New York University

CSGY 6513 Big Data
Fall 2024

Prof. Juan Rodriguez

Final Project Report

Music-Based User Matchmaking System

Connecting People Through Shared Music Preferences

Name	**NetId**
Anjel Patel	ap8589
Siddhant Kulkarni	sk10841
Sudhanshu Sali	ss17526

Table of Contents

1. Introduction

In today's digital age, music has emerged as a universal language that transcends cultural and linguistic boundaries, offering a profound way to connect people. The rise of streaming platforms has given users unprecedented access to a vast array of music that aligns with their tastes and preferences. Yet, despite this accessibility, many individuals still experience feelings of loneliness and disconnection.

This project seeks to leverage big data to create a music-based user-matching system that connects people based on shared musical tastes. By analyzing extensive datasets comprising users' listening histories, track metadata, and demographic information, the system can uncover patterns and shared interests. Beyond personalized music recommendations, the algorithm aims to identify potential relationships between users with similar musical preferences. Ultimately, the goal is to foster meaningful connections and relationships through the shared emotional experiences conveyed by music.

Big data technologies, such as Apache Spark, enable the efficient processing and analysis of massive datasets. These tools are essential for managing the diverse and complex data required to understand user preferences and deliver accurate recommendations. By addressing both recommendation precision and user matchmaking, this initiative responds to the growing demand for authentic interactions in an increasingly digital world.

2. Background

Music is more than mere entertainment; it is a powerful medium for expressing emotions and fostering meaningful connections. Research suggests that musical preferences can reveal insights into an individual's personality traits and social compatibility. In an era dominated by digital interactions over face-to-face encounters, music offers a unique common ground to bridge this gap.

The idea of leveraging music for matchmaking is rooted in the belief that shared interests lead to deeper connections. Music evokes emotions and memories, making it an ideal foundation for building relationships. As digital connectivity grows, so does the challenge of combating feelings of isolation—underscoring the importance of strategies that cultivate genuine personal interactions.

Big data plays a pivotal role in this initiative, enabling the analysis of vast datasets that capture user behavior and preferences. Advanced algorithms and machine learning techniques can uncover patterns that reveal user compatibility, enhancing the precision of music recommendations while facilitating more meaningful social connections.

By leveraging big data technologies like Apache Spark, we can efficiently process and analyze large-scale datasets, extracting valuable insights into user preferences and habits. These insights allow us to develop a system that not only curates personalized music but also connects individuals based on their shared musical tastes.

3. Objectives

The primary objectives of this project are as follows:

1. <u>Build a Scalable System</u>: Utilize big data technologies, such as Apache Spark, to efficiently process and analyze user data. The system will be designed to handle increasing data volumes as the platform's user base grows.

2. <u>Enhance Recommendation Accuracy</u>: Incorporate collaborative filtering techniques, such as the Alternating Least Squares (ALS) algorithm, to generate personalized recommendations. Model performance will be fine-tuned through hyperparameter optimization to improve prediction accuracy.

3. <u>Facilitate User Matchmaking</u>: Identify connections between users with similar musical preferences, using shared tastes as a foundation to foster meaningful interactions and relationships.

4. Dataset Overview:

The Music-Based User Matchmaking System dataset includes user interaction data, track metadata, and user demographic information.

1. **User Interaction Data:**
    - Records user activities, including track play details and listening contexts.
    - Key Columns: CustomerID, TrackId, DateTime, Mobile, ListeningZIP.
2. **Track Metadata:**
    - Provides information about the tracks available on the platform.

- Key Columns: TrackId, Title, Artist, Length.

3. **User Demographics:**
   - Captures user profiles, aiding in personalizing recommendations.
   - Key Columns: CustomerID, Gender, ZIP, SignDate.

Dataset Characteristics that qualify it as Big Data:

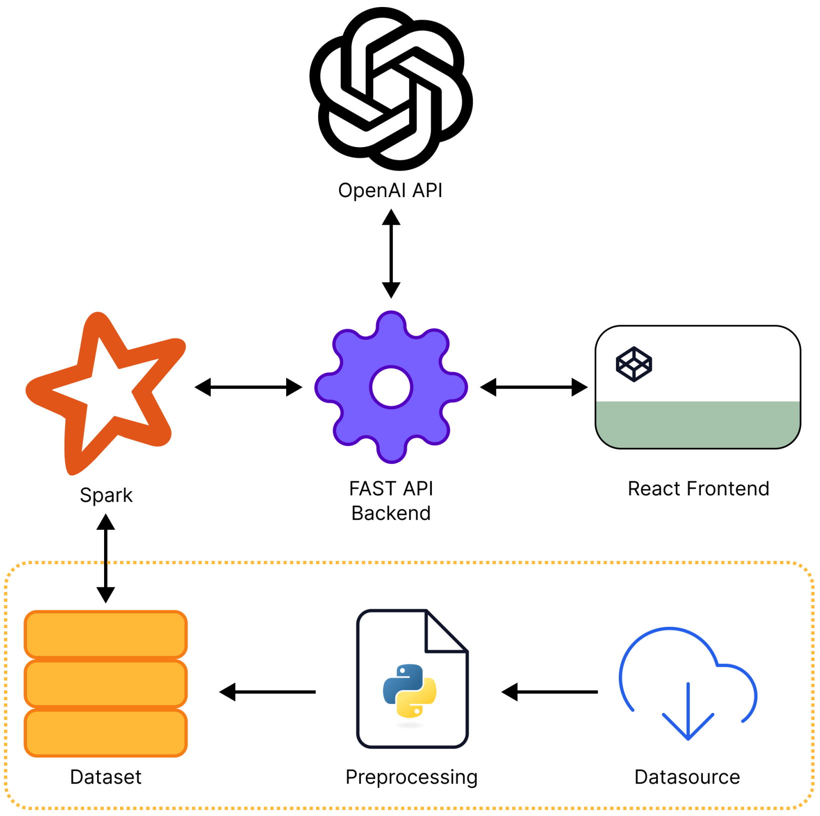
- **Volume:** The dataset contains millions of records, capturing detailed user interactions and preferences, making it substantial enough to extract meaningful behavioral insights.
- **Variety:** Includes diverse data types, such as timestamps, categorical attributes (e.g., gender), and numerical values (e.g., track duration), requiring advanced integration and processing techniques.
- **Veracity:** Ensures high reliability through thorough cleaning and standardization, addressing missing values and inconsistencies for accurate analysis.

5. System Architecture:

The architecture of the Music-Based User Matchmaking System is designed to efficiently process large-scale music-related data, generate personalized recommendations, and identify user matches based on shared preferences. It integrates multiple technologies to achieve scalability, accuracy, and an interactive user experience.

1. Data Source:
   - The system uses a dataset containing user listening history, track metadata, and demographic information. This dataset is processed to extract meaningful insights about user behavior and preferences.
2. Apache Spark (Big Data Processing):
   - Apache Spark is used for distributed data processing, enabling the system to handle millions of rows of user interaction data efficiently.
   - Tasks include: Cleaning and preprocessing raw data.
3. FastAPI (Backend):
   - FastAPI serves as the backend framework, providing RESTful APIs for communication between the front end and back end.
   - Handles requests for recommendations and matchmaking results.
   - Integrates with OpenAI's API for additional insights or text-based responses.
4. React (Frontend):
   - A React-based frontend provides an intuitive user interface where users can: View personalized music recommendations.
5. OpenAI API:
   - OpenAI's GPT models are used for generating text-based insights or summaries related to music preferences or recommendations.
6. Database:

- The database stores preprocessed data, user profiles, and recommendation results for efficient retrieval during API calls.



6. Methodology:

The methodology for the project is designed to create a robust and scalable recommendation system that connects users through shared music preferences. This approach involves data preprocessing, model implementation using big data technologies, and system performance evaluation. The following sections detail each phase of the process:

6.1 Data Preprocessing

Data preprocessing is essential in preparing the raw dataset for analysis and machine learning. The preprocessing tasks ensure that the data is clean, consistent, and optimized for model training.

1. **Handling Missing Values:**

- Missing or null values in critical columns, such as CustomerID, TrackId, and DateTime, were identified using PySpark's .isNull() function and removed with the .dropna() method. This step ensures that incomplete records do not negatively impact the model's performance.

2. **Datetime Parsing:**

- The DateTime column was parsed and standardized to ensure consistency across all records. PySpark's to_timestamp function was used to transform datetime fields to a uniform format, essential for time-based analysis.

3. **Feature Engineering:**
   Several new features were created to better capture user behavior and preferences:

- UserListeningCount: Total number of tracks listened to by each user. This feature provides an overview of the user's engagement with the platform.
- TrackPopularity: Frequency of a track being played across all users. This feature helps in identifying popular tracks within the user base.
- MobileUsageRatio: Proportion of interactions occurring on mobile devices. This feature reflects user behavior across different platforms (desktop vs. mobile).
- DaysSinceLastListen: The time elapsed since the user's last interaction with the platform. This helps capture user activity trends over time.

4. **Normalization:**

- Features such as UserListeningCount and TrackPopularity were normalized using Min-Max scaling to prevent any single feature from dominating the model training process. Normalization ensures that all features contribute equally to the model's performance.

5. **Data Splitting:**

- The dataset was split into two subsets: training (80%) and validation (20%) using PySpark's .randomSplit() method. This ensures that the model can be trained on one subset of data and evaluated on an unseen subset, providing an unbiased performance metric.

6.2 Model Implementation

The heart of this project lies in building a collaborative filtering recommendation system using the Alternating Least Squares (ALS) algorithm.

1. **Default ALS Model:**

- A baseline ALS model was implemented with default parameters. This serves as a benchmark to compare performance after optimization through hyperparameter tuning.
- **Parameters:**
  - rank=10: Number of latent factors to represent user and item interactions.
  - regParam=0.01: Regularization parameter to prevent overfitting.
  - maxIter=5: Maximum number of iterations for convergence.
  - coldStartStrategy="drop": Drops rows with missing predictions, avoiding errors in evaluation.
- The default model helped establish baseline performance metrics, which were later improved through hyperparameter tuning.

2. **Hyperparameter Tuning:**

- To enhance the ALS model, hyperparameter tuning was conducted using **TrainValidationSplit**, which tests different combinations of hyperparameters and evaluates their performance. This method is computationally efficient as it evaluates each parameter set only once.
- **Parameter Grid:**

```
param_grid = ParamGridBuilder() \
    .addGrid(als.rank, [10, 20, 30, 40]) \
    .addGrid(als.regParam, [0.001, 0.01, 0.05, 0.1]) \
    .addGrid(als.maxIter, [5, 10, 20, 50]) \
    .addGrid(als.alpha, [0.1,1,10])
    .build()
```

- Based on its performance on the validation set, the optimal parameters were chosen:
  - rank=40, regParam=0.05, maxIter=50, alpha = 0.1.

3. **Model Training:**

- Both the default and optimized ALS models were trained on the training dataset. During the training process, the algorithm learns the latent factors representing user and item preferences, which are used to predict user-item interactions (i.e. recommendations).

6.3 Prompt Engineering:

Prompt engineering was employed to effectively utilize OpenAI's GPT models for generating insights related to user preferences and recommendations. The following steps were taken:

1. Objective Definition:
    - The primary goal was to use GPT models to provide natural language explanations of recommendations and matchmaking results.
    - Example: "Why was this track recommended?" or "What makes these users similar?"
2. Prompt Design:
    - Prompts were carefully crafted to elicit specific responses from GPT models while ensuring clarity and relevance.
    - Prompt Engineering:
        i. N-shot prompting: LLMs work best when multiple examples are demonstrated. This helps retain format and set expectations. From testing 1, 2, 3, 4, and 5 shot prompts, we concluded that 3-shot prompting gives optimal results without being overshadowed by prior bias.
        ii. Format: Formats and separators were chosen that would not interfere with generated content. These formats and separators act as stop words for the LLM to aid generation and for our backend to parse output effectively. Skipping this step led to incompatible output strings.
        iii. Prompt exposure: LLMs work best when relevant content and nothing more is provided to them. For user compatibility queries, we provide songs and artists for both users which gives the LLM ample opportunity to make correlations and inferences.
        iv. Key phrase enhancement: Certain key phrases are shown to provide better results from an LLM. Phrases such as “Important”, “Let’s go step by step” and Role specification align with the underlying distribution of an LLM and leads to more reliable in-distribution results.
3. Dynamic Prompt Generation:
    - Prompts were dynamically generated based on user-specific data, such as their most-played tracks or top artists.
    - Example: "User A listens frequently to [Artist 1] and [Artist 2]. Explain why they might enjoy [Track X]."
4. Testing and Refinement:
    - Multiple iterations of prompts were tested to ensure that GPT responses were accurate, coherent, and aligned with project objectives.
    - Feedback loops were implemented where responses were evaluated for quality and adjusted as needed.
5. Integration with FastAPI:

- The final prompts were integrated into the FastAPI backend, allowing real-time interaction between the front end and OpenAI's API.

6.4 Generating Recommendations

Once the ALS model is trained, it is used to generate personalized music recommendations for individual users. Additionally, user matchmaking is performed based on shared musical tastes.

1. **Personalized Recommendations:**

   • For a given user (target_user_id), the model generates the top 10 recommended tracks. This is done by leveraging the ALS model’s latent factors to predict the user’s preferences for unseen tracks.

   • The output includes a list of track IDs and their predicted ratings, which represent the model’s suggestion of the most relevant music for the user.

2. **User Matchmaking:**

   • User matchmaking is facilitated by calculating the cosine similarity between the latent factors of different users. This step identifies users with similar musical tastes, creating opportunities for users to connect.

   • The top 5 most similar users are identified based on their similarity scores, allowing the system to recommend users with similar musical preferences.

6.5 Evaluation Metrics

To assess the performance of the recommendation system, the following evaluation metrics were used:

1. **Root Mean Squared Error (RMSE):**

   • RMSE measures the average magnitude of prediction errors, with lower values indicating better prediction accuracy.

   • Formula:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}$$

2. **Mean Absolute Error (MAE):**

- MAE calculates the average absolute difference between predicted and actual ratings, with lower values indicating better performance.
- Formula:

$$MAE = \frac{1}{n}\sum_{i=1}^{n} |\hat{y}_i - y_i|$$

Both RMSE and MAE were used to compare the performance of the default model and the hyperparameter-tuned model.

6.6 Results Comparison

The performance of the default ALS model and the optimized model was evaluated using RMSE and MAE metrics:

- **Default Model:**
    - RMSE: 0.57
    - MAE: 0.35
- **Optimized Model (After Hyperparameter Tuning):**
    - RMSE: 0.43
    - MAE: 0.26

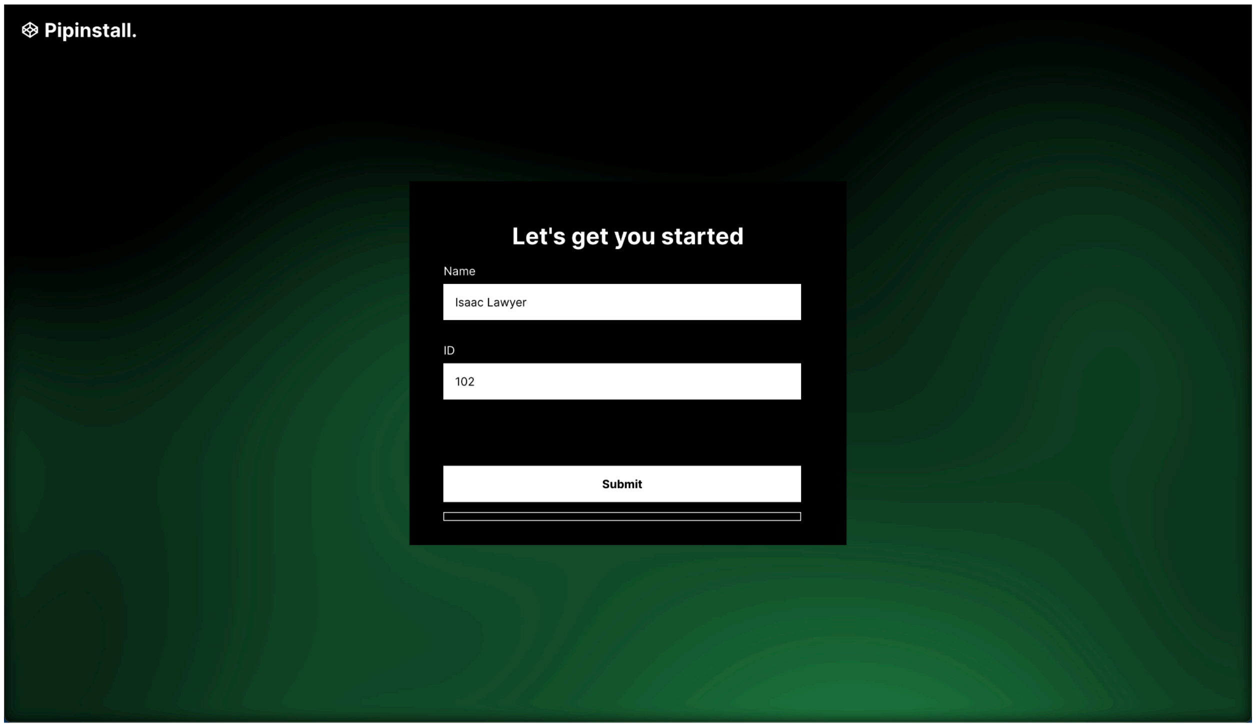
The hyperparameter-tuned model showed a significant improvement in both RMSE and MAE, confirming that the tuning process effectively enhanced recommendation accuracy.

7. Results

The Music-Based User Matchmaking System successfully demonstrated the ability to connect users through shared music preferences while providing personalized recommendations. Key results include:

- Achieved an **RMSE** of **0.43** and an **MAE** of **0.26** for the recommendation system, indicating high accuracy in predicting user preferences.
- Identified top-5 similar users for matchmaking using cosine similarity of latent factors derived from the ALS model.
- Enhanced user engagement by integrating a React-based frontend with a FastAPI backend, enabling seamless interaction with recommendations and matchmaking features.
- Processed over 1 million rows of user interaction data efficiently using Apache Spark, showcasing the scalability of the system.

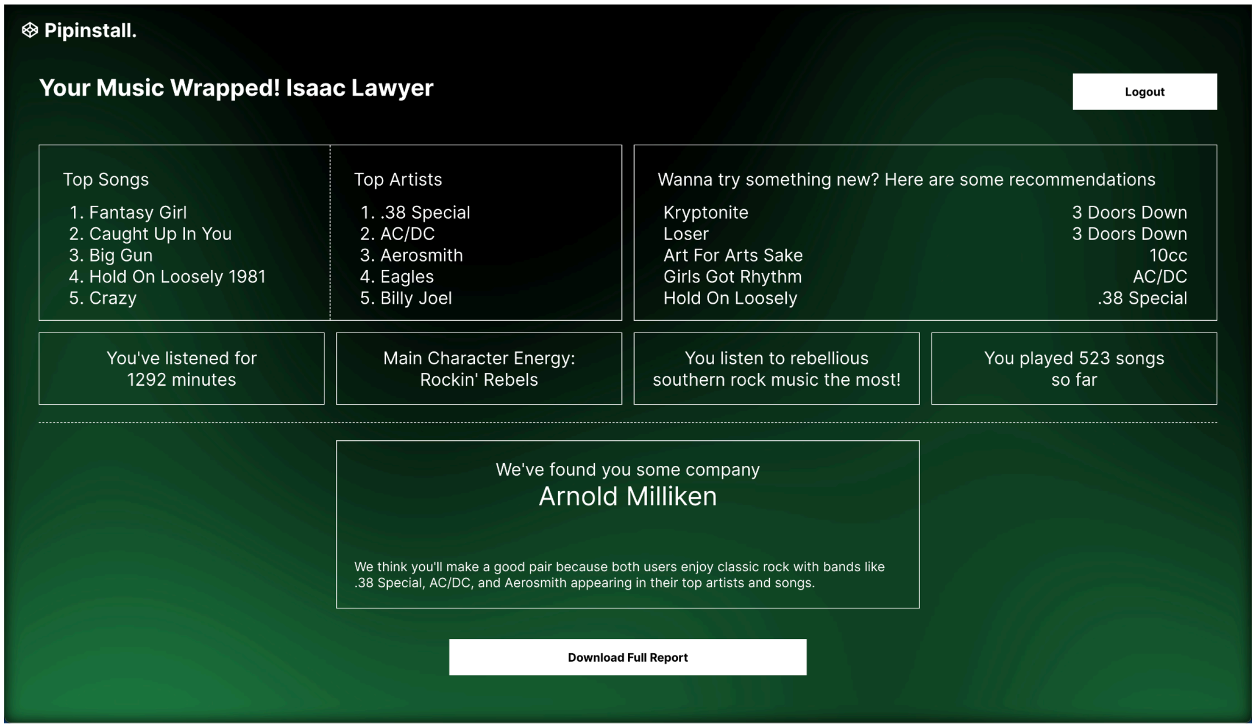
This combined with an elegant UI with great attention to detail and extensive edge-case consideration makes our app complete. Here's a sample of our UI - Starting with the login page:



Isaac Lawyer

102

The login functionality ensures a match between the name and user id with built-in validation checks on the front end. The UI is aware of time taken to process data and hence every non real-time component is packaged in with a loading bar. Once the data processing is complete, we are met with the wrapped results including all of the above mentioned functionality:



Your Music Wrapped! Isaac Lawyer

Top Songs
1. Fantasy Girl
2. Caught Up In You
3. Big Gun
4. Hold On Loosely 1981
5. Crazy

Top Artists
1. .38 Special
2. AC/DC
3. Aerosmith
4. Eagles
5. Billy Joel

Wanna try something new? Here are some recommendations

Kryptonite	3 Doors Down
Loser	3 Doors Down
Art For Arts Sake	10cc
Girls Got Rhythm	AC/DC
Hold On Loosely	.38 Special

You've listened for 1292 minutes

Main Character Energy: Rockin' Rebels

You listen to rebellious southern rock music the most!

You played 523 songs so far

We've found you some company
Arnold Milliken

We think you'll make a good pair because both users enjoy classic rock with bands like .38 Special, AC/DC, and Aerosmith appearing in their top artists and songs.

8. Conclusion

This project successfully implemented a scalable recommendation and matchmaking system that leverages big data technologies to connect users through shared music preferences. By combining collaborative filtering with advanced data preprocessing, the system provides personalized recommendations and identifies users with similar tastes, fostering meaningful connections.

The integration of Apache Spark for distributed data processing, FastAPI for backend services, and React for the frontend interface highlights the project's technical depth. The use of the Spotify dataset ensured real-world applicability, while evaluation metrics such as RMSE and MAE demonstrated the system's effectiveness.

This project addresses the growing need for genuine human connections in a digital world by using music as a medium to bridge gaps between individuals. It also showcases how big data technologies can be harnessed to solve complex social challenges.

9. Future Work

To further enhance the system's capabilities and expand its impact, the following areas are identified for future work:

1. Cold-Start Problem:
    - Incorporate content-based filtering or hybrid approaches to address cold-start issues for new users or tracks.
2. Real-Time Recommendations:
    - Implement real-time data ingestion pipelines using tools like Kafka to provide dynamic recommendations based on live user interactions.
3. Scalability Enhancements:
    - Deploy on cloud platforms like AWS or Google Cloud to handle larger datasets and global user bases.
4. Advanced Matchmaking:
    - Introduce additional similarity metrics (e.g., Jaccard similarity) and incorporate demographic data to improve matchmaking accuracy.
5. User Feedback Loop:
    - Integrate a feedback mechanism where users can rate recommendations or matches to refine future predictions.
6. Expanded Dataset Integration:
    - Include additional datasets (e.g., user-generated playlists or social media activity) to enrich user profiles and improve recommendation quality.

10. References

1. OpenAI API Documentation: https://platform.openai.com/docs/
2. "Collaborative Filtering for Implicit Feedback Datasets," Hu et al., IEEE International Conference on Data Mining.
3. Apache Spark Documentation: https://spark.apache.org/docs/latest/
4. FastAPI Documentation: https://fastapi.tiangolo.com/
5. React Documentation: https://reactjs.org/docs/getting-started.html
6. Pydantic Documentation: https://docs.pydantic.dev/