

Netflix ChatBot



Data Science

Corso di Laurea Magistrale in Ingegneria Informatica e
dell'Automazione
Università Politecnica delle Marche

Christopher Buratti
Luca Guidi
Vito Scaraggi
Alessandro Rongoni

Anno accademico 2023/2024

Indice

1	Introduzione	2
1.1	ChatBot	2
1.2	Framework RASA	3
1.3	Altri strumenti utilizzati	4
2	Netflix ChatBot	6
2.1	Il Dataset	6
2.2	La struttura	7
3	Funzionalità del Chatbot	9
3.1	Visualizzare tutti i generi dei contenuti presenti	9
3.2	Visualizzare tutti i paesi dei contenuti presenti	10
3.3	Visualizzare tutte le categorie di visione dei contenuti presenti	11
3.4	Visualizzare tutte le informazioni del contenuto tramite il titolo	13
3.5	Action di ricerca contenuti filtrati a seconda delle preferenze dell'utente	15
3.6	Ottenere consigli di visione in base alle preferenze dell'utente (FORM)	23
4	Connessione a Telegram e Test	28
4.1	Connessione	28
4.2	Testing	28
5	Conclusioni e sviluppi futuri	32

1 Introduzione

L'obiettivo di questa relazione è illustrare i dettagli della progettazione e dell'implementazione di un ChatBot dedicato a fornire informazioni sui titoli di film e serie TV presenti sulla piattaforma Netflix. Questo chatbot sarà progettato per guidare gli utenti nella ricerca di contenuti di loro interesse, fornendo dettagli come trama, cast, durata e altro ancora. Inoltre, il ChatBot consentirà agli utenti di visualizzare titoli filtrati secondo le loro esigenze e di ricevere consigli personalizzati in base alle loro preferenze.

1.1 ChatBot

Un **chatbot** è un software progettato per simulare e processare conversazioni umane, sia scritte che parlate, permettendo agli utenti di interagire con dispositivi digitali come se fossero in comunicazione con un altro essere umano. Questa simulazione è resa possibile attraverso l'impiego di tecniche di intelligenza artificiale, apprendimento automatico (machine learning) e elaborazione del linguaggio naturale (Natural Language Processing).

Per essere efficace, un chatbot dovrebbe possedere le seguenti caratteristiche fondamentali:

- **Interattività:** La capacità di sostenere una comunicazione bidirezionale, comprendendo gli input forniti dagli utenti e fornendo risposte pertinenti attraverso l'uso del deep learning e dell'elaborazione del linguaggio naturale.
- **Iteratività:** La capacità di ricordare le interazioni precedenti con l'utente e di fornire informazioni pertinenti in base al contesto specifico dell'interazione corrente.
- **Additività:** La flessibilità di non essere limitato a un singolo compito, ma di essere dinamico e adattabile, imitando le capacità di un cervello umano e reagendo all'ambiente circostante.
- **Contestualizzazione:** La capacità di identificare ed estrarre elementi contestuali dall'input dell'utente, come il compito, l'obiettivo, il tempo e il luogo, al fine di fornire un output adeguato e mirato.

I chatbot offrono numerosi vantaggi alle aziende che li impiegano, in quanto sono disponibili 24 ore su 24, 7 giorni su 7, e possono completare i compiti in meno tempo rispetto a un operatore umano.

Esistono due principali categorie di chatbot:

1. **Chatbot dedicati alle attività (dichiarativi):** Sono programmi monouso focalizzati su una specifica funzione. Operano principalmente tramite regole, elaborazione del linguaggio naturale (NLP) e limitato impiego di machine learning (ML), generando risposte automatizzate ma colloquiali alle richieste degli utenti. Questi chatbot sono altamente specifici e strutturati, adatti per funzioni di assistenza e servizio, come rispondere a domande frequenti o gestire semplici transazioni. Sebbene utilizzino la NLP, le loro capacità sono piuttosto basilari.
2. **Chatbot predittivi basati sui dati (di conversazione):** Noti anche come assistenti virtuali o digitali, sono più sofisticati, interattivi e personalizzati rispetto ai chatbot dedicati alle attività. Questi chatbot comprendono il contesto, sfruttano la comprensione della lingua naturale (NLU), la NLP e la ML per apprendere e migliorare nel tempo. Sono capaci di fornire raccomandazioni personalizzate, anticipare le esigenze degli utenti e avviare conversazioni. Esempi includono Siri di Apple e Alexa di Amazon.

Gli assistenti digitali avanzati possono anche coordinare diversi chatbot monouso, estrarre e combinare informazioni da ciascuno per eseguire compiti complessi, mantenendo il contesto per evitare confusione.

1.2 Framework RASA

RASA è un framework open source in Python dedicato alla creazione di chatbot conversazionali. Esso si basa sul machine learning supervisionato, insieme alle tecniche di Natural Language Processing (NLP), per comprendere gli intenti degli utenti e fornire ad essi una risposta coerente.



Figura 1: Logo RASA Framework

A fronte del grande successo riscontrato dai chatbot nel contesto aziendale, sono stati sviluppati molti altri framework per creare chatbot, come Dialogflow, Amazon Lex e Luis. Tuttavia, i vantaggi nell'utilizzo di Rasa sono diversi:

1. **RASA è open source:** ciò permette di avere a disposizione il codice e poter intervenire manualmente sul bot in caso di necessità, offrendo una maggiore flessibilità e personalizzazione.
2. **RASA è un progetto in continuo sviluppo:** con una vasta community di sviluppatori a supporto. Gli stessi creatori sono disponibili a fornire chiarimenti e a risolvere bug e problemi vari, garantendo così un'evoluzione costante del framework.
3. **RASA non è un servizio cloud:** può quindi essere ospitato in locale sulle macchine aziendali. Questo è un vantaggio soprattutto nel caso in cui nelle chat con gli utenti passassero dati sensibili che, per motivi di privacy, non dovrebbero essere esposti all'esterno. La privacy dell'utente viene così garantita, avendo tutti i dati in locale.

L'architettura RASA è costituita da due componenti fondamentali:

- **RASA NLU:** si occupa di capire e classificare la volontà dell'utente (chiamato anche intento), prendendo come input del testo "libero" scritto da quest'ultimo e restituendo dati strutturati.
- **RASA CORE:** dopo aver classificato il messaggio dell'utente, elabora quella che sarà la risposta sulla base di ciò che è stato recepito al momento presente, ma anche in passato.

RASA si basa sull'apprendimento supervisionato e quindi è necessario fornirgli dati di training, in particolare i cosiddetti intenti, che altro non sono che frasi di esempio che mappano le possibili motivazioni che un utente può avere per utilizzare il chatbot in questione. Ad ogni frase scritta dall'utente, infatti, corrisponde un'intenzione e l'obiettivo del chatbot, in prima istanza, sarà proprio quello di individuare e classificare correttamente tale aspetto, quindi si dovrà addestrare il chatbot fornendogli un opportuno numero di esempi per ciascuna tipologia di richiesta.

Una volta aver incluso tutti i possibili intenti relativi al dominio di interesse, il chatbot sarà in grado di classificare un input ricevuto associandogli l'intento che più si addice, sulla base di un punteggio. Quindi, una volta che la componente RASA NLU rileva l'intento dell'input, si attiva il RASA Core che crea un modello di machine learning per imparare dagli esempi forniti e predire la risposta più adatta da restituire all'utente, scegliendola tra le utterances.

Se invece la risposta prevede un'elaborazione più articolata, come ad esempio l'invocazione di una API o una query su database, si deve ricorrere alle Actions. Tutti i file da utilizzare hanno estensione .yaml, tranne le actions che sono un file Python.

1.3 Altri strumenti utilizzati

Pandas

Pandas è una libreria software scritta per il linguaggio di programmazione Python per la manipolazione e l'analisi dei dati. È particolarmente utile per lavorare con dati tabellari (come fogli di calcolo e database SQL) ed è ampiamente utilizzato in vari campi, tra cui il machine learning, la scienza dei dati, l'analisi statistica e l'automazione.

- **Strutture dati:** Pandas fornisce due strutture dati principali, **DataFrame** e **Series**, che consentono di lavorare in modo efficiente con dati tabellari e unidimensionali, rispettivamente.
- **Lettura e scrittura di file:** Pandas supporta vari formati di file, tra cui CSV, Excel, SQL, JSON e HTML, consentendo una facile lettura e scrittura dei dati.
- **Pulizia e preparazione dei dati:** Pandas offre funzionalità per gestire dati mancanti, unire e concatenare dati, trasformare e modificare colonne, filtrare righe e molto altro.
- **Analisi dei dati:** Con Pandas, è possibile eseguire operazioni di gruppo, aggregazioni, trasformazioni e calcoli statistici sui dati.
- **Manipolazione del tempo:** Pandas fornisce funzionalità avanzate per lavorare con serie temporali, inclusa la gestione di date e orari, la conversione di fusi orari e la generazione di intervalli di tempo.

Utilizzo di Pandas nei chatbot:

Nel contesto dei chatbot, Pandas può essere utilizzato per diverse finalità:

- **Gestione dei dati:** Pandas può essere utilizzato per organizzare e manipolare i dati utilizzati dal chatbot, come database di risposte predefinite, log delle conversazioni o informazioni sugli utenti.
- **Analisi delle conversazioni:** I log delle conversazioni possono essere analizzati con Pandas per estrarre statistiche utili, identificare le domande più frequenti o valutare le prestazioni del chatbot.
- **Integrazione con API e database:** Pandas può essere utilizzato per interfacciarsi con API esterne o database per recuperare informazioni richieste dagli utenti, come orari dei treni, previsioni meteo o dati di prodotti.

Esempio di codice Pandas

```
import pandas as pd

# Creazione di un DataFrame
data = {'Nome': ['Alice', 'Bob', 'Charlie'],
        'Età': [25, 30, 35],
        'Città': ['Roma', 'Milano', 'Torino']}
df = pd.DataFrame(data)

# Visualizzazione del DataFrame
print(df)

# Accesso ai dati
print(df['Nome'])

# Filtraggio dei dati
print(df[df['Età'] > 30])
```

In questo esempio, viene creato un DataFrame con dati su persone, viene visualizzato il DataFrame, si accede alla colonna "Nome" e si filtrano le righe in base all'età.

FuzzyWuzzy

FuzzyWuzzy è una libreria Python che viene utilizzata per la comparazione di stringhe, consentendo di determinare quanto due stringhe siano simili l'una all'altra. Si basa sull'algoritmo di distanza di Levenshtein per

calcolare le differenze tra sequenze. Questa libreria è particolarmente utile in casi in cui i dati testuali potrebbero contenere errori di battitura o variazioni nella denominazione, rendendo difficile il confronto diretto delle stringhe.

Caratteristiche principali di FuzzyWuzzy:

- **Calcolo del Rapporto di Somiglianza:** FuzzyWuzzy fornisce funzioni come `ratio()` e `partial_ratio()` per calcolare il rapporto di somiglianza tra due stringhe, restituendo un valore percentuale che indica quanto le stringhe siano simili.
- **Ricerca della Migliore Corrispondenza:** La funzione `get_best_match()` consente di confrontare una stringa con un elenco di stringhe per trovare quella con la somiglianza più alta.
- **Tokenizzazione:** FuzzyWuzzy supporta la tokenizzazione delle stringhe per migliorare l'accuratezza del confronto, utilizzando funzioni come `token_sort_ratio()` e `token_set_ratio()` che ordinano o combinano i token prima del confronto.

Esempio di utilizzo di FuzzyWuzzy:

```
from fuzzywuzzy import fuzz
```

```
# Confronto diretto
ratio = fuzz.ratio("Fuzzy Wuzzy era un orso", "Fuzzy era un orso")
print(f"Rapporto di somiglianza: {ratio}%")

# Confronto parziale
partial_ratio = fuzz.partial_ratio("Fuzzy Wuzzy", "Fuzzy")
print(f"Rapporto di somiglianza parziale: {partial_ratio}%")

# Confronto con tokenizzazione e ordinamento
token_sort_ratio = fuzz.token_sort_ratio("Fuzzy Wuzzy era un orso",
                                         "orso Fuzzy Wuzzy era")
print(f"Rapporto di somiglianza con tokenizzazione e ordinamento: {token_sort_ratio}%")
```

In questo esempio, vengono calcolati e stampati vari rapporti di somiglianza tra stringhe utilizzando diverse funzioni di FuzzyWuzzy.

Applicazioni di FuzzyWuzzy:

- **Correzione di errori di battitura:** FuzzyWuzzy può essere utilizzato per identificare e correggere errori di battitura nei dati testuali.
- **Matching di dati:** È utile per abbinare dati testuali provenienti da fonti diverse, come ad esempio il matching di nomi di prodotti in diversi database.
- **Ricerca approssimativa:** Può essere impiegato per implementare funzionalità di ricerca che tollerano piccole variazioni nella query dell'utente.

Installazione di FuzzyWuzzy:

FuzzyWuzzy può essere facilmente installato utilizzando pip, il gestore di pacchetti Python:

```
pip install fuzzywuzzy
```

Per un confronto più veloce, è possibile installare anche la libreria opzionale `python-Levenshtein`:

```
pip install python-Levenshtein
```

Questa libreria ottimizza i calcoli di distanza di Levenshtein, rendendo FuzzyWuzzy più rapido nelle operazioni di confronto.

2 Netflix ChatBot

Il chatbot oggetto di questo progetto ha l'obiettivo di assistere gli utenti nella ricerca di film e serie TV su Netflix, fornendo informazioni dettagliate e consigli personalizzati. Il chatbot è stato sviluppato per interagire e rispondere in italiano, rendendo l'esperienza utente semplice e intuitiva.



Figura 2: Logo del Chatbot

Si è focalizzati su un set di possibili informazioni che l'utente potrebbe voler conoscere, basandosi sui dati disponibili nel dataset utilizzato. Con il dataset a disposizione, sono state individuate tutte le informazioni di maggiore interesse per gli utenti. Data la vastità delle informazioni, queste sono state raggruppate in più funzionalità per evitare conversazioni eccessivamente lunghe. Le informazioni considerate riguardano principalmente le preferenze del consumatore, come ad esempio il genere, il regista, gli attori, l'anno di uscita e la durata dei contenuti.

Le possibili azioni che gli utenti possono eseguire tramite il chatbot sono le seguenti:

- Ricerca per attore,
- Ricerca per regista,
- Ricerca per genere,
- Ricerca per anno,
- Ricerca per categoria,
- Fornire informazioni dettagliate in base al titolo,
- Ricerca per durata,
- Ricerca per tipo di contenuto (film o serie TV),
- Ricerca per paese di produzione,
- Ottenere consigli di visione personalizzati in base alle preferenze,
- Ottenere l'elenco dei generi disponibili,
- Ottenere l'elenco dei paesi di produzione disponibili,
- Ottenere l'elenco delle categorie disponibili.

Le funzioni descritte richiedono l'accesso a un dataset che consente al chatbot di trovare le informazioni richieste dall'utente e di fornire suggerimenti pertinenti. Il dataset è un file .csv tabulare con colonne specifiche, come cast, registi, generi, paesi di produzione e durata.

2.1 Il Dataset

Per il progetto, si è utilizzato un dataset disponibile su Kaggle all'indirizzo <https://www.kaggle.com/datasets/shivamb/netflix-shows>. Il dataset è in formato CSV e contiene informazioni dettagliate su film e serie TV disponibili su Netflix. I dati includono informazioni come l'ID dello show, il tipo di contenuto (film o serie TV), il titolo, il regista, il cast, il paese di produzione, la data di aggiunta a Netflix, l'anno di uscita, la valutazione, la durata, i generi e una descrizione del contenuto.

Colonna	Descrizione
show_id	Identificativo unico dello show.
type	Tipo di contenuto (Film o Serie TV).
title	Titolo dello show.
director	Regista dello show.
cast	Cast dello show.
country	Paese di produzione dello show.
date_added	Data di aggiunta dello show a Netflix.
release_year	Anno di uscita dello show.
rating	Valutazione dello show.
duration	Durata dello show.
listed_in	Generi associati allo show.
description	Descrizione dello show.

Successivamente, sono state apportate delle modifiche e pulizie al dataset originale per facilitare la ricerca dei contenuti all'utente e l'elaborazione dei dati. Il dataset finale è strutturato come segue:

Colonna	Descrizione
type	Tipo di contenuto (Film o Serie TV).
title	Titolo dello show.
director	Regista dello show.
cast	Cast dello show.
country	Paese di produzione dello show.
date_added	Data di aggiunta dello show a Netflix in formato YYYY-MM-DD.
release_year	Anno di uscita dello show.
description	Descrizione dello show.
duration_number	Durata numerica dello show.
duration_unit	Unità di misura della durata (minuti o stagioni).
rating_category	Categoria di valutazione dello show.
genres_list	Elenco dei generi associati allo show in formato lista.

2.2 La struttura

Dopo aver creato un ambiente virtuale tramite Conda e una directory per lo sviluppo del progetto, è possibile inizializzare il progetto RASA utilizzando il comando `rasa init`.

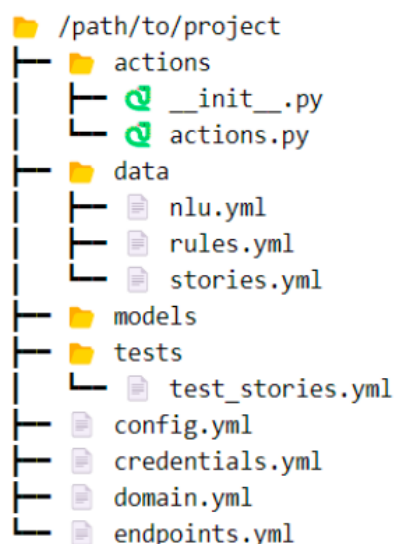


Figura 3: Struttura di un progetto RASA

Questo comando genera la struttura di base necessaria per lo sviluppo di un chatbot. Le componenti principali della struttura di un progetto RASA sono:

- **actions.py**: Uno script Python che definisce le classi corrispondenti alle azioni menzionate nel file **domain.yml**. Queste azioni permettono di generare risposte basate su calcoli o elaborazioni, anziché risposte predefinite.
- **nlu.yml**: Un file che contiene gli esempi utilizzati per l'addestramento degli intenti e delle entità.
- **rules.yml**: Un file che contiene regole predefinite per le politiche di dialogo.
- **stories.yml**: Un file che contiene esempi di conversazioni, denominate percorsi, che il chatbot seguirà.
- **config.yml**: Un file che contiene la configurazione del modello di machine learning.
- **domain.yml**: Un file che contiene tutte le informazioni necessarie al chatbot per operare. Questo file include una lista di tutti gli intenti, le entità, gli slot per memorizzare i dati, le utterances con le risposte predefinite, la lista di tutte le azioni e le form.

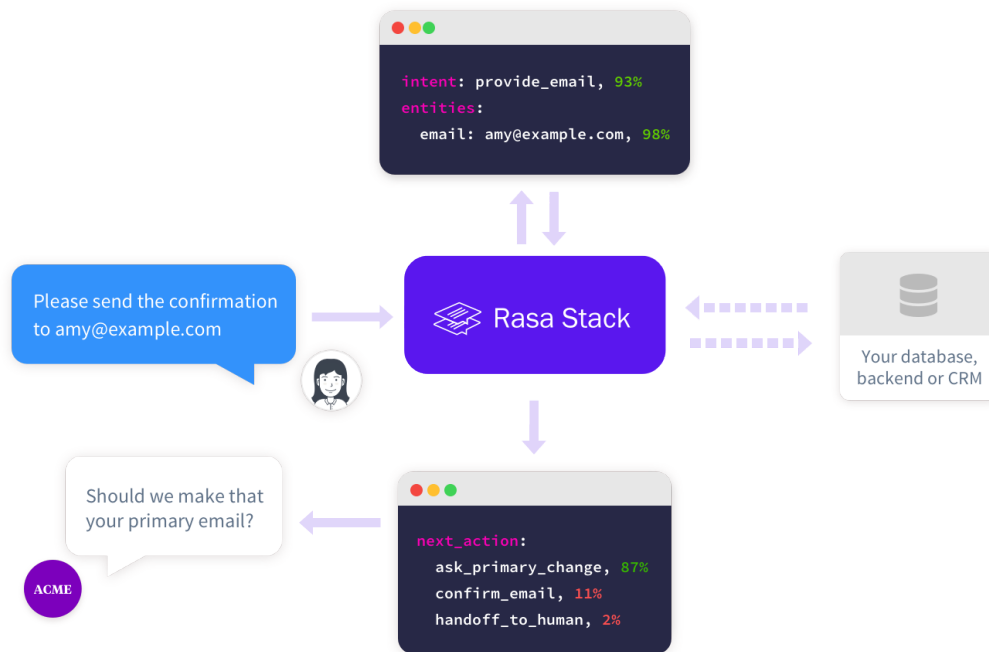


Figura 4: Workflow di RASA

3 Funzionalità del Chatbot

In questo capitolo descriveremo le diverse funzionalità del chatbot, spiegando le scelte e le strategie adottate per svilupparle.

3.1 Visualizzare tutti i generi dei contenuti presenti

La funzione del chatbot consente di visualizzare tutti i generi dei contenuti disponibili, come Crime TV Shows, Horror Movies, e altri ancora. Per realizzare questa funzionalità, è stato necessario fornire al modello di machine learning esempi di frasi che gli utenti potrebbero scrivere per chiedere i generi dei vari contenuti, e successivamente il modello è stato addestrato su tali dati.

```
- intent: generi_disponibili
examples: |
- che generi conosci?
- quali generi di film conosci?
- quali generi di serie tv conosci?
- dammi la lista dei generi
- che generi posso scegliere?
- che generi posso scrivere?
- quali generi posso selezionare?
- quali generi posso indicare?
- quali generi posso specificare?
- quali generi posso menzionare?
- quali sono i generi disponibili?
- quali generi posso chiedere?
- quali sono i generi di film?
- quali sono i generi di serie tv?
- dammi i generi disponibili
- generi?
- generi
```

Figura 5: Intent per ottenere i generi disponibili

Per implementare questa funzionalità, è stata creata una classe di azione che, tramite una connessione al dataset, esegue una query per estrarre distintamente tutti i generi presenti ed associati a ciascun contenuto. Il codice implementato è riportato di seguito:

```
1 class GetGeneriFromData(Action):
2
3     def name(self) -> Text:
4         return "action_get_generi_from_data"
5
6     def run(self, dispatcher: CollectingDispatcher, tracker: Tracker, domain: Dict[Text, Any])
7         -> List[Dict[Text, Any]]:
8         # Carica il dataset
9         df = pd.read_csv(PATH_TO_CSV)
10
11         # Estraiamo i generi, convertiamo da stringa a lista Python, e poi concateniamo tutto
12         in un'unica lista
13         all_genres = []
14         for genres_string in df['genres_list']:
15             genres_list = ast.literal_eval(genres_string) # Converte la stringa in lista
16             all_genres.extend(genres_list) # Aggiunge gli elementi della lista alla lista
17         all_genres
18
19         # Otteniamo generi unici e li ordiniamo
20         unique_genres = sorted(set(all_genres))
21
22         # Creiamo la stringa per l'output, mettendo ogni genere su una nuova linea
23         genres_message = "- " + "\n- ".join(unique_genres)
24
25         # Creo il messaggio da inviare all'utente
26         messaggio = "Ecco alcuni generi che potresti cercare:\n\n" + genres_message
27
28         # Invio il messaggio
```

```

26     dispatcher.utter_message(text=messaggio)
27
28     return []

```

Questa classe di azione permette di ottenere una lista di tutti i generi presenti nel dataset e di mostrarli all'utente in modo chiaro e ordinato.

```

Your input -> generi, per favore
Ecco alcuni generi che potresti cercare:
- Action & Adventure
- Anime Features
- Anime Series
- British TV Shows
- Children & Family Movies
- Classic & Cult TV
- Classic Movies
- Comedies
- Crime TV Shows
- Cult Movies
- Documentaries
- Docuseries
- Dramas
- Faith & Spirituality
- Horror Movies
- Independent Movies
- International Movies
- International TV Shows
- Kids' TV
- Korean TV Shows
- LGBTQ Movies
- Movies
- Music & Musicals
- Reality TV
- Romantic Movies
- Romantic TV Shows
- Sci-Fi & Fantasy
- Science & Nature TV
- Spanish-Language TV Shows
- Sports Movies
- Stand-Up Comedy
- Stand-Up Comedy & Talk Shows
- TV Action & Adventure
- TV Comedies
- TV Dramas
- TV Horror
- TV Mysteries
- TV Sci-Fi & Fantasy
- TV Shows
- TV Thrillers
- Teen TV Shows
- Thrillers

```

Figura 6: Lista dei generi disponibili

3.2 Visualizzare tutti i paesi dei contenuti presenti

La funzione del chatbot consente di visualizzare tutti i paesi dei contenuti disponibili, come Italy, France, e altri ancora. Per realizzare questa funzionalità, è stato necessario fornire al modello di machine learning esempi di frasi che gli utenti potrebbero scrivere per chiedere i paesi dei vari contenuti, e successivamente il modello è stato addestrato su tali dati.

```

- intent: paesi_disponibili
examples: |
    - che paesi conosci?
    - quali paesi di film conosci?
    - quali paesi di serie tv conosci?
    - quali sono i paesi disponibili?
    - quali paesi posso chiedere?
    - quali paesi posso selezionare?
    - quali paesi posso indicare?
    - quali paesi posso specificare?
    - quali paesi posso menzionare?
    - quali paesi posso scrivere?
    - paesi?
    - paesi
    - dammi la lista dei paesi
    - che paesi posso scegliere?
    - che paesi posso scrivere?
    - quali paesi ci sono?
    - che paesi hai disponibili?
    - paesi disponibili

```

Figura 7: Intent per ottenere i paesi disponibili

Per implementare questa funzionalità, è stata creata una classe di azione che, tramite una connessione al dataset, esegue una query per estrarre distintamente tutti i paesi presenti ed associati a ciascun contenuto. Il codice implementato è riportato di seguito:

```

1 class GetPaesiFromData(Action):
2
3     def name(self) -> Text:
4         return "action_get_paesi_from_data"
5
6     def run(self, dispatcher: CollectingDispatcher, tracker: Tracker, domain: Dict[Text, Any])
7         -> List[Dict[Text, Any]]:
8         # Carica il dataset
9         df = pd.read_csv(PATH_TO_CSV)
10
11         # Splitting di ogni riga sulla virgola e poi unire tutto in un'unica lista
12         all_countries = []
13         for country_string in df['country'].dropna(): # Usiamo dropna per escludere valori
14             NaN
15             countries_list = [country.strip() for country in country_string.split(',')]
16             all_countries.extend(countries_list)
17
18         # Otteniamo paesi unici e li ordiniamo
19         unique_countries = sorted(set(all_countries))
20
21         # Creiamo la stringa per l'output, mettendo ogni paese su una nuova linea
22         countries_message = "- " + "\n- ".join(unique_countries)
23
24         # Creo il messaggio da inviare all'utente
25         messaggio = "Ecco alcuni paesi dai quali abbiamo contenuti:\n\n" + countries_message
26
27         # Invio il messaggio
28         dispatcher.utter_message(text=messaggio)
29
30         return []

```

Questa classe di azione permette di ottenere una lista di tutti i paesi presenti nel dataset e di mostrarli all'utente in modo chiaro e ordinato.

3.3 Visualizzare tutte le categorie di visione dei contenuti presenti

La funzione del chatbot consente di visualizzare tutte le categorie di visione dei contenuti disponibili, come Teens, Adults Only, e altri ancora. Per realizzare questa funzionalità, è stato necessario fornire al modello di

```

Your input -> paesi, per favore
Ecco alcuni paesi dai quali abbiamo contenuti:
-
- Afghanistan
- Albania
- Algeria
- Angola
- Argentina
- Armenia
- Australia
- Austria
- Azerbaijan
- Bahamas
- Bangladesh
- Belarus
- Belgium
- Bermuda
- Botswana
- Brazil
- Bulgaria
- Burkina Faso
- Cambodia
- Cameroon

```

Figura 8: Lista dei paesi disponibili

machine learning esempi di frasi che gli utenti potrebbero scrivere per chiedere i paesi dei vari contenuti, e successivamente il modello è stato addestrato su tali dati.

```

- intent: categorie_disponibili
examples: |
  - che categorie conosci?
  - quali categorie conosci?
  - che categorie hai?
  - che vuol dire categorie?
  - quali categorie posso chiedere?
  - quali categorie posso selezionare?
  - quali categorie posso indicare?
  - quali categorie posso specificare?
  - quali categorie posso menzionare?
  - quali categorie posso scrivere?
  - categorie?

```

Figura 9: Intent per ottenere le categorie di visione disponibili

Per implementare questa funzionalità, è stata creata una classe di azione che, tramite una connessione ad una mappa chiave-valore, stampa distintamente tutte le categorie di visione. Il codice implementato è riportato di seguito:

```

1 class ActionGetCategorieFromMap(Action):
2
3     def name(self) -> Text:
4         return "action_get_categorie_from_map"
5
6     def run(self, dispatcher: CollectingDispatcher, tracker: Tracker, domain: Dict[Text,
7 Any]) -> List[Dict[Text, Any]]:
8         # Estraiamo le categorie dalla mappatura delle categorie
9         categorie = list(mappatura_categorie.keys())
10
11         # Creiamo la stringa per l'output, mettendo ogni categoria su una nuova linea
12         categorie_message = "- " + "\n- ".join(categorie)
13
14         # Creo il messaggio da inviare all'utente
15         messaggio = "Ecco alcune categorie che potresti cercare:\n\n" + categorie_message
16
17         # Invio il messaggio
18         dispatcher.utter_message(text=messaggio)

```

```

18
19         return []

```

Questa classe di azione permette di ottenere una lista di tutte le categorie presenti nel dataset e di mostrarle all'utente in modo chiaro e ordinato.

```

Your input -> categorie, per favore
Ecco alcune categorie che potresti cercare:
- Adults Only - Not suitable for children under 17 years
- General - Suitable for all audiences
- Kids - Suitable for all children
- Mature - Suitable for adults only
- Non Specificato
- Not Rated - No specific classification
- Older Kids - Suitable for children over 7 years
- Older Kids - Suitable for children over 7 years with fantasy violence elements
- Parental Guidance - Parents urged to give parental guidance under 12 years
- Parental Guidance Suggested - Suitable for children with parental guidance
- Restricted - Suitable for adults (contains adult material)
- Teens - Suitable for over 13 years
- Teens - Suitable for over 14 years
- Unrated - Uncensored or extended version not submitted for classification

```

Figura 10: Lista delle categorie di visione disponibili

3.4 Visualizzare tutte le informazioni del contenuto tramite il titolo

La funzione del chatbot consente di visualizzare tutte le informazioni del contenuto passato tramite titolo, come Generi, Cast, Regia e altri ancora. Per realizzare questa funzionalità, è stato necessario fornire al modello di machine learning esempi di frasi che gli utenti potrebbero scrivere per chiedere i dettagli dei vari contenuti, e successivamente il modello è stato addestrato su tali dati. In questo caso abbiamo utilizzato la entity 'titolo' per addestrare il Chatbot affinché capisse l'oggetto di riferimento all'interno del contesto passato.

```

- intent: informazioni_contenuto
examples: |
  - Informazioni su [The Social Network](titolo)
  - Dettagli su [The Social Network](titolo)
  - Info su [The Crown](titolo)
  - Descrizione su [The Breakfast Club](titolo)
  - Informazioni su [The Queen's Gambit](titolo)
  - Dettagli su [The Queen's Gambit](titolo)
  - Info su [The Queen's Gambit](titolo)
  - Informazioni su [The Breaking Bad](titolo)
  - Dettagli su [The Breaking Bad](titolo)
  - Info su [Strange Things](titolo)
  - Descrizione su [Strange Things](titolo)
  - Informazioni su
  - Dettagli su
  - Info su
  - Descrizione su
  - Info su [Joker](titolo)

```

Figura 11: Intent per ottenere le informazioni dei contenuti

Per implementare questa funzionalità, è stata creata una classe di azione che, tramite una connessione al dataset, stampa tutte le informazioni relative al titolo passato dall'utente. Il codice implementato è riportato di seguito:

```

1 # Azione per ottenere informazioni sul contenuto passato
2 class ActionGetInfoFromTitle(Action):
3
4     def name(self) -> Text:
5         return "action_informazioni_contenuto"
6
7     def run(self, dispatcher: CollectingDispatcher, tracker: Tracker, domain: Dict[Text, Any])
8         -> List[Dict[Text, Any]]:
9         df = pd.read_csv(PATH_TO_CSV)
10
11         text = tracker.latest_message.get('text') # Ottiene il testo completo del messaggio
12         # Isola il titolo dalla richiesta dell'utente
13         isolated_title = isolate_title_from_request(text)
14
15         # Usa fuzzy matching per trovare il titolo più simile nel dataset
16         titles = df['title'].tolist()

```

```

16     best_match, score = process.extractOne(isolated_title, titles)
17
18     if score > 90: # Soglia di accettazione per il fuzzy matching
19         result = df[df['title'].str.lower() == best_match.lower()]
20
21         if not result.empty:
22             tipo = result.iloc[0]['type']
23             descrizione = result.iloc[0]['description']
24             regista = "Non specificato" if pd.isnull(result.iloc[0]['director']) else
result.iloc[0]['director']
25             cast = "Non specificato" if pd.isnull(result.iloc[0]['cast']) else result.iloc
[0]['cast']
26             categoria = result.iloc[0]['rating_category']
27             generi = ast.literal_eval(result.iloc[0]['genres_list']) if pd.notna(result.
iloc[0]['genres_list']) else ["Non specificato"]
28             titolo = result.iloc[0]['title']
29
30             messaggio = f"- {titolo} e' un {'Film' if tipo == 'Movie' else 'Serie TV'} con
le seguenti informazioni:\n\n"
31             messaggio += f"    Descrizione: {descrizione}\n"
32             messaggio += f"    Regia: {regista}\n"
33             messaggio += f"    Cast: {cast}\n"
34             messaggio += f"    Categoria di pubblico: {categoria}\n"
35             messaggio += f"    Genere/i: {'', '.join(generi)}"
36
37             dispatcher.utter_message(text=messaggio)
38         else:
39             dispatcher.utter_message(text=f"Non ho trovato informazioni per il titolo '{
titolo}').")
40         else:
41             dispatcher.utter_message(text="Non ho potuto identificare il titolo nella tua
richiesta.")
42
43     return []

```

Listing 1: Action per ottenere le informazioni dei contenuti

Questa classe di azione permette di ottenere un messaggio contenente tutte le informazioni relative al titolo del contenuto e di mostrarle all'utente in modo chiaro e ordinato.

```

Your input -> informazioni su Kung Fu Panda
- Kung Fu Panda è un Film con le seguenti informazioni:
  Descrizione: When a powerful villain comes after peace in his valley, a lazy panda rises to the challeng
e to realize his destiny and become a kung fu warrior.
  Regia: John Stevenson, Mark Osborne
  Cast: Jack Black, Dustin Hoffman, Angelina Jolie, Ian McShane, Seth Rogen, Lucy Liu, Jackie Chan, David
Cross, Randall Duk Kim, James Hong
  Categoria di pubblico: Parental Guidance Suggested - Suitable for children with parental guidance
  Genere/i: Children & Family Movies, Comedies

```

Figura 12: Messaggio con tutte le informazioni del titolo

Uso delle Funzioni di Gestione dei Titoli

Le funzioni `merge_movie_titles` e `isolate_title_from_request` sono state progettate per facilitare l'estrazione e la manipolazione dei titoli dei contenuti all'interno delle richieste degli utenti.

1. Funzione `merge_movie_titles`: Questa funzione è utilizzata per unire più entità riconosciute come parti di un titolo in una singola stringa. Questo è particolarmente utile quando l'utente fornisce un titolo lungo o composto che viene suddiviso in più entità dal sistema di riconoscimento delle entità. La funzione segue i seguenti passaggi:
 - Filtra solo le entità di tipo 'titolo'.
 - Ordina le entità per la loro posizione di inizio nel testo.
 - Unisce i valori delle entità per formare il titolo completo.
 - Utilizza espressioni regolari per verificare che il titolo unificato corrisponda a un pattern che esclude comandi o altre parti non rilevanti.

2. Funzione **isolate_title_from_request**: Questa funzione è utilizzata per isolare il titolo del contenuto dalla richiesta dell'utente. Rimuove le frasi introduttive comuni che non fanno parte del titolo stesso, come "info su" o "informazioni su". La funzione segue i seguenti passaggi:

- Definisce un pattern che include le frasi introduttive comuni.
- Utilizza un'espressione regolare per cercare il pattern nella richiesta dell'utente.
- Restituisce il titolo isolato se il pattern viene trovato, altrimenti restituisce il testo originale.

Queste funzioni sono essenziali per garantire che il chatbot possa identificare e gestire correttamente i titoli dei contenuti nelle interazioni con l'utente.

3.5 Action di ricerca contenuti filtrati a seconda delle preferenze dell'utente

L'azione **ActionRicercaContenuti** è progettata per gestire le richieste degli utenti relative alla ricerca di film o serie TV basata su vari criteri, come genere, attore, regista, anno di uscita, paese di produzione e durata. Questa azione elabora le entità estratte dal messaggio dell'utente per costruire una query di ricerca e fornire risultati pertinenti.

Le entità coinvolte in questa azione sono:

- **tipo**: Indica se l'utente sta cercando un film o una serie TV.
- **persona**: Può rappresentare un attore o un regista, a seconda del contesto, infatti è un entity composta, si suddivide in due ruoli: attore e regista.
- **genere**: Il genere del contenuto, come commedia, horror o thriller.
- **anno**: L'anno di uscita del contenuto.
- **paese**: Il paese di produzione del contenuto.
- **categoria**: La categoria di visione, come adatto ai bambini o per adulti.
- **durata**: La durata del contenuto, in minuti o stagioni.

```
- intent: ricerca_contenuti
examples: |
- Cerca un [film](tipo) con [Tom Hanks]({entity: "persona", "role": "attore"})
- Dammi i [film](tipo) con [Tom Hanks]({entity: "persona", "role": "attore"})
- Quali [film](tipo) con [Tom Hanks]({entity: "persona", "role": "attore"}) hai?
- Dammi [serie](tipo) [horror](genere) dirette da [Steven Spielberg]({entity: "persona", "role": "regista"})
- Quali [film](tipo) [commedia](genere) hai?
- Cerca [film](tipo) [commedia](genere) del [2020](anno)
- Dammi [serie tv](tipo) del genere [drammatico](genere)
- Dammi [serie tv](tipo) diretti da [Steven Spielberg]({entity: "persona", "role": "regista"})
- [film](tipo) [commedia](genere) del [1990](anno) diretto da [Woody Allen]({entity: "persona", "role": "regista"})
- [serie tv](tipo) [drammatico](genere) dalla durata di [120 minuti](durata)
- [horror](genere) del [2012](anno) con [Tom Hanks]({entity: "persona", "role": "attore"})
- Vorrei vedere una [serie](tipo) [thriller](genere) recente
- Cerca [film](tipo) [d'azione](genere) del [2019](anno)
- Quali [serie](tipo) [italiane](paese) posso guardare?
- Dammi i [film](tipo) di [Quentin Tarantino]({entity: "persona", "role": "regista"})
- Quali [serie](tipo) con [Meryl Streep]({entity: "persona", "role": "attore"}) sono disponibili?
- Esplora [film](tipo) [commedia](genere) americani del [2000](anno)
- Cerca [film](tipo) della [categoria](categoria) [per adulti](categoria)
- Quali [serie tv](tipo) [fantascienza](genere) hai del [2018](anno)?
- Dammi [film](tipo) con una durata inferiore ai [90 minuti](durata)
- Trova [serie](tipo) [storiche](genere) prodotte in [Germania](paese)
- [film](tipo) [drammatici](genere) con [Leonardo DiCaprio]({entity: "persona", "role": "attore"})
- Ho bisogno di [serie](tipo) [comiche](genere) da guardare con la famiglia
- Elenco di [film](tipo) [biografici](genere) sul [titolo](titolo) [The Social Network](titolo)
- Dammi [serie](tipo) ambientate negli anni [80](anno)
- Cerca [film](tipo) sul tema della [guerra](categoria) del [2021](anno)
- Mostrami [serie](tipo) con episodi di [30 minuti](durata)
- Trova [film](tipo) [horror](genere) da [Spagna](paese)
```

Figura 13: Intent per la ricerca dei contenuti, con diverse entità per addestrare il chatbot a riconoscere il contesto

L'azione segue questi passaggi per elaborare la richiesta:

1. **Estrazione delle Entità**: Le entità riconosciute vengono estratte dal messaggio dell'utente.

2. **Inferenza dei Ruoli:** Per le entità di tipo persona, viene inferito il ruolo (attore o regista) in base al contesto della frase.
3. **Costruzione della Query:** Viene costruita una query di ricerca basata sulle entità estratte e sui ruoli inferiti.
4. **Ricerca nel Dataset:** La query viene utilizzata per filtrare il dataset e trovare i contenuti corrispondenti ai criteri specificati dall'utente.
5. **Formattazione dei Risultati:** I risultati della ricerca vengono formattati e inviati all'utente come risposta.

```

1 class ActionRicercaContenuti(Action):
2     def name(self) -> Text:
3         return "action_ricerca_contenuti"
4
5     def run(self, dispatcher: CollectingDispatcher,
6             tracker: Tracker,
7             domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:
8
9         df = pd.read_csv(PATH_TO_CSV, converters={"genres_list": ast.literal_eval})
10        entita_riconosciute = tracker.latest_message['entities']
11
12        query = {} # Dizionario per costruire la query di ricerca
13        tipo_specified = None
14        # Usa la funzione migliorata per inferire i ruoli di tutte le entita' 'persona'
15        roles = infer_roles_for_all_entities(tracker.latest_message['text'],
16        entita_riconosciute)
17
18        for entita in entita_riconosciute:
19            if entita['entity'] == 'tipo':
20                tipo_valori = ['film', 'movie', 'titolo', 'pellicola', 'serie', 'tv show', 'tv
21                serie', 'telefilm', 'programma tv', 'serie tv', 'show', 'programma']
22                matched_type = process.extractOne(entita['value'].lower(), tipo_valori, scorer
23                =fuzz.token_set_ratio)
24                if matched_type[1] > 90:
25                    tipo_specified = 'Movie' if matched_type[0] in ['film', 'movie', 'titolo',
26                    'pellicola'] else 'TV Show'
27                    query['type'] = tipo_specified
28
29            elif entita['entity'] == 'persona':
30                # Utilizziamo il ruolo inferito dalla funzione migliorata
31                role = roles.get(entita['value'], None)
32                query_value = entita['value'].lower()
33                columns_to_search = []
34                if role == 'attore':
35                    columns_to_search = ['cast']
36                    print("attore")
37                elif role == 'regista':
38                    columns_to_search = ['director']
39                    print("regista")
40                else:
41                    columns_to_search = ['cast', 'director']
42                final_match = None
43                highest_score = 0
44                final_column = None
45                for column in columns_to_search:
46                    search_list = df[column].str.split(', ').explode().unique()
47                    match = process.extractOne(query_value, search_list, score_cutoff=95)
48                    if match and match[1] > highest_score:
49                        highest_score = match[1]
50                        final_match = match[0]
51                        final_column = column
52                if final_match and final_column:
53                    df = df[df[final_column].str.contains(final_match, case=False, na=False)]
54                    query[final_column] = final_match
55                else:
56                    dispatcher.utter_message(text="Non ho trovato risultati corrispondenti
57                    alla tua ricerca.")
58                    return []
59
60            elif entita['entity'] == 'genere':
61                matched_genres = [] # Lista per tenere traccia dei generi corrispondenti
62                # Verifica se il tipo e' specificato e se e' 'Movie'
63                if tipo_specified == 'Movie':

```

```

59         matched_genre = trova_corrispondenza(entita['value'],
mappatura_generi_film)
60         if matched_genre:
61             matched_genres.append(matched_genre)
62             # Verifica se il tipo e' specificato e se e' 'TV Show'
63             elif tipo_specified == 'TV Show':
64                 matched_genre = trova_corrispondenza(entita['value'],
mappatura_generi_serie_tv)
65                 if matched_genre:
66                     matched_genres.append(matched_genre)
67                     # Se il tipo non e' specificato, cerca in entrambi (film e serie TV)
68                 else:
69                     matched_genre_film = trova_corrispondenza(entita['value'],
mappatura_generi_film)
70                     if matched_genre_film:
71                         matched_genres.append(matched_genre_film)
72
73                     matched_genre_serie = trova_corrispondenza(entita['value'],
mappatura_generi_serie_tv)
74                     if matched_genre_serie:
75                         matched_genres.append(matched_genre_serie)
76                     if matched_genres:
77                         query['genres_list'] = matched_genres
78
79
80             elif entita['entity'] == 'paese':
81                 matched_country = trova_corrispondenza(entita['value'], mappatura_paesi)
82                 if matched_country:
83                     query['country'] = matched_country
84
85
86             elif entita['entity'] == 'categoria':
87                 matched_category = trova_corrispondenza(entita['value'], mappatura_categorie)
88                 if matched_category:
89                     query['rating_category'] = matched_category
90
91
92             elif entita['entity'] == 'anno':
93                 query['release_year'] = int(entita['value'])
94
95
96             elif entita['entity'] == 'durata':
97                 # Estrazione del numero e dell'unita' dalla durata
98                 durata_match = re.search(r'(\d+)\s*(minuti|stagioni|stagione|min)', entita['
value'], re.IGNORECASE)
99                 if durata_match:
100                     numero, unita = durata_match.groups()
101                     if 'min' in unita.lower() or 'minuti' in unita.lower():
102                         query['duration_number'] = int(numero)
103                         query['duration_unit'] = 'min'
104                     else:
105                         query['duration_number'] = int(numero)
106                         if int(numero) == 1:
107                             query['duration_unit'] = 'season'
108                         else:
109                             query['duration_unit'] = 'seasons'
110
111             # Applica i filtri basati sulla query costruita dalle entita'
112             for key, value in query.items():
113                 if key == 'genres_list':
114                     # Crea una condizione per il filtro basata sui generi
115                     genre_condition = df['genres_list'].apply(lambda genres: any(genre in query['
genres_list'] for genre in genres))
116                     # Se il tipo e' specificato, usa anche il filtro per tipo
117                     if 'type' in query:
118                         df = df[(df['type'] == query['type']) & genre_condition]
119                     else:
120                         # Se il tipo non e' specificato, ignora il filtro per tipo e utilizza solo
quello per genere
121                         df = df[genre_condition]
122
123                 elif key == 'country':
124                     df = df[df['country'].str.contains(value, case=False, na=False)]
125
126                 elif key == 'rating_category':

```

```

127         df = df[df['rating_category'] == value]
128
129     elif key == 'type':
130         df = df[df['type'] == value]
131
132     elif key == 'release_year':
133         df = df[df['release_year'] == int(value)]
134
135     elif key == 'duration_number':
136         # Gestione della durata per i film
137         if 'duration_unit' in query and query['duration_unit'] == 'min':
138             df = df[(df['type'] == 'Movie') & (df['duration_number'] <= int(value))]
139         # Gestione delle stagioni per le serie TV
140         elif 'duration_unit' in query and query['duration_unit'] != 'min':
141             df = df[(df['type'] == 'TV Show') & (df['duration_number'] <= int(value))]
142
143     # Invio dei risultati
144     if df.empty:
145         dispatcher.utter_message(text="Non ho trovato risultati corrispondenti alla tua
ricerca.")
146     else:
147         shuffled_results = df.sample(frac=1).reset_index(drop=True) # Mescola i risultati
148         risultati = shuffled_results.head(20) # 20 titoli massimi
149         intro = "Ecco i risultati trovati" # Messaggio di default
150
151         if 'type' in query:
152             if query['type'] == 'Movie':
153                 intro = "Ecco i film trovati"
154             elif query['type'] == 'TV Show':
155                 intro = "Ecco le serie trovate"
156
157         if 'cast' in query:
158             intro += f" con {query['cast'].title()}"
159
160         if 'director' in query:
161             intro += f" diretti da {query['director'].title()}"
162
163     # Costruzione del messaggio di risposta con l'intro modificato
164     filtri_applicati = []
165     for k, v in query.items():
166         if k == 'type':
167             continue
168         elif k == 'genres_list':
169             filtri_applicati.append(f"# Genere/i: {'', '.join([g.title() for g in v])}")
170
171         elif k == 'duration_number':
172             if query['duration_unit'] == 'min':
173                 filtri_applicati.append(f"# Durata massima: {v} min")
174             else:
175                 if v == 1:
176                     filtri_applicati.append(f"# Durata massima: {v} stagione")
177                 else:
178                     filtri_applicati.append(f"# Durata massima: {v} stagioni")
179         elif k == 'release_year':
180             filtri_applicati.append(f"# Anno di uscita: {v}")
181         elif k == 'rating_category':
182             filtri_applicati.append(f"# Categoria di visione: {v}")
183         elif k in ['attore', 'regista']:
184             filtri_applicati.append(f"{k}: {v.title()}")
185         else:
186             filtri_applicati.append(f"{k}: {str(v).title()}")
187
188     if filtri_applicati:
189         intro += f" con i seguenti filtri:\n{'', '.join(filtri_applicati)}.\n"
190
191     risposta = f"{intro}\n\n"
192     for index, row in risultati.iterrows():
193         titolo = row['title'].title()
194         tipo = row['type']
195         descrizione = row['description']
196         regista = row['director']
197         cast = row['cast']
198         categoria = row['rating_category']
199         if tipo == 'Movie':
200             durata = f"{row.get('duration_number', 'N/D')} min" if 'duration_number'

```

```

200         in row else 'N/D'
201         else:
202             durata = f"{row.get('duration_number', 'N/D')} stagione/i" if '
duration_number' in row else 'N/D'
203             risposta += f"\n- {titolo} ({'Film' if tipo == 'Movie' else 'Serie TV'},
Durata massima: {durata}):\n  Descrizione: {descrizione}\n  Regia: {regista}\n  Cast: {
cast}\n  Categoria di pubblico: {categoria}\n  Genere/i: {'', '.join(row['genres_list'])
}\n\n"
204
205         dispatcher.utter_message(text=risposta)
206
207     return []

```

Listing 2: Action per filtrare tutti i contenuti dati dall'utente

L'intent **ricerca_contenuti** è associato a questa azione e contiene esempi di frasi che l'utente potrebbe utilizzare per effettuare una ricerca. Gli esempi includono vari criteri di ricerca, come genere, attore, regista, anno di uscita, paese di produzione e durata. Questa azione e l'intent associato consentono al chatbot di rispondere in modo flessibile e preciso alle richieste di ricerca dei contenuti, fornendo all'utente informazioni utili sui film o sulle serie TV di interesse.

```

Your input -> film con Leonardo DiCaprio diretti da Martin Scorsese dalla duarata di 200 minuti
Ecco i film trovati con Leonardo Dicaprio diretti da Martin Scorsese con i seguenti filtri:
cast: Leonardo Dicaprio, director: Martin Scorsese, # Durata massima: 200 min, duration_unit: Min.

- Gangs Of New York (Film, Durata massima: 167.0 min):
  Descrizione: In the crime-ridden slums of New York in the 1860s, vengeful orphan Amsterdam Vallon joins
the cutthroat gang of his father's killer, Bill the Butcher.
  Regia: Martin Scorsese
  Cast: Leonardo DiCaprio, Daniel Day-Lewis, Cameron Diaz, Jim Broadbent, John C. Reilly, Henry Thomas, Li
am Neeson, Brendan Gleeson, Gary Lewis, Stephen Graham
  Categoria di pubblico: Restricted - Suitable for adults (contains adult material)
  Genere/i: Dramas

- Shutter Island (Film, Durata massima: 139.0 min):
  Descrizione: A U.S. marshal's troubling visions compromise his investigation into the disappearance of a
patient from a hospital for the criminally insane.
  Regia: Martin Scorsese
  Cast: Leonardo DiCaprio, Mark Ruffalo, Ben Kingsley, Max von Sydow, Michelle Williams, Emily Mortimer, P
atricia Clarkson, Jackie Earle Haley, Ted Levine, John Carroll Lynch, Elias Koteas
  Categoria di pubblico: Restricted - Suitable for adults (contains adult material)
  Genere/i: Thrillers

```

Figura 14: Output della ricerca dei film con Leonardo DiCaprio, girati da Martin Scorsese dalla durata massima di 200 minuti

Il Chabot gestisce le parole chiave "diretto da" per identificare in maniera più sicura i registi, e la parola chiave "con" per identificare gli attori. Utilizzando queste parole nel messaggio, il Chatbot riconosce meglio le entità e restituisce la query correttamente.

Funzioni per Gestione degli Attori e Registi e Trova Corrispondenze con Fuzzy

Le funzioni **trova_corrispondenza**, **infer_role_from_context**, **infer_roles_for_all_entities** e **merge_split_names** sono state progettate per migliorare la capacità del chatbot di interpretare e gestire le richieste degli utenti relative alla ricerca di contenuti come film e serie TV. Queste funzioni consentono al chatbot di comprendere il contesto delle richieste, identificare i ruoli delle persone menzionate (come attori o registi) e associare i termini utilizzati dall'utente ai termini corrispondenti nel dataset.

1. Funzione **trova_corrispondenza**: Questa funzione è utilizzata per trovare la corrispondenza esatta o approssimativa tra l'input dell'utente e le chiavi di una mappa di corrispondenza. Viene impiegata per associare sinonimi o varianti di termini usati dall'utente ai termini standard utilizzati nel dataset. Il fuzzy matching viene utilizzato come fallback per gestire casi in cui l'input non corrisponde direttamente a nessuna chiave o valore.
2. Funzione **infer_role_from_context**: Questa funzione determina il ruolo di una persona (attore o regista) in base al contesto in cui viene menzionata nella frase. Utilizza una "finestra" di testo che precede l'entità e cerca parole chiave associate a ciascun ruolo per fare inferenze.

3. Funzione **infer_roles_for_all_entities**: Questa funzione applica `infer_role_from_context` a tutte le entità di tipo 'persona' riconosciute nel testo, assegnando un ruolo a ciascuna in base al contesto. Questo aiuta il chatbot a comprendere se l'utente sta cercando un film o una serie TV con un certo attore o diretto da un certo regista.
4. Funzione **merge_split_names**: A volte, i riconoscitori di entità possono suddividere i nomi propri in più entità. Questa funzione unisce i nomi divisi per garantire che vengano trattati come un'entità unica.

Le mappe chiave-valore come **mappatura_categorie**, **mappatura_generi_film**, **mappatura_generi_serie_tv** e **mappatura_paesi** sono state create per gestire sinonimi e termini correlati, consentendo al chatbot di mappare i termini usati dall'utente ai termini standard usati nel dataset. Questo approccio aumenta la flessibilità del chatbot nel comprendere e rispondere a una gamma più ampia di richieste degli utenti.

Includendo queste funzioni e mappe nel nostro chatbot, abbiamo potuto migliorare significativamente la sua capacità di interpretare e gestire le richieste di ricerca dei contenuti in modo più accurato e contestualmente appropriato.

```

1 mappatura_paesi = {
2     "Italy": ["Italia", "italiano", "italiani"],
3     "United States": ["Stati Uniti", "USA", "americano", "americani"],
4     "France": ["Francia", "francese", "francesi"],
5     "Germany": ["Germania", "tedesco", "tedeschi"],
6     "United Kingdom": ["Regno Unito", "inglese", "inglesi", "britannico", "britannici"],
7     "Canada": ["Canada", "canadese", "canadesi"],
8     "Spain": ["Spagna", "spagnolo", "spagnoli"],
9     "Australia": ["Australia", "australiano", "australiani"],
10    "Japan": ["Giappone", "giapponese", "giapponesi"],
11    "China": ["Cina", "cinese", "cinesi"],
12    "Russia": ["Russia", "russo", "russi"],
13    "India": ["India", "indiano", "indiani"],
14    "Brazil": ["Brasile", "brasiliiano", "brasiliani"],
15    "Mexico": ["Messico", "messicano", "messicani"],
16    "Argentina": ["Argentina", "argentino", "argentini"],
17    "South Africa": ["Sudafrica", "sudafricano", "sudafricani"],
18    "South Korea": ["Corea del Sud", "coreano", "coreani", "sudcoreano", "sudcoreani"],
19    "Egypt": ["Egitto", "egiziano", "egiziani"],
20    "Turkey": ["Turchia", "turco", "turchi"],
21    "Netherlands": ["Paesi Bassi", "Olanda", "olandese", "olandesi"],
22    "Belgium": ["Belgio", "belga", "belgi"],
23    "Sweden": ["Svezia", "svedese", "svedesi"],
24    "Switzerland": ["Svizzera", "svizzero", "svizzeri"],
25    "Norway": ["Norvegia", "norvegese", "norvegesi"],
26    "Denmark": ["Danimarca", "danese", "danesi"],
27    "Finland": ["Finlandia", "finlandese", "finlandesi"],
28    "Poland": ["Polonia", "polacco", "polacchi"],
29    "Austria": ["Austria", "austriaco", "austriaci"],
30    "Ireland": ["Irlanda", "irlandese", "irlandesi"],
31    "Portugal": ["Portogallo", "portoghese", "portoghesi"],
32    "Greece": ["Grecia", "greco", "greci"],
33    "Hungary": ["Ungheria", "ungherese", "ungheresi"],
34    "Czech Republic": ["Repubblica Ceca", "ceco", "cechi"],
35    "Slovakia": ["Slovacchia", "slovacco", "slovacchi"],
36    "Slovenia": ["Slovenia", "sloveno", "sloveni"],
37    "Croatia": ["Croazia", "croato", "croati"],
38    "Serbia": ["Serbia", "serbo", "serbi"],
39    "Afghanistan": ["Afghanistan", "afghano", "afghani"],
40    "Albania": ["Albania", "albanese", "albanesi"],
41    "Algeria": ["Algeria", "algerino", "algerini"],
42    "Angola": ["Angola", "angolano", "angolani"],
43    "Armenia": ["Armenia", "armeno", "armeni"],
44    "Azerbaijan": ["Azerbaijan", "azero", "azeri"],
45    "Bahamas": ["Bahamas"],
46    "Bangladesh": ["Bangladesh", "bengalese", "bengalesi"],
47    "Belarus": ["Bielorussia", "bielorosso", "bielorussi"],
48    "Bermuda": ["Bermuda"],
49    "Botswana": ["Botswana"],
50    "Bulgaria": ["Bulgaria", "bulgaro", "bulgari"],
51    "Burkina Faso": ["Burkina Faso"],
52    "Cambodia": ["Cambogia", "cambogiano", "cambogiani"],
53    "Cameroon": ["Camerun", "camerunese", "camerunesi"],
54    "Cayman Islands": ["Isole Cayman"],
55    "Chile": ["Cile", "cileno", "cileni"],
56    "Colombia": ["Colombia", "colombiano", "colombiani"],

```

```

57 "Cuba": ["Cuba", "cubano", "cubani"],
58 "Cyprus": ["Cipro", "cipriota", "ciprioti"],
59 "Dominican Republic": ["Repubblica Dominicana", "dominicano", "dominicani"],
60 "East Germany": ["Germania Est", "tedesco orientale", "tedeschi orientali"],
61 "Ecuador": ["Ecuador", "ecuadoriano", "ecuatoriani"],
62 "Ethiopia": ["Etiopia", "etiopico", "etiopici"],
63 "Georgia": ["Georgia", "georgiano", "georgiani"],
64 "Ghana": ["Ghana", "ghanaiano", "ghanaiani"],
65 "Guatemala": ["Guatemala", "guatemalteco", "guatemaltechi"],
66 "Hong Kong": ["Hong Kong", "hongkonghese", "hongkonghesi"],
67 "Iceland": ["Islanda", "islandese", "islandesi"],
68 "Indonesia": ["Indonesia", "indonesiano", "indonesiani"],
69 "Iran": ["Iran", "iraniano", "iraniani"],
70 "Iraq": ["Iraq", "iracheno", "iracheni"],
71 "Jamaica": ["Giamaica", "giamaicano", "giamaicani"],
72 "Jordan": ["Giordania", "giordano", "giordani"],
73 "Kazakhstan": ["Kazakistan", "kazako", "kazaki"],
74 "Kenya": ["Kenya", "keniota", "kenioti"],
75 "Kuwait": ["Kuwait", "kuwaitiano", "kuwaitiani"],
76 "Latvia": ["Lettonia", "lettone", "lettoni"],
77 "Lebanon": ["Libano", "libanese", "libanesi"],
78 "Liechtenstein": ["Liechtenstein"],
79 "Lithuania": ["Lituania", "lituano", "lituani"],
80 "Luxembourg": ["Lussemburgo", "lussemburghese", "lussemburghesi"],
81 "Malawi": ["Malawi"],
82 "Malaysia": ["Malesia", "malese", "malesi"],
83 "Malta": ["Malta", "maltese", "maltesi"],
84 "Mauritius": ["Mauritius", "mauriziano", "mauriziani"],
85 "Mongolia": ["Mongolia", "mongolo", "mongoli"],
86 "Morocco": ["Marocco", "marocchino", "marocchini"],
87 "Mozambique": ["Mozambico", "mozambicano", "mozambicani"],
88 "Namibia": ["Namibia", "namibiano", "namibiani"],
89 "Nepal": ["Nepal", "nepalese", "nepalesi"],
90 "New Zealand": ["Nuova Zelanda", "neozelandese", "neozelandesi"],
91 "Nicaragua": ["Nicaragua", "nicaraguense", "nicaraguensi"],
92 "Nigeria": ["Nigeria", "nigeriano", "nigeriani"],
93 "Non disponibile": ["Non disponibile"],
94 "Pakistan": ["Pakistan", "pachistano", "pachistani"],
95 "Palestine": ["Palestina", "palestinese", "palestinesi"],
96 "Panama": ["Panama", "panamense", "panamensi"],
97 "Paraguay": ["Paraguay", "paraguaiano", "paraguayani"],
98 "Peru": ["Peru", "peruviano", "peruviani"],
99 "Philippines": ["Filippine", "filippino", "filippini"],
100 "Puerto Rico": ["Porto Rico", "portoricano", "portoricani"],
101 "Qatar": ["Qatar", "qatariota", "qatarioti"],
102 "Samoa": ["Samoa", "samoano", "samoani"],
103 "Saudi Arabia": ["Arabia Saudita", "saudita", "sauditi"],
104 "Senegal": ["Senegal", "senegalese", "senegalesi"],
105 "Singapore": ["Singapore", "singaporiano", "singaporiani"],
106 "Somalia": ["Somalia", "somalo", "somali"],
107 "Soviet Union": ["Unione Sovietica", "sovietico", "sovietici"],
108 "Sri Lanka": ["Sri Lanka", "singalese", "singalesi"],
109 "Sudan": ["Sudan", "sudaneese", "sudanesi"],
110 "Syria": ["Siria", "siriano", "siriani"],
111 "Taiwan": ["Taiwan", "taiwanese", "taiwanesi"],
112 "Uganda": ["Uganda", "ugandese", "ugandesi"],
113 "Ukraine": ["Ucraina", "ucraino", "ucraini"],
114 "United Arab Emirates": ["Emirati Arabi Uniti", "emiratino", "emiratini"],
115 "Uruguay": ["Uruguay", "uruguaiano", "uruguayani"],
116 "Vatican City": ["Citta del Vaticano", "vaticano"],
117 "Venezuela": ["Venezuela", "venezuelano", "venezuelani"],
118 "Vietnam": ["Vietnam", "vietnamita", "vietnamiti"],
119 "West Germany": ["Germania Ovest", "tedesco occidentale", "tedeschi occidentali"],
120 "Zimbabwe": ["Zimbabwe", "zimbabwese", "zimbabwesi"]
121 }
122
123 mappatura_categorie = {
124     'Adults Only - Not suitable for children under 17 years': ['solo maggiorenni', 'non adatto ai minori di 17 anni', 'maggiorenni', 'Adults Only'],
125     'General - Suitable for all audiences': ['generale', 'adatto a tutti', 'per tutti', 'anziani', 'per tutti i pubblici', 'tutti', 'general'],
126     'Kids - Suitable for all children': ['bambini', 'adatto ai bambini', 'per bambini', 'i piu piccoli', 'per bambini piccoli', 'Kids'],
127     'Mature - Suitable for adults only': ['maturi', 'adatto agli adulti', 'solo adulti', 'per adulti', 'Mature'],

```

```

128 'Non Specificato': ['non specificato'],
129 'Not Rated - No specific classification': ['non classificato', 'senza classificazione', '
Not Rated', 'Non valutato'],
130 'Older Kids - Suitable for children over 7 years': ['ragazzi piu grandi', 'adatto a
bambini oltre i 7 anni', 'Older Kids'],
131 'Older Kids - Suitable for children over 7 years with fantasy violence elements': ['
ragazzi piu grandi con elementi di violenza fantastica', 'bambini oltre i 7 anni con
violenza fantastica', 'Older Kids', 'fantasy violence'],
132 'Parental Guidance - Parents urged to give parental guidance under 12 years': ['guida
genitoriale', 'consigliato la guida dei genitori sotto i 12 anni', 'Parental Guidance', '
adatto con guida dei genitori'],
133 'Parental Guidance Suggested - Suitable for children with parental guidance': ['guida
genitoriale suggerita', 'adatto con guida dei genitori', 'Parental Guidance Suggested'],
134 'Restricted - Suitable for adults (contains adult material)': ['ristretto', 'adatto agli
adulti (contiene materiale per adulti)', 'Restricted'],
135 'Teens - Suitable for over 13 years': ['adolescenti', 'adatto a ragazzi oltre i 13 anni',
'13 Teens', 'teens'],
136 'Teens - Suitable for over 14 years': ['adolescenti oltre i 14 anni', 'adatto a ragazzi
oltre i 14 anni', '14 Teens', 'teens'],
137 'Unrated - Uncensored or extended version not submitted for classification': ['non
valutato', 'versione non censurata o estesa non sottoposta a classificazione', 'Unrated'],
138 }
139
140 mappatura_generi_film = {
141 'Classic Movies': ['classici', 'classico', 'classic', 'epoca', 'vintage', 'anni 50', 'anni
60', 'anni 70', 'anni 80', 'anni 90'],
142 'Horror Movies': ['horror', 'spaventoso', 'dell\'orrore', 'di paura', 'pauroso', '
terrificante', 'paura', 'spaventosi'],
143 'Action & Adventure': ['azione e avventura', 'd\'azione', 'di avventura', 'action', '
avventuroso', 'avventura', 'azione', 'missioni', 'avventurieri'],
144 'Sports Movies': ['sportivi', 'sport', 'sportivo', 'atleti', 'atletica', 'atletico'],
145 'Documentaries': ['documentari', 'documentario', 'docufilm', 'docu', 'documentaristici', '
documentaristica', 'educativi', 'educazionali'],
146 'Music & Musicals': ['musical', 'musica', 'cinema musicale', 'balli', 'danza', 'musicali',
'concerti'],
147 'Cult Movies': ['cult', 'di culto'],
148 'International Movies': ['internazionali', 'cinema mondiale', 'cinema estero', 'estere', '
stranieri', 'straniero'],
149 'Dramas': ['drammatici', 'drammi', 'drama', 'opere drammatiche', 'drammatiche', '
drammatico', 'piangere', 'tristi', 'triste', 'emozionanti', 'emozionante', 'emozioni', '
emozionare'],
150 'Thrillers': ['thriller', 'gialli', 'suspense', 'tensione', 'ansia', 'ansioso', '
suspensivi', 'suspenseful'],
151 'Anime Features': ['anime', 'animazione giapponese', 'cartoni giapponesi', 'anime
giapponesi', 'cartoni animati', 'animazione', 'cartoni', 'd\'animazione'],
152 'LGBTQ Movies': ['lgbtq', 'tematiche lgbtq', 'lgbtq+', 'lgbtq friendly', 'lgbtq amichevole
', 'lgbtq amichevoli'],
153 'Sci-Fi & Fantasy': ['sci-fi', 'fantasy', 'fantascienza', 'viaggi spazio-temporali', '
mondi fantastici', 'fantasia', 'fantascientifici', 'fantascienza e fantasy'],
154 'Comedies': ['commedia', 'comiche', 'commedie', 'comici', 'comico', 'comiche', 'divertenti
', 'divertente', 'ridere', 'umorismo', 'umoristici', 'umoristico', 'comedy'],
155 'Romantic Movies': ['romantici', 'romanticismo', 'cinema romantico', 'amore', 'storie d\'
amore', 'romantico', 'romantica', 'romantiche'],
156 'Independent Movies': ['indipendenti', 'indie', 'cinema indipendente', 'indipendente'],
157 'Faith & Spirituality': ['spiritualita', 'fede e spiritualita', 'religione', 'religiosi',
'religioso', 'fede', 'Dio', 'Gesù', 'divinita', 'divino'],
158 'Children & Family Movies': ['famiglia', 'per bambini', 'per famiglie', 'neonati', '
infanzia', 'bambini', 'famiglie', 'piccoli'],
159 }
160
161 mappatura_generi_serie_tv = {
162 'Anime Series': ['anime', 'animazione giapponese', 'cartoni giapponesi', 'anime giapponesi
', 'cartoni animati', 'animazione', 'cartoni', 'd\'animazione'],
163 'TV Horror': ['horror', 'spaventoso', 'dell\'orrore', 'di paura', 'pauroso', 'terrificante
', 'paura', 'spaventosi'],
164 'TV Mysteries': ['mistero', 'giallo', 'di mistero', 'gialle', 'misteri', 'misteriosi', '
misterioso', 'suspense', 'tensione'],
165 'TV Comedies': ['commedia', 'comiche', 'commedie', 'comici', 'comico', 'comiche', '
divertenti', 'divertente', 'ridere', 'umorismo', 'umoristici', 'umoristico', 'comedy'],
166 'Faith & Spirituality': ['spiritualita', 'fede e spiritualita', 'religione', 'religiosi',
'religioso', 'fede', 'Dio', 'Gesù', 'divinita', 'divino'],
167 'TV Dramas': ['drammatici', 'drammi', 'drama', 'opere drammatiche', 'drammatiche', '
drammatico', 'piangere', 'tristi', 'triste', 'emozionanti', 'emozionante', 'emozioni', '
emozionare'],
168 'Science & Nature TV': ['scienza e natura', 'documentari scientifici', 'natura', 'scienza',

```



```

169     'naturale'],
170     'Crime TV Shows': ['crime', 'gialli', 'polizieschi', 'investigativi', 'investigazioni', 'investigativo', 'poliziesco'],
171     'Stand-Up Comedy & Talk Shows': ['talk show', 'stand-up', 'standup'],
172     'Korean TV Shows': ['coreani', 'k-drama', 'drammi coreani', 'corea'],
173     'Romantic TV Shows': ['romantici', 'romanticismo', 'cinema romantico', 'amore', 'storie d\'amore', 'romantico', 'romantica', 'romantiche'],
174     'Teen TV Shows': ['adolescenziali', 'per adolescenti', 'teen', 'giovani', 'teens', 'ragazzi', 'ragazze', 'amici', 'amicizia'],
175     'British TV Shows': ['britannici', 'UK show', 'UK', 'inglesi', 'britannico'],
176     'TV Action & Adventure': ['azione e avventura', 'd\'azione', 'di avventura', 'action', 'avventuroso', 'avventura', 'azione', 'missioni', 'avventurieri'],
177     'TV Thrillers': ['thriller', 'suspense', 'tensione'],
178     'Classic & Cult TV': ['cult', 'classici', 'culto'],
179     'International TV Shows': ['internazionali', 'mondiali', 'estero'],
180     "Kids' TV": ['per bambini', 'cartoni', 'infanzia', 'infantili'],
181     'Spanish-Language TV Shows': ['in spagnolo', 'spagnole', 'lingua spagnola', 'drammi spagnoli'],
182     'Reality TV': ['reality', 'vita reale', 'competizioni'],
183     'Docuseries': ['docu serie', 'documentaristiche', 'inchieste'],
184     'TV Sci-Fi & Fantasy': ['fantascienza', 'fantasy', 'mondi fantastici', 'avventure spaziali'],
185     'Music & Musicals': ['musical', 'musica', 'cinema musicale', 'balli', 'danza', 'musicali', 'concerti'],
186 }

```

3.6 Ottenere consigli di visione in base alle preferenze dell'utente (FORM)

Funzione del chatbot che permette di ottenere dei consigli di visione partendo dalla tipologia del contenuto che l'utente vuole e successivamente dall'estrapolazione del genere, del cast e del regista di un titolo di film/serie TV che l'utente inserisce.

```

- intent: consiglia_contenuto
examples: |
- Dammi dei consigli su cosa guardare stasera
- Che film mi consigli?
- Consigliami una serie TV avvincente
- Ho bisogno di un buon film da vedere
- Suggestioni per una visione leggera
- Quali sono le ultime uscite da guardare?
- Cosa mi consigli di guardare?
- Quali sono i migliori film da vedere?
- Consigliami una serie TV interessante
- Quali sono i film più popolari?
- Dammi dei suggerimenti su cosa guardare
- Che serie TV mi consigli?
- Consigliami un film da vedere
- Quali sono le ultime uscite da non perdere?
- Sto cercando qualcosa di divertente, qualche idea?
- Ho voglia di qualcosa di emozionante, suggerimenti?
- Cerco ispirazione per il mio prossimo binge-watching, aiuto?
- Qualche proposta per una serata cinema a casa?
- Quali sono i titoli imperdibili di quest'anno?
- Che genere di contenuti sono trend ora?
- Quali sono i film più visti del momento?
- consigli?
- consigli
- suggerimenti

```

Figura 15: Intent per l'attivazione della form per permettere di ottenere dei contenuti simili ai gusti dell'utente

Una volta che l'intent è stato riconosciuto, viene attivata la FORM, la quale attiva due moduli principali: il **FormValidationAction** e la **Action** vera e propria.

```

1 # Action per validate FORM consigli contenuti solo per tipo e per titolo
2 class ValidateConsigliContenutiForm(FormValidationAction):
3     def name(self) -> Text:
4         return "validate_consigliare_contenuto_form"
5
6     def validate_tipo(
7         self,

```



```

8         slot_value: Any,
9         dispatcher: CollectingDispatcher,
10        tracker: Tracker,
11        domain: Dict[Text, Any],
12    ) -> Dict[Text, Any]:
13        tipo_contenuto = tracker.get_slot('tipo').lower()
14        logger.debug(f"Validating slot 'tipo' with value '{tipo_contenuto}'")
15
16        if tipo_contenuto in ["film", "serie", "serie tv", "movie", "tv show", "show"]:
17            if tipo_contenuto in ["film", "movie"]:
18                return {"tipo": "Movie"}
19            elif tipo_contenuto in ["serie", "serie tv", "tv show", "show"]:
20                return {"tipo": "TV Show"}
21        else:
22            dispatcher.utter_message(text="Per favore, scegli una delle opzioni disponibili: '
Film' o 'Serie'.")
23            return {"tipo": None}
24
25    def validate_titolo(
26        self,
27        slot_value: Any,
28        dispatcher: CollectingDispatcher,
29        tracker: Tracker,
30        domain: Dict[Text, Any],
31    ) -> Dict[Text, Any]:
32
33        titolo = slot_value
34        print(titolo)
35        logger.debug(f"Validating slot 'titolo' with value '{titolo}'")
36        try:
37            df = pd.read_csv(PATH_TO_CSV) # Aggiorna con il percorso corretto
38            titles = df['title'].str.lower().tolist()
39            best_match, score = process.extractOne(titolo, titles)
40        except Exception as e:
41            logger.error(f"Error reading CSV or validating title: {e}")
42            dispatcher.utter_message(text="Si e' verificato un errore nella validazione del
titolo.")
43            return {"titolo": None}
44
45        if score > 80:
46            return {"titolo": best_match}
47        else:
48            dispatcher.utter_message(text="Non ho trovato il titolo che hai inserito, potremmo
non averlo in catalogo. Puoi riprovare?")
49            return {"titolo": None}

```

Il modulo `ValidateConsigliContenutiForm` del chatbot è progettato per validare e gestire la form di input degli utenti che cercano consigli sui contenuti da visualizzare, specificamente basati su tipo e titolo del contenuto. La classe `FormValidationAction` permette di verificare l'adeguatezza dei valori inseriti negli slot tipo e titolo.

- **Validazione del Tipo:** La validazione del tipo di contenuto (film o serie TV) avviene attraverso il metodo `validate_tipo`. Questo metodo controlla se l'input dell'utente corrisponde a termini accettabili (ad esempio, "film", "serie", "movie", ecc.). Se l'input è valido, lo standardizza in "Movie" o "TV Show" per mantenere la coerenza all'interno del sistema.
- **Validazione del Titolo:** Il metodo `validate_titolo` verifica l'esistenza del titolo fornito dall'utente contro un database specificato (caricato da un file CSV). Utilizza una tecnica di fuzzy matching per trovare la corrispondenza più vicina e considera un match valido solo se il punteggio è superiore a una soglia predeterminata, garantendo così la precisione dei risultati forniti al richiedente.

```

1 # Azione per la sottomissione della form di consiglio sui contenuti.
2 # Questa azione raccoglie gli slot compilati dall'utente durante la sessione della form.
3 # Gli slot raccolti vengono utilizzati per eseguire una ricerca personalizzata nel dataset.
4 # In particolare, il titolo fornito dall'utente viene impiegato per estrarre dettagli chiave
5 # relativi al contenuto, quali il cast, il regista e i generi associati.
6 # Successivamente, questi elementi vengono utilizzati come filtri per identificare e suggerire
7 # film o serie TV che condividono almeno uno di questi attributi con il titolo di riferimento.
8 # Il processo risulta quindi in una raccolta curata di contenuti che presentano similitudini
9 # significative con le preferenze espresse dall'utente tramite il titolo indicato.
10 # Nel caso in cui la ricerca non produca risultati, viene notificato all'utente
11 # tramite un messaggio che indica l'impossibilità di trovare contenuti corrispondenti ai
   criteri specificati.
12 class SubmitConsigliareContenutoForm(Action):

```

```

13 def name(self) -> Text:
14     return "action_submit_consigliare_contenuto_form"
15
16 def run(self, dispatcher: CollectingDispatcher, tracker: Tracker, domain: Dict[Text, Any])
17     -> List[Dict[Text, Any]]:
18     df = pd.read_csv(PATH_TO_CSV)
19     slot_values = tracker.slots
20
21     # Estraiamo il titolo dallo slot
22     title = slot_values['titolo']
23     # Estraiamo il tipo dallo slot
24     tipo = slot_values['tipo']
25     print(f"Titolo: {title}, Tipo: {tipo}")
26     titles = df['title'].tolist()
27     best_match, score = process.extractOne(title, titles)
28     if score > 90:
29         result = df[df['title'].str.lower() == best_match.lower()]
30
31     if not result.empty:
32         # Estraiamo i dettagli relativi al titolo selezionato
33         cast = result.iloc[0]['cast'].split(", ")
34         director = result.iloc[0]['director']
35         genres = ast.literal_eval(result.iloc[0]['genres_list'])
36
37         # Prepariamo i filtri basati su cast e director
38         cast_filter = df['cast'].apply(lambda x: any(actor in x for actor in cast if x !=
39 'Non disponibile'))
40         director_filter = df['director'].str.contains(director, case=False, na=False)
41         genres_filter = df['genres_list'].apply(lambda x: any(genre in ast.literal_eval(x)
42 for genre in genres if x))
43
44         if tipo == 'Movie':
45             similar_content = df[(df['title'].str.lower() != best_match.lower()) &
46 (df['type'] == 'Movie') &
47 (cast_filter | director_filter | genres_filter)]
48
49         elif tipo == 'TV Show':
50             similar_content = df[(df['title'].str.lower() != best_match.lower()) &
51 (df['type'] == 'TV Show') &
52 (cast_filter | director_filter | genres_filter)]
53
54         if not similar_content.empty:
55             shuffled_results = similar_content.sample(frac=1).reset_index(drop=True)
56             results = shuffled_results.head(30) # ridai 30 titoli per semplicità
57             intro = f"Ecco alcuni contenuti simili a '{title}':\n\n"
58             risposta = intro
59             for index, row in results.iterrows():
60                 titolo = row['title'].title()
61                 tipo = row['type']
62                 descrizione = row['description']
63                 regista = row['director']
64                 cast = row['cast']
65                 categoria = row['rating_category']
66                 generi = ast.literal_eval(row['genres_list']) if pd.notna(row['genres_list
67 ']) else ["Non specificato"]
68                 risposta += f"\n- {titolo} ({'Film' if tipo == 'Movie' else 'Serie TV'}):\n
69 n Descrizione: {descrizione}\n Regia: {regista}\n Cast: {cast}\n Categoria di
70 pubblico: {categoria}\n Genere/i: {'', '.join(generi)}\n\n"
71             similar_content.to_csv("actions/risultati_consigliati.csv", index=False)
72             dispatcher.utter_message(text=risposta)
73         else:
74             dispatcher.utter_message(text=f"Non ho trovato contenuti simili a '{title}'.")
75         else:
76             dispatcher.utter_message(text=f"Non ho trovato informazioni per il titolo '{title
77 }'.")
78     return []

```

La classe SubmitConsigliareContenutoForm estende la funzionalità del chatbot elaborando gli input validati dalla form. Una volta che un utente completa la form, questa classe prende il comando per:

- **Estrarre Dettagli:** Basandosi sul titolo fornito, estrae informazioni dettagliate come cast, regista e generi dal dataset.
- **Filtrare e Suggestire Contenuti Correlati:** Utilizza i dettagli estratti per filtrare e identificare altri contenuti (film o serie TV) nel database che condividono attributi simili con il contenuto di riferimento.

Questo processo si basa su un insieme di filtri che considerano cast, regista e generi.

- **Risposta all'Utente:** Il chatbot quindi presenta una lista di consigli che rappresentano contenuti simili al titolo di riferimento. Se non trova contenuti correlati, informa l'utente dell'assenza di risultati.

L'output ottenuto è una lista di Film/Serie TV (la decisione spetta all'utente) la quale condivide degli aspetti simili al titolo che passa l'utente, come il cast, il regista o i generi.

```
Your input -> cosa mi consigli?
Certo! Dimmi cosa ti piace e ti consiglierò qualcosa.
Dimmi se stai cercando un film o una serie TV.
Your input -> film
Dimmi un titolo a cui fare riferimento, lo userò direttamente come base per consigliarti altri contenuti simili.
Your input -> Kung Fu Panda
Grazie per avermi fornito le tue preferenze. Sto cercando i contenuti che potrebbero interessarti.
Ecco alcuni contenuti simili a 'kung fu panda':

- Latte And The Magic Waterstone (Film):
  Descrizione: When a greedy bear steals a magic stone to keep the forest's water for himself, a brave hedgehog and a timid squirrel must work together to retrieve it.
  Regia: Regina Welker, Nina Wels
  Cast: Ashley Bornancin, Carter Hastings
  Categoria di pubblico: Kids - Suitable for all children
  Genere/i: Children & Family Movies, Comedies

- Ride Like A Girl (Film):
  Descrizione: The daughter of a horse trainer, an ambitious girl sets her sights on becoming the first female jockey to win the Melbourne Cup. Based on a true story.
  Regia: Rachel Griffiths
  Cast: Teresa Palmer, Sam Neill, Sullivan Stapleton, Stevie Payne, Brooke Satchwell, Magda Szubanski, Aaron Glenane, Damien Garvey, Sophia Crawford, Henry Nixon
  Categoria di pubblico: Parental Guidance Suggested - Suitable for children with parental guidance
  Genere/i: Children & Family Movies, Dramas, Independent Movies
```

Figura 16: Output della lista di film che hanno in comune, con "Kung Fu Panda", il cast, il regista e il genere. Per comodità vengono stampati solo i primi 30 risultati, in maniera randomica. In questa maniera l'utente otterrà sempre nuovi titoli.

Implementazione delle Rules per la Form di Consigli sui Contenuti

Per garantire un'interazione efficace e strutturata con l'utente nel processo di consigli sui contenuti, il chatbot sfrutta una serie di rules specifiche nel suo sistema di dialogo. Queste rules sono essenziali per attivare e gestire correttamente la form di consigli sui contenuti, assicurando che gli input dell'utente siano raccolti e processati in modo ordinato e logico.

```
#RULE FORM CONSIGLIA CONTENUTI
- rule: Activate form consiglia contenuto
  steps:
    - intent: consiglia_contenuto
    - action: utter_consigli_di_visione
    - action: consigliare_contenuto_form
    - active_loop: consigliare_contenuto_form
    - slot_was_set:
      - requested_slot: tipo
    - slot_was_set:
      - requested_slot: titolo
    - active_loop: null
    - action: utter_submit
    - action: action_submit_consigliare_contenuto_form
    - action: utter_slots_values
    - action: action_reset_slots
```

Figura 17: Rule per il corretto funzionamento e attivazione della FORM.

Di seguito sono descritte le rules utilizzate:

- **Activate form consiglia contenuto:** Questa rule è progettata per essere attivata quando l'utente esprime l'intento di ricevere consigli sui contenuti (`consiglia_contenuto`). La sequenza di passaggi definiti nella rule comprende:
 1. **Riconoscimento dell'intent:** L'utente mostra interesse per i consigli sui contenuti attraverso l'intento `consiglia_contenuto`.
 2. **Messaggio Iniziale:** Viene inviato un messaggio preliminare all'utente (`utter_consigli_di_visione`) per informarlo sui tipi di consigli disponibili e prepararlo a inserire i dettagli richiesti.
 3. **Attivazione della Form:** La form `consigliare_contenuto_form` viene attivata per raccogliere e validare gli input dell'utente relativi al tipo e al titolo del contenuto desiderato.
 4. **Loop Attivo e Raccolta Slot:** La form rimane in uno stato attivo (`active_loop`), durante il quale vengono impostati gli slot per tipo e titolo, garantendo che questi dati siano raccolti prima di procedere.

5. **Chiusura del Loop Attivo:** Una volta raccolti tutti gli input necessari, il loop attivo viene chiuso (`active_loop: null`), segnalando la fine della raccolta dati.
6. **Sottomissione dei Dati Raccolti:** L'azione `utter_submit` comunica all'utente che le informazioni sono state raccolte e che il processo di consiglio è in corso.
7. **Elaborazione dei Dati:** La form sottomette i dati alla logica di backend attraverso l'azione `action_submit_consigliare_contenuto_form`, che utilizza gli input raccolti per trovare contenuti correlati.
8. **Comunicazione dei Risultati:** Il chatbot comunica i valori degli slot raccolti (`utter_slots_values`) e, successivamente, resetta gli slot per future interazioni (`action_reset_slots`).

Gestione dell'uscita anticipata dalla FORM

La story "stop form consiglia contenuto" è stata progettata per gestire una situazione in cui l'utente desidera interrompere il processo di compilazione della form di consigli sui contenuti. Questo scenario è particolarmente utile per migliorare l'esperienza utente, offrendo la flessibilità di fermarsi a metà di un processo senza perdere il controllo dell'interazione. La sequenza di passaggi definiti nella story è la seguente:

```
- story: stop form consiglia contenuto
  steps:
    - intent: consiglia_contenuto
    - action: utter_consigli_di_visione
    - action: consigliare_contenuto_form
    - active_loop: consigliare_contenuto_form
    - slot_was_set:
      - requested_slot: tipo
    - intent: stop_intent
    - action: action_deactivate_loop
    - active_loop: null
    - slot_was_set:
      - requested_slot: null
    - action: action_reset_slots
```

Figura 18: Storia per la gestione dello stop della FORM

- **Riconoscimento dell'intent per il consiglio da parte del chatbot:** L'utente inizia esprimendo l'intento di ricevere consigli sui contenuti (`consiglia_contenuto`).
- **Messaggio Iniziale:** Il sistema risponde con un messaggio iniziale (`utter_consigli_di_visione`) che descrive i tipi di consigli disponibili e invita l'utente a fornire maggiori dettagli sul contenuto desiderato.
- **Attivazione della Form:** Viene attivata la form `consigliare_contenuto_form` per raccogliere i dettagli specifici dall'utente, come il tipo di contenuto.
- **Raccolta del Tipo di Contenuto:** La form imposta uno slot per raccogliere il tipo di contenuto desiderato dall'utente (`requested_slot: tipo`).
- **Riconoscimento dell'Intento di Stop:** L'utente decide di interrompere il processo e esprime questo desiderio attraverso un intento specifico (`stop_intent`).
- **Disattivazione del Loop Attivo:** In risposta all'intento di stop, il chatbot esegue l'azione `action_deactivate_loop`, che termina il loop attivo della form, indicando che non sono più richiesti ulteriori input dall'utente.
- **Reset dello Stato del Loop e degli Slot:** Il loop viene segnato come non attivo (`active_loop: null`) e lo slot richiesto viene resettato (`requested_slot: null`), ripristinando lo stato iniziale della conversazione.
- **Reset di Tutti gli Slot:** Infine, viene eseguita l'azione `action_reset_slots` per cancellare qualsiasi dato precedentemente inserito negli slot durante la sessione corrente, garantendo che eventuali tentativi futuri di interazione inizino senza residui di sessioni precedenti.

Questa story assicura che gli utenti possano interrompere il processo di raccolta dati in qualsiasi momento, migliorando così la loro autonomia e soddisfazione nell'uso del chatbot. Garantendo un'uscita pulita e controllata dal processo, il sistema mostra un'attenzione particolare verso le esigenze e il comfort dell'utente.

4 Connessione a Telegram e Test

4.1 Connessione

Nell'ultima fase del progetto, il chatbot è stato integrato con una piattaforma di messaggistica istantanea, specificatamente Telegram, per permettere agli utenti di interagire efficacemente attraverso un'interfaccia grafica moderna e accessibile da smartphone. L'uso del servizio ngrok ha facilitato la creazione di un endpoint di comunicazione per il server locale, essenziale per il corretto funzionamento del chatbot in un ambiente reale.



Figura 19: Configurazione della connessione nel file `credentials.yml`

I parametri configurati nel file `credentials.yml` comprendono:

- **access_token**: Token fornito da Telegram tramite BotFather, necessario per autenticare il bot.
- **verify**: Nome del bot assegnato durante la fase di configurazione su Telegram.
- **webhook_url**: Endpoint generato con ngrok, che stabilisce un canale di comunicazione diretto tra il server locale e Telegram.

4.2 Testing

Dopo aver completato l'integrazione del chatbot con la piattaforma di messaggistica istantanea selezionata, abbiamo intrapreso una fase di test approfonditi per simulare le interazioni tipiche che gli utenti potrebbero avere con il sistema. Particolare attenzione è stata dedicata alla simulazione di comportamenti di utenti distratti, ovvero utenti che potrebbero inserire input errati accidentalmente. Questo scenario di test è stato cruciale per verificare l'efficacia dei controlli di validità che abbiamo implementato nel sistema.

Di seguito sono riportate alcune immagini che illustrano i risultati ottenuti durante questa fase di test.

Avvio del Bot

Per avviare il bot all'interno dell'applicazione telegram, bisogna premere sul pulsante "AVVIA" presente nella chat con il bot.

Visualizzare tutti i generi dei contenuti presenti

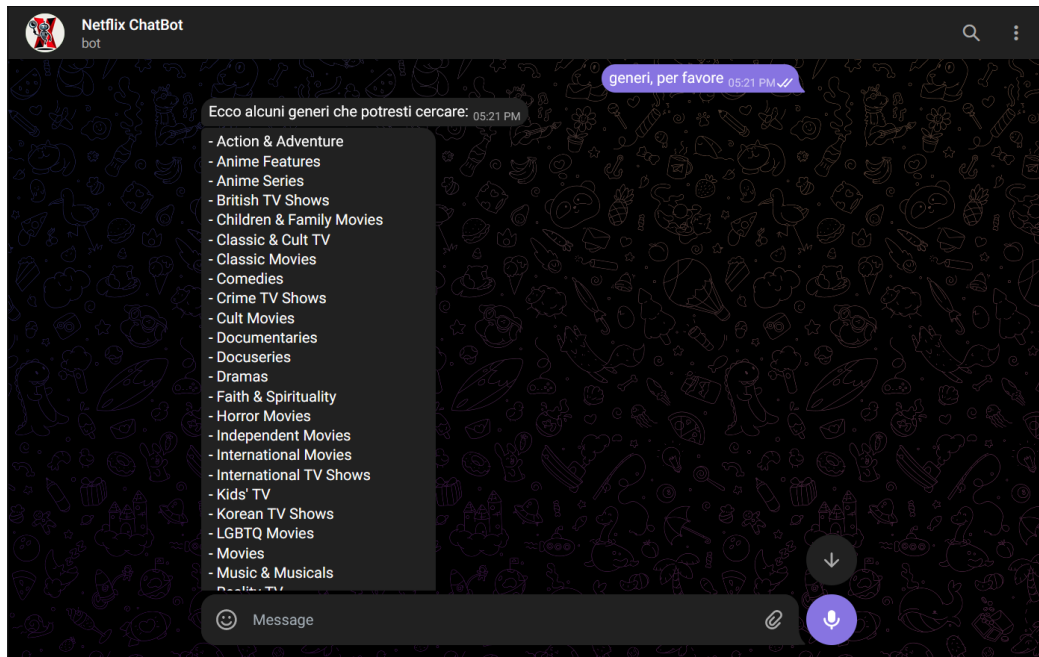


Figura 20: Funzione di visualizzazione dei generi sull'app Telegram

Visualizzare tutti i paesi dei contenuti presenti

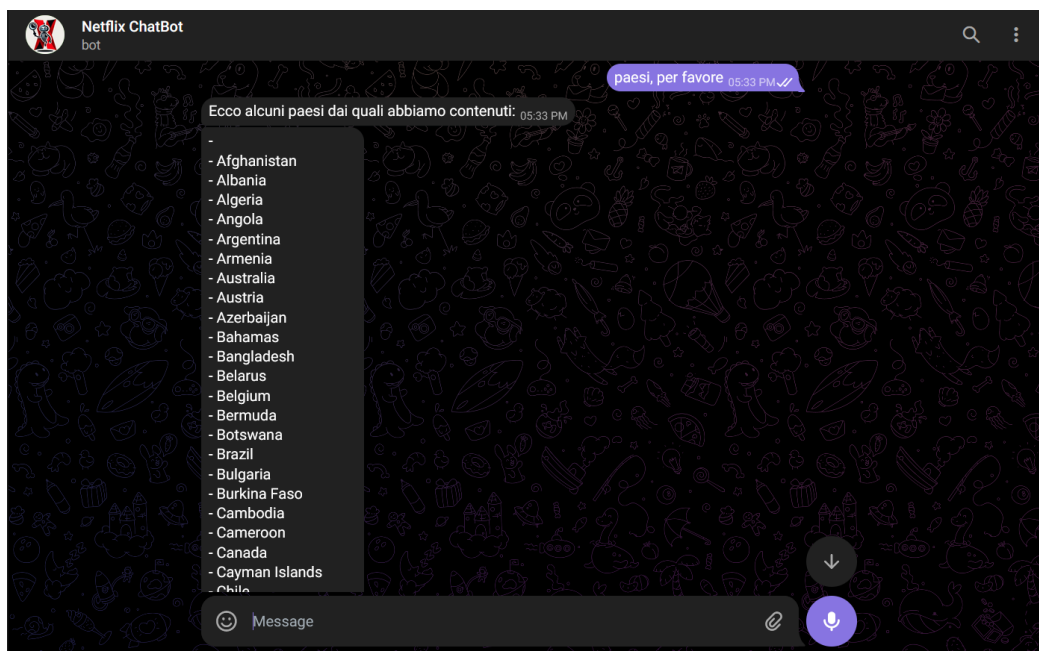


Figura 21: Funzione di visualizzazione dei paesi sull'app Telegram

Visualizzare tutte le categorie di visione dei contenuti presenti

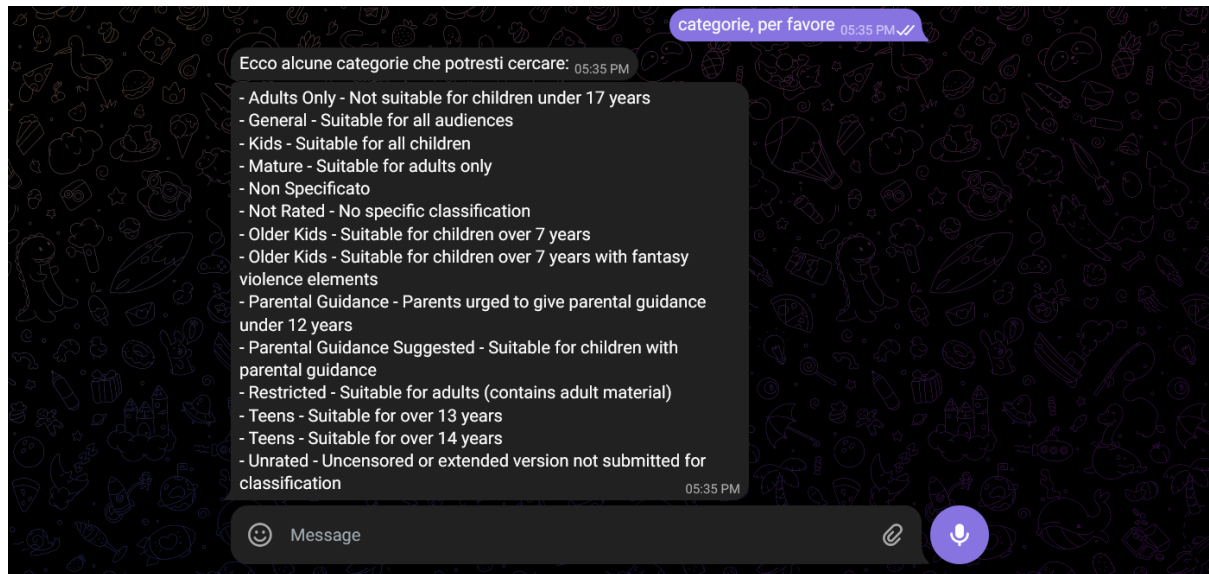


Figura 22: Funzione di visualizzazione delle categorie sull'app Telegram

Visualizzare tutte le informazioni del contenuto tramite il titolo

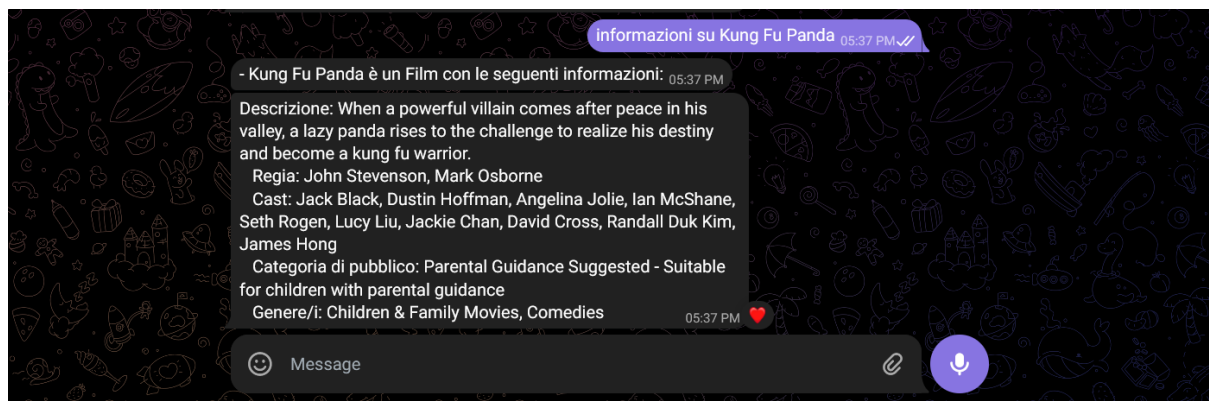


Figura 23: Funzione di visualizzazione delle informazioni dei contenuti Netflix sull'app Telegram

Action di ricerca contenuti filtrati a seconda delle preferenze dell'utente

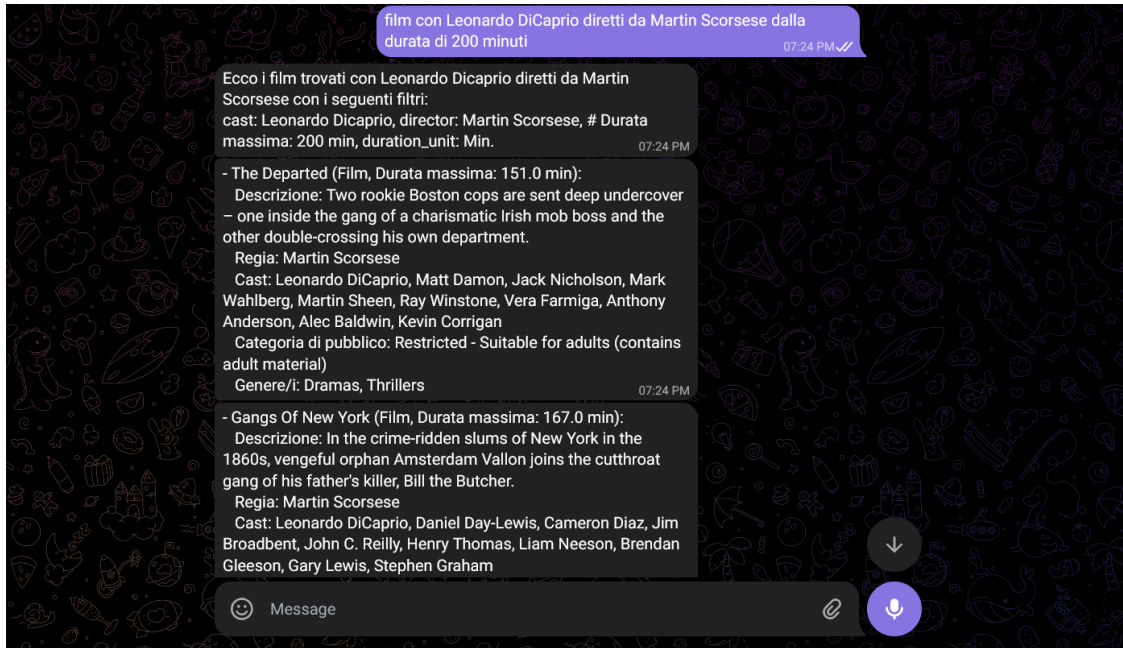


Figura 24: Funzione di ricerca dei contenuti filtrati secondo le preferenze dell'utente sull'app Telegram

Ottenere consigli di visione in base alle preferenze dell'utente (FORM)

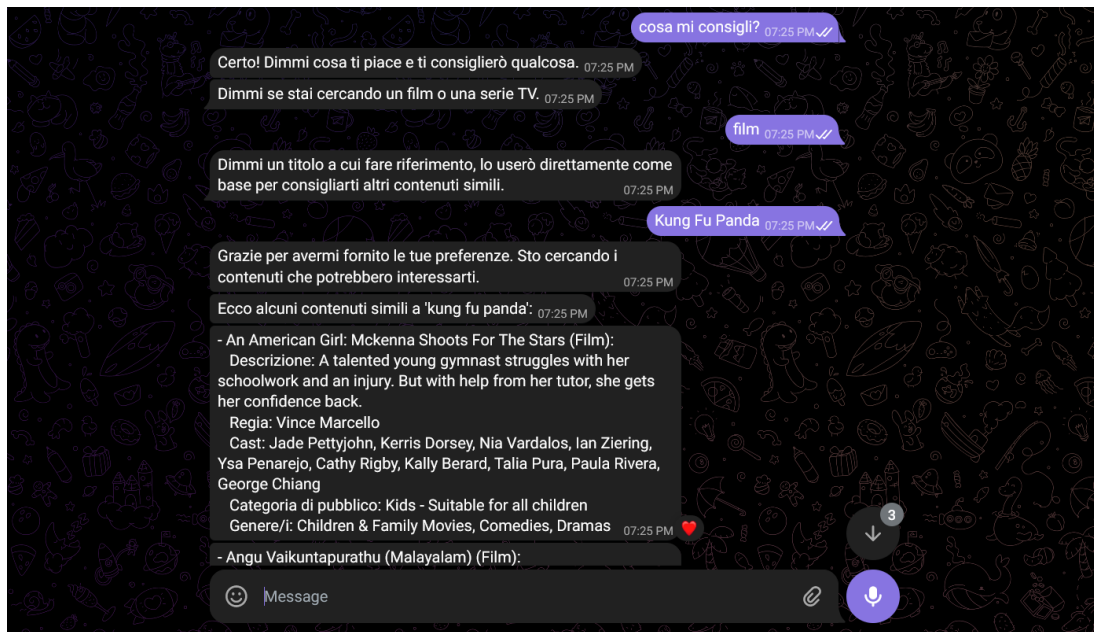


Figura 25: Visualizzazione dei consigli del Chatbot per l'utente in base al titolo e tipo di contenuto passato sull'app Telegram

5 Conclusioni e sviluppi futuri

Nel presente progetto, abbiamo impiegato il framework Rasa AI per lo sviluppo di un chatbot destinato al supporto degli utenti nella navigazione e selezione dei contenuti offerti da Netflix. Le funzionalità realizzate, approfonditamente illustrate nel Capitolo 3, hanno lo scopo di facilitare l'interazione con l'ampio catalogo disponibile, permettendo una ricerca intuitiva e personalizzata basata sulle preferenze degli utenti.

Sviluppi Futuri

Guardando al futuro, proponiamo diverse evoluzioni per il chatbot:

- **Ampliamento del Database:** Estendere le capacità del chatbot integrando un database più vasto che comprenda titoli da diverse piattaforme di streaming.
- **Funzionalità per le nuove uscite:** Implementare una nuova funzionalità che restituisca all'utente le nuove aggiunte nella piattaforma, secondo un range temporale definito.
- **Analisi Sentimentale:** Implementare funzionalità di analisi sentimentale per affinare le raccomandazioni basate sulle recensioni degli utenti.
- **Interfaccia Utente Avanzata:** Sviluppare un'interfaccia più interattiva e visuale che possa essere integrata con le principali app di messaggistica e dispositivi smart home.
- **Dettagli Approfonditi sui Contenuti:** Arricchire le informazioni fornite dal chatbot con dettagli più completi sui contenuti, come valutazioni e recensioni critiche.

Tali miglioramenti potrebbero trasformare il chatbot in un assistente virtuale all'avanguardia per l'intrattenimento, rispondendo proattivamente alle richieste di un pubblico eterogeneo e in continua evoluzione.



Link al repository GitHub:

https://github.com/AlessandroRongoni/Chat_Bot