Book Recommendation System

Using Streamlit and Sentence Transformers

Susanta Baidya

Master's in AI and ML

GitHub: https://github.com/Susanta2102

July 9, 2024

Contents

1 Title

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

2 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.

3 Approach

3.1 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.

3.2 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.

3.3 Top 10 Books Extraction

- From the top 100 books, further narrow down to the top 10 based on their average ratings.

3.4 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.

4 Reasoning Behind Model Selection

4.1 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.

4.2 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.

4.3 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.

4.4 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.

4.5 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.

5 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.

6 Implementation

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

```
!pip install streamlit pandas transformers torch pyngrok fuzzywuzzy
    python-Levenshtein

import streamlit as st
import pandas as pd
from transformers import AutoModel, AutoTokenizer
from torch.nn import functional as F
from fuzzywuzzy import fuzz

@st.cache
def load_data(file_path):
    return pd.read_csv(file_path)

def get_top_100_books(genre, df):
    genre_books = df[df['Genre'].str.contains(genre, case=False, na=
    False)]
    top_100 = genre_books.nlargest(100, 'Avg_Rating')
    return top_100

def get_top_10_books(top_100_df):
    top_10 = top_100_df.nlargest(10, 'Avg_Rating')
    return top_10

def recommend_best_book_bert(books_list):
    try:
        model_name = "sentence-transformers/all-MiniLM-L6-v2"
        model = AutoModel.from_pretrained(model_name)
        tokenizer = AutoTokenizer.from_pretrained(model_name)
        book_titles = [book['Book'] for book in books_list]
        prompt = "Which book from the following list would you
    recommend? " + ", ".join(book_titles)
        prompt_embedding = model.encode(prompt, convert_to_tensor=True)
        book_embeddings = model.encode(book_titles, convert_to_tensor=
    True)
        cosine_scores = F.cosine_similarity(prompt_embedding,
    book_embeddings)
        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

class LanGraphAgent:
    def __init__(self, genre):
        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

books_df = load_data('goodreads_data.csv')

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')
!ngrok authtoken {ngrok_auth_token}

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

!streamlit run app.py --server.port 8501
```

References

- Sentence Transformers Documentation
- Streamlit Documentation
- Pandas Documentation