Book Recommendation System Using Streamlit and Sentence Transformers

Title:

Approach and Reasoning Behind Using the `'sentence-transformers/all-MiniLM-L6-v2'` Model in a Book Recommendation System

---

Introduction:

This document outlines the approach and reasoning behind the selection of the `'sentence-transformers/all-MiniLM-L6-v2'` model for building a book recommendation system using Streamlit. The primary goal is to provide users with accurate and relevant book recommendations based on their genre preferences.

Approach:

1. **Data Loading and Preparation:**
   - Load the book dataset containing various genres and ratings using Pandas.
   - Ensure efficient data handling and preprocessing to facilitate smooth user interactions.
2. **Top 100 Books Identification:**
   - Filter books based on the user-specified genre.
   - Select the top 100 books with the highest average ratings within the chosen genre.
3. **Top 10 Books Extraction:**
   - From the top 100 books, further narrow down to the top 10 based on their average ratings.
4. **Book Recommendation Using Sentence Transformers:**
   - Use the `'sentence-transformers/all-MiniLM-L6-v2'` model to encode book titles and user prompts.
   - Compute cosine similarity scores to determine the best book recommendation from the top 10.

Reasoning Behind Model Selection:

1. **BERT-Based Semantic Understanding:**

   - The `'sentence-transformers/all-MiniLM-L6-v2'` model is based on BERT architecture, which excels at capturing semantic relationships between text inputs. This ensures accurate comparison between user prompts and book titles for relevant recommendations.
2. **High-Quality Embeddings:**
   - The model generates high-quality embeddings that effectively represent the semantic meaning of book titles and user queries. This quality directly impacts the accuracy and relevance of book recommendations.
3. **Efficiency and Real-Time Application:**
   - Designed for computational efficiency, the MiniLM-L6 variant allows the recommendation engine to operate in real-time, providing quick responses to user queries. This is essential for maintaining a responsive and engaging user experience in the Streamlit application.
4. **Scalability:**
   - The model's scalability ensures it can handle large datasets and multiple concurrent user interactions. This capability is crucial as the user base grows, ensuring the recommendation engine remains robust and efficient.
5. **Community Support and Updates:**
   - Being part of the Sentence Transformers library, the model benefits from ongoing community support, updates, and refinements. This ensures the model stays aligned with current best practices in natural language processing (NLP) and maintains optimal performance.

Conclusion:

The selection of the `'sentence-transformers/all-MiniLM-L6-v2'` model for the book recommendation system is driven by its ability to leverage advanced BERT-based embeddings for semantic understanding, efficiency in real-time applications, scalability, and robust community support. These factors collectively enhance the effectiveness and user experience of the book recommendation system, providing users with accurate and relevant book suggestions based on their preferences.

Implementation:

Below is the core code implemented in the `app.py` file for this project:

```
!pip install streamlit pandas transformers torch pyngrok fuzzywuzzy
sentence_transformers

import pandas as pd
import streamlit as st
from sentence_transformers import SentenceTransformer, util

# Load the dataset
@st.cache_data
def load_data(file_path: str):
    try:
        df = pd.read_csv(file_path)
        st.write("Data loaded successfully!")
        return df
    except Exception as e:
        st.error(f"Error loading data: {e}")
        return pd.DataFrame()

# Function to get top 100 books in a given genre
def get_top_100_books(genre: str, books_df: pd.DataFrame):
    try:
        genre_books = books_df[books_df['Genres'].str.contains(genre,
case=False, na=False)]
        if genre_books.empty:
            st.write(f"No books found for genre: {genre}")
            return pd.DataFrame()
        top_100_books = genre_books.nlargest(100, 'Avg_Rating')
        return top_100_books
    except Exception as e:
        st.error(f"Error getting top 100 books: {e}")
        return pd.DataFrame()

# Function to get top 10 books from the top 100
def get_top_10_books(books: pd.DataFrame):
    try:
        if books.empty:
            st.write("No books available to select top 10 from.")
            return pd.DataFrame()
        top_10_books = books.nlargest(10, 'Avg_Rating')
        return top_10_books
    except Exception as e:
        st.error(f"Error getting top 10 books: {e}")
        return pd.DataFrame()

# Function to recommend the best book from the top 10 using BERT-based
similarity
def recommend_best_book_bert(top_10_books: pd.DataFrame):
    try:
        if top_10_books.empty:
            st.error("No books to recommend from.")
            return None

        model =
SentenceTransformer('sentence-transformers/all-MiniLM-L6-v2')

        books_list = top_10_books.to_dict('records')
        book_titles = [book['Book'] for book in books_list]

        # Prompt for BERT model
        prompt = "Which book from the following list would you
recommend? " + ", ".join(book_titles)

        # Embed the prompt and book titles
        prompt_embedding = model.encode(prompt,
convert_to_tensor=True)
        book_embeddings = model.encode(book_titles,
convert_to_tensor=True)

        # Compute cosine similarities
        cosine_scores = util.pytorch_cos_sim(prompt_embedding,
book_embeddings)[0]

        # Find the book with the highest similarity score
        highest_score_idx = cosine_scores.argmax().item()
        recommended_book = books_list[highest_score_idx]

        return recommended_book

    except Exception as e:
        st.error(f"An error occurred: {e}")
        return None

# LanGraph-like logic structure to simulate workflow
class LanGraphAgent:
    def __init__(self, genre: str):
        self.genre = genre
        self.top_100_books = None
        self.top_10_books = None
        self.recommended_book = None

    def execute(self):
        self.find_top_100_books()
        self.find_top_10_books()
        self.recommend_book()
        self.conclude_task()

    def find_top_100_books(self):
        self.top_100_books = get_top_100_books(self.genre, books_df)
        st.write("Top 100 books found:")
        st.write(self.top_100_books)

    def find_top_10_books(self):
        self.top_10_books = get_top_10_books(self.top_100_books)
        st.write("Top 10 books found:")
        st.write(self.top_10_books)

    def recommend_book(self):
        try:
            st.write("Attempting to recommend a book...")
            self.recommended_book =
recommend_best_book_bert(self.top_10_books)
            if self.recommended_book:
                st.write(f"Recommended book: {self.recommended_book}")
            else:
                st.write("No book could be recommended.")
        except Exception as e:
            st.error(f"An error occurred while recommending a book:
{e}")
```

```
    def conclude_task(self):
        st.write("Thank you for using our book recommendation
service!")
        return self.recommended_book

# Load the dataset
books_df = load_data('goodreads_data.csv')

# Streamlit application
st.title("Book Recommendation Agent")

genre = st.text_input("Enter the genre you are interested in:", "")

if genre:
    agent = LanGraphAgent(genre)
    agent.execute()

    if agent.recommended_book:
        st.write("Recommended Book:")
        st.write(f"*Title*: {agent.recommended_book['Book']}")
        st.write(f"*Genre*: {genre}")
        st.write(f"*Rating*:
{agent.recommended_book['Avg_Rating']:.2f}")
    else:
        st.write("Could not determine a single recommended book.
Please try again.")

from google.colab import userdata
ngrok_auth_token = userdata.get('NGROK_AUTH_TOKEN')  # Retrieve your
Ngrok auth token

!ngrok authtoken {ngrok_auth_token}

# Run Streamlit with Ngrok
from pyngrok import ngrok

# Set up Ngrok
```

```
public_url = ngrok.connect(8501)
print(f"Streamlit app running at: {public_url}")

# Run the Streamlit app
!streamlit run app.py --server.port 8501
```

References:

- Sentence Transformers Documentation
- Streamlit Documentation
- Pandas Documentation