Machine Learning Techniques to Improve Users' Music Listening Experiences

By Samantha Noggle

A Thesis Submitted to the Department of Computer Science in Partial Fulfillment of the Requirements for the Departmental Honors Baccalaureate

Millersville, Pennsylvania

May 2023

This Senior Thesis was completed in the Department of Computer Science. Defended before and approved by the following members of the Thesis committee:

Signed: _____

Stephanie M. Schwartz, Ph.D (Thesis Advisor)

Department chair, Professor of Computer Science

Signed: _____

Chris Cain, Ph.D

Assistant Professor of Computer Science

Signed: _____

Chad M. Hogg, Ph.D

Assistant Professor of Computer Science

Signed: _____

Robert Buchanan, Ph.D

Professor of Mathematics

Abstract

Recommender systems play a crucial role in delivering a personalized user experience. This is especially true for music streaming services like Spotify, which aim to curate music based on listeners' preferences to maintain engagement and stay ahead in a competitive market. This paper proposes a model designed to predict whether or not a user will skip a song based on information available within the user's current listening session. A dataset provided by Spotify for a machine learning challenge is utilized and augmented with features that we propose to help measure the "distance" between a song and songs that have been either listened to or skipped within the same session. We demonstrate that using our model, a prediction can be made on whether a user will skip the final song in their session that is 78.23% accurate. An analysis of feature importance is also presented to better understand which factors might be most influential in predicting this user behavior. The ability to predict whether or not a song will be skipped or played could be utilized in future work to dynamically impact song selection within a listening session to minimize the number of songs being skipped by the user.

Table of Contents

Chapter 1: Introduction

1.1 Recomender Systems

Music streaming platforms implement extensive and complex recommender systems in order to predict exactly what a user wants to listen to. Giving relevant recommendations is extremely important as it improves a user's listening experience and overall satisfaction. In a competitive market with many services vying for each user's time and money, the ability to accurately predict what an individual user wants becomes a critical factor in the success of a music streaming service such as Spotify, Apple Music, or YouTube Music. In this paper, we will focus on Spotify since we are utilizing a dataset provided by the service for a machine learning challenge [6].

Recommender systems are a type of information filtering system that provides content that is most relevant to a user. With a user's previous history and actions within a platform, the system has what it needs to recommend additional content. A variety of techniques are employed such as collaborative filtering, content-based filtering, deep learning, and context-aware filtering. In many cases, multiple strategies are paired together as a hybrid method to make the most accurate predictions.

1.2 Spotify's BART

Given the importance of its recommendations to its success, Spotify has naturally devoted considerable resources to creating its own multi-armed bandit model for content recommendation. A multi-armed bandit algorithm is a reinforcement learning technique named after the one-armed bandit slot machines at casinos [17]. The agent is tasked with multiple "arms" to pull or, rather, choices to make, each with unknown rewards. The "arms" in this case are different songs to recommend, and the rewards are user interactions with the song chosen, such as a "skip" or a "like". Spotify's model is called Bandits for Recommendations as Treatments (BaRT), and it uses three main functions to

make the most accurate song suggestions for users: natural language processing (NLP), collaborative filtering, and raw audio analysis. Natural language processing is used to parse lyrics, user-created playlist names, and web-crawled data from media about a given song. This information helps to categorize tracks by their content and how listeners and critics perceive them. Collaborative filtering is based on the idea that people with similar listening histories will have similar tastes in the new music they hear. It identifies users who have similar preferences based on their past interactions with the system. Collaborative filtering then recommends items that other similar users have liked but the active user has not yet interacted with.

Finally, Spotify analyzes the raw audio of each track by first generating a time-frequency representation of its audio frames. This is then used as input to a convolutional neural network (CNN) as seen in Figure 1.1. CNNs are typically used with visual data, but for this case, it is examining raw audio instead of pixels. From this model, audio features are output such as key, tempo, danceability, and loudness. This method provides a deep understanding of each song despite having a lack of user data in the case of new or unpopular music [7].

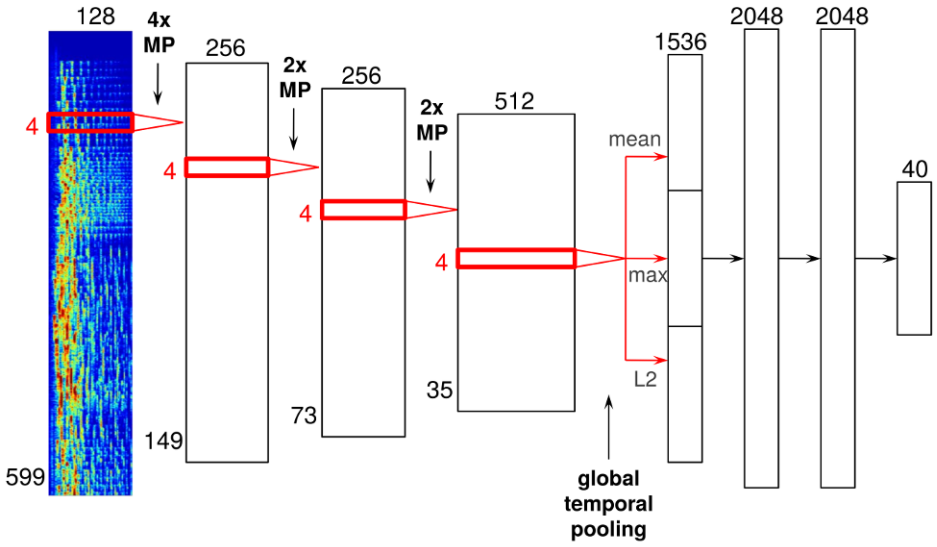

Figure 1.1: A convolutional neural network architecture used by Spotify

These techniques efficiently generate tailored playlists, personalize the home screen, and recommend new music.

1.3 Motivation

When making a recommendation, systems often face the *exploration vs exploitation trade-off* [15]. That is, when a user faces a decision about what to listen to or watch next, they may opt to *explore* new content or *exploit* their existing knowledge of what they know and enjoy. This trade-off occurs at multiple levels in recommendation systems. For example, a system could opt to try to find out more about the user's preferences by presenting them with music the system is not sure they'll like (exploring the user's preferences) or the system could exploit its knowledge about the similarity of a song to others that the user already enjoys. A user could also explicitly opt into exploration by selecting "Discovery Mode" or they might decide to play a tried and true playlist of their own creation.

Given the nature of exploration, both on the part of the user and the system, it is natural that a user will be presented with some songs that they simply do not like and will opt to skip. However, an analysis of user listening behavior showed that an astounding 48.6% of songs on Spotify are skipped before being played until the end and 24.14% are skipped within the first five seconds [12]. This is, of course, more than just users skipping a song that they are unfamiliar with and do not enjoy. It is common for users to skip through the playlists that they have created themselves (and obviously already enjoy) to find what they are most in the mood for during a particular listening session. It is this level of behavior that we are interested in understanding. In this situation, the user's general music taste is irrelevant because they are listening to a collection they have already proclaimed to enjoy. But they are in a particular mood, skipping through songs that they like, but do not want to listen to at the moment. Given this behavior, the most relevant information (and therefore the information we wish to exploit) is limited to the specific session. Our research question, therefore, is whether we can use the session data to accurately predict whether or not a user will skip a particular song. If we can, then this ability could then be used to queue songs that the user actually wants to hear at that moment. Additionally, we wanted to gain a better understanding of which features are most important in guiding

these predictions.

Chapter 2: Related Work

2.1 Session-based predictions

Recommender systems for session-based data must use substantially less information than other methods. Instead of having a user's long-running history, only a few of their interactions with the platform are available. There is no user profile to pull information from. However, for our purpose of fine-tuning a shuffle algorithm on a session-by-session basis, it is ideal that only the current session's data is used for that decision.

2.1.1 Session-based Recommendations with Recurrent Neural Networks

Hidasi et al. [10] propose the importance and novelty of session-based recommendations. It cannot be guaranteed that for every situation, there will be ample user data across more than one session. For example, it may be unsafe, or unrealistic to track users in order to accumulate data across multiple sessions for each user. Also, despite being able to collect data from more than one session, the data may simply not exist and a user could only ever have one or two small sessions on a single platform. Because of this, being able to recommend a user content within the context of their current session becomes quite useful.

In previous works, neighborhood methods such as k-nearest neighbor and matrix factorization have been the most popular solution in session-based recommendations. These methods work by grouping together certain types of items or users. When there is a lack of a user profile to pull from, item-to-item recommendations are commonplace. In this case, items that are similar to what a user had just interacted with will be recommended. Though, this method is simple and only takes into account the user's most recent interaction.

Recurrent neural networks (RNN) are a type of artificial neural network especially fit for this task because they are designed to process sequential data. They function by taking input data at each time step and using both the input data

and the output from the previous time step as inputs for the current time step. This allows the model to maintain a sort of memory that can capture information about the sequence of inputs it has seen so far. Being able to remember more than just the most recent user interaction is a great improvement from primitive item-to-item recommendations. RNNs are also able to handle sequences of arbitrary length, which is perfect for sessions that are never uniform in size.

When implementing an RNN to recommend content, the order in which the user's interactions happened must be preserved. Hidasi et al. aimed to capture how the session changed over time instead of simply recommending items that were similar and outperformed their baselines. Their model was tested on an e-commerce dataset, and a YouTube-like video-sharing platform. In both of these contexts, the user is choosing which next item to click from a selection of recommendations. The framing of this situation is slightly different from the case of whether or not a user will skip a song being played, but it demonstrates the task of only using the context of a session of varying lengths to make future predictions.

2.2 Solutions to Spotify's Sequential Skip Challenge

In 2019, Spotify in collaboration with The ACM International Conference on Web Search and Data Mining (WSDM) released a dataset of roughly 130 million listening sessions. It was released as part of a challenge hosted by AIcrowd [6] to predict if users will skip a track or not based on their behavior within the first half of a given session. During the challenge, the training data was comprised of the first half of the session and the test data (withheld from the participants) was the second half of the session.

2.2.1 Recurrent Neural Networks

The 2nd place submission to Spotify's Sequential Skip Challenge (explained in chapter 3) was achieved by Hansen et al. [8] using two stacked recurrent neural networks. The first RNN was meant to encode the first half of the session, while the second RNN was tasked with making the predictions, taking into account that encoded the first half of the session. This solution is similar to sequence-to-sequence networks which are used for machine translation.

First, for each track, a "track embedding" is created to better understand each song. An encoding of the entire session

is also created to give the model an understanding of how it changes over time and the types of variety that can occur within one. This encoding is created using a sequence of the embedded tracks. Last, a "playback encoding" is created that only encodes the first half of a session. This "playback encoding" can then be put into the "prediction network" which will then be able to make a skip prediction on every song in the second half of the session. This process was on average 64.1% accurate, and 80.7% accurate on the first prediction.

2.2.2 Gradient Boosting Trees

While neural networks are a popular and powerful way to make predictions on user behavior within a session, they are not the only effective methods. Beres et al. [3] was ranked 10th place in the challenge by using gradient-boosting trees (GBT). This solution entailed a great deal of preprocessing and feature engineering as they ended up with 508 features for each track in a session, compared to the initial 29 that Spotify provides, not including metadata.

They first included information regarding the statistics of skips for each track. They used the entire dataset to calculate how popular each track was and also their skip ratios. Because this information may depend on the position of the track within each session, they divided it based on whether the track was in the first half or second half of each session. Additionally, Beres et al. analyzed the degree of heterogeneity within each session. They calculated the distance between each song vector, and the average vector of the entire session's tracks to find outliers. This use of distance metrics to measure the similarities between tracks inspired some of our engineered features. Repeat count, the ratio of unique tracks, and how long a song had been played were also added to their dataset.

2.2.3 Deep Reinforcement Learning

In more recent work, Meggetto et al. [13] proposed a deep reinforcement learning (DRL) approach to this challenge in order to understand why people skip music. Their idea was that while RNNs are extremely efficient solutions, they cannot dynamically understand the user's behavior in the long term and they approach the problem as a static one overall. They proposed that a DRL model can continue to learn with a user and gather a deeper understanding of their behavior, leading to more accurate predictions overall.

The problem is modeled as a Markov Decision Process (MDP), where the state represents a listening session at a specific time, and actions are binary indicators of whether the user will skip the current track or not. A positive reward of 1 is given for correct predictions, and 0 (no reward) for incorrect ones. The agent will make fully informed decisions by having access to its previous actions within the environment as well as all previous points in time. While the majority of this training is online learning, there is also an offline learning procedure with a static dataset in order to pre-train the model. This method of being able to switch between online and offline learning allows the agent to grasp basic skipping trends, and then be fine-tuned to the user's listening patterns.

Chapter 3: Approach

In order to predict whether a listener will skip a song, a random forest classifier was trained to label each final song in a session as being skipped or not. In particular, using Python, *sklearn*'s RandomForestClassifier was trained for this task. Highly accurate solutions already exist [3, 8, 10, 11, 13] that use a variety of deep learning methods however, random forests were chosen for this task because they allow the overall importance of each feature to be measured. A random forest is a supervised machine learning method that is a grouping of decision tree models that each make their own prediction and then take a majority vote. These decision trees are built by recursively splitting the data into subsets to properly categorize it based on each feature in the data. Each node in the tree is a decision on a feature and the goal of the model is to split the data as cleanly and accurately as possible, being able to categorize each record based on its features. Using a random forest instead of a single decision tree reduces overfitting by allowing many different trees to vote on one decision. Each decision tree in the ensemble uses a random set of features and a different subset of training data for this purpose [5].

Spotify's Sequential Skip Prediction challenge had ended before our research project but the training data was still available and useful for exploring our research questions. In the next section, we discuss the data in detail, including

the preprocessing and feature engineering that we performed.

Chapter 4: Data

Our dataset of 200,000 sessions was a subset of the training data from the Spotify Sequential Skip Challenge described previously. Since we were using the original training data as both our training and test datasets, we treated the first half of the sessions (the original training data) as an entire session and made it the goal of our model to predict whether the last song in a session would be skipped or not.

For each track in a listening session, there are two categories of features available in the original data: 1) track features and 2) metadata features. The track features (discussed in Section 4.2) include those output by the CNN shown in Figure 1.1 given the track's time-frequency representation while the metadata features provide context for the session (as discussed in Section 4.3). We also engineered a number of our own features to help to examine our hypotheses (Section 4.4). Because one of our goals for this project was to understand the importance of various features and categories of features in predicting whether a song will be skipped in a session, we include an analysis of feature importance as we discuss the data in the following sections.

session_id	session_position	session_length	track_id_clean	not_skipped	hour_of_day	date	premium
0_00006f66-33e5-4de7-a324-2d18e439fc1e	1	20	t_0479f24c-27d2-46d6-a00c-7ec928f2b539	TRUE	16	2018-07-15	TRUE
0_00006f66-33e5-4de7-a324-2d18e439fc1e	2	20	t_9099cd7b-c238-47b7-9381-f23f2c1d1043	TRUE	16	2018-07-15	TRUE
0_00006f66-33e5-4de7-a324-2d18e439fc1e	3	20	t_fc5df5ba-5396-49a7-8b29-35d0d28249e0	TRUE	16	2018-07-15	TRUE
0_00006f66-33e5-4de7-a324-2d18e439fc1e	4	20	t_23cff8d6-d874-4b20-83dc-94e450e8aa20	TRUE	16	2018-07-15	TRUE
0_00006f66-33e5-4de7-a324-2d18e439fc1e	5	20	t_64f3743c-f624-46bb-a579-0f3f9a07a123	TRUE	16	2018-07-15	TRUE
0_00006f66-33e5-4de7-a324-2d18e439fc1e	6	20	t_c815228b-3212-4f9e-9d4f-9cb19b248184	FALSE	16	2018-07-15	TRUE
0_00006f66-33e5-4de7-a324-2d18e439fc1e	7	20	t_e23c19f5-4c32-4557-aa44-81372c2e3705	FALSE	16	2018-07-15	TRUE
0_00006f66-33e5-4de7-a324-2d18e439fc1e	8	20	t_0be6eced-f56f-48bd-8086-f2e0b760fdee	FALSE	16	2018-07-15	TRUE
0_00006f66-33e5-4de7-a324-2d18e439fc1e	9	20	t_f3ecbd3b-9e8e-4557-b8e0-39cfcd7e65dd	FALSE	16	2018-07-15	TRUE
0_00006f66-33e5-4de7-a324-2d18e439fc1e	10	20	t_2af4dfa0-7df3-4b7e-b7ab-353ba48237f9	FALSE	16	2018-07-15	TRUE
0_00006f66-33e5-4de7-a324-2d18e439fc1e	11	20	t_79570b2a-639a-4ec0-9853-71c5299ac44d	FALSE	16	2018-07-15	TRUE
0_00006f66-33e5-4de7-a324-2d18e439fc1e	12	20	t_69016f19-84aa-40c0-afa9-54404397b7a2	FALSE	16	2018-07-15	TRUE
0_00006f66-33e5-4de7-a324-2d18e439fc1e	13	20	t_59dc3fcd-7aec-4da5-a747-b59b19bab3bb	FALSE	16	2018-07-15	TRUE
0_00006f66-33e5-4de7-a324-2d18e439fc1e	14	20	t_a157ba98-7eae-4e7d-99d5-ed760b2c0978	TRUE	16	2018-07-15	TRUE
0_00006f66-33e5-4de7-a324-2d18e439fc1e	15	20	t_87d95b75-af5c-4ef6-8dc4-cd888ae17cce	FALSE	16	2018-07-15	TRUE
0_00006f66-33e5-4de7-a324-2d18e439fc1e	16	20	t_db945033-3dc5-4a22-8889-c72fd2781299	FALSE	16	2018-07-15	TRUE
0_00006f66-33e5-4de7-a324-2d18e439fc1e	17	20	t_bff5b6c6-6968-41be-b723-7e860b372975	FALSE	16	2018-07-15	TRUE
0_00006f66-33e5-4de7-a324-2d18e439fc1e	18	20	t_bf496d77-9eb2-4fe1-80f7-2c7ecd6f9b8e	FALSE	16	2018-07-15	TRUE
0_00006f66-33e5-4de7-a324-2d18e439fc1e	19	20	t_1051bc37-1a73-4301-812b-f83ae0c9bbcd	FALSE	16	2018-07-15	TRUE
0_00006f66-33e5-4de7-a324-2d18e439fc1e	20	20	t_358c9cce-7a1e-4dd4-81de-206dda80363f	FALSE	16	2018-07-15	TRUE

Figure 4.1: Example of session data using one session and limited to the relevant features

4.1 Preprocessing

We treat each session as a row in our dataset. On average, each session contains 15 tracks. For each track, the data includes variables called "skip_1", "skip_2", and "skip_3" which tell how long a song was listened to before being skipped. In our experiment, we ignored these fields and use a boolean "not_skipped" feature instead. For the last track in a session, the "not_skipped" value was what was learned and predicted by our model.

The session datasets were first stripped of the majority of their metadata. Information regarding how each track ended was removed as it implicitly indicates whether songs have been skipped. The only additional information that was kept was data that could be generalized across the entire session such as the date, hour of the day, and if the user is a premium Spotify user. Then, the last song of each session was isolated into a separate dataset to prevent any of the target song's features from being included in the training process.

Along with the metadata features, each song within a session of the original dataset contains a track ID of the song. The track ID can be used to locate the track features of songs in a separate file. An 8-dimensional acoustic vector that represents the track is also provided. The only preprocessing required for the track features was to change the "Mode" column to numerical values of 1 and 0 rather than "Major" and "Minor", and to normalize the data. These changes were necessary in order to treat the features as a vector for distance calculations.

4.2 Track Features

Each track in a session has 21 numerical features that describe the song and its acoustic structure.

- Duration
- Release Year
- U.S. Pop. Estimate
- Acousticness
- Beat Strength
- Bounciness
- Danceability
- Dynamic Range Mean
- Energy
- Flatness
- Instrumentalness
- Key
- Liveness
- Loudness
- Mechanism

- Mode
- Organism
- Speechiness
- Tempo
- Time signature
- Valence

A majority of these features, specifically duration, release year, key, mode, and tempo are exactly what they seem. However, Spotify includes a few values that are not so straightforward. According to Spotify's API, *danceability* tells if a track is suitable for being danced to based on its tempo, beat strength, and rhythm stability. A track's *valence* tells how positive and euphoric a track is. For example, 'Toxic' by Britney Spears is rated as 0.924, while Lana Del Rey's 'Summertime Sadness' scores a 0.22. Also, *liveness* conveys how likely it is that the track was recorded live, with an audience present. The 8-dimensional acoustic vector is also given, though each value is quite abstract as it represents the song's structure at a lower level.

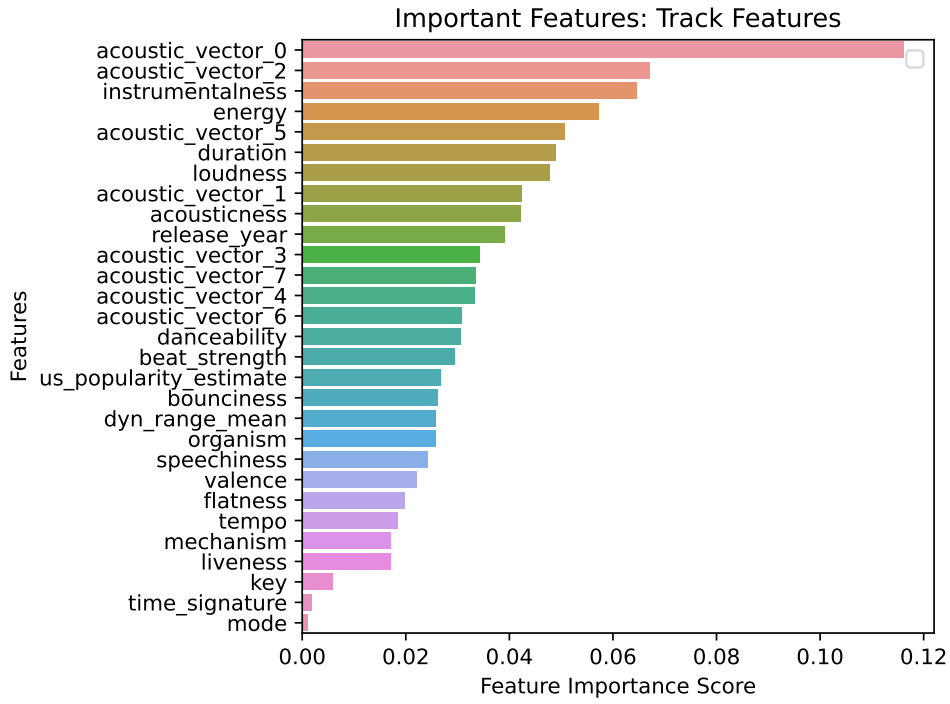

Figure 4.2: Ranking of the features of a model only trained on Spotify track features

A baseline model was trained to predict if the final song of a session will be skipped based only on its track features. Figure 4.2 shows how predictive each of these features were. When making a prediction solely on the structure of a given song, the model may be learning which songs fade out into interludes, or contain a few seconds of silence at the end. The acoustic vector features are too abstract to understand exactly which quality of the track it represents, but

duration as a feature is easy to justify. The longer a song lasts, the more time there is for a user to skip it. This model was only 53.67% accurate at predicting whether the final song would be skipped in a session, so the track features as raw data only perform roughly as well as a random guess.

4.3 Metadata

Additionally, the data for each session includes features providing context for the session. This context can provide information about factors that have been shown to impact user behavior such as the fact that users have been shown to skip less often at times of the day when they are not paying attention to the music (for example, when they are asleep or at work) [12]. The hours in between those typically less attentive times contain more skips. Similarly, users skip more during weekends than weekdays possibly because they are able to pay more attention to their music [12]. The following four features were included in our dataset:

- Month
- Hour of day
- Day of the week
- Spotify premium user

Note that the metadata features exist at the track level in the original dataset but that we generalize them to the session level so that we can attempt to capture some of the behavior described above (like skipping more on weekends). There were some features for which this generalization did not make sense such as how a track ended, if there was a pause before a track, and the type of collection the user is listening to. The first two very obviously apply to a track rather than a session. But the type of collection that a user was listening to may also vary across tracks in a single session.

In order to get a sense of the relative importance of each of these features, we trained our model using only this metadata. Figure 4.3 shows the Feature Importance Scores of each of the four features. The fact that *hour_of_day* and *day_of_week* are more important than *month* seems to support our hypothesis (and others' observations [12]) about there being regular times when users skip more or fewer songs. Whether or not a user is a *premium* Spotify user impacts how many skips they are permitted per hour, so it is not surprising that this feature is relatively important in

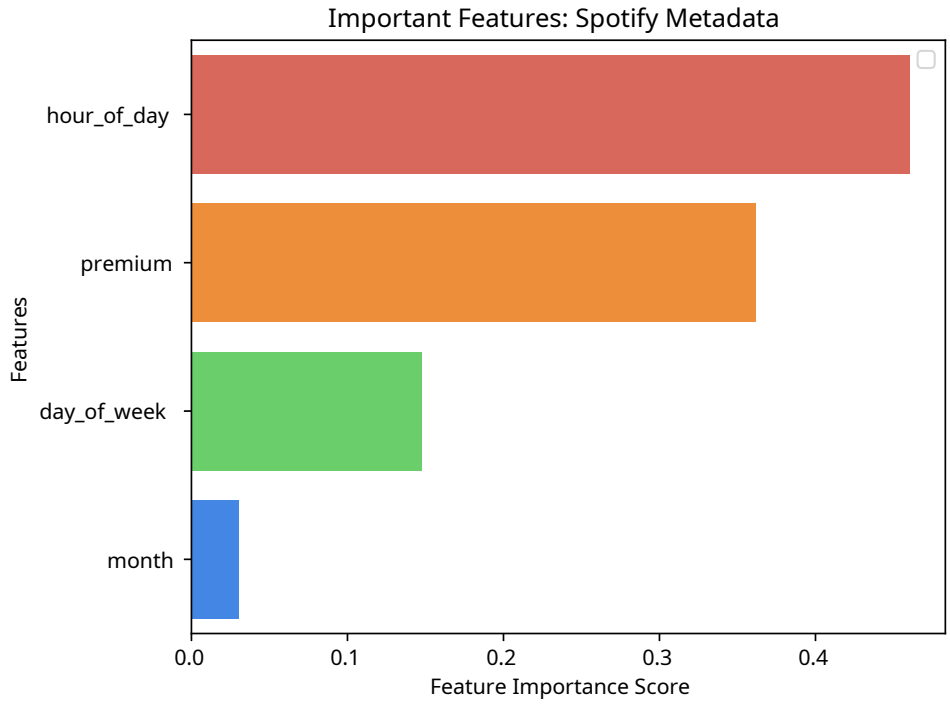

Figure 4.3: Ranking of the features of a model only trained on Spotify metadata

this set. The accuracy of the model trained only on this data is 59.55%. It is interesting that this model performed better overall than the model trained only on the Spotify track features, and shows that skipping behavior may be more closely related to overall behavior patterns than to individual song attributes.

4.4 Feature Engineering

Our primary hypothesis in this project is that user behavior exhibits a degree of consistency within a session. Based on this premise, it is posited that listeners are likely to skip songs that are similar to those they have previously skipped, while also being inclined to listen to songs that are similar to those they have already listened to within the same session. In order to test this hypothesis, two categories of additional features were created regarding 1) the "closeness" of the last song to the previous tracks in a session and 2) contextual features from the session.

4.4.1 Song Similarity Metrics

To evaluate the similarity of songs within a session, we represent the 21 track features in combination with the supplied 8-dimensional acoustic vector as a 29-dimensional vector, which we compare using linear algebra. Because we were unsure which distance measures would be most effective, we calculated several different distance metrics, augmented our dataset, and examined the resulting random forests to learn more about the behavior of these metrics.

The idea of using feature vectors was inspired by the way Spotify compares its millions of tracks by representing them as vectors in a high-dimensional space. To facilitate this process, Spotify's developers created a library called ANNOY, which stands for "Approximate Nearest Neighbors Oh Yeah" [4]. ANNOY is designed to handle large-scale data sets efficiently and uses a tree structure for this purpose. The data is split recursively into smaller sections using randomly placed hyperplanes, with each node in the binary tree representing a part of the data. The two children of each node represent a further subdivision of the parent data. By randomly generating multiple trees out of the data, the nearest neighbors can be found using the union of all trees. This allows ANNOY to perform approximate nearest neighbor searches quickly and accurately.

Spotify uses ANNOY to make specific recommendations for a user based on their previous favorites, playlists, etc. Our problem is more specific, given that we want to model the user's behavior in a single session to find out what they're interested in listening to at that moment. We hypothesized that if a song is similar (or "close") to other songs the user has listened to in a session, it is more likely that they will want to listen to the song. Similarly, if a song is closer to other songs the user has skipped in this session, the user is more likely to skip the song. There are many different ways to mathematically express "closeness" between vectors, so thirteen different features were proposed and calculated so that we could evaluate their relative effectiveness in a random forest model.

The first distance calculation used was *Euclidean distance* which is defined as the square root of the sum of the squared differences between the elements of two vectors [16]. It is a common metric that measures the straight-line distance between two points in N-dimensional space. *Manhattan distance* is another metric that measures the distance between two vectors. It is the sum of the absolute differences between the elements of both vectors [2]. Instead of a straight line, this calculation gives the distance by simulating movement along a grid, only being able to move horizontally or

vertically. It can be represented as follows:

$$d_{Manhattan}(p, q) = \sum_{i=1}^{n} |p_i - q_i|$$

where p and q are the two points being compared, and n is the number of dimensions in the space.

We also utilized the *angle between two vectors*, calculated using the cosine of the angle between the vectors, which is the dot product of the vectors divided by the product of their magnitudes [9]. The smaller the angle is, the more similar the vectors are.

Each of these metrics was used to find the distance between the final song (the one we are trying to predict whether the user will skip) and the most recently played song, the most recently skipped song, the average played song vector, and the average skipped song vector. The average vectors were utilized to represent the aggregate of the previously played or skipped songs in the session as a single vector. In addition to these metrics, a *k-d tree* [18] was used to find the nearest neighbor of the last song played.

Table 4.1 lists the distance metrics that were calculated and added to the dataset for each session. Given that the first four metrics were calculated with Euclidean, Manhattan, and Angle, this resulted in twelve separate features. With neighborSkipped, there were thirteen total features.

Table 4.1: Distance Metrics

Name	Description
AvPlay	Distance between the final song and the non-skipped average vector, calculated with Euclidean, Manhattan, or Angle
AvSkip	Distance between the final song and the skipped average vector, calculated with Euclidean, Manhattan, or Angle
LastPlay	Distance between the final song and the last non-skipped song, calculated with Euclidean, Manhattan, or Angle
LastSkip	Distance between the final song and the last skipped song, calculated with Euclidean, Manhattan, or Angle
neighborSkipped	A boolean indicating whether the final song's nearest neighbor was skipped

Notice that we are trying to capture and understand the relative importance of both the aggregate behavior over the session (the averages of played and skipped songs) and the most recent behavior (the last skipped or played song). For sessions in which all songs were only skipped or played, the distance between the final song and that empty category was recorded as -1.

When considering these similarity metrics, we knew that there would be a high degree of correlation between the Euclidean, Manhattan, and Angle features but we wanted to explore the results from the random forest to see if one category of distance measure tended to perform better overall. We utilized all thirteen of the features collectively in order to train a random forest model. When examining the importance of the features as shown in Figure 4.4, the Manhattan metrics ranked the highest relative to the Euclidean and Angle metrics. This aligns with the findings of Aggarwal et al. [1] who demonstrate that in high dimensions, Manhattan is preferable compared to Euclidean.

Given this analysis, we utilized the Manhattan metrics (*manAvPlay*, *manAvSkip*, *manLastPlay*, and *manLastSkip*), along with *neighborSkipped*, in our final dataset. In terms of feature importance within this set, the results are shown in Figure 4.5. Here it makes sense that the similarity to the aggregation of skipped songs would be the least important metric since we would anticipate a wide variety of songs could be skipped within a session, whereas we hypothesize that played songs would have more common attributes/similarity. A random forest trained using only these five features achieved a 67.57% accuracy, indicating that the feature engineering we did in using a vector space to abstract some of the context of the session in terms of the types of music being listened to and skipped had a positive impact.

4.4.2 Contextual Features

We also calculated two contextual features to include in the dataset. These features were intended to provide additional information beyond the distance metrics used to compare the songs within the session. These features are shown in Table 4.2.

We hypothesized that the percentage of songs a user has skipped in their session might indicate their current level of engagement with their music, as discussed in Section 4.3. A user skipping a majority of the songs that have been played suggests that they are engaged and paying attention and are likely to skip again, only fully listening to songs

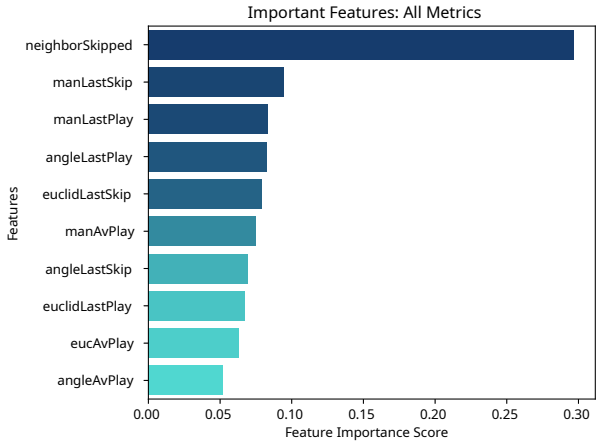

Figure 4.4: Top 10 most important features of a model trained on similarity metrics. Features not shown ranked $\leq .05$

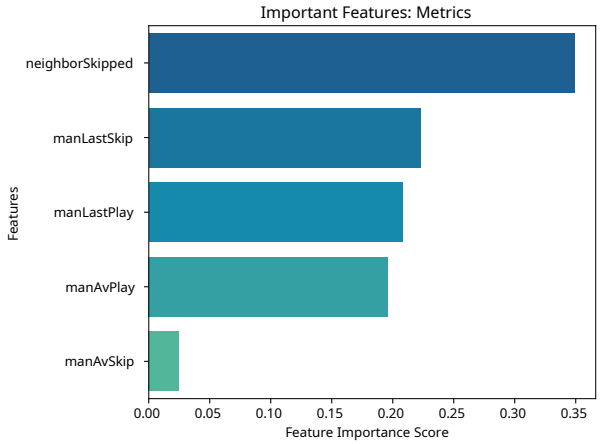

Figure 4.5: Ranking of the features of a model only trained on final metrics

Table 4.2: The two contextual features that were added and their descriptions

Name	Description
percent_skipped	The percentage of songs in the session that the user has skipped
prevSongPlayed	If the user skipped or played the previous song

they really want to hear. Similarly, if the user skipped the most recently played track, this might indicate that they are currently active and paying attention at that moment (even if they were not skipping much earlier) and are more likely to skip again.

As with the other categories of metrics, we trained a random forest using just these two features. In terms of importance, both ranked highly with *percent_skipped* having a score of 53.3 and *prevSongPlayed* having a score of 46.7. Given that there were only two features being utilized, we did not have expectations of high accuracy with this model. However, it was quite surprising that the random forest accurately predicted whether a user would skip the last song

in a session 76.01% of the time. We will discuss our ideas about possible reasons for this in the next section.

Chapter 5: Results

After preprocessing and feature engineering were complete, each row of our dataset (representing a session) contained 5 metric features, 2 contextual features, 4 metadata features, 21 original track features, and an 8-dimensional acoustic vector totaling 40 features and the target value of whether or not the final song was skipped. We then split the 200,000 sessions into a training set containing 70% of the sessions and a test set containing 30% of the sessions.

The random forest models were implemented in Python and utilized the random forest classifier from the sklearn library. The final ensemble was made up of 100 trees, each with a maximum depth of 6. In order to understand the role of different features and feature categories in the model's success, several random forests were created, each using different sets of features. The results of these models were presented as we discussed each category of feature, but are summarized for comparison here, along with the results of our final models.

Table 5.1: Random forest results using different groups of features.

Source	Categories	Accuracy
Spotify	Track Features	53.67%
	Metadata	59.55%
	Track Features & Metadata	59.81%
Engineered	Metrics	67.57%
	Contextual Data	76.01%
	Metrics & Contextual Data	78.26%
All Data		77.69%
Final Model		78.23%

As a baseline, the first iteration only used the 29 original track features of the final song in a session. In this case, the context of a session is ignored completely. The model had to rely on the structure of a single song in order to decide if it will be skipped or not. It has been shown by Montecchio et al. [14] that a user's behavior (skipping or not) can be connected to the musical structure of a track, but it is clear that the track features in this dataset are not allowing us to capture this. This model is the least complex and had 53.67% accuracy as shown in Table 5.1. Then, using only certain metadata from Spotify as described in section 4.3, an accuracy of 59.55% was reached. Combining these groups of

data from Spotify did not meaningfully improve the random forest's predictions.

In order to gauge the effectiveness of our engineered features, they were each used in isolation. Metrics alone achieved 67.57%, and Contextual Data, despite having only two features, was even more accurate at 76.01%. It is evident that the track features are more effective when used to try to reason about the musical context of a session than when a single track's features are used without this context (as in Section 4.2). When these two groups of features (Metrics and Contextual Data) were combined, the model achieved an accuracy of 78.26%. It was surprising to us that the similarity metrics representing musical context only yielded a small improvement in accuracy over the Contextual Data alone. This seems to indicate that the overall user pattern of behavior (skipping or not skipping) may be more important than the user's musical mood.

Table 5.2: Confusion Matrix for Final Model

	Predicted Played	Predicted Skipped
Actual Played	67%	33%
Actual Skipped	14%	86%

Using both the original Spotify features and our newly engineered features in a model (shown as "All Data") resulted in slightly lower accuracy than when only the engineered features were used. This was an unexpected outcome, but we hypothesized that it was due to the near-random-guess performance of the Track Features as a whole. So our final model uses Metrics, Contextual Data, and Metadata, and achieves a predictive accuracy of 78.23%.

Because the classes being predicted (skipped and played) are roughly balanced in both our training and test data, we knew that there was some learning taking place to attain an accuracy of 78%. However, we were interested in the types of mistakes the model was making and where, potentially, improvements could be made. The confusion matrix presented in Table 5.2, shows that the performance of the model in predicting songs that are skipped is better than its performance in predicting the songs that will be played. This, combined with the small increase in accuracy from the inclusion of our engineered similarity Metrics with the Contextual Data leads us to believe that we are leaning too heavily on the user's general behavioral patterns (skipping vs not skipping) and that there is more exploration to be done in learning the type of music that the user is currently interested in hearing. This observation is further upheld in our analysis of feature importance in the next section.

5.1 Feature Importance

One of the goals of this research was to discover which features in the data were the most predictive. When using a random forest, it is possible to calculate which features contributed most to the final predictions. For this, Mean Decrease Impurity (MDI) was used to calculate feature importance. MDI measures the average decrease in the impurity of nodes when splitting the data on a particular feature. Nodes are impure when they contain data from different classes. If a feature has a high MDI score, it effectively splits data into more homogeneous groups, making the most progress toward a prediction. Using the Python libraries matplotlib and seaborn, visualizations of these scores were generated.

Figure 5.1 shows the importance of a track's acoustic features and metadata when making a prediction of whether a user will skip a song. The prominence of the *premium* feature is straightforward to understand. Users without Spotify premium have a limited amount of skips, and therefore will use them more sparingly. The rest of the metadata features are ranked lower than expected. The *month* feature was found to be not important at all. This can be attributed to the fact that the data was not equally distributed across all months, causing the feature to not be properly understood by the random forest. It seems to be the same situation for *day_of_week* as this feature also scored unexpectedly low.

Of our engineered features, the contextual data is extremely predictive. When combined with the metrics it dominates the rankings in Figure 5.2. If we had not run the metrics by themselves first, it would seem as if they are not predictive at all. According to the MDI scores, the contextual data is simply more relevant in making successful predictions in this model. As mentioned previously, this suggests that our future work should be concentrated on improving our metrics in order to be able to better recognize the songs that the user is most interested in listening to.

Focusing only on the metrics, *manLastSkip* and *manLastPlay* both rank higher than the distances between the average vectors. From this result, we can assume that the type of song a user skipped or played most recently is more relevant than the type of songs they have skipped or played on average throughout their session. It is possible that because the average vectors represent every song that has been skipped or played in a session, as a user's session grows, these vectors will become more obscure and less relevant. Also, as the sessions get longer, *manAvPlay* and *manAvSkip* will become less relevant. The average vector will not be able to provide an accurate generalization of so many tracks and

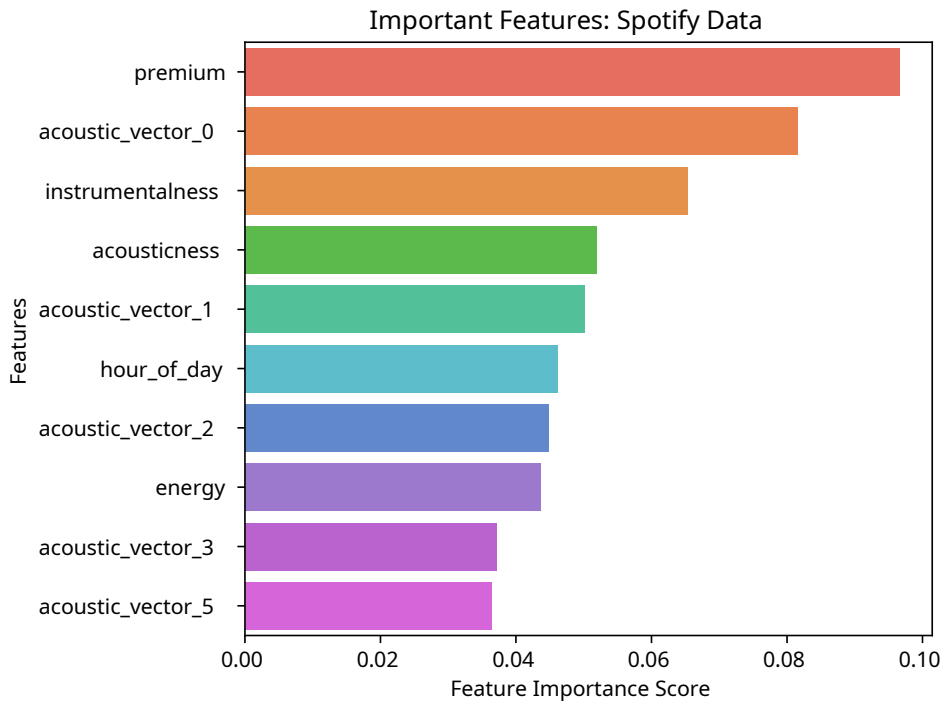

Figure 5.1: The top 10 most important features of a model trained on Spotify's data (Track Features and Metadata). Features not shown ranked $\leq .033$

it will not help with predictions. For this reason, in future work, an average of the last n skipped or played tracks could be more helpful.

Finally, as shown in Figure 5.3, the engineered features greatly outperform those relating to Spotify metadata. As stated before, this group of features was not expected to do so poorly. Though, this outcome may be attributed to the lack of diversity of dates in the data used. It is interesting to note that the order of feature importance of the engineered features remained consistent between when they were used in isolation vs alongside metadata, reinforcing the fact that the metadata was barely influential.

It was surprising to observe how poorly the *premium* feature performed, considering our initial high expectations. One possible explanation for this outcome is that the contextual features already provide information about whether or not a user is a Spotify premium subscriber, rendering this feature redundant. Specifically, if a user frequently skips tracks, it is implicit that they have Spotify premium, and conversely, if they rarely skip tracks, they are likely not premium.

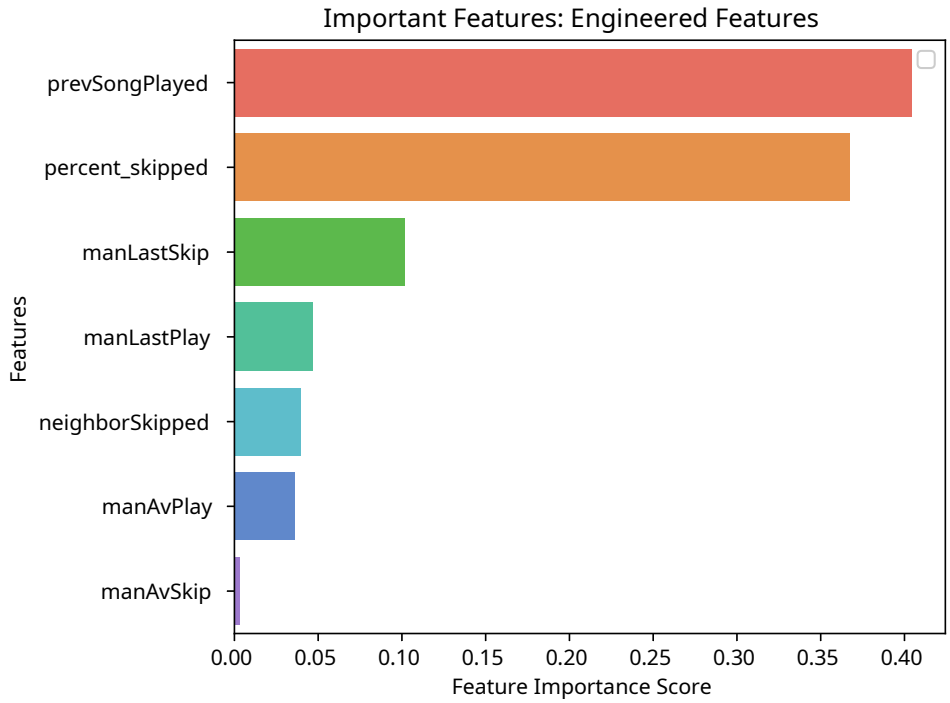

Figure 5.2: The most important features in a model trained on only our engineered features (Metrics, and Contextual data).

5.2 Further Exploration (Session "Baby" Trees)

As stated earlier, the raw track features performed so poorly that they had to be left out of the final model, and were only used when using distance metrics to find the similarities between tracks. In an effort to salvage their potential impact, we explored alternative ways of incorporating them. Our idea is called "Session" or "Baby Trees". This method entails that for each session, a single decision tree classifier is trained on only the track features of each song in a session, and tested on the final one. In this way, we could hopefully capture in a single model, the user's current musical mood. The baby tree's prediction on the final song in the session would then be added to our final preprocessed dataset as a boolean feature. We then had hoped that we could use these decision trees to rank the track features more accurately, and begin to add weight to the ones that were more important.

When deploying this new feature, it only increased our final model's accuracy by 0.01%, and the baby trees themselves were only 56% accurate on average. The feature itself ranked higher than *manAvSkip*, placing it as the 2nd least important feature. These small trees trained on only one session could not make an accurate enough prediction with the data available. Additionally, each tree would have a variable amount of training data as the sessions were not

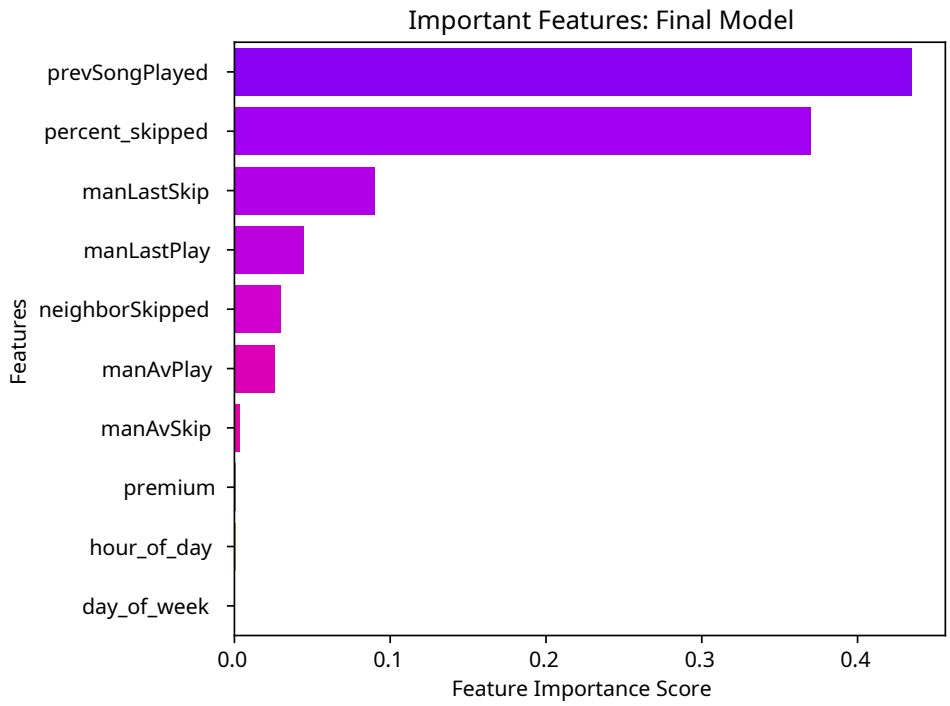

Figure 5.3: The top 10 most important features (no Track Features) used in the final model. Features not shown ranked $\leq .0001$

uniform in length. Because the baby trees were so inaccurate, the MDI scores from them are not worth investigating and a different technique is needed to properly model the type of acoustic structure that a user is currently interested in.

Chapter 6: Future Work

Our goal is that this model could eventually be used not only to make a prediction but to choose songs from the playlist a user is listening to that they are least likely to skip. As the session progresses, and more songs are listened to, the model will have access to more information and be able to make better predictions in real time. A simple way to accomplish this would be to classify every song yet to be played and use the model's confidence to rank which songs it is most certain will not be skipped. This, however, would have linear complexity as the size of a playlist grows. Instead, a random sample of songs from the playlist may be picked and ranked in order to keep up with the speed at which a user can skip songs.

Alternatively, a more complex model using a recurrent neural network similar to the solution of Hansen et al. [8] could be trained to compare the accuracy between artificial intelligence techniques. Our belief is that prediction accuracy might increase with an RNN, but our initial use of decision trees allowed for influential features to be identified and better understood.

If data was available containing more than one session for a user, habits could be learned over time to better anticipate the user's moods. Listening preferences of a user by time of day, day of the week, or even month could be learned. With this information, the system could adjust its choice of queued songs at the beginning of each session. So, if a user typically listens to slow and sad music on weekdays after 6:00 PM, when they shuffle their playlist at this time, the first songs that play are more likely to be slow and sad.

A challenge of this feature would be the balance between true randomization and tailored randomization. The learning aspect must be subtle enough to not negatively impact the user's experience. It is important to avoid locking the system into playing a specific genre of music at a particular time, as this would detract from the intended variety offered by a shuffle algorithm. To achieve seamless and virtually unnoticeable customization, it is crucial to find the proper balance between tailored and random songs.

Chapter 7: Conclusion

In this paper, we have explored our research question of whether we can utilize session data to accurately predict whether or not a user will skip a particular song. We hypothesized that user "skipping" behavior could influence the overall pattern within a session and that users would be more likely to listen to similar songs within a session according to their mood. We have shown that it is indeed possible for a model to make these predictions accurately, though the most influential information was not necessarily as we hypothesized. When predicting whether a user will skip a song, the frequency of skipping and recent skipping behavior are more influential factors than the similarity of the song to previously skipped songs in our current model. In future work, we will explore whether we can more accurately predict whether a user will play a song by utilizing different metrics and strategies. However, our final model described here attained 78.23% accuracy and shows that user behavior can be leveraged to predict whether or not a song will be

skipped in a given session. In the future, this model could be utilized to adapt queued songs according to the user's behavior and current preferences, resulting in fewer skips overall and an improved user experience.

References

[1] Charu C. Aggarwal, Alexander Hinneburg, and Daniel A. Keim. On the surprising behavior of distance metrics in high dimensional space. In Jan Van den Bussche and Victor Vianu, editors, *Database Theory — ICDT 2001*, pages 420–434, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[2] Margherita Barile and Eric W Weisstein. Taxicab metric.

[3] Ferenc Béres, Domokos Miklós Kelen, András Benczúr, et al. Sequential skip prediction using deep learning and ensembles. 2019.

[4] Erik Bernhardsson. Nearest neighbors and vector models – part 2 – algorithms and data structures, Apr 2020.

[5] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, Oct 2001.

[6] Brian Brost, Rishabh Mehrotra, and Tristan Jehan. The music streaming sessions dataset. In *Proceedings of the 2019 Web Conference*. ACM, 2019.

[7] Sander Dieleman. Recommending music on spotify with deep learning, Aug 2014.

[8] Christian Hansen, Casper Hansen, Stephen Alstrup, Jakob Grue Simonsen, and Christina Lioma. Modelling sequential music track skips using a multi-rnn approach. 2019.

[9] Anne Marie Helmenstine. Angle between two vectors and vector scalar product, Aug 2020.

[10] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. Session-based recommendations with recurrent neural networks, 2015.

[11] Dietmar Jannach and Malte Ludewig. When recurrent neural networks meet the neighborhood for session-based recommendation. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, RecSys '17, page 306–310, New York, NY, USA, 2017. Association for Computing Machinery.

[12] Paul Lamere. The skip, May 2014.

[13] Francesco Meggetto, Crawford Revie, John Levine, and Yashar Moshfeghi. Why people skip music? on predicting music skips using deep reinforcement learning. *arXiv preprint arXiv:2301.03881*, 2023.

[14] Nicola Montecchio, Pierre Roy, and Franç ois Pachet. The skipping behavior of users of music streaming services and its relation to musical structure. *PLOS ONE*, 15(9):e0239418, sep 2020.

[15] Xinxi Wang, Yi Wang, David Hsu, and Ye Wang. Exploration in interactive personalized music recommendation: A reinforcement learning approach, 2013.

[16] Eric W Weisstein. Euclidean metric.

[17] Lilian Weng. The multi-armed bandit problem and its solutions. *lilianweng.github.io*, 2018.

[18] Brandon Wheeless. A look into k-dimensional trees, Apr 2021.

Python Library References

matplotlib

Matplotlib is used for data visualization. It provides extensive tools for creating and exporting plots, charts, and other visualizations, with support for many different styles.

numpy

Numpy is a popular and fundamental Python library used for numerical computing. It provides high-performance arrays and matrices, along with many mathematical functions for manipulating them. Numpy is often used for scientific computing, data analysis, and machine learning.

pandas

Pandas is a Python library used for data manipulation and analysis. It provides data structures for storing and querying large datasets, as well as tools for cleaning, filtering, and transforming data. This library is generally used in data science and machine learning applications for data preprocessing and exploratory data analysis.

scipy

Scipy is a library that provides algorithms for scientific computing in Python. It has a wide range of tools for optimization, integration, linear algebra, etc. Scipy is often used in physics, engineering, and other scientific disciplines for data analysis and simulation.

sklearn (SciKit Learn)

Sklearn (scikit-learn) is a Python library used for machine learning. It provides a wide range of tools for data preprocessing, model selection, and model evaluation, as well as implementations of many popular machine learning algorithms, specifically *RandomForestClassifier* and *DecisionTreeClassifier*.