dataFrame = pd.read_csv(clustering.BUFFER_FILENAME)
        dataFrame['Vote'] = vote
        dataFrameVotes = pd.read_csv(clustering.VOTES_FILENAME)
        username = dataFrame.iloc[0].Username
        flag = False

        # Se c'è già il voto di un utente con lo stesso username, aggiorniamo il voto
        for i in range(len(dataFrameVotes['Username'])):
            if(dataFrameVotes['Username'][i] == username):
                dataFrameVotes['Vote'][i] = vote
                flag = True
        print(dataFrameVotes)

        # Salviamo i risultati in un csv
        if(not flag):
            dataFrame.to_csv(clustering.VOTES_FILENAME, index=False, mode='a', header=False)
        else:
            dataFrameVotes.to_csv(clustering.VOTES_FILENAME, index=False, header=True)

        return True
    except:
        return False
```

6.3 Retraining del modello

Una volta elaborato il meccanismo di feedback, si è reso necessario elaborare un sistema che permettesse di migliorare effettivamente il modello sulla base di esso. È stato elaborato, pertanto, un insieme di funzioni Python, e metodi Java, che adempie al compito. Il meccanismo, ad alto livello, opera nel seguente modo:

1. Si verifica che il file votes.csv contenga abbastanza voti, e che essi siano negativi per una percentuale > del 50%;
2. Se la condizione precedente si verifica, i campioni degli utenti corrispondenti ai voti vengono aggiunti al dataset di addestramento, e si effettua il retraining del modello di clustering;

Non avendo implementato, almeno nella prima release, un meccanismo lato client che permetta di effettuare questi controlli, abbiamo presupposto che, periodicamente, il controllo venga effettuato da un operatore.

Con questo sistema messo appunto tuttavia, il modello potrà continuare ad evolvere e a migliorarsi nel tempo sulla base dei feedback degli utenti, rispondendo anche al secondo obbiettivo dell'agente che si intendeva realizzare.

6.3.1 PersonalizationBridge - checkRetraining()

Il metodo checkRetraining() di PersonalizationBridge è l'interfaccia Java verso l'omonima funzione Python, che verifica se il modello dev'essere riallenato, ed effettua l'operazione se necessario.

```java
/**
 * This method calls a python script to establish whatever the clusterer has to be retraining
 * on new data, based on the votes of the users.
 *
 * @throws CheckRetrainingException if the tag prediction fails
 */
public synchronized void checkRetraining() {
    //creiamo l'istanza dell'interprete
    try {
        SharedInterpreter si = new SharedInterpreter();
        //gestione della path
        String path = HomeServlet.EXECUTION_PATH.replace( target: "\\",  replacement: "/");
        si.exec( s: "import os");
        si.exec( s: "os.chdir('" + path + "')");
        //chiamiamo lo script per verificare se il clusterizzatore necessita retraining
        si.exec( s: "from samplePrediction import checkRetraining");
        si.exec( s: "checkRetraining()");
        si.close();
    } catch (JepException e) {
        throw new CheckRetrainingException(e);
    }
}
```

6.3.2 Funzione checkRetraining()

La funzione checkRetraining() ha lo scopo di verificare la necessità o meno di riallenare il clusterizzatore, e riallenarlo se la risposta è positiva. Essa esegue le seguenti operazioni:

1. Verifica che il modello debba essere riallenato, attraverso a funzione hasToRetrain(), che verifica se il numero di voti è sufficiente e che quelli negativi siano più del 50%, e se ciò non si verifica non effettua ulteriori operazioni;
2. Legge il file votes.csv, chiamando la format(), passando il parametro votes a True. Questo poiché tale parametro indica alla funzione format() che dovrà, oltre a codificare i dati passati con la One-Hote-Encode, anche aggiungere i dati passati al dataset di training temporaneo, salvato in datasetTemp.csv;
3. Sostituisce il dataset precedente con quello aggiornato, letto da datasetTemp.csv, che contiene nuove eventuali features dei dati, oltre che i nuovi campioni di training provenienti dai voti;
4. Effettua il riallenamento del modello di clustering, e lo serializza chiamando la saveClusterer();

Mostriamo di seguito il codice delle funzioni checkRetraining():

```
def checkRetraining():
    print(clustering.hasToRetrain())
    if(clustering.hasToRetrain()):
        #leggiamo il dataset con le nuove features
        df = pd.read_csv(clustering.DATASET_TEMP_FILENAME)

        # Rimozione di eventuali colonne "Unnamed"
        df = df.filter(regex="^(?!Unnamed)", axis=1)
        df.drop(columns=['Vote'], inplace=True, axis=1)

        #sovrascriviamo il dataset vecchio con quello con le nuove features
        df.to_csv(clustering.DATASET_FILENAME)

        #leggiamo il dataset dei voti
        format(votes=True)

        #leggiamo il dataset con le nuove features
        df = pd.read_csv(clustering.DATASET_TEMP_FILENAME)
        df = df.filter(regex="^(?!Unnamed)", axis=1)
        df.drop(columns=['Vote'], inplace=True, axis=1)

        # Rimozione di eventuali colonne "Unnamed"
        df = df.filter(regex="^(?!Unnamed)", axis=1)

        print(df)

        #sovrascriviamo il dataset vecchio con quello con le nuove features
        df.to_csv(clustering.DATASET_FILENAME)

        #alleniamo il clusterizzatore e serializziamolo salvandolo
        clustererTrained, mostCommonTagsList = clustering.trainClusterer(clustering.NUM_CLUSTERS)
        clustering.saveClusterer(clustererTrained, mostCommonTagsList)
```

E di hasToRetrain():

```
def hasToRetrain() -> bool:
    df = pd.read_csv(VOTES_FILENAME)
    votes = json.loads(df['Vote'].value_counts(normalize=True).to_frame().to_json())
    if ('False' not in votes['Vote']):
        return False
    negativeVotesPerc = votes['Vote']['False']
    return (negativeVotesPerc > 0.5 and df.size >= MIN_VOTES_RETRAIN)
```

7 Glossario

- **Machine Learning:** branca dell'intelligenza artificiale che si occupa di prevedere il comportamento di una funzione a partire da una distribuzione di dati, "apprendendo" da essi e migliorando le proprie performance con un processo di addestramento;
- **Clustering:** tecnica di apprendimento basata sulla suddivisione dell'insieme di dati (**dataset**) in input in un insieme di gruppi con caratteristiche similari, sulla base di una funzione di distanza. Viene utilizzata, generalmente, quando non si hanno a disposizione a priori etichette da associare ai dati, per poterli "classificare";
- **User Segmentation:** pratica che consiste nel dividere gli utenti di un servizio in gruppi con caratteristiche simili. L'idea alla base è che gli utenti in questi gruppi avranno probabilmente comportamenti simili, e risponderanno altrettanto similmente a pratiche di marketing;
- **Steam:** piattaforma sviluppata da Valve Corporation che si occupa di distribuzione digitale, di gestione dei diritti digitali, di modalità di gioco multi giocatore e di comunicazione. Viene usata per gestire e distribuire una vasta gamma di giochi (alcuni esclusivi) e il loro relativo supporto. Tutte queste operazioni sono effettuate via Internet;
- **Dump**: operazione su un database che consiste nella creazione di un riepilogo della struttura delle tabelle del database medesimo e/o i relativi dati, ed è normalmente nella forma di una lista di dichiarazioni SQL. Tale operazione è utilizzata per lo più per fare il backup del database;
- **Pandas:** libreria Python open-source per la gestione di insiemi di dati di vario tipo ed in vari formati;
- **One-Hot-Encode:** codifica per dati di tipo testuale, che associa ad ogni possibile valore della feature una colonna, che viene posta ad 1 se il campione ha quel valore nella feature, ed a 0 altrimenti;
- **DBScan:** algoritmo di clustering density-based che crea i cluster a partire da due parametri: un raggio ε, ed un minimo numero di punti nella circonferenza di esplorazione minPts. Partendo da un punto casuale ancora non posto in un cluster, se esso ha nel raggio ε almeno minPts campioni, allora vengono uniti in un unico cluster, e si riapplica lo stesso algoritmo a tutti i punti nella circonferenza di esplorazione, altrimenti il punto viene etichettato come outlier e viene scelto un nuovo punto casuale da cui riiniziare lo "scan";
- **Centroide:** punto medio di un cluster, è calcolato come $m_i = \frac{1}{|D_i|}\sum_{x \in D_i} x$, dove D_i è l'i-esimo cluster;
- **Classificatore:** modello di machine learning che, dato un insieme di dati, ciascuno dei quali associato a un insieme di etichette, viene allenato a riconoscere, dato un nuovo campione mai visto, a quale etichetta dev'essere associato;
- **Somma dei quadrati degli errori (SSE):** nell'ambito del clustering, la somma dei quadrati degli errori viene calcolata come $SSE = \sum_{i=1}^{n} \sum_{x \in D_i} ||x - m_i||^2$;
- **K-Means:** algoritmo di clustering di tipo algebrico, che sceglie inizialmente n centroidi casuali, e poi ad ogni iterazione, se i centroidi sono uguali all'iterazione precedente o differiscono di poco si ferma, altrimenti calcola, per ogni campione, il centroide più vicino e lo inserisce nel rispettivo cluster, ripetendo poi l'operazione d'accapo. Il risultato sarà un clustering che minimizza la somma dei quadrati degli errori SSE;
- **Silhouette Score:** misura di prestazione dei cluster dati in output da un algoritmo di clustering, valuta la densità del cluster e la distanza dai cluster, variando tra -1 ed 1;
- **Punto di gomito:** punto in cui la curva della SSE al variare del numero di cluster forma un "gomito" o un flesso. Tale punto dovrebbe rappresentare un numero di cluster vicino a quello ottimale;

- **Overfitting:** condizione in cui un modello di apprendimento di adatta eccessivamente all'insieme dei dati di training, acquisendo ottime prestazione ad esempio nella classificazione dei campioni di training, ma pessime prestazioni con campioni mai visti;
- **Serializzazione:** pratica che consiste nel salvare un oggetto su un file per memorizzarlo in modo persistente e permettere di ricaricarlo in un secondo momento;
- **Pickle:** libreria Python per la serializzazione ed il caricamento di oggetti da e su file;
- **Jython:** implementazione del linguaggio Python in linguaggio Java; consente di eseguire script Python dal linguaggio Java e codice Java all'interno di script Python;
- **JEP:** framework che sfrutta il servizio di naming di Java, JNI, per eseguire un interprete CPython all'interno della Java Virtual Machine, in modo "embedded"; consente di eseguire script Python dal linguaggio Java e codice Java all'interno di script Python, supportando anche l'inclusione di librerie scritte in C o altri linguaggi a basso livello;
- **Singleton:** design pattern che prevede che al più una sola istanza di una classe possa esistere allo stesso momento;
- **Adapter:** design pattern che prevede la creazione di una classe che faccia da interfaccia tra due sistemi, due linguaggi o due tecnologie differenti;
- **Metodi synchronized:** synchronized è una keyword Java utilizzata nell'ambito della sincronizzazione tra thread; quando si accede ad una sezione di codice denotata con la keyword (es. metodo), viene posto un *lock* sull'oggetto corrente al fine che nessun altro thread, fino al termine dell'esecuzione della porzione di codice, possa accedere a tale oggetto;
- **Race condition:** situazione in cui il risultato di un'operazione è determinato dall'ordine in cui sono eseguiti i processi o i thread che sono coinvolti in essa; la sezione di codice che può generare una condizione di questo tipo è detta *sezione critica*;
- **Proxy:** design pattern che prevede che un oggetto non sia disponibile immediatamente al consumo, ma venga sostituito da un "surrogato", detto Proxy, che supporta un sottoinsieme limitato delle sue operazioni, ma avente la stessa interfaccia pubblica dell'oggetto originale. Al momento della richiesta di una delle operazioni non supportate dal proxy, esso viene sostituito con l'oggetto effettivo, completando l'inizializzazione. Viene utilizzato quando la creazione dell'oggetto reale è particolarmente onerosa e si preferisce ritardarla il più possibile;
- **Sessione:** spazio di memoria riservato ad un utente di un servizio web, che persiste tra una richiesta e l'altra, ma viene deallocato una volta che l'utente smette di utilizzare il servizio;
- **Servlet:** classe Java eseguita su un server atta alla gestione di richieste HTTP da parte degli utenti di un servizio web;
- **JSP:** particolare tipologia di servlet il cui scopo è scrivere codice HTML sulla risposta HTTP. Sono scritte in un linguaggio che combina codice HTML e codice Java;