Graph-based recommendation system for music platforms

Emma Scinti Roger
273971@studenti.unimore.it

November 11, 2024

1 Abstract

This study presents a graph-based recommendation system tailored for music platforms, demonstrating the advantages of graph structures in capturing complex relationships between songs, genres, artists, and user interactions. Traditional recommendation systems often struggle with cold start issues and limited personalization in sparse datasets. By leveraging the Neo4j graph database and combining content-based filtering with graph traversal methods like shortest path and cosine similarity, this system provides both precise and diverse music recommendations. The implementation allows for playlist generation tailored to user preferences, including mood-based, genre-based, and temporal playlists. Preliminary analyses confirm that graph-based recommendations effectively broaden the variety of suggestions and offer contextually rich recommendations, even with limited data. Future enhancements may include integrating demographic data and additional playlist functionalities, enhancing the system's scalability and adaptability to varied user needs.

2 Introduction

Graph-based recommendation systems offer significant advantages over traditional methods by capturing more complex relationships between items and users. Studies in various fields, including music [3], have demonstrated the effectiveness of using graph structures to model entities and relationships for personalized recommendations. In this paper, graph-based methods are applied to a music dataset, exploring how relationships between songs, genres, and streaming events can be leveraged to generate recommendations. By comparing multiple recommendation techniques, this study aims to demonstrate the versatility and power of graph-based systems in the domain of music recommendations.

3 Background

Recommendation systems have become an integral part of modern digital platforms, helping users discover content by predicting their preferences based on past behavior or content attributes. These systems are widely used in domains such as e-commerce [10], streaming services, and social media [12].

3.1 The Use of Graphs in Recommendation Systems

Graphs are powerful tools in recommendation systems due to their ability to naturally model relationships between entities. In a graph, nodes represent entities like users, items (e.g., songs, movies), or attributes (e.g., genres, tags), while edges represent relationships or interactions between them.

Representing data as a graph enables capturing complex relationships, such as shared attributes and user preferences, in a scalable way. Graph-based recommendation systems can traverse these connections to identify patterns and make relevant suggestions.

Graph databases facilitate the integration of multiple data types, like user demographics, item attributes, and interactions, by linking nodes with various edges (e.g., "liked," "belongs to"). Algorithms

can then uncover implicit connections, allowing recommendations based on inferred relationships, such as shared genres or common user tastes.

The three primary types of recommendation systems are Collaborative Filtering, Content-Based Filtering, and Hybrid Systems [5].

3.2 Collaborative Filtering

Collaborative Filtering (CF) is one of the most common methods for recommendation systems and works by leveraging the preferences or interactions of a large number of users. There are two main types of CF:

- User-Based Collaborative Filtering: This method identifies users with similar preferences or behavior patterns and recommends items that those similar users have interacted with.

- Item-Based Collaborative Filtering: Instead of focusing on users, this method recommends items that are similar to those the user has previously interacted with.

Collaborative Filtering works well in environments with rich user interaction data. However, it suffers from the cold-start problem, where new users or new items lack interaction data, making recommendations difficult.

3.3 Content-Based Filtering

Content-Based Filtering (CBF) focuses on the attributes of the items themselves rather than relying on user interactions. Each item is described by a set of features, and the system recommends items with similar features to those the user has liked or interacted with in the past.

The advantage of content-based filtering is that it doesn't rely on other users' preferences, making it effective for personalized recommendations. However, it may limit the diversity of recommendations because it tends to suggest items very similar to what the user has already experienced.

3.4 Hybrid Systems

Hybrid systems combine the strengths of both Collaborative Filtering and Content-Based Filtering to improve the quality of recommendations. By integrating both methods, hybrid approaches can overcome the weaknesses of each individual approach.

For example, a hybrid system may initially recommend items based on a user's past preferences (content-based), but also incorporate collaborative filtering to introduce diversity by suggesting items popular among similar users.

3.5 Related Work and Previous Studies

Graph-based recommendation systems offer novel approaches to music recommendation, addressing limitations of traditional methods. Guo et al. [4] proposed a framework that enhances recommendation novelty by combining two types of graph structures: preference-directed graphs, with one-way edges representing a user's preferences between items, and positive-correlation undirected graphs, where undirected edges link items with a strong positive correlation. This hybrid approach leverages both user-driven preferences and item correlations to improve the diversity and relevance of recommendations. Similarly, Lee & Lee [7] developed a system using positively rated items to create a highly-connected graph, enhancing both novelty and relevance. Oramas et al. [8] leveraged knowledge graphs to enrich music and sound item descriptions, combining collaborative information with content features for improved recommendations. These graph-based approaches consistently outperformed traditional methods in terms of accuracy, novelty, and diversity, demonstrating their potential to enhance music discovery and user experience in recommendation systems.

The database management system Neo4j has been effectively utilized in developing recommendation systems across various domains. In e-commerce, Neo4j demonstrated superior efficiency over MySQL in managing large datasets, enabling more precise product recommendations based on user behavior patterns [11]. For social media platforms, Neo4j's

graph structure facilitated real-time recommendations by analyzing relationships between users and content [13]. In the aquaculture industry, Neo4j was employed to create a knowledge graph of fish species, providing recommendations for optimal ecosystems and species selection [6]. Neo4j's Graph Data Science library and Jaccard Similarity method have been leveraged to analyze customer purchase patterns, resulting in personalized product recommendations [1]. These studies highlight Neo4j's versatility and effectiveness in handling complex relationship networks and large-scale data, making it a valuable tool for developing sophisticated recommendation systems across diverse applications.

4 Dataset

For this study, we used the Streaming Activity Dataset available on Kaggle [2], which contains two csv files: Scrobble_Features.csv and My Streaming Activity.csv. The Scrobble_Features.csv file contains detailed information about the music tracks, including genre, duration, popularity, and various audio features. On the other hand, the My Streaming Activity.csv file offers a little more than 4 years worth of music streaming data from multiple platforms. In the Scrobble_Features.csv file these columns are specified:

- Performer: The name of the performer or artist of the song.
- Song: The title of the song.
- SongID: The title of the song and the performer
- Spotify_track_id: The id of the song in spotify
- spotify_genre: The genre(s) of the song according to Spotify.
- spotify_track_preview_url: URLs providing previews for each song on Spotify.
- spotify_track_duration_ms: The duration of the song in milliseconds.
- spotify_track_popularity: The popularity score of the song on Spotify.
- spotify_track_explicit: Indicates whether the song contains explicit content.
- danceability: A measure of how suitable a song is for dancing based on a combination of musical elements.
- energy: An indicator measuring the intensity and activity level present in a song's composition.
- key: Represents which key (C, D, E. . .) in which the track is composed.
- loudness: Reveals how loud or soft a given track is overall in decibels (dB).
- mode: Indicates whether the song is in a major or minor key. (i.e Major mode denoted by '1', minor mode denoted by value '0')
- Speechiness: A measure of the presence of spoken words in the song.
- Acousticness: A measure of the acoustic quality of the song determined by musical elements.
- Instrumentalness: A measure indicating the likelihood of the song being instrumental.
- liveness: A measure indicating the presence of a live audience in the song.
- valence: A measure indicating the positiveness conveyed through the song. '1'represents most positive , '0'mostly one (most presumably sad)
- tempo: The tempo of the song in beats per minute (BPM).
- time_signature: The time signature structure of the song.

In My Streaming Activity.csv these other attributes are found:

- Performer: The name of the artist or performer who created the song.
- Song: The title of the song.
- TimeStamp_Central: The timestamp of the streaming activity in the Central Time Zone.
- Album: The name of the album to which the song belongs.
- TimeStamp_UTC The timestamp of the streaming activity in Coordinated Universal Time (UTC).

The dataset was initially analyzed and explored through the use of Pandas to check the size of the dataset, the data format, etc. Pandas is a software library wrote for Python to perform manipulation

and analysis of data[9]. Through the *info* function of pandas it was seen that the song file had 13396 entries and 22 columns, while the streams file had 62907 entries and 7 columns, in Figures 7 and 8, in the Appendix, we can see the details.

This preliminary analysis showed that the streaming dataset is complete, with no null entries. However, in the song file, not all entries include all features. In particular, the Spotify URL field has a significant number of null values, with only 8,970 non-null entries. As noted on Kaggle, the streaming events are sourced from multiple platforms, which likely explains why only 8,970 entries have the Spotify attributes. Additionally, there are approximately 2,000 null values across various musical attributes, such as mode, energy, and danceability. The genre attribute is missing in fewer than 500 entries, while all entries include SongID, Performer, and Song information.

The statistics of some features were then explored to get an idea of their distribution. The results are shown in Figure 1, providing information on the mean, median, standard deviation, minimum and maximum values for each characteristic, as well as identifying any outliers.

	danceability	energy	valence	tempo	liveness
count	11843.000000	11843.000000	11843.000000	11843.000000	11843.000000
mean	0.563948	0.633178	0.507471	119.742096	0.199220
std	0.159638	0.233469	0.241694	30.568219	0.177449
min	0.000000	0.000898	0.000000	0.000000	0.010200
25%	0.456500	0.476000	0.322000	95.110500	0.095500
50%	0.571000	0.663000	0.505000	116.988000	0.126000
75%	0.676000	0.821000	0.696500	139.992500	0.244000
max	0.979000	1.000000	0.994000	232.690000	0.996000

	mode	speechiness	acousticness	instrumentalness
count	11843.000000	11843.000000	11843.000000	11843.000000
mean	0.674069	0.083196	0.268218	0.116702
std	0.468741	0.105924	0.302263	0.260084
min	0.000000	0.000000	0.000001	0.000000
25%	0.000000	0.034800	0.016100	0.000001
50%	1.000000	0.047900	0.134000	0.000231
75%	1.000000	0.082400	0.459000	0.031400
max	1.000000	0.955000	0.996000	0.994000

Figure 1

These preliminary analyses were conducted to gain an initial understanding of the data before transferring it to the graph, which offered a clearer foundation for modeling. More complex analyses will subsequently be performed directly on the constructed graph, due to the greater advantages this approach provides, as outlined in the previous section.

4.1 Graph Modelling

To build the graph from the dataset, we imported the data from CSV files into Neo4j and structured the graph using nodes and arcs representing entities and relationships relevant to a music recommendation system.

4.1.1 Nodes

1. **Song**: Each song is represented as a `Song` node, containing detailed information about its musical characteristics, which we specified earlier. The attributes are: `songId` (String), `title` (String), `key` (String), `liveness` (Float), `loudness` (Float), `mode` (Integer), `popularity` (Integer), `speechiness` (Float), `valence` (Float), `tempo` (Float), `instrumentalness` (Float), `time_signature` (String), `duration_ms` (Integer), `preview_url` (String), `acousticness` (Float), `danceability` (Float), `explicit` (String), `energy` (Float)

2. **Genre**: Musical genres are represented as `Genre` nodes. Each song can be associated with none, one, or multiple genres. The attribute of this node is: `name` (String)

3. **Performer**: Each artist or performer of a song is represented as a `Performer` node, with the following attribute: `name` (String)

4. **StreamingEvent**: Each streaming event is represented as a `StreamingEvent` node. This node records details about individual streaming instances with the following attributes: `timestamp_central` (String), `timestamp_utc` (String)

5. **Album**: Each song has specified the name of the album to which it belongs, it is represented as a `Album` node, with the following attribute: `name` (String)

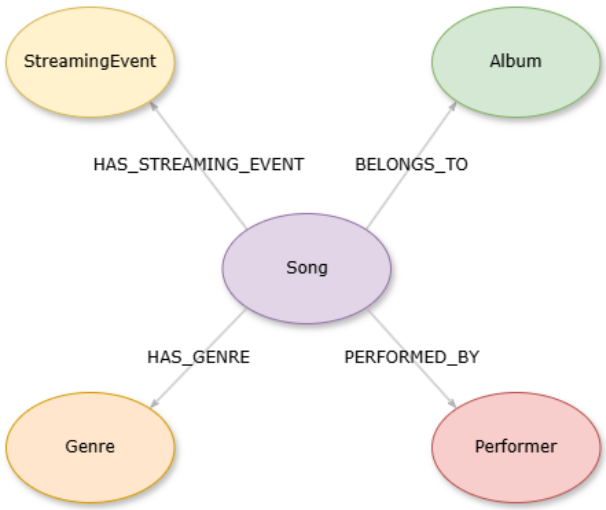

Figure 2: Graph scheme

4.1.2 Archs

Once the nodes were created, the arcs were added. All the modeled relations are listed here:

- **HAS_GENRE**: This relationship connects a `Song` node to one or more `Genre` nodes, allowing each song to be categorized under one or multiple genres. This connection enables genre-based recommendations and analysis.
- **PERFORMED_BY**: This relationship links a `Song` node to a single `Performer` node, indicating the artist who performed or created the song.
- **HAS_STREAMING_EVENT**: This relationship connects a `Song` node to one or more `StreamingEvent` nodes. Each `StreamingEvent` node represents an individual instance of a song being played at a specific time, thereby capturing user interaction data.
- **BELONGS_TO**: This relationship links a `Song` node to an `Album` node, indicating the album to which the song belongs.

The structure of the graph thus constructed in Neo4j is graphically illustrated in Figure 2:

5 Technical Implementation

The system is built using a combination of technologies. At its core, the **Neo4j graph database** serves as the foundation for data storage and querying.

The application layer is powered by the **Django web framework**, which provides a robust and flexible platform for building web applications. Django's features streamline development, ensuring that the recommendation engine is delivered seamlessly to users through an intuitive interface. Additionally, Django facilitates rapid deployment and integration with Neo4j, making it easier to manage both the backend and frontend.

The heart of the recommendation engine lies in the **Cypher queries**. Cypher, Neo4j's query language, enables the extraction of complex patterns from the graph database with precision and efficiency.

Together, these components create a multi-layered approach to music recommendation, ensuring that users receive varied, personalized, and contextually appropriate music recommendations while maintaining computational efficiency and scalability.

5.1 Graph exploration

Once the graph was created several queries were made through Neo4j in order to previously analyze it and explore the capabilities and utilities it can bring.

First, the number of nodes created for each entity were displayed, resulting in Table 1.

Node	Count
num_performers	5230
num_songs	17143
num_genres	1280
num_streaming_events	62907
num_albums	10464

Table 1: Number of nodes in the graph

It was observed that the number of Song nodes exceeds those listed in the CSV file containing song data. This discrepancy arose because, during graph construction, it was noted that some streaming events were linked to songs not present in the song data

CSV. Consequently, these additional songs were incorporated into the graph.

Other graph exploration queries have been executed; for example grouping all songs of a given genre or all songs that an artist has sung, find the top 10 most popular songs of a given genre or artist, and so on.

One can then also do searches based on specific characteristics of the songs, so grouping songs by, for example, duration, danceability level, energy, valence, etc. or find the most prolific artists.

To determine whether the spotify_popularity property of the songs was derived from the streaming events data, the three most popular songs of an artist were compared with the artist's three most-streamed songs (those with the highest number of associated StreamingEvents). Below the query is shown:

```
MATCH (p:Performer {name: "Eminem"})<-[:
    PERFORMED_BY]-(s1:Song)-[:
    HAS_STREAMING_EVENT]->(e:StreamingEvent)
WITH p, s1, COUNT(e) AS numero_streaming
ORDER BY numero_streaming DESC
LIMIT 3

WITH p, COLLECT({canzone: s1.title,
    numero_streaming: numero_streaming}) AS
    top_streamed

MATCH (p)<-[:PERFORMED_BY]-(s2:Song)
WHERE s2.popularity IS NOT NULL
WITH p, top_streamed, s2
ORDER BY s2.popularity DESC
LIMIT 3

WITH top_streamed, COLLECT({canzone: s2.title,
    popularity: s2.popularity}) AS top_popular

RETURN top_streamed, top_popular
```

As expected, the results showed differences: the two rankings, 3 and 2, do not align. This confirms that the streaming data under analysis is not the source used to assign spotify_popularity.

The streaming event data refers to a single user, as evidenced by the absence of a user identifier in the dataset. To further confirm this, the following query, was performed to check if there were any streaming events that occurred at exactly the same time.

Top Streamed Songs	Streams
Lose Yourself - From 8 Mile	23
Never Enough	19
Lose Yourself	13

Table 2: Top Streamed Songs by Eminem

Top Popular Songs	Popularity
'Till I Collapse	84
Without Me	82
The Real Slim Shady	80

Table 3: Top Popular Songs by Eminem

```
MATCH (e:StreamingEvent)
WITH e.timestamp_utc AS minute_timestamp, COUNT(
    e) AS event_count
RETURN minute_timestamp, event_count
ORDER BY event_count DESC
LIMIT 1
```

The issue is that the timestamp precision is limited to the minute, with seconds always set to zero. The query showed that the maximum number of streaming events occurring at the same time, or rather, within the same minute, is 5. This is a reasonable result when considering how often a user typically skips songs while listening on shuffle mode.

6 Recommendation System

The implemented recommendation system employs multiple approaches to provide diverse and personalized music recommendations to users. The system combines content-based filtering, path-based recommendations, and an hybrid approach to create a comprehensive recommendation engine.

6.1 Content-Based Recommendation

The primary content-based recommendation method is implemented in the 'get_similar_songs' function, which uses cosine similarity to find songs with similar audio features.

The cosine similarity method is employed to measure the similarity between songs based on a set of musical features. Cosine similarity calculates the cosine of the angle between two vectors in a multidimensional space, where each dimension represents a specific feature, in this case a musical feature: Danceability, Energy, Valence and Tempo. When two vectors are close in terms of their direction, this indicates that they have similar feature profiles. In such cases, the angle between them is near zero, and the cosine similarity value approaches 1, indicating a high level of similarity.

This technique is especially useful in music recommendation systems as it identifies songs with similar sound profiles, making it possible to recommend tracks that "sound" similar to those a user has previously listened to. By focusing solely on the intrinsic properties of each track, cosine similarity enables the system to make nuanced recommendations based on acoustic and musical qualities alone.

The 'get_similar_songs' function:

1. Takes a reference song title as input
2. Compares its feature vector with all other songs in the database
3. Uses cosine similarity with a threshold of 0.7 to ensure quality matches
4. Returns the top 50 most similar songs

The following piece of code is the query used in the function:

```
MATCH (s1:Song {title: $song_title})
MATCH (s2:Song)-[:PERFORMED_BY]->(p:Performer)
WHERE s1 <> s2
WITH s1, s2, p,
    [s1.danceability, s1.energy, s1.valence, s1.
        tempo] AS vector1,
    [s2.danceability, s2.energy, s2.valence, s2.
        tempo] AS vector2
WITH s1, s2, p, gds.similarity.cosine(vector1,
    vector2) AS similarity
WHERE similarity > 0.7 AND similarity < 1
RETURN s2.title AS RecommendedSong, p.name AS
    Artist, s2.preview_url as Url
ORDER BY similarity DESC
LIMIT 50;
```

Through this function, on the "Find Similar Songs" page, the user can enter the title of a song, and if it is in the database, the 50 most similar songs will be displayed.

The cosine similarity will be used also for other functionalities, as will be seen in next sections.

The system includes a variant, 'get_similar_unheard_songs', that focuses on recommending songs that the user hasn't streamed yet. This implementation:

1. Uses the same audio features as the basic similarity search
2. Applies a higher similarity threshold (0.9) to ensure very close matches
3. Specifically excludes songs with streaming events
4. Returns a set of 2 recommendations

This variant will be used alongside other methods to create additional recommendations on the "Discover New Songs By Similarity" page.

6.2 Shortest Path

The system implements a graph traversal approach through the 'get_shortest_path_unheard_songs' function, which:

1. Finds unheard songs that are connected to the input song through various relationships
2. Uses Neo4j's 'allShortestPaths' algorithm to find paths of length 1-2
3. Provides recommendations based on graph proximity rather than feature similarity
4. Returns diverse recommendations by considering different types of connections in the graph

This function is also used in conjunction with other methods to create the recommendations shown on the "Discover New Songs By Similarity" page. Below we see the query used in this case:

```
MATCH (s1:Song {title: $song_title}), (s2:Song)
       WHERE NOT (s2)-[:HAS_STREAMING_EVENT]->()
            AND s1 <> s2
       MATCH path = allShortestPaths((s1)
           -[*1..2]-(s2)), (s2:Song)-[:
           PERFORMED_BY]->(p:Performer)
       WITH DISTINCT s2.title AS RecommendedSong
           , p, s2.preview_url as Url
       RETURN DISTINCT RecommendedSong, p.name
           AS Artist, Url
       LIMIT 2;
```

Using shortest path analysis offers several advantages. By identifying paths of length 1 to 2 between the input song and potential recommendations, the system can uncover songs that are not only similar but also contextually related in a way that might not be immediately obvious from feature-based similarity alone. For example, these paths may connect songs through common genres, artists, or even albums, offering users a richer discovery experience. If two songs belong to the same album or are performed by the same artist, they are likely to be close in the graph. This closeness in graph terms reflects a shared context that makes these songs relevant recommendations for users interested in the input song. The shortest path approach ensures that recommendations are not randomly selected but are closely tied to the user's musical taste.

6.3 Genre-Based Discovery

The Genre-Based Discovery approach in this system is designed to provide users with song recommendations that align with their favorite genres while introducing them to new, unheard tracks. This method aims to balance variety with user familiarity by exploring genres that the user has recently engaged with, extracting songs from each, and filtering out any that have already been played. The two primary functions used in this process:

1. 'get_playlist_by_genre': This function first reviews the user's recent listening history and tags each song with its associated genre. This allows the system to form a list of the genres the user is currently interested in.
2. 'get_unheard_songs_by_genre': For each genre identified, this function pulls songs that belong to the genre but have not yet been listened to by the user. These songs are added to the recommendation pool, limited to a maximum of 10 songs per genre to prevent over-representation and ensure diversity.

6.4 Mood-Based Playlists

The system implements a mood-based playlist generator through the 'playlist_by_mood' function. This approach dynamically generates playlists based on a wide range of input parameters and filters, which allows it to cater to specific moods and scenarios. In the figure 3 is shown the menu that let's the user generate one of the playlists that will be described in the following.

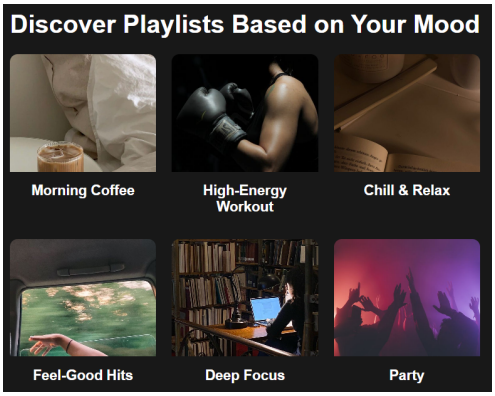

Figure 3

This flexible playlist generator uses the graph database to run customized queries, applying various musical attributes such as tempo, energy, acousticness, and popularity to filter and recommend songs. The main query structure and filtering logic is organized in this way:

- **Primary Filtered Query**: The function begins by constructing a filtered_query using the filters provided as input. This query matches songs (s:Song) and their associated artists (p:Performer), applies the user-specified filters

as conditions in the WHERE clause, which narrows down song choices to those that best match the desired mood or activity. Finally it excludes Christmas-related songs unless the current month is December, ensuring seasonally appropriate recommendations, since it was seen that if low values of some features were required, like in the morning playlist, a lot of Christmas songs were recommended. Finally the function returns a sorted list of song titles, artists, and preview URLs ordered by popularity.

- **Similarity Query for Popular Time-of-Day Tracks**: When a time_of_day parameter is provided, the function generates an additional similarity_query. This query uses historical streaming data to identify songs that are popular during the specified time (e.g., morning or late night). Songs from the filtered pool are compared to the most popular tracks within the time window using a cosine similarity score based on musical features like danceability and tempo. Only songs with a similarity score above 0.75 are included, ensuring the playlist's coherence and suitability for the time. Similar to the filtered query, the similarity-based query excludes Christmas-related songs when it's not December.

- **Combining Results**: The function then combines results from both queries: When time_of_day is specified, it merges recommendations from both the filtered and similarity-based results, providing a well-rounded playlist tailored to both user filters and temporal preferences. If time_of_day is not specified, only the filtered results are used.

The queries used by the function that was just described can be found in the Appendix 1.

Thanks to this playlist generator the followings playlists can be created by the user by navigating in the "Mood" page:

- Morning Coffee: Filters for low energy (s.energy <0.5), high acousticness (s.acousticness >0.7), and a tempo under 100 BPM (s.tempo <100). Ideal for a calm, uplifting start of the day, including songs similar to what the user usually listens to in the morning.

- Late Night Vibes: Features tracks with low energy (s.energy <0.5), low liveness (s.liveness <0.3), and a tempo under 80 BPM (s.tempo <80). Perfect for relaxing late at night, including songs similar to the personal taste of the user during this time.

- High-Energy Workout: Selects high-energy songs (s.energy >0.8), with an upbeat tempo (s.tempo >120), high danceability (s.danceability >0.7), and loud volume (s.loudness >-6). Optimized to boost motivation during workouts.

- Chill and Relax: Filters for very low energy (s.energy <0.4), high acousticness (s.acousticness >0.6), and low volume (s.loudness <-10). Ideal for unwinding, meditating, or studying.

- Feel-Good Hits: Features tracks with positive valence (s.valence >0.7), moderate energy (between 0.4 and 0.7), popularity over 50 (s.popularity >50), and high danceability (s.danceability >0.5). A collection of upbeat, moderately energetic, and highly danceable tracks, ideal for fostering a positive and motivated mood.

- Deep Focus: Filters for instrumental tracks (s.instrumentalness >0.7), low energy (s.energy <0.5), minimal speech (s.speechiness <0.3), and moderate-to-low volume (s.loudness <-8). Designed for studying or working with minimal distractions.

- Party: Includes high-energy tracks (s.energy >0.8), high danceability (s.danceability >0.7), popularity over 60 (s.popularity >60), and loud volume (s.loudness >-5). Perfect for parties or social gatherings.

- Instrumental Chillout: Contains instrumental tracks (s.instrumentalness >0.7), high acousticness (s.acousticness >0.5), and low speech

(s.speechiness <0.3). Perfect for creating a calm, unobtrusive background.

- On The Road: Filters for medium-to-high energy (s.energy >0.5), positive valence (s.valence >0.5), tempo between 90 and 130 BPM (s.tempo $\geq$ 90 AND s.tempo $\leq$ 130), and popularity over 40 (s.popularity >40). Ideal for road trips with uplifting and energetic tracks.

- Romantic Vibes: Curates romantic tracks with positive valence (s.valence >0.5), a slow tempo (s.tempo <90), high acousticness (s.acousticness >0.4), and no explicit content (s.explicit = 'False'). Ideal for a cozy evening.

6.5 Custom Playlist Generation

The system also offers personalized playlist creation through the 'PlaylistRecommender' class, allowing users to specify their preferred value of : Danceability, Energy, Valence, Tempo, Acousticness

The algorithm then:

1. Finds songs within ±0.2 range of specified audio features
2. Calculates similarity using feature vectors
3. Returns top 50 matches with similarity >0.7

The interface displayed in the Figure 4 leverages the functionality provided by the class described above.

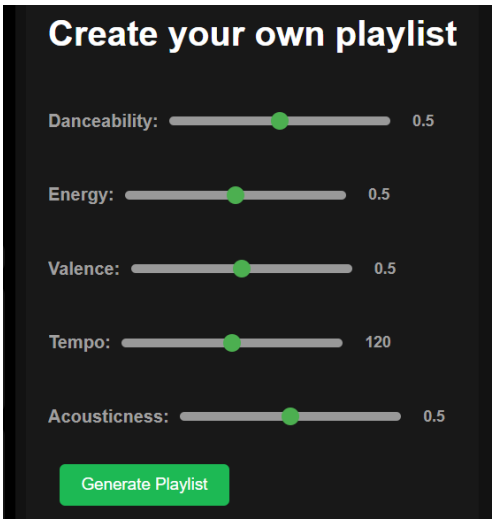

Figure 4

6.6 Hybrid Discovery System

The function 'discover_songs_by_similarity' aims to create a list of music recommendations based on a combination of approaches to maximize the variety and relevance of the proposed tracks. It uses the function 'get_recently_listened_songs' to obtain a list of songs that the user has recently listened to. These songs act as a "seed" for the recommendations. For each recently listened song, the function 'get_similar_unheard_songs', as seen above, is called to return two similar songs that the user has not yet listened to. The found tracks are added to the recommendations set (recommended_songs_set), including the title, artist, and URL. If fewer than two similar songs are found, the function uses the approach based on shortest paths (get_shortest_path_unheard_songs). The function stops if at least 40 songs are found. However, if 40 recommendations are not reached, the function proceeds to an additional search using 'get_similar_songs', as seen above.

This hybrid approach ensures:

- Diverse recommendations
- Balance between similarity and discovery
- Focus on unheard content

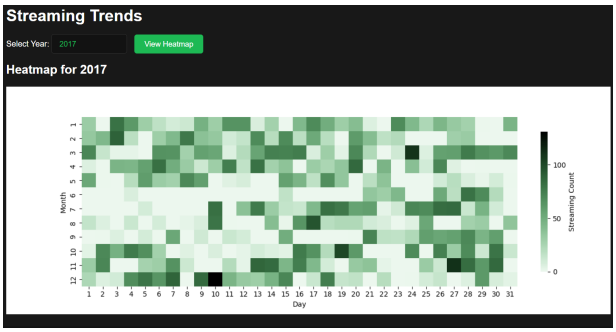

Figure 5

- Robust recommendation generation even with limited data

6.7 Streaming Trends

On the Streaming Trends page, the user can view the frequency of listens throughout the selected year 5. A heat map is displayed, where darker tones represent a higher number of listens for each day. The x-axis shows the days, while the y-axis represents the months. These maps allow users to identify listening patterns and observe periods when they listened to more or less music.

A similar analysis was conducted to verify that the streaming data was genuine or at least realistic, by examining the listening trends for songs containing the words "Christmas," "Xmas," or "Santa" over the year. The query used 2 and the result 9 are presented in the appendix. The results align with expectations, showing that the streamings are concentrated in December, with only a few exceptions.

7 Results and Discussion

The recommendation system implemented several techniques: content-based recommendations, graph-based recommendations (using shortest path analysis), and a hybrid approach combining multiple methodologies.

Compared to traditional methods, a graph-based system offers significant advantages, such as the ability to overcome the cold start problem, related to the absence of explicit preferences, and to expand the diversity of recommendations. The graph model effectively captures complex connections between songs, genres, and artists, allowing it to infer less obvious relationships that contribute to a more varied recommendation experience.

The content-based recommendations, leveraging cosine similarity, effectively identified songs with similar sound characteristics to those preferred by users, achieving high precision in recommendations.

A test is shown below, where through the page "Find Similar Song" we searched for song similar to 'Lose Yourself', the first result was the same song, but in a different version, with a slightly different title, this confirm us that the similarity method works, showing as first the most similar song.

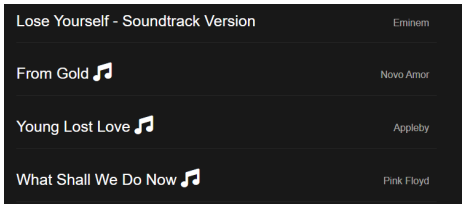

Figure 6: Songs similar to 'Lose Yourself'

The graph traversal approach, including shortest path analysis, enables the system to offer recommendations that go beyond direct user-item interactions. By leveraging relationships such as shared genres, performers, and albums, the graph-based recommendations deliver content that aligns with user tastes but also promotes discovery by suggesting related but previously unexplored tracks.

The hybrid recommendation approach, which combines content-based and graph-based methods, balances accuracy and diversity effectively, leading to a robust recommendation system capable of addressing various user needs and preferences.

The mood-based and genre-based playlists generated by the system demonstrated substantial variety and relevance to user preferences. Playlists like "Morning Coffee" and "High-Energy Workout" effectively met specific user needs, adding a temporal dimension to the recommendations, enhancing personalization.

Several challenges were encountered, the main one is the fact that the streaming data reflected the behavior of a single user so other type of recommendations, like collaborative filtering, couldn't be tested. Future enhancements could address these challenges to improve scalability and broaden the system's applicability.

8 Conclusions and Future Works

This study demonstrated that a graph-based recommendation system offers a powerful and versatile approach to music recommendation, even when the data are referred to one single user.

Future work could focus on integrating user demographic data for even more personalized recommendations. Many additional features could be added to the recommendation system in Django, leveraging the capabilities offered by the graph structure. For instance, playlists could be also created based on a specific artist or on albums. Additional statistics could be included on the "Streaming Trends" page. To enhance the app's functionality, it would be beneficial to save the created playlists and generate new ones by querying the graph only periodically, at a fixed interval of time or upon user request once they have already listened to the current songs and want to discover new ones.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13396 entries, 0 to 13395
Data columns (total 22 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   index                     13396 non-null  int64
 1   SongID                    13396 non-null  object
 2   Performer                 13396 non-null  object
 3   Song                      13396 non-null  object
 4   spotify_genre             12923 non-null  object
 5   spotify_track_id          11843 non-null  object
 6   spotify_track_preview_url  8970 non-null  object
 7   spotify_track_duration_ms  11843 non-null  float64
 8   spotify_track_popularity  11843 non-null  float64
 9   spotify_track_explicit    11843 non-null  object
 10  danceability              11843 non-null  float64
 11  energy                    11843 non-null  float64
 12  key                       11843 non-null  float64
 13  loudness                  11843 non-null  float64
 14  mode                      11843 non-null  float64
 15  speechiness               11843 non-null  float64
 16  acousticness              11843 non-null  float64
 17  instrumentalness          11843 non-null  float64
 18  liveness                  11843 non-null  float64
 19  valence                   11843 non-null  float64
 20  tempo                     11843 non-null  float64
 21  time_signature            11843 non-null  float64
dtypes: float64(14), int64(1), object(7)
memory usage: 2.2+ MB
```

Figure 7: Song dataset info

A Additional Figures and Code

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 62907 entries, 0 to 62906
Data columns (total 7 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   index              62907 non-null  int64
 1   SongID             62907 non-null  object
 2   TimeStamp_Central  62907 non-null  object
 3   Performer          62907 non-null  object
 4   Album              60348 non-null  object
 5   Song               62907 non-null  object
 6   TimeStamp_UTC      62907 non-null  object
dtypes: int64(1), object(6)
memory usage: 3.4+ MB
```

Figure 8: Streams dataset info

Listing 1: Query to Generate the Playlists of the Mood Page

```
    filtered_query = '''
    MATCH (s:Song)-[:PERFORMED_BY]->(p:Performer
        )
    WHERE ''' + ' AND '.join(filters)

    if current_month != 12:
        filtered_query += '''
        AND NOT (s.title CONTAINS 'Christmas' OR
                s.title CONTAINS 'Santa' OR
                s.title CONTAINS 'Noel' OR
                s.title CONTAINS 'Mistletoe' OR
                s.title CONTAINS 'Silent Night' OR
                s.title CONTAINS 'Bell')'''

    filtered_query += '''
    RETURN s.title AS title, p.name AS artist, s
        .popularity AS popularity, s.preview_url
         as Url
    ORDER BY s.popularity DESC
    LIMIT $num
    '''

    similarity_query = '''
    MATCH (popular:Song)-[:HAS_STREAMING_EVENT
        ]->(e:StreamingEvent)
    WITH apoc.date.parse(e.timestamp_utc, 'ms',
        'MM/dd/yyyy hh:mm:ss a') AS timestamp_ms
        , popular, e
    WITH datetime({epochMillis: timestamp_ms})
        AS timestamp_utc, popular, e
    WHERE '''

    if time_of_day == 'morning':
        similarity_query += "time(timestamp_utc)
            >= time('06:00:00') AND time(
            timestamp_utc) < time('12:00:00')"
    elif time_of_day == 'night':
        similarity_query += "time(timestamp_utc)
            >= time('22:00:00') OR time(
            timestamp_utc) < time('04:00:00')"

    similarity_query += '''
    WITH popular, COUNT(e) AS listen_count
    ORDER BY listen_count DESC
    LIMIT 5

    MATCH (s2:Song)-[:PERFORMED_BY]->(p:
        Performer)
    WHERE NOT (s2)-[:HAS_STREAMING_EVENT]->()
        AND popular <> s2
    WITH popular, s2, p,
         [popular.danceability, popular.energy,
             popular.liveness, popular.tempo,
             popular.acousticness] AS vector1,
         [s2.danceability, s2.energy, s2.liveness
             , s2.tempo, s2.acousticness] AS
             vector2
    WITH popular, s2, p, gds.similarity.cosine(
        vector1, vector2) AS similarity
    WHERE similarity > 0.75'''

    if current_month != 12:
        similarity_query += '''
        AND NOT (s2.title CONTAINS 'Christmas' OR
                s2.title CONTAINS 'Santa' OR
                s2.title CONTAINS 'Noel' OR
                s2.title CONTAINS 'Silent Night'
                    OR
                s2.title CONTAINS 'Silver Bells')
                    '''

    similarity_query += '''
    RETURN s2.title AS title, p.name AS artist,
        similarity, s2.preview_url as Url
    ORDER BY similarity DESC
```

```
    LIMIT 25
    '''
```

Listing 2: Query to Retrieve Christmas songs streaming trends

```
MATCH (s:Song)-[:HAS_STREAMING_EVENT]->(e:
    StreamingEvent)
WHERE s.title CONTAINS "Christmas" OR s.title
    CONTAINS "Xmas" OR s.title CONTAINS "Santa"
AND e.timestamp_utc IS NOT NULL
WITH apoc.date.parse(e.timestamp_utc, 'ms', 'MM/
    dd/yyyy hh:mm:ss a') AS timestamp_ms, s,e
WITH datetime({epochMillis: timestamp_ms}) AS
    timestamp_utc, s,e
WITH date(timestamp_utc) AS day, COUNT(e) AS
    value
RETURN day, value;
```

Figure 9: Christmas Songs Streaming Trends

References

[1] Fitrio Dermawan, Chang Hong Kwang, Muhammad Dimas Adijanto, Nur Aini Rakhmawati, and Naufal Rafiawan Basara. Product recommendations through neo4j by analyzing patterns in customer purchases. *2024 ASU International Conference in Emerging Technologies for Sustainability and Intelligent Systems (ICETSIS)*, pages 1–4, 2024. 3

[2] The Devastator. Streaming activity dataset, 2020. Accessed: 2024-10-21. 3

[3] Sanjay G, Aakash T, Pragatheesh A, and Kiruthika N. 1. dynamic perspective defined graph attention network towards sequential music recommendation. 2024. 1

[4] Ruixing Guo, Chuang Zhang, Ming Wu, and Yutong Gao. A graph-based novelty research on the music recommendation. *2016 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 1345–1350, 2016. 2

[5] Yin Biao Guo. 4. research on recommendation algorithm based on heterogeneous graph neural network. 2024. 2

[6] Tejaswini H, Manohara Pai M M, and Radhika M. Pai. Knowledge graph for aquaculture recommendation system. *2021 IEEE Mysore Sub Section International Conference (MysuruCon)*, pages 366–371, 2021. 3

[7] Kibeom Lee and Kyogu Lee. Escaping your comfort zone: A graph-based recommender system for finding novel recommendations among relevant items. *Expert Syst. Appl.*, 42:4851–4858, 2015. 2

[8] Sergio Oramas, Vito Claudio Ostuni, T. D. Noia, Xavier Serra, and Eugenio Di Sciascio. Sound and music recommendation with knowledge graphs. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 8:1 – 21, 2016. 2

[9] The pandas development team. pandas-dev/pandas: Pandas, Feb. 2020. 4

[10] Yeming Qiu and Qi Qi. 1. e-commerce recommendation algorithm of agricultural products based on knowledge graph. 2024. 1

[11] Abhinav Sharma, Preksha Agrawal, and Surendra Kumar Keshari. Recommender system in e-commerce. *International Journal of Innovative Science and Research Technology (IJISRT)*, 2024. 2

[12] Dharahas Tallapally, John Wang, Katerina Potika, and Magdalini Eirinaki. 4. using graph neural networks for social recommendations. *Algorithms*, 2023. 1

[13] Enzhi Zhang, Jinan Fiaidhi, and Sabah Mohammed. Social recommendation using graph database neo4j: Mini blog, twitter social network graph case study. 2017. 3