AAI520_Final_Group_6_Retrieval_Based_Model

October 16, 2023

0.1 AAI-520

0.2 Final Project - Group 6

0.3 Retrieval-Based Chatbot for Movie Info utilizing the Cornell Movie Dialogs Corpus

[ ]:
```
#@title 1: Load the related Libraries
from __future__ import absolute_import, division, print_function,␣
 ↪unicode_literals
import argparse
import codecs
import csv
import os
import pandas as pd
```

##2: Load Movie Data Corpus

[ ]:
```
#@title 2.1: Load line_data
#Load the Line file

def loadLines(filePath, fields):
    """
    Args:
        filePath (str): full path to the file to load
        fields (set<str>): fields to extract
    Return:
        dict<dict<str>>: the extracted fields for each line
    """
    lines = {}

    with open(filePath, 'r', encoding='iso-8859-1') as f:
        for line in f:
            values = line.split(" +++$+++ ")

            # Extract fields
            lineObj = {}
            for i, field in enumerate(fields):
                lineObj[field] = values[i]
```

```
            lines[lineObj['lineID']] = lineObj

    return lines

# Usage example
fields_to_extract = ['lineID', 'characterID', 'movieID', 'character', 'text']
file_path = "/content/drive/MyDrive/AAI-520/Final/Data/movie_lines.txt"
lines_data = loadLines(file_path, fields_to_extract)
```

[ ]:
```
#@title 2.2: Load the charachter_data
def loadCharacterMetadata(filePath, fields):
    """
    Args:
        filePath (str): full path to the character metadata file to load
        fields (set<str>): fields to extract
    Return:
        dict<dict<str>>: the extracted fields for each character
    """
    characters = {}

    with open(filePath, 'r', encoding='iso-8859-1') as f:
        for line in f:
            values = line.split(" +++$+++ ")

            # Extract fields
            characterObj = {}
            for i, field in enumerate(fields):
                characterObj[field] = values[i]
                value = values[i]
                # Remove commas from the movieTitle field
                if field == 'movieTitle':
                    value = value.replace(',', '')  # Remove commas
                characterObj[field] = value

            characters[characterObj['characterID']] = characterObj

    return characters

# Usage example
character_fields_to_extract = ['characterID', 'characterName', 'movieID',␣
 ↪'movieTitle', 'gender', 'position']
character_file_path = "/content/drive/MyDrive/AAI-520/Final/Data/
 ↪movie_characters_metadata.txt"
character_data = loadCharacterMetadata(character_file_path,␣
 ↪character_fields_to_extract)
```

[ ]:
```
#@title 2.3: Load conversation_data
def loadConversations(filePath, fields):
    """
    Args:
        filePath (str): full path to the conversations file to load
        fields (set<str>): fields to extract
    Return:
        list<dict<str>>: a list of dictionaries representing conversations
    """

    conversations = {}

    with open(filePath, 'r', encoding='iso-8859-1') as f:
        for line in f:
            values = line.split(" +++$+++ ")

            # Extract fields
            conversationObj = {}
            for i, field in enumerate(fields):
                conversationObj[field] = values[i]


            conversations[conversationObj['movieID']] = conversationObj

    return conversations

# Usage example
conversation_fields_to_extract = ['characterID1', 'characterID2', 'movieID',␣
 ↪'utteranceIDs']
conversation_file_path = "/content/drive/MyDrive/AAI-520/Final/Data/
 ↪movie_conversations.txt"
conversation_data = loadConversations(conversation_file_path,␣
 ↪conversation_fields_to_extract)
```

[ ]:
```
#@title 2.4: Load the title_data
def loadMovieTitlesMetadata(filePath, fields):
    """
    Args:
        filePath (str): full path to the movie titles metadata file to load
        fields (set<str>): fields to extract
    Return:
        dict<dict<str>>: the extracted fields for each movie title
    """
    movie_titles = {}

    with open(filePath, 'r', encoding='iso-8859-1') as f:
        for line in f:
```

```
            values = line.split(" +++$+++ ")

            # Extract fields
            movieTitleObj = {}
            for i, field in enumerate(fields):
                movieTitleObj[field] = values[i]

            movie_titles[movieTitleObj['movieID']] = movieTitleObj

    return movie_titles

# Usage example
movie_title_fields_to_extract = ['movieID', 'movieTitle', 'releaseYear',␣
 ↪'imdbRating', 'numVotes', 'genres']
movie_title_file_path = "/content/drive/MyDrive/AAI-520/Final/Data/
 ↪movie_titles_metadata.txt"
movie_title_data = loadMovieTitlesMetadata(movie_title_file_path,␣
 ↪movie_title_fields_to_extract)
```

[ ]:
```
def loadMovieTitlesMetadata(filePath, fields):
    """
    Args:
        filePath (str): full path to the movie titles metadata file to load
        fields (set<str>): fields to extract
    Return:
        dict<dict<str>>: the extracted fields for each movie title
    """
    movie_titles = {}

    with open(filePath, 'r', encoding='iso-8859-1') as f:
        for line in f:
            values = line.split(" +++$+++ ")

            # Extract fields
            movieTitleObj = {}
            for i, field in enumerate(fields):
                value = values[i]
                # Remove commas from the movieTitle field
                if field == 'movieTitle':
                    value = value.replace(',', '')  # Remove commas
                movieTitleObj[field] = value

            movie_titles[movieTitleObj['movieID']] = movieTitleObj

    return movie_titles

# Usage example
```

```
movie_title_fields_to_extract = ['movieID', 'movieTitle', 'releaseYear',␣
 ↪'imdbRating', 'numVotes', 'genres']
movie_title_file_path = "/content/drive/MyDrive/AAI-520/Final/Data/
 ↪movie_titles_metadata.txt"
movie_title_data = loadMovieTitlesMetadata(movie_title_file_path,␣
 ↪movie_title_fields_to_extract)
```

[ ]:
```
#@title 2.5: Load the url_data
def loadRawScriptUrls(filePath, fields):
    """
    Args:
        filePath (str): full path to the raw script URLs file to load
        fields (list<str>): fields to extract
    Return:
        dict<str, dict<str>>: a dictionary with movieID as keys and␣
 ↪dictionaries with field values as values
    """
    urls = {}

    with open(filePath, 'r', encoding='iso-8859-1') as f:
        for line in f:
            values = line.split(" +++$+++ ")

            # Extract fields
            loadRawScriptUrls = {}
            for i, field in enumerate(fields):
                loadRawScriptUrls[field] = values[i]
                value = values[i]
                # Remove commas from the movieTitle field
                if field == 'url':
                    value = value.replace(',', '')  # Remove commas
                loadRawScriptUrls[field] = value
                # Remove commas from the movieTitle field
                if field == 'scriptURL':
                    value = value.replace(',', '')  # Remove commas
                loadRawScriptUrls[field] = value

            urls[loadRawScriptUrls['movieID']] = loadRawScriptUrls

    return urls

# Usage example
raw_script_urls_fields_to_extract = ['movieID', 'scriptURL', 'url']
raw_script_urls_file_path = "/content/drive/MyDrive/AAI-520/Final/Data/
 ↪raw_script_urls.txt"
script_urls_data = loadRawScriptUrls(raw_script_urls_file_path,␣
 ↪raw_script_urls_fields_to_extract)
```

[ ]:
```
#@title 3: Convert dictionaries/arrays to DataFrames
df_lines = pd.DataFrame.from_dict(lines_data, orient='index')
df_characters = pd.DataFrame.from_dict(character_data, orient='index')
df_conversations = pd.DataFrame.from_dict(conversation_data, orient='index')
df_movie_titles = pd.DataFrame.from_dict(movie_title_data, orient='index')
df_script_urls =  pd.DataFrame.from_dict(script_urls_data, orient='index')
```

[ ]:
```
#@title 3.1: Split the genres and create separate columns
# Remove square brackets and single quotes and split the genres
df_movie_titles['genres'] = df_movie_titles['genres'].str.replace(r"\[|\]|'",␣
 ↪'').str.split(', ')


max_genres = df_movie_titles['genres'].apply(len).max()  # Find the maximum␣
 ↪number of genres in any row
for i in range(1, max_genres + 1):
    df_movie_titles[f'genre_{i}'] = df_movie_titles['genres'].apply(lambda x:␣
 ↪x[i - 1] if len(x) >= i else None)

# Drop the original "genres" column
df_movie_titles = df_movie_titles.drop(columns=['genres'])
```

```
<ipython-input-9-d46e1b3df1bf>:3: FutureWarning: The default value of regex will
change from True to False in a future version.
  df_movie_titles['genres'] = df_movie_titles['genres'].str.replace(r"\[|\]|'",
'').str.split(', ')
```

[ ]:
```
#@title 3.2: Verify the new fields
df_movie_titles.head()
```

[ ]:

	movieID	movieTitle	releaseYear	imdbRating	numVotes	\
m0	m0	10 things i hate about you	1999	6.90	62847	
m1	m1	1492: conquest of paradise	1992	6.20	10421	
m2	m2	15 minutes	2001	6.10	25854	
m3	m3	2001: a space odyssey	1968	8.40	163227	
m4	m4	48 hrs.	1982	6.90	22289	

	genre_1	genre_2	genre_3	genre_4	genre_5	genre_6	genre_7	\
m0	comedy	romance\n	None	None	None	None	None	
m1	adventure	biography	drama	history\n	None	None	None	
m2	action	crime	drama	thriller\n	None	None	None	
m3	adventure	mystery	sci-fi\n	None	None	None	None	
m4	action	comedy	crime	drama	thriller\n	None	None	

	genre_8	genre_9	genre_10	genre_11
m0	None	None	None	None
m1	None	None	None	None

```
m2    None    None    None    None
m3    None    None    None    None
m4    None    None    None    None
```

[ ]:
```
#@title 4: Merge DataFrames based on Movie ID
merged_data = df_lines.merge(df_characters, on='movieID', how='inner')
merged_data = merged_data.merge(df_conversations, on='movieID', how='inner')
merged_data = merged_data.merge(df_movie_titles, on='movieID', how='inner')
merged_data = merged_data.merge(df_script_urls, on='movieID', how='inner')
#drop duplicate fields
merged_data = merged_data.drop("movieTitle_y", axis=1)
merged_data = merged_data.drop("scriptURL", axis=1)
```

[ ]:
```
#@title 4.1: Verify merged_data df
merged_data.head()
```

[ ]:
```
  lineID characterID_x movieID character            text characterID_y  \
0  L1045            u0      m0    BIANCA  They do not!\n            u0
1  L1045            u0      m0    BIANCA  They do not!\n            u1
2  L1045            u0      m0    BIANCA  They do not!\n            u2
3  L1045            u0      m0    BIANCA  They do not!\n            u3
4  L1045            u0      m0    BIANCA  They do not!\n            u4

  characterName               movieTitle_x gender position  … genre_3  \
0        BIANCA  10 things i hate about you      f      4\n  …    None
1         BRUCE  10 things i hate about you      ?      ?\n  …    None
2       CAMERON  10 things i hate about you      m      3\n  …    None
3      CHASTITY  10 things i hate about you      ?      ?\n  …    None
4          JOEY  10 things i hate about you      m      6\n  …    None

  genre_4 genre_5 genre_6 genre_7 genre_8 genre_9 genre_10 genre_11  \
0    None    None    None    None    None    None     None     None
1    None    None    None    None    None    None     None     None
2    None    None    None    None    None    None     None     None
3    None    None    None    None    None    None     None     None
4    None    None    None    None    None    None     None     None

                                                 url
0  http://www.dailyscript.com/scripts/10Things.ht…
1  http://www.dailyscript.com/scripts/10Things.ht…
2  http://www.dailyscript.com/scripts/10Things.ht…
3  http://www.dailyscript.com/scripts/10Things.ht…
4  http://www.dailyscript.com/scripts/10Things.ht…

[5 rows x 28 columns]
```

[ ]:
```
#@title 4.2: Verify the number of rows in the DataFrame
num_rows = merged_data.shape[0]

# Print the number of rows
print("Number of Rows:", num_rows)
```

```
Number of Rows: 4874548
```

[ ]:
```
#@title 5: Prepare the work for the Chatbot
#Load transformer for chatbot operations
!pip3 install transformers
```

```
Collecting transformers
  Downloading transformers-4.34.0-py3-none-any.whl (7.7 MB)
                             7.7/7.7 MB
49.2 MB/s eta 0:00:00
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-
packages (from transformers) (3.12.4)
Collecting huggingface-hub<1.0,>=0.16.4 (from transformers)
  Downloading huggingface_hub-0.18.0-py3-none-any.whl (301 kB)
                             302.0/302.0
kB 27.0 MB/s eta 0:00:00
Requirement already satisfied: numpy>=1.17 in
/usr/local/lib/python3.10/dist-packages (from transformers) (1.23.5)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from transformers) (23.2)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-
packages (from transformers) (6.0.1)
Requirement already satisfied: regex!=2019.12.17 in
/usr/local/lib/python3.10/dist-packages (from transformers) (2023.6.3)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-
packages (from transformers) (2.31.0)
Collecting tokenizers<0.15,>=0.14 (from transformers)
  Downloading
tokenizers-0.14.1-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl
(3.8 MB)
                             3.8/3.8 MB
80.1 MB/s eta 0:00:00
Collecting safetensors>=0.3.1 (from transformers)
  Downloading
safetensors-0.4.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl
(1.3 MB)
                             1.3/1.3 MB
66.1 MB/s eta 0:00:00
Requirement already satisfied: tqdm>=4.27 in
/usr/local/lib/python3.10/dist-packages (from transformers) (4.66.1)
Requirement already satisfied: fsspec>=2023.5.0 in
```

```
/usr/local/lib/python3.10/dist-packages (from huggingface-
hub<1.0,>=0.16.4->transformers) (2023.6.0)
Requirement already satisfied: typing-extensions>=3.7.4.3 in
/usr/local/lib/python3.10/dist-packages (from huggingface-
hub<1.0,>=0.16.4->transformers) (4.5.0)
Collecting huggingface-hub<1.0,>=0.16.4 (from transformers)
  Downloading huggingface_hub-0.17.3-py3-none-any.whl (295 kB)
                            295.0/295.0
kB 28.0 MB/s eta 0:00:00
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.10/dist-packages (from requests->transformers) (3.3.0)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-
packages (from requests->transformers) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.10/dist-packages (from requests->transformers) (2.0.6)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.10/dist-packages (from requests->transformers)
(2023.7.22)
Installing collected packages: safetensors, huggingface-hub, tokenizers,
transformers
Successfully installed huggingface-hub-0.17.3 safetensors-0.4.0
tokenizers-0.14.1 transformers-4.34.0
```

[ ]:
```python
#@title 5.1: Building the Chatbot class: Include the building of recognizing␣
 ↪the input words, this is done with NLP services. Then a query to match to␣
 ↪the dataset and produce a response
#Load related libraries
import random
import json
import re
import numpy as np
from transformers import AutoTokenizer, AutoModelForSeq2SeqLM
#from transformers import AutoModelForCausalLM, AutoTokenizer
import torch

# Build the Retrieval-Based chatbot. Define the If and Else for query of data
class Chatbot:
    def __init__(self, data_frame, pre_trained_model_name):
        self.corpus = data_frame.to_dict(orient="records") #Convert data to␣
 ↪Dictionaries

        #Load the tokenizer service
        self.tokenizer = AutoTokenizer.from_pretrained(pre_trained_model_name)
        #Load the NLP model
        self.model = AutoModelForSeq2SeqLM.
 ↪from_pretrained(pre_trained_model_name)
```

```
        #self.model = AutoModelForCausalLM.
↪from_pretrained(pre_trained_model_name)

        #Can use to view the entire loaded data
        print("Loaded movie data:")
        for movie in self.corpus[:5]:
            print(movie)

        print(f"Total movies loaded: {len(self.corpus)}")

    def generate_response(self, query, conversation, subject): # How to respond
        #response = None

        if subject == "releaseYear":
            response = f"The movie {conversation['movieTitle_x']} was released␣
↪in {conversation['releaseYear']}."
        elif subject == "imdbRating":
            response = f"The IMDb rating of {conversation['movieTitle_x']} is␣
↪{conversation['imdbRating']}."
        elif subject == "genre_1":
            response = f"The genres of {conversation['movieTitle_x']} is␣
↪{conversation['genre_1']}."
        elif subject == "url":
            response = f"I don't have actor info but here is the URL for␣
↪{conversation['movieTitle_x']}: {conversation['url']} for more info."

        else:
            response = "I can only answer questions about release year or IMDB␣
↪rating."

        return response

    def identify_subject(self, query):# Identify the subject in the input text
        movie_keywords = ["hi","hello", "help", "genre", "genres", "movie",␣
↪"film", "imdb", "rated", "top_rated", "top_rating", "top", "lowest_rated",␣
↪"lowest_rating", "lowest", "rating", "ratings", "release", "year", "votes",␣
↪"actors","actor", "actress", "genres", "gender", "info", "information",␣
↪"url", "website"]
        subject = None
        movie = None
        tokens = query.lower().split()

        for keyword in movie_keywords:
            if keyword in tokens:
                if keyword in ["genres", "genre"]:
                    subject = "genre_1"
```

```
                elif keyword in ["movie", "film", "movies"]:
                    subject = "movieTitle_x"
                    movie_idx = tokens.index(keyword)
                    if movie_idx < len(tokens) - 1:
                        movie = tokens[movie_idx + 1].rstrip('?')
                elif keyword in ["imdb", "rating", "ratings"]:
                    subject = "imdbRating"
                elif keyword in ["top_rated", "top_rating", "top"]:
                    subject = "timdbRating"
                    movie = None
                elif keyword in ["lowest_rated", "lowest_rating", "lowest"]:
                    subject = "limdbRating"
                    movie = None
                elif keyword in ["release", "year"] and subject is None:
                    subject = "releaseYear"
                elif keyword in ["release", "year"] and subject is␣
↪'movieTitle_x':
                    subject = "releaseYear"
                elif keyword in ["actors","actor", "actress", "script",␣
↪"information"]:
                    subject = "url"
                elif keyword in ["hi","hello"]:
                    subject = "hi"
                elif keyword in ["help"]:
                    subject = "help"

        #print(f"Identified subject: {subject}, movie: {movie}")
        print('Please make sure to use the key words to your question, such as,␣
↪movie, year, release, script, etc. ' )
        return subject, movie


    def answer_question(self, query): #The section where the system matches the␣
↪query
        subject, movie = self.identify_subject(query) #taken from the def␣
↪identify_subject

        if subject and movie:
            matching_conversations = [conv for conv in self.corpus if␣
↪isinstance(conv, dict) and movie.lower() in conv.get("movieTitle_x", "").
↪lower()]

            if matching_conversations:
                conversation = matching_conversations[0]
                response = self.generate_response(query, conversation, subject)
                return response
```

```
            else:
                return "I couldn't find any information about that movie."

        elif subject == "releaseYear":
            year = re.search(r'\b\d{4}\b', query)
            if year:
                year = year.group(0)
                matching_movies = set()  # Use a set to store unique movie␣
↪titles
                for conv in self.corpus:
                    if conv["releaseYear"] == year:
                        matching_movies.add(conv["movieTitle_x"])

                if matching_movies:
                    response = f"The following movies were released in {year}:␣
↪{', '.join(matching_movies)}."
                else:
                    response = f"No movies were released in {year}."
                return response

        elif subject == "timdbRating":
            year_match = re.search(r'\b\d{4}\b', query)
            if year_match:
                year = year_match.group(0)
                movies_for_year = [conv for conv in self.corpus if␣
↪isinstance(conv, dict) and conv.get("releaseYear") == year]
                if movies_for_year:
                    highest_rated_movie = max(movies_for_year, key=lambda x:␣
↪float(x.get("imdbRating", 0)))
                    highest_rating = highest_rated_movie.get("imdbRating", 0)
                    movie_title = highest_rated_movie.get("movieTitle_x",␣
↪"Unknown Movie")
                    response = f"The highest rated movie in {year} was␣
↪'{movie_title}' with an IMDb rating of {highest_rating}."
                else:
                    response = f"No movies were released in {year}."
                return response

        elif subject == "limdbRating":
            year_match = re.search(r'\b\d{4}\b', query)
            if year_match:
                year = year_match.group(0)
                movies_for_year = [conv for conv in self.corpus if␣
↪isinstance(conv, dict) and conv.get("releaseYear") == year]
                if movies_for_year:
```

```
                    lowest_rated_movie = min(movies_for_year, key=lambda x:␣
↪float(x.get("imdbRating", 0)))
                    lowest_rating = lowest_rated_movie.get("imdbRating", 0)
                    movie_title = lowest_rated_movie.get("movieTitle_x",␣
↪"Unknown Movie")
                    response = f"The lowest rated movie in {year} was␣
↪'{movie_title}' with an IMDb rating of {lowest_rating}."
                else:
                    response = f"No movies were released in {year}."
                return response

        elif subject == "url" and movie:
            matching_conversations = [conv for conv in self.corpus if␣
↪isinstance(conv, dict) and movie.lower() in conv.get("movieTitle_x", "").
↪lower()]

        elif subject == "genre_1" and movie:
            matching_conversations = [conv for conv in self.corpus if␣
↪isinstance(conv, dict) and movie.lower() in conv.get("movieTitle_x", "").
↪lower()]

            if matching_conversations:
                conversation = matching_conversations[0]
                if "url" in conversation:
                    response = f"I don't have actor info but here is the URL␣
↪for {conversation['movieTitle_x']}: {conversation['url']} for more info."
                else:
                    response = f"Sorry, I don't have that info for␣
↪{conversation['movieTitle_x']}."
            else:
                response = "I couldn't find any information about that movie."

        elif subject == "hi":
            response = f"Hello, how can I answer a movie question for you?"
            return response

        elif subject == "help":
            response = f"I can answer question regarding a movie name, the year␣
↪it was released, the rating score, the character names, and some of their␣
↪lines. I do not have the actor names, but I can provide a website with more␣
↪info on the movie."
            return response

        else:
            return "I can only answer movie-related questions."
```

```
            return "Please make sure to use the words like 'movie', 'year',␣
 ↪'released', 'rating' etc. I'm not great at guessing :("
```

```
<>:69: SyntaxWarning: "is" with a literal. Did you mean "=="?
<>:69: SyntaxWarning: "is" with a literal. Did you mean "=="?
<ipython-input-19-a131c37bd40b>:69: SyntaxWarning: "is" with a literal. Did you
mean "=="?
  elif keyword in ["release", "year"] and subject is 'movieTitle_x':
```

[ ]:
```
#@title 6: Main Chatbot Function-This will show 5 rows of the dataset and then␣
 ↪the input and output chatbot
#Note: The words: movie, year, release, imdb must be present in the input for␣
 ↪the system to understand
def main():
    pre_trained_model_name = "facebook/bart-large-cnn"
    chatbot = Chatbot(merged_data, pre_trained_model_name)

    while True:
        query = input("You: ")
        response = chatbot.answer_question(query)
        print("Chatbot:", response)

if __name__ == "__main__":
    main()
```

```
Loaded movie data:
{'lineID': 'L1045', 'characterID_x': 'u0', 'movieID': 'm0', 'character':
'BIANCA', 'text': 'They do not!\n', 'characterID_y': 'u0', 'characterName':
'BIANCA', 'movieTitle_x': '10 things i hate about you', 'gender': 'f',
'position': '4\n', 'characterID1': 'u10', 'characterID2': 'u11', 'utteranceIDs':
"['L929', 'L930', 'L931', 'L932', 'L933']\n", 'releaseYear': '1999',
'imdbRating': '6.90', 'numVotes': '62847', 'genre_1': 'comedy', 'genre_2':
'romance\n', 'genre_3': None, 'genre_4': None, 'genre_5': None, 'genre_6': None,
'genre_7': None, 'genre_8': None, 'genre_9': None, 'genre_10': None, 'genre_11':
None, 'url': 'http://www.dailyscript.com/scripts/10Things.html\n'}
{'lineID': 'L1045', 'characterID_x': 'u0', 'movieID': 'm0', 'character':
'BIANCA', 'text': 'They do not!\n', 'characterID_y': 'u1', 'characterName':
'BRUCE', 'movieTitle_x': '10 things i hate about you', 'gender': '?',
'position': '?\n', 'characterID1': 'u10', 'characterID2': 'u11', 'utteranceIDs':
"['L929', 'L930', 'L931', 'L932', 'L933']\n", 'releaseYear': '1999',
'imdbRating': '6.90', 'numVotes': '62847', 'genre_1': 'comedy', 'genre_2':
'romance\n', 'genre_3': None, 'genre_4': None, 'genre_5': None, 'genre_6': None,
'genre_7': None, 'genre_8': None, 'genre_9': None, 'genre_10': None, 'genre_11':
None, 'url': 'http://www.dailyscript.com/scripts/10Things.html\n'}
{'lineID': 'L1045', 'characterID_x': 'u0', 'movieID': 'm0', 'character':
'BIANCA', 'text': 'They do not!\n', 'characterID_y': 'u2', 'characterName':
'CAMERON', 'movieTitle_x': '10 things i hate about you', 'gender': 'm',
```

```
'position': '3\n', 'characterID1': 'u10', 'characterID2': 'u11', 'utteranceIDs':
"['L929', 'L930', 'L931', 'L932', 'L933']\n", 'releaseYear': '1999',
'imdbRating': '6.90', 'numVotes': '62847', 'genre_1': 'comedy', 'genre_2':
'romance\n', 'genre_3': None, 'genre_4': None, 'genre_5': None, 'genre_6': None,
'genre_7': None, 'genre_8': None, 'genre_9': None, 'genre_10': None, 'genre_11':
None, 'url': 'http://www.dailyscript.com/scripts/10Things.html\n'}
{'lineID': 'L1045', 'characterID_x': 'u0', 'movieID': 'm0', 'character':
'BIANCA', 'text': 'They do not!\n', 'characterID_y': 'u3', 'characterName':
'CHASTITY', 'movieTitle_x': '10 things i hate about you', 'gender': '?',
'position': '?\n', 'characterID1': 'u10', 'characterID2': 'u11', 'utteranceIDs':
"['L929', 'L930', 'L931', 'L932', 'L933']\n", 'releaseYear': '1999',
'imdbRating': '6.90', 'numVotes': '62847', 'genre_1': 'comedy', 'genre_2':
'romance\n', 'genre_3': None, 'genre_4': None, 'genre_5': None, 'genre_6': None,
'genre_7': None, 'genre_8': None, 'genre_9': None, 'genre_10': None, 'genre_11':
None, 'url': 'http://www.dailyscript.com/scripts/10Things.html\n'}
{'lineID': 'L1045', 'characterID_x': 'u0', 'movieID': 'm0', 'character':
'BIANCA', 'text': 'They do not!\n', 'characterID_y': 'u4', 'characterName':
'JOEY', 'movieTitle_x': '10 things i hate about you', 'gender': 'm', 'position':
'6\n', 'characterID1': 'u10', 'characterID2': 'u11', 'utteranceIDs': "['L929',
'L930', 'L931', 'L932', 'L933']\n", 'releaseYear': '1999', 'imdbRating': '6.90',
'numVotes': '62847', 'genre_1': 'comedy', 'genre_2': 'romance\n', 'genre_3':
None, 'genre_4': None, 'genre_5': None, 'genre_6': None, 'genre_7': None,
'genre_8': None, 'genre_9': None, 'genre_10': None, 'genre_11': None, 'url':
'http://www.dailyscript.com/scripts/10Things.html\n'}
Total movies loaded: 4874548
You: do you have the actors for the movie xxx?
Please make sure to use the key words to your question, such as, movie, year,
release, script, etc.
Chatbot: I don't have actor info but here is the URL for xxx:
http://www.dailyscript.com/scripts/xXx.txt
 for more info.
You: do you have the script for the movie xxx?
Please make sure to use the key words to your question, such as, movie, year,
release, script, etc.
Chatbot: I can only answer questions about release year or IMDB rating.
You: do have the movie xxx script?
Please make sure to use the key words to your question, such as, movie, year,
release, script, etc.
Chatbot: I can only answer questions about release year or IMDB rating.
You: script?
Please make sure to use the key words to your question, such as, movie, year,
release, script, etc.
Chatbot: I can only answer movie-related questions.
You: do you have more inforamtion about the movie xxx?
Please make sure to use the key words to your question, such as, movie, year,
release, script, etc.
Chatbot: I can only answer questions about release year or IMDB rating.
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-20-c5e590dc5482> in <cell line: 12>()
     11
     12 if __name__ == "__main__":
---> 13     main()

<ipython-input-20-c5e590dc5482> in main()
      6
      7     while True:
----> 8         query = input("You: ")
      9         response = chatbot.answer_question(query)
     10         print("Chatbot:", response)

/usr/local/lib/python3.10/dist-packages/ipykernel/kernelbase.py in␣
 ↪raw_input(self, prompt)
    849                 "raw_input was called, but this frontend does not␣
 ↪support input requests."
    850             )
--> 851         return self._input_request(str(prompt),
    852             self._parent_ident,
    853             self._parent_header,

/usr/local/lib/python3.10/dist-packages/ipykernel/kernelbase.py in␣
 ↪_input_request(self, prompt, ident, parent, password)
    893             except KeyboardInterrupt:
    894                 # re-raise KeyboardInterrupt, to truncate traceback
--> 895                 raise KeyboardInterrupt("Interrupted by user") from Non
    896             except Exception as e:
    897                 self.log.warning("Invalid Message:", exc_info=True)

KeyboardInterrupt: Interrupted by user
```