Music Recommendation System

Juan Carlos Gonzalez Aguilar, Eduardo Pereira, Utku Atay, Tingjun Yuan, Yi Yang

January 01, 2025

1 Introduction

Music holds crucial part in our life. It accompanies us with our moods and moments like when we feel happy or sad. However, finding the right music which resonate with us can be hard to find. Because there are a lot of music and it is getting harder each day to not lost into those songs and albums.

We wanted to solve this issue with our Music Recommendation System. Our music recommendation system trying to introduce a fresh approach to personalized music recommendations. This project brings a different perspective by combining knowledge representation with music recommendation. In the following sections, we will explain the technologies used, the architecture of our system, and the recommendation algorithms we implemented.

2 Application Description

2.1 Application System

Our Music Recommendation System is designed to effectively analyze user preferences and generate accurate music suggestions. This section explains the components of the system and how they interact with each other to deliver personalized music recommendations.

1. **User Interface (Front-End)**

   The user interface is built using JavaScript and Flask. It allows users to:

   - Enter their favorite artists.
   - View music recommendations.
   - Give feedback on suggested songs to improve future recommendations.

- The user input is collected and sent to the back-end for processing.

2. **Recommendation Engine (Back-End)**

   The back-end is the core of the system, responsible for analyzing user input and generating recommendations. It consists of two main modules:

   Content-Based Filtering: This module analyzes the content features of music tracks such as genre, tempo, mood, and instruments.

   Collaborative Filtering: This module compares the user's preferences with other users' listening patterns to find similarities and suggest music that others with similar tastes enjoy.

   The recommendation engine continuously improves by learning from user feedback.

3. **Semantic Data Integration**

   To enrich the recommendation process, the system integrates external semantic knowledge sources:

   YAGO and Wikidata provide structured information about artists, genres, and related musical concepts. This semantic data allows the system to understand deeper relationships between music entities and offer more meaningful recommendations.

4. **Music Information Retrieval**

   The system retrieves real-time music information using APIs like Spotify API. This enables the recommendation engine to access updated track data, artist details, and playlists.

5. **Data Flow and Interaction**

   The interaction between system components follows this flow:

   User Input: The user inputs their music preferences via the front-end interface.

   Data Processing: The back-end processes the input using semantic data from YAGO and Wikidata.

   Recommendation Generation: The recommendation engine applies content-based and collaborative filtering algorithms.

   Music Retrieval: Relevant song data is fetched from Spotify and presented to the user.

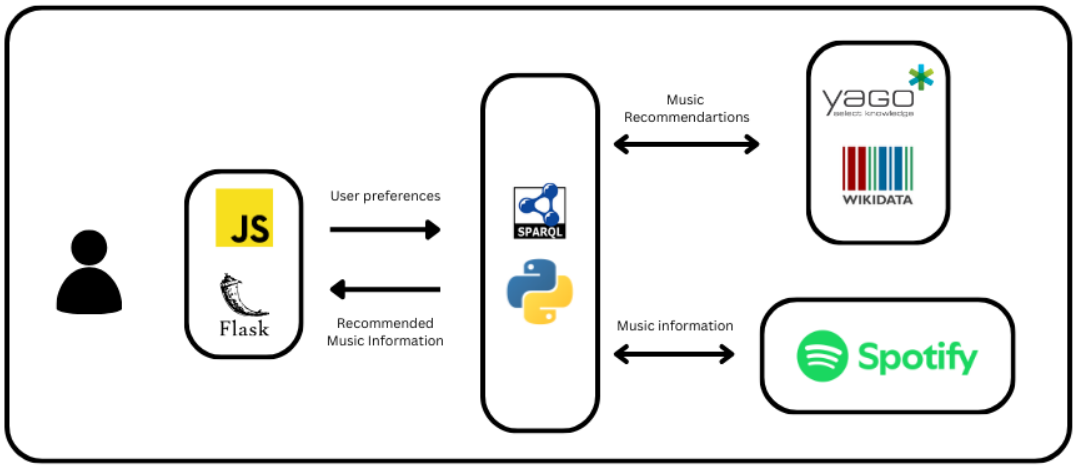
   Feedback Loop: User feedback is collected to refine future recommendations.

6. **System Overview**

   This architecture ensures a smooth flow of data and interaction among the components, resulting in an efficient and dynamic music recommendation system. By combining semantic data and advanced filtering techniques, the system delivers accurate and engaging music suggestions tailored to individual user preferences.

2.2 Technical Details

The following diagram represents the infrastructure of the application:



- User interacts with JavaScript front-end, indicating its music preferences.
- Flask facilitates communication with the python back-end.
- The Python back-end uses SPARQL queries to extract data from Yago and request library for querying directly to Wikidata ontologies.
- The Python back-end uses Spotify API to retrieve additional data from the spotify database.

2.3 Recommendation System

The general workflow is described on the Figure 1:

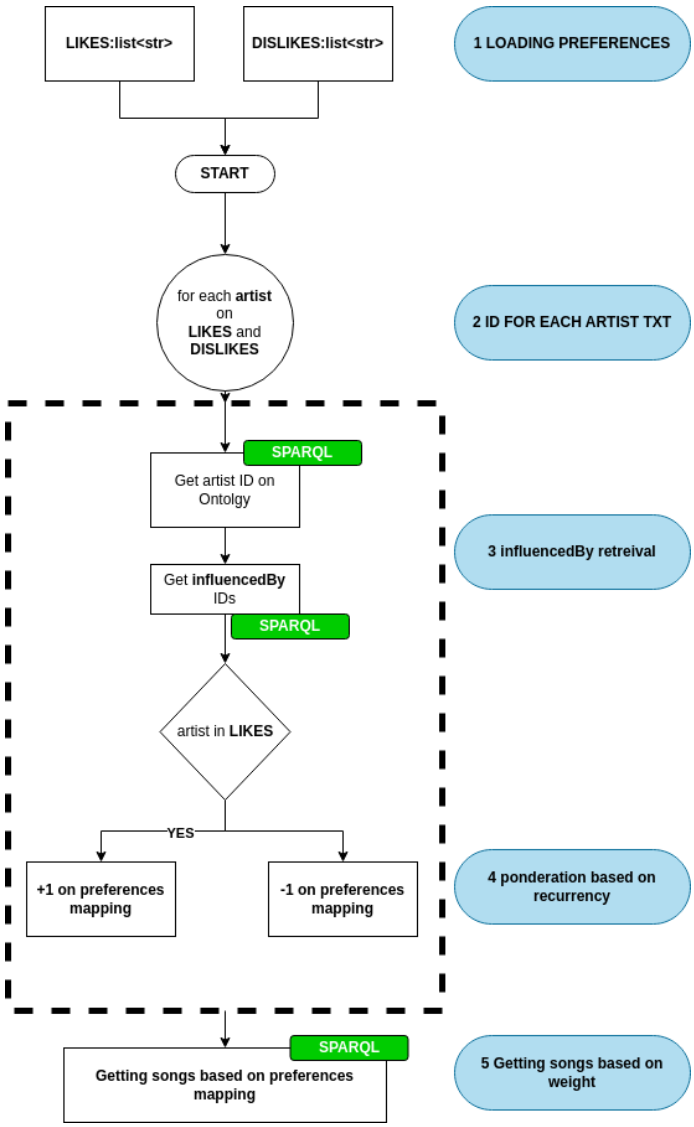


Figure 1: Recommendation workflow used to develop our Recommendation System.

For the recommendation system it was written on python using SPARQL and request

data depending on platform requirements.

Ontology Name	**Ontology Link**	**Ontology Description**	**Sparql Link**	**Python Library to use it**
YAGO	https://yago-knowledge.org/	"YAGO is a large knowledge base with general knowledge about people, cities, countries, movies, and organizations." (Suchanek et al., 2021)	https://yago-knowledge.org/sparql/query	SPARQLWrapper
WIKIDATA	https://www.wikidata.org/wiki/Wikidata:Main_Page	"Wikidata is a free and open knowledge base that can be read and edited by both humans and machines. Wikidata acts as central storage for the structured data of its Wikimedia sister projects including Wikipedia, Wikivoyage, Wiktionary, Wikisource, and others." (Vrandečić & Krötzsch, 2021)	https://query.wikidata.org/sparql	requests

Table 1: Ontology Resources with SPARQL Links and Python Libraries

For scoring artist we used a sum based on how many times the artist was mentioned on influencedBy RDFs.

$$\text{weight}(\text{artist}) = \sum_{x \in \text{likes}} 1(\text{artist} \in \text{influencedBy}(x)) - \sum_{x \in \text{dislikes}} 1(\text{artist} \in \text{influencedBy}(x))$$

You can find an explanation on the Figure 2.

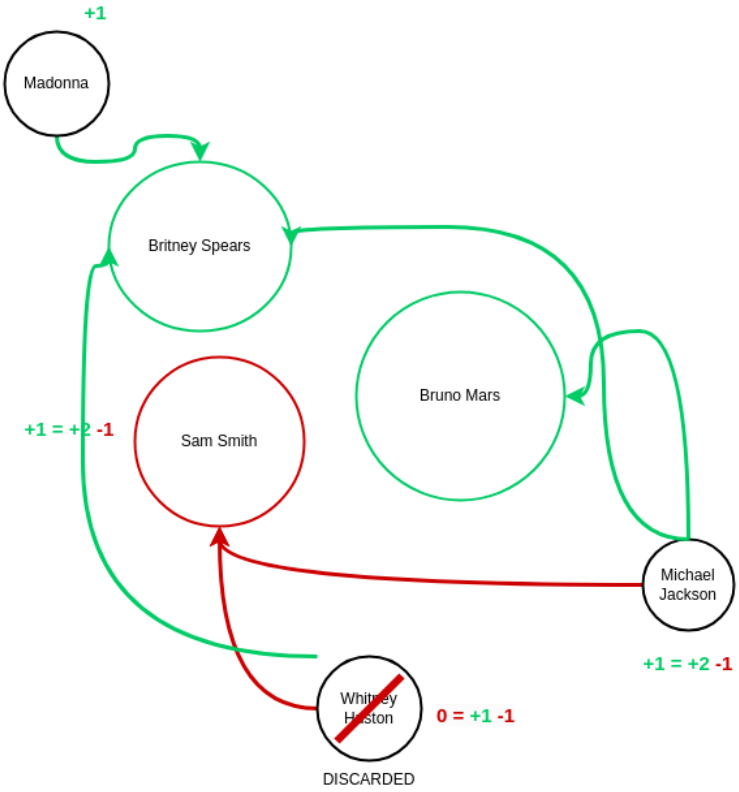


Figure 2: Weight Score Calculation Process for Artists Mentioned in `influencedBy`.

2.3.1 Recommendation Workflow

The recommendation workflow presented on this section is the core for getting recommendation using Sparql and ontologies knowledge. The algorithm 1 was made to use user-preferences to suggest songs based on influcendedBy attribute.

Algorithm 1 Recommendation Workflow Based on `influencedBy` Field

procedure RecommendationWorkflow
  Input: List of liked artists *likes*[], disliked artists *dislikes*[]
  Output: Sorted list of recommended songs *recommended_songs*[]
  ▷ Step 1: Load User Preferences
  User provides *likes*[] and *dislikes*[].
  ▷ Step 2: Retrieve IDs for Each Artist
  for all artist a in *likes*[] and *dislikes*[] **do**
    Query ontology for entities where:
      Type is "Musician" or "Group Musician".
      English label matches a (case-insensitive).
    Add retrieved IDs to *like_ids*[] or *dislike_ids*[].
  end for
  ▷ Step 3: Retrieve InfluencedBy Entities
  Initialize an empty dictionary $influenced\_by\_scores = \{\}$.
  for all ID *id* in *like_ids*[] **do**
    Query ontology for $influencedBy$ entities.
    for all influencedBy entity e **do**
      Increment weight of e in $influencedByCounters$.
    end for
  end for
  for all ID *id* in *dislike_ids*[] **do**
    Query ontology for $influencedBy$ entities.
    for all influencedBy entity e **do**
      Decrement weight of e in $influencedByCounters$.
    end for
  end for
  ▷ Step 4: Adjust Scores Based on Recurrence
  for all artist a in $influencedByCounters$ **do**
    if score of $a == 0$ **then**
      Remove a from $influencedByCounters$.
    end if
  end for
  ▷ Step 5: Retrieve Songs Based on Weighted Artists
  Sort $influencedByCounters$ by weight (descending).
  Limit to top N artists.
  for all artist a in top N **do**
    Query ontology for all songs by a.
    Add songs to *recommended_songs*[] with a's weight.
  end for
  Sort *recommended_songs*[] by artist weight.
  ▷ Step 6: Output Recommendations
  Return *recommended_songs*[].
end procedure

2.4 Application Usage

Our Music Recommendation System is designed to provide users with an enjoyable and interactive experience while discovering new music. In this section, we will explain the key functionalities of the application through several use cases.

- **Use Case 1:** Discovering New Music Based on Favorite Artists

  Scenario:

  A user wants to explore new songs similar to their favorite artist.

  Steps:

  The user opens the web application.

  They enter their favorite artist's name (e.g., Coldplay) in the search bar.

  The system retrieves related data from YAGO and Wikidata to find artists and songs connected to Coldplay.

  Outcome:

  Recommendations such as songs from similar bands (e.g., OneRepublic) are presented to the user.

- **Use Case 2:** Improving Recommendations with User Feedback

  Scenario:

  A user wants the recommendations to be more accurate over time.

  Steps:

  The user listens to the recommended songs.

  They can like, dislike tracks using feedback buttons.

- **Outcome:**

  The system collects this feedback and adjusts future recommendations accordingly.

3 Discussion

While working on our Music Recommendation System, we built up a lot of practical and theoretical knowledge. We had the opportunity to put our classroom lessons into practice, and we discovered how these ideas can be both demanding and rewarding in real applications.

Lessons Learned

The Importance of Data Quality

One of our biggest takeaways was the crucial role of high-quality, well-structured data. We used sources like YAGO and Wikidata to enrich our system, but we also spent a lot of time cleaning and organizing incomplete or inconsistent data. This showed us that data preparation is often just as important as the algorithms themselves.

Balancing Recommendation Methods

We noticed that using only content-based filtering or collaborative filtering did not give us the best results. Combining these approaches helped us offer more accurate and diverse recommendations. This taught us that blending different methods can improve both system performance and user satisfaction.

Semantic Web Technologies

It was our first time working with semantic web technologies. By dealing with ontologies and structured data, we realized how we can make meaningful connections between music genres, artists, and songs. This experience helped us understand that recognizing relationships in data is key to smarter recommendations.

User Feedback Matters

We also saw how important user feedback is. By allowing users to like or dislike recommendations, we could continuously refine our system. This process highlighted how dynamic and user-centric recommendation systems need constant updates to stay relevant.

Opportunities and Possibilities

Scalability with More Data Sources

We learned about the potential of semantic web and knowledge representation for scaling up the system. The more datasets we include, the more diverse and interesting recommendations we can provide.

Combining with Machine Learning

We realized that using machine learning algorithms with semantic data could significantly improve recommendation accuracy. For instance, deep learning models could better predict user preferences by spotting patterns in complex listening habits.

Cross-Domain Recommendations

The theories and methods we used aren't limited to music. We can adapt them to recommend movies, books, or even games based on music tastes, offering a broader entertainment experience.

Limitations and Challenges

Complexity of Semantic Data Working with semantic data wasn't as straightforward as it looked. Querying and processing large datasets like YAGO and Wikidata often slowed down the system. Optimizing performance turned out to be more difficult than expected.

Cold Start Problem

A common issue was the cold start problem. For new users with no listening history, providing reliable recommendations was tough. This is a well-known challenge in many recommendation systems and requires creative solutions.

Scalability Issues with Ontologies

Ontologies improved recommendation quality but also increased system complexity. Managing larger ontologies could slow things down and demand more computational resources.

User Privacy Concerns

Collecting user data to offer personalized recommendations raised privacy questions. We learned how important it is to find the right balance between personalization and user data protection.

4 Conclusion

This project allowed us to apply the theoretical knowledge we gained in class to a real-world scenario. We discovered the advantages of combining semantic web technologies with recommendation algorithms, but we also faced significant challenges, like handling big datasets and ensuring user privacy. Looking ahead, we believe that with further optimizations and more advanced techniques, the system can become even faster, more user-friendly, and more comprehensive.

References

Suchanek, F. M., Kasneci, G., & Weikum, G. (2021). Yago knowledge base [Accessed via https://yago-knowledge.org/]. *YAGO Homepage*.

Vrandečić, D., & Krötzsch, M. (2021). Wikidata: A free and open knowledge base [Accessed via https://www.wikidata.org/wiki/Wikidata:Main_Page]. *Wikidata Homepage*.