Project Report: Movie Recommendation Chatbot

Introduction

The Movie Recommendation Chatbot is an interactive web application designed to provide movie recommendations based on user queries. Built using Streamlit, Cohere, and Pinecone, this chatbot searches through a dataset of Netflix titles to find relevant suggestions for users. The goal of this project is to create an engaging and user-friendly tool that leverages natural language processing and similarity search to enhance user experience in discovering movies.

Features

- **Interactive Interface:** Users can input their queries and receive movie recommendations in real-time.
- **Natural Language Processing:** Utilizes Cohere for understanding and processing user inputs.
- **Similarity Search:** Uses Pinecone for efficient and accurate similarity searches within the dataset.
- **Netflix-Themed Design:** The UI/UX is styled to resemble the Netflix website, providing a familiar and visually appealing experience.

Dataset

The dataset used for this project is a CSV file containing titles of Netflix movies and shows. This dataset includes various fields such as title, genre, director, cast, country, release year, rating, and description.

Building the Chatbot

Step 1: Setting Up the Environment

First, we set up the project environment by installing the necessary dependencies and configuring API keys for Cohere and Pinecone.

Step 2: Loading the Dataset

We load the `netflix_title.csv` dataset using the `CSVLoader` from `langchain_community.document_loaders`.

```
python
Copy code
loader = CSVLoader("netflix_title.csv")
```

```
documents = loader.load()
```

Step 3: Preprocessing the Data

The dataset is split into chunks using `CharacterTextSplitter` to facilitate efficient searching.

```
python
Copy code
text_splitter = CharacterTextSplitter(chunk_size=1000, chunk_overlap=50)
docs = text_splitter.split_documents(documents)
```

Step 4: Setting Up Cohere and Pinecone

We initialize Cohere for natural language processing and Pinecone for similarity search.

```
python
Copy code
llm = Cohere()
embeddings = CohereEmbeddings()
pc = Pinecone(api_key=os.environ["PINECONE_API_KEY"])
```

Step 5: Creating or Using an Existing Pinecone Index

We create a Pinecone index if it doesn't already exist and load our documents into this index.

```
python
Copy code
index_name = "rag-test"
existing_indexes = [index_info["name"] for index_info in pc.list_indexes()]

if index_name not in existing_indexes:
    pc.create_index(
        name=index_name,
        dimension=4096,
        metric="cosine",
        spec=ServerlessSpec(cloud="aws", region="us-east-1"),
    )
    while not pc.describe_index(index_name).status["ready"]:
        time.sleep(1)

index = pc.Index(index_name)
docsearch = PineconeVectorStore.from_documents(docs, embeddings,
index_name=index_name)
```

Step 6: Implementing the Chatbot Response Function

The `chatbot_response` function takes a user query, performs a similarity search, and generates an appropriate response.

```
python
Copy code
```

```
chain = load_qa_chain(llm, chain_type="stuff")

def chatbot_response(query):
    docs = docsearch.similarity_search(query, k=3)
    answer = chain.run(input_documents=docs, question=query)
    return answer
```

Step 7: Building the Streamlit Interface

We use Streamlit to create a user-friendly web interface. The interface includes an input box for user queries and a display area for chat messages.

```
python
Copy code
st.set_page_config(page_title="Movie Recommendation Chatbot",
page_icon=":clapper:", layout="centered")
st.markdown(
    """
    <style>
    .main {
        background: linear-gradient(rgba(0, 0, 0, 0.5), rgba(0, 0, 0, 0.5)),
url('netflix_logo.png') no-repeat center center fixed;
        background-size: cover;
        color: #e5e5e5;
        font-family: 'Helvetica Neue', sans-serif;
    }
    .chat-container {
        width: 100%;
        max-width: 700px;
        margin: auto;
        background-color: rgba(20, 20, 20, 0.8);
        border-radius: 8px;
        padding: 20px;
        box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
    }
    .user-message, .bot-response {
        padding: 10px;
        border-radius: 5px;
        margin-bottom: 10px;
    }
    .user-message {
        background-color: #e50914;
        color: white;
        text-align: right;
    }
    .bot-response {
        background-color: #333;
        color: #e5e5e5;
        text-align: left;
    }
    .stTextInput, .stButton {
        width: 100%;
    }
    .stTextInput>div>input {
        background-color: #333;
        color: #e5e5e5;
        border: 1px solid #e50914;
    }
    .stTextInput>label {
        color: white;
    }
    .stButton>button {
        background-color: #e50914;
        color: white;
        border: none;
        padding: 10px;
        border-radius: 5px;
        cursor: pointer;
        font-size: 16px;
        font-weight: bold;
    }
    .stButton>button:hover {
        background-color: #f40612;
    }
    h1 {
        color: white;
    }
    </style>
    """,
    unsafe_allow_html=True
)

st.title("Movie Recommendation Chatbot :clapper:")

if "messages" not in st.session_state:
    st.session_state.messages = []

with st.form("chat_form", clear_on_submit=True):
    user_input = st.text_input("Type your message:")
    submitted = st.form_submit_button("Send")

if submitted and user_input:
    st.session_state.messages.append({"role": "user", "content": user_input})
    response = chatbot_response(user_input)
    st.session_state.messages.append({"role": "bot", "content": response})

for message in st.session_state.messages:
    if message["role"] == "user":
        st.markdown(f"<div class='user-message'>{message['content']}</div>",
unsafe_allow_html=True)
    else:
        st.markdown(f"<div class='bot-response'>{message['content']}</div>",
unsafe_allow_html=True)
```

Step 8: Running the Application

To run the application, use the following command:

```
bash
Copy code
```

```
streamlit run app.py
```

Output and User Interaction

Upon starting the application, users are presented with a chat interface where they can type in their movie-related queries. The chatbot processes these queries and returns relevant movie recommendations. The responses are displayed in a conversational format, with the user's messages and bot's responses styled distinctly.

Example Interaction

User: Recommend a good action movie.

Bot: Based on your interest in action movies, you might enjoy "The Dark Knight" directed by Christopher Nolan.

Conclusion

The Movie Recommendation Chatbot successfully integrates natural language processing and similarity search to provide personalized movie recommendations. The use of Streamlit for the frontend ensures an interactive and user-friendly experience. This project demonstrates the potential of combining advanced NLP techniques with efficient data retrieval methods to enhance user engagement and satisfaction.