Complete Movie Recommendation System with NLP, ML, and RL

```
# @title Complete Movie Recommendation System with NLP, ML, and RL
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
import random
import ipywidgets as widgets
from IPython.display import display, clear_output
import torch
import torch.nn as nn
import torch.optim as optim
from collections import deque
import gym
from gym import spaces

# Download NLTK resources
nltk.download('stopwords')
nltk.download('wordnet')
# The 'punkt' resource is needed for tokenization, and 'punkt_tab' is a dependency.
# We'll ensure both are downloaded explicitly before use.
# nltk.download('punkt') # Already covered by punkt_tab download
# nltk.download('punkt_tab') # Download the missing resource - doing this before preprocess

# ======================
# 1. Dataset Preparation
# ======================
def create_movie_dataset(size=100):
    """Create synthetic movie dataset with rich features"""
    genres = ['Action', 'Adventure', 'Comedy', 'Drama', 'Sci-Fi',
              'Thriller', 'Romance', 'Horror', 'Fantasy', 'Mystery']
    themes = ['space', 'love', 'revenge', 'friendship', 'war',
              'family', 'technology', 'crime', 'superhero', 'history']
    descriptors = ['epic', 'emotional', 'dark', 'funny', 'suspenseful',
                   'heartwarming', 'mind-bending', 'gritty', 'inspirational']

    plot_options = [
        'the protagonist must save the world',
        'an unlikely hero emerges',
        'secrets from the past are revealed',
        'relationships are tested'
    ]

    movies = []
    for i in range(1, size+1):
        movie_genres = random.sample(genres, random.randint(1, 3))
        theme = random.choice(themes)
        desc = random.choice(descriptors)

        title = f"{desc.capitalize()} {theme.capitalize()}"
        if random.random() > 0.7:
            title = f"The {title}"

        plot = f"A {desc} story about {theme} where {random.choice(plot_options)}"

        movies.append({
            'movie_id': i,
            'title': title,
            'genre': ', '.join(movie_genres),
            'plot': plot,
            'year': random.randint(1990, 2023),
            'rating': round(random.uniform(3.0, 5.0), 1),
            'popularity': random.randint(1, 100)
        })
    return pd.DataFrame(movies)

movies_df = create_movie_dataset(150)
print("Dataset created with 150 movies")

# ======================
# 2. NLP Processing
# ======================
lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))

def preprocess_text(text):
```

What can I help you build?

```
        """Clean and preprocess text using NLP techniques"""
        # Convert to lowercase
        text = text.lower()
        # Remove special characters
        text = re.sub(r'[^a-zA-Z0-9\s]', '', text)
        # Download punkt_tab resource just before tokenizing
        try:
            nltk.data.find('tokenizers/punkt_tab')
        except LookupError:
            nltk.download('punkt_tab')
        # Tokenize
        words = nltk.word_tokenize(text)
        # Remove stopwords and lemmatize
        words = [lemmatizer.lemmatize(word) for word in words if word not in stop_words]
        return ' '.join(words)

    # Combine text features and preprocess
    movies_df['processed_text'] = (movies_df['plot'] + ' ' +
                                   movies_df['genre'] + ' ' +
                                   movies_df['title'])
    movies_df['processed_text'] = movies_df['processed_text'].apply(preprocess_text)

    # TF-IDF Vectorization (NLP feature extraction)
    tfidf = TfidfVectorizer(max_features=2000)
    tfidf_matrix = tfidf.fit_transform(movies_df['processed_text'])
    content_sim = cosine_similarity(tfidf_matrix)

    print("NLP processing completed!")
    print(f"TF-IDF matrix shape: {tfidf_matrix.shape}")

    # ======================
    # 3. Reinforcement Learning
    # ======================
    class MovieRecEnv(gym.Env):
        def __init__(self, df, similarity_matrix):
            super(MovieRecEnv, self).__init__()
            self.df = df
            self.sim_matrix = similarity_matrix
            self.current_movie = None
            self.recommended = []
            self.feedback = []

            # Action space: 0=content-based, 1=genre-based, 2=popularity-based
            self.action_space = spaces.Discrete(3)

            # Observation space (TF-IDF vector)
            self.observation_space = spaces.Box(low=0, high=1,
                                                shape=(tfidf_matrix.shape[1],),
                                                dtype=np.float32)

        def reset(self):
            self.current_movie = self.df.sample(1).iloc[0]
            self.recommended = []
            self.feedback = []
            return self._get_obs()

        def _get_obs(self):
            idx = self.df[self.df['movie_id'] == self.current_movie['movie_id']].index[0]
            return self.sim_matrix[idx]

        def step(self, action):
            # Get recommendations based on action
            if action == 0:  # Content-based
                idx = self.df[self.df['movie_id'] == self.current_movie['movie_id']].index[0]
                sim_scores = list(enumerate(self.sim_matrix[idx]))
                sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
                movie_indices = [i[0] for i in sim_scores[1:6]]  # Top 5 similar
                self.recommended = self.df.iloc[movie_indices]
            elif action == 1:  # Genre-based
                genre = self.current_movie['genre'].split(', ')[0]
                genre_movies = self.df[self.df['genre'].str.contains(genre)]
                self.recommended = genre_movies.sample(min(5, len(genre_movies)))
            else:  # Popularity-based
                self.recommended = self.df.sort_values('popularity', ascending=False).head(5)

            # Simulate user feedback (higher rating = more likely positive)
            self.feedback = [1 if random.random() < 0.3 + (m['rating']-3)*0.15 else 0
                            for _, m in self.recommended.iterrows()]

            # Calculate reward
            reward = sum(self.feedback)
            # Move to new movie
            self.current_movie = self.df.sample(1).iloc[0]

            return self._get_obs(), reward, False, {}

    env = MovieRecEnv(movies_df, content_sim)
    print("RL environment created!")

    # Deep Q-Network
    class DQN(nn.Module):
        def __init__(self, input_size, output_size):
            super(DQN, self).__init__()
            self.fc1 = nn.Linear(input_size, 128)
            self.fc2 = nn.Linear(128, 64)
            self.fc3 = nn.Linear(64, output_size)

        def forward(self, x):
            x = torch.relu(self.fc1(x))
            x = torch.relu(self.fc2(x))
            return self.fc3(x)

    class DQNAgent:
        def __init__(self, state_size, action_size):
            self.state_size = state_size
            self.action_size = action_size
            self.memory = deque(maxlen=2000)
            self.gamma = 0.95
            self.epsilon = 1.0
            self.epsilon_min = 0.01
            self.epsilon_decay = 0.995
            self.model = DQN(state_size, action_size)
            self.optimizer = optim.Adam(self.model.parameters(), lr=0.001)
            self.criterion = nn.MSELoss()

        def remember(self, state, action, reward, next_state):
            self.memory.append((state, action, reward, next_state))

        def act(self, state):
            if random.random() <= self.epsilon:
                return random.randrange(self.action_size)
            state = torch.FloatTensor(state)
            act_values = self.model(state)
            return torch.argmax(act_values).item()

        def replay(self, batch_size):
            if len(self.memory) < batch_size:
                return
            minibatch = random.sample(self.memory, batch_size)
            for state, action, reward, next_state in minibatch:
                state = torch.FloatTensor(state)
                next_state = torch.FloatTensor(next_state)
                target = reward + self.gamma * torch.max(self.model(next_state))
                current = self.model(state)[action]
                loss = self.criterion(current, target.detach())
                self.optimizer.zero_grad()
                loss.backward()
                self.optimizer.step()
            if self.epsilon > self.epsilon_min:
                self.epsilon *= self.epsilon_decay

    # Initialize RL agent
    state_size = tfidf_matrix.shape[1]
    action_size = 3
    agent = DQNAgent(state_size, action_size)

    # ======================
    # 4. Chatbot Interface
    # ======================
    def get_recommendations(movie_title, method='content'):
        """Get recommendations based on specified method"""
        try:
            if movie_title not in movies_df['title'].values:
                return None

            if method == 'content':
                idx = movies_df[movies_df['title'] == movie_title].index[0]
                sim_scores = list(enumerate(content_sim[idx]))
                sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
                movie_indices = [i[0] for i in sim_scores[1:6]]  # Top 5 similar
                return movies_df.iloc[movie_indices]
            elif method == 'genre':
                movie_genre = movies_df[movies_df['title'] == movie_title]['genre'].values[0].split(', ')[0]
                genre_movies = movies_df[movies_df['genre'].str.contains(movie_genre)]
                return genre_movies[genre_movies['title'] != movie_title].sample(min(5, len(genre_movies)))
            else:  # popularity
                return movies_df[movies_df['title'] != movie_title].sort_values('popularity', ascending=False).head(5)
        except:
            return None

    # Create widgets
    output = widgets.Output()
    input_box = widgets.Text(
        placeholder='Type "Recommend movies like [title]"',
        description='You:',
        layout=widgets.Layout(width='80%')
    )
    send_button = widgets.Button(
        description="Send",
        layout=widgets.Layout(width='20%')
    )

    def on_click(b):
        with output:
            user_input = input_box.value
            input_box.value = ''

            if not user_input.strip():
                return

            if user_input.lower() in ['quit', 'exit']:
                print("Chatbot: Goodbye! Happy watching! 🎬")
                return

            # Process recommendation request
            if any(w in user_input.lower() for w in ['recommend', 'suggest', 'like']):
                # Find mentioned movie
                movie_title = None
                for title in movies_df['title']:
                    if title.lower() in user_input.lower():
                        movie_title = title
                        break

                if movie_title:
                    # Let RL agent choose method
                    movie_idx = movies_df[movies_df['title'] == movie_title].index[0]
                    state = content_sim[movie_idx]
                    method_idx = agent.act(state)
                    methods = ['content', 'genre', 'popularity']
                    method = methods[method_idx]

                    print(f"\nChatbot: Finding movies similar to '{movie_title}'...")
                    recs = get_recommendations(movie_title, method)

                    if recs is not None and not recs.empty:
                        print(f"\nRecommended Movies ({method}-based):")
                        for _, row in recs.iterrows():
                            print(f" - {row['title']} ({row['year']}) | {row['genre']} | ⭐{row['rating']}")

                        # Simulate feedback and train RL agent
                        feedback = [1 if r > 4.0 else 0 for r in recs['rating'].values]
                        reward = sum(feedback)
                        next_state = content_sim[recs.index[0]]
                        agent.remember(state, method_idx, reward, next_state)
                        agent.replay(32)
                    else:
                        print("\nSorry, I couldn't find recommendations for that movie.")
                else:
                    print("\nPlease mention a movie from our database. Try:")
                    print(" - 'Recommend movies like The Dark Knight'")
                    print("\nSome movies you could try:")
                    for title in movies_df['title'].sample(5):
                        print(f" - {title}")

            elif 'list movies' in user_input.lower():
                print("\nSome movies in our database:")
                for _, row in movies_df.sample(10).iterrows():
                    print(f" - {row['title']} ({row['year']})")

            elif any(w in user_input.lower() for w in ['hello', 'hi', 'hey']):
                print("\nHi! I'm a smart movie recommendation chatbot.")
                print("I use AI to suggest movies you'll love!")
                print("Try: 'Recommend movies like Inception'")

            else:
                print("\nI can recommend movies based on what you like. Try:")
                print(" - 'Recommend movies like The Matrix'")
                print(" - 'List movies'")

    send_button.on_click(on_click)

    # Handle Enter key
    def handle_enter(sender):
        on_click(sender)

    input_box.on_submit(handle_enter)

    # Display the chatbot
    display(widgets.VBox([
        widgets.HTML("<h2>AI Movie Recommendation Chatbot</h2>"),
        widgets.HTML("<p>I use NLP, Machine Learning, and Reinforcement Learning!</p>"),
        output,
        widgets.HBox([input_box, send_button])
    ]))

    print("\nChatbot ready! Try asking:")
    print(" - 'Recommend movies like Epic Space'")
    print(" - 'List movies'")
    print(" - 'Exit' to quit")
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
Dataset created with 150 movies
NLP processing completed!
TF-IDF matrix shape: (150, 42)
RL environment created!
/usr/local/lib/python3.11/dist-packages/ipywidgets/widgets/widget_string.py:107: DeprecationWarning: on_submit is deprecated. Instea
  warnings.warn("on_submit is deprecated. Instead, set the .continuous_update attribute to False and observe the value changing with
```

AI Movie Recommendation Chatbot

I use NLP, Machine Learning, and Reinforcement Learning!

```
Please mention a movie from our database. Try:
 - 'Recommend movies like The Dark Knight'

Some movies you could try:
 - Inspirational Love
 - Mind-bending Friendship
 - Epic Superhero
 - Epic Space
 - The Inspirational Family
/usr/local/lib/python3.11/dist-packages/ipywidgets/widgets/widget_output.py:111: DeprecationWarning: Kernel._parent_header is
deprecated in ipykernel 6. Use .get_parent()
  if ip and hasattr(ip, 'kernel') and hasattr(ip.kernel, '_parent_header'):
```

You: Type "Recommend movies like [title]" Send

```
Chatbot ready! Try asking:
 - 'Recommend movies like Epic Space'
 - 'List movies'
```