

# ”Le Iene” \*IA

## Documentazione Progetto FIA

Nasto Maria Chiara  
Cardaropoli Giuseppe  
Ercolino Andrea †

Marzo 2021

## Contents

<b>1</b>	<b>Introduzione</b>	<b>2</b>
<b>2</b>	<b>Raccolta dei Dati</b>	<b>2</b>
2.1	Studio dei Dati . . . . .	2
<b>3</b>	<b>Implementazione</b>	<b>3</b>
3.1	Clustering . . . . .	3
3.1.1	Motivazioni . . . . .	3
3.1.2	K-Means . . . . .	4
3.1.3	Fase di Testing . . . . .	4
3.2	Scelta del film . . . . .	6
3.3	Microservizio. . . . .	8
3.4	Risultato finale . . . . .	9

---

\*Citazione al cult ”Le Iene” del 1982 diretto da Quentin Tarantino

†GitHub repo: <https://github.com/x-mariachiara/CineHub-ModuloFIA>

# 1 Introduzione

Il sistema che si intende realizzare è un microservizio che fornisce a **CineHub** la possibilità di consigliare ai propri utenti un film da guardare, sulla base dei propri gusti, hobby ed età. Lo scopo è quello di permettere agli utenti di **CineHub** di scoprire un nuovo film che potrebbe piacergli e invogliarlo a guardarla. Il modulo in quanto micro-servizio potrà essere anche riutilizzato in altri sistemi. In questa documentazione illustriamo le fasi che ci hanno portato alla realizzazione del sistema partendo dalla raccolta dei dati, avvenuta tramite un questionario online, fino alla fase implementativa nella quale abbiamo prima effettuato uno studio dei dati raccolti, per poi procedere alla loro standardizzazione e clustering, terminando con l'implementazione dell' endpoint della RESTapi.

## 2 Raccolta dei Dati

Per la raccolta dei dati, non avendo a disposizione un dataset, abbiamo costruito un questionario nel quale abbiamo raccolto le seguenti informazioni:

- Sesso
- Fascia d'età
- Genere cinematografico preferito
- Periodo della giornata preferito per guardare un film
- Hobby
- Cosa fai durante la visione di un film?
- Preferisci guardare un film in compagnia o da solo?
- Quanti film guardi in media in una settimana?
- Quante puntate di serie tv guardi in media in una settimana?

Abbiamo ottenuto 382 risposte che coprivano la quasi totalità delle fasce d'età che avevamo considerato.

### 2.1 Studio dei Dati

Una volta terminata la fase di raccolta dei dati, dalle risposte ottenute dai questionari abbiamo costruito un dataset per procedere all'analisi degli stessi e attraverso la matrice di correlazione abbiamo studiato quali coppie di attributi erano più correlate tra loro.

Analizzando la matrice di correlazione dei dati abbiamo notato che le variabili erano molto poco correlate tra loro; sospettiamo sia dovuto ad una scarsità di dati e all'inesperienza nel raccoglierli.

**HeatMap.** Attraverso vari tentativi abbiamo individuato che gli attributi dei campioni migliori da considerare fossero: hobby, sesso, genere preferito e fascia d'età.

**PairPlot.** Per identificare l'eventuale presenza di pattern tra le variabili abbiamo generato il Pair Plot relativo ad ogni coppia possibile tra le variabili: hobby, sesso, genere preferito e fascia d'età da noi considerate.

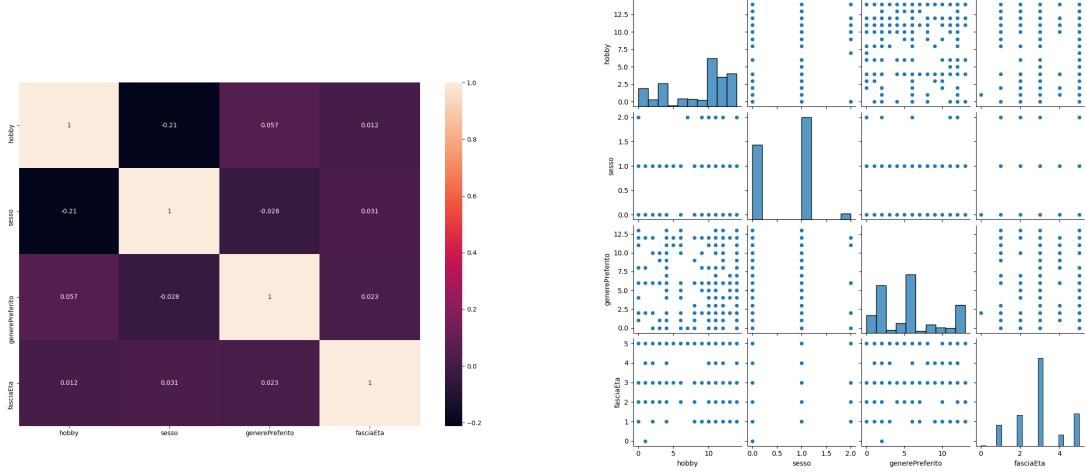


Figure 1: heatmap costruita attraverso la matrice di correlazione e pairplot di tutte le coppie di variabili maggiormente correlate

**Standardizzazione.** Successivamente abbiamo utilizzato un label encoder in modo tale da trasformare tutte le stringhe (genere preferito, fascia d’età) in un array binario. I valori dell’attributo *sesso* sono stati etichettati con i numeri da 0 a 2 (*uomo*, *donna*, *preferisco non rispondere*), invece quelli dell’attributo *hobby* sono stati etichettati con i numeri da 0 a 13. Tutto questo per evitare problemi legati alla diversa scala delle variabili. Infatti la standardizzazione è quella operazione che rende la distribuzione di una variabile aleatoria ”standard”, ovvero riconduce  $\mu = 0$  e  $\sigma = 1$ .

### 3 Implementazione

Per l’implementazione abbiamo utilizzato il linguaggio di scripting python e le seguenti librerie:

- pandas e numpy per la gestione dei dati;
- seaborn e matplotlib per la visualizzazione dei dati, creazione della heatmap e dei pairplot;
- sklearn per la creazione e manipolazione del modello;
- flask per la realizzazione della RESTapi.

#### 3.1 Clustering

Il clustering è un problema di apprendimento non supervisionato in quanto non conosciamo le etichette dei dati, consiste nel raggruppare i dati in cluster analizzando la similarità dei dati. Abbiamo scelto di utilizzare il clustering poiché esso permette di raggruppare facilmente gruppi di utenti in base alle loro caratteristiche.

Realizzare un buon modello attraverso algoritmi di clustering è molto complesso poiché i clusters possono manifestarsi in diverse forme e dimensioni.

##### 3.1.1 Motivazioni

Abbiamo scelto di utilizzare il Clustering poiché abbiamo deciso di raggruppare gli utenti simili tra loro all’interno di cluster, in modo tale da ridurre il range dei possibili film da consigliare ad un utente, considerando solo quelli visti dagli altri utenti del medesimo cluster ipotizzando una similarità di gusti tra di essi.

### 3.1.2 K-Means

L'algoritmo di clustering che abbiamo scelto di utilizzare è il K-Means, ovvero un algoritmo di partizionamento basato sulla somma degli errori quadratici. Il K-Means funziona nel seguente modo:

1. Vengono selezionati k centroidi;
2. Per ogni campione del dataset valutiamo la distanza del campione da tutti i centroidi e lo assegniamo al più vicino. Al termine di questa operazione saranno generati k cluster;
3. I centroidi potrebbero non combaciare con i cluster generati e quindi venire ricalcolati;
4. Vengono ripetutiti i punti 2 e 3 finché la variazione di ogni centroide non supera una certa soglia.

La scelta di k, ovvero di quanti cluster vogliamo realizzare, non deve essere né troppo grande né troppo piccolo. Per scegliere il miglior k abbiamo calcolato l'indice di silhouette e l'elbow point numerose volte.

**Indice di Silhouette.** Quantifica quanto i dati sono ben disposti nei cluster assegnati, considerando quanto bene sono ammassati i dati nel cluster e quanto ogni dato è distanza da qualsiasi altro cluster. Questo coefficiente di forma si valuta in funzione del numero di cluster e si cerca il valore più alto possibile (compreso tra -1 e 1).

**Elbow Point.** Costruiamo un grafico che rappresenta l'andamento del valore della somma degli errori quadratici al variare del numero cluster. Dato questo grafico andiamo ad individuare il punto a gomito. Da quel punto in poi la somma degli errori quadratici potrebbe diminuire ma non significativamente, in quanto all'aumentare dei cluster aumenta la complessità e potrebbe diminuire la somma degli errori.

### 3.1.3 Fase di Testing

Abbiamo realizzato il modulo python `clustering.py` contenente tutte le operazioni necessarie alla realizzazione dei grafici, all'analisi dei dati e al clustering; in modo da poter essere usato sia durante la fase di testing sia dal modulo `service.py` che implementa la logica di business del microservizio.

Tra i metodi implementati nel modulo `clustering.py`, oltre a quelli che realizzano i grafici troviamo:

Listing 1: Modulo Clustering .`fitModel(n-cluster)`

---

```
def fitModel(n_cluster = 1):
    scaled_dataframe = standardizeDataset()
    KMeans_model = KMeans(n_clusters = n_cluster)
    KMeans_model.fit(scaled_dataframe)
    scaled_dataframe[ 'cluster' ] = KMeans_model.labels_
    return KMeans_model, scaled_dataframe
```

---

.`fitModel()` usato appunto per effettuare il fit del modello usando `n-cluster`, questo metodo è principalmente utilizzato dal modulo `service`.

Listing 2: Modulo Clustering .`bestKMeans()`

---

```
def bestKMeans(n_cluster = 2):
    scaled_dataframe = standardizeDataset()
    k_to_test = range(2, n_cluster, 1)
    silhouette_scores = []
    elbow_scores = []

    for k in k_to_test:
        model_kmeans_k = KMeans( n_clusters = k )
        model_kmeans_k.fit( scaled_dataframe )
        labels_k = model_kmeans_k.labels_
```

---

```

score_k = metrics.silhouette_score(scaled_dataframe, labels_k)
silhouette_scores[k] = score_k
elbow_scores.append(model_kmeans_k.inertia_)

return silhouette_scores, elbow_scores

```

---

Per scegliere il  $k$  migliore abbiamo rieseguito il metodo `.bestKMeans()` 50 volte per ogni  $k$  compreso tra 2 e 13. Ad ogni esecuzione abbiamo calcolato l'indice di silhouette e abbiamo scelto il  $k$  con l'indice più alto. Dalle nostre esecuzioni è risultato che il numero migliore di cluster da utilizzare fosse 10.

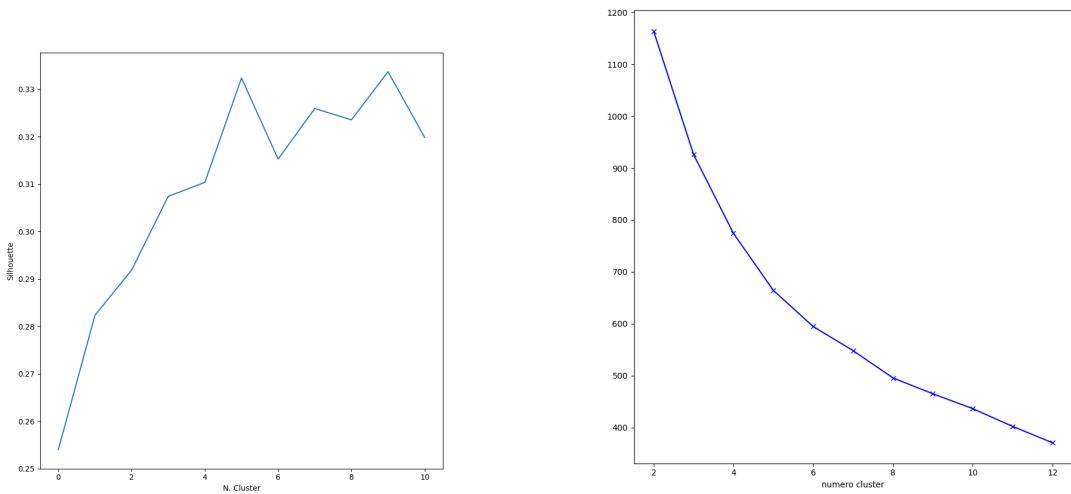


Figure 2: Indice di Silhouette e Elbow Point

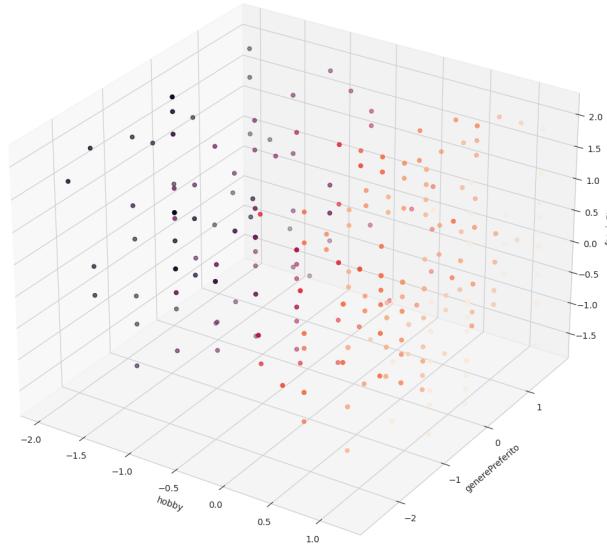


Figure 3: Scatterplot 3d  $K = 10$  e coordinate:  $x = \text{hobby}$ ,  $y = \text{fascia età}$ ,  $z = \text{genere preferito}$

### 3.2 Scelta del film

La scelta del film consigliato per un determinato utente avviene in tre fasi, e viene richiesta ogni volta che l'utente accede alla home. Nel caso in cui non vi sia nessun utente in sessione appare un film casuale dal catalogo. Per scegliere il film consigliato di un dato utente  $u$ , andiamo a trovare il suo cluster di appartenenza, poi andiamo a prendere tutti i film visti da tutti gli altri utenti che appartengono allo stesso cluster, poiché potrebbero avere gusti simili.

Otteniamo l'insieme dei film visti dagli utenti che appartengono allo stesso cluster di  $u$  andando a escludere i film già visti da  $u$  (*quelli che u ha già recensito*); una volta ricavatoci questo insieme andiamo ad assegnare un punteggio ad ogni film in questo modo:

- ad ogni film viene assegnato un punteggio base pari a cinque;
- se nel film hanno recitato uno o più attori presenti nella lista degli attori preferiti di  $u$  al punteggio base del film viene sommato cinque;
- se tra i generi del film vi è quello preferito di  $u$  al punteggio base del film viene sommato cinque;
- se non si verifica nessuno dei casi precedenti il punteggio del film diventa due;

Una volta assegnato un punteggio ad ogni film procediamo con un algoritmo ispirato ad un algoritmo di tipo Roulette Wheel per estrarre il film da consigliare ad  $u$ .

Ad ogni Film viene quindi assegnato un settore di una circonferenza di arco dato dal suo punteggio relativo, ovvero:

$$P(\text{film}_i) = \frac{\text{punteggio}(\text{film}_i)}{\text{Totale punteggi}}$$

Usando poi l'interfaccia `Random` di Python andiamo ad estrarre un float tra 0 e 1 e controlliamo in quale porzione del cerchio ricade, scegliendo così il vincitore.

Abbiamo scelto questo approccio per evitare di consigliare sempre film con punteggio massimo, infatti ogni film ha una probabilità di essere scelto proporzionale al proprio punteggio.

**Esempio.** Sia  $u$  un utente già registrato, di cui conosciamo le seguenti informazioni:

- Email ( $PK_{utente}$ ): "andrea@cinehub.it"
- Sesso: 0
- Fascia Età: "19 - 25"
- Hobby: 9
- Genere Preferito: "Azione"
- Lista film visti ( $rispettivi\ id$ ): [543, 519, 400]
- Lista Attori in ordine di preferenza ( $rispettivi\ id$ ): [80, 40, 85]

andremo a controllare se la mail dell'utente è già presente nel modello. In caso positivo ci basterà andare a recuperare dal modello tutte le email degli utenti che appartengono allo stesso cluster di  $u$ , ad esempio  $g$  e  $m$ . Se  $g$  ha visto i film con id [543, 511, 404, 212] e  $m$  ha visto i film con id [543, 519, 511, 212] allora l'insieme di film consigliabili a  $u$  è: {511, 404, 212}. Inoltre: 511 ha generi ["Azione", "Drammatici"] e cast [40, 83, 81]; 404 ha generi ["Drammatici", "Thriller"] e cast [82, 83, 45]; 212 ha generi ["Azione", "Avventura"] e cast [65, 66, 70].

Quindi  $punteggio(511) = 15$ ;  $punteggio(404) = 2$  e  $punteggio(212) = 10$ . Il primo sarà scelto con probabilità  $\frac{15}{27}$ , il secondo con probabilità  $\frac{2}{27}$  e il terzo con probabilità  $\frac{10}{27}$ .

### 3.3 Microservizio.

Il modulo di Intelligenza Artificiale ”Le Iene” è stato implementato come microservizio in modo tale da poter essere utilizzato anche da altri sistemi in maniera trasparente. Inizialmente abbiamo definito un endpoint, ovvero:

Listing 3: RESTapi endpoint /consigliato

---

```
@app.route("/consigliato", methods=["POST"])
def get_consigliati():
    data = json.dumps(request.get_json()) # DTO con utente, recensioni dell'utente
    utente = service.jsonToUtente(data)
    list_simili = service.getSimilar(utente.email)
    winner = service.rouletteWheel(list_simili, utente, dao)
    print("Winner:", winner)
    return str(winner)
```

---

che è in ascolto di una richiesta POST. Il body della richiesta conterrà email, data di nascita, hobby, sesso, genere preferito, eta, film visti e attori preferiti dell’utente a cui consigliare un film e restituirà l’id del film consigliato.

Il modulo `service` implementa la logica di business, in particolare vi sono i metodi `.getSimilar()` e `.rouletteWheel()`.

Listing 4: Modulo Service .getSimilar()

---

```
def getSimilar(self, utente_id: str) -> list:
    *r, self._scaled_dataframe = fitModel(n_cluster=10)
    self._dataset_dataframe[ 'cluster' ] = self._scaled_dataframe[ 'cluster' ]
    self._dataset_dataframe.info()
    list_email = self._dataset_dataframe[ 'email' ].tolist()
    indice = list_email.index(utente_id)
    cluster_appartenenza = self._dataset_dataframe.iat[indice, 7]
    list_email_simili = self._dataset_dataframe.loc[
        self._dataset_dataframe[ 'cluster' ] == cluster_appartenenza, 'email'
    ].tolist()

    return list_email_simili
```

---

Il metodo `.getSimilar()` data l’email di un utente  $u$  restituisce la lista delle email degli utenti che appartengono allo stesso cluster di  $u$ .

Listing 5: Modulo Service .rouletteWheel()

---

```
def rouletteWheel(self, lista_simi: list, utente, dao) -> float:
    ruota = []
    posizione = 0.0
    lista_film_consigliati = set()

    for u in lista_simi:
        for film in self._dataset_dataframe.iat[lista_simi.index(u), 5]:
            if not film in utente.id_film_visti:
                lista_film_consigliati.add(film)

    totalePunteggi = sum(
        [self.getPunteggioFilm(x, utente, dao) for x in lista_film_consigliati])

    for idFilm in lista_film_consigliati:
        punteggio_film = self.getPunteggioFilm(idFilm, utente, dao)
        ruota.append(Spicchio(posizione, punteggio_film / totalePunteggi, idFilm))
        posizione += punteggio_film / totalePunteggi

    randomGenerator = Random()
    randomNumber = round(randomGenerator.uniform(0, 1), 6)
    for index, spicchio in enumerate(ruota):
        if spicchio.posizioneIniziale <= randomNumber < (spicchio
            .posizioneIniziale + spicchio.lunghezza):
            return spicchio.idFilm
```

---

Il metodo `.rouletteWheel()` data la lista delle email di utenti nello stesso cluster di  $u$  restituisce l'id del film consigliato.

### 3.4 Risultato finale

