LyricLink

Song Recommendation System

Prashant Singh | Technical Project | 11/04/2024

Contents

Introduction

Welcome to our Song Recommender project!

Imagine you're scrolling through your favourite music streaming platform, searching for new songs to listen to. Sometimes, it can be overwhelming to find the perfect song that matches your mood or taste. That's where our Song Recommender comes in!

Our project is like having a personal music assistant that helps you discover new songs based on your preferences. Whether you're looking for recommendations similar to a specific song you love or just want to explore random songs, our Song Recommender has you covered.

Here's how it works:

- **Search for Songs**: Have a favourite song in mind? Simply type its name into our search bar, and our system will recommend similar songs that you might enjoy.
- **Browse Random Songs**: Feeling adventurous? Explore our collection of random songs to discover hidden gems and broaden your musical horizons.

Our project consists of two main parts:

- **Backend**: The backend, powered by Python and advanced machine learning techniques, analyzes the lyrics of songs to determine their similarity and provide accurate recommendations.
- **Frontend**: This is the user interface where you interact with our Song Recommender. Built using React.js and TailwindCSS, it provides a sleek and intuitive experience for discovering new music.

The best part? Our entire project is packaged into easy-to-use Docker containers, making it simple to deploy and use on any device.

So, whether you're a music enthusiast looking for your next favourite song or just someone who enjoys exploring new music, our Song Recommender is here to help you find the perfect tune for every moment.

Let's embark on a musical journey together!

For User guide and Installation guide, refer to UseMe.pdf and InstallMe.pdf respectively.

Project Description

The Song Recommender project is an innovative application designed to provide users with personalized song recommendations based on their musical preferences. Leveraging advanced machine learning techniques and modern web technologies, our project offers an intuitive interface for discovering new music and expanding one's playlist.

KEY COMPONENTS:

1. ### Backend:

    - The backend of our project is powered by Python and utilizes the FastAPI framework along with the Uvicorn ASGI server.
    - One of the core functionalities of the backend is its recommendation engine, which employs unsupervised learning algorithms to analyze song lyrics and identify patterns of similarity between songs.
    - Using natural language processing (NLP) techniques, the backend processes large datasets of song lyrics to extract meaningful features and construct a comprehensive similarity matrix.
    - The backend exposes RESTful APIs that allow the frontend to communicate with the recommendation engine, enabling seamless integration and real-time updates of song recommendations.

2. ### Frontend:

    - The frontend of our project is built using React.js, a popular JavaScript library for building user interfaces, and styled with TailwindCSS for responsive and aesthetically pleasing designs.
    - The frontend interface provides users with a sleek and intuitive platform for interacting with the song recommender system.
    - Users can search for specific songs by name or artist, browse through random selections of songs, and view personalized recommendations based on their input.

3. ### Dockerization:

    - To ensure easy deployment and scalability, our project is containerized using Docker technology.
    - Docker containers are employed to encapsulate both the backend and frontend components, allowing for consistent development, testing, and deployment environments across different platforms.
    - The use of Docker simplifies the deployment process and minimizes dependencies, making it easier to maintain and manage the project in production environments.

PROJECT WORKFLOW:

1. **Data Collection and Pre-processing:**

    - The project begins with the collection of song data, from Kaggle.

- The collected data is pre-processed to clean and normalize the text, remove noise, and extract relevant features for subsequent analysis.

2. Similarity Calculation:
    - The model calculates similarity between songs using techniques such as TF-IDF vectorization and cosine similarity.
    - By comparing the lyrics of different songs, the model identifies songs that are most similar to each other and recommends them to users based on their preferences.
3. **Backend Development:**
    - The backend infrastructure is developed using the FastAPI framework, which provides a high-performance and asynchronous web server for handling API requests.
    - Endpoints are defined to support different functionalities, such as searching for songs, retrieving recommendations, and serving static assets to the frontend.
4. **Frontend Implementation:**
    - The frontend user interface is designed and implemented using React.js and TailwindCSS, with a focus on usability, responsiveness, and visual appeal.
    - Components are created to represent different elements of the application, such as search bars, song cards, and navigation menus.
    - API calls are made from the frontend to the backend endpoints to fetch and display song data, recommendations, and search results in real-time.
5. **Integration and Testing:**
    - The backend and frontend components are integrated to create a cohesive and functional application.

PROJECT STRUCTURE

This project employs a separation of concerns by dividing the codebase into a backend and frontend:

- **python_backend/** (Backend code)
    - **ml_recommend/** (Machine Learning Recommendation logic)
        - **recommender.ipynb:** Jupyter Notebook file containing the core recommendation logic.
        - **spotify_millsongdata.csv:** CSV file containing song data.
    - **env/** (Virtual environment directory) - This directory is ignored by Git and should not be committed.
    - **pycache/** (Cache directory) - This directory is automatically generated by Python and should not be committed.
    - **Dockerfile:** Defines instructions for building a Docker image for the backend.

- **tempTestFile.py:** (Optional) A temporary test script.
- **requirements.txt:** Text file containing dependency requirements for the backend project.
- **main.py:** The main entry point for the backend application.
- **data.pkl:** A pickled data file used by the backend.
- **Song.py:** Python class definition for representing song data.
- **similarity_dict.pkl:** A pickled dictionary used for storing song similarities.

- **frontend/** (Frontend code)
  - **node_modules/** (Dependency directory) - This directory is typically ignored by Git and should not be committed. It's automatically generated by npm/yarn and contains downloaded project dependencies.
  - **src/** (Source code directory)
    - **index.js:** Main JavaScript entry point for the frontend application.
    - **index.css:** Main CSS stylesheet for the frontend application.
    - **components/** (Reusable UI components) - This directory contains reusable UI components for building the frontend interface.
    - **pages/** - This directory contains the various pages used in the project.
    - **App.js:** Main React component for the application.
    - **App.css:** CSS styles specific to the App component.
  - **public/** (Public assets directory)
    - **robots.txt:** File instructing search engines how to crawl the website.
    - **manifest.json:** Manifest file for web applications or Progressive Web Apps .
    - **logo512.png** & **logo192.png:** App icons in different sizes.
    - **index.html:** Main HTML file for the frontend application.
    - **favicon.ico:** Favicon for the website.
  - **tailwind.config.js:** Configuration file for Tailwind CSS, a utility-first CSS.
  - **package-lock.json:** Lock file generated by npm/yarn to ensure consistent dependency versions.
  - **package.json:** File containing project metadata, dependencies, and scripts for the frontend.
  - **Dockerfile:** Defines instructions for building a Docker image for the frontend.

Credits

- This project was developed by Prashant Singh. I would like to thank the creators and maintainers of the following technologies and libraries used in this project:
    - FastAPI
    - React.js
    - scikit-learn
    - NLTK
    - Docker
- Special thanks to technical director and hiring staff at JTP Ltd. Co. for their valuable review and comments.

References

- For more information on the technologies and libraries used in this project, refer to the following documentation:
    - FastAPI Documentation
    - React.js Documentation
    - scikit-learn Documentation
    - NLTK Documentation
    - Docker Documentation
- To explore the dataset, follow the given link
    - Spotify Million Song Dataset