Development of a Personalized Music Streaming Interface

Victor Regly

University of Lausanne
Advanced Programming
Professor S. Scheidegger

30th of August, 2024

Overview

- Personalized playlist generation system
- Dynamic user profiles based on interactions
- Content-based recommendation engine
- Feedback loop for continuous improvement
- Streamlit-based user interface

Research Question

Research Question:

- How can a personalized playlist generation system effectively adapt to individual user preferences in real-time using dynamic profiles and feedback loops?

Objectives

Project Objectives:

- Design a system that captures and updates user preferences
- Implement a recommendation engine based on content similarity
- Develop a user-friendly interface for managing playlists and recommendations
- Integrate feedback loops to continually refine the recommendation accuracy

System Architecture Overview

- Dynamic User Profiles: Captures and updates preferences based on user interactions
- Recommendation Engine: Uses content-based filtering to suggest songs that match user preferences
- Feedback Loop: Refines recommendations based on real-time user feedback

```
AP_capstone/
│
├── data/
│   ├── dataset1.csv          # primary dataset with song features
│   ├── users.csv             # dataset for storing user credentials
│   └── playlists.csv         # dataset for storing user playlists
│
├── myenv/                    # virtual environment folder
│
├── pages/                    # Folder for additional pages
│   ├── welcome.py            # page for the welcome screen
│   ├── accueil.py            #  page for the login screen
│   └── register.py           #  page for user registration
│
├── utils/                    # Utility functions and modules
│   ├── playlist_management.py # managing playlists
│   └── user_management.py    # managing users
│
├── app.py                    # Main entry point
│
├── requirements.txt          # List of Python packages
│
│
└── README.md                 # explaining the project
```

Methodology

Feature Representation:

$$x_i = [f_1, f_2, \ldots, f_n]$$

Scaling:

$$z_j = \frac{f_j - \mu_j}{\sigma_j}$$

User Profile Representation:

$$c_u = \frac{1}{m} \sum_{i=1}^{m} z_i$$

Recommendation Algorithm:

$$\text{Recommend } \arg \min_{j \notin \text{playlist}} d(c_u, z_j)$$

Technology Stack

- **Python:** Core programming language used for developing the system.
- **Pandas:** For data manipulation and processing tasks.
- **Scikit-learn:** Implemented for feature scaling and nearest neighbors algorithm.
- **Streamlit:** Used to build the interactive user interface.
- **Git:** Version control for tracking code changes and collaboration.

Implementation Process - Part 1

1. **Setting Up the Environment:**
   - Install Python and set up a virtual environment.
   - Install necessary libraries including Pandas, Scikit-learn, Streamlit.
   - Initialize a Git repository for version control.
2. **Developing Core Modules:**
   - Implement user authentication and playlist management modules.
   - Develop the recommendation engine using content-based filtering techniques.

Implementation Process - Part 2

3. **Building the User Interface:**
   - Use Streamlit to create a web-based interface.
   - Design intuitive navigation for login, playlist management, and exploring recommendations.
4. **Integrating the Feedback Loop:**
   - Allow users to interact with recommendations and adjust preferences in real-time.
   - Implement mechanisms to update user profiles based on interactions.

User Interface - Login and Registration

- Users can log in or create an account using simple forms.
- Input validation ensures the correctness and security of user data.

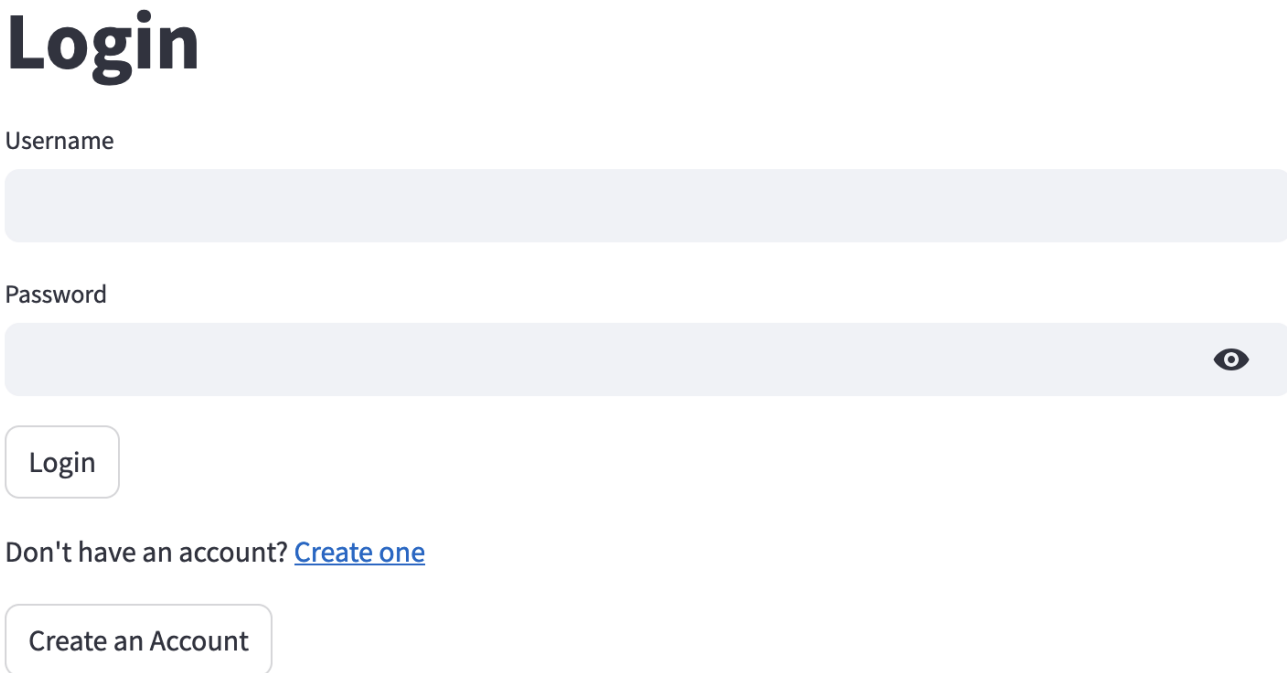
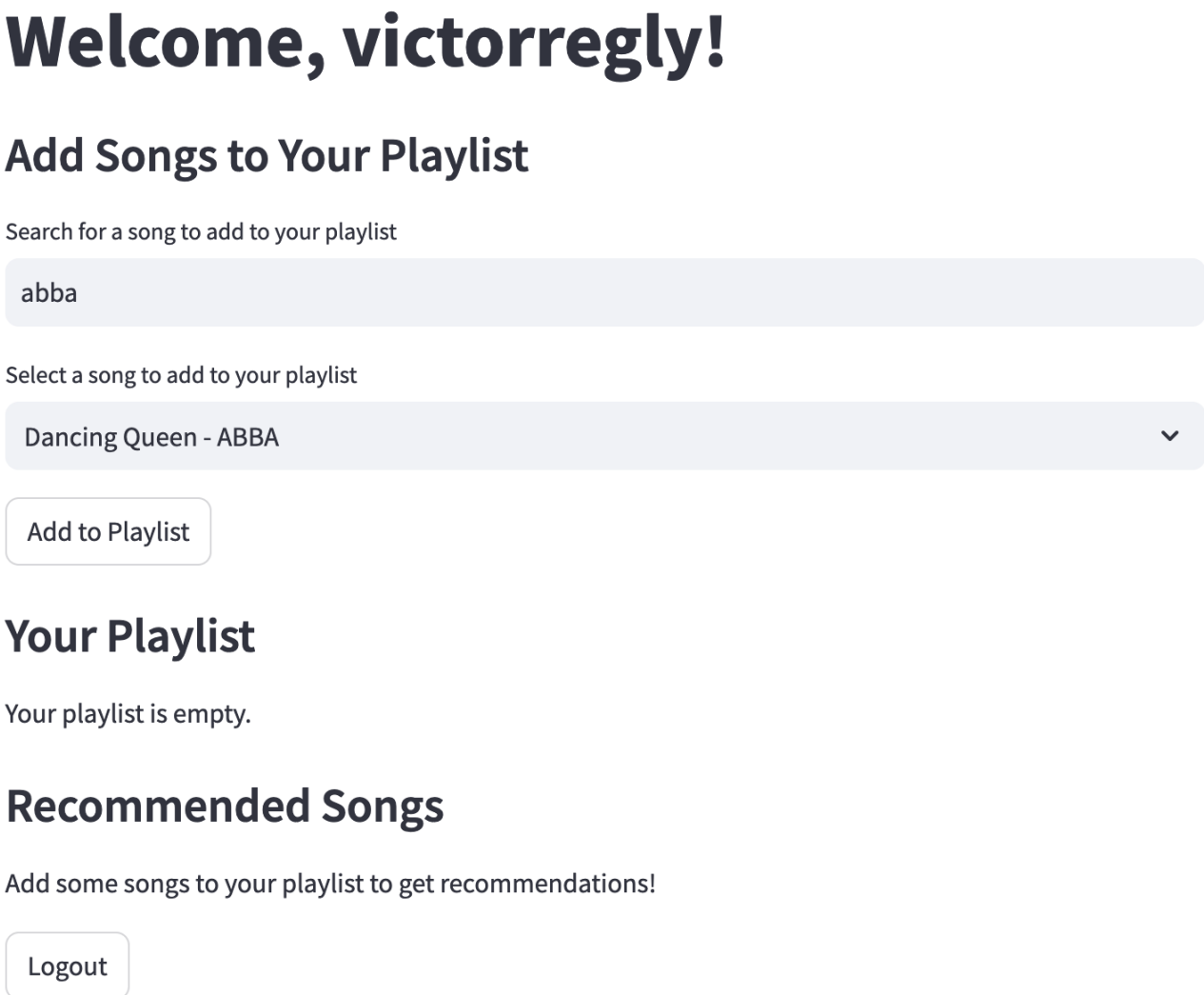


Figure: Login Screen

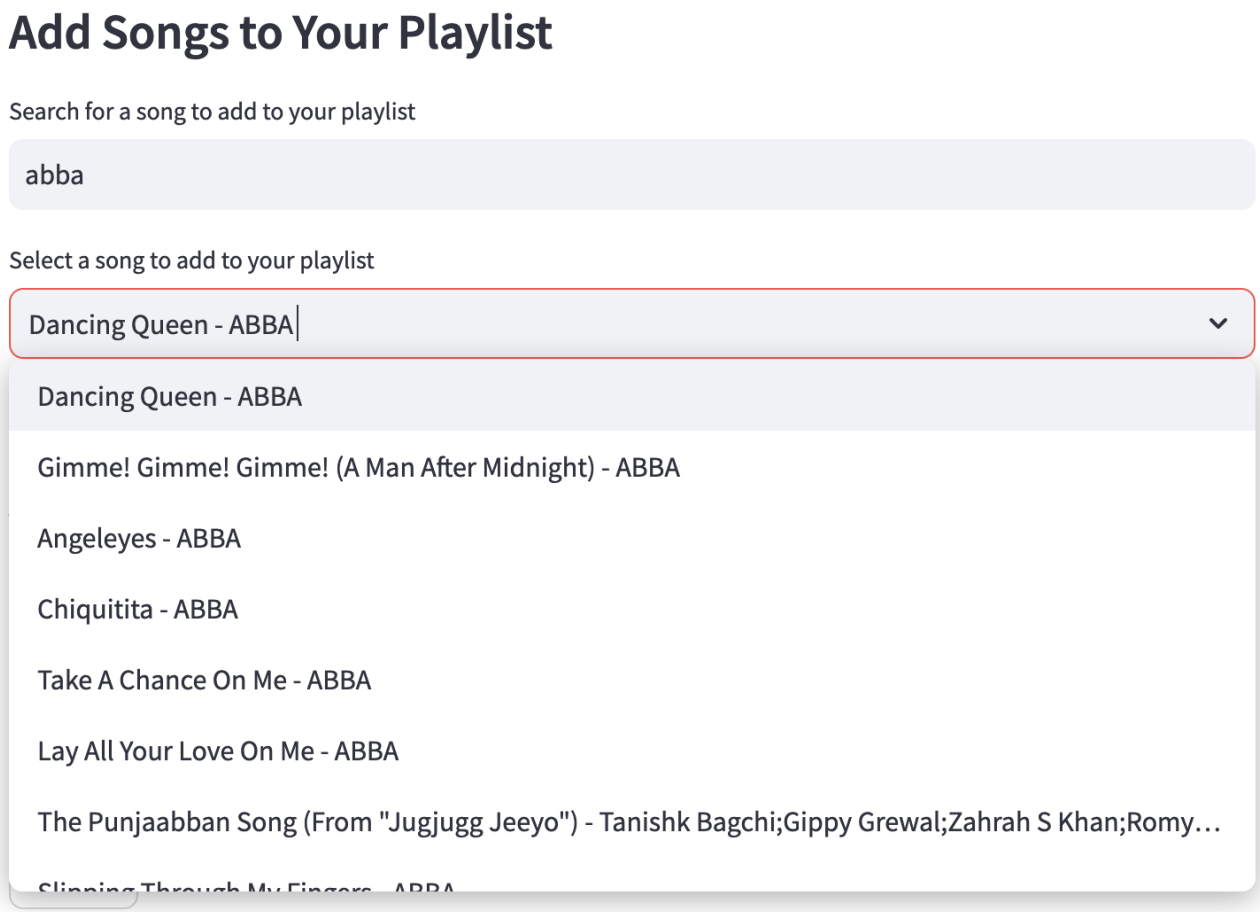
User Interface - Playlist Management

- Once logged in, users can view and manage their playlists.
- Users can search for new songs, add them to their playlist, or remove existing ones.



User Interface - Search and Recommendations

- Search bar allows users to find songs based on keywords and popularity.
- Recommendations are based on the user's current playlist and updated dynamically.



User Interface - Visualization and Plotting

- Users can view a plot showing the average characteristics of their playlist.
- Helps users understand the features that influence their recommendations.

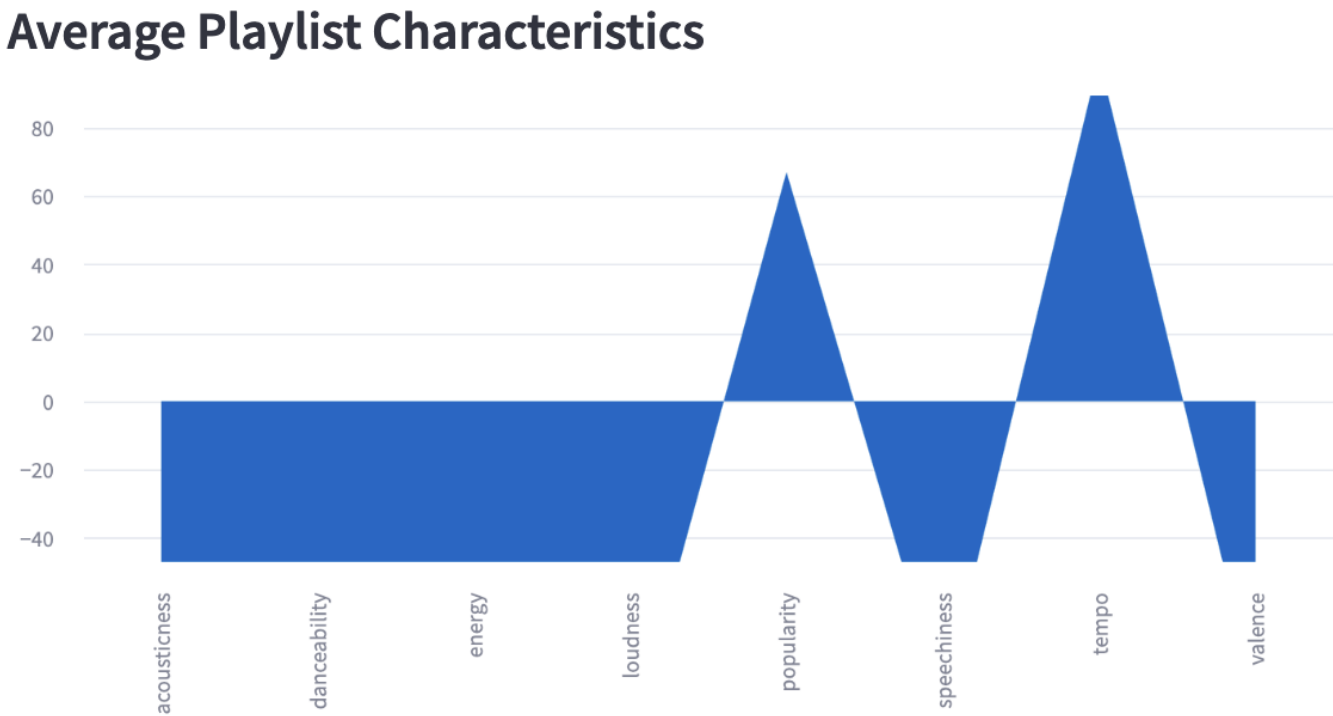


Figure: Plotting Feature

Performance Testing - Part 1

Response Time:

Playlist Size	Response Time (s)
10	0.124
50	0.127
100	0.137
500	0.254
1000	0.524

Memory Usage:

Playlist Size	Memory Usage (MB)
10	24.88
50	24.12
100	24.13
500	24.20
1000	24.30

Performance Testing - Part 2

Scalability:

- The system scales well with an increase in playlist size.
- Response time increases modestly with larger playlists, indicating good scalability.

Number of Recommendations:

- The system consistently provides the expected number of recommendations (5) regardless of playlist size.

Challenges and Solutions

- **Scalability of Data Storage:** Transition from CSV files to a database can improve scalability.
- **Real-Time Responsiveness:** Streamlit's reactivity ensures timely updates in the user interface.
- **Security Considerations:** Implementing password hashing with 'bcrypt' to secure user data.
- **Balancing Accuracy and Speed:** Optimization strategies include feature scaling and efficient data handling.

Conclusion and Future Work

Conclusion:

- The project successfully demonstrates a scalable, adaptive music recommendation system.
- Content-based filtering and feedback loops are effective in delivering personalized experiences.

Future Work:

- Explore advanced algorithms like deep learning for better recommendation accuracy.
- Migrate to a relational database to handle a larger user base.
- Expand the system to support other media types (e.g., podcasts, videos).

Questions?

Thank you for your attention! Any questions?