Spotify Playlist Continuation

Anurag Agarwal[1] **Shivam Agarwal**[2] **Divyam Agrawal**[3]
University Of Washington
{anurag96,shivam96,diagra}@uw.edu

1 Introduction

In today's world, the consumption of music has undergone a profound transformation, driven by the spread of streaming platforms like Spotify. With access to a vast repository of songs and user-generated playlists, Spotify has become a central hub for music enthusiasts to discover, curate, and share their favorite tracks. Amidst this abundance of musical content, the challenge of navigating through the endless array of songs to find personalized recommendations has become increasingly complex. Recognizing the significance of enhancing music discovery experiences, our project aims to explore the domain of playlist continuation, leveraging insights from the Spotify Million Playlist Dataset (MPD) to predict new songs for playlists with precision and relevance.

By analyzing user behaviors, preferences, and historical interactions with music content, these algorithms strive to generate tailored recommendations that resonate with individual tastes. Our project builds upon this foundation, drawing inspiration from recent advancements in recommendation systems and leveraging state-of-the-art methodologies to tackle the challenge of playlist continuation. Through an exploration of existing literature and research, we aim to glean insights into effective strategies for optimizing recommendation algorithms and enhancing music discovery experiences for Spotify users. At the heart of our approach lies the Spotify Million Playlist Dataset (MPD), a rich repository of user-generated playlists spanning diverse genres, moods, and themes. By tapping into the wealth of data captured within the MPD, we endeavor to uncover underlying patterns, trends, and contextual associations that inform playlist creation and curation. Our methodology entails a two-stage architecture for playlist continuation, consisting of candidate generation and re-ranking stages, each meticulously designed to refine precision and relevance in song recommendations. Through rigorous experimentation and evaluation, our goal is to craft a recommendation system that not only excels in predicting new songs for playlists but also elevates the overall music discovery journey for Spotify users worldwide.

2 Related work

The exploration of music recommendation systems has yielded valuable insights into enhancing the precision and relevance of track suggestions within playlists. (Bonnin and Jannach, 2013) conduct a comprehensive examination of various algorithms, including K-nearest neighbors (KNN) [4], Cosine similarity, and association rules [5], to optimize playlist recommendations along with touchbasing on the novel "Collocated artists - greatest hits". The topics of KNN [4] , Cosine Similarity and association rules [5] are few topics discussed in the papers which has been discussed in class. This foundational work underscores the significance of selecting methodologies that effectively capture the sequential patterns inherent in music consumption behaviors. (Abdollahpouri et al., 2023) propose an innovative approach utilizing the maximum flow problem to generate calibrated recommendations within recommender systems. By aligning closely with a user's historical interactions across different item categories, their method optimally balances the consideration of all relevant items, thereby significantly enhancing recommendation quality.

(Zamani et al., 2019) extend this exploration by analyzing strategies from the ACM RecSys Challenge 2018, specifically focusing on automatic playlist continuation. Their research intersects with sequential recommendation, emphasizing the importance of suggesting tracks that seamlessly complement an existing playlist. Furthermore, their suggestions to incorporate natural language processing (NLP) techniques and sequence-specific playlist characteristics into recommendation models offer promising avenues for future research.

The three papers are interconnected in their pursuit of delving into novel methodologies to improve recommendations for playlist songs. While (Bonnin and Jannach, 2013) lay the groundwork by exploring various algorithmic approaches, (Abdollahpouri et al., 2023) introduce an innovative method to optimize recommendation quality, and (Zamani et al., 2019) extend these insights by focusing on automatic playlist continuation. Despite differences in datasets, the overarching goal remains consistent: to enhance

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

```
{
    "name": "musical",
    "collaborative": "false",
    "pid": 5,
    "modified_at": 1493424000,
    "num_albums": 7,
    "num_tracks": 12,
    "num_followers": 1,
    "num_edits": 2,
    "duration_ms": 2657366,
    "num_artists": 6,
    "tracks": [
        {
            "pos": 0,
            "artist_name": "Degiheugi",
            "track_uri": "spotify:track:7vqa3sDmtEaVJ2gcvxtRID",
            "artist_uri": "spotify:artist:3V2paBXEOZIAhfZRJmo2j",
            "track_name": "Finalement",
            "album_uri": "spotify:album:2KrRMJ9z7Xjoz1Az406UML",
            "duration_ms": 166264,
            "album_name": "Dancing Chords and Fireflies"
        },
        ...
    ]
}
```

Listing 1: **JSON File Structure**

music recommendation systems through sophisticated algorithmic techniques. The papers underscore the importance of preprocessing the datasets for effective implementation and computation of the algorithms, a key aspect addressed explicitly by Abdollahpouri et al. (2023), which is crucial for successful model deployment and performance improvement.

3 Data Section

3.1 Data Summary

The primary dataset for this project is the **Million Playlist Dataset (MPD)**, initially introduced in 2018 by Spotify for the RecSys Challenge. This dataset is compiled from a vast collection of over 4 billion public playlists on Spotify, offering a representative sample of **1 million playlists**. It contains more than **66,346,428 tracks**, out of which there are **2,262,292 unique tracks**. Additionally, there are **734,684 unique albums** and **295,860 unique artists** represented in the dataset. The playlists were sourced from Spotify users in the United States between January 2010 and November 2017.

Some other statistics from the dataset include:

- Number of unique playlist titles: **92,944**
- Number of unique normalized playlist titles: **17,381**
- Average playlist length (tracks): **66.35**

Additionally, this project will leverage Spotify's API for artists and tracks. Through API queries, we aim to gather comprehensive data on various aspects, such as audio features for each track, track popularity, artist popularity, associated genres for the songs, and other relevant details.

3.2 Data Processing

The data was divided into splices of **1000** playlists stored in individual JSON files, resulting in over **1000 JSON files** to process, each following a specific structure given in **Listing 1**.

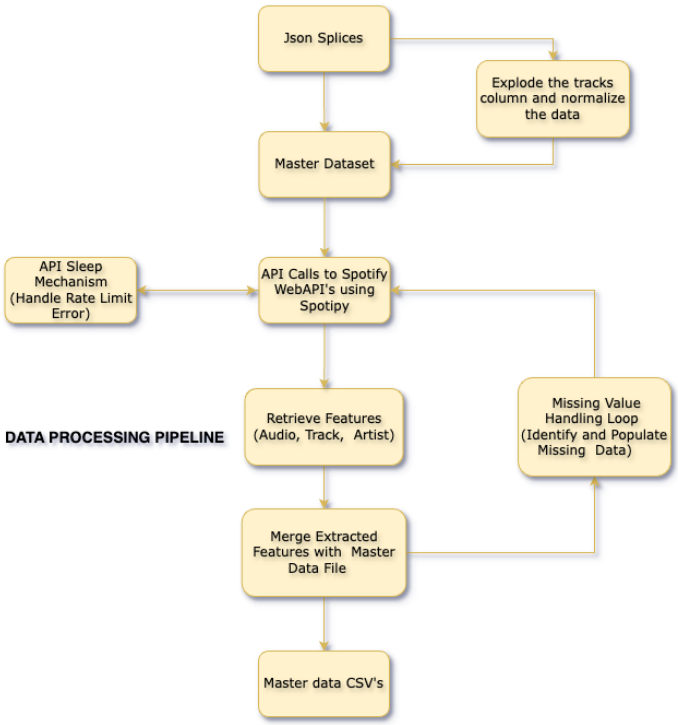


Figure 1: **Data Processing Pipeline**

To handle this large volume of data more efficiently, we decided to process the data by implementing a data pipeline **(Figure 1)**and store it in smaller chunks using multiple CSV files. The initial step of our data pipeline involved addressing the grouped nature of the data by playlists. To ensure compatibility with our modeling objectives, we needed to break down these groupings, a process commonly referred to as data explosion, and normalize the format. This normalization was crucial for subsequent model usage. To achieve this, we developed and implemented a data pipeline.

Once we had the formatted data, our next step was to extract the audio, tracks, and artists' features. To accomplish this, the second step of the pipeline created lists of unique artists and tracks, which were stored in separate CSV files. These lists were then utilized to populate the audio features data and track-related data, respectively, for each chunk.

For data population, we relied on **Spotify's WebAPI's** and specifically the Python package called **"Spotipy"**. This library facilitated API calls without additional development efforts. The subsequent step in the pipeline used the artists and tracks lists for each chunk to call the API and retrieve the desired features, including audio features such as '**acousticness**', '**danceability**', '**duration_ms**', '**energy**', '**instrumentalness**', '**liveness**', '**loudness**', '**mode**', '**speechiness**', '**tempo**', '**type**', '**valence**', artist features like "**popularity**", "**genres**", and track features such as "**track release date**" and "**popularity**".

During this step we encountered API rate limit error.To handle the rate limit error, we implemented an **API sleep mechanism** which introduced a lag of **3 seconds** between subsequent API calls which error out. Although this increased the overall compute time, it ensured smooth API operation and data retrieval.

After obtaining these details for each chunk, the pipeline merged the extracted features with the corresponding master file chunk. This merging process utilized Track URI and Artist URI to merge the dataframes. Additionally, we implemented a missing value loop to identify and populate missing data using the same API call if necessary.

The processed data for each chunk was then stored in a separate CSV file, following a consistent naming convention to identify the chunk. This approach allowed us to work with smaller, more manageable file sizes, improving performance, manageability, fault tolerance, and scalability. When needed, we could read and combine the individual CSV files into a single data structure for further analysis or processing.

4 Methodology

4.1 Baseline Song Recommendation System Model

The baseline song recommendation system leverages content-based filtering, collaborative filtering, and natural language processing to recommend songs. This condensed report outlines the key algorithms and mathematical approaches used in the system.

Preprocess Data: Standardize numerical features and tokenize text descriptions, then train a Word2Vec model to create vector representations of song descriptions.

Compute Similarity: Use cosine similarity to measure the similarity between song vectors based on their numeric and text-derived features.

Recommend Songs: For a given playlist, calculate similarity scores between songs in the playlist and other songs, and recommend the top n songs with the highest similarity.

The final recommendation score combines the results from content-based and collaborative filtering:

$$\text{score}(x_k) = \lambda \cdot \text{sim}_{\text{content}}(x_k, p) + (1 - \lambda) \cdot \text{sim}_{\text{collab}}(x_k, p)$$

where λ is a tunable parameter balancing the two similarity measures.

Algorithm 1 Song Recommendation System

procedure PREPROCESSDATA($data$)
    Standardize numerical features in $data$
    $data \leftarrow$ Tokenize and lowercase text descriptions
    $word\_vectors \leftarrow$ Train Word2Vec($data$.descriptions)
    $data$.description_vec $\leftarrow$ Average Word Vectors($word\_vectors$)
    return $data$
end procedure
procedure TRAINWORD2VEC($texts$)
    $model \leftarrow$ Word2Vec($texts$, parameters)
    return $model$
end procedure
procedure RECOMMENDSONGS($data, playlist\_id, top\_n$)
    $playlist\_songs \leftarrow$ Filter songs by $playlist\_id$
    $other\_songs \leftarrow$ Filter other songs
    for $song$ **in** $other\_songs$ **do**
        $similarity \leftarrow$ ComputeSimilarity($playlist\_songs, song$)
        $song.similarity \leftarrow similarity$
    end for
    $recommendations \leftarrow$ Sort $other\_songs$ by $similarity$ descending
    return $recommendations[: top\_n]$
end procedure

4.2 Two-Stage Song Recommendation System

Stage 1: Candidate Generation

The main goal of the first stage is to quickly retrieve candidate song set with sufficiently high recall. Given a high-dimensional song feature matrix $A \in \mathbb{R}^{m \times n}$, we apply Truncated Singular Value Decomposition (SVD) to reduce its dimensionality:

$$A \approx U \Sigma V^T$$

where $U \in \mathbb{R}^{m \times k}$ and $V \in \mathbb{R}^{n \times k}$ contain the left and right singular vectors, $\Sigma \in \mathbb{R}^{k \times k}$ is a diagonal matrix with the largest k singular values, and k is the number of retained components. This reduction captures the most significant latent structures in the song data.

SVD is chosen for its mathematical robustness in providing the optimal low-rank approximation of the original matrix, minimizing the reconstruction error $\|A - U\Sigma V^T\|_F$ in the Frobenius norm. Additionally, it efficiently handles the sparsity commonly present in user-item interaction matrices without necessitating data imputation.

While alternatives such as PCA, t-SNE, and UMAP offer certain advantages, SVD is selected for its global structure preservation and computational efficiency, essential for real-time recommendation systems. Unlike t-SNE and UMAP, which focus more on capturing complex local structures and are generally more suited for data visualization, SVD maintains interpretability and scalability. Furthermore, clustering techniques like k-means and collaborative filtering approaches often require more complex data handling or do not capture the fine-grained similarities as effectively as nearest neighbor searches within the context of our application.

Nearest Neighbor Search in Reduced Space

Following the dimensionality reduction, we define a metric space $(U\Sigma, d)$, where d represents the Euclidean distance. We utilize this space to execute a nearest neighbor search:

$$\text{NN}(q) = \underset{x \in U\Sigma}{\operatorname{argmin}}\, d(q, x)$$

Here, q represents a query point in the reduced space $U\Sigma$, and x denotes the dataset points.

The nearest neighbor search is pivotal for preserving the local structure of the data, crucial for identifying songs with similar musical characteristics or user preferences. This method's adaptability to various data types and similarity measures ensures it is tailored to the specific metrics relevant to our application.

Algorithm 2 Generate Candidate Songs

function GENERATECANDIDATES($dataMatrix$)
    $svd \leftarrow$ TruncatedSVD($n\_components = 50$)
    $itemFactors \leftarrow svd$.fit_transform($dataMatrix$)
    $nn \leftarrow$ NearestNeighbors($n\_neighbors = 50, algorithm =' ball\_tree'$)
    nn.fit($itemFactors$)
    return nn.kneighbors($itemFactors, return\_distance = False$)
end function

Stage 2: Ranking and Filtering

Given the candidates retrieved by the first stage, the goal of the second stage is to accurately re-rank these candidates maximizing the accuracy at the top of the recommended list. Since the candidate set is small, second stage model can be more expensive and trade off efficiency for accuracy. We thus focus on pairwise interactions here and develop a model that jointly maps (playlist, song) pairs to relevance scores

For each candidate c, the feature vector $\mathbf{f}_c$ is a concatenation of collaborative and content-based features:

$$\mathbf{f}_c = \begin{bmatrix} \mathbf{f}_{c,\text{collab}} \\ \mathbf{f}_{c,\text{content}} \end{bmatrix}$$

where $\mathbf{f}_{c,\text{collab}}$ represents user-item interaction features and $\mathbf{f}_{c,\text{content}}$ includes song-specific attributes such as genre and tempo.

Using a XGBoost model, we predict the relevance score s_c for each candidate. The objective function optimized during training is given by:

$$\text{minimize} \sum_{i=1}^{n} l(y_i, s_{c_i}) + \Omega(f)$$

where l is the loss function quantifying prediction errors, and Ω represents the regularization term to prevent overfitting.

Combining both collaborative and content-based features offers a comprehensive understanding of each song, enhancing the prediction of user preferences. This integrated approach not only facilitates precise similarity assessments but also significantly ameliorates the cold start problem, utilizing established patterns of similar users or items to infer preferences for new users or songs.

XGBoost is chosen for its computational efficiency and robustness, crucial for managing the vast datasets typical in recommendation systems. Key attributes include:

- **Efficiency:** XGBoost's block structure optimizes memory access patterns, enhancing computational speed. Support for parallel processing further accelerates both training and prediction phases.
- **Adaptability to Data:** The model effectively handles diverse feature types and is resilient to missing data. It autonomously manages feature interactions, which diminishes the necessity for intricate manual feature engineering.
- **Handling of Class Imbalance:** XGBoost incorporates mechanisms to effectively manage class imbalances, a common challenge in recommendation systems.

While alternatives such as Random Forests and deep learning approaches offer certain benefits, XGBoost outperforms Random Forests in bias-variance tradeoff optimization and is more resource-efficient than deep learning models. Deep learning, although capable of capturing complex patterns, demands substantial computational resources and extensive tuning, rendering XGBoost a more pragmatic choice for our application.

The songs are then sorted according to the scores given by the XGBoost model to finalize the ranking.

Algorithm 3 Re-rank Candidate Songs using Machine Learning Models

function RERANKCANDIDATES($candidates$, $itemFeatures$, $userProfile$)
    $features \leftarrow$ ExtractFeatures($candidates, itemFeatures, userProfile$)
    $model \leftarrow$ LoadModel($'xgboost\_model'$)
    $scores \leftarrow model$.predict($features$)
    $rankedCandidates \leftarrow$ SortByScore($candidates, scores$)
    return $rankedCandidates$
end function
function EXTRACTFEATURES($candidates$, $itemFeatures$, $userProfile$)
    $featureList \leftarrow []$
    for all $candidate \in candidates$ **do**
        $collabFeatures \leftarrow$ CollaborativeFeatures($candidate, userProfile$)
        $contentFeatures \leftarrow itemFeatures[candidate]$
        $combinedFeatures \leftarrow collabFeatures \oplus contentFeatures$
        $featureList$.append($combinedFeatures$)
    end for
    return $featureList$
end function

4.3 Cold Start Problem

Dealing with the cold start problem in playlist recommendations presents significant challenges, particularly when minimal metadata is available. Our approach leverages text vectorization and popularity metrics to generate recommendations for new playlists, which lack historical user interaction data.

A novel method is employed that utilizes playlist names and song metadata to mitigate the limitations associated with the cold start scenario.

Textual metadata is converted into a semantic vector space using a pre-trained Word2Vec model. This conversion allows us to capture the thematic essence of both playlists and songs based on their textual descriptions.

- **Playlist Names:** Each playlist name is transformed into a thematic vector, $\mathbf{t}_p = \text{Word2Vec}(n_p)$, where n_p is the name of the playlist p.
- **Song Metadata (Names and Album names):** For each song x_i, the names and album names are combined and vectorized:

$$\mathbf{s}_i = \gamma \cdot \text{Word2Vec}(n_i) + \delta \cdot \frac{1}{|d_i|} \sum_{k=1}^{|d_i|} \text{Word2Vec}(w_{ik})$$

  where n_i and d_i represent the name and album name of the song, respectively, and w_{ik} are the words in d_i. The factors γ and δ balance the influence of the song name and album name in the vector representation.

5 Evaluation

5.1 Recommendation Accuracy

The evaluation of our playlist recommendation system was conducted on a diverse test set, specifically designed to simulate a variety of real-world scenarios. The test set comprises ten groups, each containing 1,000 playlists. These groups represent distinct playlist completion tasks, ranging from cold start scenarios to playlists with extensive initial information. The tasks are as follows:

1. Predict tracks for a playlist given its title only.
2. Predict tracks for a playlist given its title and the first track.

3. Predict tracks for a playlist given its title and the first 5 tracks.
4. Predict tracks for a playlist given its first 5 tracks (no title).
5. Predict tracks for a playlist given its title and the first 10 tracks.
6. Predict tracks for a playlist given its first 10 tracks (no title).
7. Predict tracks for a playlist given its title and the first 25 tracks.
8. Predict tracks for a playlist given its title and 25 random tracks.

This partitioning aims to test the model's capability to handle various stages of playlist creation, from inception to playlists that are nearly complete. As discussed above, to promote simplicity and improve generalization we train a single model for all playlists. To achieve this, we create a validation set with nearly identical playlist length distribution to the test set. For each start test group we randomly sample playlists from the MPD that have the same length and partition them as per test test.

To accurately assess the performance of our recommendation system, we utilized the following metrics:

- **R-Precision**: Measures the ratio of relevant items retrieved to the total number of relevant items in the test set.
- **Normalized Discounted Cumulative Gain (NDCG)**: Evaluates the ranking quality of the recommendations, particularly the placement of relevant tracks higher on the list.

5.2 Computational Efficiency

The evaluation of our playlist recommendation system extends beyond the accuracy of its outputs to include the efficiency of its computational processes. To quantitatively assess recommendation efficiency, we conducted a comparative analysis of computation times between two distinct models under identical system conditions. This comparison involved analyzing the differential impacts of employing Python versus PySpark, as well as the effects of various file formats on the performance. Through this approach, we aimed to identify the configurations that optimize processing speed without compromising the quality of the recommendations provided.

5.3 User Research

User research constituted a vital component in evaluating the practical effectiveness of our model, which was especially crucial in the context of this project. To ascertain the real-world relevance and applicability of our recommendation system, a survey was conducted involving active Spotify users. The user research was circulated among active Spotify users online via a Google Form, and their responses were used to assess the real-world validity of our model (both baseline and enhanced). The aim was to gather a diverse range of participants to ensure comprehensive coverage of different musical tastes and preferences. A large sample size enhances the statistical validity of the findings, enabling reliable conclusions about the performance and usability of the recommendation system. Therefore, concerted efforts were made to recruit a substantial number of Spotify users for the survey. In addition to targeting university students for a younger audience, responses were collected from various age groups and demographics. Utilizing WhatsApp groups and social media pages engaged university students, while platforms like Reddit, Twitter, and email were employed to diversify the response pool.

Survey participants received four songs from different playlists in the test set. They were asked to consider four track options and decide if they belonged in the given playlist. These tracks were part of the model's top 4 predictions and helped in measuring real-world accuracy. For instance, if the participant felt only 2 tracks should be in the playlist, the accuracy was 50%. The survey asked users to predict tracks for three playlists to ensure robust insights and encourage participation by not consuming too much time. Once responses were gathered, the average accuracy of the model's predictions for the next 4 songs in the playlist was calculated and compared with user opinions. The real-world accuracy was compared between the baseline and enhanced models for the same users across the two surveys. Efforts were made to recruit many Spotify users, facilitating a thorough assessment of the model's efficacy in meeting diverse user needs and preferences.

6 Results

6.1 Accuracy Metrics

The accuracy of the playlist recommendation system was evaluated using two key metrics: R-Precision and Normalized Discounted Cumulative Gain (NDCG). R-Precision measures the ratio of relevant items retrieved to the total number of relevant items, while NDCG evaluates the ranking quality by assigning higher scores to relevant items ranked higher in the recommendation list.

The results comparing the baseline model and the enhanced two-stage model are shown in Table 1. The enhanced model demonstrates noticeable improvements in both R-Precision and NDCG scores compared to the baseline.

Metric	Baseline	Enhanced Two-Stage Model
R-Precision	0.1048	0.1621
NDCG	0.3054	0.3571

Table 1: **Accuracy metric comparisons between baseline and enhanced models**

The enhanced model achieves an R-Precision score of 0.1621, a 54.5% improvement over the baseline's 0.1048. This indicates that the enhanced model retrieves a higher proportion of relevant songs for the playlists in the test set.

Additionally, the NDCG score of 0.3571 for the enhanced model represents a 17.0% increase compared to the baseline's 0.3054. This higher NDCG demonstrates the enhanced model's ability to rank relevant songs higher in the recommendation list, thereby improving the overall ranking quality and user experience.

These accuracy metric enhancements underscore the efficacy of the two-stage architecture and the incorporation of advanced techniques, such as truncated SVD for candidate generation and the XGBoost model for re-ranking. By leveraging both collaborative and content-based features, the enhanced model achieves superior accuracy in predicting and ranking relevant songs for playlists.

Playlist 1	**Playlist 2**	**Playlist 3**
1. Drake - Sneakin	1. Stone & Van Linden - Summerbreeze	1. London Philharmonic Orchestra - Symphony No. 40 in G Minor, K. 550: Allegro molto
2. Logic - The Incredible True Story	2. Havana Brown - We Run The Night	2. London Philharmonic Orchestra - Requiem, K. 626: Lacrimosa dies illa
3. Travis Scott - Birds In The Trap Sing McKnight	3. Calvin Harris - Sweet Nothing	3. Elisabeth Ganter - Concerto in A Major for Clarinet and Orchestra, K. 622: II. Adagio
4. Big Sean - I Decided	4. David Guetta - Titanium	4. Tbilisi Symphony Orchestra - Symphony No.25 In G Minor, K. 183 I. Allegro Con Brio
5. The Weeknd - Starboy	5. Calvin Harris - Feel So Close	5. Tbilisi Symphony Orchestra - Requiem Mass In D Minor, K. 626 : II. Dies Irae
Model Recommendations		
Drake - Fake Love	Rihanna - We Found Love	London Philharmonic Orchestra - String Quintet No.4 in G Minor, K.516: I. Allegro
Migos - Bad and Boujee	Swedish House Mafia - Don't You Worry Child	Takács Quartet- String Quintet No.3 in C Minor, K.515: III. Andante
Post Malone - Congratulations	Calvin Harris - Summer	Columbia Symphony Orchestra - Serenade in G major, K. 525 i. Allegro

Figure 2: **Model Predictions for given playlist**

Examples of test playlists continuations are shown in Figure 2. It shows the first five songs contained in each playlist and the top three suggestions produced by the model.It can be inferred that that the model is able to accurately capture and continue the playlist genre – from Hip-Hop/Pop in the first playlist, to EDM in the second playlist, and classical pieces in the last one. The recommendations are also diverse and feature songs by different artists.

6.2 Efficiency Metrics

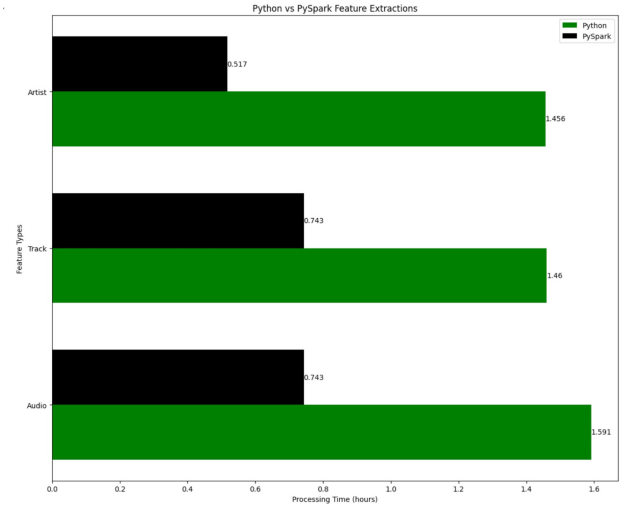
6.2.1 Pre-processing using Python Vs PySpark

Given the original dataset size of **33.54 GB**, we recognized the need for an efficient and scalable approach to preprocess the data. Initially, the baseline implementation relied on Python for preprocessing, which took over **6 hours** to transform the dataset. Seeking to enhance processing efficiency and scalability, we transitioned to PySpark for the preprocessing step. In Figure 3(b), we can observe a significant reduction in processing time to approximately **2.8 hours** , resulting in a notable speedup of almost **53.33%**. This improvement underscores PySpark's capacity for parallel processing across multiple nodes, facilitating the effective utilization of computational resources.

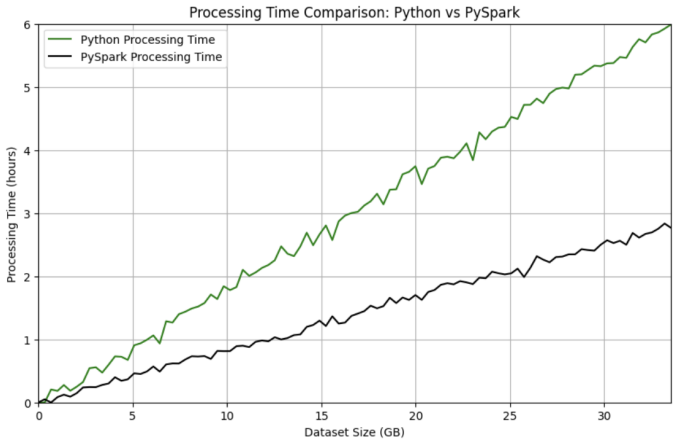
Feature Type	Processing Time (hours)	
	Python	PySpark
Audio	3.4	1.591
Track	1.46	0.743
Artist	1.456	0.517
Total	6.316	2.851

Table 2: **Comparison of Processing Times for Feature Extraction and Formatting**

By parallelizing data processing tasks, PySpark minimized the time required to process each record, effectively utilizing the available computational resources and reducing the overall processing time. Additionally, PySpark's in-memory computation capabilities optimized data processing by efficiently handling intermediate results, further contributing to the observed improvement in processing efficiency. In particular, PySpark demonstrates significant reductions in processing time across all feature categories, with the most substantial improvements observed in the extraction and formatting of artist and audio features.



(a) Feature-based processing time - Python Vs PySpark



(b) Overall processing time - Python Vs PySpark

Figure 3: **Comparison of processing times in Python and PySpark**

6.2.2 Comparison between CSV and Parquet in PySpark Processing

In our preprocessing pipeline for the Million Playlist Dataset (MPD), we witnessed the efficiency gains achieved by transitioning from Python to PySpark. Following this improvement, we further investigated the impact of different file formats, specifically CSV and Parquet, on processing time and resource utilization.

Metric	CSV Processing	Parquet Processing
Initial Dataset Size	33.54 GB(JSON)	33.54 GB(JSON)
Converted Dataset Size	∼20 GB	∼13 GB
Processing Time	6 hours	2.8 hours
Resource Utilization	Moderate to high memory usage and disk I/O	Lower memory usage and disk I/O

Table 3: **Comparison between CSV and Parquet Formats**

From Table 3, we can see that Parquet file format exhibited better performance compared to CSV, particularly due to its columnar storage format. Unlike CSV, which stores data in a row-based format, Parquet organizes data by columns, allowing for more efficient access to specific attributes during processing. Since our preprocessing involved operations on attributes such as **artist_uri**, **track_uri**, and **album_uri**, which are more suited for columnar processing, Parquet's structure provided significant advantages.

Furthermore, Parquet incorporates advanced compression techniques, which further reduced the size of the dataset compared to CSV. This reduction in dataset size not only optimized storage space but also minimized disk I/O operations during processing, contributing to the overall improvement in processing time.

6.3 User Research Assessment Evaluation

For the user research assessment, the Google Form was distributed among over 100 active Spotify users, of which 71 completed the assessment for the baseline model. However, only 57 of these users agreed to participate in the assessment for the enhanced model. Thus, the data from these 57 users was utilized to evaluate the real-world accuracy.

For the baseline model, the mean real-world accuracy was determined to be 46.26%. In contrast, the enhanced model exhibited a mean real-world accuracy of 54.11%, showcasing an increase of approximately 7.85%, as seen in Figure 4.

While these results are promising, it's important to note that they are derived from a subset of 57 individuals. As such, they may not provide a comprehensive representation of our model's predictive capabilities. However, it's worth emphasizing that a sample size exceeding 50 is generally considered sufficient for obtaining a reasonable estimate of the model's performance when deployed among

the general population. Nevertheless, for more robust conclusions, further validation with a larger and more diverse sample size is recommended.

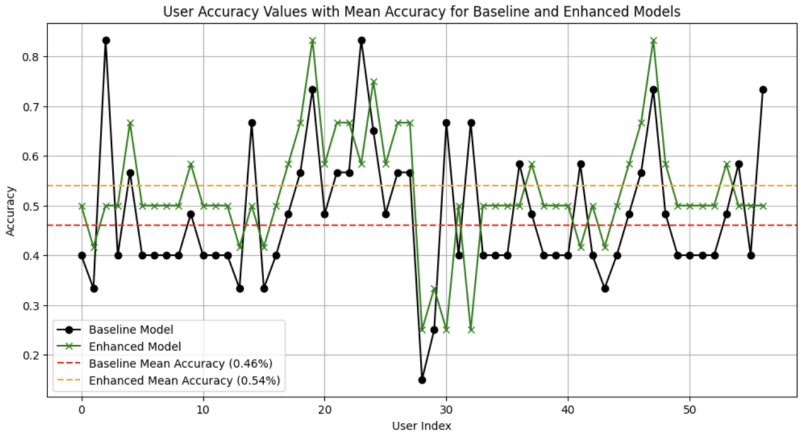


Figure 4: **User Research Assessment Results**

7 Future Work and Limitations

The current implementation of the playlist recommendation system has certain limitations. The implementation of API rate limit handling introduced a delay of 3 seconds between subsequent API calls, increasing the overall compute time. Additionally, the model utilizes the original dataset from the 2018 RecSys Challenge, which might be outdated, and hence the recommendations could be less relevant to current music trends. Furthermore, the playlists in the dataset span diverse genres, moods, and themes, and some of these categories might be underrepresented, leading to potential biases or inaccuracies in the recommendations.

To address these limitations and further enhance the system, several avenues for future work can be explored. Exploring cross-domain recommendations by leveraging signals from other domains like movies, books, etc., could provide more diverse recommendations. Broadening the sample size beyond the current 57 users would obtain a more representative measure of real-world accuracy. Investigating the use of more sophisticated language models like BERT and GPT could better capture semantic and contextual information from playlist and song descriptions. Additionally, evaluating the system on even larger and more diverse datasets spanning more musical cultures and tastes could yield additional insights.

8 Conclusion

We developed a novel two-stage pipeline for recommending new songs to continue playlists on the Spotify platform. By leveraging the Million Playlist Dataset (MPD) and incorporating techniques such as truncated SVD for candidate generation and XGBoost for re-ranking, our system demonstrates significant improvements in recommendation accuracy compared to the baseline model.

The evaluation results, based on metrics like R-Precision and Normalized Discounted Cumulative Gain (NDCG), highlight the enhanced model's ability to retrieve a higher proportion of relevant songs and rank them more effectively within the recommendation list. Additionally, the user research assessment further validates the real-world applicability of our system, with the enhanced model exhibiting a noticeable increase in mean accuracy compared to the baseline model.

Furthermore, we have addressed the computational efficiency aspect by transitioning from Python to PySpark for preprocessing and leveraging the Parquet file format, resulting in substantial reductions in processing time and resource utilization. While our implementation has certain limitations, we have identified several avenues for future work including exploring cross-domain recommendations, investigating more sophisticated language models, and evaluating the system on larger and more diverse datasets spanning various musical cultures and tastes.

Overall, our playlist recommendation system showcases the potential of leveraging advanced techniques and datasets to enhance the music discovery experience for Spotify users. By addressing the challenges of playlist continuation, our work contributes to the ongoing efforts in improving recommendation systems and optimizing user experiences within the music streaming domain.

Contribution

Anurag Agarwal: Conducted the user study, including designing the study protocol, recruiting participants, collecting data, and analyzing the results. Supported on detailing the study design, methodology, and findings. Also performed Exploratory Data Analysis (EDA) to gain insights into the data. Also contributed to baseline model preparation.

Divyam Agrawal: Responsible for modeling tasks, including selecting appropriate machine learning algorithms, training and evaluating models, and optimizing model performance along with creating the solution to the cold start problem. Contributed to the modeling methodology and results sections of the paper.

Shivam Agarwal: Worked on preprocessing and data transformations, including cleaning and preparing the data for modeling. Implemented techniques for handling missing values, feature engineering, and data normalization and data extraction using API for both Python and PySpark implementations.

References

[1] Bonnin and Jannach, (2013) A Comparison of Playlist Generation Strategies for Music Recommendation and a New Baseline Scheme. *AAAI 2013: Intelligent Techniques for Web Personalization and Recommendation*

[2] Abdollahpouri et al., (2023) Calibrated Recommendations as a Minimum-Cost Flow Problem *Search and Data Mining (WSDM '23)*

[3] Zamani et al., (2019) An Analysis of Approaches Taken in the ACM RecSys Challenge 2018 for Automatic Music Playlist Continuation *arXiv:1810.01520v2 [cs.IR]*

[4] T. Cover and P. Hart, (1967) Nearest neighbor pattern classification *IEEE Transactions on Information Theory, vol. 13, no. 1, pp. 21-27*

[5] R. Agrawal, T. Imielinski, and A. Swami, (1993) Mining Association Rules Between Sets of Items in Large Databases *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, pp. 207-216*

[6] Mikolov, T., Chen, K., Corrado, G., Dean, J, (2013) Distributed Representations of Words and Phrases and their Compositionality *26th International Conference on Neural Information Processing Systems (NIPS 2013) (pp. 3111-3119)*