

Will I like it?

Game Recommender

2020-2021



STEAM®

AUTORI

ANTONIO SILLETTI

683588

FEDERICA MANGIALARDO

676371



SOMMARIO

1. Introduzione	2
2. Diagramma delle classi	3
3. Prima parte: Game Recommender	4
3.1 STEAM	4
3.2 IGDB e STEAMSpy	5
3.3 Bayesian Network	6
4. Seconda parte: Recommending tramite Clustering	8
5. Usage	9
5.1 NaiveRecommender_usage	10
5.2 Clustering_usage	13

1. Introduzione

‘*Will I like it? Game Recommender*’ è un progetto in Python diviso in due parti principali:

- La prima, in grado di predire quanto un determinato gioco possa piacere all’utente;
- La seconda, in grado di clusterizzare l’utente e i suoi amici, per poi consigliare giochi molto piaciuti di utenti che appartengono al suo stesso cluster.

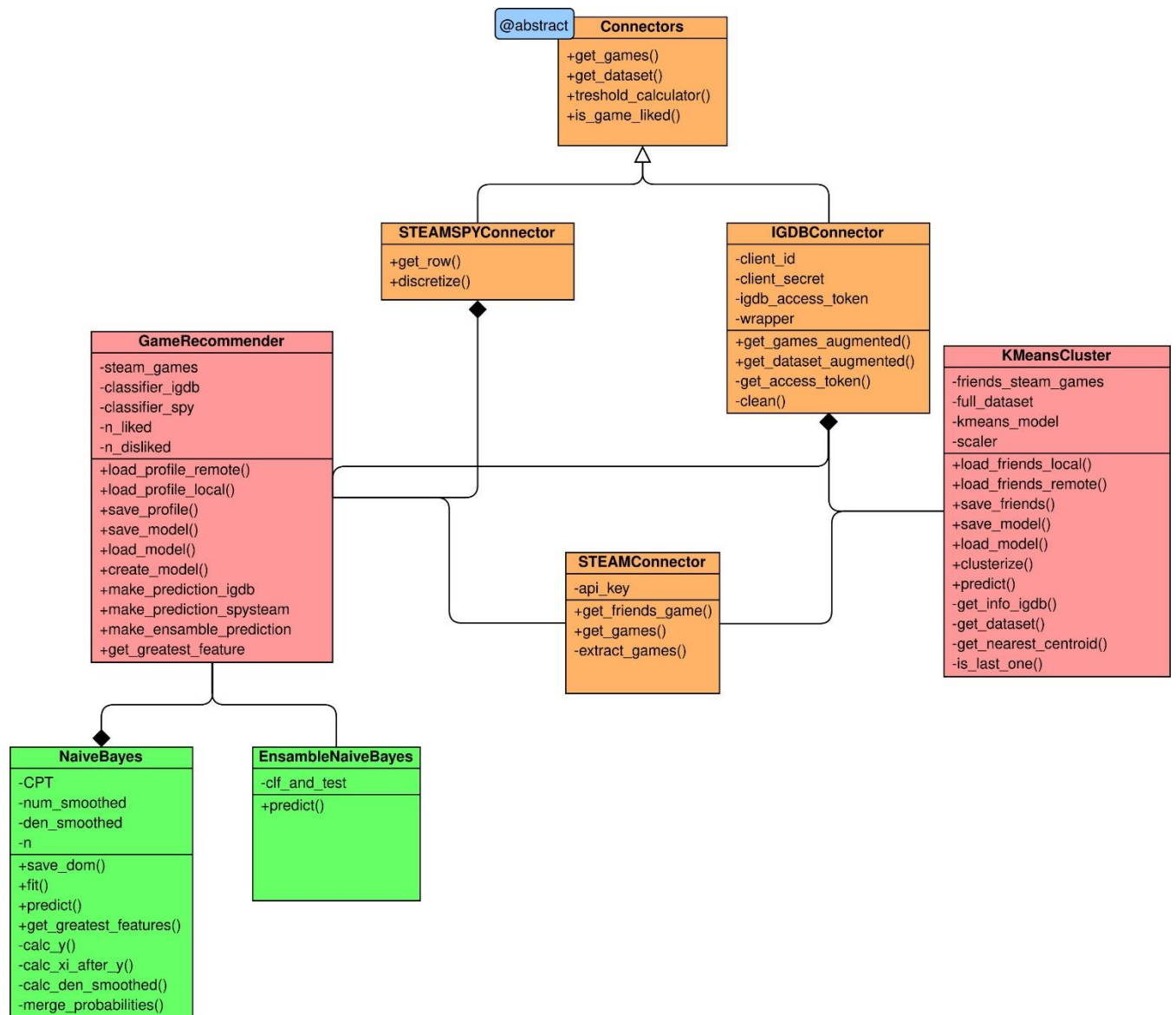
L’applicazione utilizza due sorgenti di informazione quali IGDB e SteamSpy, al fine di dare la possibilità di effettuare ensemble prediction su dataset contenenti diverse feature.

I profili sono estratti da STEAM e possono essere salvati in locale per una maggiore rapidità di utilizzo del progetto nelle sue parti, così come è possibile salvare i modelli addestrati.

Il progetto è installabile tramite ‘setup.py’, che installa in maniera automatica tutte le dipendenze, e contiene due esempi di utilizzo tipici, che verranno illustrati nel capitolo 4 (*Usage*).

Il codice è interamente documentato tramite docstring, nel caso si voglia espandere il progetto.

2. Diagramma delle classi



Come si evince dal diagramma delle classi, la classe *NaiveBayes* e *EnsambleNaiveBayes* sono due classi **Entità**, mentre la classe *GameRecommender* e *KMeansCluster* sono due classi di **Controllo** in quanto regolano la logica delle due parti del progetto.

Le 3 classi *STEAMConnector*, *IGDBConnector*, *STEAMSPYConnector* sono invece classi di tipo **Boundary**, in quanto si interfacciano con l'esterno (Steam, IGDB, SteamSpy rispettivamente).

3. Prima parte: Game Recommender

Idea: dato un gioco, capire se può essere gradito dall'utente o meno.

Di solito, nei recommender system di questo tipo, la raccolta dei rating è esplicita: vengono cioè richiesti alcuni item piaciuti in maniera diretta, cosicché il RS possa manipolare tali rating per effettuare le predizioni.

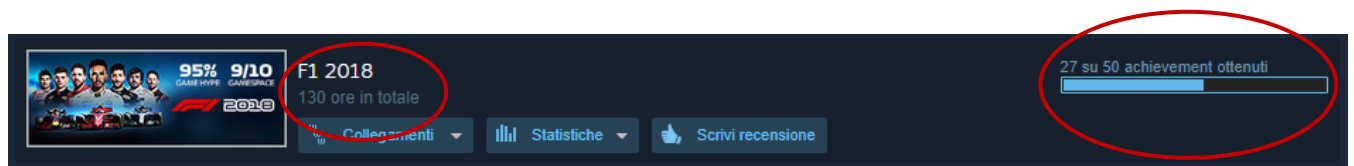
Sin dall'inizio del progetto, l'obiettivo è sempre stato quello di basarsi su rating impliciti, dove è il sistema che interpreta i gusti dell'utente.

In questo caso, è stata fatta un'assunzione intuitiva che determina se un item (gioco) è stato gradito dall'utente o meno. Prima di esplicitarla, vanno descritte quali informazioni è possibile estrarre da un profilo STEAM.

3.1 STEAM

Steam è la principale piattaforma di distribuzione digitale di videogiochi, ma offre anche una parte social in cui è possibile interagire con gli altri utenti.

Steam, inoltre, tiene traccia dei giochi acquistati dagli utenti, nonché di altre informazioni, tra cui le ore di gioco e la percentuale di obiettivi completati in un determinato gioco:



E sono proprio quest'ultime due informazioni che il progetto utilizza per determinare se un utente abbia gradito o meno un gioco:

dopo aver trovato la media delle ore di gioco giocate su tutti i giochi del profilo e la media della percentuale di obiettivi raccolti su tutti i giochi del profilo,

SE un dato gioco è stato giocato in un numero di ore maggiore alla media
OR

SE per un dato gioco sono stati raccolti un numero di obiettivi superiore alla media,

THEN il gioco è piaciuto, ELSE il gioco non è piaciuto.

Inoltre, per evitare di inserire rumore nel dataset sono stati esclusi i giochi giocati meno di 30 minuti (poiché probabilmente solo acquistati e mai giocati). Per collegarsi a Steam è stata usata un API (*steamapi*) da noi leggermente modificata ai fini del progetto, in quanto non veniva eseguito il calcolo della percentuale degli obiettivi raggiunti, ma solo indicati quali erano stati sbloccati e quali no.

3.2 IGDB e STEAMSpy

Dopo aver estratto il profilo, altro punto focale del recommender system è l'interazione con le basi di conoscenza per estrapolare una descrizione significativa degli item (giochi): la più famosa e più usata in questo ambito è *IGDB*.

Da essa è possibile estrapolare le più disparate informazioni. Le feature che sono state scelte ai fini del progetto sono le seguenti:

- *genres* → i generi del gioco
- *game_modes* → le modalità di gioco (singleplayer o multiplayer),
- *player_perspectives* → visuale di gioco (prima persona, terza, etc.),
- *themes* → tematiche del gioco
- *total_rating* → voto medio tra quelli della critica e quelli degli utenti
- *involved_companies* → compagnie che hanno creato il gioco

In un primo momento è stato pensato di costruire una rete Bayesiana con dipendenze, ma nella realtà tali dipendenze erano inesistenti o molto deboli: la visuale di gioco non dipende dal genere, il voto non dipende dalle tematiche trattate, etc.

Si è così virati verso una rete Bayesiana Naïve, dove tutte le feature sono per l'appunto indipendenti.

Alcuni utenti possono avere pochi giochi nel proprio profilo, con conseguente diminuzione delle performance del recommender, per questo si è pensato di dare la possibilità di applicare la **data augmentation**: specificando, al momento della creazione del modello, il parametro '*augmentation*', per ogni gioco contenuto nel profilo dell'utente vengono individuati *n* giochi (*n* è proprio il valore del parametro *augmentation*) che verranno poi aggiunti al dataset dell'utente. Per fare ciò, viene sfruttata la funzionalità di IGDB che è in grado di restituire una lista di giochi simili ad un determinato gioco.

Tuttavia, dato che IGDB spesso aveva alcune feature tra quelle scelte non avvalorate, si è pensato di affiancarlo con un'altra sorgente di informazione. Inizialmente, il candidato ideale è stato individuato in RAWG (di cui è ancora presente commentata l'implementazione della classe RAWGConnector), successivamente poi scartato poiché le API diverranno a pagamento il primo aprile.

È stato dunque scelto STEAMSpy, altro database contenente informazioni simili a quelle di RAWG. Le feature estratte da tale database sono:

- *genres* → i generi del gioco
- *publishers* → i publisher del gioco
- *positive* → Il numero di recensioni positive su STEAM del gioco
- *negative* → Il numero di recensioni negative su STEAM del gioco
- *average_forever* → Ore medie degli utenti passate sul gioco
- *tags* → I tag caratteristici del gioco

Dopo aver strutturate le informazioni estratte sia da IGDB che da STEAMSpy, è possibile allenare due (o più) classificatori NaiveBayes per poter eseguire una **predizione ensemble**, che allo stadio attuale è una semplice media delle predizioni effettuate dai classificatori allenati sulle diverse feature estratte dalle due sorgenti di informazione.

È comunque possibile effettuare le singole predizioni dei singoli classificatori con appositi metodi.

3.3 Bayesian Network

Come già detto, la rete Bayesiana è di tipo Naive, è stata creata manualmente e non sono state fatte particolari assunzioni o modifiche rispetto all'algoritmo classico. Viene calcolata la prob. a priori, le prob. a posteriori e vengono salvate nella CPT. Per evitare le probabilità nulle viene usato lo **smoothing di Laplace**. Per esempio, il calcolo della probabilità a posteriori per la feature genere diventa:

$$P(\text{genre} = \text{rpg} | \text{likes}) = \frac{\text{positive} + \alpha}{N + \alpha * K}$$

Dove:

- *positive* è il numero di item piaciuti aventi come genere 'rpg'

-
- N è il numero totale degli item piaciuti
 - α è il parametro di smoothing (impostato a **1** nel nostro caso)
 - K è il numero di feature

Particolarità dell'implementazione è che, per calcolare le probabilità a posteriori, si dovrebbero calcolare le prob. su tutti i possibili valori del dominio. Tuttavia, ciò viene evitato considerando come dominio tutti i valori presenti nel dataset dell'utente, non tutti quelli *realmente* possibili.

ESEMPIO:

- Se una persona ha giocato solo giochi di genere *rpg* e *action*, le prob. a posteriori calcolate per il 'genere' sono solo:
 - $P(\text{genre} = \text{rpg} | \text{likes})$
 - $P(\text{genre} = \text{rpg} | - \text{likes})$
 - $P(\text{genre} = \text{action} | \text{likes})$
 - $P(\text{genre} = \text{action} | - \text{likes})$

Se si vuole calcolare la predizione su un gioco che abbia come genere 'adventure', ma tale valore non è presente nella CPT, la prob. a posteriori per quel valore dovrà avere valore minimo, cioè proprio quello indicato dallo *smoothing di Laplace*.

La classe GameRecommender dà inoltre la possibilità di visualizzare i valori delle feature più graditi dall'utente (quelli che hanno le prob. a posteriori più alte) tramite il metodo *get_greatest_feature()*, e di salvare e caricare su disco locale sia il profilo dell'utente di STEAM, sia il modello già addestrato, per velocizzare la fase di predizione (la fase *online*).

4. Seconda parte: Recommending tramite Clustering

Idea: dato un utente, clusterizzare i suoi amici in base ai loro giochi giocati, e consigliare all'utente giochi che gli utenti appartenenti al suo stesso cluster hanno gradito e che non sono presenti nel suo profilo.

Per realizzare ciò, è stata usata la libreria *sklearn* che contiene algoritmi di apprendimento automatico allo stato dell'arte. Come algoritmo di clustering viene usato il KMeans.

Il primo passo è quello di estrarre il profilo degli utenti da STEAM. Inizialmente, si è pensato di clusterizzare gli interi profili, tuttavia si sarebbe dovuto fare feature selection attraverso il transformer `DictVectorizer()` di *sklearn* data la struttura dati utilizzata per contenere le informazioni dei giochi, ma si è preferito una soluzione alternativa:

- Clusterizzare i singoli giochi,
- Assegnare ai singoli utenti un cluster in base ai cluster dei giochi giocati: ossia quello più frequente all'interno del suo profilo

I dati, prima di essere clusterizzati, vengono normalizzati per evitare problemi legati alla diversa scala delle variabili tramite *StandardScaler()* di *sklearn*.

Prima però i valori delle feature vengono trasformati da liste di id a valori float effettuando la media dei valori contenuti nella lista. Questo permette a *sklearn* di lavorare senza ulteriore bisogno di manipolare il dataset.

- e.g. *genre* = [1, 5] dove 1 rappresenta il genere 'rpg', '5' rappresenta il genere 'azione' diventa *genre* = 3 dopo la trasformazione

In fase di predizione, l'utente verrà clusterizzato esattamente com'è stato fatto in fase di fit (ridurre alla stessa scala le feature, assegnazione del cluster più frequente). Una problematica frequente quando gli amici dell'utente sono pochi, è quella che spesso gli viene assegnato un cluster in cui non sono presenti altri utenti: ciò è stato ovviato assegnandogli il cluster che abbia almeno un utente all'interno più vicino secondo la distanza euclidea. La soluzione qui illustrata è stata implementata tramite le funzioni *get_nearest_centroid()* e *is_last_one()*.

Per espandere gli item con le loro proprietà, tra le sorgenti implementate (spysteam e igdb) è stato scelto IGDB poiché da esso sono ritrovati gli id numerici delle proprietà mentre da SPYSteam vengono ritrovate le proprietà nel loro valore testuale.

Anche per la classe KMeansCluster si dà la possibilità di salvare e caricare su disco locale sia i profili di steam degli amici dell'utente, sia il modello già addestrato, per velocizzare la fase di predizione (la fase *online*).

5. Usage

Per poter usare il progetto, è possibile installarlo tramite *setup.py* o tramite pip con il seguente comando:

```
pip install git+https://github.com/Silleellie/GameRecommender.git
```

Verranno installate tutte le dipendenze necessarie al corretto funzionamento del progetto. Dopo averlo installato è possibile utilizzarlo in maniera diretta. Seguono due esempi di utilizzo sia per la classe GameRecommender, sia per la classe KMeansCluster, che è possibile ritrovare nel root di progetto github come 'NaiveRecommender_usage.py' e 'Clustering_usage.py' rispettivamente. Il progetto è comunque interamente documentato tramite docstring, nel caso lo si vorrà estendere in futuro.

5.1 NaiveRecommender_usage

Per poter usare il game recommender, si deve prima istanziare un oggetto della classe relativa, poi si deve estrarre il profilo dell'utente. È possibile estrarlo:

1. Da remoto con il metodo `load_profile_remote()`
2. Da locale con il metodo `load_profile_local()`

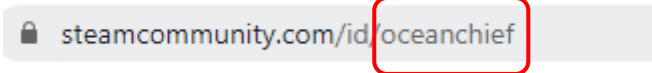
```
rec = GameRecommender()

rec.load_profile_remote('oceanchief')

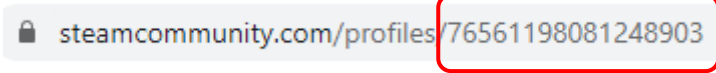
# OPPURE

rec.load_profile_local('example_database/single_profile/oceanchief.dat')
```

Nel primo caso va inserita la parte finale dell'url che punta al proprio profilo:

 steamcommunity.com/id/oceanchief

Il metodo funziona anche nel caso non sia stato definito un id personalizzato:

 steamcommunity.com/profiles/76561198081248903

L'alternativa è quella di usare il metodo `load_profile_local()` indicando come parametro il filepath del file precedentemente salvato tramite metodo `save_profile()`. A quest'ultimo va passato il nome con cui salvare il profilo dell'utente su disco. Il file viene salvato come file `.dat`

```
# ESEMPIO DI ESTRAZIONE DEL PROFILO DA REMOTO E SALVATAGGIO
# SU DISCO
rec.load_profile_remote('oceanchief')

rec.save_profile('oceanchief')

rec.load_profile_local('oceanchief.dat')
```

Dopo aver estratto il profilo (da remoto o da locale), si crea il modello tramite il metodo `create_model()` che addestra due classificatori naive sia sul dataset di igdb sia di spysteam, previo ritrovamento delle informazioni da entrambe le sorgenti di conoscenza.

```
rec.create_model()
```

In alternativa è possibile caricare in memoria un modello già addestrato e precedentemente salvato tramite il metodo `save_model()` e `load_model()`

```
# ESEMPIO DI SALVATAGGIO DEL MODELLO SU DISCO E
# CARICAMENTO
rec.create_model()

rec.save_model("oceanchief_rec_model")

rec.load_model('example_database/rec_model/oceanchief_rec_model.rec')
```

Una volta creato o caricato il modello, è possibile effettuare predizioni. Serve sapere l'appid del gioco (id univoco del gioco assegnato da steam), che è possibile ricavare andando sulla pagina di steam del gioco scelto e copiando la penultima parte dell'url:

store.steampowered.com/app/421020/DiRT_4/

Quindi, è possibile effettuare la predizione utilizzando il classificatore allenato sul dataset di IGDB tramite metodo `make_prediction_igdb()` al quale va passato come parametro l'appid del gioco. Il metodo ritorna sia la predizione, sia il nome del gioco.

```
pred, game_name = rec.make_prediction_igdb('421020')
print("-----")
print("La probabilità secondo dataset igdb che {} ti piaccia è del {}".format(game_name, pred))
#
```

Output:

```
-----
La probabilità secondo dataset igdb che DiRT 4 ti piaccia è del 0.9337088450944376
-----
```

Oppure, è possibile effettuare la predizione utilizzando il classificatore allenato sul dataset di STEAMSpy tramite metodo `make_prediction_steamspy()` al quale va passato come parametro l'appid del gioco. Il metodo ritorna sia la predizione, sia il nome del gioco:

```
pred, game_name = rec.make_prediction_steamspy('421020')
print("-----")
print("La probabilità secondo dataset steamspy che {} ti piaccia è del {}".format(game_name, pred))
```

Output:

```
-----
La probabilità secondo dataset steamspy che DiRT 4 ti piaccia è del 0.7367343676124897
-----
```

Oppure è possibile effettuare l'ensemble prediction utilizzando entrambi i classificatori addestrati:

```
pred, game_name = rec.make_ensemble_prediction('421020')
print("-----")
print("La probabilità ensemble che {} ti piaccia è del {}".format(game_name, pred))
```

Output:

```
-----
La probabilità ensemble che DiRT 4 ti piaccia è del 0.8352216063534637
-----
```

E' possibile inoltre visualizzare i valori e le feature più rilevanti per l'utente tramite il metodo `get_greatest_feature()`: esso restituisce due liste dei cinque tag e i cinque generi più graditi dall'utente.

```
tags, genres = rec.get_greatest_features()
print("-----")
print("A te piacciono molto i giochi con questi tags:")
print("{} , {} , {} , {} , {}".format(tags[0], tags[1], tags[2], tags[3], tags[4]))
#
print("-----")
print("A te piacciono molto i giochi con questi genere:")
print("{} , {} , {} , {} , {}".format(genres[0], genres[1], genres[2], genres[3], genres[4]))
print("-----")
```

Output:

```
-----  
A te piacciono molto i giochi con questi tags:  
singleplayer, action, multiplayer, strategy, adventure  
-----  
A te piacciono molto i giochi con questi genere:  
simulation, strategy, action, indie, sports  
-----
```

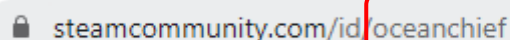
5.2 Clustering_usage

Per poter usare la parte relativa al Clustering, si deve prima istanziare un oggetto della classe relativa, poi si devono estrarre i profili degli amici dell'utente. È possibile estrarli:

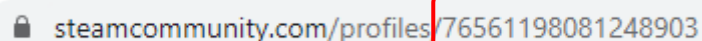
1. Da remoto con il metodo *load_friends_remote()*
2. Da locale con il metodo *load_friends_local()*

```
cl = KMeansCluster()  
  
cl.load_friends_remote('snoxe')  
  
# OPPURE  
  
cl.load_friends_local("example_database/friends_profile/snoxe_friends.dat")
```

Nel primo caso va inserita la parte finale dell'url che punta al proprio profilo:

 steamcommunity.com/id/oceanchief

Il metodo funziona anche nel caso non sia stato definito un id personalizzato:

 steamcommunity.com/profiles/76561198081248903

L'alternativa è quella di usare il metodo *load_friends_local()* indicando come parametro il filepath del file precedentemente salvato tramite metodo *save_friends()*. A quest'ultimo va passato il nome con cui salvare i profili degli amici dell'utente su disco. Il file viene salvato come file *.dat*

```
# ESEMPIO DI ESTRAZIONE DEI PROFILI DEGLI AMICI E SALVATAGGIO
# SU DISCO
cl.load_friends_remote('snoxe')

cl.save_friends('snoxe_friends')

cl.load_friends_local("example_database/friends_profile/snoxe_friends.dat")
```

Dopo aver estratto i profili degli amici (da remoto o da locale), si effettua il clustering tramite il metodo *clusterize()*.

```
cl.clusterize()
```

In alternativa è possibile caricare in memoria un modello già addestrato e precedentemente salvato tramite il metodo *save_model()* e *load_model()*

```
# ESEMPIO DI SALVATAGGIO DEL MODELLO SU DISCO E
# CARICAMENTO
cl.clusterize()

cl.save_model('friends_armadillo_clusterized')

cl.load_model('example_database/cluster_model/friends_armadillo_clusterized.cl')
```

Dopo aver creato o caricato il modello, è possibile effettuare la predizione tramite il metodo *predict()* che restituisce un dataframe contenente i 5 giochi più giocati dagli amici appartenenti allo stesso cluster dell'utente passato come parametro a *predict()*.

```
print(cl.predict('silleellie').to_string(index=False))
print("-----")
```

Output:

```
Gioco Nome_amico
PAYDAY 2 slashhouse13
Garry's Mod slashhouse13
7 Days to Die slashhouse13
Counter-Strike: Global Offensive slashhouse13
Monster Hunter: World D A N I E L W A G O N
-----
```

Si noti come è possibile effettuare predizioni anche di utenti non strettamente legati ai profili di cui si è effettuato il clustering (e.g. si clusterizza sugli amici di 'silleellie', chiamo la predict sul profilo di 'oceanchief').